Pointers

Pointer store address its size determined by the machine

4_byte -> 32 bit machine

8_byte -> 64 bit machine

Pointers

A pointer is an address of certain location in memory. It is a derived data type that stores the memory address. A pointer can also be used to refer another pointer, function or object. A pointer can be incremented/ decremented, i.e., to point to the next/ previous memory location.

C++ pointers are easy to learn. Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them.

As you know every variable has a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory.

What Are Pointers?

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

*type *var-name;*

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int    *ip;       // pointer to an integer
double *dp;       // pointer to a double
float  *fp;       // pointer to a float
char   *ch        // pointer to character
```

Using Pointers in C++:

There are few important operations, which we will do with the pointers very frequently. (a) we define a pointer variables

(b) assign the address of a variable to a pointer using & operator.

(c) finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations:

ch -> 'c'
chpte -> &ch

1- char ch = 'c';

*2- char *chptr;*

3- chptr = &ch; // pointer chptr has the address of the character ch

*Can be written directly as   char *chptr=&ch;*

4- char t;

*5- t = *chptr;   // t has the value that stored in the address chptr*

The above statements are equivalent to *t=ch;*

We see that in statement 5 above, we have used '*' before the name of the pointer. What does this asterisk operator do?
Well, this operator when applied to a pointer variable name (like in the last line above) yields the value of the variable to which this pointer points. Which means, in this case '*chptr' would yield the value kept at address held by chptr.
Since 'chptr' holds the address of variable 'ch' and value of 'ch' is 'c', so '*chptr' yeilds 'c'.
When used with pointers, the asterisk '*' operator is also known as 'value of' operator.

How to initialize location in memory using a Pointer?

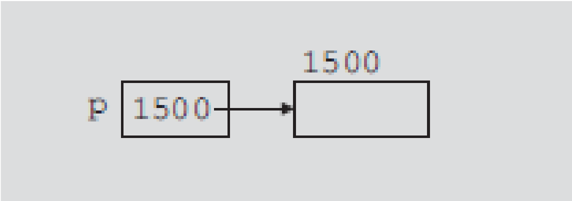
1- int *p;

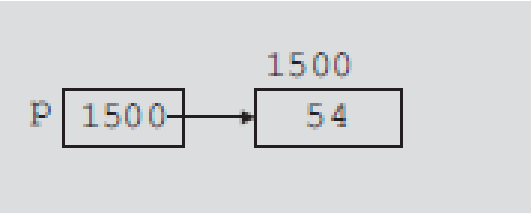
cout<<p;   //error p has no value

This statement declares p to be a pointer variable of type int. Next, consider the following statements:

2- int x; p = &x; *p=54;   the location p will have the value 54

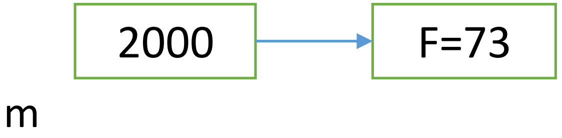
These following statements declare m as integer pointer that has the address of the integer variable f





3- int f=73;

4- int *m =&f ;



Consider the following program which will print the address of the variables defined:

```
int main ()
{    int  var1;
   char var2[10];

   cout << "Address of var1 variable: ";
   cout << &var1 << endl;

   cout << "Address of var2 variable: ";
   cout << &var2 << endl;      return 0;  }
```

<u>**The & operator is used to get the address of any variable**</u>

<u>**When the above code is compiled and executed, it produces result something as follows:**</u>

Address of var1 variable: 0x28fedc

Address of var2 variable: 0x28fed2

pointer to string

ptr -> whole string

*ptr -> first char

هااام

Examples of Pointers

Example 1:

this pointer cant modify the valur bec
its store in rom

Pointer with strings

```
{    char ch = 'c';
   char *chptr = &ch;   //initialization of pointer chptr
   int i = 20;
   int *intptr = &i;      // initialization of pointer intptr
   float f = 1.20000;
   float *fptr = &f;      // initialization of pointer fptr
   char *ptr = "I am a string";     //  initialization of pointer ptr to a string
Cout<< ch<<“    “<<i<<“    “<<f<<endl;
cout<< *chptr<<" ,"<< *intptr<<" , "<< *fptr<<" , "<<*ptr<<" ,"<< ptr;    return 0; }
```

OUTPUT :

c 20 1.200000

c, 20, 1.200000, I, I am a string

Comments about Example 1

- In the above example, we initialize char pointer chptr, integer pointer intptr and float pointer fptr

- A string is initialized using the pointer ptr.

- The program prints the values of the character ch, the integer i and the float f through their pointers

- The string is also printed using pointer. Using *ptr prints the first character in the string, while using ptr prints the whole string.

Example 2:

```
main( )
{ int y=100;  int *p;    // p is pointer to integer
        p=&y;            // p pointes to y
cout<<"   pointer p of y  "<<p<<endl<<"  value of y = "<<y<<endl;

int x=5;    p=&x;        // now p points to x
 cout<<"   new value of pointer p (address of x)   "<<p
<<endl<<"  value of x = "<<x<<endl;
 *p=12;            //assign value to x using pointer p
cout<<"   pointer p  "<<p<<endl<<" new value of x = "<<x<<endl;   }
```

Comments:

- The statement p= &y, means that p has the address of y
- The statement P= &x means that now p will points to the variable x
- Using statement *p= 12, means that the value in the address p will be 12, i.e. x will has the value 12.

<u>Output of example 2</u>

pointer p of y 0x28fed8

value of y = 100

new value of pointer p (address of x)   0x28fed4

value of x = 5

pointer p 0x28fed4

new value of x = 12

Example 3

```
int main()
{ int *p; int num1 = 5; int num2 = 8;
p = &num1; //store the address of num1 into p;
cout << "Line 3: &num1 = " << &num1<< ", p = " << p << endl;
cout << "Line 4: num1 = " << num1 << ", *p = " << *p << endl;
*p = 10;
cout << "Line 6: num1 = " << num1<< ", *p = " << *p << endl << endl;
p = &num2; //store the address of num2 into p;
cout << "Line 8: &num2 = " << &num2<< ", p = " << p << endl;
cout << "Line 9: num2 = " << num2<< ", *p = " << *p << endl;
*p = 2 * (*p);
cout << "Line 11: num2 = " << num2 << ", *p = " << *p << endl;  return 0; }
```

both are the same

both are the same

(*p)*=2

Line 3: &num1 = 0x24fe44, p = 0x24fe44
Line 4: num1 = 5, *p = 5
Line 6: num1 = 10, *p = 10

Line 8: &num2 = 0x24fe40, p = 0x24fe40
Line 9: num2 = 8, *p = 8
Line 11: num2 = 16, *p = 16

Null pointer

null pointer, meaning it does not point to any valid memory address.

We can create a null pointer by assigning null value during the pointer declaration. This method is useful when you do not have any address assigned to the pointer. A null pointer always contains value 0.

```
int *p;
  cout<<" p  "<<p<<"  any value of p   " <<endl;
  p= NULL;
  cout<<" value of pointer p  "<<p;
  cout<<*p<<endl;
```

NULL or nullptr

هاااام

NULL pointer
hold addrese 0x000000

int *ptr = NULL;
cout<< ptr <<endl; -> 0 or 0x10
cout<< *ptr <<endl; -> ERROR

bec doesnt contain vaule yet

Output

P 0x10 any value of p

value of pointer p 0

It will print random value since pointer p does not point to any variable yet or the program will be terminated.

It will give error

Operations of pointers

- C++ can determine the size of the value being pointed to

- When arithmetic is performed on a pointer, it is done using this knowledge to ensure the pointer doesn't point to intermediate memory locations

- The basic arithmetic operations (+, -, ++, --), can be used with pointers

- If an int requires 4 bytes, and iPtr is a pointer of type "int *", then the statement "iPtr++;" actually increments the pointer value by 4 bytes.

- Similarly, "iPtr2 = iPtr + 5;" stores the address "five integers " past iPtr in iPtr2

• If iPtr was 1000, then iPtr2 would contain the address 1020

• 1020 = 1000 + 5 * 4

- Pointer arithmetic is performed automatically when arithmetic is done on pointers

No special syntax is required to get this behavior, the following example shows the arithmetic on pointers

Example: 4

```
int  ival ;       // a simple integer variable
      int  *pint = &ival;  // "pint" is a pointer to int, initialized to the address of   "ival"
      cout <<" the address of ival =  "<<  " is stored at pointer  " << pint << endl;
      cout << &pint << " is the address of pointer pint itself in memory " << endl << endl;
        pint++; cout<<"  new value of pointer pint after increment "<<pint<<endl;
         pint+=4;cout<<"  new value of pointer pint after adding 4 "<<pint<<endl;
         float x;  float *p2=&x; double y; double *p3=&y;
                 // increment int and float pointer (address of pointer incremented 4 bytes)
                  cout<<" pointer p2 of float x "<<p2<<endl;
         p2++;cout<<"  new value of pointer p2 after increment "<<p2<<endl;
  // increment double pointer (address of pointer incremented 8 bytes)
                 cout<<" pointer p3 of double y "<<p3<<endl;
                  p3++;cout<<"  new value of pointer p3 after increment "<<p3<<endl;
 }
```

Output

the address of ival =   is stored at pointer 0x61fe04
0x61fdf8 is the address of pointer pint itself in memory

new value of pointer pint after increment 0x61fe08
new value of pointer pint after adding 4 0x61fe18
pointer p2 of float x 0x61fdf4
new value of pointer p2 after increment 0x61fdf8
pointer p3 of double y 0x61fde8
new value of pointer p3 after increment 0x61fdf0

Comments:

- pointer pint itself has location in memory, we can get it using &pint.
- Increment or decrement on integers and float shift address by 4 bytes, while shifting double by 8 bytes.

Constant Pointers

We can declare constant pointers to point to certain variable, so we cann’t change the address of the pointer

Exampl 5:

```
main( )  {int a= 90 ;        int b= 50 ;
   int* const ptr= &a;
   cout << "pointer of ptr "<<ptr << "\n";
   cout << " value at address ptr "<< *ptr << "\n";
   // Address what it points to
   *ptr = b;      // Acceptable to change the  value of a
   //  ptr = &b;      // Error: cannot modify a constant pointer ‘ptr’
   cout << " new value at address ptr "<<*ptr << "\n";
   cout << "new value at address ptr "<<*ptr<< "\n";
   const int x=200;
   //int *p2= &x; error, cannot convert constant int to int
  const int *p2=&x;
   cout<<" value of constant x "<<*p2<< endl;}
```

in constant pointer :

change the value of varible pointing to is   ok
change of the address pointing to   ERROR

هاام

constant varible only points to it using constant pointer

Comments

- If we declare constant pointers to point to certain variable, we cann't change the value of the pointer, from the above example
- (*int *const ptr,   int a; ptr=&a   int b; // ptr=&b; error*)

in const pointer : cant change address

- If we declare constant variables, we cann't change their values.

const int x= 200;   p2=&x; //error

*// *p2= 40; // error not valid you cann,t change the constant value*

in const var : cant change value

Output of Example 5

pointer of ptr 0x0019ff38
value at address ptr 90
new value at address ptr 50
new value at address ptr 50
value of constant x 200

In C++, the name of an array (a) is a constant pointer to its first element (&a[0]).

Dynamic Memory Allocation with Keywords new and delete

new and delete

Better dynamic memory allocation than C’s malloc and free

new - automatically creates object of proper size, calls constructor, returns pointer of the correct type.

delete - destroys object and frees space

Example:

TypeName *Ptr;

Creates pointer Ptr to a TypeName (any data type or object)

Ptr = **new** TypeName;

new creates TypeName (any data type ,int, float, struct or any object) returns pointer (which typeNamePtr is set equal to)

delete Ptr;

Calls destructor for TypeName object and frees memory

Initializing objects using new:

delete pointer mean its now dangling pointer

The pointer becomes a dangling pointer, meaning it points to a memory location that has been freed.

*int *k=new int; // get an allocation of memory for integer with pointer k*

**k=10; //put in the location k the value 10*

*cout<<k<<" "<<*k<<endl;* Output, address k=0x6613a0, while the value stored in location k= 10

delete k; // free the memory from the value stored in location k.

cout<< k; // no error   gives garbage value

*cout<<*k;* //error

cant delte pointer that points data type

*double *y= new double ; *y= 3.14159;*

Pointer of type double and initializes the value in it to 3.14159

*int *arrayPtr = new int[10];*

Creates ten integers elements in array, assign it to arrayPtr.

We use *delete [ ] arrayPtr;*   to delete arrays

delete will free the memory from the object that has been created by **new**

Step	Heap Memory Status
`new int(42);`	Memory is allocated, and `ptr` points to it.
`delete ptr;`	Memory is marked as **free**, but `ptr` still holds the old address (dangling pointer).
`ptr = nullptr;`	Now `ptr` is safe and doesn't point to invalid memory.

Examples of new and delete

Example 6

```
{int y,*p;
    //get new location for integer using new
    p= new int;      *p=16;       y=*p;
   cout<<" pointer p=  "<<p<<" value at pointer p= "<<y<<'\n';
   // delete memory location
     delete p;
     cout<<"  pointer after delete  "<<p<<endl;
     // the value that have been stored in p is removed
     cout<<" value after delete "<<*p<<'\n';  //error
     p=&y; cout<<" new address of pointer p   "<<p<<endl;
   //  delete y;  // error cann't delete variable that was not created by new
    }
```

Solution

pointer p= 0x7117b0 value at pointer p= 16
 pointer after delete 0x7117b0
value after delete 7411744
new address of pointer p   0x61fe14

Example 7

This program illustrates how to allocate dynamic memory

// using a pointer variable and how to manipulate data into

// that memory location.

//**

```
int *p;  int *q;   //line 1
p = new int;  *p = 34;   //line 2
cout << "Line 3: p = " << p << ", *p = " << *p << endl;
  q = p; // line 3
cout << "Line 4: q = " << q << ", *q = " << *q << endl;
  *p = 18;
cout << "Line 5 : p = " << p << ", *p = " << *p << endl;  // line 5
```

Example 7 (cont.)

```
cout << "Line 6: q = " << q  //line 6
<< ", *q = " << *q << endl;  //line 7
delete q;  // line 8    dangling poiter
cout<<" Line 9 pointer q after delete  =" <<q<<endl;
q = NULL; //line 10    avoid dnagling pointer
cout<< " Line 10 as null pointer =" <<q<<endl;
 q = new int; // line 11
*q = 62;
cout << "Line 11: q = " << q
<< ", *q = " << *q << endl; return 0; }
```

Output

Line 3: p = 0x0218332c, *p = 34

Line 4: q = 0x0218332c, *q = 34

Line 5 : p = 0x0218332c, *p = 18

Line 6: q = 0x0218332c, *q = 18

Line 9 pointer q after delete =0x0218332c

Line 10 as null pointer =0x00000000

Line 11: q = 0x0218332c, *q = 62

Note that:

1- address q is not changed when using delete dnagling pointer

هاام 2- If we remove line 11, we will have an error as trying to put a value in NULL pointer, we must create it again.

C++ Pointers in Detail

Pointers have many but easy concepts and they are very important to C++ programming. There are following few important pointer concepts which should be clear to a C++ programmer:

Concept	Description
C++ Null Pointers	C++ supports null pointer, which is a constant with a value of zero defined in several standard libraries.
C++ pointer arithmetic	There are four arithmetic operators that can be used on pointers: ++, --, +, -
C++ pointers vs arrays	There is a close relationship between pointers and arrays. Let us check how?
C++ array of pointers	You can define arrays to hold a number of pointers.
C++ pointer to pointer	C++ allows you to have pointer on a pointer and so on.
Passing pointers to functions	Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.
Return pointer from functions	C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.

We will describe pointers vs arrays, pointers and function and array of pointers in the second part of pointer lectures