CMPD 244
Programming II

Lab 3: Part 2
Dynamic Memory Allocation and Parameter Passing by Reference

This lesson is divided into 2 parts: Dynamic Memory Allocation and Parameter Passing by Reference. Dynamic Memory Allocation requires the creation and initialization of dynamic variable using *new* operator. In programming, all memory needs were pre-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*.

Program Sample 1: The following codes will show the example of Dynamic Memory Allocation using *new* and *delete* operator.

```
 #include <iostream>
using namespace std;

int main ()
{
 double *pvalue = NULL;
 pvalue = new double;

 *pvalue = 29494.99;
 cout << "Value of pbvalue : " << *pvalue << endl;

 delete pvalue;

 return 0;
}
```

Question 1: Trace the output of program Sample 1 above.

Program Sample 2: The following codes demonstrates a **C++ Program to store GPA of n number of students and display it where n is the number of students entered by user**

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int num;
    cout << "Enter total number of students: ";
    cin >> num;
    float* ptr;

    // memory allocation of num number of floats
    ptr = new float[num];
    cout << "Enter GPA of students." << endl;
    for (int i = 0; i < num; ++i)
    {
        cout << "Student" << i + 1 << ": ";
        cin >> *(ptr + i);
    }
    cout << "\nDisplaying GPA of students." << endl;
    for (int i = 0; i < num; ++i) {
        cout << "Student" << i + 1 << " :" << *(ptr + i) << endl;
    }
    // ptr memory is released
    delete [] ptr;
    return 0;
}
```

Question 2: Study the codes given in program Sample 2 and trace the output.

In previous lessons, we have seen the differences between function call by value (parameter passing by value) as well as function call by reference (parameter passing by reference). You can just have a copy of the variable's value, or a copy of the variable's address. If you have a copy of variable's value, any changes that you have made to the copy will not going to affect the original value. But if you make the changes to the content of the referenced address (pointer), the modification done will affect the value of the variable.

Program Sample 3: The following codes demonstrate function call by value.

```
#include<iostream>
using namespace std;

void Point(int);

int main()

{

int c = 5;

Point(c);
cout << "c = " << c << endl;

return 0;

  }

void Point(int c) {

c = 25;

}
```

Question 3: Modify the program sample below to indicate function passing by reference.

Don't forget to submit the screen capture of your output to our Brighten portal with the subject: CMPD244 Lab3 <YourStudentID><Your Full Name> at the end of this lab session. Good Luck! ☺