C++ Pointers

The pointer in C++ language is a variable, also known as a locator or indicator that points to an address of a value.

The symbol of an address is represented by a pointer. In addition to creating and modifying dynamic data structures, they allow programs to emulate call-by-reference. One of the principal applications of pointers is iterating through the components of arrays or other data structures. The pointer variable that refers to the same data type as the variable you're dealing with has the address of that variable set to it (such as an int or string).

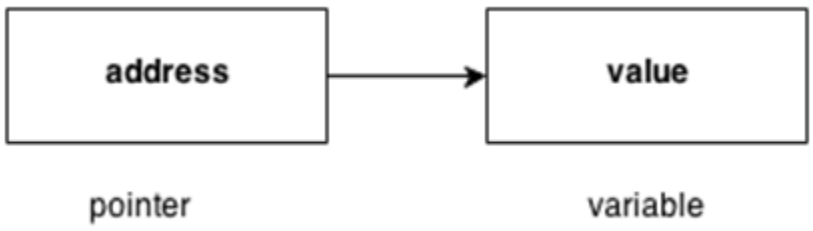
Syntax

1. datatype *var_name;
2. **int** *ptr; // ptr can point to an address which holds int data

How to use a pointer?

1. Establish a pointer variable.
2. employing the unary operator (&), which yields the address of the variable, to assign a pointer to a variable's address.
3. Using the unary operator (*), which gives the variable's value at the address provided by its argument, one can access the value
4. stored in an address.

Since the data type knows how many bytes the information is held in, we associate it with a reference. The size of the data type to which a pointer points is added when we increment a pointer.



Advantage of pointer

1) Pointer reduces the code and improves the performance, it is used to retrieve strings, trees etc. and used with arrays, structures and functions.2) We can return multiple values from a function using a pointer.

3) It allows you to access any memory location in the computer's memory.

Usage of pointer

There are many usage of pointers in C++ language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

2) Arrays, Functions and Structures

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the address of a variable.
* (asterisk sign)	Indirection operator	Access the value of an address.

Declaring a pointer

The pointer in C++ language can be declared using * (asterisk symbol).

```
1. int *  a; //pointer to int
2. char *  c; //pointer to char
```

Pointer Example

Let's see the simple example of using pointers to print the address and value.

```
#include <iostream>
using namespace std;
int main()
{
int number=30;
int *  p;
p=&number;//stores the address of number variable
cout<<"Address of number variable is:"<<&number<<endl;
cout<<"Address of p variable is:"<<p<<endl;
cout<<"Value of p variable is:"<<*p<<endl;
  return 0;
}
```

Output:

```
Address of number variable is:0x7ffccc8724c4
Address of p variable is:0x7ffccc8724c4
Value of p variable is:30
```

Pointer Program to swap 2 numbers without using 3rd variable

```
#include <iostream>
using namespace std;
int main()
{
int a=20,b=10,*p1=&a,*p2=&b;
cout<<"Before swap: *p1="<<*p1<<" *p2="<<*p2<<endl;
*p1=*p1+*p2;
*p2=*p1-*p2;
*p1=*p1-*p2;
cout<<"After swap: *p1="<<*p1<<" *p2="<<*p2<<endl;
  return 0;
}
```

Output:

Before swap: *p1=20 *p2=10
After swap: *p1=10 *p2=20

What are Pointer and string literals?

String literals are arrays of character sequences with null ends. The elements of a string literal are arrays of type const char (because characters in a string cannot be modified) plus a terminating null-character.

What is an invalid pointer?

A pointer must point to a valid address, not necessarily to useful items (like for arrays). We refer to these as incorrect pointers. Additionally, incorrect pointers are uninitialized pointers.

```
1. int *ptr1;
2. int arr[10];
3. int *ptr2 = arr+20;
```

Here, ptr1 is not initialised, making it invalid, and ptr2 is outside of the bounds of arr, making it likewise weak. (Take note that not all build failures are caused by faulty references.)

What is a null pointer?

A null pointer is not merely an incorrect address; it also points nowhere. Here are two ways to mark a pointer as NULL:

```
1. int *ptr1 = 0;
2. int *ptr2 = NULL;
```

C++ Array of Pointers

Array and pointers are closely related to each other. In C++, the name of an array is considered às a pointer, i.e., the name of an array contains the address of an element. C++ considers the array name as the address of the first element. For example, if we

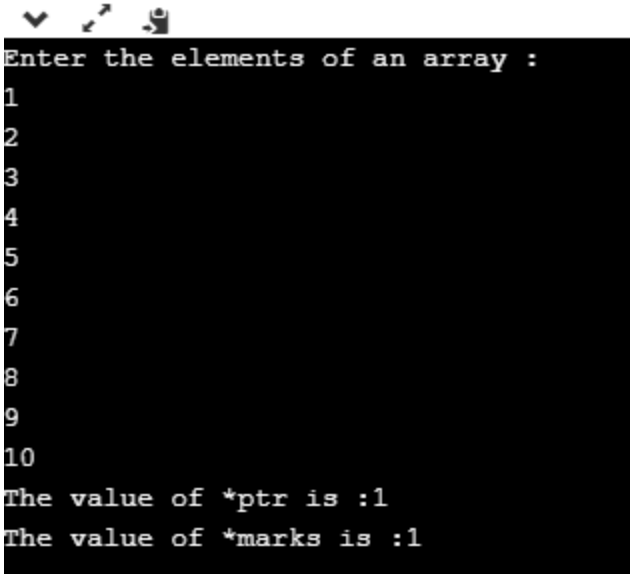
create an array, i.e., marks which hold the 20 values of integer type, then marks will contain the address of first element, i.e., marks[0]. Therefore, we can say that array name (marks) is a pointer which is holding the address of the first element of an array.

Let's understand this scenario through an example.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.     int *ptr;  // integer pointer declaration
6.     int marks[10]; // marks array declaration
7.     std::cout << "Enter the elements of an array :" << std::endl;
8.     for(int i=0;i<10;i++)
9.     {
10.       cin>>marks[i];
11.    }
12.    ptr=marks; // both marks and ptr pointing to the same element..
13.    std::cout << "The value of *ptr is :" <<*ptr<< std::endl;
14.    std::cout << "The value of *marks is :" <<*marks<<std::endl;
15. }
```

In the above code, we declare an integer pointer and an array of integer type. We assign the address of marks to the ptr by using the statement ptr=marks; it means that both the variables 'marks' and 'ptr' point to the same element, i.e., marks[0]. When we try to print the values of *ptr and *marks, then it comes out to be same. Hence, it is proved that the array name stores the address of the first element of an array.

Output



Array of Pointers

An array of pointers is an array that consists of variables of pointer type, which means that the variable is a pointer addressing to some other element. Suppose we create an array of pointer holding 5 integer pointers; then its declaration would look like:

```
1. int *ptr[5];       // array of 5 integer pointer.
```

In the above declaration, we declare an array of pointer named as ptr, and it allocates 5 integer pointers in memory.

The element of an array of a pointer can also be initialized by assigning the address of some other element. Let's observe this case through an example.

```
1. int a; // variable declaration.
2. ptr[2] = &a;
```

In the above code, we are assigning the address of 'a' variable to the third element of an array 'ptr'.

We can also retrieve the value of 'a' be dereferencing the pointer.

```
1. *ptr[2];
```

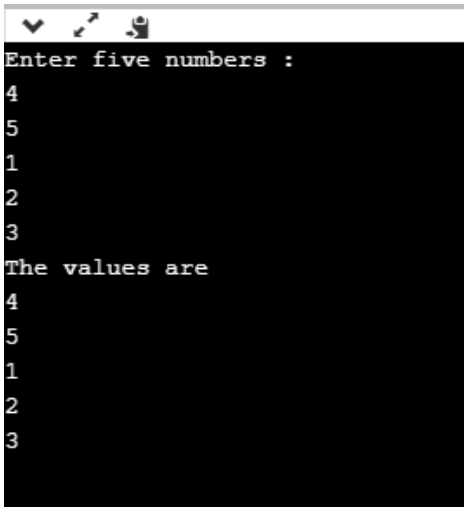
Let's understand through an example.

```
1. #include <iostream>
```

```
2.  using namespace std;
3.  int main()
4.  {
5.     int ptr1[5]; // integer array declaration
6.     int *ptr2[5]; // integer array of pointer declaration
7.     std::cout << "Enter five numbers :" << std::endl;
8.     for(int i=0;i<5;i++)
9.     {
10.       std::cin >> ptr1[i];
11.    }
12.    for(int i=0;i<5;i++)
13.    {
14.       ptr2[i]=&ptr1[i];
15.    }
16.    // printing the values of ptr1 array
17.    std::cout << "The values are" << std::endl;
18.    for(int i=0;i<5;i++)
19.    {
20.       std::cout << *ptr2[i] << std::endl;
21.    }
22.    }
```

In the above code, we declare an array of integer type and an array of integer pointers. We have defined the 'for' loop, which iterates through the elements of an array 'ptr1', and on each iteration, the address of element of ptr1 at index 'i' gets stored in the ptr2 at index 'i'.

Output



Till now, we have learnt the array of pointers to an integer. Now, we will see how to create the array of pointers to strings.

Array of Pointer to Strings

An array of pointers to strings is an array of character pointers that holds the address of the first character of a string or we can say the base address of a string.

The following are the differences between an array of pointers to string and a two-dimensional array of characters:An array of pointers to string is more efficient than the two-dimensional array of characters in case of memory consumption because an array of pointer to strings consumes less memory than the two-dimensional array of characters to store the strings.

- In an array of pointers, the manipulation of strings is comparatively easier than in the case of 2d array. We can also easily change the position of the strings by using the pointers.

Let's see how to declare the array of pointers to string.

First, we declare the array of pointers to the string:

```
char *names[5] = {"john",
                  "Peter",
                  "Marco",
                  "Devin",
                  "Ronan"};
```

In the above code, we declared an array of pointer names as 'names' of size 5. In the above case, we have done the initialization at the time of declaration, so we do not need to mention the size of the array of a pointer. The above code can be re-written as:

```
char *names[ ] = {"john",
                  "Peter",
                  "Marco",
                  "Devin",
                  "Ronan"};
```

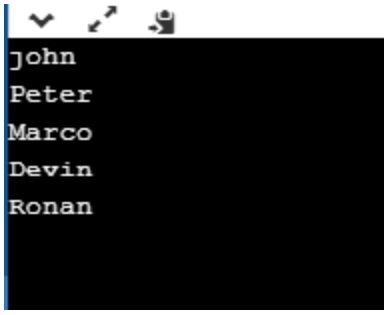
In the above case, each element of the 'names' array is a string literal, and each string literal would hold the base address of the first character of a string. For example, names[0] contain the base address of "John", names[1] contain the base address of "Peter", and so on. It is not guaranteed that all the string literals will be stored in the contiguous memory location, but the characters of a string literal are stored in a contiguous memory location.

Let's create a simple example.

```
#include <iostream>
using namespace std;
int main()
{
  char *names[5] = {"john",
                    "Peter",
                    "Marco",
                    "Devin",
                    "Ronan"};
for(int i=0;i<5;i++)
   {
      std::cout << names[i] << std::endl;
   }
   return 0;
}
```

In the above code, we have declared an array of char pointer holding 5 string literals, and the first character of each string is holding the base address of the string.

Output


john
Peter
Marco
Devin
Ronan

Dynamic memory allocation in C++

There are times where the data to be entered is allocated at the time of execution. For example, a list of employees increases as the new employees are hired in the organization and similarly reduces when a person leaves the organization. This is called managing the memory. So now, let us discuss the concept of dynamic memory allocation.

Memory allocation

Reserving or providing space to a variable is called memory allocation. For storing the data, memory allocation can be done in two ways -

- **Static allocation or compile-time allocation - Static memory allocation means providing space for the variable. The size and data type of the variable is known, and it remains constant throughout the program.**
- **Dynamic allocation or run-time allocation - The allocation in which memory is allocated dynamically. In this type of allocation, the exact size of the variable is not known in advance. Pointers play a major role in dynamic memory allocation.**

Why Dynamic Memory Allocation? Dynamically we can allocate storage while the program is in a running state, but variables cannot be created "on the fly". Thus, there are two criteria for dynamic memory allocation -

- A dynamic space in the memory is needed.
- Storing the address to access the variable from the memory

Similarly, we do memory de-allocation for the variables in the memory.

In C++, memory is divided into two parts -

- Stack - All the variables that are declared inside any function take memory from the stack.
- Heap - It is unused memory in the program that is generally used for dynamic memory allocation.

Dynamic memory allocation using the new operator

To allocate the space dynamically, the operator new is used. It means creating a request for memory allocation on the free store. If memory is available, memory is initialized, and the address of that space is returned to a pointer variable.

Syntax

Pointer_variable = new data_type;

The pointer_varible is of pointer data_type. The data type can be int, float, string, char, etc.

Example

int *m = NULL // Initially we have a NULL pointer

m = new int // memory is requested to the variable

It can be directly declared by putting the following statement in a line -

int *m = new int

Initialize memory

We can also initialize memory using new operator.

For example

int *m = new int(20);

Float *d = new float(21.01);

Allocate a block of memory

We can also use a new operator to allocate a block(array) of a particular data type.

For example

int *arr = new int[10]

Here we have dynamically allocated memory for ten integers which also returns a pointer to the first element of the array. Hence, arr[0] is the first element and so on.

Note

- The difference between creating a normal array and allocating a block using new normal arrays is deallocated by the compiler. Whereas the block is created dynamically until the programmer deletes it or if the program terminates.
- If there is no space in the heap memory, the new request results in a failure throwing an exception(std::bad_alloc) until we use *nonthrow* with the new operator. Thus, the best practice is to first check for the pointer variable.

Code

```
1. int *m = new(nonthrow) int;
2. if(!m) // check if memory is available
3. {
4.     cout<< "No memory allocated";
```

```
}
```

Now as we have allocated the memory dynamically. Let us learn how to delete it.

Delete operator

We delete the allocated space in C++ using the delete operator.

Syntax

delete pointer_variable_name

Example

delete m; // free m that is a variable

delete [] arr; // Release a block of memory

Example to demonstrate dynamic memory allocation

```
// The program will show the use of new and delete
#include <iostream>
using namespace std;
int main ()
{
    // Pointer initialization to null
    int* m = NULL;
    // Request memory for the variable
    // using new operator
    m = new(nothrow) int;
    if (!m)
        cout<< "allocation of memory failed\n";
    else
    {
```

```
    // Store value at allocated address
    *m=29;
    cout<< "Value of m: " << *m <<endl;
}
// Request block of memory
// using new operator
float *f = new float(75.25);
cout<< "Value of f: " << *f <<endl;
// Request block of memory of size
int size = 5;
int *arr = new(nothrow) int[size];
if (!arr)
    cout<< "allocation of memory failed\n";
else
{
    for (int i = 0; i< size; i++)
        arr[i] = i+1;

    cout<< "Value store in block of memory: ";
    for (int i = 0; i< size; i++)
        cout<<arr[i] << " ";
}

// freed the allocated memory
delete m;
delete f;
// freed the block of allocated memory
delete[] arr;

return 0;
}
```

Output

```
Value of m: 29
Value of f: 75.25
Value store in block of memory: 1 2 3 4 5
```

Pointer to Pointer (Multiple Indirection)

A pointer to a pointer is a form of multiple indirection or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int –

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example –

```
#include <iostream>

using namespace std;

int main () {
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   // take the address of var
   ptr = &var;
```

```
   // take the address of ptr using address of operator &
   pptr = &ptr;

   // take the value using pptr
   cout << "Value of var :" << var << endl;
   cout << "Value available at *ptr :" << *ptr << endl;
   cout << "Value available at **pptr :" << **pptr << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Value of var :3000
Value available at *ptr :3000
Value available at **pptr :3000
```