Assignment 6 -- Memory Management (100 points)

In this assignment, we are simulating memory management. When a process is loaded into memory or a process requests a block of memory dynamically, the system must allocate memory, and when a process terminates or frees memory previously requested, the system must deallocate memory.

For the sake of this assignment, we will assume we have a small computer with only 16 MB of memory. We will assume that the operating system uses the first 3 MB, leaving 13 MB for applications. Starting at that point, we will process several kinds of transactions:

--- Load a process into memory
--- Allocate a block of memory dynamically
--- Deallocate a block obtained earlier
--- Terminate a process, freeing all its memory

From time to time, we will print out the contents of the data structures involved.

A description of the input file is provided below.

Write a program in C++ on the turing system to accomplish this. Your program (which may be in several files) should do the following:

--- Define a class or struct to represent one block of memory. It should include the starting address (an integer), the size (an integer), the process ID of the owner (a string, might be blank) and the ID of the block (a string, might be blank), as well as one or two pointers to blocks.

--- Use #define to create a constant which I shall call HOWOFTEN. Give it the value 5.

--- Create two linked lists of blocks. One list contains blocks that are presently in use, and the other contains blocks that are not presently in use. I shall refer to these as InUse and Avail, respectively.

--- Initialize InUse to be empty.

--- Initialize Avail to consist of one 1 MB block, two 2MB blocks and two 4 MB blocks, in that order. The starting address for the first block should be at 3*1024*1024. (Again, we are assuming the operating system uses the first 3 megabytes of memory. While it is

loading, this memory management system is not yet in use, which is why we have these unconsolidated blocks on the list.)

--- Before any transactions, print the contents of both lists.

--- Open and read the input file and carry out the transactions.

--- Every HOWOFTEN transactions, reprint the contents of both lists.

--- After all transactions, reprint the contents of both lists.

--- Be sure you close the input file.

Notes

The program will take one command-line argument. The value should be either "B" or "F". If the value is "B", the program should use the Best-Fit algorithm for allocation, and if the value is "F", the program should use the First-Fit algorithm. If the argument is missing or the value is anything else, print an appropriate error message and exit with a negative exit value.

At the top of the output, print a heading saying this is a memory-management simulation and which allocation algorithm is in use.

Notice that the Avail list will be different from time to time. It will not always have the same five blocks.

We will arbitrarily assume memory requests are restricted to a range of 4 KB to 4 MB. Blocks of memory are limited to 4 MB.

You may find it useful to have defined constants KB and MB for the numbers 1024 and 1024*1024.

When you print a list, include a line giving the total size of each list. (The sum of the two should always be the same: 13*1024*1024. Never lose a byte.)

Notice that the Avail list is maintained in order by Starting Address (increasing). Order is not important for the InUse list.

Input File

The file can be found here:

/home/turing/t90hch1/csci480/Assign6/data6.txt

Each line in the file represents one transaction. Each line begins with 1 letter, followed by other items:

L: Load a process. This is followed by its ID, size and block ID (which in this case is the program name).
A: Allocate memory. This is followed by the process ID, size and an ID for this block of memory.
D: Deallocate memory. This is followed by the process ID and the ID of the block.
T: Terminate a process: This is followed by the process ID.

The last line in the file starts with '?' as a delimiter.

Transactions:

Load transaction

Search the Avail list for the best-fit or first-fit block.

```
If there is no sufficiently large block
  print an error message
Else
  Create a new block of the correct size, recording the process
  ID and the block ID
  Decrease the block found on Avail by that amount
  If the block on Avail is now of size 0
    Delete it
  End-If
  Insert the new block at the beginning of InUse
  Print a success message
End-If.
```

Allocate transaction

Search the Avail list for the best-fit or first-fit block.

```
If there is no sufficiently large block
  print an error message
Else
  Create a new block of the correct size, recording the process
  ID and the Block ID
  Decrease the block found on Avail by that amount
  If the block on Avail is now of size 0
    Delete it
  End-If
  Insert the new block at the beginning of InUse
  Print a success message
End-If.
```

Deallocate transaction

Search the InUse list for the block with the correct process ID
and block ID. (The combination of the two should be unique.)

```
If it is not found
  Print an error message
Else
  Detach the block from InUse
  Insert it into Avail in order by starting address
  Traverse the Avail list combining any two consecutive blocks
  for which the ending address of the first block is the starting
  address of the second block, provided the combined block is no
   more than 4 MB
  Print a success message
End-If.
```

Terminate transaction

Search the InUse list as often as necessary to find and deallocate
all blocks belonging to the indicated process.

```
If there are none
  Print an error message
Else
  Print a success message (there is no need to announce each and
  every block that is deallocated in a Terminate transaction)
End-If.
```

Comments

Copy the input file into your own directory.

You may want to use the C++ string class <string> and you may need
the C libraries <cstring.h> and <cstdlib>.

You may find it easier to write this program if Avail and InUse are
global variables. (This is up to you.)

You will need to invent functions for various purposes. Be sure to
document them.

You may use the standard template libraries if you wish.

As you work on this, you may find it useful to print out a great deal
more information as you go along. If you do so, please ensure that
the extra information is not printed by your final executable file.

You may use other names for variables if you like, but make sensible choices.

The Load and Allocate transactions are obviously very similar.

Your program should use reasonable variable names and should be appropriately indented and well documented. You can find style guidelines on the web sites of the CSCI 240 and 241 courses.

The name of the executable file should be "Assign6".

When you are done, you need to submit your work on Blackboard. As in the other assignments, you should create a tar file containing the various files involved: the program file, class files, header files and the makefile. To do this, you need the "tar" utility.

Do the following (replacing "Znumber" with your own Z-ID):

(a) Create a subdirectory named Znumber_A6_dir.
(b) Copy the files into it (source code files, headers, input file and makefile).
(c) In the parent directory of Znumber_A6_dir, use this command:
tar -cvf Znumber_A6.tar Znumber_A6_dir

Use an FTP program to retrieve the tar file and then submit it on Blackboard. The TA will move it to turing, extract the files and run your makefile, as in:

```
tar -xvf Znumber_A6.tar
cd Znumber_A6_dir
make
Assign6
```

If your makefile does not run (on the turing system) or your program does not compile and run (on the turing system), you will receive no credit.