

C++ Associate Programing

References in C++



Lesson Objectives

- *Applications of References*
- *References vs Pointers*



Section 1

APPLICATIONS OF REFERENCES

Applications of References (1)

```
▪ #include<iostream>
▪ using namespace std;
▪
▪ void swap (int& first, int& second)
▪ {
▪     int temp = first;
▪     first = second;
▪     second = temp;
▪ }
▪
▪ int main()
▪ {
▪     int a = 2, b = 3;
▪     swap( a, b );
▪     cout << a << " " << b;
▪     return 0;
▪ }
```

Modify the passed parameters in a function : If a function receives a **reference** to a variable, it can modify the value of the variable. For example, in the following program variables are swapped using **references**.

Output:

3 2



Applications of References (2)

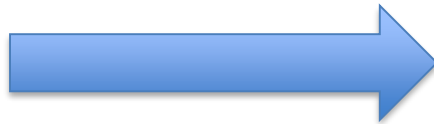
```
struct Student {  
    string name;  
    string address;  
    int rollNo;  
}  
/* If we remove & in below function, a new  
copy of the student object is created.  
We use const to avoid accidental updates  
in the function as the purpose of the function  
is to print s only. */  
void print(const Student &s)  
{  
    cout << s.name << " " << s.address << " " << s.rollNo;  
}
```

Avoiding copy of large structures : Imagine a function that has to receive a large object. If we pass it without **reference**, a new copy of it is created which causes wastage of CPU time and memory. We can use **references** to avoid this.

Applications of References (3)

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int> vect{ 10, 20, 30, 40 };
    // We can modify elements if we
    // use reference
    for (int &x : vect)
        x = x + 5;
    // Printing elements
    for (int x : vect)
        cout << x << " ";
    return 0;
}
```

In For Each Loops to modify all objects : We can use **references** in for each loops to modify all elements




OUTPUT:
15 25 35 45

Applications of References (4)

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<string> vect{"geeksforgeeks
practice",
                      "geeksforgeeks write",
                      "geeksforgeeks ide"};
    // We avoid copy of the whole string
    // object by using reference
    for (const auto &x : vect)
        cout << x << endl;
    return 0;
}
```

In For Each Loops to avoid copy of objects: We can use **references** in for each loops to avoid copy of individual objects when objects are large.



```
OUTPUT:
geeksforgeeks practice
geeksforgeeks write
geeksforgeeks ide
```

Section 2

REFERENCES VS POINTERS

- *Pointers: A pointer is a variable that holds memory address of another variable. A pointer needs to be dereferenced with * operator to access the memory location it points to.*
- *References : A reference variable is an alias, that is, another name for an already existing variable. A reference, like a pointer, is also implemented by storing the address of an object.*

References vs Pointers: Differences (1)

Initialization: A pointer can be initialized in this way:

```
int a = 10;  
int *p = &a;  
    OR  
int *p;  
p = &a;
```

we can declare and initialize pointer at same step or in multiple line.

While in references,

```
int a=10;  
int &p=a; //it is correct  
    but  
int &p;  
p=a;    // it is incorrect as we should declare and initialize references  
at single step.
```

References vs Pointers: Differences (2)

Reassignment: A pointer can be re-assigned. This property is useful for implementation of data structures like linked list, tree, etc. See the following examples:

```
int a = 5;
int b = 6;
int *p;
p = &a;
p = &b;
```

On the other hand, a reference cannot be re-assigned, and must be assigned at initialization.

```
int a = 5;
int b = 6;
int &p = a;
int &p = b; //At this line it will show error as "multiple declaration is
not allowed".
```

However it is valid statement,
`int &q=p;`

References vs Pointers: Differences (3)

Memory Address: A **pointer** has its own memory address and size on the stack whereas a **reference** shares the same memory address (with the original variable) but also takes up some space on the stack

```
int &p = a;  
cout << &(&p);
```

NULL value: **Pointer** can be assigned NULL directly, whereas **reference** cannot. The constraints associated with **references** (no NULL, no reassignment) ensure that the underlying operations do not run into exception situation.

References vs Pointers: Differences (4)

Indirection: You can have pointers to pointers offering extra levels of indirection. Whereas references only offer one level of indirection. I.e,

```
In Pointers,  
int a = 10;  
int *p;  
int **q; //it is valid.  
p = &a;  
q = &p;
```

Whereas in references,

```
int &p = a;  
int &&q = p; //it is reference to reference, so it is an error.
```

Arithmetic operations: Various arithmetic operations can be performed on pointers whereas there is no such thing called Reference Arithmetic. (but you can take the address of an object pointed by a reference and do pointer arithmetics on it as in `&obj + 5`.)

References vs Pointers: When to use

The performances are exactly the same, as references are implemented internally as pointers.

Use references:

In function parameters and return types.

Use pointers:

Use pointers if pointer arithmetic or passing NULL-pointer is needed. For example for arrays (Note that array access is implemented using pointer arithmetic).

To implement data structures like linked list, tree, etc and their algorithms because to point different cell, we have to use the concept of pointers.

References vs Pointers: Example

In Pointer

```
int arr[] = {10,20,30};
int *p,i ;
p = arr;
for(i = 0; i<3 ; i++)
{
    cout << *p << endl;
    p++;
}
```

Output:

```
10
20
30
```

In Reference

```
int arr[] = {10,20,30};
int &p = arr[0];
int i;
for(i = 0; i<3 ; i++)
{
    cout << p << endl;
    p++; //here it will increment arr[0]++
}
```

Output:

```
10
11
12
```

Thank you

Q&A

