### References in C++

References are like constant pointers that are automatically dereferenced. It is a new name given to an existing storage. So when you are accessing the reference, you are actually accessing that storage.

int main()

{ int y=10;

int &r = y; // r is a reference to int y

cout << r;

}

Output : 10

There is no need to use the \* to dereference a reference variable.

#### Difference between Reference and Pointer

| **References** | **Pointers** |
| --- | --- |
| Reference must be initialized when it is created. | Pointers can be initialized any time. |
| Once initialized, we cannot reinitialize a reference. | Pointers can be reinitialized any number of time. |
| You can never have a NULL reference. | Pointers can be NULL. |
| Reference is automatically dereferenced. | \* is used to dereference a pointer. |

### References in Funtions

References are generally used for function argument lists and function return values, just like pointers.

#### Rules for using Reference in Functions

1. When we use reference in argument list, we must keep in mind that any change to the reference inside the function will cause change to the original argument outside th function.
2. When we return a reference from a function, you should see that whatever the reference is connected to shouldn't go out of scope when fnction ends. Either make that **global** or **static**

#### Example to explain use of References

int\* first (int\* x)

{ (\*x++);

return x; // SAFE, x is outside this scope

}

int& second (int& x)

{ x++;

return x; // SAFE, x is outside this scope

}

int& third ()

{ int q;

return q; // ERROR, scope of q ends here

}

int& fourth ()

{ static int x;

return x; // SAFE, x is static, hence lives till the end.

}

int main()

{

int a=0;

first(&a); // UGLY and explicit

second(a); // CLEAN and hidden

}

We have four different functions in the above program.

* **first()** takes a pointer as argument and returns a pointer, it will work fine. The returning pointer points to variable declared outside first(), hence it will be valid even after the first() ends.
* Similarly, **second()** will also work fine. The returning reference is connected to valid storage, that isint a in this case.
* But in case of **third()**, we declare a variable q inside the function and try to return a reference connected to it. But as soon as function third() ends, the local variable q is destroyed, hence nothing is returned.
* To remedify above problem, we make x as **static** in function **fourth()**, giving it a lifetime till main() ends, hence now a reference connected to x will be valid when returned.

#### Const Reference

Const reference is used in function arguments to prevent the function from changing the argument.

void g(const int& x)

{ x++; } // ERROR

int main()

{

int i=10;

g(i);

}

We cannot change the argument in the function because it is passed as const reference.

#### Argument Passing Guidelines

Usually **call by value** is used during funtcion call just to save our object or variable from being changed or modified, but whenever we pass an argument by value, its new copy is created. If we pass an object as argument then a copy of that object is created (constructor and destructors called), which affects efficiency.

Hence, we must use **const reference** type arguments. When we use, const reference, only an address is passed on stack, which is used inside the function and the function cannot change our argument because it is of const type.

So using const reference type argument reduces overhead and also saves our argument from being changed.