Chapter:3

Review on Pointers

Pointers in C++

Topics to cover:

- Overview of Pointers
- Pointer Declaration
- Pointer Assignment
- Pointer Arithmetic
- Relations Between Pointers and Arrays
- Pointers and Strings

Overview of Pointers

- A Pointer in C++ is variable whose value is a memory address.
- With pointers many memory locations can be referenced.
- Some data structures use pointers (e.g. linked list, tree).
- The * and & operators
  a. & operator is the address operator
  b. * operator is the dereferencing operator. It is used in pointers declaration

Pointer Declaration

- **Pointers are declared as follows:**

  <type> * variable_name ;

e.g.

```
int  * Ptr;    // Ptr is a pointer to data of type integer

char * Ptr;  // Ptr is a pointer to data of type character

void * Ptr;  // Ptr is a generic pointer, represents any
                type
```

Pointer Assignment

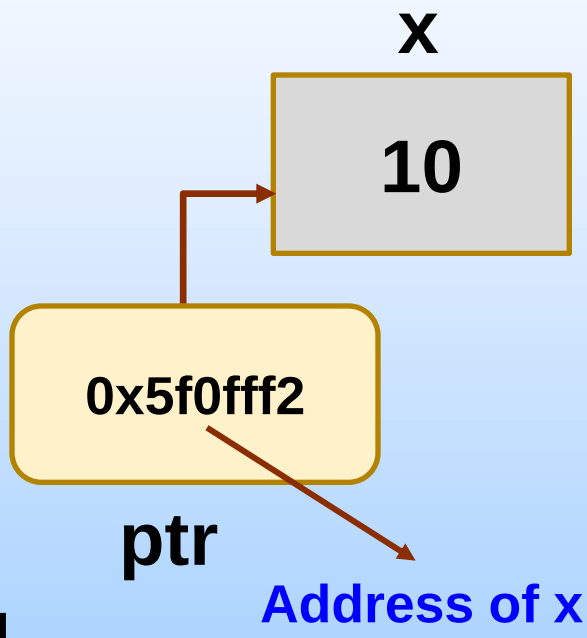
- Assignment can be applied on pointers of the same type
- Example

```
int *xPtr, *yPtr;  int x = 5;

 …

xPtr = & x;   // xPtr now points to address of  x

yPtr = xPtr;  // now yPtr and xPtr point to x
```

Pointer Assignment

- **Example:**

```
int x = 10;
int *ptr ;
ptr= &x;
```



- **Now ptr will contain address where the variable x is stored in memory**

Example:1

```
#include <iostream.h>
main()
{
   int x=5, *ptr;
   ptr = &x;
   cout << “*ptr=“ <<*ptr<<" x="<<x<< endl;
}
```

The output:

```
   *ptr=5 x=5
```

Example:2

```
// using the & and * operators
#include<iostream.h>
main ( )
{
   int a=7;      //a is an integer
   int *ptr;     // ptr is apointer to an integer
   ptr = &a;   // ptr set to address of a
   cout <<" The address of a is " << &a <<endl
          << "The value of ptr is " << ptr<< endl;
```

Example:2

```
cout << “The value of a is “ << a << endl
<< “The value of  *ptr is “ << *ptr<< endl;
cout<<” The * and & are complement of each other.” <<endl<< “ & *ptr = “<< & *ptr
<< endl<< “ *&ptr = “ << *&ptr;
return 0;
}
```

Example:2

The Output:

The address of a is oxfff4

The value of aptr is oxfff4

The value of a is 7

The value of *aptr is 7

The * and & are complements of each other

&* ptr = oxfff4

***& ptr = oxfff4**

Pointers & Functions

When calling the function, the address of the argument is passed, and this is done by writing the address operator of the argument to be processed. When the argument address is passed to the function, the * operator is used to access the value of the variable

Example:1

```
// Cube a variable using call-by-value
#include<iostream.h>
int cube(int);     // prototype
int main( )
{
  int number = 5;
  cout <<" The original value of number is "
          <<number<<endl;
```

Example:1

```
cout << " The new value of number is "
        << cube(number)<< endl;
return 0;
}
int cube (int n)
{
   return n*n*n; // cube local variable n
}
```

Example:1

The Output:

The original value of number is 5

The new value of number is 125

Example:2

- In the following program the address of the variable number is passed as an argument by reference to the function cube( )

Example:2

```
// cube a variable using call-by-reference with a pointer argument
#include<iostream.h>
void cube (int *);      // prototype
main( )
{
  int number = 5;
  cout<< " The original value of number is "
        << number <<endl;
```

Example:2

```
  cube (&number);
  cout<< " The new value of number is " <<
  number <<endl;
  return 0;
}
void cube (int *Ptr)
{
  *Ptr = *Ptr * *Ptr * *Ptr;      // cube number
}
```

Example:2

The Output

The original value of number is 5

The new value of number is 125

Example:3

```
//A program to call a function to swap two numbers.

 #include <iostream.h>
void swap(int *, int *);   // This is swap's prototype
void  main()
{
  int x = 5, y = 7;
  swap(&x , &y);   // calling swap with reference para.
  cout << "\n x is now "<< x << " and y is now " << y;
}
```

Example:3

```
// swap function is defined here
void swap(int *a, int *b)
{
   int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Example:3

The output:

x is now 7 and y is now 5

Example:4

```
//A program to test pointers and references
#include <iostream.h>
void main ( )
{   int Var = 10;
    int  *Ptr;                  // intPtr is a pointer
    Ptr = & Var;
    cout << "\nLocation of Var: " << &Var;
    cout << "\nContents of Var: " << Var;
    cout << "\nLocation of Ptr: " << &Ptr;
    cout << "\nContents of Ptr: " << Ptr;
    cout << "\nThe value that Ptr points to: " << * Ptr;
}
```

Example:4

The output:

Location of Var: 0x66ff24

Contents of Var: 10

Location of Ptr: 0x66ff20

Contents of Ptr: 0x66ff24

The value that Ptr points to: 10

Pointers & Arrays

- Previously, we knew how to access elements stored in arrays using the name of the array and the index of the element. The following example demonstrates this

Example:1

```
#include <iostream.h>
void main ( )
{
  int array1[3]={1,2,3};
  for (int i=0; i<3; i++)
    cout<<endl<<array1[i];
}
```

Example:1

The Output:

1

2

3

Pointers & 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.

Example:2

```
int *ptr;
int arr[5];
// store the address of the first element of arr in ptr
ptr = arr;
```

The code ptr = arr; stores the address of the first element of the array in variable ptr

Pointers & Arrays

- Notice that we have used arr instead of &arr[0]. This is because both are the same. So, the above code becomes as the following:

```
int *ptr;
int arr[5];
ptr = &arr[0];
```

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

Pointers & Arrays

- Suppose we need to point to the fourth element of the array using the same pointer ptr.
- Here, if ptr points to the first element in the above example then ptr + 3 will point to the fourth element. For example,

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

Pointers & Arrays

- 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];

Pointers & Arrays

- 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];
```

Pointers & Arrays

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

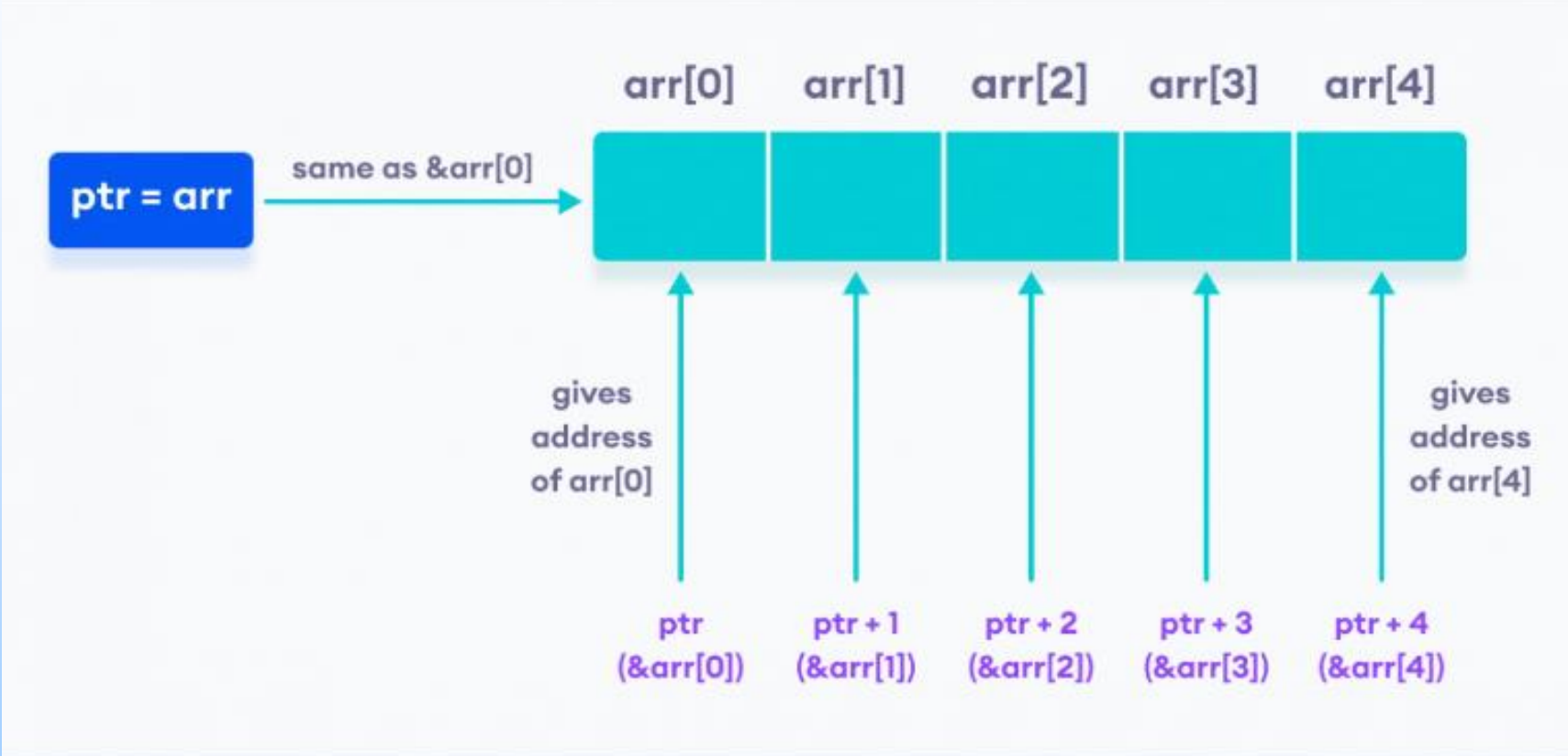
ptr - 2 is equivalent to &arr[0];

ptr - 1 is equivalent to &arr[1];

ptr + 1 is equivalent to &arr[3];

ptr + 2 is equivalent to &arr[4];

Pointers & Arrays



Working of C++ Pointers with Arrays

Pointers & Arrays

Note: The address between ptr and ptr + 1 differs by 4 bytes. It is because ptr is a pointer to an int data. And, the size of int is 4 bytes in a 64-bit operating system

Example:3

```
// C++ Program to display address of each element of an array
#include <iostream>
using namespace std;
int main()
{
    float arr[3];
    // declare pointer variable
    float *ptr;
```

Example:3

```
cout << "Displaying address using arrays: " << endl;
// use for loop to print addresses of all array elements
 for (int i = 0; i < 3; ++i)
 { cout << "&arr[" << i << "] = " << &arr[i] << endl;}
 // ptr = &arr[0]
    ptr = arr;
    cout<<"\nDisplaying address using pointers: "<< endl;
```

Example:3

```
// use for loop to print addresses of all array elements
 // using pointer notation
 for (int i = 0; i < 3; ++i)
 { cout << "ptr + " << i << " = "<< ptr + i << endl;}
 return 0;
}
```

Example:3

The output

Displaying address using arrays:

&arr[0] = 0x61fef0

&arr[1] = 0x61fef4

&arr[2] = 0x61fef8

Displaying address using pointers:

ptr + 0 = 0x61fef0

ptr + 1 = 0x61fef4

ptr + 2 = 0x61fef8

Example:4

Array elements can also be accessed using pointers.

```
#include <iostream.h>
void main ( )
{
  int arr[3]={1,2,3};
  for (int i=0; i<3; i++)
  cout<<endl<< *(arr+i);
}
```

Example:2

The Output:

1

2

3