National University of Computer and Emerging Sciences Islamabad

Programming Fundamentals Lab Fall 2022

Lab Manual 15

Pointers, Pointers Arithmetic, Pointers with Arrays, DMA

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. We can also declare pointers in the following way.

```
int* pointVar;    // preferred syntax
```

Let's take another example of declaring pointers.

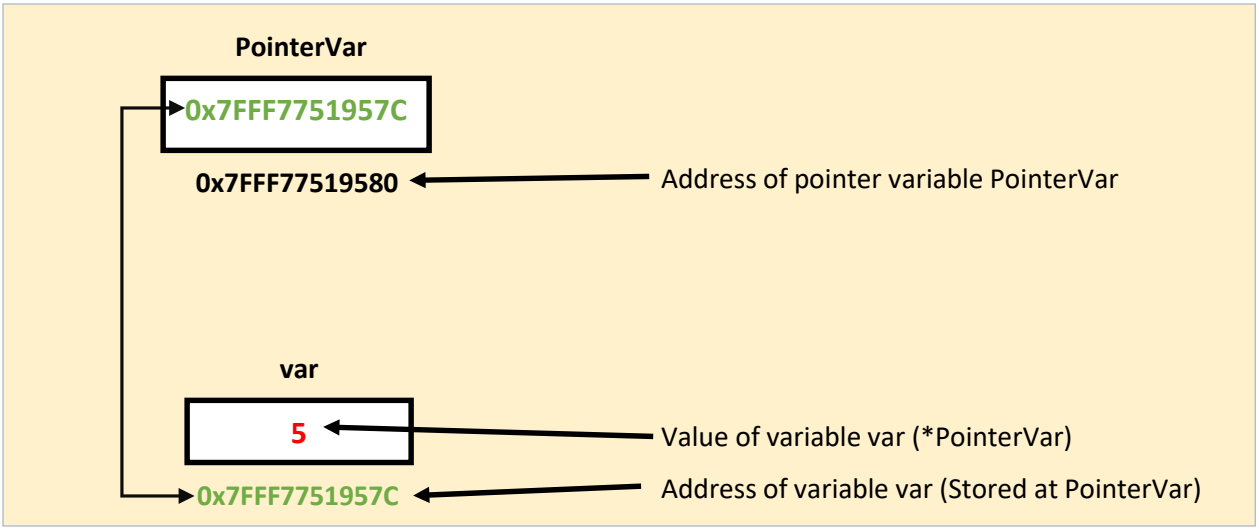
Assigning Addresses to Pointers

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 example:

```
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
```

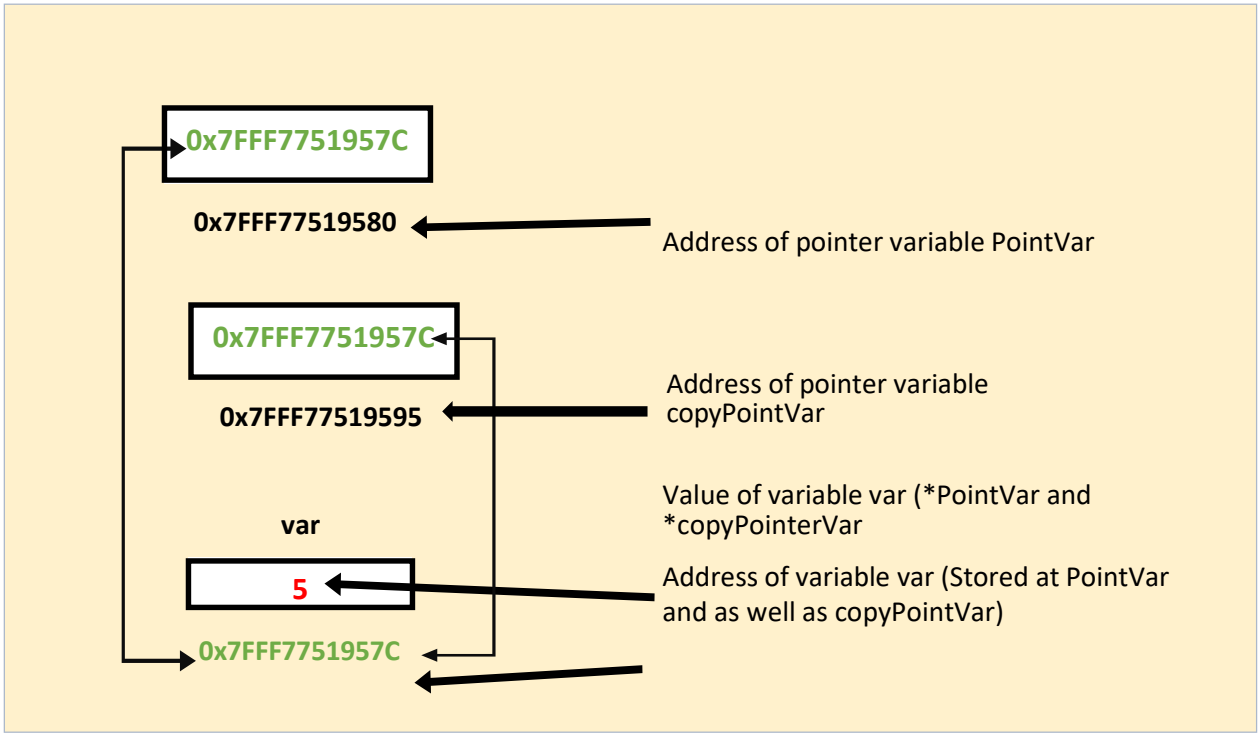
Here, *pointVar* and *&var* have the same address, the value of *var* will also be changed when **pointVar* is changed.

Copy of Pointer

As we can make copy of normal variables we can also make copy of pointers.

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

// assign address of var
pointVar = &var;

int* copyPointVar = pointVar
*copyPointVar = 1;

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

Here, you can see pointVar and copyPointVar both have the same address means both are pointing to same memory location, you can also visualize in below picture.


0x7FFF7751957C
0x7FFF77519580
Address of pointer variable PointVar
0x7FFF7751957C
0x7FFF77519595
Address of pointer variable copyPointVar
Value of variable var (*PointVar and *copyPointerVar
var
5
Address of variable var (Stored at PointVar and as well as copyPointVar)
0x7FFF7751957C

Pointer Arithmetic

There are only four arithmetic operators that can be used on pointers: ++, --, +, -

- Let p1 be an integer pointer which contains the address 5000
  - p1++; // now p1 will be 5004
- Each time p1 is incremented, it shall point to the next integer.
  - p1--; will cause p1 to be 4996 if initially it was 5000.
- Let p1 be an integer pointer which contains the address 5000
  - p1++; // now p1 will be 5004
- Each time p1 is incremented, it shall point to the next integer.
  - p1--; will cause p1 to be 4996 if initially it was 5000.

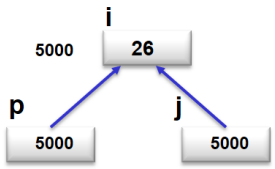
Pointer of type other than char shall increase or decrease by length of base type. You cannot add two pointers but you can subtract two pointers (if they are of same base type). Other than addition or subtraction of a pointer and an integer, OR subtraction of two pointers no other arithmetic operations can be performed on pointers.

```
int main()
{
   int *iptr, a;

   double *fptr, b;
   int x;

    a = 10;
    b = 20;
    iptr  = &a;
    fptr  = &b;

    for ( x = 0; x < 10; x++)
        cout << iptr+x << "     " << fptr+x << '\n';

    return 0;
   }
```

Passing Pointers to Functions

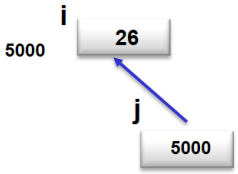
No big deal. Just declare parameter as ***type *.***

```
#include <iostream.h>
void f(int *j);//or void f(int *);
int main()
{
  int i = 26;
  int *p;
  p = &i; // p now points to i
  f(p);
  cout << i;  // i is now 100
  return 0;
}
void f(int *j)
{
 *j = 100; // var pointed to by j is assigned 100
}
```



The pointer variable not necessary. Can generate and pass the address of **i** as such to **f()**

```
#include <iostream.h>
 void f(int *j);
 int main()
 {
   int i;
   f(&i);
   cout << i;
   return 0;
 }

 void f(int *j)
 {
   *j = 100; // var pointed to by j is assigned 100
 }
```



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 cellsof 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. For example,

```
// 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;
        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
```

In your compiler addresses may be different.

```
// Using Subscripting and Pointer Notations with Arrays

// Example 1
int b[] = {10, 20, 30, 40};
int *bptr = b;   // set bptr to point to array b

cout << "Array b printed with:" << endl<< "Array subscript notation" << endl;

for (int i = 0; i <= 3; i++)
        cout <<b[i] << endl;

cout << "Pointer subscript notation" << endl;

for (i = 0; i <= 3; i++)
        cout << bptr[i] << endl;

// Example 2
int offset;
cout << "Pointer/offset notation where" << endl << "the pointer is the array name" << endl;

for (offset = 0; offset <= 3; offset++)
        cout << *(b + offset) << endl;

cout <<"Pointer/offset notation" << endl;

for (offset = 0; offset <= 3; offset++)
        cout << *(bptr + offset) << endl;
```

2D Array and Pointers

Two-dimensional arrays use rows and columns to identify array elements. This type of array needs to be mapped to the one-dimension address space of main memory. The following declares a two-dimensional array with two rows and three columns. The array is initialized using a block statement. Figure 4-2 illustrates how memory is allocated for this array. The diagram on the left shows how memory is mapped. The diagram on the right shows how it can be viewed conceptually:

// Traversal of 2D using Subscript Notation of above matrix.

```
int rows = 2, cols = 3;
for(int i=0; i < rows; i++){
        for(int j=0; j < cols; j++){
                cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
}
```

// Traversal of 2D using Pointer Notation of above matrix.

```
int rows = 2, cols = 3;
for(int i=0; i < rows; i++){
        for(int j=0; j < cols; j++){
                cout<< *((matrix + i*cols) + j) <<" ";
        }
        cout<<endl;
}
```

```
int matrix[2][3] = {{1,2,3},{4,5,6}};
```

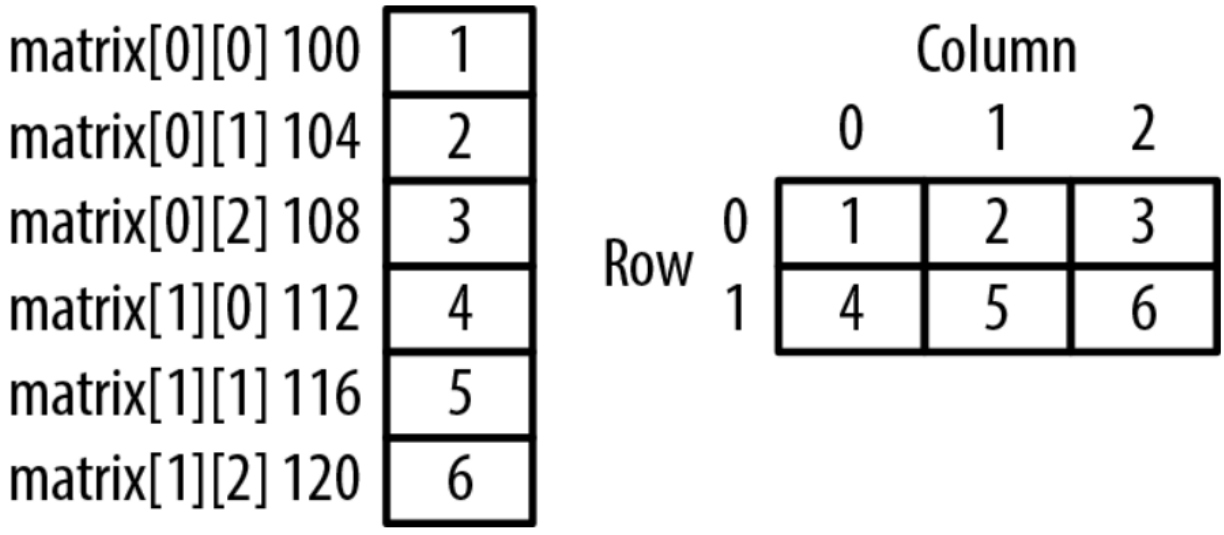


Figure 4-2. Two-dimensional array

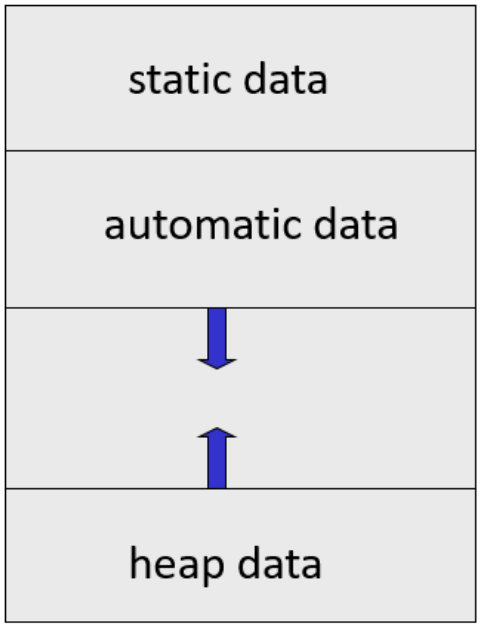
Passing array to Function

Arrays are always passed by reference as shown below.

```
#include <iostream.h>
void cube(int *n, int num);
int main()
{
  int i, nums[10];
  for(i=0; i<10; i++) nums[i] = i+1;
  cout << "Original contents: ";
  for(i=0; i<10; i++) cout << nums[i] << ' ';
  cout << '\n';
  cube(nums, 10); // compute cubes
  cout << "Altered contents: ";
  for(i=0; i<10; i++) cout << nums[i] << ' ';
  return 0;
}
void cube(int *n, int num)
{
    while(num)
    {
       *n = *n * *n * *n;
       num--;
       n++;
    }
}
```

Dynamic Memory Allocation

A good understanding of how dynamic memory really works in C++ is essential to becoming a good C++ programmer. Memory in your C++ program is divided into two parts. **The stack** – All variables declared inside the function will take up memory from the stack. **The heap** – This is unused memory of the program and can be used to allocate the memory dynamically when program runs.



space for global variables

run-time stack - activation records added and removed as program runs (expands and shrinks in an orderly LIFO manner)

space for variables allocated at run-time using "new" (allocation and de-allocation requests occur in unpredictable order)

Heap variable are accessed indirectly via a pointer variable memory space is *explicitly* allocated at run-time (using new) space is allocated from an area of run-time memory known as the heap in C++ space must be *explicitly* returned (using delete) to avoid "memory leak" C++ programmers are responsible for memory management.

```
//Heap Allocation
  int *xptr = new int;
  *xptr = 73;
// Heap Allocation, Deallocation and Initialization
  int *myptr = new int(73);
  cout << *myptr;

  delete myptr;

  //declaring and initializing int
```

Lab Tasks

Problem 1:

Write a program in C++ to add two numbers **using pointers.**

Test Data :

Input the first number : 5

Input the second number : 6

The sum of the entered numbers is : 11

Problem 2:

Write a C++ program to accept five integer values from the user and store them in an array. Then find and print the largest and smallest integer value in the array using **pointers**.

Output:

```
Enter the value of Array.
Array[0]= 5
Array[1]= 4
Array[2]= 7
Array[3]= 2
Array[4]= 9

Smallest: 2
Largest: 9
```

Problem 3:

Write a C++ function ADD that accept three parameters a, b and c the function adds a and b and store the result in c. You have to call this function from main and after returning you have to print c and it should have the result of a+b.

Problem 4:

Write a C++, which creates the matrix of 5 x 5, initialized it with 1's after that you have to fill the diagonal with zeros using pointer notation

Original Array				
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Processed Array				
0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Problem 5:

Write a C++ code which create an array of 10 on heap and also fill that array with some numbers.