Published: 11/18/66

Identification

Procedures for dynamic storage management
areamk_, area_$redef, allocate_, freen_
D. B. Wagner and M. D. McIlroy

Purpose

The implementation of the PL/I controlled and based storage classes, and of varying strings, uses the procedures described here to manipulate areas and to obtain storage from these areas and return such storage when it is no longer needed.

Manipulation of Areas

See BP.2 for a description of PL/I areas: an area has the form of a fixed-point array.

The allocation routines require that the data in the area contain certain overhead information: see Implementation, below. The areamk_ routine is used to initialize an area to contain this information. It could be called explicitly by a PL/I program for some special purpose, but it is normally called where needed by the compiled code. The call is:

```
call areamk_ (place);
```

where the arguments are declared,

```
dcl place area ((*));
```

Areamk_ initializes place according to the size found in the dope, as described in Implementation, below.

If an area occupies the highest locations currently in use in a segment it is possible to grow the segment and reinitialize the area to have a new (larger) size. An example of where this is useful might be in the File System, where a directory segment is treated as an area. The entry area_$redef is used to reinitialize an area for a larger size. This call is

```
call areamk_$redef(newsize,place);
```

with declarations

```
dcl newsize fixed bin (18),
    place area((*));
```

Areamk_$redef carefully preserves the current allocations in place while adjusting key items to make it seem to the

allocating routines that place is newsize words long. Naturally newsize must be greater than the current size of the area to guarantee that the redefinition will succeed.

Area Management

The procedures allocate_ and freen_ are used for allocating and freeing storage in areas. Their calls are:

```
call allocate_ (n, place, p);

call freen_ (p);
```

with declarations,

```
dcl n fixed bin (18),

    place area ((*)),

    p ptr;
```

Allocate_ finds a block of n consecutive words, starting at an even location, in the area place and stores a pointer to the first word of this block into p. If no such block exists in place, alloc_ executes a

```
signal area;
```

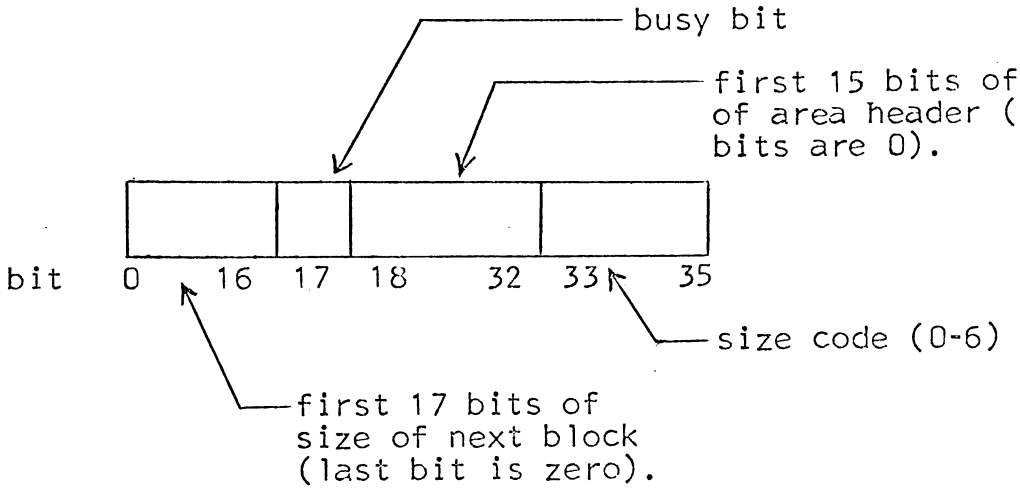
Freen_ returns a block to the area from which it was allocated. The pointer p points to the first word of the block and must have been obtained from a previous call to allocate_.

Implementation

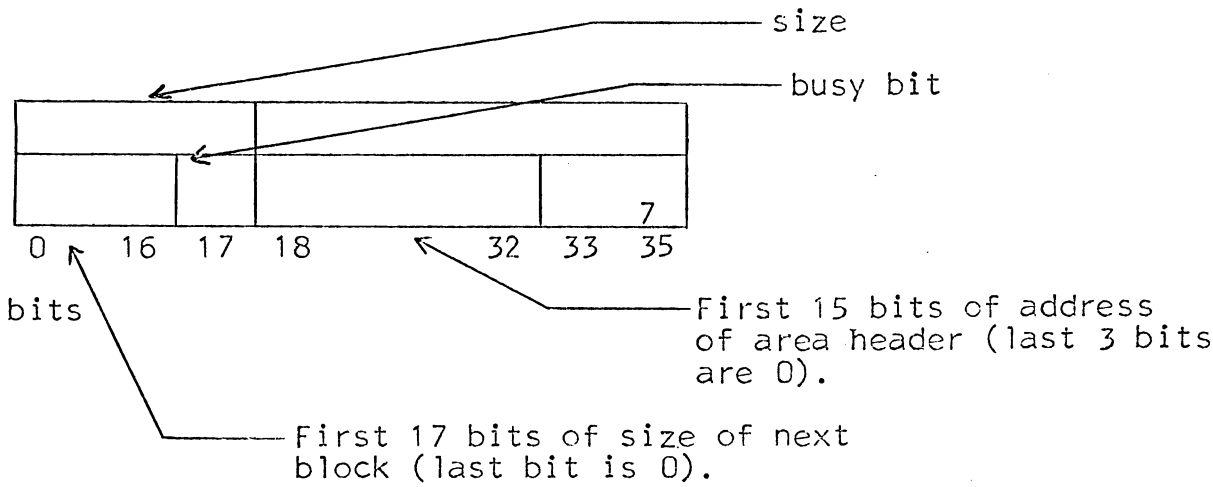
In the course of use an area is divided into: (1) busy blocks, which are presently in use, (2) idle blocks, which have been busy and are now free to be reallocated, and (3) virgin territory.

Blocks are threaded together in the following way. Each block has a trailer of one or two words depending upon the size of the block. Wherever block size is mentioned in this discussion it includes this trailer.

A short trailer is for blocks shorter than 14 words and has the form

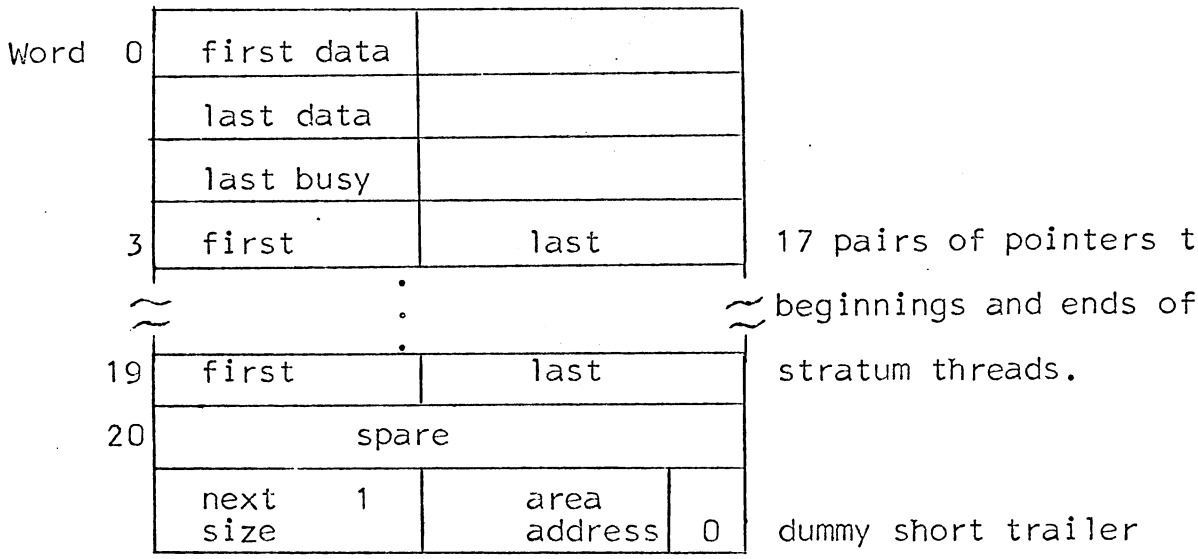


The size of the block is 2* "size code" + 2. If the "size code" field above is 7, then the trailer is a long trailer, in the form



Idle blocks are classified into seventeen strata according to their size. Stratum i is a threaded list of blocks with sizes between 2**i and 2**(i+1)-1. Each idle block has in its first location a chaining word with a back-pointer in the left half and a forward-pointer in the right half.

The header of the area starts at the first 0 mod 8 address in the area and is as follows:



"First data" and "last data" demarcate the space into which data may go. They are set up by areamk and are not changed except by areamk_$redef. "Last busy" points to the last word of busy storage.

Area_ sets up "first data" and "last data", sets "last busy" to the address of the dummy trailer, makes all the strata empty, and creates the dummy trailer. Area_$redef simply increases "last data" by the appropriate amount.

The method used by allocate_ is:

1. Pad the request to n words, where n is even and includes the necessary trailer.
2. Determine stratum (min i such that 2**i≥n).
3. Investigate consecutively that and all higher strata until one not empty is encountered. If none go to 10.
4. Split the first block in the stratum.
5. Construct a trailer for the first (allocated) part, and a pointer to it.
6. Update trailer of preceding block.
7. Remove from free list. If stratum becomes empty, zero it.
8. Construct trailer for second (free) part, and use freen_ to get it into the correct stratum.
9. Return the pointer.
10. If "last data"-"last busy" <n then signal the area condition.

11. Construct a trailer at "last busy" +n, and a pointer to this block.

12. Update preceding trailer.

13. Update "last busy".

14. Return the pointer.

The method used by freen_ is:

1. Consult preceding trailer to find length of block to be freed.

2. Look at busy bit of preceding trailer. If preceding block is free, remove it from its stratum and combine it with this block.

3. Compare addr of last word of this block with last_busy. If they are equal, scratch this block. If not, go to 6.

4. Update preceding trailer and last_busy.

5. Return.

6. Look at succeeding block. If it is free, remove from stratum and combine with this block.

7. Put free block in its proper stratum.

8. Update preceding trailer.

9. Return.