Generic Distribution and Filtering Algorithms

Larry Bush <bushl2@cs.rpi.edu>
Dan Jemiolo <jemiod@rpi.edu>
Kyle Ross <rossk2@rpi.edu>
Dave Siebecker <siebed@cs.rpi.edu>
David R. Musser <musser@cs.rpi.edu>

8 December 2002

Abstract

A generic software library is presented to solve the problem of classifying objects based on user-defined criteria and distributing the objects based on this classfication. Along with documented versions of the source code, a comprehensive discussion of the library is made from several perspectives: The user's guide & tutorial and the reference manual help the reader to quickly begin to use the library. A discussion of the design issues involved in the creation of the library provides future software designers the insight required to extend the library. A lengthy set of tests and results assures the user that the library operates correctly and efficiently.

Contents

Chapter 1

User's Guide

1.1 Introduction

1.1.1 Overview

It is often useful to take a set of items and distribute the elements of this set into two or more destinations based on some sort of decision schema. If this schema is phrased as a sequence of predicates paired with `OutputIterator`s then a generalised view of this problem can be solved by a small set of components that work together to filter elements from a batch of data into one or more destinations based on a collection of user-defined predicates.

Input is specified with either a single `InputIterator` (to specify a single element) or a pair of `InputIterator`s (to signify a range of elements); the various output destinations and their associated predicates will be supplied by creating one or more `functor` classes derived from an abstract base class within the framework. Our components work seamlessly with C++ Standard Library containers and streams, since all communication with them will be through `Input-` and `OutputIterator`s. In order to be as generic and flexible as possible, the interfaces will have to make extensive use of member template functions — this means that our work will be limited to the GNU C++ compiler, as Microsoft's Visual Studio does not yet support this important language feature.[1]

1.1.2 Document Conventions

Since English words like "destination" and "router" have numerous meanings (the latter, particularly, in the computing sciences), explicit textual formatting will be used to differentiate between abstract ideas and concrete C++ types or concepts based on normal text for the former and `special text` for the latter. For example: in the context of this discussion, it is quite useful to differentiate between an abstract destination (a place where output is being

[1] Use of a compiler front-end — like those developed by Comeau Computing — would allow use of compilers such as Microsoft's that are grossly non-compliant with the C++ standard.

sent: possibly a member of a sub-type of the `destination` class, possibly an `OutputIterator`, etc.) and the `destination` class as defined by the Generic Distribution and Filtering Algorithms Library. This should make it significantly easier to understand the present report.

1.1.3 The Generic Distribution and Filtering Algorithms Library

To solve the problem described in Section 1.1.1, the Generic Distribution and Filtering Algorithms Library (henceforth referenced as "GDFA Library") has been developed. Conceptually, the GDFA Library consists in three main components: the trivial `distribute` function, the `destination` base class, and the `router` class. As mentioned supra, each of these is templated to allow for the greatest possible generality, making the library useful over a varied domain of computing problems.

The `distribute` Function

The `distribute` function solves the trivial case of the general filtering problem: an input range is to be distributed to two output destinations based upon element-wise application of a single predicate — elements for which the predicate returns `true` are output to one destination; all other elements are output to a second destination.

The `destination` Base Class

The `destination` class is an abstract class from which users of the GDFA Library may inherit in defining their own destinations, which conceptually model the pairing of a predicate and an `OutputIterator`, as described in Section 1.1.1. `destinations` have a `distribute` method (not to be confused with the trivial `distribute` function) which takes an element, applies a predicate, takes an appropriate action, and returns `false` if and only if the element should be passed to the following `destination`. This is explained in much greater detail in Section 2.1 and is demonstrated by example in Section 1.2.

The `router` Class

For the more general case of the problem, the GDFA Library defines a `router` class. This class stores and prioritises `destination`s and provides a convenient and re-usable way to apply an arbitrarily large number of destinations in a specified order to an input sequence. The `distribute` method takes a range of elements and applies the filtering order implicitly defined by the `destinations`; hence a single `router` can be defined and used to filter *multiple* sequences of elements. The general schema for doing so is: create a `router`, add appropriate `destination`s to that `router`, call the `distribute` method for each sequence to be filtered. Again, a more detailed explanation can be found in Section 2.1 and some practical examples in Section 1.2.

1.2 Tutorial

This tutorial contains a number of "learn by example" programmes that are explained in some detail, each of which illustrates a different aspect of using the GDFA Library. The database example in Section 1.2.1 illustrates a use of the trivial `distribute` function. The sporting goods store example in Section 1.2.2 demonstrates creation of `destination`s and use of the `router` class in performing a fairly simple single-destination[2] filtering task. Finally, the criminal profiling example in Section 1.2.3 demonstrates use of a complex multi-tier[3] filtering application.

1.2.1 The Database Example

A simple example using the trivial `distribute` function is to filter database jobs by priority: database jobs each have a priority from the set $\{1, 2, 3, 4, 5\}$ where 1 is low priority and 5 is high priority. We want to separate out just the jobs that are designated high-priority. These jobs will be run first, followed by all of the other jobs. In this small example, we, of course, are not actually going to do anything with our jobs, but the idea could easily be extended and implemented for a real database.

Here is an overview of the programme we will use to solve the problem:

"run-database-jobs.cpp" 6 ≡

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <iterator>
#include <functional>

#include "router.hpp"

⟨define database job 7a⟩

int main()
{
  std::vector<DatabaseJob> urgentJobs;
  std::vector<DatabaseJob> normalJobs;

  ⟨read and filter the jobs 9a⟩
  ⟨execute the jobs 10a⟩

  return EXIT_SUCCESS;
}
```

[2] "Single-destination" refers to the fact that each element will be placed in at most (or, in the case, exactly) on `destination`.

[3] "Multi-tier" refers to the fact that the input data are filtered and then the results of this filtering are *again* filtered and so on.

A simple class to represent a database job is required. It must provide storage of the job's name, its priority, and its PID. We need functionality to access the priority-level, check if the job is high-priority or not, check if the job has finished running, compare two jobs by priority, and run the job. We define an input operator as a `friend`.

⟨define database job 7a⟩ ≡

```
class DatabaseJob
{
public:
  ⟨priority-level accessor 7b⟩
  ⟨is the job high-priority? 7c⟩
  ⟨is the job finished? 8a⟩
  ⟨run the job 8b⟩
  ⟨compare jobs by priority 8c⟩
  friend std::istream& operator>>(std::istream& in,
                                  DatabaseJob& job);

private:
  std::string m_scriptName;
  int m_priorityLevel;
  std::string m_processID;
};

⟨input a job 8d⟩
```

Used in part 6.

To get the job's priority-level we create an accessor; by convention we define priority-levels in the range [0..5].

⟨priority-level accessor 7b⟩ ≡

```
int PriorityLevel() const
{
  return m_priorityLevel;
}
```

Used in part 7a.

Is the job high-priority? We can easily check this by comparing to see if the priority is greater than 4.

⟨is the job high-priority? 7c⟩ ≡

```
bool IsHighPriority() const
{
  return m_priorityLevel>4;
}
```

Used in part 7a.

A job is finished when the process that is executing its tasks exits. For the sake of our example, we just return `true`.

⟨is the job finished? 8a⟩ ≡

```
bool IsFinished() const
{
  return true;
}
```

Used in part 7a.

We create a new process that performs the tasks this job entails; for the sake of the example, we just display the job's information.

⟨run the job 8b⟩ ≡

```
void Execute()
{
  std::cout<<m_processID<<"  "<< m_priorityLevel<<std::endl;
}
```

Used in part 7a.

In order to compare jobs by priority, we define two comparator operators.

⟨compare jobs by priority 8c⟩ ≡

```
bool operator< (const DatabaseJob& rhs) const
{
  return m_priorityLevel<rhs.m_priorityLevel;
}

bool operator> (const DatabaseJob& rhs) const
{
  return m_priorityLevel>rhs.m_priorityLevel;
}
```

Used in part 7a.

In order to read jobs from our input file, we create a simple input operator. It reads a job's PID and then its priority-level.

⟨input a job 8d⟩ ≡

```
std::istream& operator>>(std::istream& in, DatabaseJob& job)
{
  in>>job.m_processID>>job.m_priorityLevel;
  return in;
}
```

Used in part 7a.

Now that we have defined a simple class to hold a database job, we are ready to filter them! We open the database jobs file and filter the jobs as we read them.

⟨read and filter the jobs 9a⟩ ≡

```
std::ifstream inputFile("./nightly-database-routines.txt");

⟨handle file error 9b⟩
⟨filter the jobs 9c⟩
```

Used in part 6.

If the file listing the jobs is unavailable, we display an error message and exit from the programme, indicating failure to the operating system.

⟨handle file error 9b⟩ ≡

```
if (!inputFile)
{
  std::cerr<<"No ./nightly-database-routines.txt file"<<std::endl;
  return EXIT_FAILURE;
}
```

Used in part 9a.

Now that the file is open, we can filter the jobs by priority. The trivial version of `distribute` will work fine since we are splitting the jobs into exactly one of two groups based on the evaluation of a single predicate — the `IsHighPriority` method defined by the `DatabaseJob` class.

⟨filter the jobs 9c⟩ ≡

```
gdfa::distribute(std::istream_iterator<DatabaseJob>(inputFile),
                 std::istream_iterator<DatabaseJob>(),
                 std::back_inserter(urgentJobs),
                 std::back_inserter(normalJobs),
                 std::mem_fun_ref(&DatabaseJob::IsHighPriority));
```

Used in part 9a.

Now that we have the jobs filtered by priority, we can execute them: first the high-priority jobs, then all the other jobs. For the sake of this example programme, we just print the job information.

⟨execute the jobs 10a⟩ ≡

```
std::cout << "\n\nUrgentJobs\n\n";
for(std::vector<DatabaseJob>::iterator i=urgentJobs.begin();
    i!=urgentJobs.end(); ++i)
{
  (*i).Execute();
}

std::cout << "\n\nNormalJobs\n\n";
for(std::vector<DatabaseJob>::iterator i=normalJobs.begin();
    i!=normalJobs.end(); ++i)
{
  (*i).Execute();
}
```

Used in part 6.

To complete our discussion of this example, all we need is a data file.

"nightly-database-routines.txt" 10b ≡

```
JobTypeA 2 JobTypeA 2 JobTypeH 5 JobTypeH 5
JobTypeA 2 JobTypeA 2 JobTypeB 1 JobTypeA 2
JobTypeA 2 JobTypeH 5 JobTypeH 5 JobTypeA 2
JobTypeA 2 JobTypeB 1
```

Finally, we can take a look at the output from our programme, where we not that the jobs have been filtered properly:

⟨output from database example 10c⟩ ≡

```
UrgentJobs

JobTypeH  5
JobTypeH  5
JobTypeH  5
JobTypeH  5


NormalJobs

JobTypeA  2
JobTypeA  2
JobTypeA  2
JobTypeA  2
JobTypeB  1
JobTypeA  2
JobTypeA  2
JobTypeA  2
JobTypeA  2
JobTypeB  1
```

Not used.

1.2.2 The Sporting Goods Store Example

We now look at a simple example of more robust filtering using the `router` class: suppose we own a sporting goods store "Bob's Big Bad Sports Store" and we want to send a mailing to customers telling them of the sale that is currently being held at the store. We have a database of the customers and their past purchases and want to send a *single* mailing to each customer to try and encourage him or her to come into the store and to make a purchase. Here is an overview of our code:

"sports-store.cpp" 11a ≡

```
#include <algorithm>
#include <iostream>
#include <set>
#include <string>
#include <vector>

#include "router.hpp"

⟨define customer record 11b⟩
⟨define sales offer destination 12b⟩

int main()
{
  std::vector<customer> store_database;
  ⟨create the groups of items for each sport 14a⟩
  ⟨create the customers 14b⟩
  ⟨set up purchases for each customer 14c⟩
  ⟨add each customer to the store database 15a⟩
  ⟨set up the sales items offers 15b⟩
  ⟨create the junk-mailer 15c⟩
  ⟨let the junk-mail begin 16a⟩
  return EXIT_SUCCESS;
}
```

We define a record for a `customer`: we need to store the customer's name, his or her address, and the items that he or she has purchased from our store.

⟨define customer record 11b⟩ ≡

```
struct customer
{
  ⟨define customer constructor 12a⟩

  std::string name, address_line_1, address_line_2;
  std::set<std::string> items_purchased;
};
```

Used in part 11a.

Every `customer` must have a name and an address; hence we force values to be set for each of these fields in our `customer` constructor.

⟨define customer constructor 12a⟩ ≡

```
customer(const std::string &new_name,
           const std::string &new_address_line_1,
           const std::string &new_address_line_2)
:name(new_name), address_line_1(new_address_line_1),
 address_line_2(new_address_line_2)
{}
```

Used in part 11b.

What do we need to determine if a user should get a specific mailing? Well, we are going to base advertising on past purchases: if a customer has purchased basketball equipment in the past then he or she will receive an advertisement for basketball equipment now! Our predicate must keep track of what is on sale, what items are from the same sport, and where to output the advertisement letter (e.g. to some sort of file that will be routed to a printer where the junk mail will be printed, addressed, and mailed).

⟨define sales offer destination 12b⟩ ≡

```
template <typename OutputStream>
class sale_offer : public gdfa::destination<customer>
{
public:
  ⟨sales_offer constructor 12c⟩
  ⟨sales_offer distribute function 13⟩
private:
  std::set<std::string> items;
  std::string item_on_sale;
  OutputStream &where;
};
```

Used in part 11a.

The constructor for the group of items pertaining to a particular sport takes an `OutputStream` indicating the destination for the propaganda letters, two `InputIterator`s which are assumed to refer to a range of `string`s that specify the paraphernalia related to that sport, and a `std::string` indicating the item on sale.

⟨sales_offer constructor 12c⟩ ≡

```
template <typename InputIterator>
sale_offer(OutputStream &new_where, std::string new_item_on_sale,
           InputIterator first, InputIterator last)
:where(new_where), item_on_sale(new_item_on_sale),
 items(first, last)
{}
```

Used in part 12b.

Here we do the main work of the `purchased_type`'s filtering: see if the customer is a "fan" of the given sport by checking to see if the customer has purchased any paraphernalia relating to the current sport. If so, we print out the letter and return `true` to indicate that an advertisement has been sent to this customer; hence, another advert will *not* be sent to that customer.

⟨sales_offer distribute function 13⟩ ≡

```
bool distribute(const customer& recipient)
{
  std::vector<std::string> items_of_type_purchased;
  std::set_intersection(recipient.items_purchased.begin(),
                        recipient.items_purchased.end(),
                        items.begin(), items.end(),
                        std::back_inserter(
                         items_of_type_purchased));
  if(!items_of_type_purchased.empty())
  {
    where<<"Bob's Big Bad Sports Store"<<std::endl;
    where<<"112 42nd St."<<std::endl;
    where<<"Troy, NY 12180\n\n\n\n"<<std::endl;
    where<<recipient.name<<std::endl;
    where<<recipient.address_line_1<<std::endl;
    where<<recipient.address_line_2<<'\n'<<std::endl;
    where<<"Dear "<<recipient.name<<":\n"<<std::endl;
    where<<"Bob's Big Bad Sports Store is pleased to ";
    where<<"announce that we currently are running a ";
    where<<"sale on "<<item_on_sale;
    where<<"s. Since you have purchased items such as ";
    std::copy(items_of_type_purchased.begin(),
              items_of_type_purchased.end(),
              std::ostream_iterator<std::string>(where, ", "));
    where<<"we think you would be interested in this ";
    where<<"fantastic promotion. Please visit our store today ";
    where<<"to take advantage of this limited-time offer!\n";
    where<<std::endl;
    where<<"Yours sincerely,\n"<<std::endl;
    where<<"Big Bob"<<std::endl;
    return true;
  }
  return false;
}
```

Used in part 12b.

Now, we return our attention to our main programme. We need to define a database of the things that Bob's sells. These are the goods sold at Big Bob's, categorised by the sport to which they pertain:

⟨create the groups of items for each sport 14a⟩ ≡

```
char *golf_stuff[]={"golf bag", "set of clubs", "titanium driver",
                    "9-iron"};
char *lacrosse_stuff[]={"lacrosse stick", "gloves", "helmet"};
char *basketball_stuff[]={"basketball", "basketball shoes",
                          "hoop"};
```

Used in part 11a.

We create a mailing address for each customer.

⟨create the customers 14b⟩ ≡

```
customer mr_jones("Mr. Charles Jones", "123 Harcourt Terrace",
                  "Troy, NY 12180");
customer mrs_hobbins("Mrs. Bobby Hobbins", "56 N. Madison Ave.",
                   "Chicago, IL 60601");
customer ms_smith("Ms. Katie Smith", "41 Piedmont Rd. NE",
                  "Atlanta, GA 3035");
customer mr_jasper("Mr. George Jasper", "20 Bonneville",
                   "Las Vegas, NV 89101");
```

Used in part 11a.

Next, we need to set up the purchase history for each customer. We do so by assigning a set of items purchased to the record of each customer.

⟨set up purchases for each customer 14c⟩ ≡

```
char *_mr_jones_purchases[]={"golf bag", "basketball",
                             "set of clubs"};
char *_mrs_hobbins_purchases[]={"gloves"};
char *_ms_smith_purchases[]={"basketball", "basketball shoes"};
char *_mr_jasper_purchases[]={"9-iron", "lacrosse stick", "hoop"};
mr_jones.items_purchased=std::set<std::string>(
 &_mr_jones_purchases[0],
 &_mr_jones_purchases[sizeof(_mr_jones_purchases)/sizeof(char*)]);
mrs_hobbins.items_purchased=std::set<std::string>(
 &_mrs_hobbins_purchases[0],
 &_mrs_hobbins_purchases[
  sizeof(_mrs_hobbins_purchases)/sizeof(char*)]);
ms_smith.items_purchased=std::set<std::string>(
 &_ms_smith_purchases[0],
 &_ms_smith_purchases[sizeof(_ms_smith_purchases)/sizeof(char*)]);
mr_jasper.items_purchased=std::set<std::string>(
 &_mr_jasper_purchases[0],
 &_mr_jasper_purchases[sizeof(_mr_jasper_purchases)/
                       sizeof(char*)]);
```

Used in part 11a.

The customers need to be inserted into the store database.

⟨add each customer to the store database 15a⟩ ≡

```
store_database.push_back(mr_jones);
store_database.push_back(mrs_hobbins);
store_database.push_back(ms_smith);
store_database.push_back(mr_jasper);
```

Used in part 11a.

Next, we set up the advertisements for each sport. For simplicity, we dump all of the letters-to-customers to `std::cout`. The second parameter in each of the constructor calls indicates the name of the item that is on sale; the final two parameters indicate the items which relate to the same sport as the one on sale.

⟨set up the sales items offers 15b⟩ ≡

```
sale_offer<std::ostream> golf_sale(std::cout, "titanium driver",
                                   &golf_stuff[0],
                                   &golf_stuff[sizeof(golf_stuff)
                                    /sizeof(char*)]);
sale_offer<std::ostream> lacrosse_sale(std::cout, "lacrosse stick",
                                       &lacrosse_stuff[0],
                                       &lacrosse_stuff[
                                        sizeof(lacrosse_stuff)
                                        /sizeof(char*)]);
sale_offer<std::ostream> basketball_sale(std::cout, "hoop",
                                         &basketball_stuff[0],
                                         &basketball_stuff[
                                          sizeof(basketball_stuff)
                                          /sizeof(char*)]);
```

Used in part 11a.

Add each of the sale adverts to the junk mailer. Since golf equipment is more expensive than lacrosse equipment which, in turn, is more expensive than basketball equipment, we prioritise the advertisements appropriately: if a customer has purchased both golf and lacrosse equipment, we want to send him or her the golf advertisement and *not* the lacrosse advertisement since we, at Big Bob's, are trying to make the most money off of the customers and we can best do so by advertising expensive products to them.

⟨create the junk-mailer 15c⟩ ≡

```
typedef gdfa::router<customer> mailer;
mailer junk_mailer;
junk_mailer.add_destination(golf_sale,
 gdfa::router<customer>::high_priority);
junk_mailer.add_destination(lacrosse_sale);
junk_mailer.add_destination(basketball_sale,
 gdfa::router<customer>::low_priority);
```

Used in part 11a.

Finally, we produce the items to-be-mailed using the junk-mailer that we have created and the store's database: we pass the database to the mailer and the mailer determines the most expensive sale item which fits the tastes of each customer and then creates a proper mailing for that advertisement, addressed to the customer and making mention of the previously-purchased products for that sport.

⟨let the junk-mail begin 16a⟩ ≡

```
junk_mailer.distribute(store_database.begin(),
                       store_database.end());
```

Used in part 11a.

In the interest of space, we present here one of the advertisements rather than all of them, but this should give a reasonable idea of the style of the output:[4]

⟨sample of output from sporting store example 16b⟩ ≡

```
Bob's Big Bad Sports Store
112 42nd St.
Troy, NY 12180




Mr. Charles Jones
123 Harcourt Terrace
Troy, NY 12180

Dear Mr. Charles Jones:

Bob's Big Bad Sports Store is pleased to announce that we currently
are running a sale on titanium drivers. Since you have purchased
items such as golf bag, set of clubs, we think you would be
interested in this fantastic promotion. Please visit our store
today to take advantage of this limited-time offer!

Yours sincerely,

Big Bob
```

Not used.

1.2.3 The Criminal Profiling Example

We now look at a slightly more complicated use of the `router` class. Suppose that we know that two suspects were spotted by eyewitnesses leaving a crime scene matching the following descriptions: Suspect 1 is between 1.65 and 1.75m tall, weighs between 65 and 75kg, and has brown hair; Suspect 2 is between

[4] Line breaks were added to the output so as to make it readable.

1.75 and 1.85m tall, weights between 80 and 90kg and also has brown hair. Given that we have a database of criminals, we want to find out if any criminals in the database match these descriptions so that we can round up the usual suspects and interrogate them. The purpose of this programme is to illustrate a multi-tier use of the `router` class: we will first filter based on height, next filter based on weight, and, finally, filter based on hair-colour in a three-step process. This example is, of course, very contrived, but this could be used for advanced data-mining or analysis of complex data in a real programme.

"criminal.cpp" 17 ≡

```
#include <algorithm>
#include <iostream>
#include <map>
#include <string>
#include <vector>

#include "router.hpp"

⟨define criminal profile 18a⟩
⟨define classes for filtering 20b⟩

int main()
{
  std::vector<criminal> criminal_database;
  std::vector<criminal> suspect1, suspect2;

  ⟨create the criminal records 24⟩
  ⟨run the search 25a⟩
  ⟨display the results 27c⟩

  return EXIT_SUCCESS;
}
```

Since we need a way to store a criminal's profile, we create a structure in which we can store vital information such as name, height, weight, build, hair-colour, eye-colour, sex, and race.

⟨define criminal profile 18a⟩ ≡

```
struct criminal
{
  ⟨define useful enumerations for criminal data 18b⟩
  ⟨criminal constructor 18c⟩

  std::string name;
  double height;
  unsigned int weight;
  build my_build;
  hair_colour my_hair_colour;
  eye_colour my_eye_colour;
  sex my_sex;
  race my_race;
};

⟨criminal output operator 19⟩
```

Used in part 17.

A few useful enumerations are needed to help us deal with the data in an abstract sense. ``_hair'' is suffixed on each of the hair-colours because of conflicts that would otherwise arise between the constants in the `hair_colour` enumeration and the `eye_colour` and `race` enumerations.

⟨define useful enumerations for criminal data 18b⟩ ≡

```
enum build {thin, medium, stocky, heavy};
enum hair_colour {black_hair, brown_hair, blonde_hair, white_hair,
                  red_hair, grey_hair};
enum eye_colour {brown, blue, green, hazel};
enum sex {male, female};
enum race {black, white, hispanic, arab};
```

Used in part 18a.

The constructor simply initialises the right values.

⟨criminal constructor 18c⟩ ≡

```
criminal(const std::string &new_name, double new_height,
         unsigned int new_weight, build new_build,
         hair_colour new_hair_colour,
         eye_colour new_eye_colour, sex new_sex,
         race new_race)
:name(new_name), height(new_height), weight(new_weight),
 my_build(new_build), my_hair_colour(new_hair_colour),
 my_eye_colour(new_eye_colour), my_sex(new_sex), my_race(new_race)
{}
```

Used in part 18a.

It would be nice to have a convenient way to display a criminal's statistics; pursuant to this goal, we create an output operator for a `criminal` object.

⟨criminal output operator 19⟩ ≡

```
std::ostream& operator<<(std::ostream &out,
                         const criminal &to_print)
{
  ⟨create string mappings for the enumerations 20a⟩

  out<<to_print.name<<std::endl;
  out<<"  height: "<<to_print.height<<'m'<<std::endl;
  out<<"  weight: "<<to_print.weight<<"kg"<<std::endl;
  out<<"  build : "<<build_labels[to_print.my_build]<<std::endl;
  out<<"  hair  : "<<hair_labels[to_print.my_hair_colour];
  out<<std::endl;
  out<<"  eyes  : "<<eye_labels[to_print.my_eye_colour]<<std::endl;
  out<<"  sex   : "<<sex_labels[to_print.my_sex]<<std::endl;
  out<<"  race  : "<<race_labels[to_print.my_race]<<std::endl;
}
```

Used in part 18a.

For convenience during printing, we create a group of mappings of `std::string` labels for the enumerated types. The map is only initialised on the first time that the output operator is called.

⟨create string mappings for the enumerations 20a⟩ ≡

```
static bool first=true;

static std::map<criminal::build, std::string> build_labels;
static std::map<criminal::hair_colour, std::string> hair_labels;
static std::map<criminal::eye_colour, std::string> eye_labels;
static std::map<criminal::sex, std::string> sex_labels;
static std::map<criminal::race, std::string> race_labels;

if(first)
{
  build_labels[criminal::thin]="thin";
  build_labels[criminal::medium]="medium";
  build_labels[criminal::stocky]="stocky";
  build_labels[criminal::heavy]="heavy";

  hair_labels[criminal::black_hair]="black";
  hair_labels[criminal::brown_hair]="brown";
  hair_labels[criminal::blonde_hair]="blonde";
  hair_labels[criminal::white_hair]="white";
  hair_labels[criminal::red_hair]="red";
  hair_labels[criminal::grey_hair]="grey";

  eye_labels[criminal::brown]="brown";
  eye_labels[criminal::blue]="blue";
  eye_labels[criminal::green]="green";
  eye_labels[criminal::hazel]="hazel";

  sex_labels[criminal::male]="male";
  sex_labels[criminal::female]="female";

  race_labels[criminal::black]="black";
  race_labels[criminal::white]="white";
  race_labels[criminal::hispanic]="hispanic";
  race_labels[criminal::arab]="arab";

  first=false;
}
```

Used in part 19.

Next, we create several ways of filtering criminals.

⟨define classes for filtering 20b⟩ ≡

⟨filter by height 21a⟩

⟨filter by weight 22a⟩

⟨filter by hair-colour 23a⟩

Used in part 17.

We need a way to filter the criminals in our database by their heights. Since witness testimony is often inaccurate, we want to be able to search for an height range. This `destination` will allow us to do this.

⟨filter by height 21a⟩ ≡

```
template <typename OutputIterator>
class height_filter : public gdfa::destination<criminal>
{
public:
  ⟨height_filter constructor 21b⟩
  ⟨height_filter distribute 21c⟩

private:
  double min, max;
  OutputIterator where;
};
```

Used in part 20b.

The constructor just takes an `OutputIterator` and two `double`s which define the range of height values that we seek.

⟨height_filter constructor 21b⟩ ≡

```
height_filter(OutputIterator new_where, double new_min,
              double new_max)
:where(new_where), min(new_min), max(new_max)
{}
```

Used in part 21a.

Next, we compare a criminal's height against the range that we are looking for; output his or her information if an height match is found. `false` is vacuously returned because a criminal may match the description of one or more suspects.

⟨height_filter distribute 21c⟩ ≡

```
bool distribute(const criminal &suspect)
{
  if(min<=suspect.height && suspect.height<=max)
  {
    *(where++)=suspect;
  }
  return false;
}
```

Used in part 21a.

We also want to be able to filter by weight. We need a separate `weight_filter` class because in the `distribute` function, we need to access different members of the `criminal` structure.

⟨filter by weight 22a⟩ ≡

```
template <typename OutputIterator>
class weight_filter : public gdfa::destination<criminal>
{
public:
  ⟨weight_filter constructor 22b⟩
  ⟨weight_filter distribute 22c⟩

private:
  unsigned int min, max;
  OutputIterator where;
};
```

Used in part 20b.

We create another trivial constructor.

⟨weight_filter constructor 22b⟩ ≡

```
weight_filter(OutputIterator new_where, unsigned int new_min,
              unsigned int new_max)
:where(new_where), min(new_min), max(new_max)
{}
```

Used in part 22a.

We distribute exactly like we did for `filter_height` except that we change `suspect.height` to `suspect.weight`.

⟨weight_filter distribute 22c⟩ ≡

```
bool distribute(const criminal &suspect)
{
  if(min<=suspect.weight && suspect.weight<=max)
  {
    *(where++)=suspect;
  }
  return true;
}
```

Used in part 22a.

Since we also have information about the suspects' hair-colour, we create a `destination` to filter by hair-colour.

⟨filter by hair-colour 23a⟩ ≡

```
template <typename OutputIterator>
class colour_filter : public gdfa::destination<criminal>
{
public:
  typedef criminal::hair_colour colour;

  ⟨colour_filter constructor 23b⟩
  ⟨colour_filter distribute 23c⟩

private:
  colour to_match;
  OutputIterator where;
};
```

Used in part 20b.

Yet another fairly simple constructor.

⟨colour_filter constructor 23b⟩ ≡

```
colour_filter(OutputIterator new_where, colour new_to_match)
:where(new_where), to_match(new_to_match)
{}
```

Used in part 23a.

Use a simple equality comparison against a known hair-colour and the criminal's to see if he or she matches the suspect's description.

⟨colour_filter distribute 23c⟩ ≡

```
bool distribute(const criminal &suspect)
{
  if(suspect.my_hair_colour==to_match)
  {
    *(where++)=suspect;
  }
  return true;
}
```

Used in part 23a.

We look again at the main programme. The database is populated with a set of criminals,[5] each created using the `criminal` constructor.

[5]This information is pulled from the U.S. FBI's Most-Wanted database; this example is not intended to indicate the authors' support of profiling in investigations.

⟨create the criminal records 24⟩ ≡

```
criminal_database.push_back(criminal("Usama Bin Laden",
                            1.96, 72, criminal::thin,
                            criminal::brown_hair,
                            criminal::brown, criminal::male,
                            criminal::arab));
criminal_database.push_back(criminal("Hopeton Eric Brown",
                            1.73, 80, criminal::medium,
                            criminal::black_hair,
                            criminal::brown, criminal::male,
                            criminal::black));
criminal_database.push_back(criminal("James J. Bulger",
                            1.73, 70, criminal::medium,
                            criminal::grey_hair,
                            criminal::blue, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("Robert William Fisher",
                            1.83, 86, criminal::medium,
                            criminal::brown_hair,
                            criminal::blue, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("Victor Manuel Gerena",
                            1.68, 75, criminal::stocky,
                            criminal::brown_hair,
                            criminal::green, criminal::male,
                            criminal::hispanic));
criminal_database.push_back(criminal("Glen Stewart Godwin",
                            1.83, 90, criminal::medium,
                            criminal::grey_hair,
                            criminal::green, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("Richard Steve Goldberg",
                            1.83, 72, criminal::medium,
                            criminal::brown_hair,
                            criminal::brown, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("Eric Robert Rudolph",
                            1.80, 78, criminal::medium,
                            criminal::brown_hair,
                            criminal::blue, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("James Spencer Springette",
                            1.80, 109, criminal::heavy,
                            criminal::black_hair,
                            criminal::brown, criminal::male,
                            criminal::white));
criminal_database.push_back(criminal("Donald Eugene Webb",
                            1.75, 75, criminal::medium,
                            criminal::grey_hair,
                            criminal::brown, criminal::male,
                            criminal::white));
```

Used in part 17.

In an overview of our search, as mentioned earlier, we filter based on height, then based on weight, then based on hair-colour.

⟨run the search 25a⟩ ≡

```
std::vector<criminal> temp1, temp2;
typedef std::back_insert_iterator<std::vector<criminal> > BItype;
```

⟨filter based on height 25b⟩

⟨filter based on weight 26a⟩

⟨filter based on hair-colour 26d⟩

Used in part 17.

First, we separate the criminals in the database based on height.

⟨filter based on height 25b⟩ ≡

⟨set up height filters 25c⟩

⟨run height filter 25d⟩

Used in part 25a.

We know that one of the suspects is between 1.65 and 1.75m and that the other is between 1.75 and 1.85 in height; we create a `destination` for each (more precisely, a `height_filter`) and add these to a `router`.

⟨set up height filters 25c⟩ ≡

```
height_filter<BItype> suspect_1_height(std::back_inserter(
                                          suspect1),
                                       1.65, 1.75);
height_filter<BItype> suspect_2_height(std::back_inserter(
                                         suspect2),
                                      1.75, 1.85);
gdfa::router<criminal> search_by_height;
search_by_height.add_destination(suspect_1_height);
search_by_height.add_destination(suspect_2_height);
```

Used in part 25b.

Then, we distribute the criminals from the database into the vectors `suspect1` and `suspect2`; criminals who fit neither of the sets of height requirements are discarded.

⟨run height filter 25d⟩ ≡

```
search_by_height.distribute(criminal_database.begin(),
                            criminal_database.end());
```

Used in part 25b.

Next, we filter based on the weights described by the eyewitnesses.

⟨filter based on weight 26a⟩ ≡

⟨set up weight filters 26b⟩

⟨run weight filters 26c⟩

Used in part 25a.

We now set a `router` for each criminal's weight: 65 to 75kg for the first, 80 to 90kg for the second. It would be nice if we could use the trivial version of `distribute`, but there seems no logical location where the criminals whose records did not match the predicate would be output. This version does not output them to any iterator, which is exactly what is desired.

⟨set up weight filters 26b⟩ ≡

```
weight_filter<BItype> suspect_1_weight(std::back_inserter(temp1),
                                       65, 75);
gdfa::router<criminal> filter_1_by_weight;
filter_1_by_weight.add_destination(suspect_1_weight);

weight_filter<BItype> suspect_2_weight(std::back_inserter(temp2),
                                       80, 90);
gdfa::router<criminal> filter_2_by_weight;
filter_2_by_weight.add_destination(suspect_2_weight);
```

Used in part 26a.

Now, filter by weight the criminals who (so far) match the description of the respective suspects. The refined suspect lists are held in `temp1` and `temp2` for suspect1 and suspect2, respectively, we copy this back into `suspect1` and `suspect2` and clear the temporary `std::vectors`.

⟨run weight filters 26c⟩ ≡

```
filter_1_by_weight.distribute(suspect1.begin(), suspect1.end());
filter_2_by_weight.distribute(suspect2.begin(), suspect2.end());

suspect1=temp1;
suspect2=temp2;

temp1.clear();
temp2.clear();
```

Used in part 26a.

Finally, we examine the hair-colour entries for each of the criminals in our database and eliminate those who do not match.

⟨filter based on hair-colour 26d⟩ ≡

⟨set up colour filters 27a⟩

⟨run colour filters 27b⟩

Used in part 25a.

We set up a filter for each suspect to eliminate all potential suspects that do not have brown hair.

⟨set up colour filters 27a⟩ ≡

```
colour_filter<BItype> suspect_1_colour(std::back_inserter(temp1),
                                       criminal::brown_hair);
gdfa::router<criminal> filter_1_by_colour;
filter_1_by_colour.add_destination(suspect_1_colour);

colour_filter<BItype> suspect_2_colour(std::back_inserter(temp2),
                                       criminal::brown_hair);
gdfa::router<criminal> filter_2_by_colour;
filter_2_by_colour.add_destination(suspect_2_colour);
```

Used in part 26d.

Run these filters and copy the results back from the temp `std::vectors`. We are done at last!

⟨run colour filters 27b⟩ ≡

```
filter_1_by_colour.distribute(suspect1.begin(), suspect1.end());
filter_2_by_colour.distribute(suspect2.begin(), suspect2.end());

suspect1=temp1;
suspect2=temp2;
```

Used in part 26d.

Calls to `std::copy` get the data printed, and, voilá, we have a list of all criminals matching each of the two suspects described by the eyewitnesses who saw them fleeing the scene of the crime.

⟨display the results 27c⟩ ≡

```
std::cout<<"--- Criminals matching the description ";
std::cout<<"of Suspect1 ---";
std::cout<<std::endl;
std::copy(temp1.begin(), temp1.end(),
          std::ostream_iterator<criminal>(std::cout, "\n"));
std::cout<<'\n'<<std::endl;

std::cout<<"--- Criminals matching the description ";
std::cout<<"of Suspect2 ---";
std::cout<<std::endl;
std::copy(temp2.begin(), temp2.end(),
          std::ostream_iterator<criminal>(std::cout, "\n"));
```

Used in part 17.

Here is the output of our search:

⟨output from criminal profiling example 27d⟩ ≡

```
--- Criminals matching the description of Suspect1 ---
Victor Manuel Gerena
  height: 1.68m
  weight: 75kg
  build : stocky
  hair  : brown
  eyes  : green
  sex   : male
  race  : hispanic


--- Criminals matching the description of Suspect2 ---
Robert William Fisher
  height: 1.83m
  weight: 86kg
  build : medium
  hair  : brown
  eyes  : blue
  sex   : male
  race  : white
```

Not used.

Chapter 2

Reference Manual

2.1 API Reference

2.1.1 Trivial `distribute` Function

Prototype

⟨prototype for the trivial distribute function 29⟩ ≡

```
template <typename InputIterator,
          typename OutputIteratorIfTrue,
          typename OutputIteratorIfFalse,
          typename Predicate>
void distribute(InputIterator first, InputIterator last,
                OutputIteratorIfTrue out1,
                OutputIteratorIfFalse out2, Predicate p)
```

Not used.

Description

Evaluates each element in the input range against the predicate specified; the element is copied to one of the two output iterators based on the result of the test.[1] All elements will be placed in one of the destinations.

Definition

Defined in the header file `router.hpp`.

Type Requirements

- `InputIterator` models the `Input Iterator` concept, as defined in [4].

[1] This implies that the value types of the input iterator and each of the output iterators should be equal; failing that, the output iterator types should be copy constructible from objects of the input iterator's value type.

- `OutputIteratorIfTrue` and `OutputIteratorIfFalse` model the `Output Iterator` concept, as defined in [4].
- `Predicate` models the `Predicate` concept, as defined in [4].

Preconditions

- [`first`, `last`) is a valid range.

Complexity

$O(N)$ predicate evaluations and assignments (both average and worst-case) where N is `last` − `first`.

2.1.2 `destination` Base Class

Template Signature

⟨template signature for destination base class 30⟩ ≡

```
template <typename T>
destination
```

Not used.

Description

`destination` defines an abstract base class from which users of the GDFA Library can derive wrapper classes for the predicate-destination pairs used by the `router` class.

Definition

Defined in the header file `router.hpp`.

Template Parameters

Parameter	Description
`T`	The `destination`'s value type: the type of object filtered by the `destination`.

Type Requirements

None, except those imposed by derived classes.

Refinement of

`Copy Constructable` (as defined in [4]).

Valid Expressions

Expression	Return Type	Semantics
`distribute(o)`	`bool`	When this function is called on an object `o` that is being examined. The function should return `true` if and only if the given destination is the last to evaluate the item `o` (that is, the router should stop processing it), `false` else.

2.1.3 `router` Class

Prototype

⟨template signature for router class 31⟩ ≡

```
template <typename T>
router<T>
```

Not used.

Definition

Defined in the header file `router.hpp`.

Template Parameters

Parameter	Description
`T`	The `router`'s value type: the type of object distributed by the `router`.

Refinement of

`Default Constructable`, `Copy Constructable`, `Assignable`.

Members

Member	Description
`priority_level`	Internal class for specifying the relative priority of destinations managed by the router. Instances of this class can be compared with the less-than, greater-than, and equality operators.
`low_priority`, `normal_priority`, `high_priority`, `urgent_priority`	The four allowable priority levels for destinations added to the router. Each is a static instance of `router::priority_level`.[2]
`router()`	Default constructor; creates an empty `router`.
`template <typename Iterator> router(Iterator first, Iterator last, priority_level priority=normal_priority)`	Constructor; creates a `router` that manages the `destination` objects held in the range specified. The destinations found in the range must be instantiated from a sub-class of `destination<T>`. The priority selected will be assigned to every `destination` in the range.
`~router()`	Destructor.
`template <typename Iterator> void distribute(Iterator item)`	Allows each destination to evaluate the object at the location referenced by the given iterator until a match is found or all destinations have been considered.[3]
`template <typename Iterator> void distribute(Iterator first, Iterator last)`	Invokes `distribute` for each iterator in the range specified.
`template <typename Destination> void add_destination(const Destination& d, priority_level priority = normal_priority)`	Adds the given `destination` to the group managed by the `router`: its place in the group will be determined by the assigned priority. The `destination` given must be instantiated from a sub-class of `destination<T>`. The priority level must be one of the four defined by `router<T>`.

[2] The implementation of the priority levels is trivial and thus, not expensive; the only reason for their existence is to limit the range of priorities.

[3] The time bound on this operation is $O(N)$, where N is the number of destinations.

Member	Description
`template <typename Destination> void set_default(const Destination& d)`	Adds a default `destination` to the `router`; the default will be used whenever no match is found for an input. The `destination` given must be instantiated from a sub-class of `destination<T>`. No priority is assigned to the default `destination`.[4]
`void remove_default()`	Removes a default destination; if there is no default, no action is taken.

[4] The actual priority can be said to be less than `router::low_priority`.

Chapter 3

Design Issues and Source Code

This chapter will discuss the design issues involved in the creation of the GDFA Library; for each topic presented, the relevant source code will be included for reference. We will first review the ideas and concepts that led to the development of the library, and then revisit each individually as we show how ideas were transformed into implementation. Extra attention should be given to the explanation of terminology, as some words used here have definitions that are quite different from their more common meanings.

3.1 Concept of Distribution and Filtering

The foundation for the GDFA Library is in the simple `distribute` function documented earlier. To see how this utility function serves as the premise for the more advanced `router` and `destination` classes, we need to revisit the first example presented in the User's Guide (Chapter 1). Recall that the example programme was managing batch jobs that ran as part of daily database maintenance procedures; jobs were split into two categories—`normal` and `urgent`—and then executed in terms of their relative priority. The `distribute` function was used to filter the jobs into the correct container directly from the input stream, so that no unnecessary copying was performed.

3.1.1 Introduction to Distribution and Filtering

To see how this generic utility offers a real value to the programmer, let us first consider an applications-specific solution to this I/O problem—it takes a file name and two `std::vector`s in which to copy the data:

⟨database distribute example 1 35⟩ ≡

```
void distribute(const string& fileName,
                vector<DatabaseJob>& urgent,
                vector<DatabaseJob>& normal)
{
  ifstream inputFile(fileName);
  DatabaseJob job;

  while(inputFile>>job)
  {
    if(job.IsHighPriority())
    {
      urgent.push_back(job);
    }
    else
    {
      normal.push_back(job);
    }
  }
}
```

Not used.

3.1.2 Generic Input Range

The first thing to note is that we have restricted our source of input to a formatted file—we can never use this code to filter objects in an existing container or another stream (such as the standard input stream, `std::cin`). We would like to have the extensibility found in STL algorithms and allow users to transfer data similarly to the way `std::copy` moves elements from one range to another. C++ templates can be used to parameterise the type of input and remove this restriction.

A generic range of input, then, is the first definitive aspect of our filtering component. The range is defined by two parameterised input iterators (specified by the `InputIterator` parameter), which may traverse either streams or containers. Note that in the case where the input iterator is a `std::istream_iterator`, the `operator>>` method will still be necessary to read in the `DatabaseJob` objects.

⟨database distribute example 2 36⟩ ≡

```
template <typename InputIterator>
void distribute(InputIterator begin,
                InputIterator end,
                vector<DatabaseJob>& urgent,
                vector<DatabaseJob>& normal)
{
  while(begin!=end)
  {
    DatabaseJob job=*begin++;

    if(job.IsHighPriority())
    {
      urgent.push_back(job);
    }
    else
    {
      normal.push_back(job);
    }
  }
}
```

Not used.

3.1.3 Generic Output

The next concern is that the function names a specific container for the filtered `DatabaseJob` objects. Given the standard iterator interface used by all STL containers and streams, there is no reason this code should force users to put their elements in a `std::vector`. Users should be allowed to specify any destination for the `DatabaseJob` objects in the same way they are allowed to specify a generic source of input; this output destination might be a container like `std::set` or a stream such as `std::cout`.

A generic destination, however, is only part of the answer to the question of how to parameterise the output range. Why does the output have to be composed of `DatabaseJob` objects? We are looking for an efficient, generic solution that can be easily reused, and there is no reason that this algorithm cannot be applied to other types of data. Our function is reading in objects and putting them in one of two containers given the results of a conditional, which is not database-specific at all. By replacing all references to a vector of `DatabaseJobs` with a set of output iterators, the `distribute` function takes on a completely generic I/O interface. `OutputIteratorIfTrue` and `OutputIteratorIfFalse` are the two output iterator types associated with the destinations that are chosen given the Boolean result of the predicate—they are independent of each other and may be of different types.

⟨database distribute example 3 37a⟩ ≡

```
template <typename InputIterator,
          typename OutputIteratorIfTrue,
          typename OutputIteratorIfFalse>
void distribute(InputIterator begin, InputIterator end,
                OutputIteratorIfTrue out1, OutputIteratorIfFalse out2)
{
  while(begin!=end)
  {
    typename iterator_traits<InputIterator>::value_type item;
    item=*begin++;
    item.IsHighPriority() ? *out1++=item : *out2++=item;
  }
}
```

Not used.

3.1.4 Generic Predicate

There is one glaring problem left in the algorithm, and that is the call to `DatabaseJob::IsHighPriority`. Our function refers to no specific type, and even if the type in question is guaranteed to be a class (which it is certainly not), it is not reasonable to demand that the class have an `IsHighPriority` member. The correct approach to making this solution completely reusable is to use predicates in the same way that the standard library algorithms do. In the final version of `distribute`, a fifth parameter has been added to allow programmers to name the comparator that should be used to analyse the data. This component now accepts a generic range of input, evaluates it with any user-defined predicate (specified by the `Predicate` parameter), and places it in one of two independent destinations.

⟨the distribute function 37b⟩ ≡

```
template <typename InputIterator,
          typename OutputIteratorIfTrue,
          typename OutputIteratorIfFalse,
          typename Predicate>
void distribute(InputIterator begin, InputIterator end,
                OutputIteratorIfTrue out1, OutputIteratorIfFalse out2,
                Predicate p)
{
  while (begin != end)
  {
    typename std::iterator_traits<InputIterator>::value_type
     item=*begin++;
    p(item) ? *out1++=item : *out2++=item;
  }
}
```

Used in part 45.

3.2 A More Generic Phrasing of Distribution and Filtering

In order to further generalise the algorithm and add more power to its filtering capabilities, it was necessary to remove the restriction on the number of destinations to which the input elements could be written. With this limitation eliminated, users could filter data to one or more locations based on a set of criteria that was more complex than a single Boolean check—elements could be evaluated against multiple, independent predicates and have compound actions performed against them. This was by far the most complicated issue in the development of the library, as it required the construction of a full supporting framework to offer flexibility without decreasing usability. It was clear that the resulting set of classes and/or functions would have to pair the multiple predicates and output destinations. The `router` class, along with the helper `destination` class, offers this advanced functionality by building on the ideas presented so far.

3.2.1 Arbitrary Number of Destinations

Our `router` class manages a set of predicate-output pairs through which it filters incoming data. Our `router` class manages a set of predicate-output pairs through which it filters incoming data. The most complex aspect of `router` is not the implementation of its methods but rather the parameterisation of their parameters—the class makes extensive use of member templates in order to provide the desired flexibility. The aforementioned pairs are really instances of the `destination` class.

The first building block in the GDFA framework is the `destination` class for evaluating input and performing some action against it. Users may extend the class any way they wish, but they must override the pure virtual function `distribute` or the derived class will remain abstract.

⟨the destination class 38⟩ ≡

```
template <typename T>
class destination
{
public:
  virtual ~destination() {}
  virtual bool distribute(const T& element)=0;
};
```

Used in part 45.

3.2.2 Creating Derived Classes from `destination`

The purpose of the `distribute` function is to perform an action against the parameter it receives. Technically, there is no limit or restriction on what the implementation can or should do, but the problems that the framework hopes to

solve tend to follow a common pattern: the element is tested against some criteria and written to an output based on the result of that test. The Boolean value returned by the function should be true if the router is to stop evaluation of the current element—that is, if it should move on to the next element in the input range without passing the current element to other destinations. The User's Guide (Chapter 1) has multiple examples to demonstrate how the `destination` class can be used as a base for programmer tests, but the basic idea behind most implementations can be described with the following pseudo-code:

⟨deriving from destination 39⟩ ≡

```
template <typename Output>
class new_destination : destination<type>
{
private:
  Output out;

public:
  new_destination(Output _out) : out(_out) {}

  bool distribute(const type& element)
  {
    if(some test against element)
      write element to output

      return true to stop evaluation
  }
};
```

Not used.

3.2.3 The `destination_handle` Class

For any range of input, users can create multiple `destination`s through which to filter the elements—for each `destination` created, an instance is added to the `router`, which will pass the input through the `destination`s until it is told to stop or until it has considered every one. Because users must be able to create and use multiple different destination sub-classes to be placed in a single `router`, the internal mechanism for handling these destination objects is more complicated than initially planned. Fortunately, the user interface is not affected by this complexity, and users who can understand the above pseudo-derivation from `destination` can use the GDFA Library effectively.

Introduction to Handles

The `router` class manages the destination objects created by the user in a single container. Holding many objects of different sub-types in one container without complicating the interface was a challenge, but one that was necessary since we did not want to limit the number of destinations that a router could manage. Applications that used the library to filter large batches of data might have

hundreds of predicates to evaluate the many possible attributes associated with their data; imposing a limit on the number of destinations would be imposing a limit on the acceptance of the library in mainstream software development. Furthermore, a class interface that only accepted N destinations but without the aid of member template functions would be extremely cumbersome to programme, with much of the parameterisation information having to be specified explicitly. Programming `router` to hold a single container of destinations so that all actions could be performed against them in a generic way was essential to both interface and implementation simplicity—the aforementioned increase in complexity came with the added layer of abstraction within `router` to allow for the generic treatment of `destination` objects.

The `destination`s are held indirectly by the `router` class through an handle—a wrapper that holds a pointer to a `destination` and performs all of the dynamic memory management for that pointer. This handle class, `destination_handle`, is a member of `router`, and is internal to the framework—it is used in combination with member template functions to provide seamless interaction with the user-defined `destination`s. The basic functionality of the handle—its memory management—is demonstrated below; `destination_handle` uses reference counting to offer copying and assignment semantics similar to pointers while preventing memory leaks and/or data corruption. It is assumed the reader is familiar with the basic ideas behind reference counting, and we will not review them here. The key aspect to note is that the handle holds a pointer to a `destination` and takes advantage of that class's virtual functions to invoke the correct method(s) in the concrete sub-class. Thus, object-oriented polymorphism has provided a simple solution to the problem of managing objects of different types in a single container.

⟨handle class used to store destinations 40⟩ ≡

```
class destination_handle
{
private:
  destination<T> *dest;
  std::size_t *referenceCount;

  void destroy()
  {
    if(--*referenceCount==0)
    {
      delete referenceCount;
      delete dest;
    }
  }

  priority_level priority;

public:
  ⟨member template constructor 42a⟩
```

```
    destination_handle(const destination_handle& copy)
    :dest(copy.dest),
     referenceCount(copy.referenceCount)
    {
      ++*referenceCount;
      priority=copy.priority;
    }

    destination_handle& operator=
     (const destination_handle& rhs)
    {
      ++*rhs.referenceCount;

      destroy();

      dest=rhs.dest;
      referenceCount=rhs.referenceCount;
      priority=rhs.priority;

      return *this;
    }

    bool operator< (const destination_handle& rhs) const
    {
      return priority<rhs.priority;
    }

    bool operator>(const destination_handle& rhs) const
    {
      return rhs<*this;
    }

    ~destination_handle()
    {
      destroy();
    }

    ⟨destination accessor function 42b⟩

  };
```

Used in part 42c.

Abstracting Away from Pointers

One thing it did not solve, however, was the requirement that users pass a pointer to each `destination` they added to the `router`. This was undesirable both because it decreased usability (users should be able to use the `router` without having to do any dynamic memory management) and increased the potential for bugs (`destination`s declared locally on the stack which went out

of scope after being added to the router would raise exceptions and corrupt programme results). We wanted the insertion methods of the `router` class to be similar to those of STL containers—the user would pass a reference to the given destination and know that an independent copy was being kept; such an interface is as simple as possible in terms of C++ language constructs. The `destination_handle` class uses a member template constructor to create a dynamically allocated copy of the given destination; this method is central to the `router`'s ability to accept any class built off the GDFA `destination` without having to include any reflection-like code.

⟨member template constructor 42a⟩ ≡
```
template <typename Destination>
destination_handle(const Destination& d, priority_level _priority)
:priority(_priority),
 dest(new Destination(d)),
 referenceCount(new std::size_t(1)){}
```
Used in part 40.

Access to Functionality of `destination`

The `router` accesses the distribution functionality written by the user with a simple wrapper function that passes along the given element to the `destination` class. We will revisit the handle class later when discussing the priority of `destination`s vis-á-vis input evaluation.

⟨destination accessor function 42b⟩ ≡
```
bool distribute(const T& element)
{
  return dest->distribute(element);
}
```
Used in part 40.

3.2.4 The `router` Class

With a generic handle class to take away the issues of polymorphism presented by the different `destination` sub-classes, the implementation of `router` becomes fairly simple. Most of the functionality is similar to that of an ordinary STL container, with the significant addition being the processing of an input ranges through the set of destinations according to user preference. The most basic methods of router are presented below to give an idea of its internal structure before presenting the input processing methods.

⟨the router class 42c⟩ ≡
```
template <typename T>
class router
{
public:
  typedef unsigned int priority_level;
```

```
  static const priority_level low_priority=0;
  static const priority_level normal_priority=1;
  static const priority_level high_priority=2;
  static const priority_level urgent_priority=3;

private:
  ⟨handle class used to store destinations 40⟩

  std::vector<destination_handle> destinations;
  destination_handle *defaultDestination;

  // not implemented: prohibited
  router(const router& copy);
  router& operator=(const router& rhs);

public:
  router() : defaultDestination(NULL) {}

  template <typename OutputIterator>
  router(OutputIterator first, OutputIterator last)
  :defaultDestination(NULL)
  {
    std::copy(first, last,
              std::back_inserter(destinations));
    std::make_heap(first, last);
  }

  ~router()
  {
    remove_default();
  }

  template <typename InputIterator>
  void distribute(InputIterator item)
  {
    typename std::vector<destination_handle>::iterator i;

    for (i=destinations.begin(); i!=destinations.end();
         ++i)
    {
      if(i->distribute(*item))
      {
        break;
      }
    }

    if(i==destinations.end() && defaultDestination)
    {
      defaultDestination->distribute(*item);
```

```
    }
  }

  template <typename InputIterator>
  void distribute(InputIterator first, InputIterator last)
  {
    while(first!=last)
    {
      distribute(first++);
    }
  }

  template <typename Destination>
  void add_destination(const Destination& d,
                       int _priority=normal_priority)
  {
    destinations.push_back(destination_handle(d,_priority));
    std::push_heap(destinations.begin(),
    destinations.end());
  }

  template <typename Destination>
  void set_default(const Destination& d)
  {
    remove_default();
    defaultDestination=
     new destination_handle(d,normal_priority);
  }

  void remove_default()
  {
    if(defaultDestination)
    {
      delete defaultDestination;
      defaultDestination=NULL;
    }
  }
};
```

Used in part 45.

Default `destination`

The default `destination` is an optional component that provides a way to catch any elements in the input range that did not pass any of the tests conducted by the `router`'s `destination`s. If no default is set and an element is passed to every destination without one of them returning true (to tell the router to stop evaluation), no action will be taken—the router will simply move on to the next element. When a default is specified, however, an element that does not match up against any regular `destination` will be given to the default for

further processing.

We decided to keep the default `destination` completely separate from the other `destination` objects in order to simplify the implementation—the only other option would be to keep it at the end of the container of `destination`s, and this would be cumbersome to maintain even without the existing priority scheme that will be discussed later. Storing the default as a dynamically allocated resource (that is, in terms of a pointer) allowed us to easily tell whether a default existed (by comparing the pointer to NULL) and to access it without using any obtuse iterator trickery.

From the programmer's perspective, a default `destination` class need not be any different from other sub-classes of `destination`. The only difference in its use is that instances of the default `destination` must be added or removed with special member functions (`set_default` and `remove_default`) rather than the normal `add_destination` method. Regarding implementation, the only complication that arises from our choices is that `router` now owns a dynamic resource, and must implement a destructor to prevent memory leaks.

Priorities

The `router` class offers the ability to assign priorities to the `destination`s it manages. Four built-in priorities (`low_priority`, `normal_priority`, `high_priority`, and `urgent_priority`) are available to users so that they can ensure that some `destination`s are always considered before others, no matter the order in which they were added. The user specifies the `destination`'s priority when adding it to the `router`, with the default being `normal_priority` (this means that if the user has no interest in using the priority scheme, he or she will not be affected by it). As each `destination` is inserted, the `router` uses the binary heap functions of the Standard Library to adjust the `std::vector` of `destination` objects it keeps; by storing the elements as a binary heap, the `router` is able to keep them prioritised very efficiently (the amount of copying that will be done for each heap adjustment will be equal to $O(\lg N)$, and the copies themselves will only be of the lightweight handles, not the actual `destination` objects).

3.3 Trivial `distribute`, `destination`, and `router` Code

To finish everything up, we combine everything and define the header file which will form the basis of the GDFA Library. All that we need to do is add the standard `#ifdef-#endif` block to avoid issues with multiple inclusions of the file, include some standard library components, and define the `gdfa` namespace.

```
"router.hpp" 45 ≡
     #ifndef _router_
     #define _router_

     #include <algorithm>
```

```
#include <functional>
#include <iterator>
#include <vector>

namespace gdfa
{
    ⟨the distribute function 37b⟩
    ⟨the destination class 38⟩
    ⟨the router class 42c⟩
}

#endif
```

Chapter 4

Testing Approach and Results

4.1 Overview

This section includes a description and motivation behind all of the tests performed on the Generic Distribution and Filtering Algorithms. The results of those tests and a discussion of the results are also presented here.

Our testing strategy consists of tests for performance and correctness. The performance tests consist of timed tests and operation counting. Correctness and Performance testing was performed on all key features of the Generic Distribution and Filtering Algorithms. Correctness testing was performed on many conceivable use cases implemented on a thorough set of built-in types and STL types. Timed tests and Operation Counts were performed in various configurations in order to time (and, likewise, to count) different aspects of the algorithms.

4.1.1 Notation

In some of the following tests, the order, or O-bound (asymptotic complexity), of the algorithm is tested. The algorithm is initially assumed to operate in $O(MN)$ time. N represents the algorithm input size or the number of elements fed through the algorithm. These are the elements to be distributed. M represents the number of destination classes used (including the default destination).

The term "growth rate" used here is defined as the increase in measured execution time when N is doubled. Hence, we can define growth rate by the formula $G(N) = T(N)/T(N/2)$ where $G(N)$ is the growth function and $T(N)$ is the measured execution time of an N-element input. The growth rate for an algorithm of linear complexity would be 2. This measurement allows us to look more closely at the time increase relative to N. The increase from $N/2$ to N is precisely calculated and graphed. In all cases the calculated growth rate is approximately 2, which indicates a linear time complexity.

4.1.2 Computer Testing Configuration

The performance tests presented here were performed on an IBM ThinkPad Computer with 1.8 GHz Mobile Pentium 4 with Intel SpeedStep technology and 256MB of RAM. All code was compiled with g++ version 3.2 with the "-O3" flag. Performance on additional platforms, in particular the Rensselaer Polytechnic Institute CS Department Solaris Server (sparc-sun-solaris2.8)(compiled under g++ 3.2) was tested and relative (O-bound) performance was comparable to that generated by the Pentium machine.

The correctness tests presented here were also performed on an IBM ThinkPad Computer with 1.8 GHz Mobile Pentium 4 with Intel SpeedStep technology and 256MB of RAM. All code was compiled with g++ version 3.2 with the "-O3" flag. The correctness tests were run on additional platforms, in particular a Macintosh ibook with a 700MHz PowerPC G3 processor, 640MB RAM, running OSX 10.2.2 and the results were identical.

4.2 Timed Tests

4.2.1 Running Time Tests: (Time vs. N)

The following 3 timed tests attempt to test for 3 different configurations of the algorithm. All 3 tests are intended to measure the running time of the algorithm with respect to input N (the number of elements in the input range).

Timed Test 1a (Time vs. N): Each Element Distributed to m Destinations

Description This test runs the full router class using 3 destinations plus 1 default destination. This configuration attempts to capture a worst-case for the algorithm. As such, each destination object will insert the element into their respective destination, and then pass the element on to the next destination object (by returning `false`). The code implementing this test can be found in Appendix A.1.1.

Test Objective This test determines the order of the algorithm with respect to the size of the input, N. The input is the number of elements fed in through the algorithm via an iterator range.

Results The results of this test, as represented in Figure 4.1, show that the algorithm runs in linear time with respect to N. The growth rate results results, as represented in Figure 4.2 confirm these findings; in fact, the growth rate graph shows the temporal complexity of the algorithm to be consistently linear. This test is very useful in showing the variation in running time because it emphasises the most significant (time-wise) part of the algorithm, which is inserting the element into the destination via an iterator. As explained above,

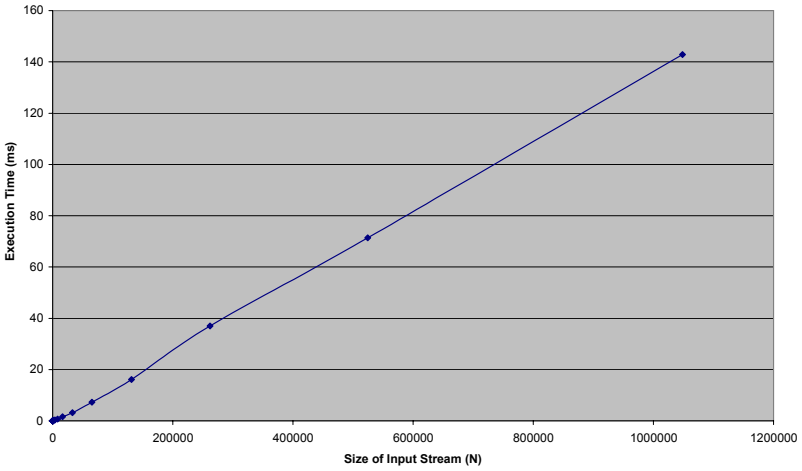

Figure 4.1: Results of Timed Test 1*a*

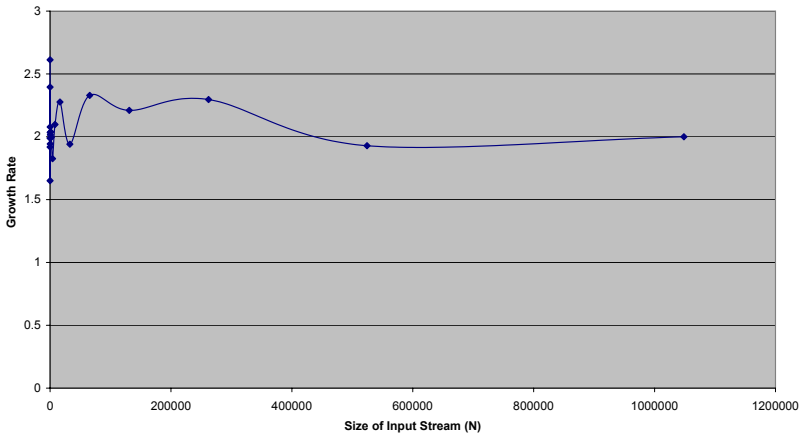

Figure 4.2: Growth Rate of Timed Test 1a

this test forces a worst-case scenario and likely does not represent a typical implementation. This atypical configuration combined with the observation that element insertion dominates the running time (and masks the decision making time) results in the need for additional testing.

Timed Test 1b (Time vs. N): Each Element Distributed to Exactly 1 Destination

Description This test runs the full router class using 3 destinations plus 1 default destination. This configuration attempts to capture a typical case for the algorithm. As such, the elements read in on the input iterator will each be distributed to 1 and only 1 destination. This is accomplished as follows. The primary destinations are set up so that the elements are always passed through to the next destination. Each element is then distributed by the default destination. The code implementing this test can be found in Appendix A.1.2.

Test Objective The purpose of this test is to determine the order of the algorithm with respect to the size of the input, N. The input is the number of elements fed through the algorithm via an iterator range.

Results The results of this test, as presented in Figure 4.3, show that the algorithm runs in linear time with respect to N. The growth rate results, as represented in Figure 4.4 confirm these findings. The growth rate graph, in fact, demonstrates that the temporal complexity of the algorithm is consistently linear. This test reflects a different case than Test 1a and is arguably a more typical case. This test de-emphasises, to some extent, the most significant (time-wise) part of the algorithm, which is inserting the element into the destination via an iterator. This test creates a worst-case scenario regarding the number of decisions concerned because, although each element is distributed by just one destination object, it is always the default destination. This typical configuration complements and confirms the above results.

Timed Test 1c: (Time vs. N) - Each Element Distributed to 0 Destinations

Description This test runs the full router class using 3 destinations plus 1 default destination. This configuration attempts to capture the decision making time complexity of the algorithm. To do this, the destination predicates were designed so that every element fit into every category and every destination class passed the element to each succeeding destination class. Therefore, M decisions were made for each element. However, in order to measure this, the time of the insertion of the element into the input iterator had to minimise. Therefore, each destination does not actually insert the element, it just decides if it should and passes the element on to the next destination (by returning `false`). Each destination object will insert the element into their respective destination, and

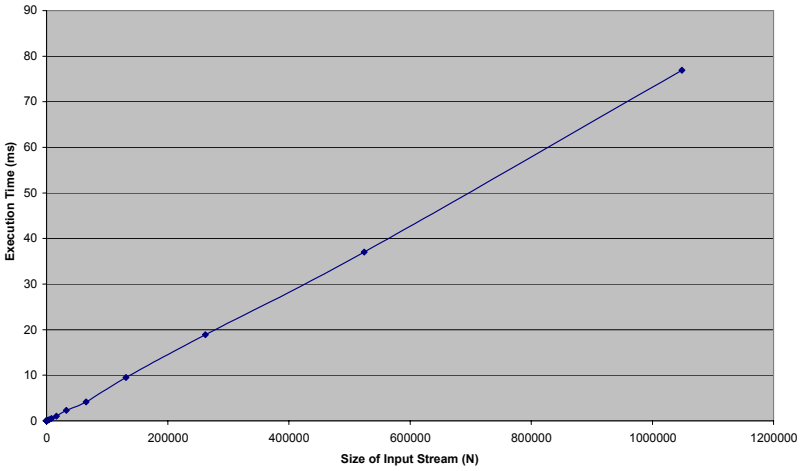

Figure 4.3: Results of Timed Test 1*b*

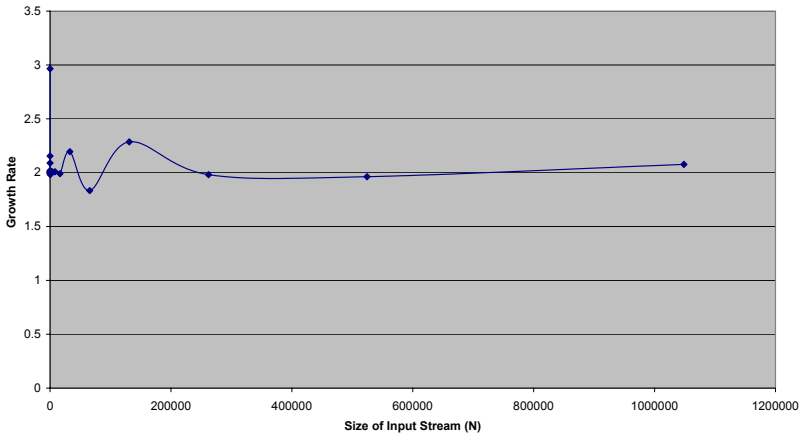

Figure 4.4: Growth Rate of Timed Test 1*b*

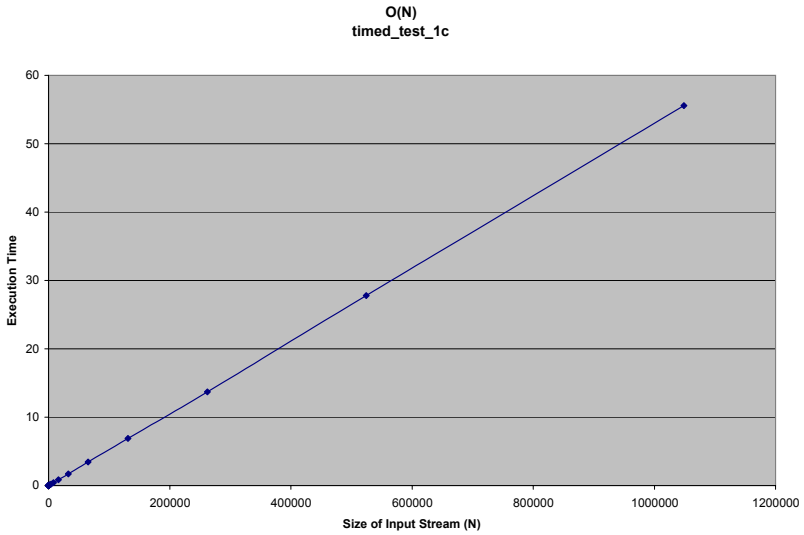

Figure 4.5: Results of Timed Test 1*c*

then pass the element on to the next destination object (by returning `false`). The code implementing this test can be found in Appendix A.1.3.

Test Objective The purpose of this test is to determine the order of the algorithm with respect to the size of the input, N. The input is the number of elements fed through the algorithm via an iterator range. This test is designed to factor out the time component associated with the insertion of the element via the input iterator in the destination object because this time is independent of this algorithm. This enables us to more accurately measure the other aspects of this algorithm.

Results The results of this test, as represented in Figure 4.5, show that the algorithm runs in linear time with respect to N. The growth rate results, as represented in Figure 4.6 confirm these findings. In fact, this growth rate graph illustrates that the complexity of the algorithm is perfectly linear. This test reflects a different case than Test *la* and Test 1*b*. This test entirely de-emphasises a large time component of the algorithm, which is inserting the element into the

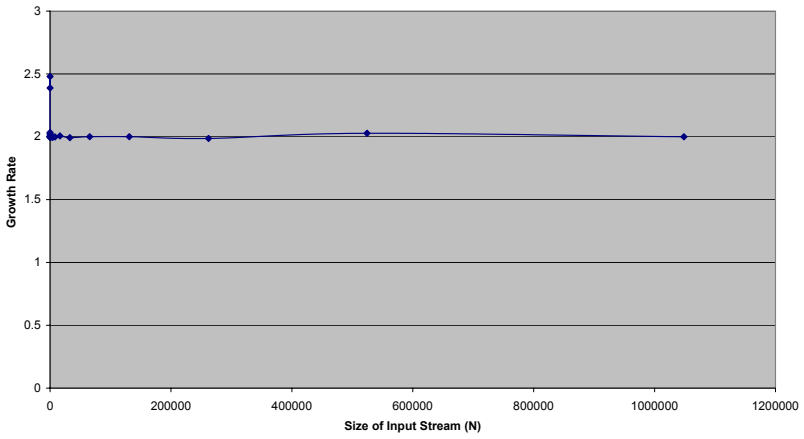

Figure 4.6: Growth Rate of Timed Test 1c

destination via an iterator. This test creates a worst-case scenario regarding the number of decisions. This configuration complements and confirms the above results.

4.2.2 Running Time compared to M destinations (Time vs. M)

The following 3 timed tests attempt to test for 3 different configurations of the algorithm. All 3 tests are intended to measure the running time of the algorithm with respect to the number of destination objects added to the router object (M).

Timed Test 2a (Time vs. M): Each Element Distributed to m Destinations

Description This test runs the full router class using M destinations plus 1 default destination (where M varies from 0 to 1048576). This configuration attempts to capture a worst-case for the algorithm. As such, each destination object will insert the element into their respective destination, and then pass the element on to the next destination object (by returning `false`). In this test N is held constant. The code for this test may be found in Appendix A.2.1.

Test Objective The purpose of this test is to determine the order of the algorithm with respect to the number of destination objects added to the router object (M).

Results The results of this test show that the algorithm runs in linear time with respect to M. The growth rate results, as represented in Figure 4.8 confirm these findings.[1] This test is very useful in showing the variation in running time because it emphasises the most significant (time-wise) part of the algorithm, which is inserting the element into the destination via an iterator. This test forces a worst-case scenario and likely does not represent a typical implementation. This atypical configuration combined with the observation that element insertion dominates the running time (and masks the decision making time) results in the need for additional testing.

Timed Test 2b (Time vs. M)

Description This test runs the full router class using M destinations plus 1 default destination (where M varies from 0 to 1048576). This configuration attempts to capture a typical case for the algorithm. As such, each element is distributed by exactly 1 destination object. In this test N is held constant. The code for this test may be found in Appendix A.2.2.

[1] See Section 12 for a possible explanation of the graph's behaviour.

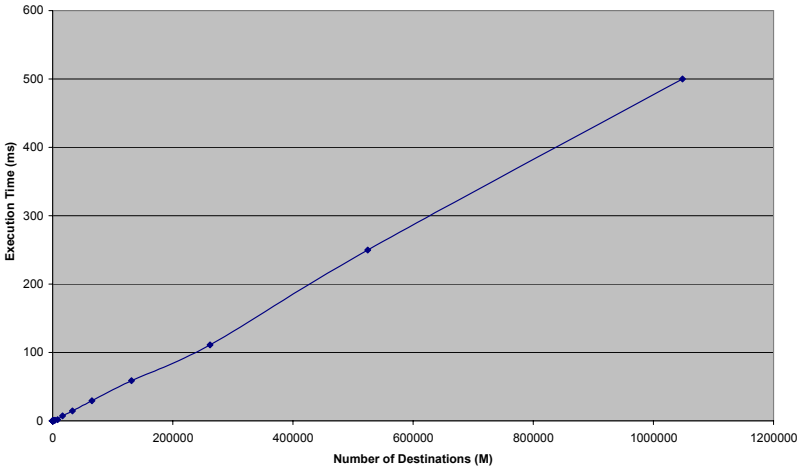

Figure 4.7: Results of Timed Test 2*a*

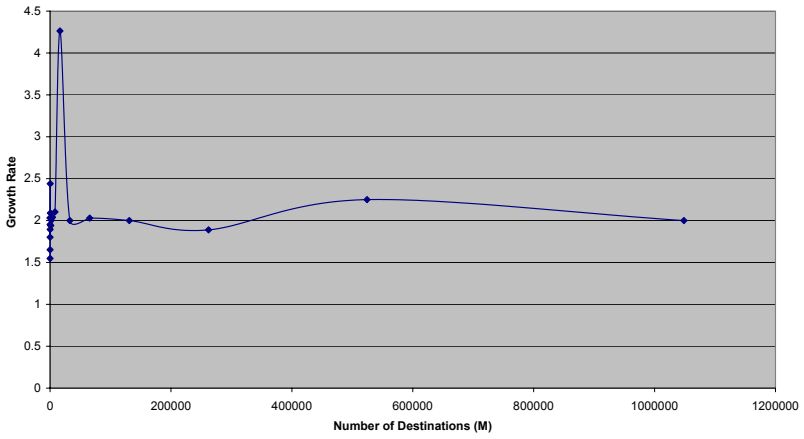

Figure 4.8: Growth Rate of Timed Test 2a

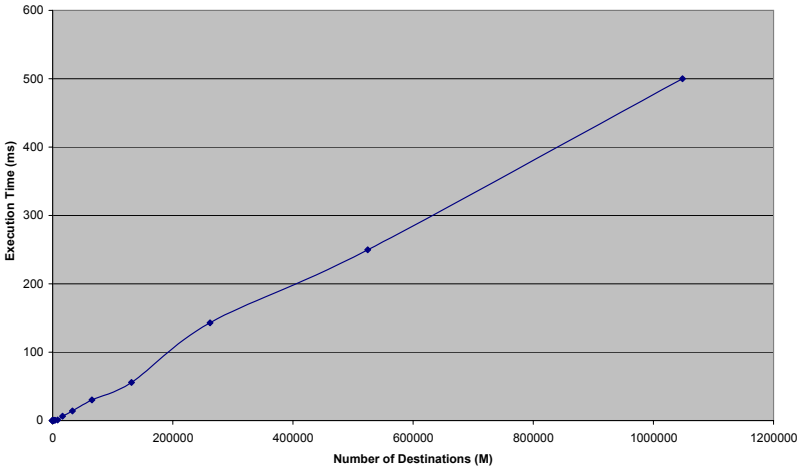

Figure 4.9: Results of Timed Test 2*b*

Test Objective The purpose of this test is to determine the order of the algorithm with respect to the number of destination objects added to the router object (M).

Results The results of this test, as represented in Figure 4.9, show that the algorithm runs in linear time with respect to M. The growth rate results, as represented in Figure 4.10 confirm these findings.[2] This typical configuration confirms the results of Test 2*a*.

Timed Test 2c (Time vs. M)

Description This test runs the full router class using M destinations plus 1 default destination (where M varies from 0 to 1048576). This configuration attempts to measure the decision making time component and de-emphasising the insertion component. This is accomplished by having destinations (for the purpose of testing) that do not actually insert the element into the input iterator.

[2]See Section 12 for a possible explanation of the graph's behaviour.

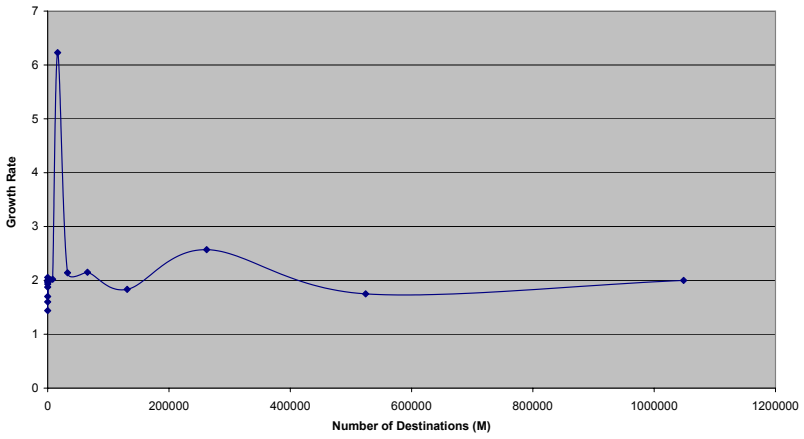

Figure 4.10: Growth Rate of Timed Test 2*b*

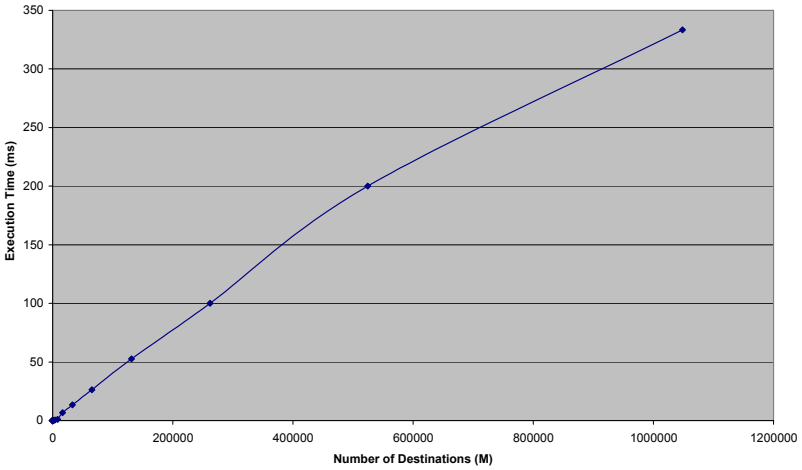

Figure 4.11: Results of Timed Test 2*c*

They just make a decision and pass the element on to the next destination object. In this test N is held constant. The code for this test may be found in Appendix A.2.3.

Test Objective The purpose of this test is to determine the order of the algorithm with respect to the number of destination objects added to the router object, M. This test is designed to factor out the time component associated with the insertion of the element via the input iterator in the destination object because this time is independent of this algorithm. This enables us to more accurately measure the other aspects of this algorithm.

Results The results of this test, as represented in Figure 4.11, show that the algorithm runs in linear time with respect to M. The growth rate results, as represented in Figure 4.12 confirm these findings.[3] This test reflects a different case than Test 1*a* and Test 1*b*. This test entirely de-emphasises a large time component of the algorithm, which is inserting the element into the destination

[3]See Section 12 for a possible explanation of the graph's behaviour.

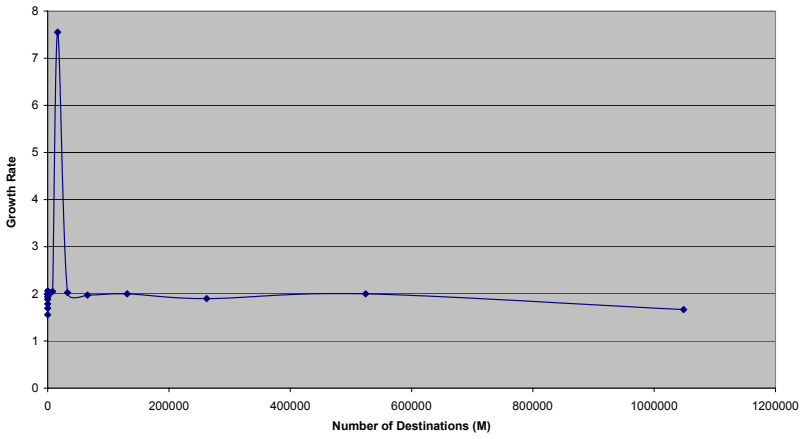

Figure 4.12: Growth Rate of Timed Test 2c

via an iterator. This test creates a worst-case scenario regarding the number of decisions. This configuration complements and confirms the above results.

Note Regarding Growth Rate

The growth rate graphs for Section 4.2.2 reveal some deviation from strict linearity. This behaviour is probably due to the re-allocation of memory for the `std::vector`'s array sub-structure. The re-allocation may have occurred toward the end of the test run and the time required to perform this costly operation was not equally amortised over runs of different input sizes (N).

4.2.3 Running Time Test (Time vs. N): Simple Version

The test presented in this section is intended to measure the running time of the simple (2-predicate) functional version of the algorithm with respect to input N (the number of elements in the input range).

Description

In this configuration, each element is distributed to exactly 1 `destination`. This test uses the simple 2-predicate function version of distribution. The `destination`s are set up so that the elements are distributed to exactly 1 of the `destination`s. The code for this test is located in A.3.

Test Objective

The purpose of this test is to determine the order of the algorithm with respect to the size of the input N. The input is the number of elements fed through the algorithm via an iterator range.

Results

The results of this test, as represented in Figure 4.13, show that the algorithm runs in linear time with respect to N. The growth rate results, as represented in Figure 4.14 confirm these findings.[4] This test is designed similarly to Test 1*b* (see Section 4.2.1). In this test, however, each element is distributed to the first (of 2) destinations.

Note Regarding Growth Rate

The sub-linear time increases are probably due to the memory being re-allocated toward the end of the previous run. Then, the next run may not have had to re-allocate memory as many times or at all, resulting in sub-linear time increase for that run.

[4] See Section 13 for an explanation of the slightly anomalous behaviour of the graph.

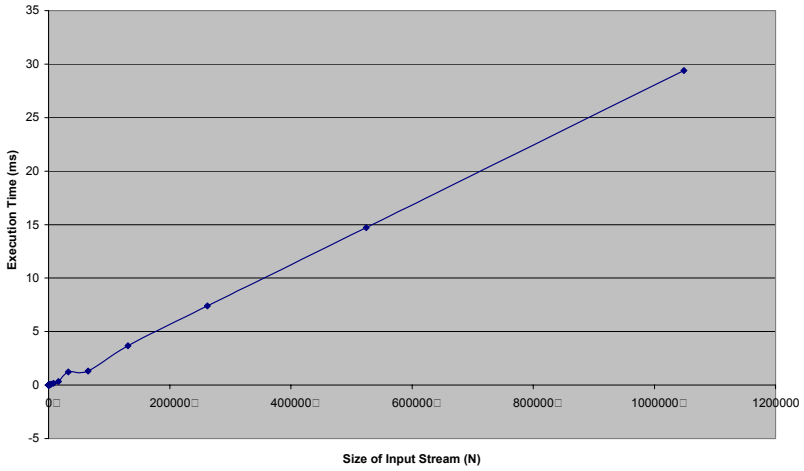

Figure 4.13: Results of Timed Test 3

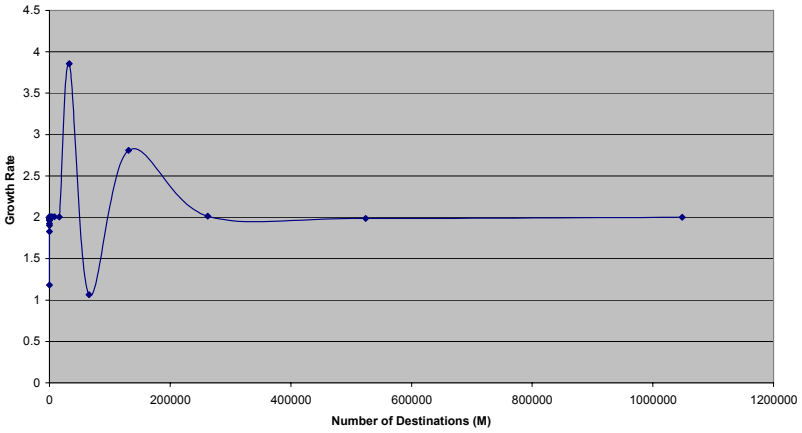

Figure 4.14: Growth Rate of Timed Test 3

4.3 Operation Counting

This section presents a group of tests performed on the GDFA. The tests are constructed to reinforce the results derived from the timed test in section timed tests. As such, these tests run the GDFA in 5 of the configurations used in the timed tests. These tests use operation counting to measure the complexity of the GDFA with respect to the size of the input range (N) and the number of `destinations` (M). Two of the timed tests that were performed were specifically designed to overcome the limitations of timing generic algorithms. However, since the motivation behind operation counting is also to overcome the limitations of timing generic algorithms, those 2 previous tests do not need to be performed again.

The tests in this section are based on the testing method presented in [1]. These evaluation methods are useful for evaluation generic algorithms such as GDFA. These tests make use of the counting adaptors implemented [1].[5] These libraries provide adaptors which count the operations supported by a given type. In particular, the tests presented in this section make use of the `iterator_counter` and `value_counter` for assignment, comparison and other operations. The graphs presented here compare the sum of the assignment operations, comparison operations, other operations and total operations against the GDFA variable parameters, specifically the size of the input range (N) and the number of `destinations` (M).

4.3.1 Operation Counts vs. N

The following 2 timed tests perform operation counting tests for 2 different configurations of the algorithm. Both tests are intended to measure the assignment operations, comparison operations, other operations and total operations of the algorithm with respect to Input N (the number of elements in the input range), where N varies from 0 to 1048576.

Operation Counting Test 1a: Each Element Distributed to M=4 Destinations

Description This test runs the full router class using 3 `destinations` plus 1 default `destination`. This configuration attempts to capture a worst-case for the algorithm. As such, each `destination` object will insert the element into its respective `OutputIterator`, then pass the element on to the next `destination` object (by returning its Boolean value `false`). The element type in this test is `int`.

Test Objective The objective of this test is to determine the complexity of the algorithm with respect to the size of the input range (N). The input is the number of elements fed through the algorithm via an iterator range. The code implementing this test may be found in Appendix B.2.1.

[5] The code for this library is included in Appendix B.1.

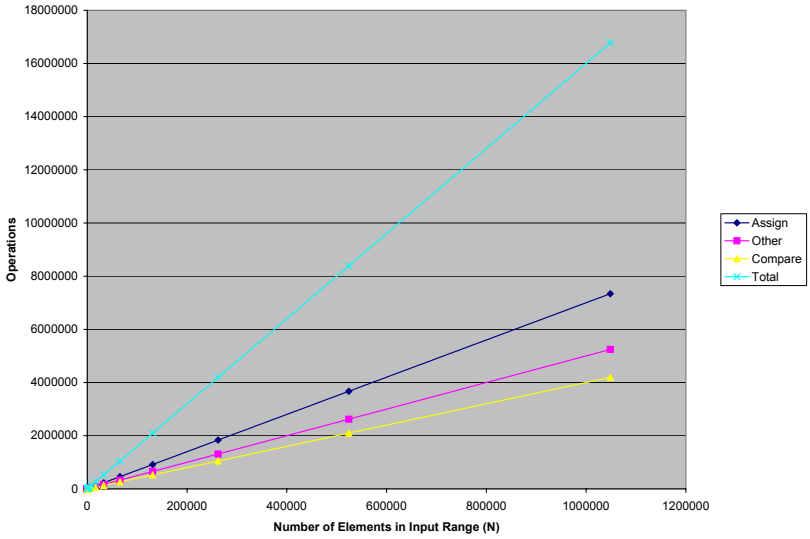

Figure 4.15: Operation Counts for Test 1*a*

Results The results of this test, as represented in Figure 4.15 show that the algorithm runs in linear time with respect to N. This typical configuration confirms the results of Timed Test 1a.

Operation Counting Test 1b: Each Element Distributed 1 Destination

Description This test runs the full `router` class using 3 `destination`s plus 1 default `destination`. This configuration attempts to capture a typical case for the algorithm. As such, the elements read in via the input iterator will each be distributed to 1 (and only 1) `destination`. This is accomplished as follows: The primary `destination`s are set up so that the elements are always passed through to the next `destination` object (by returning the Boolean value `false`). Each element is then distributed by the default `destination`. The element type in this test is also `int`. The code for this section may be found in Appendix B.2.2.

Test Objective The purpose of this test is to determine the complexity of the algorithm with respect to the size of the input (N). The input is the number of elements fed through the algorithm via an iterator range.

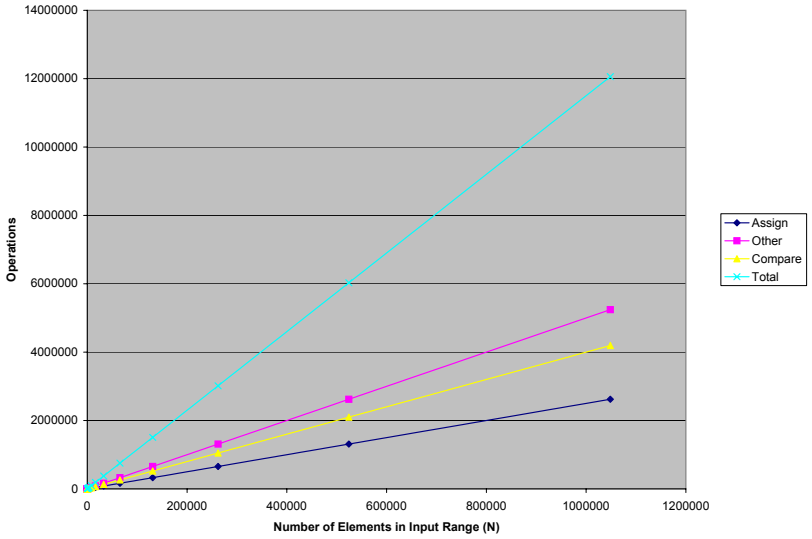

Figure 4.16: Operation Counts for Test 1*b*

Results The results of this test, as represented in Figure 4.16, show that the algorithm runs in linear time with respect to N. This typical configuration confirms the results of Operation Counting Test 1*a* as well as Timed Test 1*b*.

4.3.2 Operation Counts vs. M

The following 2 Operation Counting tests are run on 2 different configurations of the algorithm. Both tests are intended to measure the assignment operations, comparison operations, other operations and total operations of the algorithm with respect to the number of `destination` objects added to the router object M.

Operation Counting Test 2a: Each Element Distributed to M Destinations

Description This test runs the full `router` class using M `destination`s plus 1 default `destination` (where M varies from 0 to 1048576). This configuration attempts to capture a worst-case scenario for the algorithm. As such, each

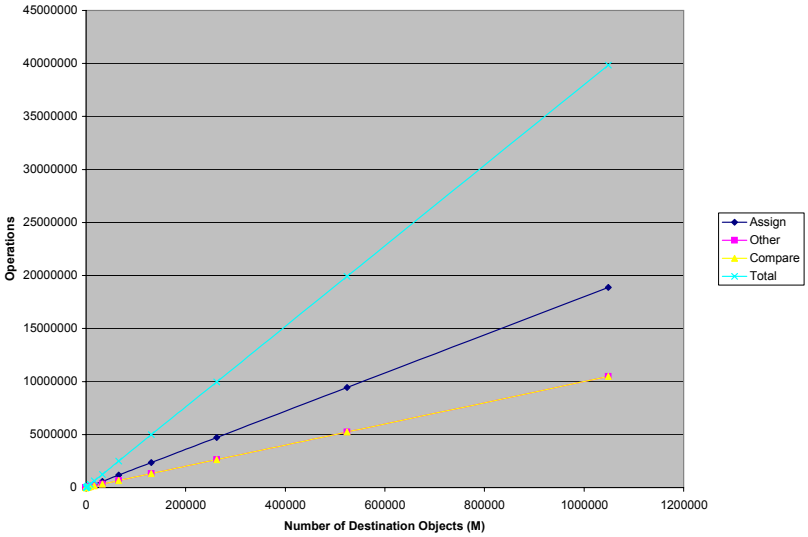

Figure 4.17: Operation Counts for Test 2*a*

`destination` object will insert the element into their respective `destination`, and then pass the element on to the next `destination` object (by returning the Boolean value `false`). In this test N is held constant. The element type in this test is also `int`. The code for this test may be found in Appendix B.3.1.

Test Objective This test aims to determine the complexity of the algorithm with respect to the number of destination objects (M) added to the `router` object.

Results The results of this test, as represented in Figure 4.17, show that the algorithm runs in linear time with respect to M. This typical configuration confirms the results of Timed Test 2*a*.

Operation Counting Test 2b: Each Element Distributed 1 Destination

Description This test runs the full `router` class using M `destination`s plus 1 default `destination` (where M varies from 0 to 1048576). This configuration attempts to capture a typical case for the algorithm. As such, each element is

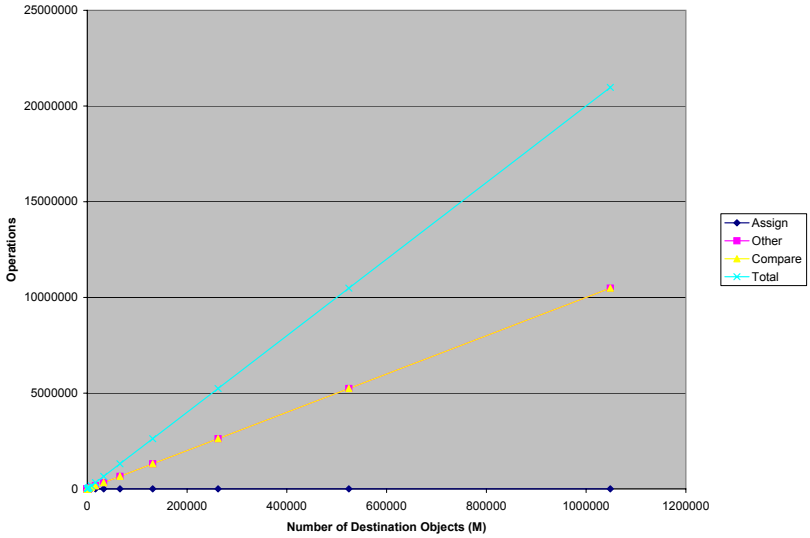

Figure 4.18: Operation Counts for Test 2*b*

distributed by exactly 1 `destination` object. In this test N is held constant. The element type in this test is int. The code for this test may be found in Appendix B.3.2.

A notable characteristic of the results of this test is the number of assignment operations. While the overall results of the test show that the algorithm is linear with respect to M, the number of assignment operations (`iterator_counter` + `value_counter`) is constant with respect to M. This is because each element is distributed by exactly 1 destination object while the size of the input range (N) is held constant.

Test Objective Here, we want to determine the order (asymptotic complexity) of the algorithm with respect to the number of `destination` objects (M) added to the `router` object.

Results The results of this test, as represented in Figure 4.18, show that the algorithm runs in linear time with respect to M. This typical configuration confirms the results of Operation Counting Test 2*a* as well as Timed Test 2*b*.

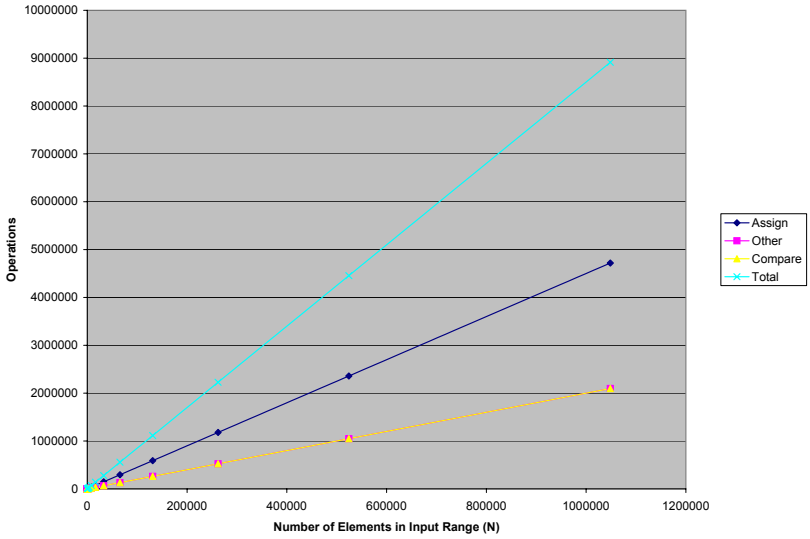

Figure 4.19: Operation Counts for Test 3

4.3.3 Simple Version: Operation Counts vs. N

Description

In this configuration, each element is distributed to exactly 1 `destination`. This test uses the simple 2-predicate function version of `distribute`. The `destination`s are set up so that the elements are distributed to exactly 1 of the `destination`s. The element type in this test is `int`. The code for this test may be found in Appendix B.4.

Test Objective

We wish to determine the order of the algorithm with respect to the size of the input range (N) (where N varies from 0 to 1048576). The input range is the number of elements in fed through the algorithm via an iterator range.

Results

The results of this test, as represented in Figure 4.19 show that the algorithm runs in linear time with respect to N. This test is designed similarly to Test 1*b*. In this test, however, each element is distributed to the first (of 2) `destination`s. This test also confirms the results of Timed Test 3.

4.3.4 Operation Counts vs. N and M

Description To more accurately describe the behavior of the GDFA `router` functionality, we now vary both the number of `destination`s (M) and the size of the input to be filtered (N). This test runs the full `router` class using M `destination`s plus 1 default `destination` (where M varies from 0 to 16384) and N input objects (where N varies from 1 to 16384). This configuration attempts to capture the trends that emerge as both M and N vary. To do this, we measure the number of total operations of the algorithm with respect to M and N. The code for this test may be found in Appendix B.5.

Test Objective In this test we attempt to demonstrate that the behaviour of the `router` class has $O(MN)$ asymptotic complexity with respect to the number of operations performed given M destinations and and input of range size N.

Results The results of this test, as represented in Figure 4.20, confirm by the fact that the graph is planer that the performance of the `router` class is as expected—$O(MN)$.

4.4 Correctness Tests

This section presents a large group of tests performed on the `router` class. The tests are constructed to validate the performance of the Generic Distribution and Filtering Algorithms. As such, the tests run the algorithm in various configurations using various built in and standard types. The output of the GDFA along with the input to the GDFA is given to a correctness valid method for that particular test. The correctness validate method performs checks, calculations and comparisons to determine if the output from the GDFA is correct for the given input.

4.4.1 Common Validation Methods

Many of the tests presented in this section use a common set of validate methods. (Other tests use validate methods specialised for those particular tests.) Code for the common set of validate methods is presented in Appendix C.1.

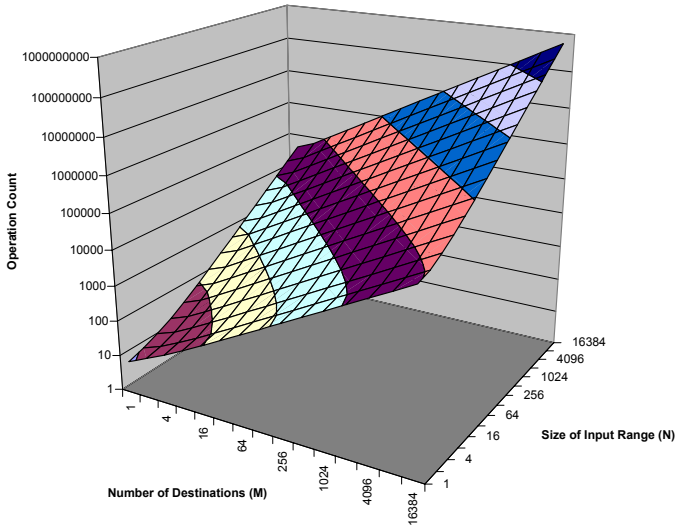

Figure 4.20: Operation Counts for Test 4

Common Validation 1

The first validate method iterates through 2 groups of data and determines if they are identical or not. The tests for which they are used have a baseline data set. The baseline data set is the result of distributing the data using a previously validated configuration. The types and priorities used are different; however, the desired result is known to be the same. Therefore, this method can be used to validate configurations for which these relationships exist. The code for this method is located in Appendix C.1.1.

Common Validation 2 (Size Validation)

The second validate method (validate size) works on an associative container of elements (`set`, `multiset`). The code for this method is located in Appendix C.1.2.

4.4.2 Helper Classes

The tests presented in this section all instantiate `router` objects. Most of the tests also instantiate `destination` objects which are added to the `router` objects used in the tests. The `destination` objects are types derived from the `destination` base class provided as part of the GDFA. The tests presented here use 10 different derived `destination` types.

The following tests use the first set of helper classes (defined in Appendix C.2.1):

Test	Sub-test
word_sorter	(fall through)
length_counter	(fall through)
contains_string	(fall through)
contains_letter	(fall through)
My_default	(fall through)

The following tests use the second set of helper classes (defined in Appendix C.2.2):

Test	Sub-test
word_sorter	(no fall through)
length_counter	(no fall through)
contains_striing	(no fall through)
contains_letter	(no fall through)
my_default	(no fall through)
word_sorter_map	(no fall through)
my_default_map	(no fall through)

The `word_sorter` classes distribute words that begin with a particular letter. The `length_counter` classes distribute words of a particular length. The

`contains_string` classes distribute words that contain a particular letter. The `contains_letter` classes distribute words that contain a particular letter. The `my_default` classes distribute all words. It is to be used as the default destination. The classes that are designated as "fall through" will always pass the element (word) on to the next destination handled by the router.

The classes that are designated as "no fall through" will not pass the element (word) on to the next destination handled by the router if the element fits its distributing requirements. In other words, a "no fall through" destination object will either distribute the element or pass it on the next destination object handled by the router, but not both.

The classes that are designated as "map" (such as `word_sorter_map` and `my_default_map`) are specialised versions which works with pairs. These are needed for distributing from an `std::map`.

4.4.3 Test Files

Many of the tests presented in this section perform operations on an input range containing `char`s or `std::string`s. The algorithms that are tested require access to the input range via `InputIterator`s. The following tests all require a text input file to be used as the source of `char`s and `std::string`s:

- `nodestinations`
- `noinput`
- `testreadcont`
- `testarray`
- `test_inserters`
- `test_char_istream`
- `test_string_istream`
- `testrwfiles`

This file is redirected to standard input in all cases, except for `testrwfile.cpp`. In the `testrwfile.cpp` test, the file is indicated as a command-line argument. These tests are run on the text file [2].

Some of the tests presented in this section perform operations on an input range containing numbers. The following tests all require an input file of numbers to be used as a data source:

- `test_int_istream`
- `test_double_istream`

These files are generated by the following test helper programmes:

- `genints`

- `gendoubles`

These programmes are all found in Appendix C.

4.4.4 Testing with Built-In Types

Test Objective

This test is designed to check that the GDFA works properly when computing on an input range of type `char`, `std::string`, `int`, and `double`. This set is comprised of 4 tests, 1 for each type.

Description of Tests

All of the tests described in this section use an `std::vector` as the input container. Therefore, the use of a `std::vector` and its associated interator is also tested.

`char` Type Test In the first test, a `router` object operates on a range of `char`. The `router` is set up to sort the range of `char`s into a set of containers. There are 27 containers: one for each letter of the alphabet and a default container for all other characters. Each of these containers is associated with a `destination` object via an iterator.

The desired output of the test is known based on the purposeful construction of the input range. An input range of 27000 characters is generated. The range consists of 1000 instances of each letter of the alphabet as well a 1000 instances of the character '{'. Once the input is generated, it is shuffled using the `std::random_shuffle` function. Then the test is run.

After testing, each container should contain (and only contain) 1000 instances of the character associated with that container.

The validate method for this test checks the number of elements in each container and then makes sure that there are no misplaced characters. It checks for misplaced characters by iterating through each container to check that only the correct character is in that container.

The code for this test may be found in Appendix C.3.1.

`std::string` Type Test The second test uses the word sorter helper class (described in Section 4.4.2) to sort `std::string`s in the input range based on its first character (insensitive to case).

As in the above test, the validating of this test is dependent on the purposeful construction of the data in the input range. An input range consisting of 3-letter words (character sequences) is generated. The sequence consists of every 3 letter combination of the letters of the 26-character English alphabet as well as the non-alphabetical character '{'. This results in 27^3 input strings and 27^2 strings beginning with each character. The generated data is shuffled using the `std::random_shuffle` function. Then the test is run.

After testing, the validation test iterates through each container to check that each string in the container does in fact start with the correct letter and that each container has 27^2 strings.

The code for this test may be found in Appendix C.3.4.

`int` Type Test The third test uses a `destination` called range checker. This `destination` distributes elements based on a value range.

For this test, 10 range checker destination objects are instantiated. Each checks a sub-range in the range $[0, 1000000)$. The sub-ranges are equal in size and are constructed as follows:

- $(0, 100000)$
- $(100000, 200000)$
- $(200000, 300000)$
- $(300000, 400000)$
- $(400000, 500000)$
- $(500000, 600000)$
- $(600000, 700000)$
- $(700000, 800000)$
- $(800000, 900000)$
- $(900000, 1000000)$

Like the above test, the validating of this test is dependent on the purposeful construction of the data in the input range. For this test data for the input range are generated, consisting of `int`s in the range $[0, 1000000)$. The input data are shuffled using the `std::random_shuffle` function.

Based on the construction of the destination ranges and the overall range of the input data, after distribution, there should be 99999 elements in each range and 10 elements that get caught by the default case $(0, 100000, 200000, \ldots, 900000)$.

Therefore, the validate method for this test checks the size of the respective resultant containers. The validate method also checks the containers to be sure that all elements contained in them fall within their respective ranges. This is quickly accomplished by using the `std::min_element` and `std::max_element` functions from the Standard Library.

Type `int` works in the same way as type `long` and type `short`. In fact, on most computers, `int`s are equivalent either to `short` or to `long`. The range of values for an `int` is at least the same as the range for `short int`s and no larger than the range for `long`s. Therefore, it is assumed that if GDFA works properly with `int`s, it will also work properly with these other similar types.

The code for this test may be found in Appendix C.3.3.

`double` Type Test The forth test also uses a destination called range checker to distributes elements based on a specified value range.

For this test, 10 range checker destination objects are instantiated. Each checks a sub-range in the range $[0, 1)$. The sub-ranges are equal in size and are constructed as follows:

- $(0.0, 0.1)$
- $(0.1, 0.2)$
- $(0.2, 0.3)$
- $(0.3, 0.4)$
- $(0.4, 0.5)$
- $(0.5, 0.6)$
- $(0.6, 0.7)$
- $(0.7, 0.8)$
- $(0.8, 0.9)$
- $(0.9, 1.0)$

Like the above test, validation of this test is dependent on the purposeful construction of the data in the input range. For this test data for the input range is generated, consisting of integers in the range $[0, 1)$. The input range is constructed starting with 0.0. Each subsequent value is incremented by 0.000001. The input data is shuffled using the `std::random_shuffle` function.

Based on the construction of the destination ranges and the overall range of the input data, after distribution, there should be 99999 elements in each range and 10 elements that get caught by the default case $(0.0, 0.1, 0.2, \ldots, 0.9)$.

Therefore, the validate method for this test checks the size of the respective resultant containers. The validate method also checks the containers to be sure that all elements contained in them fall within their respective ranges. This is quickly accomplished by using the `std::min_element` and `std::max_element` functions from the Standard Library.

Type `double` functions in the same way as type `float` (although the range and precision differ). Therefore, it is assumed that if GDFA works properly with type `double`, it will also work properly with type `float`.

Note that floating point accuracy causes inconsistencies in the comparison of elements of type `double`. Therefore, the input generator, `destination`s, and `validate` method make use of a variable `epsilon` which is the precision that we use for distribution and validation purposes. Specifically, two `double`s, `one` and `two`, are considered to be the "equal" if `abs(one - two)<epsilon`. A class was created to facilitate the comparison based on this predetermined level of precision. This class was necessary to assure correct filtering. It is incumbent on the user to instill floating point precision comparison consistency.

The code for this test may be found in Appendix C.3.2.

Results

The following are outputs from the respective test programmes; each indicates that the GDFA Library passes the test in question.

`test_char.hpp` Output

⟨output from test_char.hpp 79a⟩ ≡

```
Test:  test_char.cpp

Test Using type char as input.
**********************************

Results:

a: 1000
b: 1000
c: 1000
d: 1000
e: 1000
f: 1000
g: 1000
h: 1000
i: 1000
j: 1000
k: 1000
l: 1000
m: 1000
n: 1000
o: 1000
p: 1000
q: 1000
r: 1000
s: 1000
t: 1000
u: 1000
v: 1000
w: 1000
x: 1000
y: 1000
z: 1000
Number of elements caught by default case: 1000
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

`test_string.cpp` Output

⟨output from test_string.cpp 79b⟩ ≡

```
Test:  test_string.cpp

Test Using strings as input.
*********************************

Results:

a: 729
b: 729
c: 729
d: 729
e: 729
f: 729
g: 729
h: 729
i: 729
j: 729
k: 729
l: 729
m: 729
n: 729
o: 729
p: 729
q: 729
r: 729
s: 729
t: 729
u: 729
v: 729
w: 729
x: 729
y: 729
z: 729
Number of elements caught by default case: 729
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

test_int.cpp Output

⟨output from test_int.cpp 80⟩ ≡

```
Test:  test_int.cpp

Test Using integers as input.
*********************************

Results:

number of elements in each range, with the max and min element in the ranges
```

```
(0,100000): 99999 99999 1
(100000,200000): 99999 199999 100001
(200000,300000): 99999 299999 200001
(300000,400000): 99999 399999 300001
(400000,500000): 99999 499999 400001
(500000,600000): 99999 599999 500001
(600000,700000): 99999 699999 600001
(700000,800000): 99999 799999 700001
(800000,900000): 99999 899999 800001
(900000,1000000): 99999 999999 900001
Number elements caught by default case: 10
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

`test_double.cpp` Output

⟨output from test_double.cpp 81⟩ ≡

```
Test:  test_double.cpp

Test Using type double as input.
**********************************

Results:

number of elements in each range, with the max and min element in the ranges
(0,0.1)  : 99999 0.099999 1e-06
(0.1,0.2): 99999 0.199999 0.100001
(0.2,0.3): 99999 0.299999 0.200001
(0.3,0.4): 99999 0.399999 0.300001
(0.4,0.5): 99999 0.499999 0.400001
(0.5,0.6): 99999 0.599999 0.500001
(0.6,0.7): 99999 0.699999 0.600001
(0.7,0.8): 99999 0.799999 0.700001
(0.8,0.9): 99999 0.899999 0.800001
(0.9,1)  : 99999 0.999999 0.900001
Number of elements caught by default case: 10
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

4.4.5 Testing of Priorities

Test Objective

This set of tests validates that the GDFA priority levels function properly.

Description

This test validates the use priorities in the GDFA library. GDFA allows the use of 4 priority levels. The priority levels determine the order in which an element is passed to the `router`'s `destination` objects.

This test set runs configurations of 2, 3, and 4 priorities.

For a given test with N priority levels, a `router` is created with a set of associated `destination`s. The `destination`s are assigned `N` different priority levels. For example, the 4-priority test is set up by creating a `router` with multiple `destination`s where at least one `destination` is assigned an `urgent_priority`, one is assigned an `high_priority`, one is assigned to `normal_priority` and the fourth assigned to `low_priority`. There are 4 different types of `destination` classes used in the 4-priority test. These `destination` types are `contains_string`, `contains_letter`, `length_counter`, and `word_sorter`. Multiple instances of each `destination` type are used. For example, there are 2 `length_counter` `destination`s and 26 `word_sorter` `destination`s, 1 `contains_letter` `destination`, and 1 `contains_string` `destination` used in the 4-priority example. This setup is useful in testing the priorities because the `destination` types are not mutually exclusive. In other words, one element may meet the requirements of more than one `destination`. Consequently, priority matters, because the order in which an element is passed to a `destination` affects the results of the distribution.

The test is performed on this `router` by creating the needed input and storing it a storage `std::vector` for use with this test. The `router.distribute()` member function is then run on the data, and the results are printed and stored for validation.

Validation of the test results is done by a validate function which checks the contents of each resulting container. The input data generated for this test is constructed such that we know the expected results. The requirements of the data for the various destinations overlap. The construction of the data intentionally creates a scenario where the expected results require proper ordering of the destinations based on priority.

The input data for the 4-priority test is constructed as follows:

81 strings containing the sub-string "ea" are generated Of those, 27*2 with 'e' as first letter and 27 with "ea" as the last 2 letters for a total of 81.

4267 strings containing the string "a" are generated. Of those, 27*27*3 begin with a and 26*27*2 have a as the second letter.

There are 26 other characters ('a' through 'z' and '{'), each of which begin strings, and among those each has $27 * 2$ strings that have 'a' as the second character) and 26^2 have 'a' as the third letter for a total of 4267 (there are 26^2 strings with the third character being 'a' that do not have an 'a' as the first or second character)

there are 19683 strings of length 1 (27^3)
there are 19683 strings of length 2 (27^3)
there are 2187 strings beginning with each letter ('a'-'z') ($27^2 * 3$)
there are 2187 strings beginning with a nonletter ($27^2 * 3$)
there are 59049 strings total ($27^3 * 3$)

Therefore, the expected results (in the nofallthrough test) are:

81 strings with "ea"
4186 strings with "a" ($4267 - 81$)
18954 strings of length 1 ($19683 - 27^2$ (one letter strings of "a"))
18252 strings of length 2
($19683 - 27^2$ (2-letter strings beginning with "a") - $26 * 27$ (2-letter strings not beginning with "a" but ending with "a"))
676 strings beginning with anything but "a"

The validation method tests the actual results with these expected results.

The same data set is used for the 2- and 3-priority tests. The expected results for these tests are different. Therefore, a separate validate method is constructed to test those results.

This test is also run in the "no fall through" configuration for the 2-, 3-, and 4-priority configurations. These configurations are run using the same input data set. The expected results for these tests are different. Therefore, a separate validate method is constructed to test those results.

Each of the above tests is run for every combination of priority levels that comport with the above ordering. For example, if there are 2 priority levels, the levels could be:

Destination A	Destination B
urgent	normal
urgent	low
high	normal
high	low
normal	low

These 2-priority configurations are logically the same. Therefore, the test validates that all configurations produce the same desired output.

Results

The following are outputs from the respective test programmes; each indicates that the GDFA Library passes the test in question.

`priority2_fallthrough.cpp` Output The code implementing this test can be found in Appendix C.4.1.

⟨output from priority2_fallthrough.cpp 83⟩ ≡

```
Test:  priority2_fallthrough.cpp

Test router with 2 priorities (allowing fall through)
**********************************************************

Note: Fall through means that a destination
passes the element on to the next destination.

*********************************************

Results:

results of testing urgent priority and high priority
words of length 1: 19683
words of length 2: 19683
counts for all other words, based on starting character (case insensitive)
a: 2187
b: 2187
c: 2187
d: 2187
e: 2187
f: 2187
g: 2187
h: 2187
i: 2187
j: 2187
k: 2187
l: 2187
m: 2187
n: 2187
o: 2187
p: 2187
q: 2187
r: 2187
s: 2187
t: 2187
u: 2187
v: 2187
w: 2187
x: 2187
y: 2187
z: 2187
Default: 59049
Passed test for urgent and high priorities
Testing Urgent and normal priority
Passed test for urgent and normal priority
Testing Urgent and low priority
Passed test for urgent and low priority
Testing high and normal priority
Passed test for high and normal priority
```

```
Testing high and low priority
Passed test for high and low priority
Testing normal and low priority
Passed test for normal and low priority

*************     END OF RESULTS     *************
```

Not used.

`priority2_nofallthrough.cpp` **Output** The code implementing this test can be found in Appendix C.4.1.

⟨output from priority2_nofallthrough.cpp 85⟩ ≡

```
Test:  priority2_nofallthrough.cpp

Test router with 2 priorities (No fall through)
***********************************************

(Note: Fall through means that a destination
passes the element on to the next destination.

***********************************************

Results:

results of testing urgent priority and high priority
words of length 1: 19683
words of length 2: 19683
counts for all other words, based on starting character (case insensitive)
a: 729
b: 729
c: 729
d: 729
e: 729
f: 729
g: 729
h: 729
i: 729
j: 729
k: 729
l: 729
m: 729
n: 729
o: 729
p: 729
q: 729
r: 729
s: 729
t: 729
u: 729
```

```
v: 729
w: 729
x: 729
y: 729
z: 729
Default: 729
Passed test for urgent and high priorities
Testing Urgent and normal priority
Passed test for urgent and normal priority
Testing Urgent and low priority
Passed test for urgent and low priority
Testing high and normal priority
Passed test for high and normal priority
Testing high and low priority
Passed test for high and low priority
Testing normal and low priority
Passed test for normal and low priority

*************     END OF RESULTS     *************
```

Not used.

`priority3_fallthrough.cpp` **Output** The code implementing this test can be found in Appendix C.4.2.

⟨output from priority3_fallthrough.cpp 86⟩ ≡

```
Test:  priority3_fallthrough.cpp

Test router with 3 priorities (allowing fall through)
***********************************************************

Note: Fall through means that a destination
passes the element on to the next destination.

*********************************************

Results:

Results of testing urgent, high and normal priorities
words containing the letter (lowercase) a: 4267
words of length 1: 19683
words of length 2: 19683
counts for all other words, based on starting character, case insensitive
a: 2187
b: 2187
c: 2187
d: 2187
e: 2187
f: 2187
g: 2187
```

```
h: 2187
i: 2187
j: 2187
k: 2187
l: 2187
m: 2187
n: 2187
o: 2187
p: 2187
q: 2187
r: 2187
s: 2187
t: 2187
u: 2187
v: 2187
w: 2187
x: 2187
y: 2187
z: 2187
Default: 59049
Passed test for urgent, high and normal priorities
Testing urgent, high and low priority
Passed test for urgent, high and low priority
Testing urgent, normal and low priority
Passed test for urgent, normal and low priority
Testing high, normal and low priority
Passed test for high, normal and low priority

*************     END OF RESULTS     *************
```

Not used.

`priority3_nofallthrough.cpp` **Output** The code implementing this test can be found in Appendix C.4.2.

⟨output from priority3_nofallthrough.cpp 87⟩ ≡

```
Test:  priority3_nofallthrough.cpp

Test router with 3 priorities (No fall through)
***********************************************

(Note: Fall through means that a destination
passes the element on to the next destination.

***********************************************

Results:

Results of testing urgent, high and normal priorities
words containing the letter (lowercase) a: 4267
```

```
words of length 1: 18954
words of length 2: 18252
counts for all other words, based on starting character, case insensitive
a: 0
b: 676
c: 676
d: 676
e: 676
f: 676
g: 676
h: 676
i: 676
j: 676
k: 676
l: 676
m: 676
n: 676
o: 676
p: 676
q: 676
r: 676
s: 676
t: 676
u: 676
v: 676
w: 676
x: 676
y: 676
z: 676
Default: 676
Passed test for urgent, high and normal priorities
Testing urgent, high and low priority
Passed test for urgent high and low priority
Testing urgent, normal and low priority
Passed test for urgent, normal and low priority
Testing high, normal and low priority
Passed test for high, normal and low priority

*************     END OF RESULTS     *************
```

Not used.

`priority4_fallthrough.cpp` **Output** The code implementing this test can be found in Appendix C.4.3.

⟨output from priority4_fallthrough.cpp 88⟩ ≡

```
Test:  priority4_fallthrough.cpp

Test router with 4 priorities (allowing fall through)
************************************************************
```

```
Note: Fall through means that a destination
passes the element on to the next destination.

*********************************************

Results:

Results of testing urgent, high, normal and low priorities
Words containing the string "ea": 81
words containing the letter (lowercase) a: 4267
words of length 1: 19683
words of length 2: 19683
counts for all other words, based on starting character, case insensitive
a: 2187
b: 2187
c: 2187
d: 2187
e: 2187
f: 2187
g: 2187
h: 2187
i: 2187
j: 2187
k: 2187
l: 2187
m: 2187
n: 2187
o: 2187
p: 2187
q: 2187
r: 2187
s: 2187
t: 2187
u: 2187
v: 2187
w: 2187
x: 2187
y: 2187
z: 2187
Default: 59049
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

`priority4_nofallthrough.cpp` **Output** The code implementing this test can be found in Appendix C.4.3.

⟨output from priority4_nofallthrough.cpp 89⟩ ≡

```
Test:  priority4_nofallthrough.cpp

Test router with 4 priorities (No fall through)
***********************************************

(Note: Fall through means that a destination
passes the element on to the next destination.

***********************************************

Results:

Results of testing urgent, high, normal and low priorities
Words containing the string "ea": 81
words containing the letter (lowercase) a: 4186
words of length 1: 18954
words of length 2: 18252
counts for all other words, based on starting character, case insensitive
a: 0
b: 676
c: 676
d: 676
e: 676
f: 676
g: 676
h: 676
i: 676
j: 676
k: 676
l: 676
m: 676
n: 676
o: 676
p: 676
q: 676
r: 676
s: 676
t: 676
u: 676
v: 676
w: 676
x: 676
y: 676
z: 676
Default: 676
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

4.4.6 Insert Iterator Tests

Test Objective

This set of tests is intended to validate that the GDFA work properly with all of the various front- and back-insertion iterators as the `OutputIterator` of a `destination` class.

Description

This test validates the use of the types generated by the following types, as the `OutputIterator` of a the `destination` objects:

- `std::front_insert_iterator(std::deque)`
- `std::front_insert_iterator(std::list)`
- `std::back_insert_iterator(std::deque)`
- `std::back_insert_iterator(std::list)`
- `std::back_insert_iterator(std::vector)`

The testing procedure works as follows:

First we use the `word_sorter` class to construct a `router` with 27 `destination`s that output to an `std::vector` via an iterator.

Second, we perform a baseline test using `std::back_insert_iterator(std::vector)` to define the input range. The input element type is `std::string`.

Third, we perform the above sequence using the types to be tested as the location that the `destination` class outputs to.

Then we validate each test by comparing the results to the results of the baseline test. Due to the fact that all of the containers are sequential we can validate the results of the tests by explicitly comparing the results to determine if they are identical. If the results are identical to the baseline test, the test passes. Otherwise, the test is failed.

The `validate` method for the front-insertion sequences is slightly different. Since the baseline is a back-insertion sequence, the results (the distributed sequence elements produced by the call to `router.distribute()`) of the tests using the front-insertion sequences will differ because we are pushing onto the front of the container instead of the back. Consequently, the resulting sequences will be in the reverse order of the `std::back_insert_iterator(std::vector)`. The code implementing this test may be found in Appendix C.5.

Results

The following is the output from the test programme; it indicates that the GDFA Library passes these tests.

⟨output of testinserters.cpp 91⟩ ≡

```
testinserters.cpp

Test of Reading from Standard Library Containers
using front or back inserters (where applicable).
************************************************

Results:

Results for filtering into vector using a back insert iterator.  Will be used for testing other i
Number of words beginning with each letter, case insensitive:
a: 3228
b: 1417
c: 1255
d: 894
e: 532
f: 1020
g: 661
h: 2394
i: 2020
j: 94
k: 200
l: 740
m: 1080
n: 639
o: 1653
p: 813
q: 64
r: 544
s: 2365
t: 4282
u: 396
v: 163
w: 2108
x: 1
y: 460
z: 2
Default: 823

Testing back insert iterator deque
Passed test for back insert iterator deque
Testing back insert iterator list
Passed test for back insert iterator list
Testing front insert iterator deque
Passed test for front insert iterator deque
Testing front insert iterator list
Passed test for front insert iterator list

*************     END OF RESULTS     *************
```
Not used.

4.4.7 Reading from Standard Library Containers Tests

Test Objective

This set of tests is intended to validate that the GDFA work properly with all of the Standard Library container classes via their associated input iterators.

Description of Tests

In this set of tests, the following Standard Library containers are used to hold the range of elements operated on by the `router` object:

- `std::list`
- `std::deque`
- `std::multiset`
- `std::map`

The tests in Section C.4 use an `std::vector` as the input container. Therefore, it is not necessary to re-test the use of that container.

All of these tests use the `word_sorter` derived destination class to sort strings in the input range based on its first character (insensitive to case).

To administer these tests we first read all the input into an `std::vector` for use later. This is used to store the unadulterated data set to be used (insert into) to test the `std::list`, `std::deque`, `std::multiset` and `std::map`. We then establish a baseline test by distributing the data using our established `router` object. We assume, based on previous tests, that the GDFA work correctly with an `std::vector`. We then test `list`, `deque`, `std::multiset`, and `std::map` by validating that the results are match the results obtained in our baseline test.

An `std::multiset` (but not an `std::set`) is used because, for any given set of input strings, a given string can occur multiple times. Therefore, an `std::multiset` ensures that all of the recurrent strings are accounted for. An `std::set` cannot be used with the GDFA on arbitrary input.

An `std::map` (rather than a `std::multimap`) can be used with GDFA if constructed properly. The type `std::map<std::string,int>` is used. The `std::map` stores `std::pairs` of `<std::string,int>`. The `std::string` itself is used as the key. The second member of the `std::pair` keeps track of the number of instances of each unique string.

The results of the test run using `std::multiset` and `std::map` "matches" the baseline test. However, due to the associative nature of these containers the results will not be identical to the results of the baseline test. However the number of words beginning with each letter will be the same as in the baseline test. Therefore, the `validate` method used for these containers will accept if the number of elements in each resulting container matches the number of elements in the respective containers obtained from the baseline test. A separate `validate` method was created to test `std::multiset` and `std::map`.

The `std::multiset` `validate` method works with individual elements and the `std::map` `validate` method works with `std::pairs` where the second member of the `std::pair` is the number of instances of the unique `std::string` stored in the `std::map`.

The validation the `std::list` and `std::deque` tests is performed using the `validate` methods explained in Section 4.4.1.

The code for these tests may be found in Appendix C.6.

Results

The following is the output from the test programme; it indicates that the GDFA Library passes these tests.

⟨output of testreadcont.cpp 94⟩ ≡

```
Test:   testreadcont.cpp

Test Reading from Standard Library Containers
*********************************************

Results:

Results for reading from a vector, used for comparison testing
Number of words beginning with each letter, case insensitive
a: 3228
b: 1417
c: 1255
d: 894
e: 532
f: 1020
g: 661
h: 2394
i: 2020
j: 94
k: 200
l: 740
m: 1080
n: 639
o: 1653
p: 813
q: 64
r: 544
s: 2365
t: 4282
u: 396
v: 163
w: 2108
x: 1
y: 460
z: 2
```

```
default: 823

Testing reading from a deque
Passed test for reading from a deque
Testing reading from a list
Passed test for reading from a list
Testing reading from a multiset
Passed test for reading from a multiset
Testing reading from a map
Passed test for reading from a map

*************     END OF RESULTS     *************
```

Not used.

4.4.8 Test Array as Input

Test Objective

This set of tests is intended to validate that the GDFA work properly with an array as the input container and a pointer as the input iterator.

Description

This test uses the `word_sorter` derived `destination` class to sort `std::strings` in the input range based on their first characters (insensitive to case).

This test validates the use of array as the container to hold the input range. Two pointers, which are valid `InputIterators`, are used to specify the beginning and end of the data range contained in the array.

When using an array as input, the address of the first element must be passed as the iterator pointing to the first element of the input range and the address of one past the last element must be passed to point to one past the last element in the input range. This is accomplished by indexing the first element of the array and the memory location one past the last element in the array and passing these memory locations by reference to the `router.distribute()` member function.

The test is administered as follows: First, we read the input into a `std::vector<std::string>` for storage. Second, we copy the input to an array for testing. Third, we distribute the data from both the array and the vector using the constructed router object. Forth, we validate the results of the array based distribution by comparing them to the results of the vector based distribution.

The array validation is performed using the `validate` methods explained in Section 4.4.1.

The code for this test may be found in C.7.

Results

The following is the output from the test programme; it indicates that the GDFA Library passes these tests.

⟨output from testarray.cpp 96⟩ ≡

```
Test:  testarray.cpp

Test Using an Array as input.
**********************************

Results:

a: 3228
b: 1417
c: 1255
d: 894
e: 532
f: 1020
g: 661
h: 2394
i: 2020
j: 94
k: 200
l: 740
m: 1080
n: 639
o: 1653
p: 813
q: 64
r: 544
s: 2365
t: 4282
u: 396
v: 163
w: 2108
x: 1
y: 460
z: 2
default: 823

Testing using an array as input
Passed array input test

*************     END OF RESULTS     *************
```

Not used.

4.4.9 Tests Reading from and Writing to Files

Test Objective

This set of tests validates that the GDFA is able to work properly while reading and writing to files via `std::iostream`s in the `destination` objects.

Description

This test validates the use of `std::iostream`s in the `destination` objects. This test complements a previous test in that it also writes directly to files.

In the test we use an `std::ifstream` to initialise the `InputIterator` given to `router.distribute()` and an `std::ofstream` to initialise the `OutputIterator` used in the `destination` classes. The functionality is the same as in the previous examples using word sorter except for the source and destination of the data. The results of the test are stored in files labeled "output_alpha" where "alpha" takes a value $[a - z]$ or "default".

The element type in this test is `std::string`.

The validation test is by necessity a seperate programme from the actual test, because the test outputs the results to files, rather than a storage container that can be easily checked. The code for this test may be found in Appendix C.8.

Thus the validation test works as follows: it reads each file generated by the test, and checks to make sure that the letter that starts each string in the file is the appropriate letter for that file. The default case is validated by making sure that no character in the ranges $[A - Z]$ or $[a - z]$ start a string in the default case file.

Results

The following is the output from the test programme; it indicates that the GDFA Library passes these tests.

⟨output from testrwfiles.cpp 97⟩ ≡

```
output from testrwfiles.cpp

Test:  testrwfiles.cpp

Test of reading and writing from and to files with ifstream and ofstream.
**********************************************************


*************     END OF TEST     *************




Test:  validaterwfiles.cpp

Validating the testrwfiles test.
**********************************************************

Results:

Passed rwfiles test

*************     END OF RESULTS     *************
```

Not used.

4.4.10 Testing `std::istream_iterators` with Built-In Types

Test Set Objective

The purpose of this set of tests is to validate that GDFA works properly if an `std::istream_iterator<element_type>` is used to define the input range operated on by the `router.distribute()` member function. Essentially, it tests that GDFA works using file streams via input file stream iterators using 4 element types (`char`, `std::string`, `int`, `double`).

Description

This set of tests reads from standard input, distributes the data and finally performs a validation test. There are 4 tests in this set: `int`, `double`, `char`, `std::string`.

Following are descriptions for each of the 4 tests.

Testing `std::istream_iterator<int>` This test ensures that using an `std::istream_iterator` type as the input to `router.distribute()` will result in the correct operation of the GDFA library.

This test takes as input a series of `int`s in the range $[0, 1000000]$ from standard input. The `router` class will then sort the input into 10 ranges:

- $(0, 100000)$
- $(100000, 200000)$
- $(200000, 300000)$
- $(300000, 400000)$
- $(400000, 500000)$
- $(500000, 600000)$
- $(600000, 700000)$
- $(700000, 800000)$
- $(800000, 900000)$
- $(900000, 1000000)$

A default case handles all other values.

A validation test is performed that checks the maximum and minimum element in each filtered container to ensure that all elements in the container are correctly distributed.

To generate input, a sample programme is included (generateints.cpp; see Appendix C.9.1) that takes as a command-line argument the number of integers to generate and prints to the command-line that many integers in the range $[0, 1000000)$.

The code for this test may be found in Appendix C.9.2.

Testing `std::istream_iterator<double>` This test is to ensure that using an `std::istream_iterator<double>` will result in the correct operation of the GDFA library.

This test takes as input a series of `double`s in the range $[0, 1]$ from standard input. The `router` class will then sort the input into 10 ranges:

- $(0.0, 0.1)$
- $(0.1, 0.2)$
- $(0.2, 0.3)$
- $(0.3, 0.4)$
- $(0.4, 0.5)$
- $(0.5, 0.6)$
- $(0.6, 0.7)$
- $(0.7, 0.8)$
- $(0.8, 0.9)$
- $(0.9, 1.0)$

A default case handles all other values.

A validation test is performed that checks the maximum and minimum element in each filtered container to ensure that all elements in the container are correctly filtered.

To generate input, a sample programme is included (generatedoubles.cpp; see Appendix C.9.1) that takes as a command-line argument the number of integers to generate and prints to the command-line that many doubles in the range $[0, 1]$.

The code for this test may be found in Appendix C.9.2.

Testing `std::istream_iterator<char>` This test is to ensure that using an `std::istream_iterator<char>` will result in the correct operation of the GDFA library.

This test takes as input text from standard input and attempts to sort the input according to the value of each character that is read. The input is filtered into 27 containers, one for each letter of the alphabet (uppercase and lowercase are treated as being the same) and a default case to catch all other characters.

Note that the default configuration for an `std::istream_iterator` is to ignore whitespace (space, tab and new line characters) meaning that the number of characters output by the test may not be the same as the number of characters in the input.

The test outputs the number of characters of each type that it sees. For a validation test it checks each container to ensure that only the correct letter is in the appropriate container. For the default case the validation test is to make sure that there are no letters in the default container, where a letter is $[A - Z]$ or $[a - z]$.

The code for this test may be found in Appendix C.9.2.

Testing `std::istream_iterator<std::string>` This test is to ensure that using an `std::istream_iterator<std::string>` will result in the correct operation of the GDFA library.

This test takes as input text from standard input and attempts to sort the input according to the value of the first letter of the string that is read. The input is filtered into 27 containers, one for each letter of the alphabet (uppercase and lowercase are treated as being the same) and a default case to catch all other characters.

The test outputs the number of strings that begin with each letter that it sees. For a validation test it checks each container to ensure that only strings beginning with the appropriate letter are in the appropriate container. For the default case the validation test is to make sure that there are no `std::strings` beginning with a letter in the default container, where a letter is $[A - Z]$ or $[a - z]$.

The code for this test may be found in Appendix C.9.2.

Results

The following results all confirm that the GDFA Library functions properly when used with `std::istream_iterators`.

Test with `std::istream_iterator<char>`

⟨output from test_char_istream.cpp 100⟩ ≡

```
Test:  test_char_istream.cpp

Test Uses istream iterator and type char as input.
**********************************

Results:

a: 9695
b: 2055
c: 3308
d: 5903
e: 15657
```

```
f: 2574
g: 3096
h: 8543
i: 8797
j: 144
k: 1052
l: 4796
m: 3020
n: 8340
o: 10218
p: 2294
q: 103
r: 7522
s: 8233
t: 11564
u: 3568
v: 1086
w: 3157
x: 199
y: 2486
z: 85
Number of elements caught by default case: 7903
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

Test with `std::istream_iterator<std::string>`

⟨output from test_string_istream.cpp 101⟩ ≡

```
Test:  test_string_istream.cpp

Test Uses istream iterator and type string as input.
**********************************

Results:

a: 3228
b: 1417
c: 1255
d: 894
e: 532
f: 1020
g: 661
h: 2394
i: 2020
j: 94
k: 200
l: 740
m: 1080
```

```
n: 639
o: 1653
p: 813
q: 64
r: 544
s: 2365
t: 4282
u: 396
v: 163
w: 2108
x: 1
y: 460
z: 2
Number of elements caught by default case: 823
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

Test with `std::istream_iterator<int>`

⟨output from test_int_istream.cpp 102a⟩ ≡

```
Test:  test_int_istream.cpp

Test Uses istream iterator and type int as input.
***********************************

Results:

number of elements in each range, with the max and min element in the ranges
(0,100000): 9853 99993 13
(100000,200000): 10013 199985 100022
(200000,300000): 10087 299968 200013
(300000,400000): 10145 399999 300004
(400000,500000): 9944 499960 400006
(500000,600000): 9984 599989 500034
(600000,700000): 10025 699993 600023
(700000,800000): 9863 799997 700001
(800000,900000): 10040 899997 800013
(900000,1000000): 10045 999964 900008
Number elements caught by default case: 1
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

Test with `std::istream_iterator<double>`

⟨output from test_double_istream.cpp 102b⟩ ≡

```
Test:  test_double_istream.cpp

Test Uses istream iterator and type double as input.
**********************************

Results:

number of elements in each range, with the max and min element in the ranges
(0,0.1): 10084 0.0999923 2.3006e-06
(0.1,0.2): 9967 0.199997 0.100015
(0.2,0.3): 9898 0.299995 0.200009
(0.3,0.4): 10059 0.399989 0.30001
(0.4,0.5): 10120 0.499999 0.400008
(0.5,0.6): 9930 0.599995 0.500007
(0.6,0.7): 10002 0.699988 0.600007
(0.7,0.8): 10039 0.799995 0.700002
(0.8,0.9): 9863 0.899999 0.800033
(0.9,1): 10037 0.999996 0.900004
Number of elements caught by default case: 1
Passed Validation Test

*************     END OF RESULTS     *************
```

Not used.

4.4.11 Testing Unexpected Conditions

This section explores several boundary conditions for the `router` class.

First, we run the GDFA without adding any `destination`s to the `router` object. The proper functionality would be for the GDFA to nothing.

The code for this test may be found in Appendix C.9.3.

Second, we run the GDFA with an input range that is empty. The proper action is to do nothing, which results in unaltered containers (the call to `router.distribute()` results in nothing being written to the locations pointed to by its `destination` objects' `OutputIterator`s.

The code for this test may be found in Appendix C.9.3.

Test Objective

This set of tests attempts to determine how the GDFA would behave under certain atypical conditions. These are conditions which are incongruous with the logical operation of the library. However, it is desirable for the `router` class to be robust and as such operate satisfactorily and predictably under these conditions.

Results

These results indicate that the `router` class is able to properly handle the boundary conditions specified.

Test of `router` with no `destinations`

⟨output of nodestinations.cpp 104a⟩ ≡

```
Test:  nodestinations.cpp

Test router without adding and destinations to it.
**************************************************

Results:


*************     END OF RESULTS     *************
```

Not used.

Test of `router` with no Input

⟨output of noinput.cpp 104b⟩ ≡

```
Test:  noinput.cpp

Test behavior of router on an empty input range.
******************************************

Results:

a: 0
b: 0
c: 0
d: 0
e: 0
f: 0
g: 0
h: 0
i: 0
j: 0
k: 0
l: 0
m: 0
n: 0
o: 0
p: 0
q: 0
r: 0
s: 0
t: 0
u: 0
v: 0
w: 0
x: 0
y: 0
z: 0
```

```
Default: 0


*************      END OF RESULTS      *************
```
Not used.

Chapter 5

Conclusion

5.1 Summary

This report presents the GDFA Library. The goal in so doing is to demonstrate and explain the usefulness of the GDFA Library through a broad conceptual overview and a practical set of "learn by example" applications. Technical information and complete set of descriptions of the GDFA Library components and features is also provided.

The testing of the GDFA Library components demonstrates their correctness and efficiency. The correctness tests demonstrate that the components work properly with an exhaustive set of relevant Standard Library types and built-in types. Additionally, a variety of priority configurations as well as unexpected situations are tested. The components perform correctly and predictably in all tests.

The efficiency tests include timed tests and operation counting tests. These methods are used in order to establish a sound foundation for our conclusions. Both tests clearly show that the algorithms used in the GDFA Library are linear with respect to N (the input range size) and M (the number of `destination`s) and is consequently an `O(MN)` algorithm. These results are exactly what we expect. In practice, the GDFA Library is likely to be used with a large input size (N) and a modest number of destinations (M). Consequently, the linear running time with respect to N is of primary importance to users of the library.

5.2 Future of the GDFA Library

While the foundation of the GDFA Library is the simple `distribute` function, it has already been extended into the more powerful set of components presented in this document. These components make use of sub-type polymorphism (inheritance and virtual functions), the functor concept, iterators, and extensive

parametric polymorphism (C++ templates).[1]

Many future opportunities exist for the library: in terms of network programming, generalised multi-level distribution, and thread safety.

It is clear from the examples presented in the tutorial and testing sections that the GDFA components would work well with a networked application. Therefore, a fully implemented example of this sort would be a nice extension of this project and would complement our existing examples.

Another interesting application is an integrated multi-level distribution and filtering paradigm. This concept is used in the criminal profiling example in the tutorial. A larger application of these notions would be a very interesting extension of this project.

Finally, a thread-safe version of GDFA, in the sense that it would allow the user to add and delete `destination` objects while the application is running, would allow the combination of the above two ideas in a single comprehensive application. Specifically, this ability would facilitate the creation of a multi-level filtering paradigm running in tandem on multiple processors in multiple locations, each running a particular layer of the filter. The application would be continually running; as such the ability to maintain (update) the filtering configuration would be necessary. Currently, this setup is possible if the `destination` configurations are not changed. However, this could be made easier if the startup of the algorithm were constructed to be more flexible. Basically, the `router` objects must be willing to wait for a subsequent iteration on their input streams rather than terminate. Without belaboring additional implementation issues, it is fair to say that this would constitute a substantial increase in functionality.

5.3 Final Thoughts

We hope that the GDFA Library will be a powerful new set of components with many useful applications over a wide range of software development and programming problems. We strongly encourage the reader to use, extend, and modify the GDFA Library to tackle new problems in distribution and filtering.

[1] The capabilities used to create this feature set gracefully permute into countless empowering abstractions. The result is a supple and powerful set of components with virtually unlimited applications.

Appendix A

Code for Timed Tests

A.1 Test 1

A.1.1 Test 1a

"timed_test1a.cpp" 108 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
```

```
{
  private:
    OutputIterator out;
    int n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element < n)
      {
        *(out++) = element;
        return false;//continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      *(out++) = n;
      return true;
    }
};


void test() {
  cout << "Test 1a\nTesting N\nTiming distribute on integers.\n";
  cout << "Distribute to all destinations." << endl;
 srand(time(0));

  // Initial Setup
  router<int> *r;
  r = new router<int>;
  vector<int> v, out1, out2, out3, defaultItems;
  vector<int> out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<std::vector<int> > BItype;
    delete r;
```

```
  r = new router<int>;

out1.clear();
out2.clear();
out3.clear();
defaultItems.clear();

less_than_n<BItype> test1(back_inserter(out1), 1048576);
less_than_n<BItype> test2(back_inserter(out2), 1048576);
less_than_n<BItype> test3(back_inserter(out3), 1048576);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->add_destination(test1);
r->add_destination(test2);
r->add_destination(test3);
r->set_default(defaultTest);


unsigned long N, N1, N2;
unsigned int reps;

N1 = 1;
N2 = 1048576;
reps = 10;

timer tim;

for (N = N1; N <= N2; N *= 2) {



    v.clear();
  v.reserve(N);
  for (int i = 0; i < N; ++i) {
  v.push_back(i);
  }

  // Compute the baseline time for N
  tim.start_baseline(reps);
  do {
  ////////////////////////////
    // Baseline Operations Here

out1.clear();
out2.clear();
out3.clear();
defaultItems.clear();

  // End of Baseline Operations
  ////////////////////////////
```

```
    } while (tim.check());
    tim.report(false);

    tim.start(reps, N);
    do {

    ////////////////////////////
      // Baseline Operations Here

  out1.clear();
  out2.clear();
     out3.clear();
  defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
  r->distribute(v.begin(), v.end());
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test();
  return 0;
}
```

A.1.2 Test 1b

"timed_test1b.cpp" 111 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"
```

```
using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
{
  private:
    OutputIterator out;
    int n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element > n)
      {
        *(out++) = element;
        return false;//continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      *(out++) = n;
      return true;
    }
};
```

```
void test() {
  cout << "Test 1b\nTesting N\nTiming distribute on integers.\n";
  cout << "Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  // Initial Setup
  router<int> *r;
  r = new router<int>;
  vector<int> v, out1, out2, out3, defaultItems;
  vector<int> out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<std::vector<int> > BItype;
    delete r;
    r = new router<int>;

  out1.clear();
  out2.clear();
  out3.clear();
  defaultItems.clear();

  less_than_n<BItype> test1(back_inserter(out1), 1048576);
  less_than_n<BItype> test2(back_inserter(out2), 1048576);
  less_than_n<BItype> test3(back_inserter(out3), 1048576);
  my_default<BItype> defaultTest(back_inserter(defaultItems));

  r->add_destination(test1);
  r->add_destination(test2);
  r->add_destination(test3);
  r->set_default(defaultTest);


  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
  reps = 10;

  timer tim;

  for (N = N1; N <= N2; N *= 2) {



      v.clear();
    v.reserve(N);
    for (int i = 0; i < N; ++i) {
    v.push_back(i);
    }
```

```
    // Compute the baseline time for N
    tim.start_baseline(reps);
    do {
    ////////////////////////////
      // Baseline Operations Here

  out1.clear();
  out2.clear();
  out3.clear();
  defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    } while (tim.check());
    tim.report(false);

    tim.start(reps, N);
    do {

    ////////////////////////////
      // Baseline Operations Here

  out1.clear();
  out2.clear();
     out3.clear();
  defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
  r->distribute(v.begin(), v.end());
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test();
  return 0;
}
```

A.1.3 Test 1c

"timed_test1c.cpp" 114 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
{
  private:
    OutputIterator out;
    int n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element < n)
      {
        //no output destination
        return false;//continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
```

```
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      //no output destination
      return true;
    }
};


void test() {
  cout << "Test 1c\nTesting N\nTiming distribute on integers.\n";
  cout<<"Time Decision Making only.  \nFactor out insertion.";
  cout<<endl;

  srand(time(0));

  // Initial Setup
  router<int> *r;
  r = new router<int>;
  vector<int> v, out1, out2, out3, defaultItems;
  vector<int> out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<std::vector<int> > BItype;
    delete r;
    r = new router<int>;

  out1.clear();
  out2.clear();
  out3.clear();
  defaultItems.clear();

  less_than_n<BItype> test1(back_inserter(out1), 1048576);
  less_than_n<BItype> test2(back_inserter(out2), 1048576);
  less_than_n<BItype> test3(back_inserter(out3), 1048576);
  my_default<BItype> defaultTest(back_inserter(defaultItems));

  r->add_destination(test1);
  r->add_destination(test2);
  r->add_destination(test3);
  r->set_default(defaultTest);


  unsigned long N, N1, N2;
  unsigned int reps;
```

```
N1 = 1;
N2 = 1048576;
reps = 10;

timer tim;

for (N = N1; N <= N2; N *= 2) {



    v.clear();
  v.reserve(N);
  for (int i = 0; i < N; ++i) {
  v.push_back(i);
  }

  // Compute the baseline time for N
  tim.start_baseline(reps);
  do {
  ////////////////////////////
  // Baseline Operations Here
  out1.clear();
  out2.clear();
  out3.clear();
  defaultItems.clear();

  // End of Baseline Operations
  ////////////////////////////
  } while (tim.check());
  tim.report(false);

  tim.start(reps, N);
  do {

  ////////////////////////////
  // Baseline Operations Here
  out1.clear();
  out2.clear();
     out3.clear();
  defaultItems.clear();
  // End of Baseline Operations
  ////////////////////////////
  // Main Timed Operation
  r->distribute(v.begin(), v.end());
  ////////////////////////////////////
  }
  while (tim.check());
  tim.report(false);
}
```

```
}

int main()
{
  test();
  return 0;
}
```

A.2 Test 2

A.2.1 Test 2a

"timed_test2a.cpp" 118 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
{
  private:
    OutputIterator out;
    int n;

  public:
```

```
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element < n)
      {
        *(out++) = element;
        return false;// continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      *(out) = n;
      return true;
    }
};


void test2() {
  cout << "Test 2a\nTesting M\nTiming distribute on integers.\n";
  cout << "Distribute to all destinations." << endl;
  srand(time(0));

  // Initial Setup
  router<int> *r;
  r = new router<int>;
  vector<int> v, out1, out2, defaultItems;
  vector<int> out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<std::vector<int> > BItype;
    delete r;
    r = new router<int>;

  out1.clear();
  out2.clear();
  defaultItems.clear();
```

```
less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
less_than_n<BItype> test20(back_inserter(out2), 20);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->set_default(defaultTest);
    v.clear();
  v.reserve(10);
  for (int i = 0; i < 10; ++i) {
  v.push_back(i);
  }


unsigned long N, N1, N2;
unsigned int reps;

N1 = 1;
N2 = 1048576;
reps = 10;

timer tim;

for (N = N1; N <= N2; N *= 2) {

  for (int i = 0; i < (N/2); ++i) {
  r->add_destination(test1048576);
  }


  // Compute the baseline time for N
  tim.start_baseline(reps);
  do {
  ////////////////////////////
    // Baseline Operations Here
  out1.clear();
  defaultItems.clear();

  // End of Baseline Operations
  ////////////////////////////
  } while (tim.check());
  tim.report(false);

  tim.start(reps, N);
  do {

  ////////////////////////////
    // Baseline Operations Here
  out1.clear();
  defaultItems.clear();

  // End of Baseline Operations
```

```
    ////////////////////////////
    // Main Timed Operation
  r->distribute(v.begin(), v.end());
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test2();
  return 0;
}
```

A.2.2 Test 2b

"timed_test2b.cpp" 121 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
{
```

```
  private:
    OutputIterator out;
    int n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element > n)
      {
        *(out++) = element;
        return false;// continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      *(out) = n;
      return true;
    }
};


void test2() {
  cout << "Test 2b\nTesting M\nTiming distribute on integers.\n";
  cout << "Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  // Initial Setup
  router<int> *r;
  r = new router<int>;
  vector<int> v, out1, out2, defaultItems;
  vector<int> out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<std::vector<int> > BItype;
    delete r;
    r = new router<int>;
```

```
  out1.clear();
  out2.clear();
  defaultItems.clear();

  less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
  less_than_n<BItype> test20(back_inserter(out2), 20);
  my_default<BItype> defaultTest(back_inserter(defaultItems));

  r->set_default(defaultTest);
      v.clear();
    v.reserve(10);
//    for (int i = 0; i < 100; ++i) {
    for (int i = 0; i < 10; ++i) {
    v.push_back(i);
    }


  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
  reps = 10;

  timer tim;

  for (N = N1; N <= N2; N *= 2) {

    for (int i = 0; i < (N/2); ++i) {
    r->add_destination(test1048576);
    }


    // Compute the baseline time for N
    tim.start_baseline(reps);
    do {
    ////////////////////////////
      // Baseline Operations Here
    out1.clear();
    defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    } while (tim.check());
    tim.report(false);

    tim.start(reps, N);
    do {
```

```
    ////////////////////////////
      // Baseline Operations Here
    out1.clear();
    defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
  r->distribute(v.begin(), v.end());
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test2();
  return 0;
}
```

A.2.3 Test 2c

"timed_test2c.cpp" 124 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;
```

```
using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<int>
{
  private:
    OutputIterator out;
    int n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const int& element)
    {
      if (element < n)
      {
        //no output destination
        return false;// continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<int>
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const int& n)
    {
      //no output destination
      return true;
    }
};


void test2() {
  cout << "Test 2c\nTesting M\nTiming distribute on integers.\n";
  cout<<"Time Decision Making only.  \nFactor out insertion.";
  cout<<endl;
  srand(time(0));

  // Initial Setup
  router<int> *r;
```

```
r = new router<int>;
vector<int> v, out1, out2, defaultItems;
vector<int> out1_saved, out2_saved, defaultItems_saved;

typedef std::back_insert_iterator<std::vector<int> > BItype;
  delete r;
  r = new router<int>;

out1.clear();
out2.clear();
defaultItems.clear();

less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
less_than_n<BItype> test20(back_inserter(out2), 20);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->set_default(defaultTest);
    v.clear();
  v.reserve(10);
  for (int i = 0; i < 10; ++i) {
  v.push_back(i);
  }


unsigned long N, N1, N2;
unsigned int reps;

N1 = 1;
N2 = 1048576;
reps = 10;

timer tim;

for (N = N1; N <= N2; N *= 2) {

  for (int i = 0; i < (N/2); ++i) {
  r->add_destination(test1048576);
  }


  // Compute the baseline time for N
  tim.start_baseline(reps);
  do {
  ////////////////////////////
    // Baseline Operations Here
  //out1.clear();
  //defaultItems.clear();

  // End of Baseline Operations
  ////////////////////////////
```

```
    } while (tim.check());
    tim.report(false);

    tim.start(reps, N);
    do {

    ////////////////////////////
      // Baseline Operations Here
    //out1.clear();
    //defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
  r->distribute(v.begin(), v.end());
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test2();
  return 0;
}
```

A.3 Test 3

"timed_test3.cpp" 127 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include "timer.h"
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "router.hpp"

using namespace std;
using std::cout;
```

```
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;


bool less_than_n(const int& element) {
      if (element < 1048576) {
        return true;
      }

      return false;
}




void test() {
  cout << "Test 3\nSimple Version of Distribute\nTesting N\n";
  cout << "Timing distribute on integers." << endl;
  srand(time(0));

  // Initial Setup
  vector<int> v, out1, defaultItems;

  typedef std::back_insert_iterator<std::vector<int> > BItype;

  out1.clear();
  defaultItems.clear();

  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
  reps = 10;

  timer tim;

  for (N = N1; N <= N2; N *= 2) {



      v.clear();
    v.reserve(N);
```

```
    for (int i = 0; i < N; ++i) {
    v.push_back(i);
    }

    // Compute the baseline time for N
    tim.start_baseline(reps);
    do {
    ////////////////////////////
    // Baseline Operations Here
    out1.clear();
    defaultItems.clear();
    // End of Baseline Operations
    ////////////////////////////
    } while (tim.check());
    tim.report(false);

    tim.start(reps, N);
    do {

    ////////////////////////////
    // Baseline Operations Here
    out1.clear();
    defaultItems.clear();
    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
    gdfa::distribute(v.begin(), v.end(), back_inserter(out1),
    back_inserter(defaultItems), less_than_n);
    ////////////////////////////////////
    }
    while (tim.check());
    tim.report(false);
  }

}

int main()
{
  test();
  return 0;
}
```

A.4 Auxiliary Code

The following code is taken from [3] and was used by the test code in Sections A.1.1-A.3.

"timer.h" 129 ≡

```
// Define a timer class for analyzing algorithm performance.
#include <iostream>
#include <iomanip>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

class timer {
public:
  timer();  // Default constructor
  // Start a series of r trials for problem size N:
  void start(unsigned int r, unsigned long N);
  // Start a series of r trials to determine baseline time:
  void start_baseline(unsigned int r);
  // Returns true if the trials have been completed, else false
  bool check();
  // Report the results of the trials on cout
  // with additional output if verbose is true:
  void report(bool verbose);
  // Returns the results for external use
  const map<unsigned int, double>& results() const;

private:
  unsigned int reps;  // Number of trials
  // For storing loop iterations of a trial
  vector<long> iterations;
  // For saving initial and final times of a trial
  time_t initial, final;
  // For counting loop iterations of a trial
  unsigned long count;
  // For saving the problem size (N) for current trials
  unsigned int problem_size;
  // For storing (problem size, time) pairs
  map<unsigned int, double> result_map;
  // true if this is a baseline computation, false otherwise
  bool baseline;
  // For recording the baseline time
  double baseline_time;
};

timer::timer() { baseline = false; }
void timer::start(unsigned int r, unsigned long N)
{
  reps = r;
  problem_size = N;
  count = 0;
  iterations.clear();
  iterations.reserve(reps);
  initial = time(0);
```

```
}
void timer::start_baseline(unsigned int r)
{
  baseline = true;
  start(r, 0);
}
bool timer::check()
{
  ++count;
  final = time(0);
  if (initial < final) {
    iterations.push_back(count);
    initial = final;
    count = 0;
  }
  return (iterations.size() < reps);
}
void timer::report(bool verbose)
{
  if (verbose) {
    for (unsigned int k = 0; k < iterations.size(); ++k) {
      cout << iterations[k] << " ";
      if ((k+1) % 10 == 0)
        cout << endl;
    }
    cout << endl;
  }
  sort(iterations.begin(), iterations.end());
  if (verbose) {
    cout << "Sorted counts:" << endl;
    for (unsigned int k = 0; k < iterations.size(); ++k) {
      cout << iterations[k] << " ";
      if ((k+1) % 10 == 0)
        cout << endl;
    }
    cout << endl;
  }

  int selected_count = iterations[reps/2];
  if (verbose)
    cout << "Selected count: " << selected_count << endl;

  if (baseline) {
    baseline_time = 1000.0/selected_count;
    cout << "Baseline time: " << baseline_time << endl;
    baseline = false;
  } else {
    double calculated_time, growth_factor;
    result_map[problem_size] = calculated_time =
      1000.0/selected_count - baseline_time;
```

```
    cout << setiosflags(ios::fixed) << setprecision(4)
         << setw(35) << problem_size << setw(12)
         << calculated_time << " ms ";
    if (result_map.find(problem_size/2) != result_map.end()) {
      growth_factor = calculated_time / result_map[problem_size/2];
      cout << setiosflags(ios::fixed) << setprecision(4)
           << setw(8) << growth_factor;
    }
    cout << endl;
  }
}
const map<unsigned int, double>& timer::results() const
{
  return result_map;
}
```

Appendix B

Code for Operation Counting Tests

B.1 Operation Counting Iterator Adapters

"counters.h" 133 ≡

```
#ifndef COUNTERS_H
#define COUNTERS_H

#include <iostream>
#include <iomanip>
#include <string>
#include <typeinfo>
#include <algorithm>
#include <functional>
#include <numeric>
#include <map>
#include <vector>
#include <stack>
#include <utility>

#include <boost/iterator_adaptors.hpp>

namespace counters {

#ifndef DEFAULT_COUNTER_TYPE
#  define DEFAULT_COUNTER_TYPE unsigned long
#endif

enum operator_type {
  DEFAULT_CTOR, COPY_CTOR,
  OTHER_CTOR, BASE, DTOR, GENERATION,
  ASSIGNMENT, CONVERSION, LESS_THAN, EQUALITY,
```

```
  POST_INCREMENT, PRE_INCREMENT,
  POST_DECREMENT, PRE_DECREMENT,
  UNARY_PLUS, UNARY_MINUS, PLUS, MINUS, TIMES, DIVIDE, MODULO,
  LEFT_SHIFT, RIGHT_SHIFT,
  PLUS_ASSIGN, MINUS_ASSIGN,
  TIMES_ASSIGN, DIVIDE_ASSIGN, MODULO_ASSIGN,
  LEFT_SHIFT_ASSIGN, RIGHT_SHIFT_ASSIGN,
  DEREFERENCE, MEMBER, SUBSCRIPT, FUNCTION_CALL,
  NEGATE, AND, AND_ASSIGN, OR, OR_ASSIGN, XOR, XOR_ASSIGN,
  NCOUNTERS,
};


struct default_recorder_visitor {

  template <typename ForwardIterator>
  typename std::iterator_traits<ForwardIterator>::value_type
    choose_element(const ForwardIterator& first,
                   const ForwardIterator& last) const {

    typedef
        typename std::iterator_traits<ForwardIterator>
                ::value_type T;

    // because nth_element is mutative, we must copy the range
    // to a temporary container.
    std::vector<T> tmp(first, last);

    typename std::vector<T>::iterator midpoint = tmp.begin() +
                                          tmp.size() / 2;
    std::nth_element(tmp.begin(), midpoint, tmp.end());

    // OK, now find the midpoint value in the original
    // container and return that iterator.
    return *std::find(first, last, *midpoint);
  }

  template <typename ForwardIterator>
  ForwardIterator transmute(ForwardIterator first,
                            ForwardIterator last) const {
    return last;
  }
};
template <typename Base = default_recorder_visitor>
struct min_element_recorder_visitor : public Base {

  template <typename ForwardIterator>
  typename std::iterator_traits<ForwardIterator>::value_type
    choose_element(ForwardIterator first,
                   ForwardIterator last) const {
```

```
    return *std::min_element(first, last);
  }
};
template <typename Base = default_recorder_visitor>
struct max_element_recorder_visitor : public Base {

  template <typename ForwardIterator>
  typename std::iterator_traits<ForwardIterator>::value_type
    choose_element(ForwardIterator first,
                   ForwardIterator last) const {
    return *std::max_element(first, last);
  }
};
template <typename Base = default_recorder_visitor>
class scale_recorder_visitor : public Base {
  double factor;

 public:

  scale_recorder_visitor() : factor(1.0) {}
  scale_recorder_visitor(double x) : factor(x) {}

  double get_factor() const { return factor; }
  void set_factor(double x) { factor = x; }

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
  template <typename ForwardIterator>
  ForwardIterator transmute(ForwardIterator first,
                            ForwardIterator last) const {
    last = Base::transmute(first, last);
    typedef
        typename iterator_traits<ForwardIterator>::value_type
        T;
    while (first != last) {
     *first = static_cast<T>(*first * factor);
     ++first;
    }
    return last;
  }
#ifdef __BORLANDC__
#  pragma option pop
#endif

  std::vector<std::string>::iterator
    transmute(std::vector<std::string>::iterator first,
              std::vector<std::string>::iterator last) const {
    return Base::transmute(first, last);
  }
```

```
};
template <typename Base = default_recorder_visitor>
class filter_recorder_visitor : public Base {
  bool filter[NCOUNTERS];

 public:
  filter_recorder_visitor(bool state = true) {
    show_all(state);
  }
  filter_recorder_visitor(const filter_recorder_visitor& x)
    : Base(x) {
    copy(&x.filter[0],
         &x.filter[sizeof(x.filter) / sizeof(x.filter[0])],
         &filter[0]);
  }
  filter_recorder_visitor(operator_type op,
                          bool state = true) {
    show_all(!state);
    show(op, state);
  }
  template <typename InputIterator>
  filter_recorder_visitor(InputIterator first,
                          InputIterator last,
                          bool state = true) {
    show_all(!state);
    show(first, last, state);
  }

  void show_all(bool state = true) {
    std::fill(&filter[0],
         &filter[sizeof(filter) / sizeof(filter[0])], state);
  }
  void hide_all() { show_all(false); }
  void show(operator_type op, bool state = true) {
    filter[op] = state;
  }
  void hide(operator_type op) {
    show(op, false);
  }
#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
  template <typename InputIterator>
  void show(InputIterator first, InputIterator last,
            bool state = true) {
    while (first != last)
      filter[*first++] = state;
  }
#ifdef __BORLANDC__
#  pragma option pop
```

```
#endif
  template <typename InputIterator>
  void hide(InputIterator first, InputIterator last) {
    show(first, last, false);
  }

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
  template <typename ForwardIterator>
  ForwardIterator transmute(ForwardIterator first,
                            ForwardIterator last) const {
    last = Base::transmute(first, last);

    int i = 0;
    while (first != last && filter[i++])
      ++first;
    if (first == last)
      return last;
    ForwardIterator result = first;
    for (--i; first != last; ++first)
      if (filter[i++]) {
        *result = *first;
        ++result;
      }
    return result;
  }
#ifdef __BORLANDC__
#  pragma option pop
#endif
};
template <typename Base = default_recorder_visitor>
class group_recorder_visitor
  : public filter_recorder_visitor<Base> {

  std::vector<std::string> names;
  std::vector<std::vector<operator_type> > groups;

 public:
  group_recorder_visitor() {}
  group_recorder_visitor(const std::string& name,
                     const std::vector<operator_type>& group) {
    add(name, group);
  }
  group_recorder_visitor(const group_recorder_visitor& x)
    : filter_recorder_visitor<Base>(x),
      names(x.names), groups(x.groups) {
  }
  template <typename InputIterator>
  group_recorder_visitor(const std::string& name,
```

```
                        InputIterator first,
                        InputIterator last) {
    add(name, first, last);
  }

  void add(const std::string& name,
           const std::vector<operator_type>& group) {
    names.push_back(name);
    groups.push_back(group);
    hide(group.begin() + 1, group.end());
  }

  template <typename InputIterator>
  void add(const std::string& name,
           InputIterator first, InputIterator last) {
    add(name, std::vector<operator_type>(first, last));
  }

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
  template <typename RandomAccessIterator>
  RandomAccessIterator
    transmute(RandomAccessIterator first,
              RandomAccessIterator last) const {
    for (int i = 0, n = groups.size(); i < n; ++i)
      for (int j = 1, m = groups[i].size(); j < m; ++j)
        first[groups[i][0]] += first[groups[i][j]];
    return filter_recorder_visitor<Base>::transmute(first,
                                                    last);
  }

  std::vector<std::string>::iterator
    transmute(std::vector<std::string>::iterator first,
              std::vector<std::string>::iterator last) const {
    for (int i = 0, n = groups.size(); i < n; ++i)
      first[groups[i][0]] = names[i].c_str();
    return filter_recorder_visitor<Base>::transmute(first,
                                                    last);
  }
#ifdef __BORLANDC__
#  pragma option pop
#endif
};
template <typename Pair>
struct iter_ref_2nd
  : public std::unary_function<Pair,
                          std::pair<typename Pair::first_type,
      typename std::iterator_traits<typename Pair::second_type>
                  ::value_type> > {
```

```
  typedef std::pair<typename Pair::first_type,
      typename std::iterator_traits<typename Pair::second_type>
                  ::value_type> result_type;
  result_type operator()(const Pair& x) {
    return std::make_pair(x.first, *x.second);
  }
};

template <typename Pair>
struct myselect2nd :
  public std::unary_function<Pair,
                             typename Pair::second_type> {
  typename Pair::second_type& operator()(Pair& x) const {
    return x.second;
  }
  const typename Pair::second_type&
    operator()(const Pair& x) const {
    return x.second;
  }
};

template <typename Sequence>
struct subscript
  : public std::unary_function<Sequence,
                           typename Sequence::value_type> {

  subscript(const typename Sequence::size_type& x = 0)
    : n(x) {}

  typename Sequence::reference operator()(Sequence& x) const {
    return x[n];
  }

  typename Sequence::const_reference
    operator()(const Sequence& x) const {
    return x[n];
  }

 protected:
  typename Sequence::size_type n;
};

template <typename Key, typename T, typename Compare,
          typename Allocator>
struct subscript<std::map<Key, T, Compare, Allocator> >
  : public std::unary_function<std::map<Key, T,
                                        Compare, Allocator>,
      typename std::map<Key, T,
                        Compare, Allocator>::mapped_type> {
```

```
  subscript() : n() {}
  subscript(const typename std::map<Key, T, Compare,
                                    Allocator>::key_type& x)
    : n(x) {}

  typename std::map<Key, T,
                    Compare, Allocator>::mapped_type&
  operator()(std::map<Key, T,
                      Compare, Allocator>& x) const {
    return x[n];
  }

  const typename std::map<Key, T,
                          Compare, Allocator>::mapped_type&
  operator()(const std::map<Key, T,
                            Compare, Allocator>& x) const {
    return x.find(n)->second;
  }

 protected:
  typename std::map<Key, T, Compare, Allocator>::key_type n;
};

template <typename Counter>
class recorder;
template <typename Counter>
std::ostream& operator<<(std::ostream& out,
                         const recorder<Counter>&rhs);

template <typename Counter = DEFAULT_COUNTER_TYPE>
class recorder {

 public:
 typedef Counter counter_type;
 typedef std::vector<Counter> counter_vector_type;
 typedef std::multimap<std::string,
                       counter_vector_type*> map_type;

 private:
  map_type counters;
  typedef std::multimap<std::string,
                        counter_vector_type> snapshot_type;
  std::vector<snapshot_type> snapshots;

  typedef std::map<std::string, counter_vector_type> datum_type;
  typedef std::vector<datum_type> data_type;
  data_type data;

  static const char* const counter_tags[NCOUNTERS];
  static map_type* registry;
```

```
 friend std::ostream& operator<< <Counter>(std::ostream& out,
                              const recorder<Counter>&rhs);

public:
 recorder() : counters(*registry), data() {
   pause();
 }

 recorder(const map_type& x) : counters(x), data() {
   pause();
 }

 void clear() { data.clear(); }
 void pause();
 void resume();
 void record();
 void reset();

 static counter_vector_type* create(const std::string& x,
      typename counter_vector_type::size_type n = NCOUNTERS) {
   if (registry == NULL)
     registry = new map_type();
   counter_vector_type* counters = new counter_vector_type(n);
   registry->insert(make_pair(x, counters));
   return counters;
 }

 std::ostream& report(std::ostream& out,
                        const char* delim = "\t") const {
   return report(out, default_recorder_visitor(), delim);
 }
 template <typename Visitor>
 std::ostream& report(std::ostream& out, const Visitor& visitor,
                 const char* delim = "\t") const;
 std::ostream& report_headings(std::ostream& out,
                           const char* delim = "\t") const {
   return report_headings(out, default_recorder_visitor(),
                           delim);
 }

 template <typename Visitor>
 std::ostream& report_headings(std::ostream& out,
                               const Visitor& visitor,
                           const char* delim = "\t") const;
 std::ostream& report_table(std::ostream& out,
                       const char* delim = "\t") const {
   return report_table(out, default_recorder_visitor(), delim);
 }
```

```
template <typename Visitor>
std::ostream& report_table(std::ostream& out,
                           const Visitor& visitor,
                       const char* delim = "\t") const;
std::ostream& report_totals(std::ostream& out,
                        const char* delim = "\t") {
  return report_totals(out, default_recorder_visitor(), delim);
}

template <typename Visitor>
std::ostream& report_totals(std::ostream& out,
                        const Visitor& visitor,
                        const char* delim = "\t");

typedef typename boost::transform_iterator_generator<
            subscript<counter_vector_type>,
            typename boost::projection_iterator_generator<
              myselect2nd<typename datum_type::value_type>,
              typename datum_type::iterator>::type>::type
    column_iterator;
typedef typename boost::transform_iterator_generator<
          subscript<counter_vector_type>,
          typename boost::projection_iterator_generator<
            myselect2nd<typename datum_type::value_type>,
            typename datum_type::const_iterator>::type>::type
    const_column_iterator;

static column_iterator
  make_column_iterator(typename datum_type::iterator ij,
                   typename counter_vector_type::size_type k) {
  return boost::make_transform_iterator(
      boost::make_projection_iterator(ij,
              myselect2nd<typename datum_type::value_type>()),
      subscript<counter_vector_type>(k));
}

static const_column_iterator
  make_column_iterator(typename datum_type::const_iterator ij,
                   typename counter_vector_type::size_type k) {
  return boost::make_transform_iterator(
      boost::make_projection_iterator(ij,
              myselect2nd<typename datum_type::value_type>()),
      subscript<counter_vector_type>(k));
}

typedef typename boost::transform_iterator_generator<
          subscript<counter_vector_type>,
          typename boost::transform_iterator_generator<
            subscript<datum_type>,
            typename data_type::iterator>::type>::type
```

```
      pole_iterator;
  typedef typename boost::transform_iterator_generator<
            subscript<counter_vector_type>,
            typename boost::transform_iterator_generator<
              subscript<datum_type>,
              typename data_type::const_iterator>::type>::type
      const_pole_iterator;

  static pole_iterator
    make_pole_iterator(typename data_type::iterator i,
                       const typename datum_type::key_type& j,
                    typename counter_vector_type::size_type k) {
    return boost::make_transform_iterator(
        boost::make_transform_iterator(i,
                        subscript<datum_type>(j)),
        subscript<counter_vector_type>(k));
  }

  #ifdef __BORLANDC__
  #  pragma option push -w-inl
  #endif
  static const_pole_iterator
    make_pole_iterator(typename data_type::const_iterator i,
                       const typename datum_type::key_type& j,
                    typename counter_vector_type::size_type k) {
    return boost::make_transform_iterator(
        boost::make_transform_iterator(i,
                        subscript<datum_type>(j)),
        subscript<counter_vector_type>(k));
  }
  #ifdef __BORLANDC__
  #  pragma option pop
  #endif
};

template <typename Counter>
const char* const
  recorder<Counter>::counter_tags[NCOUNTERS] = {
  "x()", "x(y)",
  "x(?)", "base", "~x", "gen",
  "op=", "cast()", "op<", "op==",
  "op++", "++op",
  "op--", "--op",
  "op+()", "op-()", "op+", "op-", "op*", "op/", "op%",
  "op<<", "op>>",
  "op+=", "op-=", "op*=", "op/=", "op%=",
  "op<<=", "op>>=",
  "*op", "op->", "op[]", "op()",
  "op~", "op&", "op&=", "op|", "op|=", "op^", "op^=",
};
```

```
template <typename Counter>
typename recorder<Counter>::map_type*
recorder<Counter>::registry;

template <typename Counter>
void recorder<Counter>::pause() {
  snapshots.push_back(snapshot_type());
  snapshot_type& snapshot = snapshots.back();
  transform(counters.begin(), counters.end(),
            inserter(snapshot, snapshot.end()),
            iter_ref_2nd<typename map_type::value_type>());
}
template <typename Counter>
void recorder<Counter>::resume() {
  // obtain the current counter values
  snapshot_type current;
  transform(counters.begin(), counters.end(),
            inserter(current, current.end()),
            iter_ref_2nd<typename map_type::value_type>());
  // get the saved values from the pause action
  snapshot_type &snapshot = snapshots.back();
  // baseline += current - snapshot
  snapshot_type& baseline = snapshots[snapshots.size() - 2];
  for (typename snapshot_type::iterator i = current.begin(),
                                j = snapshot.begin(),
                                k = baseline.begin();
       i != current.end();
       ++i, ++j, ++k) {
    transform(k->second.begin(), k->second.end(),
              i->second.begin(), k->second.begin(),
              std::plus<counter_type>());
    transform(k->second.begin(), k->second.end(),
              j->second.begin(), k->second.begin(),
              std::minus<counter_type>());
  }
  snapshots.pop_back();
}
template <typename Counter>
void recorder<Counter>::record() {
  data.push_back(datum_type());
  datum_type& datum = data.back();
  snapshot_type& baseline = snapshots.back();
  typename snapshot_type::iterator j = baseline.begin();
  for (typename map_type::iterator i = counters.begin();
       i != counters.end();
       ++i, ++j) {
    counter_vector_type& elt = datum[i->first];
    if (elt.size() < i->second->size()) {
      elt.reserve(i->second->size());
```

```
      elt.insert(elt.end(), i->second->size() - elt.size(), 0);
    }
    transform(i->second->begin(), i->second->end(),
              elt.begin(), elt.begin(),
              std::plus<counter_type>());
    transform(elt.begin(), elt.end(),
              j->second.begin(), elt.begin(),
              std::minus<counter_type>());
  }
  reset();
}
template <typename Counter>
void recorder<Counter>::reset() {
  snapshot_type& baseline = snapshots.back();
  baseline.clear();
  transform(counters.begin(), counters.end(),
            inserter(baseline, baseline.end()),
            iter_ref_2nd<typename map_type::value_type>());
}
template <typename Counter>
template <typename Visitor>
std::ostream&
recorder<Counter>::report(std::ostream& out,
                          const Visitor& visitor,
                          const char* delim) const {
  report_headings(out, visitor, delim);
  report_table(out, visitor, delim);
  const_cast<recorder<Counter>*>(this)->report_totals(out,
                                                      visitor,
                                                      delim);
  return out;
}
template <typename Counter>
template <typename Visitor>
std::ostream&
recorder<Counter>::report_headings(std::ostream& out,
                                   const Visitor& visitor,
                                   const char* delim) const {
  std::vector<std::string> tmp(counter_tags,
                     counter_tags +
                       sizeof(counter_tags) /
                       sizeof(counter_tags[0]));
  std::vector<std::string>::iterator last =
          visitor.transmute(tmp.begin(), tmp.end());

  out << std::setw(20) << "Type" << delim;
  std::copy(tmp.begin(), last,
       std::ostream_iterator<std::string>(out, delim));
  return out << "Total" << std::endl;
}
```

```
template <typename Counter>
template <typename Visitor>
std::ostream&
  recorder<Counter>::report_table(std::ostream& out,
                                  const Visitor &visitor,
                                  const char* delim) const {
  counter_vector_type tmp;
  for (typename datum_type::const_iterator
       i = data.front().begin();
       i != data.front().end();
       ++i) {
    tmp.clear();
    for (size_t j = 0; j < i->second.size(); ++j)
      tmp.push_back(visitor.choose_element(
              make_pole_iterator(data.begin(), i->first, j),
              make_pole_iterator(data.end(), i->first, j)));
    typename counter_vector_type::iterator last =
        visitor.transmute(tmp.begin(), tmp.end());

    out << std::setw(20) << i->first.c_str() << delim;
    std::copy(tmp.begin(), last,
         std::ostream_iterator<counter_type>(out, delim));
    out << std::accumulate(tmp.begin(), last, counter_type(0))
        << std::endl;
  }
  return out;
}
template <typename Counter>
template <typename Visitor>
std::ostream&
recorder<Counter>::report_totals(std::ostream& out,
                                 const Visitor& visitor,
                                 const char* delim) {
  counter_vector_type tmp, tmp2;
  for (size_t op_idx = 0; op_idx < NCOUNTERS; ++op_idx) {
    tmp.clear();
    for (size_t j = 0; j < data.size(); ++j)
      if (op_idx == GENERATION)
        tmp.push_back(*std::max_element(
                make_column_iterator(data[j].begin(), op_idx),
                make_column_iterator(data[j].end(), op_idx)));
      else
        tmp.push_back(std::accumulate(
                make_column_iterator(data[j].begin(), op_idx),
                make_column_iterator(data[j].end(), op_idx),
                counter_type(0)));

    // At this point, tmp is a vector of counts of this
    //  operation.  apply the functor to get a single count
    // from this set (max, min, etc)
```

```
    tmp2.push_back(visitor.choose_element(tmp.begin(),
                                          tmp.end()));
  }
  typename counter_vector_type::iterator last =
      visitor.transmute(tmp2.begin(), tmp2.end());

  out << std::setw(20) << "Total" << delim;
  copy(tmp2.begin(), last,
       std::ostream_iterator<counter_type>(out, delim));
  return out << std::accumulate(tmp2.begin(), last,
                                counter_type(0))
             << std::endl;
}
template <typename Counter>
std::ostream& operator<< (std::ostream& out,
                          const recorder<Counter>&rhs) {
  return rhs.report(out);
}
template <typename Difference, typename Counter>
class difference_counter;




template <typename Difference,
          typename Counter = DEFAULT_COUNTER_TYPE>
class difference_counter {
public:
  typedef Difference difference_type;
  typedef Counter counter_type;

  difference_counter() : base_difference(), generation(0) {
    ++counters[DEFAULT_CTOR];
  }

  difference_counter(const difference_counter& x)
    : base_difference(x.base_difference),
      generation(x.generation + 1) {
    ++counters[COPY_CTOR];
    counters[GENERATION] = max(counters[GENERATION], generation);
  }

  difference_counter(const difference_type& x)
    : base_difference(x), generation(0) {
    ++counters[OTHER_CTOR];
  }
  ~difference_counter() {
    ++counters[DTOR];
  }
  difference_counter& operator=(const difference_counter& x) {
```

```
  ++counters[ASSIGNMENT];
  base_difference = x.base_difference;
  generation = x.generation;
  return *this;
}
operator difference_type() const {
  ++counters[CONVERSION];
  return static_cast<difference_type>(base_difference);
}
difference_type& base() {
  ++counters[BASE];
  return base_difference;
}

const difference_type& base() const {
  ++counters[BASE];
  return base_difference;
}

difference_counter operator~() const {
  ++counters[NEGATE];
  return difference_counter<Difference,
                            Counter>(~base_difference);
}

difference_counter
  operator&(const difference_counter& n) const {
  ++counters[AND];
  return difference_counter<Difference, Counter>(
      base_difference & n.base_difference);
}

difference_counter operator&(const difference_type& n) const {
  ++counters[AND];
  return difference_counter<Difference,
                Counter>(base_difference & n);
}

difference_counter& operator&=(const difference_counter& n) {
  ++counters[AND_ASSIGN];
  base_difference &= n.base_difference;
  return *this;
}

difference_counter& operator&=(const difference_type& n) {
  ++counters[AND_ASSIGN];
  base_difference &= n;
  return *this;
}
```

```
difference_counter
  operator|(const difference_counter& n) const {
  ++counters[OR];
  return difference_counter<Difference, Counter>(
      base_difference | n.base_difference);
}

difference_counter operator|(const difference_type& n) const {
  ++counters[OR];
  return difference_counter<Difference,
                Counter>(base_difference | n);
}

difference_counter& operator|=(const difference_counter& n) {
  ++counters[OR_ASSIGN];
  base_difference |= n.base_difference;
  return *this;
}

difference_counter& operator|=(const difference_type& n) {
  ++counters[OR_ASSIGN];
  base_difference |= n;
  return *this;
}

difference_counter
  operator^(const difference_counter& n) const {
  ++counters[XOR];
  return difference_counter<Difference, Counter>(
      base_difference ^ n.base_difference);
}

difference_counter operator^(const difference_type& n) const {
  ++counters[XOR];
  return difference_counter<Difference,
                Counter>(base_difference ^ n);
}

difference_counter& operator^=(const difference_counter& n) {
  ++counters[XOR_ASSIGN];
  base_difference ^= n.base_difference;
  return *this;
}

difference_counter& operator^=(const difference_type& n) {
  ++counters[XOR_ASSIGN];
  base_difference ^= n;
  return *this;
}
```

```
difference_counter operator+() const {
  ++counters[UNARY_PLUS];
  return difference_counter<Difference,
                            Counter>(base_difference);
}

difference_counter operator-() const {
  ++counters[UNARY_MINUS];
  return difference_counter<Difference,
                            Counter>(-base_difference);
}

difference_counter& operator++() {
  ++counters[PRE_INCREMENT];
  ++base_difference;
  return *this;
}

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
difference_counter operator++(int) {
  difference_counter result = *this;
  ++counters[POST_INCREMENT];
  base_difference++;
  return result;
}
#ifdef __BORLANDC__
#  pragma option pop
#endif

difference_counter& operator--() {
  ++counters[PRE_DECREMENT];
  --base_difference;
  return *this;
}

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
difference_counter operator--(int) {
  difference_counter result(*this);
  ++counters[POST_DECREMENT];
  base_difference--;
  return result;
}
#ifdef __BORLANDC__
#  pragma option pop
#endif
```

```
difference_counter operator+(const difference_type& x) const {
  ++counters[PLUS];
  return difference_counter<Difference,
          Counter>(base_difference + x);
}

difference_counter
  operator+(const difference_counter& x) const {
  ++counters[PLUS];
  return difference_counter<Difference,
                            Counter>(
      base_difference + x.base_difference);
}

difference_counter& operator+=(const difference_type& x) {
  ++counters[PLUS_ASSIGN];
  base_difference += x;
  return *this;
}

difference_counter& operator+=(const difference_counter& x) {
  ++counters[PLUS_ASSIGN];
  base_difference += x.base_difference;
  return *this;
}

difference_counter operator-(const difference_type& x) const {
  ++counters[MINUS];
  return difference_counter<Difference,
                 Counter>(base_difference - x);
}

difference_counter
  operator-(const difference_counter& x) const {
  ++counters[MINUS];
  return difference_counter<Difference, Counter>(
      base_difference - x.base_difference);
}

difference_counter& operator-=(const difference_type& x) {
  ++counters[MINUS_ASSIGN];
  base_difference -= x;
  return *this;
}

difference_counter& operator-=(const difference_counter& x) {
  ++counters[MINUS_ASSIGN];
  base_difference -= x.base_difference;
  return *this;
}
```

```
difference_counter operator*(const difference_type& x) const {
  ++counters[TIMES];
  return difference_counter<Difference,
                Counter>(base_difference * x);
}

difference_counter
  operator*(const difference_counter& x) const {
  ++counters[TIMES];
  return difference_counter<Difference, Counter>(
      base_difference * x.base_difference);
}

difference_counter& operator*=(const difference_type& x) {
  ++counters[TIMES_ASSIGN];
  base_difference *= x;
  return *this;
}

difference_counter& operator*=(const difference_counter& x) {
  ++counters[TIMES_ASSIGN];
  base_difference *= x.base_difference;
  return *this;
}

difference_counter operator/(const difference_type& x) const {
  ++counters[DIVIDE];
  return difference_counter<Difference,
                Counter>(base_difference / x);
}

difference_counter
  operator/(const difference_counter& x) const {
  ++counters[DIVIDE];
  return difference_counter<Difference, Counter>(
      base_difference / x.base_difference);
}

difference_counter& operator/=(const difference_type& x) {
  ++counters[DIVIDE_ASSIGN];
  base_difference /= x;
  return *this;
}

difference_counter& operator/=(const difference_counter& x) {
  ++counters[DIVIDE_ASSIGN];
  base_difference /= x.base_difference;
  return *this;
}
```

```
difference_counter
  operator%(const difference_counter& x) const {
  ++counters[MODULO];
  return difference_counter<Difference, Counter>(
      base_difference % x.base_difference);
}

difference_counter operator%(const difference_type& x) const {
  ++counters[MODULO];
  return difference_counter<Difference,
                Counter>(base_difference % x);
}

difference_counter& operator%=(const difference_counter& x) {
  ++counters[MODULO_ASSIGN];
  base_difference %= x.base_difference;
  return *this;
}

difference_counter& operator%=(const difference_type& x) {
  ++counters[MODULO_ASSIGN];
  base_difference %= x;
  return *this;
}

difference_counter operator<<(int n) const {
  ++counters[LEFT_SHIFT];
  return difference_counter<Difference,
                Counter>(base_difference << n);
}

difference_counter& operator<<=(int n) {
  ++counters[LEFT_SHIFT_ASSIGN];
  base_difference <<= n;
  return *this;
}

difference_counter operator>>(int n) const {
  ++counters[RIGHT_SHIFT];
  return difference_counter<Difference,
                Counter>(base_difference >> n);
}

difference_counter& operator>>=(int n) {
  ++counters[RIGHT_SHIFT_ASSIGN];
  base_difference >>= n;
  return *this;
}
```

```
friend bool operator==(const difference_counter& lhs,
                       const difference_counter& rhs) {
  ++difference_counter::counters[EQUALITY];
  return lhs.base_difference == rhs.base_difference;
}

friend bool operator==(const difference_counter& lhs,
                       const difference_type& rhs) {
  ++difference_counter::counters[EQUALITY];
  return lhs.base_difference == rhs;
}

friend bool operator==(const difference_type& lhs,
                       const difference_counter& rhs) {
  ++difference_counter::counters[EQUALITY];
  return lhs == rhs.base_difference;
}

friend bool operator!=(const difference_counter& lhs,
                       const difference_counter& rhs) {
  ++difference_counter::counters[EQUALITY];
  return !(lhs.base_difference == rhs.base_difference);
}

friend bool operator!=(const difference_counter& lhs,
                       const difference_type& rhs) {
  return !(lhs.base_difference == rhs);
}

friend bool operator!=(const difference_type& lhs,
                       const difference_counter& rhs) {
  return !(lhs == rhs.base_difference);
}
friend bool operator<(const difference_counter& lhs,
                      const difference_counter& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return lhs.base_difference < rhs.base_difference;
}

friend bool operator<(const difference_counter& lhs,
                      const difference_type& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return lhs.base_difference < rhs;
}

friend bool operator<(const difference_type& lhs,
                      const difference_counter& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return lhs < rhs.base_difference;
}
```

```
friend bool operator>(const difference_counter& lhs,
                      const difference_counter& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return rhs.base_difference < lhs.base_difference;
}

friend bool operator>(const difference_counter& lhs,
                      const difference_type& rhs) {
  return rhs < lhs.base_difference;
}

friend bool operator>(const difference_type& lhs,
                      const difference_counter& rhs) {
  return rhs.base_difference < lhs;
}

friend bool operator<=(const difference_counter& lhs,
                       const difference_counter& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return !(rhs.base_difference < lhs.base_difference);
}

friend bool operator<=(const difference_counter& lhs,
                      const difference_type& rhs) {
  return !(rhs < lhs.base_difference);
}

friend bool operator<=(const difference_type& lhs,
                      const difference_counter& rhs) {
  return !(rhs.base_difference < lhs);
}

friend bool operator>=(const difference_counter& lhs,
                       const difference_counter& rhs) {
  ++difference_counter::counters[LESS_THAN];
  return !(lhs.base_difference < rhs.base_difference);
}

friend bool operator>=(const difference_counter& lhs,
                      const difference_type& rhs) {
  return !(lhs.base_difference < rhs);
}

friend bool operator>=(const difference_type& lhs,
                      const difference_counter& rhs) {
  return !(lhs < rhs.base_difference);
}
template <typename T>
  friend T* operator+(T* p, const difference_counter& n) {
```

```
    ++counters[PLUS];
    return p + n.base_difference;
  }

template <typename T>
  friend const T* operator+(const T* p,
                            const difference_counter& n) {
    ++counters[PLUS];
    return p + n.base_difference;
  }

template <typename T>
  friend T*& operator+=(T*& p, const difference_counter& n) {
    ++counters[PLUS];
    return p += n.base_difference;
  }

template <typename T>
  friend T* operator-(T* p, const difference_counter& n) {
    ++counters[MINUS];
    return p - n.base_difference;
  }

template <typename T>
  friend const T* operator-(const T* p,
                            const difference_counter& n) {
    ++counters[MINUS];
    return p - n.base_difference;
  }

template <typename T>
  friend T*& operator-=(T*& p, const difference_counter& n) {
    ++counters[MINUS];
    return p -= n.base_difference;
  }

friend difference_counter
  operator&(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[AND];
  return difference_counter<Difference, Counter>(
      lhs & rhs.base_difference);
}

friend difference_type&
  operator&=(const difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[AND];
  return lhs &= rhs.base_difference;
}
```

```
friend difference_counter
  operator|(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[OR];
  return difference_counter<Difference, Counter>(
      lhs | rhs.base_difference);
}

friend difference_type&
  operator|=(const difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[OR];
  return lhs |= rhs.base_difference;
}

friend difference_counter
  operator^(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[XOR];
  return difference_counter<Difference, Counter>(
      lhs ^ rhs.base_difference);
}

friend difference_type&
  operator^=(const difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[XOR];
  return lhs ^= rhs.base_difference;
}

friend difference_counter
  operator+(const difference_type& lhs,
            const difference_counter& rhs) {
  ++difference_counter::counters[PLUS];
  return difference_counter<Difference, Counter>(
      lhs + rhs.base_difference);
}

friend difference_type&
  operator+=(difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[PLUS];
  lhs += rhs.base_difference;
  return lhs;
}

friend difference_counter
  operator-(const difference_type& lhs,
            const difference_counter& rhs) {
```

```
  ++counters[MINUS];
  return difference_counter<Difference, Counter>(
      lhs - rhs.base_difference);
}

friend difference_type&
  operator-=(difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[MINUS];
  lhs -= rhs.base_difference;
  return lhs;
}

friend difference_counter
  operator*(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[TIMES];
  return difference_counter<Difference, Counter>(
      lhs * rhs.base_difference);
}

friend difference_type&
  operator*=(difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[TIMES];
  lhs *= rhs.base_difference;
  return lhs;
}

friend difference_counter
  operator/(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[DIVIDE];
  return difference_counter<Difference, Counter>(
      lhs / rhs.base_difference);
}

friend difference_type&
  operator/=(difference_type& lhs,
             const difference_counter& rhs) {
  ++counters[DIVIDE];
  lhs /= rhs.base_difference;
  return lhs;
}

friend difference_counter
  operator%(const difference_type& lhs,
            const difference_counter& rhs) {
  ++counters[MODULO];
  return difference_counter<Difference, Counter>(
```

```
        lhs % rhs.base_difference);
  }

  friend difference_type&
    operator%=(difference_type& lhs,
               const difference_counter& rhs) {
    ++counters[MODULO];
    lhs %= rhs.base_difference;
    return lhs;
  }

 protected:
  difference_type base_difference;
  Counter generation;

  static typename recorder<Counter>::counter_vector_type&
         counters;
};

template <typename Difference,
          typename Counter>
  typename recorder<Counter>::counter_vector_type&
    difference_counter<Difference, Counter>::counters =
      *recorder<Counter>::create("difference_counter");




template <typename Iterator, typename Difference,
          typename Counter>
class iterator_counter;
template <typename Iterator, typename Difference,
          typename Counter>
bool operator==(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);
template <typename Iterator, typename Difference,
          typename Counter>
bool operator!=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);
template <typename Iterator, typename Difference,
          typename Counter>
bool operator<(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
```

```
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);
template <typename Iterator, typename Difference,
          typename Counter>
bool operator>(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);
template <typename Iterator, typename Difference,
          typename Counter>
bool operator<=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);
template <typename Iterator, typename Difference,
          typename Counter>
bool operator>=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs);


template <typename Iterator,
          typename Difference =
            difference_counter<typename
              std::iterator_traits<Iterator>::difference_type>,
          typename Counter = DEFAULT_COUNTER_TYPE>
class iterator_counter
#if defined(_RWSTD_VER) && (_RWSTD_VER <= 0x020101)
  : public std::iterator<
      typename std::iterator_traits<Iterator>::
               iterator_category,
      typename std::iterator_traits<Iterator>::value_type,
      typename std::iterator_traits<Iterator>::difference_type,
      typename std::iterator_traits<Iterator>::pointer,
      typename std::iterator_traits<Iterator>::reference>
#endif
{
 public:
  typedef Iterator iterator_type;
  typedef Difference difference_type;
  typedef Counter counter_type;
  typedef typename std::iterator_traits<Iterator>::value_type
      value_type;
  typedef typename std::iterator_traits<Iterator>::pointer
      pointer;
  typedef typename std::iterator_traits<Iterator>::reference
```

```
    reference;
typedef typename
    std::iterator_traits<Iterator>::iterator_category
    iterator_category;

iterator_counter() : base_iterator(), generation(0) {
  ++counters[DEFAULT_CTOR];
}

iterator_counter(const iterator_counter& x)
  : base_iterator(x.base_iterator),
    generation(x.generation + 1) {
  ++counters[COPY_CTOR];
  counters[GENERATION] = max(counters[GENERATION], generation);
}

iterator_counter(const iterator_type& x)
  : base_iterator(x), generation(0) {
  ++counters[OTHER_CTOR];
}
~iterator_counter() {
  ++counters[DTOR];
}
iterator_counter& operator=(const iterator_counter& x) {
  ++counters[ASSIGNMENT];
  base_iterator = x.base_iterator;
  generation = x.generation;
  return *this;
}
iterator_type& base() {
  ++counters[BASE];
  return base_iterator;
}

const iterator_type& base() const {
  ++counters[BASE];
  return base_iterator;
}

reference operator*() const {
  ++counters[DEREFERENCE];
  return *base_iterator;
}

pointer operator->() const {
  ++counters[MEMBER];
  return &(*base_iterator);
}

iterator_counter& operator++() {
```

```
  ++counters[PRE_INCREMENT];
  ++base_iterator;
  return *this;
}

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
iterator_counter operator++(int) {
  iterator_counter result(*this);
  ++counters[POST_INCREMENT];
  base_iterator++;
  return result;
}
#ifdef __BORLANDC__
#  pragma option pop
#endif

iterator_counter& operator--() {
  ++counters[PRE_DECREMENT];
  --base_iterator;
  return *this;
}

#ifdef __BORLANDC__
#  pragma option push -w-inl
#endif
iterator_counter operator--(int) {
  iterator_counter result(*this);
  ++counters[POST_DECREMENT];
  base_iterator--;
  return result;
}
#ifdef __BORLANDC__
#  pragma option pop
#endif

iterator_counter operator+(const difference_type& n) const {
  ++counters[PLUS];
  return iterator_counter<Iterator, Difference, Counter>(
      base_iterator + unadapted_difference_type(n));
}

iterator_counter& operator+=(const difference_type& n) {
  ++counters[PLUS_ASSIGN];
  base_iterator += unadapted_difference_type(n);
  return *this;
}

iterator_counter operator-(const difference_type& n) const {
```

```
  ++counters[MINUS];
  return iterator_counter<Iterator, Difference, Counter>(
      base_iterator - unadapted_difference_type(n));
 }

 difference_type operator-(const iterator_counter& i) const {
  ++counters[MINUS];
  return difference_type(base_iterator - i.base_iterator);
 }

 iterator_counter& operator-=(const difference_type& n) {
  ++counters[MINUS_ASSIGN];
  base_iterator -= unadapted_difference_type(n);
  return *this;
 }

 reference operator[](const difference_type& n) const {
  ++counters[SUBSCRIPT];
  return base_iterator[unadapted_difference_type(n)];
 }

 friend bool operator== <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend bool operator!= <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend bool operator< <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend bool operator> <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend bool operator<= <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend bool operator>= <Iterator, Difference, Counter>(
     const iterator_counter& lhs,
     const iterator_counter& rhs);
 friend iterator_counter operator+(
     const difference_type& n,
     const iterator_counter& i) {
  ++iterator_counter::counters[PLUS];
  return iterator_counter<Iterator, Difference, Counter>(
      i.base_iterator + unadapted_difference_type(n));
 }

protected:
 iterator_type base_iterator;
 Counter generation;
```

```
  static typename recorder<Counter>::counter_vector_type&
         counters;

  template <typename T>
    static const
      typename std::iterator_traits<Iterator>::difference_type&
        unadapted_difference_type(const T& n) {
    return n;
  }

  template <typename xDifference, typename xCounter>
    static const
      typename std::iterator_traits<Iterator>::difference_type&
        unadapted_difference_type(
          const difference_counter<xDifference,
                                   xCounter>& n) {
    return n.base();
  }
};

template <typename Iterator, typename Difference,
          typename Counter>
  typename recorder<Counter>::counter_vector_type&
    iterator_counter<Iterator, Difference,
                     Counter>::counters =
      *recorder<Counter>::create("iterator_counter");

template <typename Iterator, typename Difference,
          typename Counter>
bool operator==(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[EQUALITY];
  return lhs.base_iterator == rhs.base_iterator;
}

template <typename Iterator, typename Difference,
          typename Counter>
bool operator!=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[EQUALITY];
  return !(lhs.base_iterator == rhs.base_iterator);
```

```
}
template <typename Iterator, typename Difference,
          typename Counter>
bool operator<(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[LESS_THAN];
  return lhs.base_iterator < rhs.base_iterator;
}

template <typename Iterator, typename Difference,
          typename Counter>
bool operator>(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[LESS_THAN];
  return rhs.base_iterator < lhs.base_iterator;
}

template <typename Iterator, typename Difference,
          typename Counter>
bool operator<=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[LESS_THAN];
  return !(rhs.base_iterator < lhs.base_iterator);
}

template <typename Iterator, typename Difference,
          typename Counter>
bool operator>=(
    const iterator_counter<Iterator, Difference,
                           Counter>& lhs,
    const iterator_counter<Iterator, Difference,
                           Counter>& rhs) {
  ++iterator_counter<Iterator, Difference,
                     Counter>::counters[LESS_THAN];
  return !(lhs.base_iterator < rhs.base_iterator);
}
```

```
template <typename Iterator, typename Difference,
          typename Counter,
          typename Predicate, typename T>
inline iterator_counter<Iterator, Difference, Counter>
  __stable_partition_aux(
    iterator_counter<Iterator, Difference,
                     Counter> first,
    iterator_counter<Iterator, Difference,
                     Counter> last,
    Predicate pred, T*, Difference*) {

  _Temporary_buffer<
    iterator_counter<Iterator, Difference, Counter>, T>
    buf(first, last);
  if (buf.size() > 0)
    return __stable_partition_adaptive(first, last, pred,
                  Difference(buf.requested_size()),
                  buf.begin(),
                  // conversion to Difference added
                  Difference(buf.size()));
  else
    return __inplace_stable_partition(first, last, pred,
                  Difference(buf.requested_size()));
}
template <typename Value, typename Counter>
class value_counter;
template <typename Value, typename Counter>
bool operator==(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
template <typename Value, typename Counter>
bool operator!=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
template <typename Value, typename Counter>
bool operator<(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
template <typename Value, typename Counter>
bool operator>(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
template <typename Value, typename Counter>
bool operator<=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
template <typename Value, typename Counter>
bool operator>=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs);
```

```
template <typename Value,
          typename Counter = DEFAULT_COUNTER_TYPE>
class value_counter {
 public:
  typedef Value value_type;
  typedef Counter counter_type;

  value_counter() : base_value(), generation(0) {
    ++counters[DEFAULT_CTOR];
  }

  value_counter(const value_counter& x)
    : base_value(x.base_value), generation(x.generation + 1) {
    ++counters[COPY_CTOR];
    counters[GENERATION] = std::max(counters[GENERATION],
                                    generation);
  }

  explicit value_counter(const value_type& x)
    : base_value(x), generation(0) {
    ++counters[OTHER_CTOR];
  }
  ~value_counter() {
    ++counters[DTOR];
  }
  value_counter& operator=(const value_counter& x) {
    ++counters[ASSIGNMENT];
    base_value = x.base_value;
    generation = x.generation;
    return *this;
  }
  value_type& base() {
    ++counters[BASE];
    return base_value;
  }

  const value_type& base() const {
    ++counters[BASE];
    return base_value;
  }

  friend bool operator== <Value, Counter>(
      const value_counter& lhs,
      const value_counter& rhs);
  friend bool operator!= <Value, Counter>(
      const value_counter& lhs,
      const value_counter& rhs);
  friend bool operator< <Value, Counter>(
      const value_counter& lhs,
```

```
      const value_counter& rhs);
  friend bool operator> <Value, Counter>(
      const value_counter& lhs,
      const value_counter& rhs);
  friend bool operator<= <Value, Counter>(
      const value_counter& lhs,
      const value_counter& rhs);
  friend bool operator>= <Value, Counter>(
      const value_counter& lhs,
      const value_counter& rhs);

 protected:
  value_type base_value;
  Counter generation;

  static typename recorder<Counter>::counter_vector_type&
         counters;
};

template <typename Value, typename Counter>
  typename recorder<Counter>::counter_vector_type&
    value_counter<Value, Counter>::counters =
      *recorder<Counter>::create("value_counter");

template <typename Value, typename Counter>
bool operator==(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[EQUALITY];
  return lhs.base_value == rhs.base_value;
}

template <typename Value, typename Counter>
bool operator!=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[EQUALITY];
  return !(lhs.base_value == rhs.base_value);
}
template <typename Value, typename Counter>
bool operator<(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[LESS_THAN];
  return lhs.base_value < rhs.base_value;
}

template <typename Value, typename Counter>
bool operator>(
    const value_counter<Value, Counter>& lhs,
```

```
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[LESS_THAN];
  return rhs.base_value < lhs.base_value;
}

template <typename Value, typename Counter>
bool operator<=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[LESS_THAN];
  return !(rhs.base_value < lhs.base_value);
}

template <typename Value, typename Counter>
bool operator>=(
    const value_counter<Value, Counter>& lhs,
    const value_counter<Value, Counter>& rhs) {
  ++value_counter<Value, Counter>::counters[LESS_THAN];
  return !(lhs.base_value < rhs.base_value);
}
template <typename AdaptableGenerator,
          typename Counter = DEFAULT_COUNTER_TYPE>
class generator_counter {
 public:
  typedef AdaptableGenerator function_type;
  typedef Counter counter_type;
  typedef typename AdaptableGenerator::result_type result_type;

  generator_counter() : base_function(), generation(0) {
    ++counters[DEFAULT_CTOR];
  }

  generator_counter(const generator_counter& x)
    : base_function(x.base_function),
      generation(x.generation + 1) {
    ++counters[COPY_CTOR];
    counters[GENERATION] = max(counters[GENERATION], generation);
  }

  explicit generator_counter(const function_type& x)
    : base_function(x), generation(0) {
    ++counters[OTHER_CTOR];
  }
  ~generator_counter() {
    ++counters[DTOR];
  }
  generator_counter& operator=(const generator_counter& x) {
    ++counters[ASSIGNMENT];
    base_function = rhs.base_function;
    generation = x.generation;
```

```
    return *this;
  }
  function_type& base() {
    ++counters[BASE];
    return base_function;
  }

  const function_type& base() const {
    ++counters[BASE];
    return base_function;
  }

  result_type operator()() {
    ++counters[FUNCTION_CALL];
    return base_function();
  }

  result_type operator()() const {
    ++counters[FUNCTION_CALL];
    return base_function();
  }

 protected:
  function_type base_function;
  Counter generation;

  static typename recorder<Counter>::counter_vector_type&
         counters;
};

template <typename AdaptableGenerator, typename Counter>
  typename recorder<Counter>::counter_vector_type&
    generator_counter<AdaptableGenerator,
                      Counter>::counters =
      *recorder<Counter>::create("generator_counter");
template <typename AdaptableGenerator>
  generator_counter<AdaptableGenerator> function_counter(
    const AdaptableGenerator& x) {
  return generator_counter<AdaptableGenerator>(x);
}
template <typename AdaptableUnaryFunction,
          typename Counter = DEFAULT_COUNTER_TYPE>
class unary_function_counter
  : public std::unary_function<
        typename AdaptableUnaryFunction::argument_type,
        typename AdaptableUnaryFunction::result_type> {
 public:
  typedef AdaptableUnaryFunction function_type;
  typedef typename AdaptableUnaryFunction::argument_type
      argument_type;
```

```
 typedef typename AdaptableUnaryFunction::result_type
     result_type;
 typedef Counter counter_type;

 unary_function_counter() : base_function(), generation(0) {
   ++counters[DEFAULT_CTOR];
 }

 unary_function_counter(const unary_function_counter& x)
   : base_function(x.base_function),
     generation(x.generation + 1) {
   ++counters[COPY_CTOR];
   counters[GENERATION] = max(counters[GENERATION], generation);
 }

 explicit unary_function_counter(const function_type& x)
   : base_function(x), generation(0) {
   ++counters[OTHER_CTOR];
 }
 ~unary_function_counter() {
   ++counters[DTOR];
 }
 unary_function_counter&
   operator=(const unary_function_counter& x) {
   ++counters[ASSIGNMENT];
   base_function = x.base_function;
   generation = x.generation;
   return *this;
 }
 function_type& base() {
   ++counters[BASE];
   return base_function;
 }

 const function_type& base() const {
   ++counters[BASE];
           return base_function;
 }

 result_type operator()(const argument_type& arg) const {
   ++counters[FUNCTION_CALL];
   return base_function(arg);
 }

protected:
 function_type base_function;
 Counter generation;

 static typename recorder<Counter>::counter_vector_type&
        counters;
```

```
};

template <typename AdaptableUnaryFunction,
          typename Counter>
  typename recorder<Counter>::counter_vector_type&
    unary_function_counter<AdaptableUnaryFunction,
                           Counter>::counters =
      *recorder<Counter>::create("unary_function_counter");
template <typename AdaptableUnaryFunction>
  unary_function_counter<AdaptableUnaryFunction>
    function_counter1(
    const AdaptableUnaryFunction& x) {
  return unary_function_counter<AdaptableUnaryFunction>(x);
}
template <typename AdaptableBinaryFunction,
          typename Counter = DEFAULT_COUNTER_TYPE>
class binary_function_counter
  : public std::binary_function<
        typename AdaptableBinaryFunction::first_argument_type,
        typename AdaptableBinaryFunction::second_argument_type,
        typename AdaptableBinaryFunction::result_type> {
 public:
  typedef AdaptableBinaryFunction function_type;
  typedef typename AdaptableBinaryFunction::first_argument_type
      first_argument_type;
  typedef typename AdaptableBinaryFunction::second_argument_type
      second_argument_type;
  typedef typename AdaptableBinaryFunction::result_type
      result_type;
  typedef Counter counter_type;

  binary_function_counter() : base_function(), generation(0) {
    ++counters[DEFAULT_CTOR];
  }

  binary_function_counter(const binary_function_counter& x)
    : base_function(x.base_function),
      generation(x.generation + 1) {
    ++counters[COPY_CTOR];
    counters[GENERATION] = max(counters[GENERATION], generation);
  }

  explicit binary_function_counter(const function_type& x)
    : base_function(x), generation(0) {
    ++counters[OTHER_CTOR];
  }
  ~binary_function_counter() {
    ++counters[DTOR];
  }
  binary_function_counter&
```

```
    operator=(const binary_function_counter& x) {
    ++counters[ASSIGNMENT];
    base_function = x.base_function;
    generation = x.generation;
    return *this;
  }
  function_type& base() {
    ++counters[BASE];
    return base_function;
  }

  const function_type& base() const {
    ++counters[BASE];
    return base_function;
  }

  result_type operator()(const first_argument_type& x,
                         const second_argument_type& y) const {
    ++counters[FUNCTION_CALL];
    return base_function(x, y);
  }

 protected:
  function_type base_function;
  Counter generation;

  static typename recorder<Counter>::counter_vector_type&
         counters;
};

template <typename AdaptableBinaryFunction,
          typename Counter>
  typename recorder<Counter>::counter_vector_type&
    binary_function_counter<AdaptableBinaryFunction,
                            Counter>::counters =
      *recorder<Counter>::create("binary_function_counter");
template <typename AdaptableBinaryFunction>
  binary_function_counter<AdaptableBinaryFunction>
    function_counter2(
    const AdaptableBinaryFunction& x) {
  return binary_function_counter<AdaptableBinaryFunction>(x);
}

}

#endif  // COUNTERS_H
```

B.2 Test 1

B.2.1 Test 1a

"opcount_test1a.cpp" 174 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "timer.h"

#define DEFAULT_COUNTER_TYPE int
#include "counters.h"
#include "router.hpp"

using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using gdfa::router;
using gdfa::destination;

using namespace counters;

template <typename OutputIterator>
class less_than_n : public destination<value_counter<int> >
{
  private:
    OutputIterator out;
    value_counter<int> n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const value_counter<int> & element)
    {
      if (element < n)
      {
        *(out++) = element;
```

```
        return false;//continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<value_counter<int> >
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const value_counter<int> & n)
    {
      *(out++) = n;
      return true;
    }
};

void test() {
  cout<<"Test 1a\nTesting N\nTiming distribute on integers.\n";
  cout<<"Distribute to all destinations." << endl;
 srand(time(0));
  typedef counters::iterator_counter<vector
   <value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;
  // Initial Setup
  router<value_counter<int> > *r;
  r = new router<value_counter<int> >;
  //vectors of counter_ints
  count_vector v, out1, out2, out3, defaultItems;
  //vectors of counter_ints
  count_vector out1_saved, out2_saved, defaultItems_saved;

  //back_insert_iterator of count_vector
  typedef std::back_insert_iterator<count_vector> BItype;
    delete r;
    r = new router<value_counter<int> >;

  out1.clear();
  out2.clear();
  out3.clear();
  defaultItems.clear();

  less_than_n<BItype> test1(back_inserter(out1), 1048576);
```

```
  less_than_n<BItype> test2(back_inserter(out2), 1048576);
  less_than_n<BItype> test3(back_inserter(out3), 1048576);
  my_default<BItype> defaultTest(back_inserter(defaultItems));

  r->add_destination(test1);
  r->add_destination(test2);
  r->add_destination(test3);
  r->set_default(defaultTest);


  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
//  N2 = 1048576;
  reps = 1;


  //now we need to print the results
  /*this code is from quantiles-count.cpp, by DRM,
  http://www.cs.rpi.edu/~musser/gsd/quantiles/quantiles-count.cpp
  this code sets up the operations counting and reporting*/
  counters::operator_type group0[] = {
    counters::LESS_THAN, counters::EQUALITY,
  };
  counters::operator_type group1[] = {
    counters::DEFAULT_CTOR,
    counters::COPY_CTOR,
    counters::OTHER_CTOR,
    counters::ASSIGNMENT,
  };
  counters::operator_type group2[] = {
    counters::CONVERSION, counters::POST_INCREMENT,
    counters::PRE_INCREMENT,
    counters::POST_DECREMENT, counters::PRE_DECREMENT,
    counters::UNARY_PLUS, counters::UNARY_MINUS,
    counters::PLUS, counters::MINUS, counters::TIMES,
    counters::DIVIDE, counters::MODULO,
    counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
    counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
    counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
    counters::MODULO_ASSIGN,
    counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
    counters::DEREFERENCE, counters::MEMBER,
    counters::SUBSCRIPT, counters::FUNCTION_CALL,
    counters::NEGATE, counters::AND, counters::AND_ASSIGN,
    counters::OR, counters::OR_ASSIGN, counters::XOR,
    counters::XOR_ASSIGN,
  };
```

```
  counters::operator_type group3[] = {
    counters::BASE,
    counters::DTOR,
    counters::GENERATION,
  };
  //  counters::scale_recorder_visitor<
  //    counters::group_recorder_visitor<> > my_visitor(0.001);
  counters::group_recorder_visitor<> my_visitor;
  my_visitor.add("Compare", group0,
              group0 + sizeof(group0) / sizeof(group0[0]));
  my_visitor.add("Assign", group1,
              group1 + sizeof(group1) / sizeof(group1[0]));
  my_visitor.add("Other", group2,
              group2 + sizeof(group2) / sizeof(group2[0]));
  my_visitor.hide(group3,
               group3 + sizeof(group3) / sizeof(group3[0]));

  recorder<> my_recorder;
  my_recorder.clear();

  for (N = N1; N <= N2; N *= 2)
  {
    cout<<N<<endl;
    my_recorder.pause();
    v.clear();
    v.reserve(N);
    for (int i = 0; i < N; ++i)
    {
      v.push_back((value_counter<int>)i);
    }
    out1.clear();
    out2.clear();
    out3.clear();
    defaultItems.clear();
    // End of Baseline Operations
    ////////////////////////////////
    // Main Timed Operation
    my_recorder.resume();
    r->distribute(citer(v.begin()), citer(v.end()));
    my_recorder.record();
    my_recorder.report(cout,my_visitor);
    my_recorder.clear();
  }

}

int main()
{
  test();
  return 0;
```

```
}
```

B.2.2 Test 1b

"opcount_test1b.cpp" 178 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>
#include "timer.h"

#define DEFAULT_COUNTER_TYPE int
#include "counters.h"

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using namespace counters;
using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<value_counter<int> >
{
  private:
    OutputIterator out;
    value_counter<int> n;

  public:
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const value_counter<int> & element)
    {
      if (element > n)
      {
```

```
        *(out++) = element;
        return false;//continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<value_counter<int> >
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const value_counter<int> & n)
    {
      *(out++) = n;
      return true;
    }
};


void test() {
  cout<<"Test 1b\nTesting N\nTiming distribute on integers.\n";
  cout<<"Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  typedef counters::iterator_counter<
   vector<value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;
  // Initial Setup
  router<value_counter<int> > *r;
  r = new router<value_counter<int> >;
  count_vector v;
  //vectors of counter_ints
  count_vector out1, out2, out3, defaultItems;
  //vectors of counter_ints
  count_vector out1_saved, out2_saved, defaultItems_saved;

    typedef std::back_insert_iterator<count_vector> BItype;
    delete r;
    r = new router<value_counter<int> >;

  out1.clear();
  out2.clear();
  out3.clear();
```

```
defaultItems.clear();

less_than_n<BItype> test1(back_inserter(out1), 1048576);
less_than_n<BItype> test2(back_inserter(out2), 1048576);
less_than_n<BItype> test3(back_inserter(out3), 1048576);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->add_destination(test1);
r->add_destination(test2);
r->add_destination(test3);
r->set_default(defaultTest);

//now we need to print the results
/*this code is from quantiles-count.cpp, by DRM,
http://www.cs.rpi.edu/~musser/gsd/quantiles/quantiles-count.cpp
this code sets up the operations counting and reporting*/
counters::operator_type group0[] = {
  counters::LESS_THAN, counters::EQUALITY
};
counters::operator_type group1[] = {
  counters::DEFAULT_CTOR,
  counters::COPY_CTOR,
  counters::OTHER_CTOR,
  counters::ASSIGNMENT,
};
counters::operator_type group2[] = {
  counters::CONVERSION, counters::POST_INCREMENT,
  counters::PRE_INCREMENT,
  counters::POST_DECREMENT, counters::PRE_DECREMENT,
  counters::UNARY_PLUS, counters::UNARY_MINUS,
  counters::PLUS, counters::MINUS, counters::TIMES,
  counters::DIVIDE, counters::MODULO,
  counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
  counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
  counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
  counters::MODULO_ASSIGN,
  counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
  counters::DEREFERENCE, counters::MEMBER,
  counters::SUBSCRIPT, counters::FUNCTION_CALL,
  counters::NEGATE, counters::AND, counters::AND_ASSIGN,
  counters::OR, counters::OR_ASSIGN, counters::XOR,
  counters::XOR_ASSIGN,
};
counters::operator_type group3[] = {
  counters::BASE,
  counters::DTOR,
  counters::GENERATION,
};
//  counters::scale_recorder_visitor<
//    counters::group_recorder_visitor<> > my_visitor(0.001);
```

```
  counters::group_recorder_visitor<> my_visitor;
  my_visitor.add("Compare", group0,
              group0 + sizeof(group0) / sizeof(group0[0]));
  my_visitor.add("Assign", group1,
              group1 + sizeof(group1) / sizeof(group1[0]));
  my_visitor.add("Other", group2,
              group2 + sizeof(group2) / sizeof(group2[0]));
  my_visitor.hide(group3,
               group3 + sizeof(group3) / sizeof(group3[0]));

  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
//  N2 = 1048576;
//  reps = 10;
  reps = 1;
  recorder<> my_recorder;

  for (N = N1; N <= N2; N *= 2)
  {
    my_recorder.pause();
    cout<<N<<endl;
    v.clear();
    v.reserve(N);
    for (int i = 0; i < N; ++i)
    {
      v.push_back((value_counter<int>)i);
    }

    // Baseline Operations Here
    out1.clear();
    out2.clear();
    out3.clear();
    defaultItems.clear();
    // End of Baseline Operations
    ////////////////////////////////
    // Main Timed Operation
    my_recorder.resume();
    r->distribute(citer(v.begin()),citer( v.end()));
    ////////////////////////////////////
    my_recorder.record();
    my_recorder.report(cout,my_visitor);
    my_recorder.clear();
  }
}

int main()
{
```

```
  test();
  return 0;
}
```

B.3 Test 2

B.3.1 Test 2a

"opcount_test2a.cpp" 182 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "timer.h"

#define DEFAULT_COUNTER_TYPE int
#include "counters.h"

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using namespace counters;
using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<value_counter<int> >
{
  private:
    OutputIterator out;
    value_counter<int>  n;

  public:
```

```
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const value_counter<int> & element)
    {
      if (element > n)
      {
        *(out++) = element;
        return false;// continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<value_counter<int> >
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const value_counter<int> & n)
    {
      *(out) = n;
      return true;
    }
};


void test2() {
  cout << "Test 2b\nTesting M\nTiming distribute on integers.\n";
  cout<<"Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  typedef counters::iterator_counter<
   vector<value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;

  // Initial Setup
  router<value_counter<int> > *r;
  r = new router<value_counter<int> >;
  count_vector v, out1, out2, defaultItems;
  count_vector out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<count_vector> BItype;
    delete r;
    r = new router<value_counter<int> >;
```

```
out1.clear();
out2.clear();
defaultItems.clear();

less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
less_than_n<BItype> test20(back_inserter(out2), 20);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->set_default(defaultTest);
v.clear();
v.reserve(10);
//    for (int i = 0; i < 100; ++i) {
for (int i = 0; i < 10; ++i) {
  v.push_back((value_counter<int>)i);
}

//now we need to print the results
/*this code is from quantiles-count.cpp, by DRM,
http://www.cs.rpi.edu/~musser/gsd/quantiles/quantiles-count.cpp
this code sets up the operations counting and reporting*/
counters::operator_type group0[] = {
  counters::LESS_THAN, counters::EQUALITY
};
counters::operator_type group1[] = {
  counters::DEFAULT_CTOR,
  counters::COPY_CTOR,
  counters::OTHER_CTOR,
  counters::ASSIGNMENT,
};
counters::operator_type group2[] = {
  counters::CONVERSION, counters::POST_INCREMENT,
  counters::PRE_INCREMENT,
  counters::POST_DECREMENT, counters::PRE_DECREMENT,
  counters::UNARY_PLUS, counters::UNARY_MINUS,
  counters::PLUS, counters::MINUS, counters::TIMES,
  counters::DIVIDE, counters::MODULO,
  counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
  counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
  counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
  counters::MODULO_ASSIGN,
  counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
  counters::DEREFERENCE, counters::MEMBER,
  counters::SUBSCRIPT, counters::FUNCTION_CALL,
  counters::NEGATE, counters::AND, counters::AND_ASSIGN,
  counters::OR, counters::OR_ASSIGN, counters::XOR,
  counters::XOR_ASSIGN,
};
counters::operator_type group3[] = {
  counters::BASE,
```

```
    counters::DTOR,
    counters::GENERATION,
  };
  //  counters::scale_recorder_visitor<
  //    counters::group_recorder_visitor<> > my_visitor(0.001);
  counters::group_recorder_visitor<> my_visitor;
  my_visitor.add("Compare", group0,
              group0 + sizeof(group0) / sizeof(group0[0]));
  my_visitor.add("Assign", group1,
              group1 + sizeof(group1) / sizeof(group1[0]));
  my_visitor.add("Other", group2,
              group2 + sizeof(group2) / sizeof(group2[0]));
  my_visitor.hide(group3,
               group3 + sizeof(group3) / sizeof(group3[0]));

  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
  reps = 1;
  recorder<> my_recorder;

  for (N = N1; N <= N2; N *= 2)
    {
      my_recorder.pause();
      cout<<N<<endl;

      for (int i = 0; i < (N/2); ++i) {
  r->add_destination(test1048576);
      }


    ////////////////////////////
      // Baseline Operations Here
    out1.clear();
    defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
    my_recorder.resume();
    r->distribute(citer(v.begin()),citer( v.end()));
    ////////////////////////////////////
    my_recorder.record();
    my_recorder.report(cout,my_visitor);
    my_recorder.clear();
  }

}
```

```
int main()
{
  test2();
  return 0;
}
```

B.3.2 Test 2b

"opcount_test2b.cpp" 186 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "timer.h"

#define DEFAULT_COUNTER_TYPE int
#include "counters.h"

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using namespace counters;
using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<value_counter<int> >
{
  private:
    OutputIterator out;
    value_counter<int>  n;

  public:
```

```
    less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

    bool distribute(const value_counter<int> & element)
    {
      if (element > n)
      {
        *(out++) = element;
        return false;// continue filtering
      }

      return false;
    }
};

template <typename OutputIterator>
class my_default : public destination<value_counter<int> >
{
  private:
    OutputIterator out;

  public:
    my_default(OutputIterator _out) : out(_out) {}

    bool distribute(const value_counter<int> & n)
    {
      *(out) = n;
      return true;
    }
};


void test2() {
  cout << "Test 2b\nTesting M\nTiming distribute on integers.\n";
  cout<<"Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  typedef counters::iterator_counter<
   vector<value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;

  // Initial Setup
  router<value_counter<int> > *r;
  r = new router<value_counter<int> >;
  count_vector v, out1, out2, defaultItems;
  count_vector out1_saved, out2_saved, defaultItems_saved;

  typedef std::back_insert_iterator<count_vector> BItype;
    delete r;
    r = new router<value_counter<int> >;
```

```
out1.clear();
out2.clear();
defaultItems.clear();

less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
less_than_n<BItype> test20(back_inserter(out2), 20);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->set_default(defaultTest);
v.clear();
v.reserve(10);
//    for (int i = 0; i < 100; ++i) {
for (int i = 0; i < 10; ++i) {
  v.push_back((value_counter<int>)i);
}

//now we need to print the results
/*this code is from quantiles-count.cpp, by DRM,
http://www.cs.rpi.edu/~musser/gsd/quantiles/quantiles-count.cpp
this code sets up the operations counting and reporting*/
counters::operator_type group0[] = {
  counters::LESS_THAN, counters::EQUALITY
};
counters::operator_type group1[] = {
  counters::DEFAULT_CTOR,
  counters::COPY_CTOR,
  counters::OTHER_CTOR,
  counters::ASSIGNMENT,
};
counters::operator_type group2[] = {
  counters::CONVERSION, counters::POST_INCREMENT,
  counters::PRE_INCREMENT,
  counters::POST_DECREMENT, counters::PRE_DECREMENT,
  counters::UNARY_PLUS, counters::UNARY_MINUS,
  counters::PLUS, counters::MINUS, counters::TIMES,
  counters::DIVIDE, counters::MODULO,
  counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
  counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
  counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
  counters::MODULO_ASSIGN,
  counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
  counters::DEREFERENCE, counters::MEMBER,
  counters::SUBSCRIPT, counters::FUNCTION_CALL,
  counters::NEGATE, counters::AND, counters::AND_ASSIGN,
  counters::OR, counters::OR_ASSIGN, counters::XOR,
  counters::XOR_ASSIGN,
};
counters::operator_type group3[] = {
  counters::BASE,
```

```
    counters::DTOR,
    counters::GENERATION,
  };
  //  counters::scale_recorder_visitor<
  //    counters::group_recorder_visitor<> > my_visitor(0.001);
  counters::group_recorder_visitor<> my_visitor;
  my_visitor.add("Compare", group0,
              group0 + sizeof(group0) / sizeof(group0[0]));
  my_visitor.add("Assign", group1,
              group1 + sizeof(group1) / sizeof(group1[0]));
  my_visitor.add("Other", group2,
              group2 + sizeof(group2) / sizeof(group2[0]));
  my_visitor.hide(group3,
               group3 + sizeof(group3) / sizeof(group3[0]));

  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 1048576;
  reps = 1;
  recorder<> my_recorder;

  for (N = N1; N <= N2; N *= 2)
    {
      my_recorder.pause();
      cout<<N<<endl;

      for (int i = 0; i < (N/2); ++i) {
  r->add_destination(test1048576);
      }


    ////////////////////////////
      // Baseline Operations Here
    out1.clear();
    defaultItems.clear();

    // End of Baseline Operations
    ////////////////////////////
    // Main Timed Operation
    my_recorder.resume();
    r->distribute(citer(v.begin()),citer( v.end()));
    ////////////////////////////////////
    my_recorder.record();
    my_recorder.report(cout,my_visitor);
    my_recorder.clear();
  }

}
```

```
int main()
{
  test2();
  return 0;
}
```

B.4 Test 3

"opcount_test3.cpp" 190 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>
#include "timer.h"


#define DEFAULT_COUNTER_TYPE int
#include "counters.h"

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using namespace counters;
using gdfa::router;
using gdfa::destination;


bool less_than_n(const value_counter<int> & element) {
      if (element < (value_counter<int>)1048576) {
        return true;
      }

      return false;
```

```
}

void test() {
  cout << "Test 3\nSimple Version of Distribute\nTesting N\n";
  cout<<"Timing distribute on integers." << endl;
  srand(time(0));

  typedef counters::iterator_counter<
   vector<value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;

  // Initial Setup
  count_vector v, out1, defaultItems;

  typedef std::back_insert_iterator<count_vector> BItype;

  out1.clear();
  defaultItems.clear();

  //now we need to print the results
  /*this code is from quantiles-count.cpp, by DRM,
  http://www.cs.rpi.edu/~musser/gsd/quantiles/quantiles-count.cpp
  this code sets up the operations counting and reporting*/
  counters::operator_type group0[] = {
    counters::LESS_THAN, counters::EQUALITY
  };
  counters::operator_type group1[] = {
    counters::DEFAULT_CTOR,
    counters::COPY_CTOR,
    counters::OTHER_CTOR,
    counters::ASSIGNMENT,
  };
  counters::operator_type group2[] = {
    counters::CONVERSION, counters::POST_INCREMENT,
    counters::PRE_INCREMENT,
    counters::POST_DECREMENT, counters::PRE_DECREMENT,
    counters::UNARY_PLUS, counters::UNARY_MINUS,
    counters::PLUS, counters::MINUS, counters::TIMES,
    counters::DIVIDE, counters::MODULO,
    counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
    counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
    counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
    counters::MODULO_ASSIGN,
    counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
    counters::DEREFERENCE, counters::MEMBER,
    counters::SUBSCRIPT, counters::FUNCTION_CALL,
    counters::NEGATE, counters::AND, counters::AND_ASSIGN,
    counters::OR, counters::OR_ASSIGN, counters::XOR,
    counters::XOR_ASSIGN,
```

```
};
counters::operator_type group3[] = {
  counters::BASE,
  counters::DTOR,
  counters::GENERATION,
};
//  counters::scale_recorder_visitor<
//    counters::group_recorder_visitor<> > my_visitor(0.001);
counters::group_recorder_visitor<> my_visitor;
my_visitor.add("Compare", group0,
   group0 + sizeof(group0) / sizeof(group0[0]));
my_visitor.add("Assign", group1,
   group1 + sizeof(group1) / sizeof(group1[0]));
my_visitor.add("Other", group2,
   group2 + sizeof(group2) / sizeof(group2[0]));
my_visitor.hide(group3,
    group3 + sizeof(group3) / sizeof(group3[0]));

unsigned long N, N1, N2;
unsigned int reps;

N2 = 1048576;
N1 = 1;
reps = 1;
recorder<> my_recorder;

for (N = N1; N <= N2; N *= 2)
{
  my_recorder.pause();
  cout<<N<<endl;
  v.clear();
  v.reserve(N);
  for (int i = 0; i < N; ++i) {
    v.push_back((value_counter<int>)i);
  }

  ////////////////////////////////
  // Baseline Operations Here
  out1.clear();
  defaultItems.clear();
  // End of Baseline Operations
  ////////////////////////////////
  // Main Timed Operation
  my_recorder.resume();
  gdfa::distribute(citer(v.begin()), citer(v.end()),
   back_inserter(out1), back_inserter(defaultItems),
   less_than_n);
  ////////////////////////////////////
  my_recorder.record();
  my_recorder.report(cout,my_visitor);
```

```
    my_recorder.clear();
  }

}

int main()
{
  test();
  return 0;
}
```

B.5 Test 4

"opcount_test4.cpp" 193 ≡

```
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <algorithm>
#include <iostream>
#include <vector>
#include <list>
#include <iterator>

#include "timer.h"

#define DEFAULT_COUNTER_TYPE int
#include "counters.h"

#include "router.hpp"

using namespace std;
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::back_inserter;
using std::ostream_iterator;
using std::back_insert_iterator;

using namespace counters;
using gdfa::router;
using gdfa::destination;

template <typename OutputIterator>
class less_than_n : public destination<value_counter<int> >
{
```

```
private:
  OutputIterator out;
  value_counter<int>  n;

public:
  less_than_n(OutputIterator _out, int _n) : out(_out), n(_n) {}

  bool distribute(const value_counter<int> & element)
  {
    if (element > n)
      {
        *(out++) = element;
        return true;
      }

    return false;
  }
};

template <typename OutputIterator>
class my_default : public destination<value_counter<int> >
{
private:
  OutputIterator out;

public:
  my_default(OutputIterator _out) : out(_out) {}

  bool distribute(const value_counter<int> & n)
  {
    *(out) = n;
    return true;
  }
};

void test4() {
  cout << "Test 4\nTesting M x N\nTiming distribute on integers.\n";
  cout<<"Distribute to just 1 of the destinations." << endl;
  srand(time(0));

  typedef counters::iterator_counter<
    vector<value_counter<int> >::iterator> citer;
  //typedef the counter vector
  typedef vector<value_counter<int> > count_vector;

  // Initial Setup
  router<value_counter<int> > *r;
  r = new router<value_counter<int> >;
  count_vector v, out1, out2, defaultItems;
```

```
typedef std::back_insert_iterator<count_vector> BItype;
delete r;
r = new router<value_counter<int> >;

out1.clear();
out2.clear();
defaultItems.clear();

less_than_n<BItype> test1048576(back_inserter(out1), 1048576);
my_default<BItype> defaultTest(back_inserter(defaultItems));

r->set_default(defaultTest);

counters::operator_type group0[] = {
  counters::LESS_THAN, counters::EQUALITY
};
counters::operator_type group1[] = {
  counters::DEFAULT_CTOR,
  counters::COPY_CTOR,
  counters::OTHER_CTOR,
  counters::ASSIGNMENT,
};
counters::operator_type group2[] = {
  counters::CONVERSION, counters::POST_INCREMENT,
  counters::PRE_INCREMENT,
  counters::POST_DECREMENT, counters::PRE_DECREMENT,
  counters::UNARY_PLUS, counters::UNARY_MINUS,
  counters::PLUS, counters::MINUS, counters::TIMES,
  counters::DIVIDE, counters::MODULO,
  counters::LEFT_SHIFT, counters::RIGHT_SHIFT,
  counters::PLUS_ASSIGN, counters::MINUS_ASSIGN,
  counters::TIMES_ASSIGN, counters::DIVIDE_ASSIGN,
  counters::MODULO_ASSIGN,
  counters::LEFT_SHIFT_ASSIGN, counters::RIGHT_SHIFT_ASSIGN,
  counters::DEREFERENCE, counters::MEMBER,
  counters::SUBSCRIPT, counters::FUNCTION_CALL,
  counters::NEGATE, counters::AND, counters::AND_ASSIGN,
  counters::OR, counters::OR_ASSIGN, counters::XOR,
  counters::XOR_ASSIGN,
};
counters::operator_type group3[] = {
  counters::BASE,
  counters::DTOR,
  counters::GENERATION,
};

counters::group_recorder_visitor<> my_visitor;
my_visitor.add("Compare", group0,
               group0 + sizeof(group0) / sizeof(group0[0]));
my_visitor.add("Assign", group1,
```

```
                group1 + sizeof(group1) / sizeof(group1[0]));
  my_visitor.add("Other", group2,
                group2 + sizeof(group2) / sizeof(group2[0]));
  my_visitor.hide(group3,
                 group3 + sizeof(group3) / sizeof(group3[0]));

  unsigned long N, N1, N2;
  unsigned int reps;

  N1 = 1;
  N2 = 16384;
  reps = 1;
  recorder<> my_recorder;

  for (N = N1; N <= N2; N *= 2)
    {
      my_recorder.pause();
      cout<<"\nM = "<<N<<endl
          <<"**************************************\n"
          <<"**************************************\n";

      for (int i = 0; i < (N/2); ++i) {
        r->add_destination(test1048576);
      }

      for (int realN = N1; realN <= N2; realN *= 2)
        {
          my_recorder.pause();
          cout<<"N = "<<realN<<endl
              <<"**************************************\n";

          v.clear();
          v.reserve(realN);
          for (int i = 0; i < realN; ++i)
            {
              v.push_back((value_counter<int>)i);
            }
          out1.clear();
          defaultItems.clear();


          my_recorder.resume();
          r->distribute(citer(v.begin()),citer( v.end()));
          my_recorder.record();
          my_recorder.report(cout,my_visitor);
          my_recorder.clear();
        }
    }
}
```

```
int main()
{
  test4();
  return 0;
}
```

Appendix C

Code for Correctness Tests

C.1 Common Validation Methods

"validate.hpp" 198 ≡

```
//validate.h version 1.0
#ifndef _VALIDATE_H
#define _VALIDATE_H

#include <vector>
#include <iostream>
#include <list>
#include <deque>
using std::list;
using std::deque;
using std::vector;
using std::cout;
using std::endl;

⟨first validate function 199⟩
⟨second validate function 200a⟩

#endif
```

C.1.1 First Validate Function

This validate function iterates through the results of filtering the same data but using either different priorities, storage types, etc. Because of the nature of the input, the results should always be the same.

⟨first validate function 199⟩ ≡

```
template<typename conttype1,typename conttype2>
bool validate(conttype1 key[],conttype2 res[],int size,
              bool rev = false)
{
  typedef typename conttype1::iterator key_iter;
  key_iter ikey;
  typedef typename conttype2::iterator res_iter;
  res_iter ires;
  for(int i = 0;i < size;i++)
  {
    if(rev)
      reverse(res[i].begin(),res[i].end());

    ikey = key[i].begin();
    ires = res[i].begin();
    while(ikey != key[i].end() && ires != res[i].end())
    {
      if(*ikey != *ires)
      {
        cout<<"In container "<<i<<" "<<*ires<<" != "<<*ikey<<endl;
        return false;
      }
      ikey++;
      ires++;
    }
    if(ikey!=key[i].end())
    {
      cout<<"Not enough elements in container "<<i<<endl;
      return false;
    }
    if(ires!=res[i].end())
    {
      cout<<"Too many elements in container "<<i<<endl;
      return false;
    }
  }
  return true;
}
```

Used in part 198.

C.1.2 Second Validate Function

⟨second validate function 200a⟩ ≡

```
template<typename conttype1,typename conttype2>
bool validatesize(conttype1 key[],conttype2 res[],int size)
{
  typedef typename conttype1::iterator key_iter;
  key_iter ikey;
  typedef typename conttype2::iterator res_iter;
  res_iter ires;
  for(int i = 0;i < size;i++)
  {
    if(key[i].size() != res[i].size())
    {
      cout<<"match failed for container: "<<i<<endl;
      return false;
    }
  }
  return true;
}
```

Used in part 198.

C.2 Helper Classes

C.2.1 Test Classes

"test_classes.hpp" 200b ≡

```
//test_classes.h version 1.3
#ifndef TEST_CLASSES_H
#define TEST_CLASSES_H

#include <vector>
#include <iostream>
#include "router.hpp"
#include <ctype.h>
#include <map>
using std::pair;
using std::string;
using std::vector;
using std::cout;
using std::endl;
using gdfa::router;
using gdfa::destination;

/* word_sorter sorts strings based on their first letter.  if
the first letter of the string matches m_first_letter the class
will write the string its output_iterator. this class is case
insensitive meaning that if m_first_letter is 'a' words
beginning with 'A' or 'a' will be filtered.  Also note that when
an object of this class is first created that it will convert
```

```
the char given to a lower case letter so that it operates
correctly. */
template<typename output_iterator>
class word_sorter : public destination<string>
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sorter(output_iterator out, char first_letter)
         : m_out(out) { m_first_letter =
                       tolower((const int)first_letter);}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
  {
     if (tolower((const int)element[0]) == m_first_letter)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

/*length_counter filters on the length of the input string.  if the
string is of length m_length the object will write the string to
its output_iterator*/
template<typename output_iterator>
class length_counter : public destination<string>
{
   private:
  output_iterator m_out;
  int m_length;

   public:
  length_counter(output_iterator out, int length)
         : m_out(out), m_length(length) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
  {
     if(element.size() == m_length)
     {
    *m_out++ = element;
    return true;
```

```
    }

    return false;
  }

};

/*contains letter checks to see is the input string contains the
char m_letter. if it does the object will write the string to
its output_iterator.  this letter is case sensitive, meaning
that if a lowercase 'a' is given that words with 'A' will not be
filtered unless they also have an 'a'*/
template<typename output_iterator>
class contains_letter : public destination<string>
{
private:
        output_iterator m_out;
        char m_letter;
public:
        contains_letter(output_iterator out,const char letter)
                        : m_out(out), m_letter(letter) {}
        bool distribute(const string &n)
        {
                if(n.find(m_letter) < n.size())
                {
                        *m_out++ = n;
                        return true;
                }
                return false;
        }
};

/*the default class is used to catch anything that is not
filtered*/
template<typename output_iterator>
class my_default : public destination<string>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const string& n)
  {
     *m_out++ = n;
     return true;
  }
};
```

```
/*a specialization of word_sorter that works with pairs. needed
for filtering from a map*/
template<typename output_iterator>
class word_sortermap : public destination<pair<string,int> >
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sortermap(output_iterator out, char first_letter)
         : m_out(out) { m_first_letter =
                       tolower((const int)first_letter);}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const pair<string,int>& element)
  {
     if (tolower((const int)element.first[0]) == m_first_letter)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

/*contains string works much like contains letter, except that it
checks to see if the input contains a substring*/
template<typename output_iterator>
class contains_string : public destination<string>
{
private:
        output_iterator m_out;
        string m_str;
public:
        contains_string(output_iterator out,string str)
                        : m_out(out), m_str(str) {}
        bool distribute(const string &n)
        {
                if(n.find(m_str) < n.size())
                {
                        *m_out++ = n;
                        return true;
                }
                return false;
        }
};
```

```
/*a specialization of my_default that works with pairs.  needed for
filtering from a map*/
template<typename output_iterator>
class my_defaultmap : public destination<pair<string,int> >
{
   private:
  output_iterator m_out;
   public:
  my_defaultmap(output_iterator out)
         : m_out(out) {}

  bool distribute(const pair<string,int>& n)
  {
     *m_out++ = n;
     return true;
  }
};

void createinputstring(vector<string> &storage)
{
  for(int i = 0;i<27;i++)
  {
    for(int j = 0;j < 27;j++)
    {
      for(int k = 0;k<27;k++)
      {
        string foo;
        foo += (char)(i + 97);
        storage.push_back(foo);
        foo += (char)(j + 97);
        storage.push_back(foo);
        foo += (char)(k + 97);
        storage.push_back(foo);
      }
    }
  }
  random_shuffle(storage.begin(),storage.end());
}

#endif
```

C.2.2 Test Classes

"test_classes_fallthrough.hpp" 204 ≡

```
//test_classes_fallthrough.h version 1.2
//descriptions of the classes appear before their declaration
/* all classes in this file do fall through, meaning that it always
tells the router to keep attempting to filter the input*/
```

```
#ifndef TEST_CLASSES_H
#define TEST_CLASSES_H

#include <vector>
#include <iostream>
#include "router.hpp"
#include <ctype.h>
using std::string;
using std::vector;
using std::cout;
using std::endl;
using gdfa::router;
using gdfa::destination;

/* word_sorter sorts strings based on their first letter.  if
the first letter of the string matches m_first_letter the class
will write the string its output_iterator. this class is case
insensitive meaning that if m_first_letter is 'a' words
beginning with 'A' or 'a' will be filtered.  Also note that when
an object of this class is first created that it will convert
the char given to a lower case letter so that it operates
correctly. */
template<typename output_iterator>
class word_sorter : public destination<string>
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sorter(output_iterator out, char first_letter)
  :m_out(out)
  {
    m_first_letter = tolower((const int)first_letter);
  }

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
  {
     if (tolower((const int)element[0]) == m_first_letter)
     {
    *m_out++ = element;
     }

     return false;
  }
};

/*length_counter filters on the length of the input string.  if the
```

```
string is of length m_length the object will write the string to
its output_iterator*/
template<typename output_iterator>
class length_counter : public destination<string>
{
   private:
  output_iterator m_out;
  int m_length;

   public:
  length_counter(output_iterator out, int length)
         : m_out(out), m_length(length) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
  {
     if(element.size() == m_length)
     {
    *m_out++ = element;
     }

     return false;
  }

};

/*contains letter checks to see is the input string contains the
char m_letter. if it does the object will write the string to
its output_iterator.  this letter is case sensitive, meaning
that if a lowercase 'a' is given that words with 'A' will not be
filtered unless they also have an 'a'*/
template<typename output_iterator>
class contains_letter : public destination<string>
{
private:
        output_iterator m_out;
        char m_letter;
public:
        contains_letter(output_iterator out,const char letter)
                        : m_out(out), m_letter(letter) {}
        bool distribute(const string &n)
        {
                if(n.find(m_letter) < n.size())
                {
                        *m_out++ = n;
                }
                return false;
        }
};
```

```
/*contains string works much like contains letter, except that it
checks to see if the input contains a substring*/
template<typename output_iterator>
class contains_string : public destination<string>
{
private:
        output_iterator m_out;
        string m_str;
public:
        contains_string(output_iterator out,string str)
                        : m_out(out), m_str(str) {}
        bool distribute(const string &n)
        {
                if(n.find(m_str) < n.size())
                {
                        *m_out++ = n;
                }
                return false;
        }
};


/*the default class is used to catch anything that is not
filtered*/
template<typename output_iterator>
class my_default : public destination<string>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const string& n)
  {
     *m_out++ = n;
     return true;
  }
};


void createinputstring(vector<string> &storage)
{
  for(int i = 0;i<27;i++)
  {
    for(int j = 0;j < 27;j++)
    {
      for(int k = 0;k<27;k++)
      {
```

```
        string foo;
        foo += (char)(i + 97);
        storage.push_back(foo);
        foo += (char)(j + 97);
        storage.push_back(foo);
        foo += (char)(k + 97);
        storage.push_back(foo);
      }
    }
  }
  random_shuffle(storage.begin(),storage.end());
}

#endif
```

C.3 Tests On Built-In Types

C.3.1 Testing with chars

"test_char.cpp" 208 ≡

```
//test_char.cpp version 1.0
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

void createinput(vector<char> &storage);
bool validate(vector<char> sorted[]);

template<typename output_iterator>
class word_sorter : public destination<char>
{
   private:
  output_iterator m_out;
  char m_first_letter;
```

```
   public:
  word_sorter(output_iterator out, char first_letter)
         : m_out(out), m_first_letter(first_letter) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const char& element)
  {
     if (tolower((const int)element) == m_first_letter)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<char>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const char& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  test_char.cpp"
    << "\n\nTest Using type char as input.\n"
    << "*********************************\n\n"
    << "Results:\n\n";

  router<char> r;
  vector<char> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<char> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<char> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
```

```
  }
  my_default<back_insert_iterator<vector<char> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);

  vector<char> storage;
  createinput(storage);

  r.distribute(storage.begin(),storage.end());

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[26].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

void createinput(vector<char> &storage)
{
  for(int i = 0;i<27;i++)
    for(int j = 0;j<1000;j++)
      storage.push_back((char)(i+97));
  random_shuffle(storage.begin(),storage.end());
}

bool validate(vector<char> sorted[])
{
  vector<char>::iterator iter;
  for(int i = 0;i<27;i++)
  {
    iter = sorted[i].begin();
    for(;iter != sorted[i].end();iter++)
    {
      if(*iter != (char)(i + 97))
      {
        cout<<"Wrong letter: "<<*iter<<" should be ";
        cout<<(char)(i + 97)<<" "<<i<<endl;
        return false;
        break;
      }
    }
```

```
  }
  return true;
}
```

C.3.2 Testing with doubles

"test_double.cpp" 211 ≡

```
//test_double.cpp version 1.0
#include <iostream>
#include <fstream>
#include <ctype.h>
#include <algorithm>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::cin;
using std::min_element;
using std::max_element;
using gdfa::router;
using gdfa::destination;

void createinput(vector<double> &storage);
bool validate(vector<double> sorted[]);

class my_compare
{
public:
  my_compare() {}
  static const double epsilon = 1e-10;
  bool less(const double &lhs,const double &rhs)
  {
    if((rhs - lhs) < epsilon) return false;
    if(lhs<rhs) return true;
    return false;
  }
  bool greater(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon) return false;
    if(lhs>rhs) return true;
    return false;
  }
  bool lessequal(const double &lhs,const double &rhs)
  {
    if((rhs - lhs) < epsilon && (rhs - lhs) > -epsilon)
     return true;
```

```
    if(lhs<rhs) return true;
    return false;
  }
  bool greaterequal(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon && (lhs - rhs) > -epsilon)
     return true;
    if(lhs>rhs) return true;
    return false;
  }
  bool equal(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon && (rhs - lhs) < -epsilon)
     return true;
    if((rhs -lhs) < epsilon && (lhs - rhs) < -epsilon)
     return true;
    return false;
  }

};

my_compare comp;

template<typename output_iterator>
class range_checker : public destination<double>
{
   private:
      output_iterator m_out;
      double m_lower;
      double m_upper;
//      static my_compare comp;

   public:
      range_checker(output_iterator out, double lower,double upper)
                : m_out(out), m_lower(lower), m_upper(upper) {}

      /*out distribute function checks the first letter to see if
      it is the same as in the predicate*/
      bool distribute(const double& element)
      {
        if(comp.greater(element,m_lower) &&
           comp.less(element,m_upper))
        {
            if( element < .3000000001 && element > .29999999)
            {
              cout<<"dumber than dirt "<<element<<endl;
              cout<<element - m_lower<<endl;
/*              if((element - m_lower) > epsilon)
              cout<<"greater"<<endl;
              if((m_lower - element) > epsilon)
```

```
                cout<<"less than"<<endl;*/
            }
            *m_out++ = element;
            return true;
         }

         return false;
      }
};

template<typename output_iterator>
class my_default : public destination<double>
{
   private:
      output_iterator m_out;
   public:
      my_default(output_iterator out)
                 : m_out(out) {}

      bool distribute(const double& n)
      {
         *m_out++ = n;
         return true;
      }
};

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  test_double.cpp"
    << "\n\nTest Using type double as input.\n"
    << "***********************************\n\n"
    << "Results:\n\n";

  router<double> r;
  vector<double> sorted[11];//one for each letter, one for default
  range_checker<back_insert_iterator<vector<double> > > *dummy[10];
  double lower = 0.0;
  double upper = 0.1;
  for(int i = 0; i<10;i++)
  {
    dummy[i] =
     new range_checker<back_insert_iterator<vector<double> > >
     (back_inserter(sorted[i]), lower, upper);
    r.add_destination(*dummy[i]);
    lower+= 0.1;
    upper+= 0.1;
  }
  my_default<back_insert_iterator<vector<double> > >
   dflt(back_inserter(sorted[10]));
  r.set_default(dflt);
```

```
  vector<double> storage;
  createinput(storage);

  r.distribute(storage.begin(),storage.end());
  cout<<"number of elements in each range, with the max and min ";
  cout<<"element in the ranges"<<endl;
  double j;
  int i;
  for(i = 0,j = 0.0;i<10;i++,j+=.1)
  {
    cout<<"("<<j<<","<<j+.1<<"): "<<sorted[i].size()<<" ";
    cout<<*max_element(sorted[i].begin(),sorted[i].end())
      <<" "<<*min_element(sorted[i].begin(),sorted[i].end())<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[10].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}


/*the key to this test is that we will generate the information
to be filtered thus we can test if the filtering works correctly
because we know what the solution is*/
void createinput(vector<double> &storage)
{
  double max = 1.0;
  double increment = .000001;
  double i;
  for(i=0.0;comp.less(i,max);i+=increment)
  {
    storage.push_back(i);
  }
  random_shuffle(storage.begin(),storage.end());
}

bool validate(vector<double> sorted[])
{
  int i;
  double j;
  for(i = 0,j = 0.0;i<10;i++,j+=.1)
  {
    double min = *min_element(sorted[i].begin(),sorted[i].end());
```

```
    if(comp.greaterequal(j,min))
    {
      cout<<"Failed, "<<min<<" is not in range ";
      cout<<"("<<j<<","<<j+.1<<")"<<endl;
      return false;
    }
    double max = *max_element(sorted[i].begin(),sorted[i].end());
    if(comp.lessequal(j+.1,max))
    {
      cout<<"Failed, "<<max<<" is not in range ";
      cout<<"("<<j<<","<<j+.1<<")"<<endl;
      return false;
    }
  }
  return true;
}
```

C.3.3 Testing with ints

"test_int.cpp" 215 ≡

```
//test_int.cpp version 1.0
#include <iostream>
#include <fstream>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::cin;

using gdfa::router;
using gdfa::destination;

void createinput(vector<int> &storage);
bool validate(vector<int> sorted[]);

template<typename output_iterator>
class range_checker : public destination<int>
{
   private:
  output_iterator m_out;
  int m_lower;
      int m_upper;

   public:
  range_checker(output_iterator out, int lower,int upper)
```

```
        : m_out(out), m_lower(lower), m_upper(upper) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const int& element)
  {
     if(m_lower < element && m_upper > element)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<int>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const int& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
{
    cout << "\n\n\nTest:  test_int.cpp"
    << "\n\nTest Using integers as input.\n"
    << "**********************************\n\n"
    << "Results:\n\n";

  router<int> r;
  vector<int> sorted[11];//one for each letter, one for default
  range_checker<back_insert_iterator<vector<int> > > *dummy[10];
  int lower = 0;
  int upper = 100000;
  for(int i = 0; i<10;i++)
  {
    dummy[i] =
     new range_checker<back_insert_iterator<vector<int> > >
     (back_inserter(sorted[i]), lower, upper);
    r.add_destination(*dummy[i]);
```

```
    lower+= 100000;
    upper+= 100000;
  }
  my_default<back_insert_iterator<vector<int> > >
   dflt(back_inserter(sorted[10]));
  r.set_default(dflt);
  vector<int> storage;
  createinput(storage);

  r.distribute(storage.begin(),storage.end());
  cout<<"number of elements in each range, with the max and min ";
  cout<<"element in the ranges"<<endl;
  for(int i = 0;i<10;i++)
  {
    cout<<"("<<i * 100000<<","<<(i+1)*100000<<"): ";
    cout<<sorted[i].size()<<" ";
    cout<<*max_element(sorted[i].begin(),sorted[i].end())
      <<" "<<*min_element(sorted[i].begin(),sorted[i].end())<<endl;
  }
  cout<<"Number elements caught by default case: ";
  cout<<sorted[10].size()<<endl;

  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}
/*the key to this test is that we will generate the information
to be filtered thus we can test if the filtering works correctly
because we know what the solution is*/
void createinput(vector<int> &storage)
{
  int max = 1000000;
  for(int i=0;i<max;i++)
    storage.push_back(i);
  random_shuffle(storage.begin(),storage.end());
}

bool validate(vector<int> sorted[])
{
  int i;
  int j = 0;
  for(i = 0,j=0;i<10;i++,j+=100000)
  {
    int min = *min_element(sorted[i].begin(),sorted[i].end());
    if(j>=min)
```

```
    {
      cout<<"Failed, "<<min<<" is not in range ";
      cout<<"("<<j<<","<<j+100000<<")"<<endl;
      return false;
    }
    double max = *max_element(sorted[i].begin(),sorted[i].end());
    if(j+100000 <= max)
    {
      cout<<"Failed, "<<max<<" is not in range ";
      cout<<"("<<j<<","<<j+100000<<")"<<endl;
      return false;
    }
  }
  return true;
}
```

C.3.4 Testing with `std::strings`

"test_string.cpp" 218 ≡

```
//test_string.cpp version 1.0
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

void createinput(vector<string> &storage);
bool validate(vector<string> sorted[]);

template<typename output_iterator>
class word_sorter : public destination<string>
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sorter(output_iterator out, char first_letter)
```

```
        : m_out(out), m_first_letter(first_letter) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
  {
     if (tolower((const int)element[0]) == m_first_letter)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<string>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const string& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
{
    cout << "\n\n\nTest:  test_string.cpp"
    << "\n\nTest Using strings as input.\n"
    << "*********************************\n\n"
    << "Results:\n\n";

  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
```

```
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);

  vector<string> storage;
  createinput(storage);
  r.distribute(storage.begin(),storage.end());

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[26].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

void createinput(vector<string> &storage)
{
  for(int i = 0;i<27;i++)
  {
    for(int j = 0;j < 27;j++)
    {
      for(int k = 0;k<27;k++)
      {
        string foo;
        foo += (char)(i + 97);
        foo += (char)(j + 97);
        foo += (char)(k + 97);
        storage.push_back(foo);
      }
    }
  }
  random_shuffle(storage.begin(),storage.end());
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  for(int i = 0;i<27;i++)
  {
    iter = sorted[i].begin();
    for(;iter != sorted[i].end();iter++)
    {
```

```
      if((*iter)[0] != (char)(i + 97))
      {
        cout<<"Wrong letter: "<<*iter<<" should be ";
        cout<<(char)(i + 97)<<" "<<i<<endl;
        return false;
        break;
      }
    }
  }
  return true;
}
```

C.4 Priority Testing

C.4.1 2-Priority Tests

2-Priority with Fall-Through

"priority2_fallthrough.cpp" 221 ≡

```
//priority2_fallthrough.cpp version 1.1
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
#include "test_classes_fallthrough.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);

template<typename InputIterator>
bool testpriority(router<string>::priority_level p1,
 router<string>::priority_level p2,
    InputIterator begin,InputIterator end)
{
  router<string> r;//router
  vector<string> sorted[29];//one for each letter, one for default
   //2 for length filtering
```

```
  word_sorter<back_insert_iterator<vector<string> > >
   *dummy[26];//general sorters

  length_counter<back_insert_iterator<vector<string> > >
   length1(back_inserter(sorted[0]),1);//sorts by length
  length_counter<back_insert_iterator<vector<string> > >
   length2(back_inserter(sorted[1]),2);

  r.add_destination(length1,p1);
  r.add_destination(length2,p1);
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i+2]),(char)(i + 97));
    r.add_destination(*dummy[i],p2);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[28]));
  r.set_default(dflt);

  r.distribute(begin,end);
  return validate(sorted);
}

/*with this test we are testing the back inserter for output and
istream_iterator<string> for input*/

/*for testing from standard in we will sort words by their first
letter we would like to be able to count the number of words that
begin with each letter*/

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority2_fallthrough.cpp\n\n"
  <<"Test router with 2 priorities (allowing fall through)\n"
  <<"****************************"
  <<"****************************\n\n"
  <<"Note: Fall through means that a destination\n"
  <<"passes the element on to the next destination.\n\n"
  <<"*********************************************\n\n"
  <<"Results:\n\n";

  router<string> r;//router
  vector<string> sorted[29];//one for each letter, one for default
   //2 for length filtering
  word_sorter<back_insert_iterator<vector<string> > >
   *dummy[26];//general sorters

  length_counter<back_insert_iterator<vector<string> > >
```

```
 length1(back_inserter(sorted[0]),1);//sorts by length
length_counter<back_insert_iterator<vector<string> > >
 length2(back_inserter(sorted[1]),2);

r.add_destination(length1,router<string>::urgent_priority);
r.add_destination(length2,router<string>::urgent_priority);
for(int i = 0; i<26;i++)
{
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+2]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::high_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[28]));
r.set_default(dflt);

//we want to create our own input for validation purposes
vector<string> storage_for_later;
createinputstring(storage_for_later);

//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());

cout<<"results of testing urgent priority and high priority";
cout<<endl;
cout<<"words of length 1: "<<sorted[0].size()<<endl;
cout<<"words of length 2: "<<sorted[1].size()<<endl;
cout<<"counts for all other words, based on starting ";
cout<<"character (case insensitive)"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 2].size()<<endl;
}
cout<<"Default: "<<sorted[28].size()<<endl;
if(validate(sorted))
  cout<<"Passed test for urgent and high priorities"<<endl;
else
  cout<<"Failed test for urgent and high priorities"<<endl;



cout<<"Testing Urgent and normal priority"<<endl;
if(testpriority(router<string>::urgent_priority,
   router<string>::normal_priority,
   storage_for_later.begin(),storage_for_later.end()))
  cout<<"Passed test for urgent and normal priority"<<endl;
else
  cout<<"Failed test for urgent and normal priority"<<endl;
```

```
  cout<<"Testing Urgent and low priority"<<endl;
  if(testpriority(router<string>::urgent_priority,
     router<string>::low_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for urgent and low priority"<<endl;
  else
    cout<<"Failed test for urgent and low priority"<<endl;

  cout<<"Testing high and normal priority"<<endl;
  if(testpriority(router<string>::high_priority,
     router<string>::normal_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for high and normal priority"<<endl;
  else
    cout<<"Failed test for high and normal priority"<<endl;

  cout<<"Testing high and low priority"<<endl;
  if(testpriority(router<string>::high_priority,
     router<string>::low_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for high and low priority"<<endl;
  else
    cout<<"Failed test for high and low priority"<<endl;

  cout<<"Testing normal and low priority"<<endl;
  if(testpriority(router<string>::normal_priority,
     router<string>::low_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for normal and low priority"<<endl;
  else
    cout<<"Failed test for normal and low priority"<<endl;


  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 19683)
  {
    cout<<"Length 1 failed"<<endl;
    return false;
  }
  else
  {
    //check every string to make sure it is of length 1
    for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
```

```
  {
    if((*iter).size() != 1)
    {
      cout<<"String in length 1 container contains a string of ";
      cout<<"not length 1"<<endl;
      return false;
    }
  }
}
if(sorted[1].size() != 19683)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
      cout<<"not length 2"<<endl;
      return false;
    }
  }
}
for(int i = 2;i<28;i++)
{
  if(sorted[i].size() != 2187)
  {
    cout<<"Begins letter """<<(char)(i+95)<<""" failed"<<endl;
    return false;
  }
  else
  {
    /*check each string to make sure it starts with the correct
    letter*/
    for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
    {
      if((*iter)[0] != (char)(i+95))
      {
        cout<<"String in begins with "<<(char)(i+95);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
    }
  }
```

```
  }
  if(sorted[28].size() != 59049)
  {
    cout<<"Default case failed"<<endl;
    return false;
  }
  return true;
}
```

2-Priority without Fall-Through

"priority2_nofallthrough.cpp" 226 ≡

```
//priority2_nofallthrough.cpp version 1.1
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
#include "test_classes.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);

template<typename InputIterator>
bool testpriority(router<string>::priority_level p1,
 router<string>::priority_level p2,
 InputIterator begin,InputIterator end)
{
  router<string> r;//router
  vector<string> sorted[29];//one for each letter, one for default,
   //2 for length filtering
  word_sorter<back_insert_iterator<vector<string> > >
   *dummy[26];//general sorters

  length_counter<back_insert_iterator<vector<string> > >
   length1(back_inserter(sorted[0]),1);//sorts by length
  length_counter<back_insert_iterator<vector<string> > >
   length2(back_inserter(sorted[1]),2);
```

```
  r.add_destination(length1,p1);
  r.add_destination(length2,p1);
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i+2]),(char)(i + 97));
    r.add_destination(*dummy[i],p2);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[28]));
  r.set_default(dflt);

  r.distribute(begin,end);
  return validate(sorted);
}

/*with this test we are testing the back inserter for output and
istream_iterator<string> for input*/

/*for testing from standard in we will sort words by their first
letter we would like to be able to count the number of words that
begin with each letter*/

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority2_nofallthrough.cpp\n\n"
    << "Test router with 2 priorities (No fall through)\n"
    << "***********************************************\n\n"
    << "(Note: Fall through means that a destination\n"
    << "passes the element on to the next destination.\n\n"
    << "***********************************************\n\n"
    << "Results:\n\n";

  router<string> r;//router
  vector<string> sorted[29];//one for each letter, one for default,
   //2 for length filtering
  word_sorter<back_insert_iterator<vector<string> > >
   *dummy[26];//general sorters

  length_counter<back_insert_iterator<vector<string> > >
   length1(back_inserter(sorted[0]),1);//sorts by length
  length_counter<back_insert_iterator<vector<string> > >
   length2(back_inserter(sorted[1]),2);

  r.add_destination(length1,router<string>::urgent_priority);
  r.add_destination(length2,router<string>::urgent_priority);
  for(int i = 0; i<26;i++)
  {
```

```
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+2]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::high_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[28]));
r.set_default(dflt);

//we want to create our own input for validation purposes
vector<string> storage_for_later;
createinputstring(storage_for_later);

//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());

cout<<"results of testing urgent priority and high priority";
cout<<endl;
cout<<"words of length 1: "<<sorted[0].size()<<endl;
cout<<"words of length 2: "<<sorted[1].size()<<endl;
cout<<"counts for all other words, based on starting character ";
cout<<"(case insensitive)"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 2].size()<<endl;
}
cout<<"Default: "<<sorted[28].size()<<endl;
if(validate(sorted))
  cout<<"Passed test for urgent and high priorities"<<endl;
else
  cout<<"Failed test for urgent and high priorities"<<endl;


cout<<"Testing Urgent and normal priority"<<endl;
if(testpriority(router<string>::urgent_priority,
   router<string>::normal_priority,
   storage_for_later.begin(),storage_for_later.end()))
  cout<<"Passed test for urgent and normal priority"<<endl;
else
  cout<<"Failed test for urgent and normal priority"<<endl;

cout<<"Testing Urgent and low priority"<<endl;
if(testpriority(router<string>::urgent_priority,
   router<string>::low_priority,
   storage_for_later.begin(),storage_for_later.end()))
  cout<<"Passed test for urgent and low priority"<<endl;
else
  cout<<"Failed test for urgent and low priority"<<endl;

cout<<"Testing high and normal priority"<<endl;
```

```
  if(testpriority(router<string>::high_priority,
     router<string>::normal_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for high and normal priority"<<endl;
  else
    cout<<"Failed test for high and normal priority"<<endl;

  cout<<"Testing high and low priority"<<endl;
  if(testpriority(router<string>::high_priority,
     router<string>::low_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for high and low priority"<<endl;
  else
    cout<<"Failed test for high and low priority"<<endl;

  cout<<"Testing normal and low priority"<<endl;
  if(testpriority(router<string>::normal_priority,
     router<string>::low_priority,
     storage_for_later.begin(),storage_for_later.end()))
    cout<<"Passed test for normal and low priority"<<endl;
  else
    cout<<"Failed test for normal and low priority"<<endl;


  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 19683)
  {
    cout<<"Length 1 failed"<<endl;
    return false;
  }
  else
  {
    //check every string to make sure it is of length 1
    for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
    {
      if((*iter).size() != 1)
      {
        cout<<"String in length 1 container contains a string of ";
        cout<<"not length 1"<<endl;
        return false;
      }
    }
  }
```

```
if(sorted[1].size() != 19683)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
      cout<<"not length 2"<<endl;
      return false;
    }
  }
}
for(int i = 2;i<28;i++)
{
  if(sorted[i].size() != 729)
  {
    cout<<"Begins letter """<<(char)(i+95)<<""" failed"<<endl;
    return false;
  }
  else
  {
    /*check each string to make sure it starts with the correct
    letter*/
    for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
    {
      if((*iter)[0] != (char)(i+95))
      {
        cout<<"String in begins with "<<(char)(i+95);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
      /*also need to check that the string length of the
      string is greater than 2*/
      if((*iter).size() < 3)
      {
        cout<<"Begins letter """<<(char)(i+94)<<""" contains a ";
        cout<<"string of length "<<(*iter).size()<<endl;
        return false;
      }
    }
  }
}
```

```
  if(sorted[28].size() != 729)
  {
    cout<<"Default case failed"<<endl;
    return false;
  }
  else
  {/*check to make sure that every string here begins with a "{"
  and that it does not contain an 'a' and is of length greater
  than 2*/
    for(iter = sorted[28].begin();iter != sorted[28].end();iter++)
    {
      if((*iter)[0] != (char)(123))
      {
        cout<<"String in begins with "<<(char)(123);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
      /*also need to check that the string length of the string is
      greater than 2*/
      if((*iter).size() < 3)
      {
        cout<<"Begins letter """<<(char)(123)<<""" contains a ";
        cout<<"string of length "<<(*iter).size()<<endl;
        return false;
      }
    }
  }
  return true;
}
```

C.4.2 3-Priority Tests

3-Priority with Fall-Through

"priority3_fallthrough.cpp" 231 ≡

```
//priority3_fallthrough.cpp version 1.2
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "test_classes_fallthrough.hpp"
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
```

```
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);

typedef router<string>::priority_level plevel;

template<typename InputIterator>
bool testpriority(plevel p1,plevel p2,
    plevel p3,InputIterator begin,InputIterator end)
{
  router<string> r;
  vector<string> sorted[30];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  contains_letter<back_insert_iterator<vector<string> > >
   contains_a(back_inserter(sorted[0]),'a');
  r.add_destination(contains_a,p1);
  length_counter<back_insert_iterator<vector<string> > >
   length1(back_inserter(sorted[1]),1);
  length_counter<back_insert_iterator<vector<string> > >
   length2(back_inserter(sorted[2]),2);
  r.add_destination(length1,p2);
  r.add_destination(length2,p2);
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i+3]),(char)(i + 97));
    r.add_destination(*dummy[i],p3);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[29]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(sorted);
}
int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority3_fallthrough.cpp\n\n"
  <<"Test router with 3 priorities (allowing fall through)\n"
  <<"***********************************************************"
  <<"\n\n"
  <<"Note: Fall through means that a destination\n"
  <<"passes the element on to the next destination.\n\n"
  <<"*********************************************\n\n"
```

```
<<"Results:\n\n";

router<string> r;
vector<string> sorted[30];//one for each letter, one for default
word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
contains_letter<back_insert_iterator<vector<string> > >
 contains_a(back_inserter(sorted[0]),'a');
r.add_destination(contains_a,router<string>::urgent_priority);
length_counter<back_insert_iterator<vector<string> > >
 length1(back_inserter(sorted[1]),1);
length_counter<back_insert_iterator<vector<string> > >
 length2(back_inserter(sorted[2]),2);
r.add_destination(length1,router<string>::high_priority);
r.add_destination(length2,router<string>::high_priority);
for(int i = 0; i<26;i++)
{
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+3]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::normal_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[29]));
r.set_default(dflt);
/*need to read all the input into a vector for storage so that
other priorities can be tested*/

//we want to create our own input for validation purposes
vector<string> storage_for_later;
createinputstring(storage_for_later);

//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());
cout<<"Results of testing urgent, high and normal priorities";
cout<<endl;
cout<<"words containing the letter (lowercase) a: ";
cout<<sorted[0].size()<<endl;
cout<<"words of length 1: "<<sorted[1].size()<<endl;
cout<<"words of length 2: "<<sorted[2].size()<<endl;
cout<<"counts for all other words, based on starting ";
cout<<"character, case insensitive"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 3].size()<<endl;
}
cout<<"Default: "<<sorted[29].size()<<endl;
if(validate(sorted))
{
  cout<<"Passed test for urgent, high and normal priorities";
  cout<<endl;
```

```
  }
  else
  {
    cout<<"Failed test for urgent, high and normal priorities";
    cout<<endl;
  }

  cout<<"Testing urgent, high and low priority"<<endl;
  if(testpriority(router<string>::urgent_priority,
     router<string>::high_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for urgent, high and low priority"<<endl;
  else
    cout<<"Failed test for urgent, high and low priority"<<endl;

  cout<<"Testing urgent, normal and low priority"<<endl;
  if(testpriority(router<string>::urgent_priority,
     router<string>::normal_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for urgent, normal and low priority"<<endl;
  else
    cout<<"Failed test for urgent, normal and low priority"<<endl;

  cout<<"Testing high, normal and low priority"<<endl;
  if(testpriority(router<string>::high_priority,
     router<string>::normal_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for high, normal and low priority"<<endl;
  else
    cout<<"Failed test for high, normal and low priority"<<endl;



  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 4267)
  {
    cout<<"Contains letter ""a"" failed"<<endl;
    return false;
  }
  else
```

```
{
  /*check to make sure that each string in this container
  contains an "a"*/
  for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
  {
    if(!((*iter).find("a")<(*iter).size()))
    {
      cout<<"String in ""a"" container does not contain ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[1].size() != 19683)
{
  cout<<"Length 1 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 1
  for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
  {
    if((*iter).size() != 1)
    {
      cout<<"String in length 1 container contains a string of ";
      cout<<"not length 1"<<endl;
      return false;
    }
  }
}
if(sorted[2].size() != 19683)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[2].begin();iter != sorted[2].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
      cout<<"not length 2"<<endl;
      return false;
    }
  }
}
for(int i = 3;i<29;i++)
```

```
  {
    if(sorted[i].size() != 2187)
    {
      cout<<"Begins letter """<<(char)(i+94)<<""" failed"<<endl;
      return false;
    }
    else
    {
      /*check each string to make sure it starts with the correct
      letter*/
      for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
      {
        if((*iter)[0] != (char)(i+94))
        {
          cout<<"String in begins with "<<(char)(i+94);
          cout<<" container ";
          cout<<"contains a string beginning with ";
          cout<<(*iter)[0]<<endl;
          return false;
        }
      }
    }
  }
  if(sorted[29].size() != 59049)
  {
    cout<<"Default case failed"<<endl;
    return false;
  }
  return true;
}
```

3-Priority without Fall-Through

"priority3_nofallthrough.cpp" 236 ≡

```
//priority3_nofallthrough.cpp version 1.2
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "test_classes.hpp"
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
```

```
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);
typedef router<string>::priority_level plevel;

template<typename InputIterator>
bool testpriority(plevel p1,plevel p2,
          plevel p3,InputIterator begin,InputIterator end)
{
  router<string> r;
  vector<string> sorted[30];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  contains_letter<back_insert_iterator<vector<string> > >
   contains_a(back_inserter(sorted[0]),'a');
  r.add_destination(contains_a,p1);
  length_counter<back_insert_iterator<vector<string> > >
   length1(back_inserter(sorted[1]),1);
  length_counter<back_insert_iterator<vector<string> > >
   length2(back_inserter(sorted[2]),2);
  r.add_destination(length1,p2);
  r.add_destination(length2,p2);
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i+3]),(char)(i + 97));
    r.add_destination(*dummy[i],p3);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[29]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(sorted);
}
int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority3_nofallthrough.cpp\n\n"
    << "Test router with 3 priorities (No fall through)\n"
    << "***********************************************\n\n"
    << "(Note: Fall through means that a destination\n"
    << "passes the element on to the next destination.\n\n"
    << "***********************************************\n\n"
    << "Results:\n\n";


  router<string> r;
  vector<string> sorted[30];//one for each letter, one for default
```

```
word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
contains_letter<back_insert_iterator<vector<string> > >
 contains_a(back_inserter(sorted[0]),'a');
r.add_destination(contains_a,router<string>::urgent_priority);
length_counter<back_insert_iterator<vector<string> > >
 length1(back_inserter(sorted[1]),1);
length_counter<back_insert_iterator<vector<string> > >
 length2(back_inserter(sorted[2]),2);
r.add_destination(length1,router<string>::high_priority);
r.add_destination(length2,router<string>::high_priority);
for(int i = 0; i<26;i++)
{
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+3]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::normal_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[29]));
r.set_default(dflt);

//we want to create our own input for validation purposes
vector<string> storage_for_later;
createinputstring(storage_for_later);

//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());
cout<<"Results of testing urgent, high and normal priorities";
cout<<endl;
cout<<"words containing the letter (lowercase) a: ";
cout<<sorted[0].size()<<endl;
cout<<"words of length 1: "<<sorted[1].size()<<endl;
cout<<"words of length 2: "<<sorted[2].size()<<endl;
cout<<"counts for all other words, based on starting ";
cout<<"character, case insensitive"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 3].size()<<endl;
}
cout<<"Default: "<<sorted[29].size()<<endl;
if(validate(sorted))
{
  cout<<"Passed test for urgent, high and normal priorities";
  cout<<endl;
}
else
{
  cout<<"Failed test for urgent, high and normal priorities";
  cout<<endl;
}
```

```
  cout<<"Testing urgent, high and low priority"<<endl;
  if(testpriority(router<string>::urgent_priority,
     router<string>::high_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for urgent high and low priority"<<endl;
  else
    cout<<"Failed test for urgent high and low priority"<<endl;

  cout<<"Testing urgent, normal and low priority"<<endl;
  if(testpriority(router<string>::urgent_priority,
     router<string>::normal_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for urgent, normal and low priority"<<endl;
  else
    cout<<"Failed test for urgent, normal and low priority"<<endl;

  cout<<"Testing high, normal and low priority"<<endl;
  if(testpriority(router<string>::high_priority,
     router<string>::normal_priority,
     router<string>::low_priority,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for high, normal and low priority"<<endl;
  else
    cout<<"Failed test for high, normal and low priority"<<endl;



  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 4267)
  {
    cout<<"Contains letter ""a"" failed"<<endl;
    return false;
  }
  else
  {
    /*check to make sure that each string in this container
    contains an "a"*/
    for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
```

```
  {
    if(!((*iter).find("a")<(*iter).size()))
    {
      cout<<"String in ""a"" container does not contain ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[1].size() != 18954)
{
  cout<<"Length 1 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 1
  for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
  {
    if((*iter).size() != 1)
    {
      cout<<"String in length 1 container contains a string of ";
      cout<<"not length 1"<<endl;
      return false;
    }
    /*also need to check to make sure that there are no 'a's in
    this contianer*/
    if((*iter).find("a")<(*iter).size())
    {
      cout<<"String in length 1 container contains an ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[2].size() != 18252)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[2].begin();iter != sorted[2].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
      cout<<"not length 2"<<endl;
      return false;
```

```
    }
    /*also need to check to make sure that there are no 'a's in
    this contianer*/
    if((*iter).find("a")<(*iter).size())
    {
      cout<<"String in length 2 container contains an ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[3].size() != 0)
{
  /*there are strings in the container for begins with "a" and
  there shouldn't be*/
  cout<<"Begins letter """<<(char)(97)<<""" failed"<<endl;
  return false;
}
for(int i = 4;i<29;i++)
{
  if(sorted[i].size() != 676)
  {
    cout<<"Begins letter """<<(char)(i+94)<<""" failed"<<endl;
    return false;
  }
  else
  {
    /*check each string to make sure it starts with the correct
    letter*/
    for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
    {
      if((*iter)[0] != (char)(i+94))
      {
        cout<<"String in begins with "<<(char)(i+94);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
      /*also need to check to make sure that there are no 'a's in
      this container*/
      if((*iter).find("a")<(*iter).size())
      {
        cout<<"Begins letter """<<(char)(i+94);
        cout<<""" contains the ";
        cout<<"letter 'a'"<<endl;
        return false;
      }
      /*also need to check that the string length of the string
      is greater than 2*/
```

```
      if((*iter).size() < 3)
      {
        cout<<"Begins letter """<<(char)(i+94)<<""" contains ";
        cout<<"a string of length "<<(*iter).size()<<endl;
        return false;
      }
    }
  }
  }
  if(sorted[29].size() != 676)
  {
    cout<<"Default case failed"<<endl;
    return false;
  }
  else
  {/*check to make sure that every string here begins with a "{"
  and that it does not contain an 'a' and is*/
    //of length greater than 2
    for(iter = sorted[29].begin();iter != sorted[29].end();iter++)
    {
      if((*iter)[0] != (char)(123))
      {
        cout<<"String in begins with "<<(char)(123);
        cout<<" container contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
      /*also need to check to make sure that there are no 'a's in
      this container*/
      if((*iter).find("a")<(*iter).size())
      {
        cout<<"Begins letter """<<(char)(123);
        cout<<""" contains the letter 'a'"<<endl;
        return false;
      }
      /*also need to check that the string length of the string is
      greater than 2*/
      if((*iter).size() < 3)
      {
        cout<<"Begins letter """<<(char)(123);
        cout<<""" contains a string ";
        cout<<"of length "<<(*iter).size()<<endl;
        return false;
      }
    }
  }
  return true;
}
```

C.4.3 4-Priority Tests

4-Priority with Fall-Through

"priority4_fallthrough.cpp" 243 ≡

```
//priority4_fallthrough.cpp version 1.0
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "test_classes_fallthrough.hpp"
#include "router.hpp"
#include "validate.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);

typedef router<string>::priority_level plevel;

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority4_fallthrough.cpp\n\n"
  <<"Test router with 4 priorities (allowing fall through)\n"
  <<"***********************************************************"
  <<"\n\n"
  <<"Note: Fall through means that a destination\n"
  <<"passes the element on to the next destination.\n\n"
  <<"*********************************************\n\n"
  <<"Results:\n\n";

  router<string> r;
  vector<string> sorted[31];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];

  contains_string<back_insert_iterator<vector<string> > >
   contains_ea(back_inserter(sorted[0]),"ea");
  r.add_destination(contains_ea,router<string>::urgent_priority);

  contains_letter<back_insert_iterator<vector<string> > >
```

```
 contains_a(back_inserter(sorted[1]),'a');
r.add_destination(contains_a,router<string>::high_priority);


length_counter<back_insert_iterator<vector<string> > >
 length1(back_inserter(sorted[2]),1);
length_counter<back_insert_iterator<vector<string> > >
 length2(back_inserter(sorted[3]),2);
r.add_destination(length1,router<string>::normal_priority);
r.add_destination(length2,router<string>::normal_priority);
for(int i = 0; i<26;i++)
{
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+4]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::low_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[30]));
r.set_default(dflt);
/*need to read all the input into a vector for storage so that
other priorities can be tested*/
vector<string> storage_for_later;
createinputstring(storage_for_later);
//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());
cout<<"Results of testing urgent, high, normal and low ";
cout<<"priorities"<<endl;
cout<<"Words containing the string \"ea\": ";
cout<<sorted[0].size()<<endl;
cout<<"words containing the letter (lowercase) a: ";
cout<<sorted[1].size()<<endl;
cout<<"words of length 1: "<<sorted[2].size()<<endl;
cout<<"words of length 2: "<<sorted[3].size()<<endl;
cout<<"counts for all other words, based on starting ";
cout<<"character, case insensitive"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 4].size()<<endl;
}
cout<<"Default: "<<sorted[30].size()<<endl;
if(validate(sorted))
  cout<<"Passed Validation Test"<<endl;
else
  cout<<"Failed Validation Test"<<endl;


cout<<"\n*************     END OF RESULTS     *************\n\n";

return 0;
```

```
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 81)
  {
    cout<<"Contains ""ae"" failed"<<endl;
    return false;
  }
  else
  {
    /*check to make sure that each string in this container
    contains "ea"*/
    for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
    {
      if(!((*iter).find("ea")<(*iter).size()))
      {
        cout<<"String in ""ea"" container does not contain ""ea""";
        cout<<endl;
        return false;
      }
    }
  }
  if(sorted[1].size() != 4267)
  {
    cout<<"Contains letter ""a"" failed"<<endl;
    return false;
  }
  else
  {
    /*check to make sure that each string in this container
    contains an "a"*/
    for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
    {
      if(!((*iter).find("a")<(*iter).size()))
      {
        cout<<"String in ""a"" container does not contain ""a""";
        cout<<endl;
        return false;
      }
    }
  }
  if(sorted[2].size() != 19683)
  {
    cout<<"Length 1 failed"<<endl;
    return false;
  }
  else
  {
```

```
  //check every string to make sure it is of length 1
  for(iter = sorted[2].begin();iter != sorted[2].end();iter++)
  {
    if((*iter).size() != 1)
    {
      cout<<"String in length 1 container contains a string of ";
      cout<<"not length 1"<<endl;
      return false;
    }
  }
}
if(sorted[3].size() != 19683)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[3].begin();iter != sorted[3].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
      cout<<"not length 2"<<endl;
      return false;
    }
  }
}
for(int i = 4;i<30;i++)
{
  if(sorted[i].size() != 2187)
  {
    cout<<"Begins letter """<<(char)(i+93)<<""" failed"<<endl;
    return false;
  }
  else
  {
    /*check each string to make sure it starts with the correct
    letter*/
    for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
    {
      if((*iter)[0] != (char)(i+93))
      {
        cout<<"String in begins with "<<(char)(i+93);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
```

```
      }
    }
  }
  //everything should be caught by the default case
  if(sorted[30].size() != 59049)
  {
    cout<<"Default case failed"<<endl;
    return false;
  }
  return true;
}
```

4-Priority without Fall-Through

"priority4_nofallthrough.cpp" 247 ≡

```
//priority4_nofallthrough.cpp version 1.0
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "test_classes.hpp"
#include "router.hpp"
#include "validate.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<string> sorted[]);

typedef router<string>::priority_level plevel;

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  priority4_nofallthrough.cpp\n\n"
     << "Test router with 4 priorities (No fall through)\n"
     << "*********************************************\n\n"
     << "(Note: Fall through means that a destination\n"
     << "passes the element on to the next destination.\n\n"
     << "*********************************************\n\n"
     << "Results:\n\n";
```

```
router<string> r;
vector<string> sorted[31];//one for each letter, one for default
word_sorter<back_insert_iterator<vector<string> > > *dummy[26];

contains_string<back_insert_iterator<vector<string> > >
 contains_ea(back_inserter(sorted[0]),"ea");
r.add_destination(contains_ea,router<string>::urgent_priority);

contains_letter<back_insert_iterator<vector<string> > >
 contains_a(back_inserter(sorted[1]),'a');
r.add_destination(contains_a,router<string>::high_priority);


length_counter<back_insert_iterator<vector<string> > >
 length1(back_inserter(sorted[2]),1);
length_counter<back_insert_iterator<vector<string> > >
 length2(back_inserter(sorted[3]),2);
r.add_destination(length1,router<string>::normal_priority);
r.add_destination(length2,router<string>::normal_priority);
for(int i = 0; i<26;i++)
{
  dummy[i] =
   new word_sorter<back_insert_iterator<vector<string> > >
   (back_inserter(sorted[i+4]),(char)(i + 97));
  r.add_destination(*dummy[i],router<string>::low_priority);
}
my_default<back_insert_iterator<vector<string> > >
 dflt(back_inserter(sorted[30]));
r.set_default(dflt);
/*need to read all the input into a vector for storage so that
other priorities can be tested*/
vector<string> storage_for_later;
createinputstring(storage_for_later);
//distribute
r.distribute(storage_for_later.begin(),storage_for_later.end());
cout<<"Results of testing urgent, high, normal and low ";
cout<<"priorities"<<endl;
cout<<"Words containing the string \"ea\": ";
cout<<sorted[0].size()<<endl;
cout<<"words containing the letter (lowercase) a: ";
cout<<sorted[1].size()<<endl;
cout<<"words of length 1: "<<sorted[2].size()<<endl;
cout<<"words of length 2: "<<sorted[3].size()<<endl;
cout<<"counts for all other words, based on starting ";
cout<<"character, case insensitive"<<endl;
for(int i = 0;i<26;i++)
{
  cout<<(char)(i + 97)<<": "<<sorted[i + 4].size()<<endl;
```

```
  }
  cout<<"Default: "<<sorted[30].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;


  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  if(sorted[0].size() != 81)
  {
    cout<<"Contains ""ae"" failed"<<endl;
    return false;
  }
  else
  {
    /*check to make sure that each string in this container
    contains "ea"*/
    for(iter = sorted[0].begin();iter != sorted[0].end();iter++)
    {
      if(!((*iter).find("ea")<(*iter).size()))
      {
        cout<<"String in ""ea"" container does not contain ""ea""";
        cout<<endl;
        return false;
      }
    }
  }
  if(sorted[1].size() != 4186)
  {
    cout<<"Contains letter ""a"" failed"<<endl;
    return false;
  }
  else
  {
    /*check to make sure that each string in this container
    contains an "a"*/
    for(iter = sorted[1].begin();iter != sorted[1].end();iter++)
    {
      if(!((*iter).find("a")<(*iter).size()))
      {
        cout<<"String in ""a"" container does not contain ""a""";
        cout<<endl;
```

```
      return false;
    }
    //also need to check that no string contains "ea"
    if((*iter).find("ea")<(*iter).size())
    {
      cout<<"String in ""a"" container contains an ""ea"""<<endl;
      return false;
    }
  }
}
if(sorted[2].size() != 18954)
{
  cout<<"Length 1 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 1
  for(iter = sorted[2].begin();iter != sorted[2].end();iter++)
  {
    if((*iter).size() != 1)
    {
      cout<<"String in length 1 container contains a string of ";
      cout<<"not length 1"<<endl;
      return false;
    }
    /*also need to check to make sure that there are no 'a's in
    this container; dont need to check "ea" because that is a
    string of length 2*/
    if((*iter).find("a")<(*iter).size())
    {
      cout<<"String in length 1 container contains an ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[3].size() != 18252)
{
  cout<<"Length 2 failed"<<endl;
  return false;
}
else
{
  //check every string to make sure it is of length 2
  for(iter = sorted[3].begin();iter != sorted[3].end();iter++)
  {
    if((*iter).size() != 2)
    {
      cout<<"String in length 2 container contains a string of ";
```

```
      cout<<"not length 2"<<endl;
      return false;
    }
    /*also need to check to make sure that there are no 'a's in
    this contianer; dont need to check "ea" because that is
    contains an 'a' and would be caught by the check for 'a's*/
    if((*iter).find("a")<(*iter).size())
    {
      cout<<"String in length 2 container contains an ""a""";
      cout<<endl;
      return false;
    }
  }
}
if(sorted[4].size() != 0)
{
  /*there are strings in the container for begins with "a" and
  there shouldn't be*/
    cout<<"Begins letter """<<(char)(97);
    cout<<""" failed"<<endl;
    return false;
}
for(int i = 5;i<30;i++)
{
  if(sorted[i].size() != 676)
  {
    cout<<"Begins letter """<<(char)(i+93)<<""" failed"<<endl;
    return false;
  }
  else
  {
    /*check each string to make sure it starts with the correct
    letter*/
    for(iter = sorted[i].begin();iter != sorted[i].end();iter++)
    {
      if((*iter)[0] != (char)(i+93))
      {
        cout<<"String in begins with "<<(char)(i+93);
        cout<<" container ";
        cout<<"contains a string beginning with ";
        cout<<(*iter)[0]<<endl;
        return false;
      }
      /*also need to check to make sure that there are no 'a's in
      this container; dont need to check "ea" because that is
      contains an 'a' and would be caught by the check for 'a's*/
      if((*iter).find("a")<(*iter).size())
      {
        cout<<"Begins letter """<<(char)(i+93);
        cout<<""" contains the letter 'a'"<<endl;
```

```
        return false;
      }
      /*also need to check that the string length of the string
      is greater than 2*/
      if((*iter).size() < 3)
      {
        cout<<"Begins letter """<<(char)(i+93);
        cout<<""" contains a string of length ";
        cout<<(*iter).size()<<endl;
        return false;
      }
    }
  }
}
if(sorted[30].size() != 676)
{
  cout<<"Default case failed"<<endl;
  return false;
}
else
{/*check to make sure that every string here begins with a "{"
 and that it does not contain an 'a' and is of length
 greater than 2*/
  for(iter = sorted[30].begin();iter != sorted[30].end();iter++)
  {
    if((*iter)[0] != (char)(123))
    {
      cout<<"String in begins with "<<(char)(123)<<" container ";
      cout<<"contains a string beginning with ";
      cout<<(*iter)[0]<<endl;
      return false;
    }
    /*also need to check to make sure that there are no 'a's in
    tihs container; dont need to check "ea" because that is
    contains an 'a' and would be caught by the check for 'a's*/
    if((*iter).find("a")<(*iter).size())
    {
      cout<<"Begins letter """<<(char)(123);
      cout<<""" contains the letter 'a'"<<endl;
      return false;
    }
    /*also need to check that the string length of the string is
    greater than 2*/
    if((*iter).size() < 3)
    {
      cout<<"Begins letter """<<(char)(123);
      cout<<""" contains a ";
      cout<<"string of length "<<(*iter).size()<<endl;
      return false;
    }
```

```
    }
  }
  return true;
}
```

C.5 Insert Iterator Tests

"test_inserters.cpp" 253 ≡

```
//test_inserters.cpp version 1.1
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include <deque>
#include <list>
#include "router.hpp"
#include "test_classes.hpp"
#include "validate.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::list;
using std::deque;
using std::back_inserter;
using std::front_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::front_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

template<typename InputIterator>
bool testbackdeque(vector<string> results[27],InputIterator begin,
                   InputIterator end);
template<typename InputIterator>
bool testbacklist(vector<string> results[27],InputIterator begin,
                  InputIterator end);
template<typename InputIterator>
bool testfrontdeque(vector<string> results[27],InputIterator begin,
                    InputIterator end);
template<typename InputIterator>
bool testfrontlist(vector<string> results[27],InputIterator begin,
                   InputIterator end);
```

```
int main(int argc,char *argv[])
{

  cout << "\n\n\ntestinserters.cpp\n\n"
    << "Test of Reading from Standard Library Containers\n"
    << "using front or back inserters (where applicable).\n"
    << "************************************************"
    << "\n\n"
    << "Results:\n\n";


  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  typedef istream_iterator<string> in_string;
  vector<string> storage_for_later;
  in_string input(cin);
  while(input != in_string())
    storage_for_later.push_back(*input++);

  //distribute
  r.distribute(storage_for_later.begin(),storage_for_later.end());
  cout<<"Results for filtering into vector using a back insert ";
  cout<<"iterator.  Will be used for testing other insert ";
  cout<<"iterators"<<endl;
  cout<<"Number of words beginning with each letter, case ";
  cout<<"insensitive:"<<endl;

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Default: "<<sorted[26].size()<<endl<<endl;

  cout<<"Testing back insert iterator deque"<<endl;
  if(testbackdeque(sorted,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for back insert iterator deque"<<endl;
  else
```

```
    cout<<"Failed test for back insert iterator deque"<<endl;

  cout<<"Testing back insert iterator list"<<endl;
  if(testbacklist(sorted,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for back insert iterator list"<<endl;
  else
    cout<<"Failed test for back insert iterator list"<<endl;

  cout<<"Testing front insert iterator deque"<<endl;
  if(testfrontdeque(sorted,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for front insert iterator deque"<<endl;
  else
    cout<<"Failed test for front insert iterator deque"<<endl;

  cout<<"Testing front insert iterator list"<<endl;
  if(testfrontlist(sorted,storage_for_later.begin(),
     storage_for_later.end()))
    cout<<"Passed test for front insert iterator list"<<endl;
  else
    cout<<"Failed test for front insert iterator list"<<endl;

  cout<<"\n*************     END OF RESULTS     *************";
  cout<<"\n\n";

  return 0;
}


template<typename InputIterator>
bool testbackdeque(vector<string> results[27],InputIterator begin,
                   InputIterator end)
{
  router<string> r;
  deque<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<deque<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<deque<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<deque<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(results,sorted,27);
}
```

```
template<typename InputIterator>
bool testfrontdeque(vector<string> results[27],InputIterator begin,
                    InputIterator end)
{
  router<string> r;
  deque<string> sorted[27];//one for each letter, one for default
  word_sorter<front_insert_iterator<deque<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<front_insert_iterator<deque<string> > >
     (front_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<front_insert_iterator<deque<string> > >
   dflt(front_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(results,sorted,27,true);
}

template<typename InputIterator>
bool testbacklist(vector<string> results[27],InputIterator begin,
                  InputIterator end)
{
  router<string> r;
  list<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<list<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<list<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<list<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(results,sorted,27);
}

template<typename InputIterator>
bool testfrontlist(vector<string> results[27],InputIterator begin,
                   InputIterator end)
{
  router<string> r;
  list<string> sorted[27];//one for each letter, one for default
  word_sorter<front_insert_iterator<list<string> > > *dummy[26];
```

```
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<front_insert_iterator<list<string> > >
     (front_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<front_insert_iterator<list<string> > >
   dflt(front_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(begin,end);
  return validate(results,sorted,27,true);
}
```

C.6 Standard Library Containers Read Tests

"testreadcont.cpp" 257 ≡

```
//testreadcont.cpp version 1.1
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include <set>
#include <deque>
#include <list>
#include <map>
#include "router.hpp"
#include "test_classes.hpp"
#include "validate.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::deque;
using std::multiset;
using std::list;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;
using std::map;
using std::pair;

using gdfa::router;
using gdfa::destination;

template<typename InputIterator>
bool testdeque(vector<string> results[27], InputIterator begin,
```

```
              InputIterator end);
template<typename InputIterator>
bool testmultiset(vector<string> results[27], InputIterator begin,
                  InputIterator end);
template<typename InputIterator>
bool testlist(vector<string> results[27], InputIterator begin,
              InputIterator end);
template<typename InputIterator>
bool testmap(vector<string> results[27], InputIterator begin,
             InputIterator end);
template<typename conttype1,typename conttype2>
bool validatemap(conttype1 key[],conttype2 res[],int size);


int main(int argc,char *argv[])
{

  cout << "\n\n\nTest:   testreadcont.cpp\n\n"
     << "Test Reading from Standard Library Containers\n"
     << "*********************************************\n\n"
     << "Results:\n\n";

  //first thing to do is read all the input into a container
  typedef istream_iterator<string> in_string;
  string input;
  vector<string> storage;
  while(cin>>input)
  {
    storage.push_back(input);
  }
  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(storage.begin(),storage.end());
  cout<<"Results for reading from a vector, used for comparison ";
  cout<<"testing"<<endl;
  cout<<"Number of words beginning with each letter, case ";
  cout<<"insensitive"<<endl;
  for(int i = 0;i<26;i++)
  {
```

```
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"default: "<<sorted[26].size()<<endl<<endl;

  cout<<"Testing reading from a deque"<<endl;
  if(testdeque(sorted,storage.begin(),storage.end()))
    cout<<"Passed test for reading from a deque"<<endl;
  else
    cout<<"Failed test for reading from a deque"<<endl;

  cout<<"Testing reading from a list"<<endl;
  if(testlist(sorted,storage.begin(),storage.end()))
    cout<<"Passed test for reading from a list"<<endl;
  else
    cout<<"Failed test for reading from a list"<<endl;

  cout<<"Testing reading from a multiset"<<endl;
  if(testmultiset(sorted,storage.begin(),storage.end()))
    cout<<"Passed test for reading from a multiset"<<endl;
  else
    cout<<"Failed test for reading from a multiset"<<endl;

  cout<<"Testing reading from a map"<<endl;
  if(testmap(sorted,storage.begin(),storage.end()))
    cout<<"Passed test for reading from a map"<<endl;
  else
    cout<<"Failed test for reading from a map"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}

template<typename InputIterator>
bool testdeque(vector<string> results[27], InputIterator begin,
               InputIterator end)
{
  deque<string> storage(begin,end);
  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
```

```
  r.set_default(dflt);
  r.distribute(storage.begin(),storage.end());
  validate(results,sorted,27);
}

template<typename InputIterator>
bool testmultiset(vector<string> results[27], InputIterator begin,
                  InputIterator end)
{
  multiset<string> storage;
  while(begin != end)
    storage.insert(*begin++);

  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
    dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(storage.begin(),storage.end());
  /*because a set is not sequential, the sequential validate test
  is not applicable and a validation test based on the size of
  the sorted vectors is used*/
  validatesize(results,sorted,27);
}

template<typename InputIterator>
bool testlist(vector<string> results[27], InputIterator begin,
              InputIterator end)
{
  list<string> storage(begin,end);
  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
```

```
  r.set_default(dflt);
  r.distribute(storage.begin(),storage.end());
  validate(results,sorted,27);
}

template<typename InputIterator>
bool testmap(vector<string> results[27], InputIterator begin,
             InputIterator end)
{
  string input;
  map<string,int> storage;
  while(begin != end)
    storage[*begin++]++;
  router<pair<string,int> > r;
  //one for each letter, one for default
  vector<pair<string,int> > sorted[27];
  word_sortermap<back_insert_iterator<vector<pair<string,int> > > >
   *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sortermap<back_insert_iterator<vector<
      pair<string,int> > > >(back_inserter(sorted[i]),
      (char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_defaultmap<back_insert_iterator<vector<pair<string,int> > > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(storage.begin(),storage.end());
  return validatemap(results,sorted,27);
}

/*validatemap() works by getting a sum of the number of words
beginning with each letter (from res) and comparing it to the
number of words beginning with each letter in the key.  key is
generated from the baseline test*/
template<typename conttype1,typename conttype2>
bool validatemap(conttype1 key[],conttype2 res[],int size)
{
  typedef typename conttype1::iterator key_iter;
  key_iter ikey;
  typedef typename conttype2::iterator res_iter;
  res_iter ires;
  for(int i = 0;i < size;i++)
  {
    int total = 0;
    for(vector<pair<string,int> >::iterator iter = res[i].begin();
        iter!=res[i].end();iter++)
      total += iter->second;
```

```
    if(key[i].size() != total)
    {
      cout<<"match failed for container: "<<i<<endl;
      return false;
    }
  }
  return true;
}
```

C.7 Array Read Tests

"testarray.cpp" 262 ≡

```
//testarrayinput.cpp version 1.1
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
#include "test_classes.hpp"
#include "validate.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool filterarray(vector<string> results[],string storage[],
                 int size);

int main(int argc,char *argv[])
{

  cout << "\n\n\nTest:  testarray.cpp"
    << "\n\nTest Using an Array as input.\n"
    << "**********************************\n\n"
    << "Results:\n\n";

  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
```

```
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);

  typedef istream_iterator<string> in_string;
  vector<string> storage;
  in_string input(cin);

  while(input != in_string())
    storage.push_back(*input++);

  string *strarray = new string[storage.size()];
  for(int i=0;i<storage.size();i++)
    strarray[i] = storage[i];

  //filtervector and print results
  r.distribute(storage.begin(),storage.end());
  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"default: "<<sorted[26].size()<<endl<<endl;

  //step 3, 4
  //filter array and validate
  cout<<"Testing using an array as input"<<endl;
  if(filterarray(sorted,strarray,storage.size()))
    cout<<"Passed array input test"<<endl;
  else
    cout<<"Failed array input test"<<endl;

  cout<<"\n*************     END OF RESULTS     *************\n\n";

  return 0;
}
bool filterarray(vector<string> results[],string storage[],
                 int size)
{
  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
```

```
      (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  r.distribute(&storage[0],&storage[size]);
  return validate(results,sorted,27);
}
```

C.8 Tests Reading from and Writing to Files

"testrwfiles.cpp" 264 ≡

```
//testrwfiles.cpp version 1.0
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::ostream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;
using std::ifstream;
using std::ofstream;

using gdfa::router;
using gdfa::destination;

template<typename output_iterator>
class word_sorter : public destination<string>
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sorter(output_iterator out, char first_letter)
         : m_out(out), m_first_letter(first_letter) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const string& element)
```

```
  {
     if (tolower((const int)element[0]) == m_first_letter)
     {
    *m_out++ = element;
            *m_out++ = "\n";
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<string>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const string& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  testrwfiles.cpp\n\nTest of reading and";
  cout<<"writing from and to files with ifstream and ofstream.\n"
  <<"******************************"
  <<"**************************\n\n";

  //as a command line arguement it takes a file for input
  if(argc != 2)
  {
    cout<<"Usage ./app <inputfilename>"<<endl;
    return 1;
  }
  typedef ostream_iterator<string> out_string;
  router<string> r;
  ofstream output[27];//out for each letter and one for default
  word_sorter<out_string> *dummy[26];
  for(int i = 0; i<26;i++)
  {
    string filename = "output_";
    filename += (char)(i+97);
    output[i].open(filename.c_str());
```

```
      dummy[i] = new word_sorter<out_string>(out_string(output[i]),
                                             (char)(i + 97));
      r.add_destination(*dummy[i]);
    }
    output[26].open("output_default");
    my_default<out_string> dflt =
     my_default<out_string>(out_string(output[26]));
    r.set_default(dflt);
    ifstream filein;
    filein.open(argv[1]);
    if(filein == NULL)
    {
      cout<<argv[1]<<" invalid file"<<endl;
      return 1;
    }
    typedef istream_iterator<string> in_string;
    r.distribute(in_string(filein),in_string());

    cout<<"\n*************     END OF TEST     *************\n\n";

    return 0;
  }
```

"validaterwfiles.cpp" 266 ≡

```
  //validaterwfiles.cpp version 1.1

  #include <iterator>
  #include <iostream>
  #include <fstream>
  #include <string>
  #include <ctype.h>

  using std::cout;
  using std::endl;
  using std::istream_iterator;
  using std::ifstream;
  using std::string;

  typedef istream_iterator<string> string_in;

  int main(int argc, char *argv[])
  {

    cout << "\n\n\nTest:  validaterwfiles.cpp\n\n";
    cout << "Validating the testrwfiles test. "
    <<"\n******************************"
    <<"*************************\n\n"
    << "Results:\n\n";

    for(int i = 0;i<26;i++)
```

```
{
  string file = "output_";
  file += (char)(i+97);
  ifstream input(file.c_str());
  if(input == NULL)
  {
    cout<<"Could not open file: "<<file<<endl;
    return 0;
  }
  string_in reader(input);
  while(reader != string_in())
  {
    if(tolower((*reader)[0]) != (i+97))
    {
      cout<<"Test Failed"<<endl;
      cout<<"In file: "<<file<<" string: "<<*reader;
      cout<<" doesn't begin with a(n) """<<(char)(i+97);
      cout<<""."<<endl;
      return 0;
    }
    reader++;
  }
  input.close();
}
string file = "output_default";
ifstream input(file.c_str());
if(input == NULL)
{
  cout<<"Could not open file: "<<file<<endl;
  return 0;
}
string_in reader(input);
while(reader != string_in())
{
  //we want to check vs lowercase chars
  if(tolower((int)(*reader)[0]) > 96 &&
     tolower((int)(*reader)[0]) < 123)
  {
    cout<<"Test Failed"<<endl;
    cout<<"In file: "<<file<<" string: "<<*reader;
    cout<<" begins with a(n) """<<(*reader)[0];
    cout<<""."<<endl;
    return 0;
  }
  reader++;
}
input.close();
cout<<"Passed rwfiles test"<<endl;

cout<<"\n*************     END OF RESULTS     *************\n\n";
```

```
  return 0;
}
```

C.9 Tests with `std::istream_iterators`

C.9.1 Number Generation Utilities

`int` Generator

"genints.cpp" 268a ≡

```
#include <iostream>
#include "random.h"
#include <time.h>

using namespace std;

int main(int argc,char *argv[])
{
  if(argc != 2)
  {
    cout<<"Usage: ./genints <number_of_ints_to_generate>";
    cout<<endl;
    return -1;
  }
  int num = atoi(argv[1]);
  randgen.seed(time(NULL));
  for(int i = 0;i<num;i++)
  {
    cout<<randgen(1000000)<<endl;
  }
  return 0;
}
```

`double` Generator

"gendoubles.cpp" 268b ≡

```
#include <iostream>
#include "random.h"
#include <time.h>

using namespace std;

int main(int argc,char *argv[])
{
  if(argc != 2)
  {
    cout<<"Usage: ./gendouble <number_of_doubles_to_generate>";
    cout<<endl;
```

```
    return -1;
  }
  int num = atoi(argv[1]);
  randgen.seed(time(NULL));
  for(int i = 0;i<num;i++)
  {
    cout<<randgen()<<endl;
  }
  return 0;
}
```

Random Number Utility

The following utility is utilised by the random `int` generator (Appendix C.9.1) and the random `double` generator (Appendix C.9.1).

"random.h" 269 ≡

```
/*
 *
 * Copyright (c) 1994
 * Hewlett-Packard Company
 *
 * Permission to use, copy, modify, distribute and sell this
 * software and its documentation for any purpose is hereby
 * granted without fee, provided that the above copyright notice
 * appear in all copies and that both that copyright notice and
 * this permission notice appear in supporting documentation.
 * Hewlett-Packard Company makes no representations about the
 * suitability of this software for any purpose.  It is provided
 * "as is" without express or implied warranty.
 *
 */

/*
 * void randgen.seed(unsigned long j)  reseeds the generator with j
 *
 * unsigned long ranggen(unsigned long limit)
 *     returns a pseudo-random number n, 0 <= n < limit,
 *     uniformly distributed
 *
 * double randgen()
 *     returns a pseudo-random number d, 0.0 <= n < 1.0,
 *     uniformly distributed
 */
#ifndef _RANDOM_H
#define _RANDOM_H

#include <stddef.h>
#include <climits>  // for ULONG_MAX
```

```
#define __SEED 161803398

class random_generator {
protected:
    unsigned long table[55];
    size_t index1;
    size_t index2;
public:
    unsigned long operator()(unsigned long limit) {
                index1 = (index1 + 1) % 55;
                index2 = (index2 + 1) % 55;
                table[index1] -= table[index2];
                return table[index1] % limit;
    }
    inline double operator()() {
        return (double)operator()(ULONG_MAX) / ULONG_MAX;
    }
    void seed(unsigned long j);
    random_generator(unsigned long j) { seed(j); }
};

void random_generator::seed(unsigned long j) {
    unsigned long k = 1;
    table[54] = j;
    size_t i;
    for (i = 0; i < 54; i++) {
        size_t ii = (21 * (i + 1) % 55) - 1;
        table[ii] = k;
        k = j - k;
        j = table[ii];
    }
    for (int loop = 0; loop < 4; loop++) {
        for (i = 0; i < 55; i++)
            table[i] = table[i] - table[(1 + i + 30) % 55];
    }
    index1 = 0;
    index2 = 31;
}

static random_generator randgen(__SEED);
#endif
```

C.9.2 std::istream_iterator Tests

std::istream_iterator<int> Test

"test_int_istream.cpp" 270 ≡

```
//test_int.cpp version 1.0
```

```
#include <iostream>
#include <fstream>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<int> sorted[]);

template<typename output_iterator>
class range_checker : public destination<int>
{
   private:
  output_iterator m_out;
  int m_lower;
                        int m_upper;

   public:
  range_checker(output_iterator out, int lower,int upper)
         : m_out(out), m_lower(lower), m_upper(upper) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const int& element)
  {
     if(m_lower < element && m_upper > element)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<int>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
```

```
        : m_out(out) {}

  bool distribute(const int& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  test_int_istream.cpp"
    << "\n\nTest Uses istream iterator and type int as input.\n"
    << "**********************************\n\n"
    << "Results:\n\n";

  router<int> r;
  vector<int> sorted[11];//one for each letter, one for default
  range_checker<back_insert_iterator<vector<int> > > *dummy[10];
  int lower = 0;
  int upper = 100000;
  for(int i = 0; i<10;i++)
  {
    dummy[i] =
     new range_checker<back_insert_iterator<vector<int> > >
     (back_inserter(sorted[i]), lower, upper);
    r.add_destination(*dummy[i]);
    lower+= 100000;
    upper+= 100000;
  }
  my_default<back_insert_iterator<vector<int> > >
   dflt(back_inserter(sorted[10]));
  r.set_default(dflt);

  typedef istream_iterator<int> int_in;
  r.distribute(int_in(cin),int_in());
  cout<<"number of elements in each range, with the max and min ";
  cout<<"element in the ranges"<<endl;
  for(int i = 0;i<10;i++)
  {
    cout<<"("<<i * 100000<<","<<(i+1)*100000<<"): ";
    cout<<sorted[i].size()<<" ";
    cout<<*max_element(sorted[i].begin(),sorted[i].end())
      <<" "<<*min_element(sorted[i].begin(),sorted[i].end())<<endl;
  }
  cout<<"Number elements caught by default case: ";
  cout<<sorted[10].size()<<endl;

  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
```

```
  else
    cout<<"Failed Validation Test"<<endl;

  cout<<"\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";

  return 0;
}


bool validate(vector<int> sorted[])
{
  int i;
  int j = 0;
  for(i = 0,j=0;i<10;i++,j+=100000)
  {
    int min = *min_element(sorted[i].begin(),sorted[i].end());
    if(j>=min)
    {
      cout<<"Failed, "<<min<<" is not in range ("<<j<<",";
      cout<<j+100000<<")"<<endl;
      return false;
    }
    double max = *max_element(sorted[i].begin(),sorted[i].end());
    if(j+100000 <= max)
    {
      cout<<"Failed, "<<max<<" is not in range ("<<j<<",";
      cout<<j+100000<<")"<<endl;
      return false;
    }
  }
  return true;

}
```

std::istream_iterator<double> Test

"test_double_istream.cpp" 273 ≡

```
//test_double.cpp version 1.0
#include <iostream>
#include <fstream>
#include <ctype.h>
#include <algorithm>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
```

```
using std::back_insert_iterator;
using std::cin;
using std::min_element;
using std::max_element;
using gdfa::router;
using gdfa::destination;

bool validate(vector<double> sorted[]);

class my_compare
{
public:
  my_compare() {}
  static const double epsilon = 1e-10;
  bool less(const double &lhs,const double &rhs)
  {
    if((rhs - lhs) < epsilon) return false;
    if(lhs<rhs) return true;
    return false;
  }
  bool greater(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon) return false;
    if(lhs>rhs) return true;
    return false;
  }
  bool lessequal(const double &lhs,const double &rhs)
  {
    if((rhs - lhs) < epsilon && (rhs - lhs) > -epsilon)
     return true;
    if(lhs<rhs) return true;
    return false;
  }
  bool greaterequal(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon && (lhs - rhs) > -epsilon)
     return true;
    if(lhs>rhs) return true;
    return false;
  }
  bool equal(const double &lhs,const double &rhs)
  {
    if((lhs - rhs) < epsilon && (rhs - lhs) < -epsilon)
     return true;
    if((rhs -lhs) < epsilon && (lhs - rhs) < -epsilon)
     return true;
    return false;
  }

};
```

```
my_compare comp;

template<typename output_iterator>
class range_checker : public destination<double>
{
   private:
      output_iterator m_out;
      double m_lower;
      double m_upper;
//      static my_compare comp;

   public:
      range_checker(output_iterator out, double lower,double upper)
                : m_out(out), m_lower(lower), m_upper(upper) {}

      /*out distribute function checks the first letter to see if
      it is the same as in the predicate*/
      bool distribute(const double& element)
      {
        if(comp.greater(element,m_lower) &&
           comp.less(element,m_upper))
        {
            if( element < .3000000001 && element > .29999999)
            {
              cout<<"dumber than dirt "<<element<<endl;
              cout<<element - m_lower<<endl;
/*              if((element - m_lower) > epsilon)
              cout<<"greater"<<endl;
              if((m_lower - element) > epsilon)
                cout<<"less than"<<endl;*/
            }
            *m_out++ = element;
            return true;
         }

         return false;
      }
};

template<typename output_iterator>
class my_default : public destination<double>
{
   private:
      output_iterator m_out;
   public:
      my_default(output_iterator out)
                : m_out(out) {}

      bool distribute(const double& n)
```

```
      {
         *m_out++ = n;
         return true;
      }
};

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  test_double_istream.cpp"
    << "\n\nTest Uses istream iterator and type double as input.\n"
    << "***********************************\n\n"
    << "Results:\n\n";

  router<double> r;
  vector<double> sorted[11];//one for each letter, one for default
  range_checker<back_insert_iterator<vector<double> > > *dummy[10];
  double lower = 0.0;
  double upper = 0.1;
  for(int i = 0; i<10;i++)
  {
    dummy[i] =
     new range_checker<back_insert_iterator<vector<double> > >
     (back_inserter(sorted[i]), lower, upper);
    r.add_destination(*dummy[i]);
    lower+= 0.1;
    upper+= 0.1;
  }
  my_default<back_insert_iterator<vector<double> > >
   dflt(back_inserter(sorted[10]));
  r.set_default(dflt);

  typedef istream_iterator<double> double_in;
  r.distribute(double_in(cin),double_in());
  cout<<"number of elements in each range, with the max and min ";
  cout<<"element in the ranges"<<endl;
  double j;
  int i;
  for(i = 0,j = 0.0;i<10;i++,j+=.1)
  {
    cout<<"("<<j<<","<<j+.1<<"): "<<sorted[i].size()<<" ";
    cout<<*max_element(sorted[i].begin(),sorted[i].end())
      <<" "<<*min_element(sorted[i].begin(),sorted[i].end())<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[10].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;
```

```
  cout<<"\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";


  return 0;
}



bool validate(vector<double> sorted[])
{
  int i;
  double j;
  for(i = 0,j = 0.0;i<10;i++,j+=.1)
  {
    double min = *min_element(sorted[i].begin(),sorted[i].end());
    if(comp.greaterequal(j,min))
    {
      cout<<"Failed, "<<min<<" is not in range ("<<j<<","<<j+.1;
      cout<<")"<<endl;
      return false;
    }
    double max = *max_element(sorted[i].begin(),sorted[i].end());
    if(comp.lessequal(j+.1,max))
    {
      cout<<"Failed, "<<max<<" is not in range ("<<j<<","<<j+.1;
      cout<<")"<<endl;
      return false;
    }
  }
  return true;

}
```

std::istream_iterator<char> Test

"test_char_istream.cpp" 277 ≡

```
//test_char_istream.cpp version 1.0
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
```

```
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

bool validate(vector<char> sorted[]);

template<typename output_iterator>
class word_sorter : public destination<char>
{
   private:
  output_iterator m_out;
  char m_first_letter;

   public:
  word_sorter(output_iterator out, char first_letter)
         : m_out(out), m_first_letter(first_letter) {}

  /*out distribute function checks the first letter to see if it is
  the same as in the predicate*/
  bool distribute(const char& element)
  {
     if (tolower((const int)element) == m_first_letter)
     {
    *m_out++ = element;
    return true;
     }

     return false;
  }
};

template<typename output_iterator>
class my_default : public destination<char>
{
   private:
  output_iterator m_out;
   public:
  my_default(output_iterator out)
         : m_out(out) {}

  bool distribute(const char& n)
  {
     *m_out++ = n;
     return true;
  }
};

int main(int argc,char *argv[])
```

```
{
  cout << "\n\n\nTest:  test_char_istream.cpp"
    << "\n\nTest Uses istream iterator and type char as input.\n"
    << "**********************************\n\n"
    << "Results:\n\n";

  router<char> r;
  vector<char> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<char> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<char> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<char> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);

  typedef istream_iterator<char> char_in;
  r.distribute(char_in(cin),char_in());

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[26].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;


  cout<<"\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";


  return 0;
}

bool validate(vector<char> sorted[])
{
  vector<char>::iterator iter;
  for(int i = 0;i<26;i++)
  {
    iter = sorted[i].begin();
    for(;iter != sorted[i].end();iter++)
    {
```

```
        //we want to check vs lowercase chars
        if(tolower((int)*iter) != (char)(i + 97))
        {
          cout<<"Wrong letter: "<<*iter<<" should be ";
          cout<<(char)(i + 97)<<" "<<i<<endl;
          return false;
          break;
        }
      }
    }
    iter = sorted[26].begin();
    for(;iter != sorted[26].end();iter++)//check default case
    {
      //we want to check vs lowercase chars
      if(tolower((int)*iter) > 96 && tolower((int)*iter) < 123)
      {
    cout<<"Wrong letter: "<<*iter<<" should not be in default case";
    cout<<endl;
    return false;
    break;
      }
    }

    return true;
  }
```

std::istream_iterator<std::string> Test

"test_string_istream.cpp" 280 ≡

```
  //test_string.cpp version 1.0
  #include <iostream>
  #include <fstream>
  #include <string>
  #include <ctype.h>
  #include "test_classes.hpp"
  #include "router.hpp"
  using std::cout;
  using std::endl;
  using std::vector;
  using std::back_inserter;
  using std::istream_iterator;
  using std::back_insert_iterator;
  using std::string;
  using std::cin;

  using gdfa::router;
  using gdfa::destination;

  bool validate(vector<string> sorted[]);
```

```
int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  test_string_istream.cpp"
    << "\n\nTest Uses istream iterator and type string as input.\n"
    << "**********************************\n\n"
    << "Results:\n\n";

  router<string> r;
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);

  typedef istream_iterator<string> string_in;
  r.distribute(string_in(cin),string_in());

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Number of elements caught by default case: ";
  cout<<sorted[26].size()<<endl;
  if(validate(sorted))
    cout<<"Passed Validation Test"<<endl;
  else
    cout<<"Failed Validation Test"<<endl;


  cout << "\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";


  return 0;
}


bool validate(vector<string> sorted[])
{
  vector<string>::iterator iter;
  for(int i = 0;i<26;i++)
  {
```

```
    iter = sorted[i].begin();
    for(;iter != sorted[i].end();iter++)
    {
      //need to make sure that we are checking case insensitive
      if(tolower((int)(*iter)[0]) != (char)(i + 97))
      {
        cout<<"Wrong letter: "<<*iter<<" should be ";
        cout<<(char)(i + 97)<<" "<<i<<endl;
        return false;
        break;
      }
    }
  }
  iter = sorted[26].begin();
  for(;iter != sorted[26].end();iter++)//check default case
  {
    //we want to check vs lowercase chars
    if(tolower((int)(*iter)[0]) > 96
       && tolower((int)(*iter)[0]) < 123)
    {
      cout<<"Wrong letter: "<<*iter;
      cout<<" should not be in default case"<<endl;
      return false;
      break;
    }
  }
  return true;
}
```

C.9.3 Boundary Conditions Tests

Test with no destinations

"nodestinations.cpp" 282 ≡

```
//testnodestinations.cpp version 1.1
#include "router.hpp"
#include "test_classes.hpp"
#include <iostream>
#include <vector>
#include <iterator>

using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;
```

```
int main()
{
  cout << "\n\n\nTest:  nodestinations.cpp\n\n"
    << "Test router without adding and destinations to it.\n"
    << "***************************************************\n\n"
    << "Results:\n\n";

  router<string> r;
  typedef istream_iterator<string> in_string;
  r.distribute(in_string(cin),in_string());

  cout << "\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";

  return 0;

}
```

Test with no Input

"noinput.cpp" 283 ≡

```
//noinput.cpp version 1.1
#include <iostream>
#include <fstream>
#include <string>
#include <ctype.h>
#include "router.hpp"
#include "test_classes.hpp"
using std::cout;
using std::endl;
using std::vector;
using std::back_inserter;
using std::istream_iterator;
using std::back_insert_iterator;
using std::string;
using std::cin;

using gdfa::router;
using gdfa::destination;

int main(int argc,char *argv[])
{
  cout << "\n\n\nTest:  noinput.cpp\n\n"
    << "Test behavior of router on an empty input range.\n"
    << "****************************************\n\n"
    << "Results:\n\n";

  router<string> r;
```

```
  vector<string> sorted[27];//one for each letter, one for default
  word_sorter<back_insert_iterator<vector<string> > > *dummy[26];
  for(int i = 0; i<26;i++)
  {
    dummy[i] =
     new word_sorter<back_insert_iterator<vector<string> > >
     (back_inserter(sorted[i]),(char)(i + 97));
    r.add_destination(*dummy[i]);
  }
  my_default<back_insert_iterator<vector<string> > >
   dflt(back_inserter(sorted[26]));
  r.set_default(dflt);
  typedef istream_iterator<string> in_string;
  r.distribute(in_string(),in_string());

  for(int i = 0;i<26;i++)
  {
    cout<<(char)(i + 97)<<": "<<sorted[i].size()<<endl;
  }
  cout<<"Default: "<<sorted[26].size()<<endl<<endl;

  cout << "\n*************     END OF RESULTS     ";
  cout<<"*************\n\n";

    return 0;
}
```

Bibliography

[1] G.J. Derge, D.R. Musser, " Operation Counting Adaptors for the C++ Standard Template Library," http://www.cs.rpi.edu/ musser/gsd/opcount/opcount-full-report.pdf. Online available. 4.3

[2] C. Dickens, "A Christmas Carol," *Project Gutenberg Etext*, http://www.ibiblio.org/gutenberg/etext92/carol10.txt, Project Gutenberg, - Urbana, Illinois (USA) (December 1992) 4.4.3

[3] D. Musser, et al., *STL Tutorial and Reference Guide, Second Edition: C++ Programming with the Standard Template Library* Addison-Wesley, Boston, MA, 2001, ISBN 0-201-37923-6. A.4

[4] "Standard Template Library Programmer's Guide," http://www.sgi.com/tech/stl. Online available. 6, 2.1.2