Data Structures-LAB

Lab Manual # 01

Pointers in C++

Instructor: Fariba Laiq

Semester Fall-2023

Course Code: CL2001

Fast National University of Computer and Emerging Sciences Peshawar

Department of Computer Science & Software Engineering

Data Structures-LAB

Table of Contents

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.

The variables that is 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;**

```
#include<iostream>
using namespace std;
int main() {

	int a=2, b=3;
	int *x, *y;
	x = &a;
	y = &b;

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

	cout<<"\nValue of pointer  x= "<<x<<endl;
	cout<<"Value of pointer  y= "<<y<<endl;

	//dereferencing
	cout<<"\nAccessing value of a using its pointer x = "<<*x<<endl;
	cout<<"Accessing value of b using its pointer y = "<<*y<<endl;
	return 0;
}
```

Output:

```
Memory address of variable a= 0x78fe0c

Memory address of variable b= 0x78fe08



Value of pointer  x= 0x78fe0c

Value of pointer  y= 0x78fe08



Accessing value of a using its pointer x = 2

Accessing value of b using its pointer y = 3
```

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,

Program

Write a program to input a value to a variable using its pointer variable. Print out the value using the pointer variable.

```
#include<iostream>

using namespace std;

int main() {



        int a;

        int *x=&a;

        cout<<"Enter a value: "<<endl;

        cin>>*x;

        cout<<"Value of a is: "<<*x<<endl;

        cout<<"Memory address of a is: "<<x<<endl;

        return 0;

}
```

Output:

```
Enter a value:

5

Value of a is: 5

Memory address of a is: 0x78fe14
```

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 the 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**.

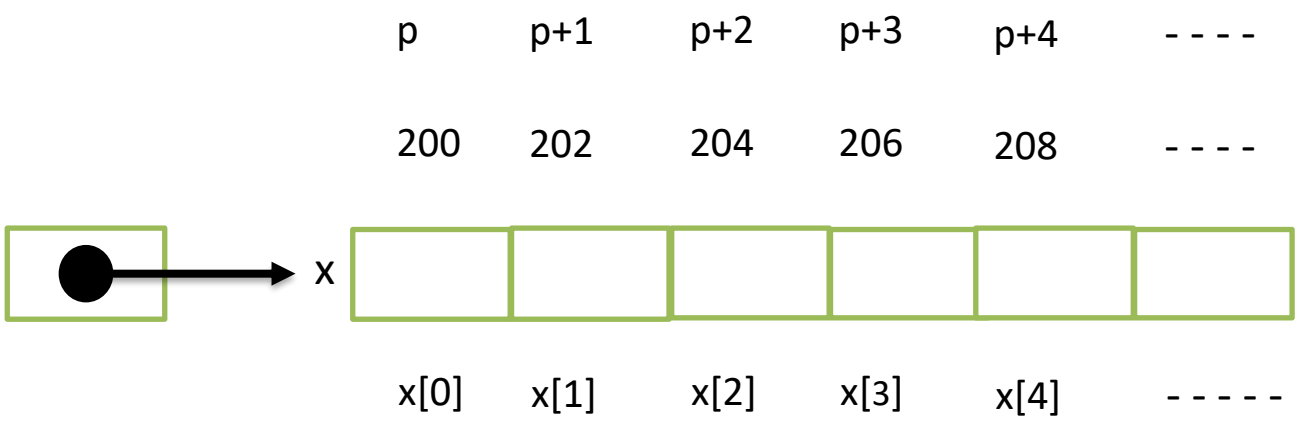
Suppose the location of first element in memory is 200. i.e., the value of pointer variable "p" is 200, and it refers to an integer variable.

When the following statement is executed,

p=p+1;

the newly 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:



Program

In the below program, an integer array with a size of 5 and an integer pointer p are initialized. The pointer p is initialized to point to the memory location of the first element of the array. In the subsequent for loop, user input values are sequentially stored in the array using the pointer p, with the pointer being incremented to point to the next memory location after each input. Following this, the pointer p is reset to the start of the array. In the second for loop, the values stored in the array are accessed and displayed using the pointer p.

```
#include<iostream>

using namespace std;

int main() {

	int size=5;

	int arr[size];


	int *p=arr;

	for(int i=0; i<size; i++) {

		cout<<"Enter value at index: "<<i<<endl;

		cin>>*p;

		p++;

	}

	p=arr;

	cout<<"Values in array: "<<endl;

	for(int i=0; i<size; i++) {

		cout<<*p<<" ";

		p++;

	}

}
```

Output:

```
Enter value at index: 0

1

Enter value at index: 1

2

Enter value at index: 2

3

Enter value at index: 3

4

Enter value at index: 4

5

Values in array:

1 2 3 4 5
```

Passing Pointers as Arguments to Functions

The pointer variables can also be passed to functions as arguments. When pointer variable is passed to a function, the address of the variable is passed to the function. Thus, a variable is passed to a function not by its value but by its reference.

The below program defines a function func that takes an integer pointer p as its parameter. Inside the function, the integer value pointed to by p is modified to its square value. In the main function, an integer variable a is declared and assigned a value of 5. An integer pointer p is then assigned the address of a. The original value of a is displayed, showing "Value of a before function: 5". The func function is called with the p pointer as an argument, resulting in the value pointed to by p being squared and modified. The modified value of a is then displayed, demonstrating "Value of a after function: 25". This code illustrates how modifying the value through a pointer inside a function affects the original value outside the function, showcasing the concept of passing values by reference using pointers.

Program

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


void func(int *p)
{
	*p=*p * *p;
}
int main() {


	int a=5;
	int *p=&a;


	cout<<"Value of a before function: "<<*p<<endl;
	func(p);
	cout<<"\nValue of a after function: "<<*p<<endl;
}
```

Output:

```
Value of a before function: 5
Value of a after function: 25
```

Returning Pointers from Function

The below program defines a function that takes an integer pointer p as its parameter. Inside the function, a static integer sq is declared to store the square of the integer pointed to by p. The sq retains its value across function calls. An integer pointer p2 is assigned the address of the static variable sq, allowing it to be returned from the function. In the main function, an integer variable a is declared with a value of 5, and an integer pointer p is assigned the address of a. The value of a is displayed. The calcSquare function is called with p as an argument, leading to the calculation of the square of a and its storage in the static variable. The square value is displayed using the pointer p2.

```
#include<iostream>

using namespace std;


int * calcSquare(int *p)

{

	static int sq=*p * *p;

	int *p2=&sq;

	return p2;

}

int main() {


	int a=5;

	int *p=&a;


	cout<<"Value of a: "<<*p<<endl;

	int *p2=calcSquare(p);

	cout<<"\nSquare of a: "<<*p2<<endl;

}
```

Output:

```
Value of a: 5

Square of a: 25
```

Pointers and Strings

A String is a sequence of characters. A string type variable is declared in the same manner as an array type variable is declared. This is because a string is an array of characters type variables.

Since a string is like an array, pointer variables can also be used to access it. For example:

char st1[] = “Pakistan”;

char *st2 = “Pakistan”

In the above statements, two string variables “**st1**” and “**st2**” are declared. The variable “**st1**” is an array of character type. The variable **“st2**” is a pointer also of character type. These two variables are equivalent. The difference between string variables “**st1**” and “**st2**” is that:

- string variable “**st1**” represents a pointer constant. Since a string is an array of character type, the “**st1**” is the name of the array. Also, the name of the array represents its address its which is a constant. Therefore, **st1** represents a pointer constant.
- string variable “**st2**” represents a pointer variable.

In the following program example, a string is printed by printing its character on by one.

```
#include<iostream>

using namespace std;



void display(char *s)

{

	while(*s!='\0')

	{

		cout<<*s<<endl;

		s++;

	}

}

int main() {

	char word[]="Pakistan";

	display(word);

}
```

The above program defines a function display that takes a pointer to a character (char *s) as its parameter. The function uses a while loop to traverse through the characters of the input string pointed to by s. It displays each character followed by a newline and then increments the pointer s to move to the next character. In the main function, a character array word containing the string "Pakistan" is declared. The display function is called with the word array as the argument. Consequently, the function iterates through the characters of the string and prints each character on a new line.