Instructions:

The algorithm was implemented in C++ and compiled under Microsoft Windows 2000, using the Cygwin / GNU g++ compiler using -O3 optimization level.
The computer that the tests were run on has and Intel Xeon 3.06 GHz CPU with 1GB of RAM.

The program can be compiled and run using the included Makefile.

To run the program, put this folder in a place that you can navigate to through Cygwin.
Run Cygwin.
Navigate to the folder (rdsv17o2).
Type make.

This will compile the program and run all of the benchmark examples.

```
#Larry Bush and Brian Bairstow
#Russian Doll Search Make File
#16.412J/6.834J COGNITIVE ROBOTICS
#Prof. Brian Williams

runtests:
    g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
    ./rds_driver.exe ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
    .txt
    ./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out. ↙
    txt
    ./rds_driver.exe ./benchmarks/16wqueens.wcsp ./out/16wqueens.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB0.wcsp ./out/CELAR6-SUB0.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB1-24.wcsp ./out/CELAR6-SUB1-24.wcsp_out.txt ./↙
    out/collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB2.wcsp ./out/CELAR6-SUB2.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/donald.wcsp ./out/donald.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa0432-003.wcsp ./out/ssa0432-003.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-130.wcsp ./out/ssa2670-130.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-141.wcsp ./out/ssa2670-141.wcsp_out.txt ./out/ ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_1.wcsp ./out/vcsp25_10_21_85_1.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_2.wcsp ./out/vcsp25_10_21_85_2.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_3.wcsp ./out/vcsp25_10_21_85_3.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_4.wcsp ./out/vcsp25_10_21_85_4.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_5.wcsp ./out/vcsp25_10_21_85_5.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_1.wcsp ./out/vcsp25_10_25_87_1.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_2.wcsp ./out/vcsp25_10_25_87_2.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_3.wcsp ./out/vcsp25_10_25_87_3.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_4.wcsp ./out/vcsp25_10_25_87_4.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_5.wcsp ./out/vcsp25_10_25_87_5.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_1.wcsp ./out/vcsp30_10_25_48_1.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_2.wcsp ./out/vcsp30_10_25_48_2.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_3.wcsp ./out/vcsp30_10_25_48_3.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_4.wcsp ./out/vcsp30_10_25_48_4.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp30_10_25_48_5.wcsp ./out/vcsp30_10_25_48_5.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_1.wcsp ./out/vcsp40_10_13_60_1.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_2.wcsp ./out/vcsp40_10_13_60_2.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_3.wcsp ./out/vcsp40_10_13_60_3.wcsp_out.↙
```

```
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_4.wcsp ./out/vcsp40_10_13_60_4.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp40_10_13_60_5.wcsp ./out/vcsp40_10_13_60_5.wcsp_out.↙
    txt ./out/collective_out.txt


rds_compile_and_run:
    g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
    ./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
    .txt

clear:
    ./rds_driver.exe ./out/collective_out.txt

test:
    g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
    ./rds_driver.exe ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/        ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/        ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
    .txt


runtests2:
    g++ -O3 rds_driver.cpp -o rds_driver.exe -D_GLIBCPP_CONCEPT_CHECKS
    ./rds_driver.exe ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out. ↙
    txt
    ./rds_driver.exe ./benchmarks/4wqueens.wcsp ./out/4wqueens.wcsp_out.txt ./out/        ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/8wqueens.wcsp ./out/8wqueens.wcsp_out.txt ./out/        ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/zebra.wcsp ./out/zebra.wcsp_out.txt ./out/collective_out↙
    .txt
    ./rds_driver.exe ./benchmarks/16wqueens.wcsp ./out/16wqueens.wcsp_out.txt ./out/      ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB0.wcsp ./out/CELAR6-SUB0.wcsp_out.txt ./out/  ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB1-24.wcsp ./out/CELAR6-SUB1-24.wcsp_out.txt ./↙
    out/collective_out.txt
    ./rds_driver.exe ./benchmarks/CELAR6-SUB2.wcsp ./out/CELAR6-SUB2.wcsp_out.txt ./out/  ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/donald.wcsp ./out/donald.wcsp_out.txt ./out/            ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/send.wcsp ./out/send.wcsp_out.txt ./out/collective_out. ↙
    txt
    ./rds_driver.exe ./benchmarks/ssa0432-003.wcsp ./out/ssa0432-003.wcsp_out.txt ./out/  ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-130.wcsp ./out/ssa2670-130.wcsp_out.txt ./out/  ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/ssa2670-141.wcsp ./out/ssa2670-141.wcsp_out.txt ./out/  ↙
    collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_1.wcsp ./out/vcsp25_10_21_85_1.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_2.wcsp ./out/vcsp25_10_21_85_2.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_3.wcsp ./out/vcsp25_10_21_85_3.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_4.wcsp ./out/vcsp25_10_21_85_4.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_21_85_5.wcsp ./out/vcsp25_10_21_85_5.wcsp_out.↙
    txt ./out/collective_out.txt
    ./rds_driver.exe ./benchmarks/vcsp25_10_25_87_1.wcsp ./out/vcsp25_10_25_87_1.wcsp_out.↙
    txt ./out/collective_out.txt
```

```
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_2.wcsp ./out/vcsp25_10_25_87_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_3.wcsp ./out/vcsp25_10_25_87_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_4.wcsp ./out/vcsp25_10_25_87_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp25_10_25_87_5.wcsp ./out/vcsp25_10_25_87_5.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_1.wcsp ./out/vcsp30_10_25_48_1.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_2.wcsp ./out/vcsp30_10_25_48_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_3.wcsp ./out/vcsp30_10_25_48_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_4.wcsp ./out/vcsp30_10_25_48_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp30_10_25_48_5.wcsp ./out/vcsp30_10_25_48_5.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_1.wcsp ./out/vcsp40_10_13_60_1.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_2.wcsp ./out/vcsp40_10_13_60_2.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_3.wcsp ./out/vcsp40_10_13_60_3.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_4.wcsp ./out/vcsp40_10_13_60_4.wcsp_out.↙
	txt ./out/collective_out.txt
	./rds_driver.exe ./benchmarks/vcsp40_10_13_60_5.wcsp ./out/vcsp40_10_13_60_5.wcsp_out.↙
	txt ./out/collective_out.txt
```

```
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// ____________________________________________
//
// ____________________________________________
//  Russian Doll Search - Main program.

#include<iostream>
#include<iomanip>
#include<string>
#include<fstream>
#include<vector>
#include<string>
#include<sstream>
#include "variable.h"
#include "variables.h"
#include "tuple.h"
#include "tuples.h"
#include "constraints.h"

//uncomment the next line to see the full walk through output
//#define DEBUG_OUTPUT_PROBLEM_FILE
//timer code
//#include "long_timer.h"
#include "timer.h"


using namespace std;


variables bnb(constraints C, std::ostream & out, int initial, vector<double> & future_cost
    , variables sa, vector<unsigned long long int> & operations, time_t & time_limit,int &
     variable_counter) {


    if(time_limit < time(0)) { return variables(); }

    variables ca = C.initialize_assignment(initial);

    //  double old_sa_eval = C.evaluate( sa,operations );// the following line is quicker,
     but this one would work also
    double old_sa_eval = future_cost[initial+1];
    C.initialize_upper_bound(sa, ca, operations);
    double new_sa_eval = C.evaluate(sa,operations );
    if(new_sa_eval==old_sa_eval) {
        future_cost[initial] = new_sa_eval;
        return sa;
    }

    double ub=C.get_upper_bound();
    double lb=0;
    double eval;

    while(1) {
        if(time_limit < time(0)) { return variables(); }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        ca.print(out);
```

```
#endif

        if(ca.empty()){
            future_cost[initial] = ub;
            return sa;
        }
        eval = C.evaluate(ca,operations, future_cost[initial+ca.size()], ub  );
        //      eval = C.evaluate(ca,operations);// the above line is faster, but this one
     would work also
        lb = eval + future_cost[initial+ca.size()];
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Evaluates to:    " << eval << endl;
        out << "Current lb:      " << lb << endl;
        out << "Current ub:      " << ub << endl;
        out << "-----------------------------------------------------"<<endl;
#endif

        // terminal case
        // change the ub if applicable
        // even if it is ok, go to the right/up
        if(C.is_last_variable(ca)){// last variable

            // if last variable, and eval < ub, then set ub = eval (ub is current best
    assignment cost)
            if(lb<ub){
                ub=lb;
                sa = ca;
            }

            if(C.is_last_value(ca)){// last value
                ca = C.back_up(ca);
                operations[2]+=1;

            } else {// not last value
                // if last variable and if not last value, try next value
                ca = C.get_next_value(ca);
                operations[2]+=1;
            }

            ////////////////////////////////////////
        } else {// not last variable yet

            // if eval >= ub && not last value, then try next value or go up(if last
    value)
            // if eval < ub, then try next variable

            if(lb<ub){//good, then try next variable
                ca = C.get_next_variable(ca);
                operations[2]+=1;
            } else {// if eval >= ub && not last value, then try next value or go up(if
    last value)
                if(C.is_last_value(ca)){// last value
                    ca = C.back_up(ca);
                    operations[2]+=1;
                } else {// not last value // not last variable
                    //  if not last value, try next value
                    ca = C.get_next_value(ca);
                    operations[2]+=1;
                }


            }



        }
```

```
    }

}


////////////////////////////////////////////////////////////////////////////////

double rds(constraints C, ostream & out, vector<unsigned long long int> & operations,
    time_t & time_limit,int & variable_counter)
{
    int n = C.get_number_of_variables();
    vector<double> future_costs(n+1, -1);
    variables sa;  //subproblem assignment

    future_costs[n] = 0;

    for(int i = n-1; i >= 0; i--)
    {
        sa = bnb(C, out, i, future_costs, sa, operations, time_limit, variable_counter);
        if(time_limit < time(0)) { return -1; }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Subproblem Optimum:   " << future_costs[i] << endl;
        sa.print(out);
        out << endl ;
#endif

        if(sa.empty()) {
            return -2;
        }

        if(future_costs[i] >= C.get_global_upper_bound())
        {
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
            out << endl << "Failure";
            //if for any subproblem that cost is worse than the global upper bound
            // then it is going to fail on any macro problem, so return flag
#endif
            return -2;//flag
        }
    }
    //#ifdef DEBUG_OUTPUT_PROBLEM_FILE
    out << endl << endl << "Value:    " << future_costs[0] << endl;
    sa.print(out);
    //#endif

    return future_costs[0];

}


////////////////////////////////////////////////////////////////////////////////


// This function converts doubles to strings.

string ltos(unsigned long long int d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
string dtos(double d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
int main(int argc, char *argv[] ) { //  start of main, with input of file argument and the
```

```
     number of arguments

    vector<unsigned long long int> operations_old;
    vector<unsigned long long int> operations;

    int time_duration = 599;

    if (argc == 2 ) {  //  if there are 2 arguments
        //open out file
        ofstream collective_out;
        collective_out.open(argv[1],ios::out);//write / clear

        if (collective_out == NULL) {// error of out file could not be opened
            cerr << "Error, could not clear the collective outfile" << endl;
            exit(0);
        }

        // 0 is tuple    count
        // 1 is variable count
        // 3 is nodes visited
        collective_out <<"Problem_Name  "<<"  Number of Variables"<<"  Number of Nodes"<< ↙
    "Run_Time(seconds)"<<"  Evaluated Tuple"<<"  Evaluated Variables"<<"  Nodes Visited"< ↙
    <"  Optimal_Value"<<endl;
        collective_out.close();//close output file
        cout<< "Cleared the file : " << argv[1] << endl << endl ;
        cout <<"Problem_Name  "<<"Run_Time(seconds)"<<"  Operation_Counts"<<endl;
        exit(0);

    }

    if (argc != 4 ) {  //  if there are not 4 arguments
        // Error message and command line argument instructions
        cout<< "Command line arguments:\n" << "example:\n" << "problem_in_file_name       ↙
    problem_out_file_name collective_out.txt\n";
        exit(0);
    } else {
        cout<< "There are 4 Command line arguments: "<< argv[0] << " and " << argv[1] << "↙
     and " << argv[2] << " and " << argv[3] << ".\n" << endl;

    }

    //open out file
    ofstream out;
    out.open(argv[2],ios::out);
    if (out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the file file" << endl;
        exit(0);
    }

    //open out file
    ofstream collective_out;
    collective_out.open(argv[3],ios::app);//append
    if (collective_out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the collective outfile" << endl;
        exit(0);
    }


    string input_file_name = argv[1];
    cout << "Input Filename = "<<input_file_name<<"\n";

    ifstream in;
    in.open(argv[1],ios::in);
    if (in == NULL) {// error message if in file could not be opened
        cerr << "Error, could not open the output file" << endl;
        exit(0);
    }
```

```
    // Instantiate variables for file input
    string problem_name;
    int number_of_variables;
    int maximum_domain_size;
    int number_of_constraint_groups;
    int global_upper_bound;

    // Read in the problem header:
    in>>problem_name>>number_of_variables>>maximum_domain_size>>
    number_of_constraint_groups>>global_upper_bound;
    // Output the problem header:
    cout<<problem_name<<" "<<number_of_variables<<" "<<maximum_domain_size<<" "<
    <number_of_constraint_groups<<" "<<global_upper_bound<<endl;

    variables X; // X is a set of variables

    // Read in the domain sizes and initialize the variables.
    for(int count_var_index = 0; count_var_index < number_of_variables ; count_var_index+
    +){
        int next_variable_domain_size;
        in>>next_variable_domain_size;
        //      cout<<next_variable_domain_size<<endl;

        int domain_value = -1;

        //  variable next_variable(count_var_index, next_variable_domain_size,
    domain_value);
        X.insert(   variable(count_var_index, next_variable_domain_size, domain_value) );
        //      cout<<next_variable<<endl;

    }

    X.print(out);

    constraints C(number_of_variables, maximum_domain_size,number_of_constraint_groups,
    global_upper_bound, X);

    //  vector<constraint> constraint_vector;

    int constraint_arity;
    int next_variables_in_constraint;
    int default_cost;
    int number_of_tuples;
    int next_tup_value;
    int non_default_value;

    // Read in each constraint group:
    for(int count = 0; count < number_of_constraint_groups ; count++){

        ////////////////////////////////////////////////////////////////////////
        // Read constraint group header line (num vars, each var, default cost, num
    tuples).

        // Read number of variables in the constraint (constraint arity).
        in>>constraint_arity;
        // Read the indecies of variables in the constraint.
        // Note: we have to remember this order, so we can apply the right value
    assingment to the right variable.
        // This should be an iterable set of variables.
        vector<int> cX_indecies; // cX is the subset of variable indecies in the
    constraint.
        for(int count_constraint_arity = 0; count_constraint_arity < constraint_arity ;
    count_constraint_arity++){
            in>>next_variables_in_constraint;
            cX_indecies.push_back(next_variables_in_constraint);
        }
```

```
        // Read the Default cost value.
        in>>default_cost;
        // Read the Number of tuples (variable set assignments) without the default cost.
        in>>number_of_tuples;

        // create the constraint
        //constraint temp_constraint(constraint_arity, default_cost, number_of_tuples,
    cX_indecies);

        // create the tuple set
        tuples temp_tuples(constraint_arity, default_cost, number_of_tuples, cX_indecies);
        //  cout<<number_of_tuples_without_default_cost_value<<endl;
        //
        ////////////////////////////////////////////////////////////////////////////////
    //

        ////////////////////////////////////////////////////////////////////////////////
    //
        // Read in each constraint tuple (variable set assignments without the default
    cost).
        for(int count_number_of_tuples = 0; count_number_of_tuples < number_of_tuples ;
    count_number_of_tuples++){
            variables c_vars; // a set of variables in the sub-constraint

            // Read each the tuple:
            // Here we should iterate through the variable set.
            // Then create a variable / value "pairing" which goes into a value object.
            // The value objects are the combined into a constraint object, which also
    contains the default value.

            for(vector<int>::iterator count_icx = cX_indecies.begin(); count_icx !=
    cX_indecies.end() ; count_icx++ ){
                //          for(int count_number_of_variables_in_constraint= 0;
    count_number_of_variables_in_constraint < number_of_variables_in_constraint ;
    count_number_of_variables_in_constraint++){

                // read the n tuple assignments
                in>>next_tup_value;

                c_vars.insert(  variable(*count_icx, X[*count_icx].get_domain_size(),
    next_tup_value) );
            }

            // read the non-default cost.
            in>>non_default_value;
            // Create a new value.
            tuple temp_tuple = tuple(c_vars,non_default_value);
            temp_tuples.insert(temp_tuple);
        }
        C.insert_tuples(temp_tuples);
    }// end read each constraint group

    C.sort_tuples_by_num_non_default_cost();



    in.close();//close input file

    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    /////                                          /////
    /////               Time Trials                /////
    /////                                          /////
    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    /////         To seconds trials first          /////
    ////////////////////////////////////////////////////
```

```
    srand(time(0));//timer
    time_t initial, final;
    initial = time(0);
    time_t time_limit = initial + time_duration;
    int memory_used = 0;
    int variable_counter = 0;
    //operations=0;
    operations.clear();
    operations.push_back(0);//
    operations.push_back(0);//
    operations.push_back(0);//

    ////////////////////////
    ///     run rds      ///
    ////////////////////////
    double optimal_value = rds(C, out,operations, time_limit, memory_used);
    cout<<"hi"<<endl;
    final = time(0);
    double elapsed_seconds = difftime(final,initial);

    if(elapsed_seconds> 1){

        ////////////////////
        //  output lines  //
        ////////////////////
        // parse the problem name
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");
        ////////////////////////////////////

        string optimal_value_string=dtos(optimal_value);
        if(optimal_value == -1){ optimal_value_string = "time expired"; }
        if(optimal_value == -2){ optimal_value_string = "failure";}



        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.                ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        ////////////////////


    } else {
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        // if it is a short time frame, then use this more precise method:
        // timer declarations
        unsigned int reps;
        unsigned long N;
        N=10;
        reps = 10;
        srand(time(0));//timer
```

```
        timer tim;
        // Compute the baseline time for N
        tim.start_baseline(reps);
        do {
            ////////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            ////////////////////////////////
            // nothing //
            // End of Baseline Operations
            ////////////////////////////////
        } while (tim.check());

        tim.report(false);

        time_t time_limit = time(0) + time_duration;
        tim.start(reps, N);
        do {

            ////////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            ////////////////////////////////

            // End of Baseline Operations
            ////////////////////////////////
            // Main Timed Operation
            // this is the actual timing
            //      for(int count_rds=1;count_rds<20000;count_rds++) {
            //operations=0;
            // 0 is tuple    count
            // 1 is variable count
            // 3 is nodes visited
            operations.clear();
            operations.push_back(0);//
            operations.push_back(0);//
            operations.push_back(0);//
            double optimal_value = rds(C, out,operations, time_limit, variable_counter);
            ////////////////////////////////////
        }
        while (tim.check());

        tim.report(false);

        cout<<"longer that 3 seconds"<<endl;
        elapsed_seconds = tim.report();
        tim.report(false);

        ////////////////////
        //  output lines  //
        ////////////////////
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");

        string elapsed_seconds_string=dtos(elapsed_seconds);
        string optimal_value_string=dtos(optimal_value);
        if(optimal_value== -1){ optimal_value_string = "time expired";}
        if(optimal_value== -2){ optimal_value_string = "time expired";}

        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
```

```
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) <
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) <
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;

        //sa.print(collective_out);//you can print out the final assignment again here if
    you wish.
        ////////////////////


    }

    out.close();//close input file

    return 0;

}
```

```
//  File:  constraints.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// ____________________________________
//
//

#ifndef _constraints_h_
#define _constraints_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
#include "variables.h"
#include "tuples.h"
using namespace std;

// constraints class
//
//
class constraints {


public:
    //constraints() {}  // default constructor // not needed or wanted
    // Constructor - assigns all variable attributes
    constraints(int number_of_variables_in, int maximum_domain_size_in, int
    number_of_constraint_groups_in, int  global_upper_bound_in, variables  X_in)
        : number_of_variables(number_of_variables_in), maximum_domain_size
    (maximum_domain_size_in),number_of_constraint_groups(number_of_constraint_groups_in),
    global_upper_bound(global_upper_bound_in), X(X_in)
    {
        global_lower_bound = 0;
    }

    void insert_tuples(tuples ts_in){ // insert a variable object into the container
        tuples_vector.push_back(ts_in);
    }


    int get_number_of_constraint_groups()   const {return number_of_constraint_groups;}
    int get_number_of_variables()   const {return number_of_variables;}

    // assessor operator, returns player k
    tuples operator[](int k) const
    {
        return tuples_vector[k];
    }

    int get_number_of_nodes() {
        vector<variable> vl = X.get_variable_list();

        int number_of_nodes=1;
        for(vector<variable>::iterator i = vl.begin(); i != vl.end(); i++)
        {
            number_of_nodes *= i->get_domain_size();// metric counter
        }
        return number_of_nodes;
```

```
    }

    int size() { return tuples_vector.size(); } // returns the number of tuple in the container

    double initialize_upper_bound(variables & sa, variables ca, vector<unsigned long long int> & operations) {

        upper_bound = 999999999;
        double temp;
        variables tempvar;
        int i;

        for(i = 0; i < sa.size(); i++)
        {
            ca.insert(sa[i]);
        }

        for(i = 0; i < ca[0].get_domain_size(); i++)
        {
            temp = evaluate(ca,operations);
            if(temp < upper_bound)
            {
                upper_bound = temp;
                tempvar = ca;
            }
            if(i < ca[0].get_domain_size()-1)
                ca = increment_first_value(ca);
            operations[2]+=1;
        }

        sa = tempvar;
        //if(upper_bound==sa_eval) {upper_bound -= 1;   cout<<"ub == sa_eval"<<endl;} // obsolete not
        return upper_bound;
    }

    double get_upper_bound() {
        return upper_bound;
    }
    double get_global_upper_bound() {
        return global_upper_bound;
    }



    // print tuple container
    void print(std::ostream & out) {

        out << "-----------------------------------------------------"<<endl;
        out << "Tuple Sets Statistics: \n";
        out << "-----------------------------------------------------"<<endl;
        out << "Number of Variables:  "<< number_of_variables << endl;
        out << "Domain Sizes:         "<< maximum_domain_size << endl;
        out << "Number of Tuple Sets: "<< number_of_constraint_groups << endl;
        out << "Global Upper Bound:   "<< global_upper_bound << endl;
        out << "-----------------------------------------------------"<<endl;
        out << "Print all Tuple Sets: \n";
        out << "-----------------------------------------------------"<<endl;

        for( int i = 0 ; i < size() ; i++ ) {
            tuples_vector[i].print(out);
        }
    }

    bool is_last_value(variables & ca_in)
    {
```

```
        if(ca_in.back().get_domain_value()==(ca_in.back().get_domain_size()-1)){
            return true;
        } else {
            return false;
        }
    }


    bool is_last_variable(variables & ca_in)
    {
        if(ca_in.back().get_var_index()==X.back().get_var_index()){
            return true;
        } else {
            return false;
        }
    }

    variables initialize_assignment(int initial ){
        current_assignment = variables();
        next_variable = initial;
        next_value = 0;
        current_assignment.insert(variable(next_variable,X[next_variable].get_domain_size ↙
    (),next_value ));
        return current_assignment;
    }

    variables increment_first_value(variables ca_in)
    {
        int n = ca_in.size();
        vector<variable> temp(n);

        for(int i = n - 1; i > 0; i--)
        {
            temp[i] = ca_in[i];
            ca_in.remove();
        }
        ca_in = get_next_value(ca_in);

        for(int count_n = 1; count_n < n; count_n ++)
        {
            ca_in.insert(temp[count_n]);
        }
        return ca_in;
    }

    variables get_next_value(variables & ca_in){

        variable temp = ca_in.back();
        ca_in.remove();
        if(!temp.increment_domain_value()) {
            cout<<"Error, domain exceeded!\n";
        }
        ca_in.insert(temp);

        return ca_in;

        return ca_in;
    }


    variables get_next_variable(variables ca_in){

        // insert the next variable if I can
        ca_in.insert(variable(X[ca_in.back().get_var_index()+1].get_var_index(),X[ca_in.  ↙
    back().get_var_index()+1].get_domain_size(),0));
        return ca_in;
    }
```

```
variables back_up(variables & ca_in){

    if(ca_in.size()==0){return ca_in;}
    ca_in.remove();
    if(ca_in.size()==0){return ca_in;}

    if(ca_in.back().increment_domain_value()) {
        ca_in = get_next_value(ca_in);
    } else {
        ca_in = back_up(ca_in);
    }
    return ca_in;
}


variables get_next_assignment(){

    // insert the next variable if I can
    if(next_variable+1 != number_of_variables){
        next_variable++;
        current_assignment.insert(variable(next_variable,X[next_variable].
get_domain_size(),0));
        return current_assignment;
    }
    // else increment the value
    while( 1 ){

        variable temp = current_assignment.back();
        current_assignment.remove();
        next_variable--;

        if(!temp.next_domain_value()){
            current_assignment.insert(temp);

            next_variable++;
            return current_assignment;
        }
    }
    return current_assignment;
}


double evaluate(variables & ca_in,vector<unsigned long long int> & operations, double
& additional_cost, double & upper_bound){

    //operations += ca_in.size();// obsolete counter

    double return_value = 0;
    for( int i = 0 ; i < size() ; i++ ) {//size is the number of tuple sets

        return_value += tuples_vector[i].evaluate(ca_in, operations);
        if( (return_value + additional_cost) > upper_bound) { return return_value; }
    }
    return return_value;
}


double evaluate(variables ca_in,vector<unsigned long long int> & operations){

    //operations += ca_in.size();// obsolete counter

    double return_value = 0;
    for( int i = 0 ; i < size() ; i++ ) {
        return_value += tuples_vector[i].evaluate(ca_in, operations);
    }
```

```
        return return_value;
    }

    variables bind_next_assignment(){
        // this function goes to the next value, rather than going deeper
        // increment the value
        while( 1 ){

            variable temp = current_assignment.back();
            current_assignment.remove();
            next_variable--;

            if(!temp.next_domain_value()){
                current_assignment.insert(temp);

                next_variable++;
                return current_assignment;
            }
        }
        return current_assignment;
    }
    // constraint sorts
    void sort_tuples_by_num_non_default_cost(){
        sort(tuples_vector.begin(),tuples_vector.end(),less_size());
    }
    class less_size {
    public:
        bool operator()(tuples x, tuples y) const { return (x.get_number_of_tuples() < y. ↙
    get_number_of_tuples()); }
    };


private:

    int number_of_variables;
    int maximum_domain_size;
    int number_of_constraint_groups;
    double global_upper_bound;
    double upper_bound;
    double global_lower_bound;
    variables X;
    vector<tuples> tuples_vector;


    variables current_assignment;
    int next_variable;
    int next_value;


};

#endif
```

```
//  File:  tuples.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bush12, bairstow }@mit.edu
// ________________________________________
//
//  tuples.h -  Contains a set of tuples and algorithm pertaining to them.
//              Stores a vector of tuples.
//

#ifndef _tuples_h_
#define _tuples_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
using namespace std;

// tuples class
//
//
class tuples {


public:
    tuples() {}  // default constructor
    // Constructor - assigns all variable attributes
    tuples(int constraint_arity_in, int default_cost_in, int number_of_tuples_in, vector
    <int> cX_indecies_in)
        : constraint_arity(constraint_arity_in), default_cost(default_cost_in),
    number_of_tuples(number_of_tuples_in), cX_indecies(cX_indecies_in)
    {
    }

    void insert(tuple t_in){ // insert a variable object into the container
        tuple_vector.push_back(t_in);
    }

    //remove a tuple from the container
    void remove(){
        tuple_vector.pop_back();
    }

    int get_constraint_arity()  const {return constraint_arity;}
    double get_default_cost()   const {return default_cost;}
    int get_number_of_tuples()  const {return number_of_tuples;}
    vector<int> get_cX_indecies()   const {return cX_indecies;}

    // assessor operator, returns player k
    tuple operator[](int k) const
    {
        return tuple_vector[k];
    }


    int size() { return tuple_vector.size(); } // returns the number of tuple in the
    container

    // print tuple container
```

```
    void print(std::ostream & out) {
        out << "-------------------------------------------------------"<<endl;
        out << "-----                Next Tuple Set              -----"<<endl;
        out << "-------------------------------------------------------"<<endl;
        out << "Tuple Set Arity:          "<<constraint_arity<<endl;
        out << "Tuple Set Variables:      ";
        for(vector<int>::iterator iter = cX_indecies.begin();iter != cX_indecies.end();
    iter++){ out << *iter << "  ";}
        out << endl;
        out << "Tuple Set Default Value: "<<default_cost<<endl;
        out << "Tuple Set Size:           "<<number_of_tuples<<endl;

        out << "-------------------------------------------------------"<<endl;

        out << "Print each Tuple in this Tuple Set: \n";
        out << "-------------------------------------------------------"<<endl;
        for( int i = 0 ; i < size() ; i++ ) {
            tuple_vector[i].print(out);
            out << "-------------------------------------------------------"<<endl;
        }
    }

    bool constraintdefined(variables & ca_in)
        //This returns true if all the variables necessary for the constraint have been
    defined.
    {
        for(int i = 0; i < constraint_arity; i++)
        {
            if(!ca_in.isIn(cX_indecies[i]))
                return false;
        }
        return true;
    }


    double evaluate(variables & ca_in, vector<unsigned long long int> & operations)
    {
        double temp;

        if(constraintdefined(ca_in)){


            for( int i = 0 ; i < size() ; i++ )//Size is the number of tuple objects.
            {

                // 0 is tuple    count
                // 1 is variable count
                // 3 is nodes visited
                operations[0] += 1;// for each tuple evaluated
                temp = tuple_vector[i].evaluate(ca_in, operations);
                //operations[1] += tuple_vector[i].size();// count each variable in each
    tuple set evaluated
                if(temp != -1)
                {
                    return temp;
                }
            }
            return default_cost;
        }
        return 0;   //This must be the neutral value
    }

private:
    int constraint_arity;
    double default_cost;
    int number_of_tuples;
    vector<tuple> tuple_vector;
```

```
    vector<int> cX_indecies;
};

#endif
```

```
//  File:  tuple.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// ________________________________________
//
//
//  tuple.h -   Contains a tuple class.
//          Stores a tuple read in from file.
//          A tuple is an assignment to a set of variables.
//          More precisely, it is an ordered set of values
//          assigned to the ordered set of variables.
//
//          In this class, there is a set of vaiables that
//          have an assignment.
//

#ifndef _tuple_h_
#define _tuple_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
using namespace std;

// tuple class
//
//
class tuple {


public:
    tuple() {}  // default constructor
    // Constructor - assigns all variable attributes
    tuple( variables c_vars_in, int non_default_value_in )
        : c_vars(c_vars_in), non_default_value(non_default_value_in)
    {
    }




    double get_non_default_value()  const {return non_default_value;}

    // assessor operator, returns player k
    variable operator[](int k) const
    {
        return c_vars[k];
    }


    int size() { return c_vars.size(); } // returns the number of variable in variables

    // print draft list container
    void print(std::ostream & out) {
        out << "Non-Default Tuple Cost:  " << non_default_value << endl;
        out << "-------------------------------------------------"<<endl;
        c_vars.print(out);
```

```
    }


    double evaluate(variables & ca_in, vector<unsigned long long int> & operations){

        if(c_vars.matches(ca_in, operations)){

            return non_default_value;

        } else {
            return -1;
        }
    }


private:
    variables c_vars;
    double non_default_value;
    int constraint_arity;

};



#endif
```

```
//  File:  variables.h
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// ________________________________________
//

//  variables.h -   Contains a variables class.
//          Stores a vector of variables read in from file.
//          Performs functions on the vector.
//          This is a wrapper for a vector.
//          The vector stores the list of all of the variables to pick from.
//

#ifndef _variables_h_
#define _variables_h_
#include <string>
#include <vector>
#include <algorithm>
#include "variable.h"
#include <iostream>
#include <fstream>
using namespace std;


class variables {


public:
    variables() {}  // default constructor

    void insert(variable v){ // insert a variable object into the container
        variable_list.push_back(v);
    }

    // remove a variable object from the container
    void remove(){
        variable_list.pop_back();
    }

    // assessor operator, returns player k
    variable operator[](int k) const
    {
        return variable_list[k];
    }


    int size() { return variable_list.size(); } // returns the number of variables in the ↙
    container

    vector<variable> get_variable_list() { return variable_list; } // returns the list of ↙
    variables

    // print container
    void print(std::ostream & out) {

        out << "-----------------------------------------------------"<< endl;
        out << "Print all Variables: \n";
        for( int i = 0 ; i < size() ; i++ ) {
```

```
            out  <<"Variable Index: " << variable_list[i].get_var_index() << ", Domain
    Size: " << variable_list[i].get_domain_size() << ", Domain Value: " << variable_list
    [i].get_domain_value() <<  endl;
        }
    }


    bool isIn(int & index)
    {
        for(int i = 0; i < size(); i++)
        {
            if(index == variable_list[i].get_var_index())
                return true;
        }
        return false;
    }

    bool isMatched(variable & a)
    {
        for(vector<variable>::iterator i = variable_list.begin(); i != variable_list.end()
    ; i++)
        {
            if(a.get_var_index() == i->get_var_index() && a.get_domain_value() == i->
    get_domain_value())
                return true;
        }

        return false;
    }


    bool matches(variables & ca_in, vector<unsigned long long int> & operations)
        //This returns true if the variable assignments (ca_in) contains the tuple
    assignments.(c_vars)
    {
        for(vector<variable>::iterator i = variable_list.begin(); i != variable_list.end()
    ; i++)
        {
            // 0 is tuple    count
            // 1 is variable count
            // 2 is nodes visited
            operations[1]+=1;
            if(!ca_in.isMatched(*i)) {

                return false;
            }
        }

        return true;
    }


    bool empty() { return variable_list.empty(); }

    variable back() {
        return variable_list.back();
    }


private:
    vector<variable> variable_list; // private vector data member

};
```

```
#endif
```

```
//
//  File:  variable.h
//
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// ____________________________________
//
//  variable.h -    Contains a variable class.
//          variable object stores variable information,
//          and performs formated output and other
//          functions on variable data.

#ifndef _variable_h_
#define _variable_h_
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
//
class variable {

    friend std::ostream& operator << ( std::ostream& outstr,
        const variable& v );

public:
    variable(){};  // default constructor --

    // Constructor - assigns all variable attributes
    variable( int var_index_in, int domain_size_in, int domain_value_in  )
        : var_index(var_index_in),domain_size(domain_size_in), domain_value
    (domain_value_in)
    {

    }

    //*********************************************************
    //*********************************************************
    // Get functions to retrieve a statistic about the variable.
    //*********************************************************
    //*********************************************************

    int get_domain_value()  const {return domain_value;} // returns variable value
    int get_var_index()     const {return var_index;} // returns variable value
    int get_domain_size()   const {return domain_size;} // returns variable value

    bool increment_domain_value()    {
        domain_value++;
        if(domain_value >= domain_size) {
            domain_value--;
            return false; // check for going to far
        } else {
            return true;
        }
    }


    bool max_domain_value() const {return domain_value==domain_size;} // returns variable
    value
```

```
    bool next_domain_value(){
        domain_value++;
        return max_domain_value();
    } // returns variable value


private:
    //*********************************
    //*********************************
    //      Private Data Members
    //*********************************
    //*********************************
    int var_index;
    int domain_size;
    int domain_value;

};

//*********************************************************
// Overloaded non-member << operator
//*********************************************************
std::ostream &
operator << ( std::ostream & outstr, const variable & v )
{
    outstr <<"Variable Index: " << v.get_var_index() << ", Domain Size: " << v.
    get_domain_size() << ", Domain Value: " << v.get_domain_value() <<  endl;
    return outstr;
}


#endif
```