GHW: Dynamically allocated objects

Objectives

1. Illustrate how to dynamically allocate objects on the freestore
2. Learn how that memory can be deallocated so that it may be "recycled" for a different object

Labwork

1. Download the following source files and upload them to your directory on compute.cse.tamu.edu (H: drive)

   dyn_alloc_objs.cpp

   https://drive.google.com/open?id=0B_ouNNuWgNZCdm5DSHVnMnZmZjA

2. Compile the source files on compute.cse.tamu.edu using "g++-7.2.0 -std=c++17 dyn_alloc_objs.cpp"

3. In the main() function of dyn_alloc_objs.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

   ```
   int* i = new int(11);
   ```

4. Call print_ptr_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.

   ```
   .-----------------------------------------------.
   |          Pointer Name          |       i      |
   +-----------------------------------------------+
   |         Pointer Address        | 0x7fff538c4358|
   +-----------------------------------------------+
   |          Pointer Value         | 0x7f99f0c04c90|
   +-----------------------------------------------+
   |       Value Pointed To         |            11|
   '-----------------------------------------------'
   ```

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_ptr_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_ptr_info(cout, i, "i", true);
   print_ptr_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|
'-----------------------------------------------'
```

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_ptr_info(cout, i, "i", true);
print_ptr_info(cout, j, "j", true);
delete j; delete i;
```

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

16a 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 after calling delete on that address. This is because that memory space has been reclaimed and may 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.

16b Update the contents of your main() function with the following code:

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

17 In your main(), immediately after delete `j`, assign `j` `nullptr`. Compile and run the resultant program.

What error message do you observe?

segmentation fault

18 The reason for the previous error is that the print_ptr_info(court, 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_ptr_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
22	What should we do after we apply the delete operator to a pointer operand?
	Assign nullptr to that pointer
	Submission
	Submit your completed copy of this document (with each question completed) in pdf format to gradescope for grading.