Special Member function

- Default constructor.
- default destructor.
- Copy Constructor.
- Assignment operator overloading.
- Address operator

Shallow Copy

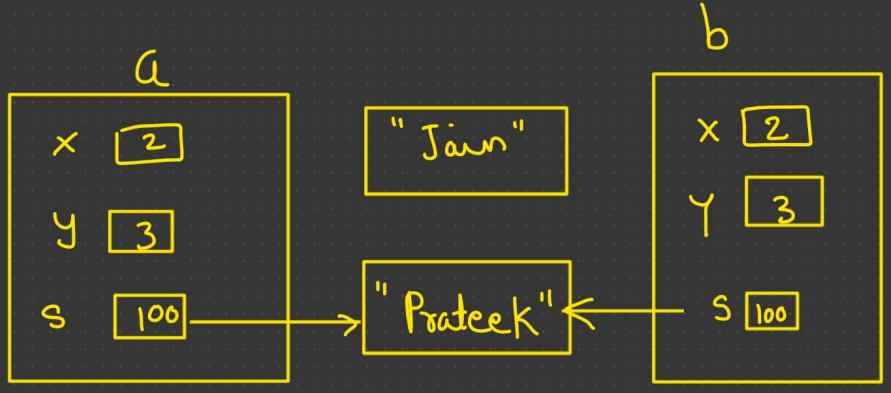
→ Pass by value

```
ABC a(2, 3, "Prateek");
ABC b = a;
```

→ Copy constructor

int x = y;

b.x = a.x
b.y = a.y
b.s = a.s



a.s = "Jain"

```
int len = strlen(s1);
char*s = new char[len+1];
strcpy(s, s1);
```

→ default constructor

How to Create our own Copy Constructor

```
ABC( ABC P )
{
    X = P.X;
    Y = P.Y;
}
```

Wrong

```
ABC b = a;
```

b → cc, a → as a parameter

ABC P = a;

ABC P = a;

Problem: infinite loop

```
ABC(Const ABC &P)
{
    X = P.X;
    Y = P.Y;
}
```

ABC &P = a;

```
ABC b = a;
```

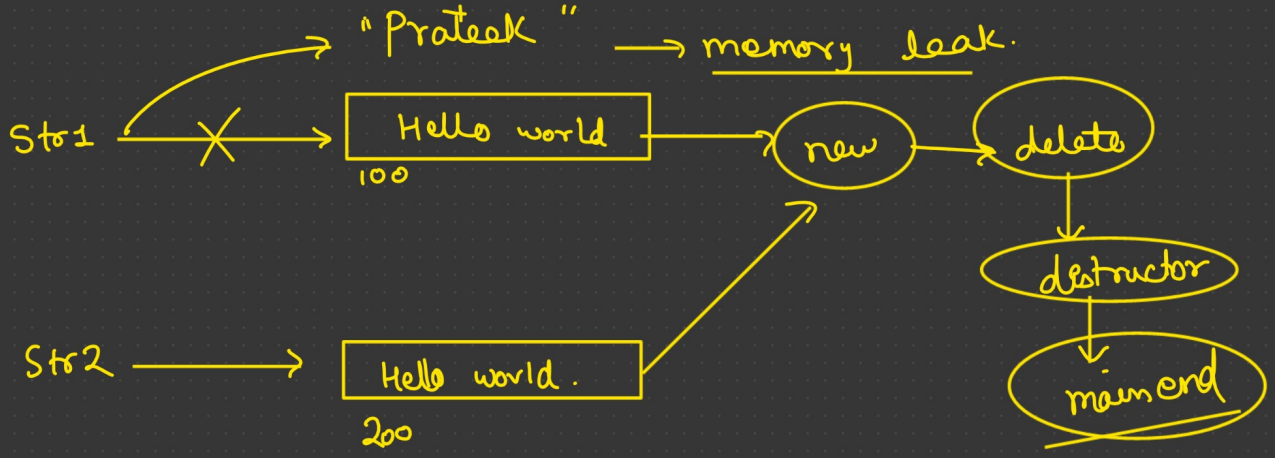
Why const ? →

① so that object may not accidently modify the content of your object.

② Compiler may create a temporary object can't be bound to non-const reference & the original program tries to do that. It doesn't make sense to modify compiler created temporary object as they can die any moment.

Copy Constructor Vs Assignment Operator

```
ABC t1, t2;   ⟶ // Default constructor
ABC t3 = t1;  ⟶ // Copy Constructor
t2 = t3;      ⟶ // Assignment operator
```



Str1.str = "prateek";

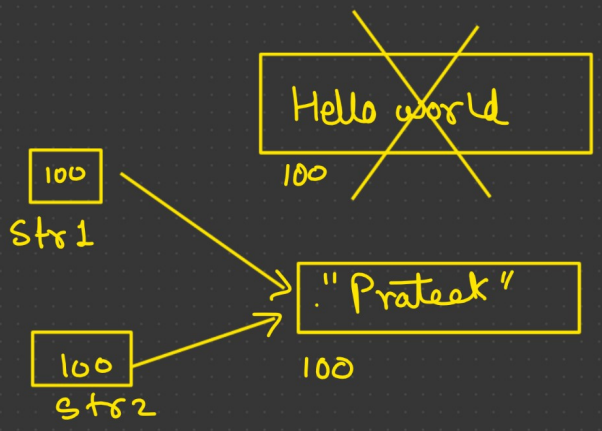
Str1.change("Prateek");

```
void change(const char *s)
{
    delete [] str;
    size = strlen(s);
    str = new char[size+1];
    strcpy(str, s);
}
```

⟶ modify old string

Shallow Copy Vs Deep Copy

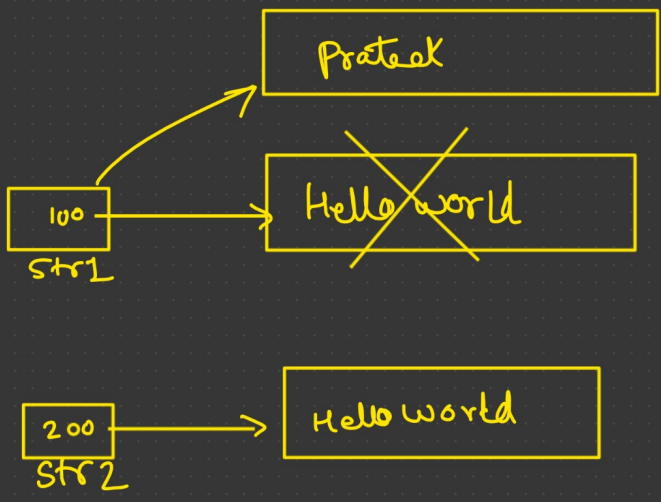
```
myString Str1 ("Hello world");
myString Str2 = str1;
```



↓

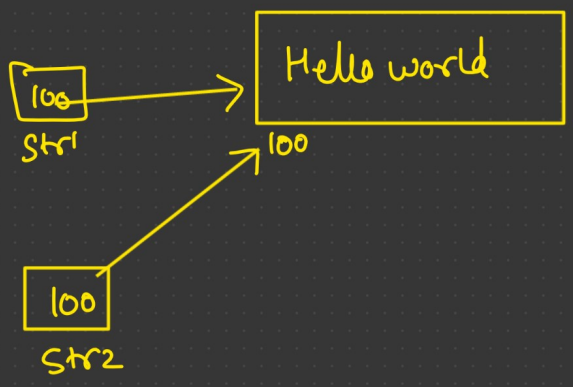
Default Copy Constructor called which is doing shallow copy.

```
str1.change("Prateek");
```



Deep copy →

```
myString Str1 ("Hello world");
myString Str2;
Str2 = str1;
```



↓

Default Assignment operator called. which will do shallow copy.

Fixing Assignment operator :-

1) delete the existing old memory. Because target object (LHS) may already refer to previously allocated data.

2) The function should protect against assignment an object to itself. [Ex :- s = s;]. Otherwise, the freeing of memory described previously could erase the object's content before they are assigned.

3) The function return a reference of the invoking object. So that it support X = Y = Z.

```
mystring & operator = ( const myString &s)
{
    if ( this == &s)
        return *this;

    delete [] str;
    size = strlen(s.str);
    str = new char[size+1];
    strcpy( str , s.str);

    return *this;
}
```

```
class myString
{
  char *str;
  int size;
Public:
  // CC
  // DE ...
};
```

Conclusion:—

we must overload copy constructor & Assigmant operator. whenever we are allocating memory using "new" in our default constructor.

"Or" we want "deep copy" instead of "shallow copy".