Data Structure Lab

Lab # 01

Pointers and Arrays in C++

Instructor: Muhammad Saood Sarwar

Email: saood.sarwar@nu.edu.pk

Course Code: CL2001

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

Arrays:

The array is the most commonly used data storage structure; it's built into most programming languages. Because arrays are so well known, they offer a convenient jumping off place for introducing data structures and for seeing how object-oriented programming and data structuresrelate to one another.

Example 1 | Array Input/Output

```
#include <iostream>

using namespace std;

int main () {

int arr[5], i;

cout<<"Enter Values into an array: "<<endl;

for(int i=0 ; i<=4 ; i++)

{

cin>>arr[i];

}

cout<<"Printing elements of an array: "<<endl;

for(int i=0 ; i<=4 ; i++)

{  cout<<arr[i]<<"\t"; }

cout<<endl;

return 0;}
```

Output

```
Enter Values into an array:

3

4

78

1

54
```

```
Printing elements of an array:

3       4       78      1       54
```

Pointers

Pointers are the most powerful feature of C and C++. These are used to create and manipulate data structures such as linked lists, queues, stacks, trees etc. The virtual functions also require the use of pointers. These are used in advanced programming techniques. To understand the use of pointers, the knowledge of memory locations, memory addresses and storage of variables in memory is required.

Memory addresses & Variables

Computer memory is divided into various locations. Each location consists of 1 byte. Each byte has a unique address.

When a program is executed, it is loaded into the memory from the disk. It occupies a certain range of these memory locations. Similarly, each variable defined in the program occupies certain memory locations. For example, an int type variable occupies two bytes and float type variable occupies four bytes.

When a variable is created in the memory, three properties are associated with it. These are:

- Type of the variable
- Name of the variable
- Memory address assigned to the variable.

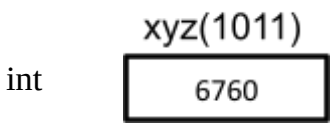
For example, an integer type variable xyz is declared as shown below

int xyz =6760;

int represents the data type of the variable.

xyz represents the name of the variable.

When variable is declared, a memory location is assigned to it. Suppose the memory address assigned to the above variable xyz is 1011. The attribute or properties of this variable can be shown as below:



The box indicates the storage location in the memory for the variable xyz. The value of the variable is accessed by referencing its name. Thus, to print the contents of variable xyz on the computer screen, the statement is written as:

cout<<xyz

The memory address where the contents of a specific variable are stored can also be accessed. The **address operator (&)** is used with the variable name to access its memory address. The address operator (&) is used before the variable name.

For example, to print the memory address of the variable xyz, the statement is written as:

cout<<&xyz

The memory address is printed in hexadecimal format.

Pointer Variables

The variables that are used to hold the memory address of another variable is called a pointer variable or simply pointer.

The data type of the variable (whose address a pointer is to hold) and the pointer variable must be the same. A pointer variable is declared by placing an asterisk (*) after data type or before the variable name in the data type statement.

For example, if a pointer variable **"p"** is to hold memory address of an integer variable, it is declared as:

int* p;

Similarly, if a pointer variable **"rep"** is to hold memory address of a floating-point variable, it is declared as:

float* rep;

The above statements indicate that both "**p"** and **"rep"** variable are pointer variables, and they can hold memory address of integer and floating-point variable respectively.

Although the asterisk is written after the data type, is usually more convenient to place the asterisk before the pointer variable. i.e. **float *rep;**

Example 2 | Working of Pointers

```
#include <iostream>

using namespace std;

int main() {

int a, b;

int *x, *y;
```

```
a =  33;

b =  66;

x = &a;

y = &b;

cout<<"Content of x= "<<*x<<endl;

cout<<"Content of y= "<<*y<<endl;

cout<<"Memory address of variable a= "<<x<<endl;

cout<<"Memory address of variable b= "<<y<<endl;

cout<<"Memory address of pointer  x= "<<&x<<endl;

cout<<"Memory address of pointer  y= "<<&y<<endl;

 return 0; }
```

Output

```
Content of x= 33

Content of y= 66

Memory address of variable a= 0x7fff5ce07330

Memory address of variable b= 0x7fff5ce07334

Memory address of pointer  x= 0x7fff5ce07338

Memory address of pointer  y= 0x7fff5ce07340
```

A pointer variable can also be used to access data of memory location to which it points.

In the above program, **x** and **y** are two pointer variables. They hold memory addresses of variables **a** and **b**. To access the contents of the memory addresses of a pointer variable, an asterisk (*) is used before the pointer variable.

For example, to access the contents of **a** and **b** through pointer variable **x** and **y**, an asterisk is used before the pointer variable. For example,

```
cout<<"Value in memory address x = "<<*x<<endl;

cout<<"Value in memory address y = "<<*y<<endl;
```

Notice the difference between the **reference** and **dereference** operators:

- & is the **reference operator** and can be read as "**address of**"
- * is the **dereference operator** and can be read as "**value pointed by**"

Thus, they have complementary (or opposite) meanings. A variable referenced with & can be dereferenced with *.

Example 3

Write a program to assign a value to a variable using its pointer variable. Print out the value using the variable name and print out the memory address of the variable using pointer variable.

```
#include <iostream>

using namespace std;

int main () {

int *p;

int a;

p=&a;

cout<<"Enter data value? ";

cin>>*p;

cout<<"Value of variable    ="<<a<<endl;

cout<<"Memory Address of variable= "<<p<<endl;

  return 0;

}
```

Output

```
Enter data value? 45

Value of variable    =45

Memory Address of variable= 0x7fffc2f1164c
```

The "void" Type Pointers

Usually the type of variable and type of pointer variable that holds memory address of the variable must bethe same. But the "**void**" type pointer variable can hold memory address of variables of any type. A

void type pointer is declared by using keyword "**void**". The asterisk is used before pointer variable name.

Syntax for declaring void type pointer variable is:

void *p;

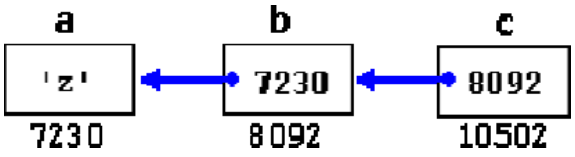
The pointer variable "**p**" can hold the memory address of variables of any data type.

Pointers to Pointers

C++ allows the use of pointers that point to pointers, that these, in its turn, point to data (or even to other pointers). To do that, we only need to add an asterisk (*) for each level of reference in their declarations:

```
char a;
char *b;
char **c;
a = 'z';
b = &a;
c = &b;
```

This, supposing the randomly chosen memory locations for each variable of 7230, 8092 and 10502, could be represented as:



The value of each variable is written inside each cell; under the cells are their respective addresses in memory. The new thing in this example is variable c, which can be used in three different levels of indirection, each one of them would correspond to a different value:

Example 4

```
#include <iostream>

using namespace std;

int main ()
```

```
{
int a;
int *b;
int **c;
a = 7;
b = &a;
c = &b;
cout<<"The Address of the Vairiable a is: "<<b<<endl;
cout<<"The Address of the Vairiable b is: "<<c<<endl;
return 0;
}
```

Output:

```
The Address of the Vairiable a is: 0x7bfe14
The Address of the Vairiable b is: 0x7bfe08
```

Pointers and Arrays

There is a close relationship between pointers and arrays. In Advanced programming, arrays are accessed using pointers.

Arrays consist of consecutive locations in the computer memory. To access an array, the memory location of the first element of the array is accessed using the pointer variable. The pointer is then incremented to access other elements of the array. The pointer is increased in value according to the size of the elements of the array.

When an array is declared, the array name points to the starting address of the array. For example, consider the following example.

int x[5];

int *p;

The array" **x**" is of type **int** and "**p**" is a pointer variable of type **int**.

To store the starting address of array "x" (or the address of first element), the following statement is used.

p = x;

The address operator (&) is not used when only the array name is used. If an element of the array is used, the & operator is used. For example, if memory address of first element of the array is to be assigned to a pointer, the statement is written as:

p = &x[0];

when integer value 1 is added to or subtracted from the pointer variable "**p**", the content of pointer variable "p" is incremented or decremented by (1 x size of the object or element), it is incremented by 1 and multiplied with the size of the object or element to which the pointer refers.

For example, the memory size of various data types is shown below:

- The array of **int** type has its object or element size of **2 bytes**. It is **4 bytes** in Xenix System.
- The array of type **float** has its object or element size of **4 bytes**.
- The array of type **double** has its object or element size of **8 bytes**.
- The array of type **char** has its object or element size of **1 byte**.

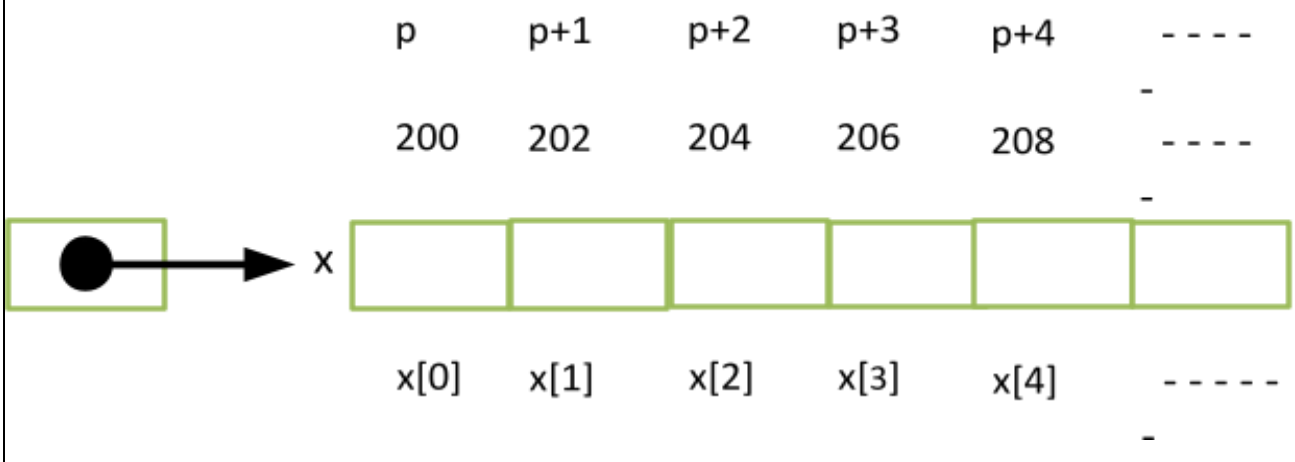
Suppose the location of the first element in memory is 200. i.e., the value of pointer variable "p" is 200, and it refers to an integer variable.

When the following statement is executed,

p=p+1;

the new value of "**p**" will be 200+(1*2)=202. All elements of an array can be accessed by using this technique.

The logical diagram of an integer type array "**x**" and pointer variable "**p**" is that refers to the elements of the array "x" is given below:



Example 5

Write a Program to input data into an array and then to print on the computer screen by using pointer notation.

```
#include <iostream>

using namespace std;

int main () {

int arr[5], *ptr1,*ptr2, i;

cout<<"Enter Values into an array: "<<endl;

for(int i=0 ; i<=4 ; i++)

{

   cin>>arr[i];

}

ptr1=arr;

cout<<"Values from array Using Pointer notation: "<<endl;

for(int i=0 ; i<=4 ; i++)

{

   cout<<*ptr1++<<"\t";

}

cout<<endl;

ptr2 = arr;

cout<<"Using While Loop: "<<endl;

while (*ptr2){

   cout<<*ptr2++<<"\t";

}

cout<<endl;

  return 0;

}
```

Output:

```
Enter Values into an array:

8

7

6

5

4

Values from array Using Pointer notation:

8      7      6      5      4

Using While Loop:

8      7      6      5      4
```