資料結構與程式設計

(Data Structure and Programming)

102 學年上學期複選必修課程 901 31900

Homework #4 (Due: 9:00pm, Friday, Nov. 22, 2013)

0. Objectives

1. Learning the memory management techniques, as an introductory understanding for data structure design.

2. Extending the software system in Homework #3: more source code package and more commands.

3. Supporting more complex command usage.

4. Being able to comprehend existing code and enhance/complete it.

1. Problem Description

In this homework, we are going to create a memory manager and its test program on top of the software system of Homework #3. The generated executable is called "**memTest**" and has the following usage:

memTest [-**File** <*dofile*>]

where the **bold words** indicate the command name or required entries, square brackets "[ ]" indicate optional arguments, and angle brackets "< >" indicate required arguments. Do not type the square or angle brackets.

This command-line memory manager test program should provide the following functionalities:

1. Create a `template class MemMgr` to manage the memory as described in the lecture notes. It should contain non-continuous memory blocks (`template class MemBlock`) with recycling capability. The size of each memory block can be determined in the constructor *MemBloc::MemBlock(size_t blockSize = 65536)*, or the function "*MemMgr::reset(size_t blockSize)*". The default size of

the memory block is 65536, and the parameter "*blockSize*" can re-define the "number of Bytes" for the block.

2. The class of objects to be managed is `class MemTestObj`. It contains an *int[8]* and a *char[2]*, which are in total 34 Bytes. However, in actual memory allocation, its size (by *sizeof*(`MemTestObj`)) will be promoted to 36 Bytes to get aligned with the size of *int*. This class contains a static data member "*static MemMgr* const _memMgr*" as its memory manager and its "*new*", "*new[]*", "*delete*", and "*delete[]*" operators are then overloaded to call the memory allocation/free functions *alloc(), allocArr(), free()* and *freeArr()*, respectively, in `class MemMgr`.
3. The allocated "`MemTestObj`" objects and arrays (by the "MTNew" command) are stored in the data members (type "vector<MemTestObj*>") of the `class MemTest` as "*_objList*" and "*_arrList*", respectively. The "indices" (See command "MTDelete") of the objects/arrays are their positions in the vectors. They are used to find the objects/arrays for the "*delete/delete[]*" related commands. When an object/array is deleted, the corresponding position in the list is set to '0'. If another "*delete/delete[]*" command is trying to delete the object/array with the same index, there will be no action and no error message.
4. The `class MemRecycleList` is a friend class only to `class MemMgr` and is to recycle memory when the "*delete*" or "*delete[]*" operators are called. There is a "`MemRecycleList`" array ("*MemMgr:: _recycleList*") of size 256 to record the recycled "`MemTestObj`" for single objects (i.e. *delete*), and arrays of size 1, 2, 3, ..., to 255 (i.e. *delete[]*). If the deleted "`MemRecycleList`" array has size '*n*' which is greater than 255, it will be recorded in a "`MemRecycleList`" that can be traced from the "*_recycleList[n % 256]*" (See class slides and reference code for more details).
5. A printing command to print out the contents in memory manager and class `MemTest`.

2. Supported Commands

Other than the commands in Homework #3, we will support these new commands:

```
MTReset:      (memory test) reset memory manager
MTNew:        (memory test) new objects
MTDelete:     (memory test) delete objects
MTPrint:      (memory test) print memory manager info
USAGE:        report the runtime and/or memory usage
```

Please refer to Homework #3 for the lexicographic notations. However, please note that the "[ ]" optional parameters can appear anywhere in the command line, while the "< >" mandatory parameters must follow the order as specified in the command usage.

2.1 Command "MTReset"

Usage: **MTReset** [(size_t blockSize)]

Description: Reset the memory manager. The optional parameter "(size_t blockSize)" specifies the number of Bytes for each memory block. Please note that the parameter "blockSize" will be promoted to the closest multiple of SIZE_T (by the MACRO "toSizeT()") before being passed to the MemMgr::reset() function. This command first release all except for the first memory blocks back to system. Then it checks whether the new blockSize is different from the size of the first (active) memory block. If yes (different), reconstruct this only block (i.e. the first block) to the new block size. Otherwise, or if the parameter is not specified, the blockSize remains unchanged and the memory of the first block will NOT be freed. The initial value of the blockSize is 65536.

Example:

mtest> mtreset // reset memory manager using original value

mtest> mtr 123 // reset memory manager with blockSize = toSizeT(123) Bytes

Guarded command errors:

1. lexSingleOption() == false
2. If the specified "blockSize" is not a legal integer or is smaller than toSizeT(sizeof(MemTestObj)).

2.2 Command "MTNew"

Usage: **MTNew** <(size_t numObjects)> [**-Array** (size_t arraySize)]

Description: Allocate memory test objects (class MemTestObj) and store in "MemTest::_objectList" or "MemTest::_arrList". The parameter "(size_t numObjects)" specifies the number of memory test objects or arrays to be allocated. If the optional parameter "**-Array** (size_t arraySize)" is specified, allocate arrays of memory test objects with "new MemTestObj[arraySize]". Otherwise, allocate single objects by "new MemTestObj". Note that the allocated memory must be promoted to multiple of sizeof(size_t). If the requested memory for the object or array is greater than the block size of `MemBlock`, an exception "bad_alloc()" should be thrown and this command should catch it and return to the command prompt (i.e. Don't crash the program). If the requested memory is larger than the remaining space of the current memory block, recycle the remaining memory space and allocate a new memory block for the requested memory.

Examples:

mtest> mtnew 100 // new MemTestObj's for 100 times

mtest> mtn 20 -a 5 // new MemTestObj[5] for 20 times

Guarded command errors:

1. lexOptions() == false
2. If the specified "numObjects" is not a legal integer or is not a positive number.
3. If the parameter "numObjects" is not specified or is specified multiple times.
4. If the specified "arraySize" is not a legal integer or is not a positive number.
5. If the parameter "arraySize" is specified multiple times.
6. Any other syntax error with respect to the command usage.
7. Requested memory of the object or array is greater than the block size of `MemBlock`.

2.3 Command "MTDelete"

Usage: **MTDelete** < **-Index** (size_t objId) | **-Random** (size_t numRandId)> [**-Array**]

Description: Delete memory test objects (class MemTestObj) and set the corresponding entries in "MemTest::_objList" or "MemTest::_arrList" to 0's. If the optional parameter "**-Array**" is specified, delete the objects from "MemTest::_arrList". Otherwise, delete from "MemTest::_objList". The parameter "**-Index** (size_t objId)" explicitly specifies the object/array to be deleted in the "_objList/_arrList" (objId as the array index). If the parameter "-Random (size_t numRandId)" is specified, randomly generate "numRandId" numbers of integers, without checking repeats, as indices for objects/arrays in the "_objList/_arrList" array to be deleted. The generated random indices must lie between 0 and (_objList or _arrList array size – 1). If the object/array with respect to the explicitly or randomly specified index has been deleted, just ignore it and *do not issue an error or re-generate the random number*.

Example:

mtest> mtdelete -i 3 // delete _objList[3]

mtest> mtd -r 5 // randomly generate 5 indices for deletion in _objList[]

mtest> mtd -i 8 -array // delete [] _arrList[8]

Guarded command errors:

1. lexOptions() == false
2. If both "-Index" and "-Random" are specified.
3. If none of "-Index" or "-Random" is specified.

4. If the parameter “-Index” is specified multiple times.
5. If the specified “objId” (for “-Index”) is not a legal integer, is smaller than 0, or is greater than or equal to the size of “_objList” (if no “-Array”) or “_arrList” (if with “-Array”).
6. If the parameter “-Random” is specified multiple times.
7. If the specified “numRandId” is not a legal integer, or is not a positive number.
8. If the parameter “-Random (size_t numRandId)” is specified but the “_objList” (if no “-Array”) or “_arrList” (if with “-Array”) is empty.
9. Any other syntax error with respect to the command usage.

2.4 Command “MTPrint”

Usage: **MTPrint**

Description: Print out the contents of the memory manager and the class MemTest. The output format is as shown in the following example ---

```
========================================
=              Memory Manager          =
========================================
* Block size            : 65536 Bytes
* Number of blocks      : 8
* Free mem in last block: 24568
* Recycle list          :
[  0] = 1         [512] = 1         [1024] = 1         [347] = 1
[101] = 1         [613] = 1         [614] = 4
========================================
=             class MemTest             =
========================================
Object list ---
oooxoooooo
Array list ---
ooooooooxooooxo
```

Please note that the printout is aligned to left. The numbers “[a] = b” under “Recycle list” mean that the number of elements in the recycle list of array size “a” is “b”. Do not show the entry if the number in the recycle list is 0. The order of the printed recycle lists follows the ascending order of the “array size % R_SIZE” (so, 0 → 512 → 1024 → 347(% 256 = 91) → 101 → 613 (% 256 = 101) → 614). However, for the recycle lists of the same “array size % R_SIZE”, print them out in the order as how they are linked together (i.e. in the chronological order

as they are constructed). The strings under “Object list” and “Array list” indicate the status of the objects and arrays, respectively. ‘o’ means the objecy/array in the corresponding position is still valid, ‘x’ means it has been deleted.

Examples:

mtest> mtprint

Guarded command errors:

1. If any parameter is specified.

2.5 Command “USAGE”

Usage: **USAGE [-All | -Time | -Memory]**

Description: report the runtime and/or memory usage.

Examples:

mtest> usage // print out both runtime and memory usage

mtest> usage -time // print out the runtime usage only

mtest> usage -m // print out the memory usage only

Guarded command errors:

1. If any of the parameters is repeatedly specified.
2. If two or more of the parameters “-All”, “-Time”, and “-Memory” are specified.

Note: This command has been implemented in package “*cmd*” and pre-compiled into “*libcmd-{64,32}.a*” in “*lib*”. You don’t need to work on it.

3. What you should do?

You are encouraged to follow the steps below for this homework assignment:

1. Read the specification carefully and make sure you understand the requirements.

2. Think first how you are going to write the program, assuming you don’t have the reference code.

3. Study the class slides and the provided source code (especially under package “*mem*”).

4. The source codes for package “*cmd*” has been precompiled as “*libcmd-32.a*” and “*libcmd-64.a*” for 32 and 64-bit platforms, respectively. Use “**make 32**” or

"**make 64**" in root directory to change the symbolic links in the directory "*lib*" to suit your platform. Note that we will not test special keys in this homework. However, if you have different keyboard mapping and would like to use the special keys, please go ahead to copy your own "*cmd*" package and modify the "REFPKGS" and "SRCPKGS" macros in Makefile accordingly. We will restore it when testing your program.

5. What you should do in this homework assignment are commented with "TODO"'s. You should be able to complete this assignment by just finishing these TODO's. However, if you like to add new member functions, please go ahead, but, *do not add/remove any **data member***. Just make sure you complete all the TODO's and your program meets all the specifications.

6. Complete your coding and compile by "*make*". Several test scripts are provided under the directory "*tests*". Please test your program thoroughly by creating more test scripts.

7. You can turn on the debugging message by typing "*make debug*". It will define the compilation flag `MEM_DEBUG` and create an executable called "**memTest.debug**". Detailed memory allocation information will be printed out. However, please note that the pointer addresses may be different in different executions. Moreover, sometimes the codes may not be re-compiled properly when switching between debug and normal modes. In that case, type "*make clean*" before "*make debug*" or "*make*" to ensure the compilation consistency.

8. Two reference programs are available under the "*ref*" directory. The "**memTest-32/64**" is the normal one without debugging information, while "**memTest-32/64.debug**" is with debugging information. Use "make 32 or 64" to switch platforms. Please note that the pointer addresses printed by "memTest.debug" may be different from yours.

9. Please also watch out the announcement in the class website, FB and BBS.

Notes:

1. Please use the global random number generator "rnGen" (defined in "util.cpp" and "rnGen.h") for the "-Random" option of the "MTDelete" command. With the fixed random seed (0) (in util/util.cpp:15), it should generate the same random number sequence every time and thus make it easier for grading. Do not change the seed.

4. Grading

We will test your submitted program with various combinations/sequences of commands to determine your grade. The results (i.e. outputs) will be compared with our reference program. Minor difference due to printing alignment, spacing, error message, etc can be tolerated. However, to assist TAs for easier grading work, please try to match your output with ours.

Please also make sure your code is platform independent. We will test your program on both 64 and 32-bit machines. For 64-bit platform users, you can compile the 32-bit version by adding "*-m32*" to the g++ flag. This can be easily done by switching lines 15 (with "*-m32*") and 17 in "src/Makefile.in". Be sure to type "**make 32**" before "**make**" to create the correct symbolic link for "libcmd.a". For 32-bit platform users, you can go to a public web-based testing platform http://cc.ee.ntu.edu.tw/~ric/dsnp100-1/memTest-64.htm. It will ask you to upload your homework "bxxxxxxxx_hw4.tgz". Please ignore the first few lines of the compilation warning if any. It will run the dofiles under /tests and shows the results on the screen. You can also type in your own dofile. Please note that you may want to "**make clean**" before uploading it. However, please don't remove the files as specified in MustRemove.txt (i.e. the .tgz file must be complete and compile-able).