Dynamic Memory

Objectives

1. Illustrate how memory can be dynamically allocated so that an object resides on the freestore
2. How that memory can be deallocated so that it maybe "recylced" for a different object

Labwork

1. Download the following source files and upload them to your directory on build.tamu.edu

   DynamicMemory.cpp https://drive.google.com/open?id=0B_ouNNuWgNZCMl9DamxJTDc1UFk

   Utilities.h https://drive.google.com/open?id=0B_ouNNuWgNZCSDRNVUFUQXlpYTg

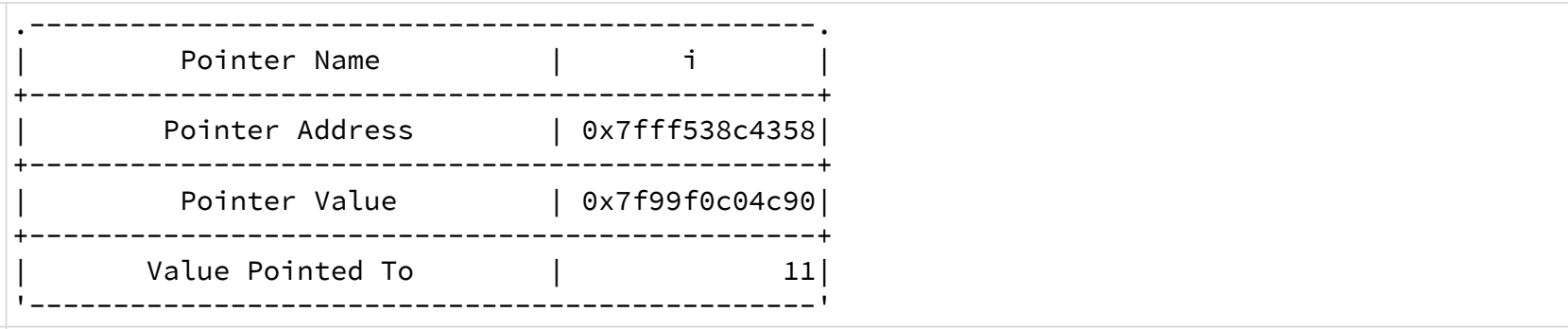
   Utilities.cpp https://drive.google.com/open?id=0B_ouNNuWgNZCTTVqdlFWZXVtZDA

2. Compile the source files using the command g++ -std=c++11 *.cpp

3. In the main() function of DynamicMemory.cpp, declare a pointer named i to an int, and initialize it with the address of a dynamically allocated int object containing the value 11

4. Call print_info(cout, i, "i", true), a function that I have written for you in the included Utilities files, directly after the assignment done in the prior step. The first argument in this function is the ostream to write to, the second is the pointer, the third is a string of the pointer's name, and the fourth a boolean argument whether or not the pointer should be dereferenced.

5. Compile and run your program. Your output should be similar to mine below (of course, the addresses that you see (i.e., the hexadecimal numbers) will be different from mine.

	
6	Call delete on the pointer to deallocate the memory that had been observed to the integer object being pointed to. Thereafter, Make a function call to to print_info(cout, i, "i", true).
7	Compile and run your program; observe the results.
8	Think about how this output has changed since your previous calls to this function in step-5.
9	Replace the current contents of your main() function with the following code:

	`int* i = new int{11};`
	`int* j = i;`
	`print_info(cout, i, "i", true);`
	`print_info(cout, j, "j", true);`
	`return 0;`

10	Compile and run your program. Observe the output. It should look similar to mine:

```
.-------------------------------------------.
|         Pointer Name          |       i       |
+-------------------------------------------+
|        Pointer Address        | 0x7fff5b5f4328|
+-------------------------------------------+
|         Pointer Value         | 0x7f8fc2c04c90|
+-------------------------------------------+
|       Value Pointed To        |             11|
'-------------------------------------------'
.-------------------------------------------.
|         Pointer Name          |       j       |
+-------------------------------------------+
|        Pointer Address        | 0x7fff5b5f4320|
+-------------------------------------------+
|         Pointer Value         | 0x7f8fc2c04c90|
+-------------------------------------------+
|       Value Pointed To        |             11|
'-------------------------------------------'
```

How are pointers i and j similar with respect to their values?

The pointers i and j point to the same object in memory

12 *How are pointers i and j similar as objects?*

The pointers i and j point are different objects in memory

13 Replace the current contents of your main() function with the following code:

```
int* i = new int{11};
int* j = i;
print_info(cout, i, "i", true);
print_info(cout, j, "j", true);
delete j; delete i;
print_info(cout, i, "i", true);
print_info(cout, j, "j", true);
```

14 Compile and run the program. You should see a similar error message to the following:

	`a.out(11082,0x7fff7b463300) malloc: *** error for object 0x7ff518d00000: pointer being freed was not allocated`
	`*** set a breakpoint in malloc_error_break to debug`
	`[1]    11082 abort      ./a.out`

	Why was this error message generated?
	We called delete twice on a dynamically allocated object
15	Remove the second delete statement. Compile and run the program.
	Does the previous error message present?
	No
16	Understand that the value stored in the memory location pointed to by i and j and interpreted as an integer may or may not contain the value 11. This is because that memory space has been re-claimed and may now be used to store a different object. Note: "deleting an object from the heap" does not zero out the bits of that object, but instead releases the memory for re-use by another object.
17	In your main(), immediately after delete j, assign j nullptr. Compile and run the resultamt program.
	What error message do you observe?
	segmentation fault
18	The reason for the previous error is that the print_info(cout, j, "j", true); call on j attempts to dereference the nullptr. This behavior is undefined.
	True or false: the nullptr should not be dereferenced
	True
19	To avoid this, change that statement to read print_info(cout, j, "j", false);.
20	Compile and run your program. Observe how the information provided about the pointers differs from before and after the delete call on j.
21	Why is the value of pointer i now problematic?
	The pointer i points to an object whose memory has been deallocated

	Submission
	Submit your completed copy of this document (with each question completed) to gradescope for grading.