Pointer Vs Ref Var

① `int a=10;`
`int *p = &a;`

OR

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

① `int a=10;`

✓ `int &p=a;`

`int a=10;`

X `int &p;`
`p=a;`

Pointer

② `int a=10, b=20;`

`int *p = &a;`

`cout << a << ", " << *p;`
10 10

`p=&b;`

`cout << a << ", " << *p << ", " << b;`
10 20 20

Ref Var

② `int a=10, b=20;`

`int &p=a;`

`cout << a << ", " << p << ", " << b;`
10 10 20

`p=b;`

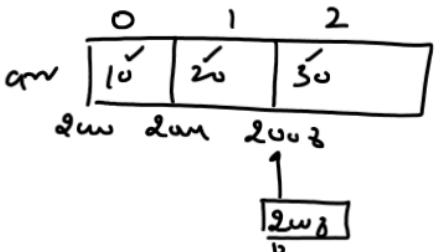
`cout << a << ", " << p << ", " << b;`
20 20 20

`p=30;`

`cout << a << ", " << p << ", " << b;`
30 30 20

(8)

```
int arr[ ] = {10, 20, 30};
int *p, i;
```

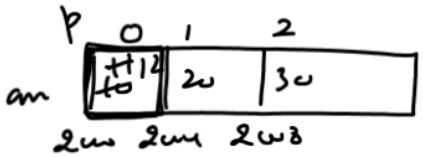



```
p = arr;
for(i=0; i<3; i++)
{
    cout<<*p<<endl;
    p++;
}
```

10
20
30

(8)

```
int arr[ ] = {10, 20, 30};
```



X `int &p = arr;`

`int &p = arr[0];`

```
int i;
for(i=0; i<3; i++)
{
    cout<<p<<endl;
    p++;
}
```

10
11
12

ARGUMENT PASSING IN C++

1. Pass By Value
2. Pass By Address (also called by Pass by ref using pointer)
3. Pass By Reference (also called as Pass by ref using ref var)

Pass by value
==========
1. This style of argument passing in C++ is **exactly same** as given by C programming language.

2. In this style of argument passing, the programmer passes the name of a variable as argument to the function while calling it and the compiler passes the variable's value

3. This value is then received by the variable declared as formal argument in the function's parameter list

4. So now both, actual argument and formal argument have the same value but their addresses are different

5. Due to this although we can access the value of actual argument through formal argument but if we make any change in the formal argument's value it will **never change the value of actual argument** .

6. To solve this problem C++ provides two more ways of argument passing and they are:

Pass by address and
Pass by reference

Pass by address
==========
1. This style of argument passing in C++ is same as pass by reference given by C language

2. In this style of argument passing we pass the address of a variable as actual argument while calling the function and the compiler copies it in a pointer declared as formal argument in the function's parameter list

3. Now since, we have a pointer to the actual argument so we can not only access the value of actual argument but we also can change it

4. But to do this we will have to dereference the pointer using INDIRECTION OPERATOR which is * .

5. Due to this our code becomes a bit compilcated in syntax.

6. As a solution to this C++ recommends the use of reference variable and the technique becomes pass by reference

Pass by reference using reference variable
==========
1. This is new style of argument passing introduced by C++ and not available in C lang

2. In this style of argument passing we still pass the name of the variable as ACTUAL ARGUMENT while calling the function just like we pass the variable name in pass by value

3. But we receive the actual argument in a reference variable declared as formal argument in the function's parameter list

4. Now since, we have a reference variable to the actual argument we can access as well as change the actual arguments value

5. The benefit of this approach is that since we are using reference variable, we don't have to use INDIRECTION OPERATOR(*) and due to this the code becomes simple to understand