# Department of Computing

**CS 250: Data Structures and Algorithms**

**Class: BEE-6AB**

# Lab 1: Pointers and Dynamic Memory

**Date: 10th September, 2015**

**Time: 10 am 1 pm- 2 pm-5 pm**

# Instructor: Mr. Faisal Shafait

# 

# Lab 1: Pointers and Dynamic Memory

**Introduction**

This lab is based on the implementation of pointers and dynamic memory allocation.

**Objectives**

The purpose of this lab is to use pointers, relationship between pointers and arrays, getting heap memory and manipulate it using pointers. Use of new and delete operator.

**Tools/Software Requirement**

Visual Studio C++

**Description**

You are aware of pointers and their implementation. To know about new operator and dynamic memory allocation, follow the tutorial below:

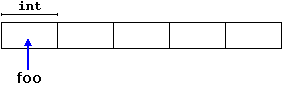
# Dynamic memory

We have seen in most cases that all memory needs were determined before program execution by defining the variables needed. But there may be cases where the memory needs of a program can only be determined during runtime. For example, when the memory needed depends on user input. On these cases, programs need to dynamically allocate memory, for which the C++ language integrates the operators new and delete.

# Operators new and new[]

Dynamic memory is allocated using operator new. new is followed by a data type specifier and, if a sequence of more than one element is required, the number of these within brackets []. It returns a pointer to the beginning of the new block of memory allocated. Its syntax is:   
  
pointer = new type  
pointer = new type [number\_of\_elements]  
  
The first expression is used to allocate memory to contain one single element of type type. The second one is used to allocate a block (an array) of elements of type type, where number\_of\_elements is an integer value representing the amount of these. For example:

|  |  |
| --- | --- |
| 1 2 | int \* foo;  foo = new int [5]; |

In this case, the system dynamically allocates space for five elements of type int and returns a pointer to the first element of the sequence, which is assigned to foo (a pointer). Therefore, foo now points to a valid block of memory with space for five elements of type int.  
  
   
Here, foo is a pointer, and thus, the first element pointed to by foo can be accessed either with the expression foo[0] or the expression \*foo (both are equivalent). The second element can be accessed either with foo[1] or \*(foo+1), and so on...  
  
There is a substantial difference between declaring a normal array and allocating dynamic memory for a block of memory using new. The most important difference is that the size of a regular array needs to be a *constant expression*, and thus its size has to be determined at the moment of designing the program, before it is run, whereas the dynamic memory allocation performed by new allows to assign memory during runtime using any variable value as size.  
  
The dynamic memory requested by our program is allocated by the system from the memory heap. However, computer memory is a limited resource, and it can be exhausted. Therefore, there are no guarantees that all requests to allocate memory using operator new are going to be granted by the system.

# Operators delete and delete[]

In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:

|  |  |
| --- | --- |
| 1 2 | delete pointer;  delete [] pointer; |

The first statement releases the memory of a single element allocated using new, and the second one releases the memory allocated for arrays of elements using new and a size in brackets ([]).

**Lab Tasks**

**Task 1**

include<iostream>

using namespace std;

int main(void)

{

int salary[20];

int i;

for (i=0;i<20;++i)

{

cout <<"Enter Salary: ";

cin >>salary[i];

}

for (i=0;i<20;++i)

salary[i]=salary[i]+salary[i]/(i+1);

return 0;

}

**Requirements:**

Rewrite this program using pointers in place of arrays.

**Task 2**

Complete the three parts to analyze pointer problem in the Lab\_1\_-\_Problem\_2.cpp file.

**Part 1:**

Write a function void analyze\_pointer(int \*ptr) that does two things:

* Write the memory location pointed by the pointer to the console.
* Write the value of the integer (which the pointer points to) to the console.

**Part 2:**

Use the function to complete two tasks:

* Allocate an int on the stack (e.g., “int iValue;"), assign a value to it, and get its memory location (with the reference operator---\&) to pass this value to analyze\\_pointer.
* Allocate an int on the heap (with the new operator). Assign a value to it, and pass it to analyze\\_pointer.

Question #1. What happens?

**Part 3:**

Now, we're going to add a couple more functions to the mix; call them int\_pointer1 and int\_pointer2. Both will return int pointers.

* int\_pointer1 will allocate an int on the heap (via new int), assign a value to it, and return that value.
* int\_pointer2 will allocate an int on the stack (via ``int iValue;" or something similar), assign a value to it, and return its memory location (as both of these functions should be of the int pointer type).

Call analyze\_pointer on the return of both of these functions from your main function:

* “analyze\_pointer(int\_pointer1());"
* “analyze\_pointer(int\_pointer2());"

{Question #2.} What happens, and why?

Two more tests on memory:

* Remember that int we allocated on the heap for function ``int\_pointer1()"? Delete it, and call analyze\_pointer on it after deleting it.
* Call: analyze\_pointer(new int);

Question #3. After completion of the problem, how many memory leaks are there?

**Task 3**

Define a struct Area that has two private variable members; units of type string and area\_value of type float. Modify the Lab\_1\_-\_Problem\_3.cpp program to create a dynamic variable of type Area.

* Input from the keyboard the area\\_value and its units. Compute one-half and one-quarter of the area and display the results
* Destroy the dynamic variable at the end

**Deliverables**

Students are required to upload the lab on LMS before deadline.

**Note:** Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks. You will submit your working **.cpp** files in one **(.zip)** folder. The name of files and folder should follow this format. i.e. **YOUR\_NAME\_Lab#**