CSCI 2270 – Data Structures

Recitation 3, Fall 2021

Dynamic Memory

Objectives

1. Compile Time Memory Allocation
2. Run Time (Dynamic) Memory Allocation
3. Freeing Memory
4. Exercise

1. Compile Time Memory Allocation

When we declare variables as shown below, the compiler allocates a space in memory for the variable so that data can be stored in it at some point during program execution. This block of memory cannot be reused by any other variable, its size cannot be changed, its allocated at compile time by the compiler, and the allotted memory is not “freed” till the end of execution of the program. You can see the memory location allocated by using the ampersand (&) operator as a prefix to the variable.

```
#include<iostream>

using namespace std;

int main() {
    int a;
    cout << &a << endl;
    return 0;
}
```

The allocation of compile-time memory happens on the Stack. The size of the memory is known to the compiler. When a function is called, the memory for the variables gets allotted on the stack and when the function exits, the memory is deallocated.

What if we do not know the memory required for the program? For ex., as seen in previous recitation examples, you would like to pass an arbitrary number of arguments in the command line and have it printed by the program? It is here, we use dynamic memory allocation.

CSCI 2270 – Data Structures

Recitation 3, Fall 2021

Dynamic Memory

2. Run Time (Dynamic) Memory Allocation

The need for dynamic memory stems from the flexibility of the programs to allocate and deallocate memory as and when required, the uncertainty of not knowing the amount of memory required on program execution and many more. Here, we can tell the compiler to allocate memory to certain variables dynamically. Unlike static memory allocation, dynamic memory allocation has larger memory available to allocate, and there is no guarantee for two subsequent elements of an array to have contiguous memory locations. **Dynamic memory allocated variables are stored in heap.**

Consider the following example. We use **new** keyword for dynamic memory allocation. As you can see, dynamic memory allocation is always done using pointers. The first pointer **ptr1** holds a dynamically allocated integer, and the second holds a dynamically allocated integer array of size 10.

```
int main()
{
   // Dynamic memory allocation
   int *ptr1 = new int;
   int *ptr2 = new int[10];
   return 0;
}
```

3. Freeing Memory

Unlike compile-time memory allocation, where the program automatically deallocates memory by removing it from the stack on function exit, a dynamically allocated memory variable **MUST BE FREED** by the programmer. This is done by calling the **delete** command on the variable. If not, the memory continues to be occupied. Although, for smaller programs coded in this class, a code might not throw an error, repeated execution of the same code in production software can lead to memory leaks and software crashes. The syntax for delete is of two types, freeing up a dynamically allocated array and everything else.

To free the memory allocated to **ptr2** (array), we call

```
   delete [] ptr2;
   ptr2 = nullptr;
```

and to free the memory allocated to an integer **ptr1**, we call

```
   delete ptr1;
   ptr1 = nullptr;
```

CSCI 2270 – Data Structures

Recitation 3, Fall 2021

Dynamic Memory

Note: Always assign a **nullptr** to a deallocated dynamic memory variable. After delete is called on a pointer, the address referenced by it is still available for manipulation to the user. To fix this scenario, one has to ensure that the deleted address is not available to anyone for manipulation. Thus, the deleted address is overwritten by assigning a **nullptr** to the dynamic memory variable.

4. Exercise

Open your exercise file and complete the TODOs in the C++ program which does the following:
1. It will read from a text file. Each line of the file contains a number. Provide the file name as a command line argument. Number of lines in the file is not known.
2. Create an array dynamically of a capacity (say 10) and store each number as you read from the file.
3. If you exhaust the array and yet to reach the end of file, dynamically resize/double the array and keep on adding.

Submission: Once you have completed the exercises, zip all the files up and submit on the Canvas link. Name the submission in the following format: recitation3_firstname_lastname.zip