Note: All the code that you write for the following labs must be uploaded to Github.

C++ Pointers

A pointer is a type of variable. It stores the address of an object in memory, and is used to access that object. A pointer can be assigned the address of another non-pointer variable, or it can be assigned a value of **nullptr**. A pointer that has not been assigned a value contains random data. A pointer can also be dereferenced to retrieve the value of the object that it points at.

Implement the following in a new project, within a file titled **IntroToPointers.cpp**, and then build and execute your program.

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

int main() {

      // declare pointer and initialize it
      // so that it doesn't store a random address
      int* pPointer = nullptr;

      int integerVar = 5;

      // assign pointer to address of object
      pPointer = &integerVar;

      //output the value of integerVar
      cout << “integerVar: ” << integerVar << endl;

      //output the address of integerVar
      cout << “Address of integerVar: ” << &integerVar << endl;

      //output the address assigned to pPointer
      cout << “pPointer: ” << pPointer << endl;

      //output the address of pPointer
      cout << “Address of pPointer” << &pPointer << endl;

      return 0;
}
```

Implement the following in a new project, within a file titled **Pointers.cpp**, and then build and execute your program.

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

int main() {

     int firstValue;
     int secondValue;

     int * pPointer = nullptr;

     //assign pointer with the address of firstValue
     pPointer = &firstValue;
     *pPointer = 10; //Indirection

     //assign pointer with the address of secondValue
     pPointer = &secondValue;
     *pPointer = 20; //Indirection

     cout << "firstValue is " << firstValue << '\n';
     cout << "secondValue is " << secondValue << '\n';

     return 0;
}
```

Pointers and Arrays

As discussed in the lectures, the name of an array works as a pointer to the first element of an array.

Implement the following in a new project, within a file titled **PointersArrays.cpp**, and then build and execute your program.

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

int main() {

      int numbersArray[5];

      int * pPointer = nullptr;

      //assign the address to the first element to the pointer
      pPointer = numbersArray;

      *pPointer = 10; //assign a value to the first element

      /*increment the pointer using pointer arithmetic
      to assign the address of the second element to the pointer*/
      pPointer++;
      *pPointer = 20; //assign a value to the second element

      //assign the address of the third element to the pointer
      pPointer = &numbersArray[2];
      *pPointer = 30; //assign a value to the third element

      /*assign the address of the fourth element to the pointer
      using pointer arithmetic*/
      pPointer = numbersArray + 3;
      *pPointer = 40; //assign a value to the fourth element

      //assign the address to the first element to the pointer
      pPointer = numbersArray;

      /*assign a value to the fifth element using indirection and pointer
      pointer arithmetic*/
      *(p+4) = 50;

      //iterate and output all the elements in the array
      for (int n = 0; n < 5; n++)
      {
         cout << numbersArray[n] << ", ";
      }

      return 0;
}
```

Pointer Arithmetic

The unary operators (++ and **--**) are called **prefix** increment (++) or decrement (--) operators when the increment or decrement operators appear **before** the operand. In this regard, when the operator appears **before** its operand, the operand is incremented or decremented and its new value is the result of the expression. In other words ++ and -- can be written **before** a variable. For instance **++x**, where the expression evaluates to the final value of **x**, once it is already increased. Therefore, the increment (or decrement) happens before the expression is evaluated in this regard.

To demonstrate, implement the following in a new project, within a file titled **Prefix.cpp**, and then build and execute your program.

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

int main() {

      int x = 3;

      //the value of x is increased before the value of x is assigned to y
      //hence the value assigned to y is the value of x after being increased
      int y = ++x;

      cout << "x: " << x << endl; //x will be 4

      cout << "y: " << y << endl; //y will be 4

      return 0;
}
```

The unary operators (++ and --) are called **postfix** increment (++) or decrement (--) operators when the increment or decrement operators appear **after** the operand. In other words, ++ and -- can be written **after** a variable, for instance **x++**, where the value is increased, but the expression evaluates to the value that **x** had before being increased. Therefore, the increment (or decrement) happens after the expression is evaluated.

To demonstrate, implement the following in a new project, within a file titled **Postfix.cpp**, and then build and execute your program.

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

int main() {

      int x = 3;

      //the value of x is increased after the initial value of x is assigned to y
      //hence the value assigned to y is the value of x before it was increased
      int y = x++;

      cout << "x: " << x << endl; //x will be 4

      cout << "y: " << y << endl; //y will be 3

      return 0;
}
```

The **prefix** and **postfix** use of the increment (++) and decrement (--) operators also applies to expressions incrementing and decrementing pointers, which can become part of more complicated expressions that also include dereference operators (*). In this regard, the **prefix** increment and decrement operators have the same precedence/priority as the dereference (*) operator, and the **postfix** increment and decrement operators have a higher precedence/priority than both the **prefix** increment and decrement operators and the deference operator (which itself is also a prefix operator). In this regard, note the following:

***pPointer++** // same as ***(pPointer++)**: increment pointer, and dereference unincremented address

***++pPointer** // same as ***(++pPointer)**: increment pointer, and dereference incremented address

++*pPointer // same as **++(*pPointer)**: dereference pointer, and increment the value it points to

(*pPointer)++ // dereference pointer, and post-increment the value it points to

Implement the following in a new project, within a file titled **PointerArithmetic.cpp**, and then build and execute your program.

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

int main() {

      //null pointer
      int* pPointer = nullptr;

      int numbersArray[3] = {10, 20, 30};

      //assign address of first element to pointer
      pPointer = numbersArray;

      //output the address of the first element
      cout << "Address at pPointer: " << pPointer << endl;
      cout << "Address of numbersArray[0]: " << numbersArray << endl;

      //output the value of the first element using the pointer and indirection
      cout << "Value at pPointer: " << *pPointer << endl;

      //This outputs the value of the second element
      cout << "Value at ++pPointer: " << *(++pPointer) << endl;

      //assign address of first element to pointer
      pPointer = numbersArray;

      //This outputs the value of the first element
      cout << "Value at pPointer++: " << *(pPointer++) << endl;

      return 0;
}
```

Dynamic Memory Allocation

As discussed in the lecturers, memory can be allocated deallocated dynamically using the **new** and **delete** operators.

Implement the following in a new project, within a file titled **MemAllocation.cpp**, and then build and execute your program.

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

int main() {

      int numberOfElements = 0;
      int* dynamicArray = nullptr;

      cout << "How many numbers would you like to type? ";
      cin >> numberOfElements;

      dynamicArray = new int[numberOfElements];

      if (dynamicArray == nullptr) {

            cout << "Error: memory could not be allocated";
      }
      else {

            for (int i = 0; i < numberOfElements; i++) {
                  cout << "Enter number: ";
                  cin >> dynamicArray[i];
            }

            cout << "You have entered: ";

            for (int j = 0; j < numberOfElements; j++) {

                  cout << dynamicArray[j] << ", ";
            }

            delete[] dynamicArray;
      }

      return 0;
}
```

Do the following:

1. Develop a C++ program that does the following:
    - Dynamically allocates an integer
    - Dynamically allocates a string
    - Through user input, with prompts for the user:
        - Assigns a value to the dynamically allocated integer
        - Assigns a value to the dynamically allocated string
    - Outputs onto the console:
        - The value of the dynamically allocated integer
        - The value of the dynamically allocated string

2. Develop a C++ program that does the following:
    - Dynamically allocates a 2 dimensional array of doubles
        - The dimensions of the array must be provided via input from the user
        - The dimensions of the array must not exceed 3, therefore, your program must enforce this and prompt the user accordingly
    - Using nested loops, your program must assign values to each element of the array
    - Your program must output the values of each element of the array onto the console

References

- https://learn.microsoft.com/en-us/cpp/cpp/raw-pointers?view=msvc-170
- https://learn.microsoft.com/en-us/cpp/c-language/prefix-increment-and-decrement-operators?view=msvc-170
- https://cplusplus.com/doc/tutorial/pointers/
- https://cplusplus.com/doc/tutorial/operators/