CL1002 – Programming Fundamentals Lab

Lab # 3

Pointers

Instructor: Muhammad Saad Rashad

Email: saad.rashad@nu.edu.pk

Department of Computer Science,

National University of Computer and Emerging Sciences FAST

Peshawar

1. **Address:**

   An address is a location in memory where data is stored. Every variable in a C++ program has an address, which is represented by a unique integer value that identifies the memory location where the variable's value is stored.

2. **Pointers:**

   A pointer is a variable that stores the memory address of another variable. In other words, a pointer "points" to the location of a value in memory.

   Syntax:
   Declaration:
   int* ptr;

   Here we have declared a pointer ptr of type int.

   Assigning Addresses to pointers:
   Example:
   Let's take an example.

   int* pc, c;

   c = 5;

   pc = &c;

   Here, 5 is assigned to the *c* variable. And, the address of *c* is assigned to the *pc* pointer.

   Display value pointed by pointers

   To get a value of variable pointed by pointers we can use:

   int* pc, c;

   c = 5;

   pc = &c;

   cout<< *pc; // Output: 5

   Changing value pointed by a pointer:

   int* pc, c;

   c = 5;

```
pc = &c;
c = 1;
cout<< c;    // Output: 1
cout<<*pc;  // Output
: 1
```

Example 2: Working of pointers:

```
#include <iostream>
using namepsace std;
int main()
{
  int* pc, c;

  c = 22;
  cout<<"Address of c: \n", &c;
  cout<<"Value of c: \n\n", c;  // 22

  pc = &c;
  cout<<"Address of pointer pc: \n", pc;
  cout<<"Content of pointer pc:\n\n", *pc; // 22

  c = 11;
  cout<<"Address of pointer pc: \n", pc;
  cout<<"Content of pointer pc: \n\n", *pc; // 11

  *pc = 2;
  cout<<"Address of c: \n", &c;
  cout<<"Value of c: \n\n", c; // 2
  return 0;
}
```

3. **Relationship Between Arrays and Pointers:**

Example 3:

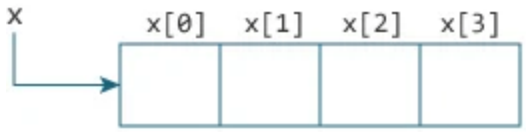
```
#include <iostream>
using namespace std;

int main() {
   int x[4];
   int i;

   for(i = 0; i < 4; ++i) {
      // Use the correct syntax for printing with cout
      cout << "&x[" << i << "] = " << &x[i] << endl;
   }

   // Printing the address of the array x
   cout << "Address of array x: " << x << endl;

   return 0;
}
```

Output:

&x[0] = 000000000062FE00
&x[1] = 000000000062FE04
&x[2] = 000000000062FE08
&x[3] = 000000000062FE0C
Address of array x: 000000000062FE00

There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of *int* is 4 bytes (on our compiler).

Notice that, the address of *&x[0]* and *x* is the same. It's because the variable name x points to the first element of the array.



From the above example, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.

Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 4:

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

int main() {
   int i, x[6], sum = 0;
   cout << "Enter 6 numbers:\n";

   for(i = 0; i < 6; ++i) {

      cin >> x[i];
      // Equivalent to sum += x[i];
      sum += x[i];
   }

   cout << "Sum = " << sum << endl;

   return 0;
}
```

Example 5:

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

int main() {
   int x[5] = {1, 2, 3, 4, 5};
   int* ptr;

   // ptr is assigned the address of the third element
   ptr = &x[2];

   // Using the correct syntax for printing with cout
   cout << "*ptr = " << *ptr << endl;   // 3
   cout << "*(ptr+1) = " << *(ptr+1) << endl; // 4
   cout << "*(ptr-1) = " << *(ptr-1) << endl;  // 2

   return 0;
}
```

Example 6: Pass by reference using pointers

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

void increment(int* num);
int main()
{
    int num=0;
    increment(&num);
    cout<<num;
}

void increment(int* num)
{
        (*num)++;
}
```

Example 7:

```
#include <iostream>
using namespace std;
  void convertToUppercase( char * );

   int main()
   {
     char phrase[] = "characters and $32.98";

     cout << "The phrase before conversion is: " << phrase;
     convertToUppercase( phrase );
     cout << "\nThe phrase after conversion is:  "
          << phrase << endl;

     return 0;  // indicates successful termination

   } // end main

   void convertToUppercase( char *sPtr )
   {
         while ( *sPtr != '\0' ) // check for current character is not '\0'
        {
       if ( islower( *sPtr ) )  // if character is lowercase,
          *sPtr = toupper( *sPtr );  // convert to uppercase

        ++sPtr;  // move sPtr to next character in string

     } // end while

   } // end function convertToUppercase
```

4. **Dynamic Memory Allocation:**

   Dynamic memory allocation in C++ allows you to allocate and deallocate memory at runtime using pointers. This is useful when you don't know the size of data in advance or need to manage memory manually.

i. **Allocating Memory for a single variable:**

Example 8:

// Allocating memory

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

int main()
{
    int *ptr = new int; // Allocates dynamic memory
    //new is the key word used while allocating dynamic memory;

    *ptr = 100;

    cout<<*ptr;
}
```

Note:

- **It is important to delete the allocated memory when the variable is no longer needed because it can cause issues such as memory leaks.**
- **It is important to make the deleted pointer *NULL* as it can cause dangling pointer issues.**

ii. Deallocating Memory:

Example 9:

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

int main()
{
    int *ptr = new int; // Allocates dynamic memory
    //new is the key word used while allocating dynamic memory;

    *ptr = 100;

    cout<<*ptr;

    delete ptr; // Deallocates memory using delete keyword

    //Set the pointer "ptr" to null
    ptr=NULL;

    if(ptr==NULL)
    {
        cout<<"\nThe pointer cannot be de-referenced anymore";
    }

    else
    {
        cout<<*ptr;
    }

}
```

5. **Dynamic Arrays:**
    - A dynamic array in C++ is an array whose size is determined at runtime, rather than at compile time.
    - This allows for flexibility in managing memory, but it requires manual memory allocation and deallocation.

Example 10:

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

int main()
{
    int *arr;
    int size;

    cout<<"Enter the size you want to allocate to array: ";
    cin>>size;

    arr = new int[size];

    cout<<"Enter the "<<size<<" values: ";
    for(int i=0;i<size;i++)
    {
        cin>>arr[size];
    }

    delete[] arr;
    arr=NULL;

}
```

- - - - -> We will discuss Dynamic Multidimensional Arrays in the next Lab

Tasks:
NOTE: USE FUNCTIONS, DYNAMIC MEMORY ALLOCATION, AND DYNAMIC ARRAYS TO SOLVE THESE QUESTIONS

1. **Assume there are two arrays as given below**

   Part A)

   array_A:

54	**2**	**37**	**7**	**45**

array_B:

51	**2**	**45**	**1**	**6**

Write a c++ program to determine (i) how many integers are placed at same location with the same value in both arrays.

PART B:

Write a c++ program to check How many elements are common in both arrays

2. **Create a program to swap two numbers using dynamic arrays.**

3. **Write a program to reverse the elements of an array using dynamic arrays.**

4. **Implement a function to perform matrix addition using dynamic arrays.**

5. **Implement a function to resize a dynamically allocated array in C++. The function should take an existing dynamic array, copy its contents into a new larger array, and return the new array.**