National University
Of Computer and Emerging Sciences

# Lab14

## Summary

| Items | Description |
|---|---|
| Course Title | Programming Fundamentals |
| Lab Title | Pointers and Dynamic Memory Allocation in C++ |
| Duration | 3 Hours |
| Operating System/Tool/Language | Ubuntu/ g++/ C++ |
| Objective | To get familiar with Pointers and Dynamic Memory Allocation in C++ |

# Pointers

In C++, pointers are variables that store the memory addresses of other variables. Here is how we can declare pointers.

```
int *pointVar;
```

Here, we have declared a pointer *pointVar* of the *int* type.

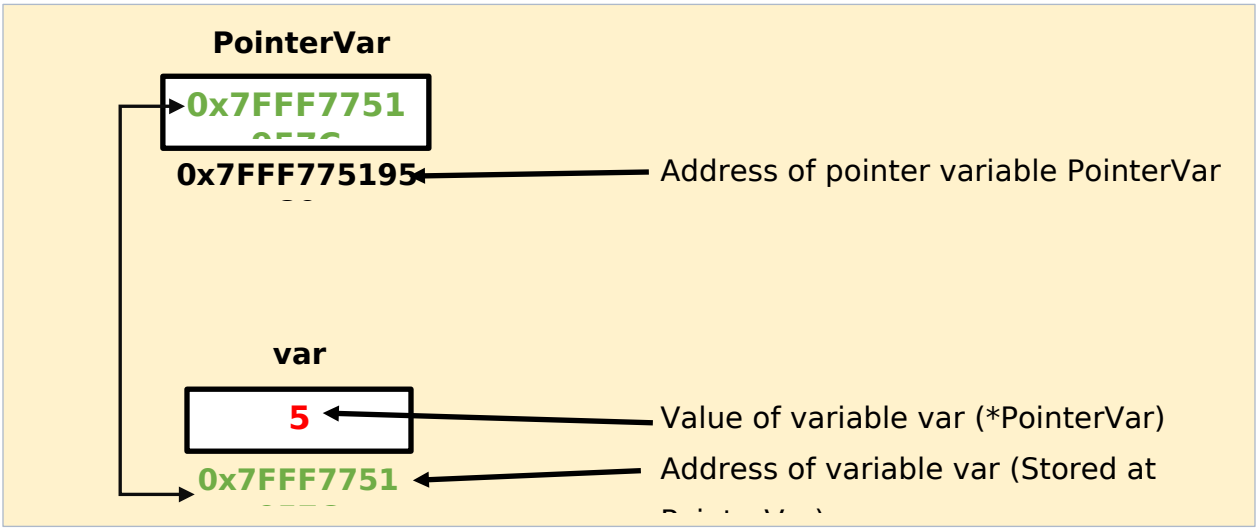
### Assigning Addresses to Pointers

Here is how we can assign addresses to pointers:

```
int *pointVar, var;
var = 5;
// assign address of var to pointVar pointer
pointVar = &var;
```

Here, *5* is assigned to the variable *var*. And, the address of *var* is assigned to the *pointVar* pointer with the code *pointVar = &var*.

**Let's see graphical representation of pointers:**




## Get the Value from the Address Using Pointers

To get the value pointed by a pointer, we use the * operator. For

```
int *pointVar, var;
var = 5;

// assign address of var to pointVar
pointVar = &var;

// access value pointed by pointVar
cout << *pointVar << endl;          // Output: 5
```

In the above code, the address of var is assigned to *pointVar*. We have used the **pointVar* to get the value stored in that address.

When * is used with pointers, it's called the **dereference operator**. It operates on a pointer and gives the value pointed by the address stored in the pointer. That is, **pointVar = var*.

### Changing Value Pointed by Pointer

If *pointVar* points to the address of *var*, we can change the value of *var* by using **pointVar*. For example,

```
int var = 5;
int* pointVar;

// assign address of var
pointVar = &var;

// change value at address pointVar
*pointVar = 1;

cout << var << endl;   // Output: 1
```

Here, *pointVar* and *&var* have the same address, the value of *var* will also be changed when **pointVar* is changed.

# C++ Pointers and Arrays

In C++, Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array. Consider this example:

```
int *ptr;
int arr[5];

 // store the address of the first element of arr in ptr
ptr = arr;
```

Here, *ptr* is a pointer variable while *arr* is an int array. The code *ptr = arr;* stores the address of the first element of the array in variable *ptr*. Notice that we have used *arr* instead of *&arr[0]*. This

is because both are the same. So, the code below is the same as the code above.

The addresses for the rest of the array elements are given by *&arr[1]*, *&arr[2]*, *&arr[3]*, and *&arr[4]*.

## Point to Every Array Elements

```
// ptr + 1 is equivalent to &arr[1]
// ptr + 2 is equivalent to &arr[2]
// ptr + 3 is equivalent to &arr[3]
// ptr + 4 is equivalent to &arr[4]
```

Similarly, we can access the elements using the single pointer.

```
// use dereference operator
    *ptr == arr[0];
// *(ptr + 1) is equivalent to arr[1];
// *(ptr + 2) is equivalent to arr[2];
// *(ptr + 3) is equivalent to arr[3];
// *(ptr + 4) is equivalent to arr[4];
```

Suppose if we have initialized *ptr = &arr[2];* then

```
// ptr - 2 is equivalent to &arr[0];
// ptr - 1 is equivalent to &arr[1];
// ptr + 1 is equivalent to &arr[3];
// ptr + 2 is equivalent to &arr[4];
```

## Example:

Write a program that asks the user to enter integers as inputs to be stored in the variables **a** and **b** respectively. Also create two integer pointers named **ptrA** and **ptrB**. Assign the addresses of **a** and **b** to **ptrA** and **ptrB** respectively. Display the values and addresses of **a** and **b** using **ptrA** and **ptrB**.

```
#include <iostream>
using namespace std;
int main(){
	int a, b, *ptrA, *ptrB;
	cout<<"Enter first integer: ";
	cin>>a;

	cout<<"Enter 2nd integer: ";
	cin>>b;

	ptrA = &a;
	ptrB = &b;

	cout<<"Value of a= "<<*ptrA<<endl;
	cout<<"Value of b= "<<*ptrB<<endl;
	cout<<"Address of a= "<<ptrA<<endl;  // In your compiler addresses may be different.
	cout<<"Address of b= "<<ptrB<<endl;
	return 0;
}
```

```
Enter first integer: 10
Enter 2nd integer: 20
Value of a= 10
Value of b= 20
Address of a= 0x70fdfc
Address of b= 0x70fdf8
```

# Dynamic Memory Allocation (1D and 2D)

Dynamic memory allocation in C/C++ refers to performing memory allocation manually by the programmer.

### *new* operator:

The *new* operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, *new* operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

### Syntax to use new operator:

```
// Pointer initialized with NULL
int *p = NULL;

// Then request momory for the variable
p = new int;

//              OR

// Combine declaration of pointer and their assignment
int *p = new int;
```

### Initialize memory:

```
// data_type Pointer_variable_name  =  new  data_type(value);
int  *p  =  new int(35);
float  *p1  =  new  float(15.25);
```

**Allocate a block of memory:**

```
int  *p  =  new int[10];
```

**delete operator:**

Since it is the programmer's responsibility to de-allocate dynamically allocated memory, programmers are provided delete operator by C++ language.

```
delete  p;
delete  p1;
```

To free the dynamically allocated array pointed by pointer-variable, use the following form of delete:

```
//  It will free the entire array pointed p.
delete[ ]  p;
```

If enough memory is not available in the heap to allocate, the new request indicates failure by throwing an exception of type **std::bad_alloc**, unless "**nothrow**" is used with the *new* operator, in which case it returns a *NULL* pointer. Therefore, it may be a good idea to check for the pointer variable produced by *new* before using it in the program.

```
int  *p  =  new(nothrow)  int;
if(!p){
    cout<<"Memory allocation faild.\n";
}
```

**Following is a simple example demonstrating DMA in a single-dimensional array.**

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

//  Dynamically allocate memory for 1d array in c++
int main(){
        int N, i;
        cout<<"Enter size of array: ";
        cin>>N;

//      Dynamically allocate memory of size N
        int *array = new int[N];

//      Assign values of allocated memory
        for(i = 0; i < N; i++){
                cout<<"array["<<i<<"]= ";
                cin>>*(array+i);
        }
//      Print the 1D array
        cout<<"\nValues of array\n";
        for(i = 0; i < N; i++){
                cout<<array[i]<<" ";   // is equal to cout<<*(array+i);
        }

//      deallocate memory
        delete[] array;

        return 0;
}
```

```
Enter size of array: 5
array[0]= 10
array[1]= 20
array[2]= 30
array[3]= 40
array[4]= 50

Values of array
10 20 30 40 50
```

**Following is a simple example demonstrating DMA in 2 dimensional array**

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

//  Dynamically memory allocation in C++ for 2d array
int main(){
        int M, N, i, j;
        cout<<"Enter number rows for 2D array: ";
        cin>>M;
        cout<<"Enter number columns for 2D array: ";
        cin>>N;

//      Dynamically create array of pointers of size M
        int **array = new int*[M];

//      Dynamic allocate memory of size N for each row
        for(i=0; i<M; i++){
                array[i] = new int[N];
        }
//      Assign values of allocated memory
        cout<<"\nValues of array\n";
        for(i = 0; i < M; i++){
                for(j = 0; j < N; j++){
                        cout<<"array["<<i<<"]["<<j<<"]= ";
                        cin>>array[i][j];
                }
        }
        cout<<endl;
//      Print the 2D array
        for(i = 0; i < M; i++){
                for(j = 0; j < N; j++){
                        cout<<array[i][j]<<"  ";
                }
                cout<<endl;
        }
//      deallocate memory
        for(i = 0; i < M; i++){
                delete[] array[i];
        }
        delete[] array;
        return 0;
}
```

```
Enter number rows for 2D array: 3
Enter number columns for 2D array: 4

Values of array
array[0][0]= 1
array[0][1]= 2
array[0][2]= 3
array[0][3]= 4
array[1][0]= 5
array[1][1]= 6
array[1][2]= 7
array[1][3]= 8
array[2][0]= 9
array[2][1]= 8
array[2][2]= 7
array[2][3]= 6

1 2 3 4
5 6 7 8
9 8 7 6
```

## Dynamic Memory Allocation:

- We went through pointers and variables in the above two examples
  - Those variables were statically declared and used in pointer
  - The variables are having a particular name
- Now we understand dynamically declaring variables and using them using pointers
  - Unlike static variables, the variables created dynamically doesn't have a name
- New keywords we shall use in dynamic memory allocation are, **`new`** and **`delete`**
- Keyword "new" is used to allocate memory (dynamic) in the Heap.
- Keyword "delete" is used to deallocate memory (dynamic) from the Heap.
- We can create a single location in the Heap or an entire array.
  - For single slot allocation:

```
int *ptr = new int;
```

  - For array allocation:

```
int *ptr = new int[3];
```

- The "`*ptr`" is used to access these dynamically allocated locations in Heap.

Terminology

| Term | Meaning/Use |
|---|---|
| **&** | Address-of, Reference Declarator |
| ***** | Dereference Operator |
| **new** | Allocate Memory |
| **delete** | Deallocate Memory |
| **new int[3]** | Allocate Memory to a Fixed Size Array of Three Integers |
| **delete []** | Deallocation Memory of Array |

# Lab Tasks

## Task#01

Write a program that prints the menu as shown below, the program should use a function showChoices to display the choices.

```
showChoice()
{
cout<<"Press 2 for task 2"<<endl;
.
.
.
}
```

The tasks that you attempt later should be part of this menu as separate functions.

For example:

```
task2();
task3();
```

Where task2() is a function having complete code for problem2 and so on.

Note: Please write the prototype of each function

## Task#02

*Create a function that takes five pointers as arguments to find the sum of 5 numbers using pointers.*

## Task#03

Create a C++ program to swap the values of two variables using pointer notation.

## Task#03

Create a C++ program to find the largest number from the array of 7 numbers using pointer.

## Task#04

Create a program which print the table of 2 upto 12 using pointers..

## Task#05

Write a C++ program where you keep entering integers one after another. All the numbers should keep summing along with each other. The stopping condition should be an entered negative integer.

Note: No static variable should be declared. Use DMA.