cs280-ass1-oa

This assignment gives you some experience with memory management by applying linked list techniques to implement a rudimentary, but highly-effective, object allocator.

The task is to implement a class named ObjectAllocator which will be used to allocate and deallocate fixed-sized memory blocks for a client. The public interface for the ObjectAllocator class is very simple, almost trivial. In addition to the public constructor (and destructor), there are essentially only 2 public methods that the client requires. These are Allocate and Free. As far as the client is concerned, these methods replace the C++ new and delete operators, respectively. There are several other public methods that are present to help aid in testing, debugging, and measuring your free list management.

All implementations must adhere to this interface file, ObjectAllocator.h in your project and implement the methods in a file called ObjectAllocator.cpp. The amount of code required to implement this interface is really not that great. You will need to spend sufficient time testing your code to make sure it handles all cases (allocating pages, providing debugging data, allocating objects, freeing objects, validating objects, error handling, etc.) A simple example test driver program and output is available to help get you started with testing. It is not an exhaustive test driver, but it is a start. You may use this sample driver as a starting point for a client program that you will create to sufficiently test your ObjectAllocator class with varying sizes of data. Also, if you are unsure of how something is supposed to work, you should ask a question. In the real world you are rarely given all of the information before starting a project (although we've tried to give you everything you will need.)

Notes

1. There is a structure called OAConfig which contains the configuration parameters for the ObjectAllocator. You should not need to modify this.
2. There is a structure called OAStats which contains the statistical data for the ObjectAllocator. Do not modify this.
3. Setting MaxPages to 0 effectively sets the maximum to unlimited (limited only by what new can allocate)
4. The DebugOn_ field is a boolean that determines whether or not the debugging routines are enabled. The default is false, which means that no checking is performed. When true, the memory allocated has certain values and depends on the current status of the memory (e.g. unallocated, allocated, freed, or padding). These values are specified in the header file on the web page for this assignment. Also, if DebugOn_ is true, validation during calls to Free is enabled (checking for double-frees, corruption, page boundaries, etc.)
5. The UseCPPMemManager_ field is a flag that will enable/disable the memory management system. Specifically, it allows you to use the C++ new and delete operators directly. (Effectively bypassing all memory management you've written.) With this flag enabled, you will only be able to count the total allocations/deallocations and the most objects in use at any one time. The other counts are undefined and should be ignored (e.g. pages in use).
6. The PadBytes_ field is the number of bytes that you will "pad" before and after the each block that is given to the client. These bytes are used to detect memory overruns and underruns.
7. There is a struct in the implementation file called GenericObject, to help with the implementation (casting).
8. The exception class is provided and should not be modified.
9. There are several public methods for testing/debugging the ObjectAllocator. These will also help you in creating flexible driver programs and to test your implementation.
10. The DumpMemoryInUse method returns the number of blocks in use by the client.
11. The FreeEmptyPages method returns the number of pages that were freed.
12. The ValidatePages method returns the number of blocks that are corrupted. Only pad bytes are validated.
13. When checking for boundaries, your code should perform the operation in constant time. This means you don't need to "walk" through the page to determine if the block is at the proper boundary. (Hint: Use the modulus operator: %)
14. Since you're dealing with raw memory, it is natural that you will need to cast between pointers of different types. You must use the C++ named cast (e.g., static_cast, reinterpret_cast). Do not use the old, unnamed casting. See the sample driver for examples. Also, look up `-Wold-style-cast` for GNU's compilers to help you find these old C-style casts. The graded drivers will be using this..
15. When using new to allocate memory, you must wrap it in a try/catch block. Do not use `std::nothrow`. If you don't know what that is, then you're probably not going to use it.
16. Do not use void pointers. In other words, do not create any members or variables of type: `void *`.They are unnecessary and will complicate your code with lots of unnecessary casting.
17. To help understand how the ObjectAllocator works, several diagrams are posted on the course website with the other documents for this project. Make sure you understand them before starting to code the assignment.
18. The purpose of this course is to learn about and understand data structures and their interfaces. That's the reason we provide public methods for accessing implementation details, which normally would remain hidden.

Other Details

You must use new and delete to allocate/deallocate the pages.

Do not use `std::nothrow` with `new`. (If you don't know what that is, you're probably not going to use it.)

You must catch the exception `std::bad_alloc` that is thrown from `new`.

You can't use any STL code in your implementation (with the exception of `std::string` for exception messages).

You can't change the public interface at all.

Given this struct:

```
struct Student
{
  int Age;
  float GPA;
  long Year;
  long ID;
};
```

`sizeof(Student)` returns:

```
 LLP64: 16
  LP64: 24
```

DO NOT assume that pointers, integers, and long integers are the same size. Doing so may cause the compiler to emit warnings and can also lead to your program crashing. See this pdf[1] for a quick refresher.

Adjusting for Microsoft's odd compiler:

```
struct Student
{
  int Age;
  float GPA;
  long long Year; // Compensate for Microsoft's compiler
  long long ID;   // Compensate for Microsoft's compiler
};
```

`sizeof(Student)` now returns:

```
 LLP64: 24
  LP64: 24
```

Using the `GenericObject` structure. Assume that `FreeList_` and `PageList_` are each pointers to a `GenericObject`:

```
GenericObject *PageList_; // the beginning of the list of pages
GenericObject *FreeList_; // the beginning of the list of objects
```

Freeing a block by putting it back on the free list (at the front of the list):

```
  // Put block on front of the free list. FreeList currently points to the
  // front of the free list. block is an address of a block of memory that
  // was used by the client, e.g. a Student object. When the client returns
  // the block by calling Free(block), you would need to put the block
  // at the front of the free list. The Free function takes a void pointer
  // so it must be cast so that there is a next pointer.
GenericObject *temp = reinterpret_cast<GenericObject *>(block);
temp->Next = FreeList_;
FreeList_ = temp;
```

The code above would probably be in some kind of loop that would execute X times, where X is the number of objects (blocks) per page. Assuming that block is a simple pointer to a character, the next block would be something like:

```
block += size_of_object; // Find the next block in the page
```

[1]docs/64.pdf

and you would then put that on the front of the free list. Having helper functions will greatly simplify the work. If you look at this code while looking at the diagrams that I posted, you should get a better insight into how this should work.

Walking the `PageList_` and freeing all of the pages. Since each page was allocated as an array of characters, you have to cast it back so that it can be deleted properly.

```
while (PageList_)
{
  GenericObject *temp = PageList_->Next;
  delete [] reinterpret_cast<char *>(PageList_);
  PageList_ = temp;
}
```

In your own programs, you can make your pagelist and freelist either `GenericObject` pointers or char pointers. When you're trying to access the Next member, you will have to cast a character pointer to a `GenericObject` pointer. When you're trying to do pointer arithmetic on the page, you will have to cast a `GenericObject` pointer to a character pointer.

In either situation, you will require a cast, so it really doesn't matter which way you go. However, DO NOT use void pointers as you will then need to cast every operation on the pointer because you can't do much with a void pointer (i.e. you can't dereference it, you can't do pointer arithmetic, you can't delete it, etc.)

Please note that these are examples! Your code will probably be different.

```
// Make sure this object hasn't been freed yet
if (is_on_freelist(anObject))
  throw OAException(OAException::E_MULTIPLE_FREE, "FreeObject: Object has already been freed.");

// If new throws an exception, catch it, and throw our own type of exception
char *newpage;
try
{
  newpage = new char[PageSize_];
}
catch (std::bad_alloc &)
{
  throw OAException(OAException::E_NO_MEMORY, "allocate_new_page: No system memory available.");
}
if ( /* Object is on a valid page boundary */ )
  // put it on the free list
else
  throw OAException(OAException::E_BAD_BOUNDARY, "validate_object: Object not on a boundary.");
```

Files Provided

The ObjectAllocator interface[2] we expect you to adhere to. Do not change the public interface. Only add to the private section and any documentation you need.

Sample driver[3] containing loads of test cases.

Random number generator interface[4] and implementation[5].

Diagrams[6] to help your understanding.

Notes from a fellow instructor[7] to help understanding.

FAQ[8] on the assignment.

There are a number of sample outputs in the **data** folder e.g.,

[2] code/ObjectAllocator.h
[3] code/driver-sample.cpp
[4] code/PRNG.h
[5] code/PRNG.cpp
[6] docs/diagrams.pdf
[7] docs/a1__helper__notes.pdf
[8] docs/faq.pdf

- 64-bit (masked addresses)[9]
- 64-bit with extra credit (masked addresses)[10]
- 32-bit (masked addresses)[11]
- 32-bit with extra credit (masked addresses)[12]

Compilation:

These are some sample command lines for compilation. GNU should be the priority as this will be used for grading.

GNU g++: (Used for grading)

```
g++ -o gnu driver-sample.cpp ObjectAllocator.cpp PRNG.cpp \
    -O -Werror -Wall -Wextra -Wconversion -std=c++14 -pedantic
```

Microsoft: (Good to compile but executable not used in grading)

```
cl -Fems driver-sample.cpp ObjectAllocator.cpp PRNG.cpp \
    /EHsc /Oy- /Ob0 /MT /Zi /D_CRT_SECURE_NO_DEPRECATE
```

Memory leak debugging

You may wish to use tools to check for leaks since you are dealing with raw ptrs. E.g., here's sample code to run your code under Valgrind in Linux and Mac OS X:

```
valgrind -q --leak-check=full --show-reachable=yes --tool=memcheck ./gnu
```

For windows you may want to check out Dr Memory[13].

Deliverables

You must submit your header file, implementation file, and compiled help file (ObjectAllocator.h, ObjectAllocator.cpp, index.chm) by the due date and time to the proper submission page as described in the syllabus.

Your code must compile cleanly with GNU g++ to receive full credit. You should do a "test run" by extracting the files into a folder and verifying that you can compile and execute what you have submitted (because that's what we're going to do)

Make sure your name and other info is in all documents.

ObjectAllocator.h

The interface to the ObjectAllocator. You will need to modify this to help you implement the algorithm, so be very careful not to remove or change any of the public interface. I will be using drivers that expect the current interface. No implementation is permitted in this file (with the exception of OAException which is already provided).

ObjectAllocator.cpp

The implementation file. All implementation goes here. You must document this file (file header comment) and functions (function header comments) using Doxygen tags as previously.

[9] data/output-sample-zeros-LP64.txt
[10] data/output-sample-zeros-ec-LP64.txt
[11] data/output-sample-zeros-ILP32.txt
[12] data/output-sample-zeros-ec-ILP32.txt
[13] http://www.drmemory.org/