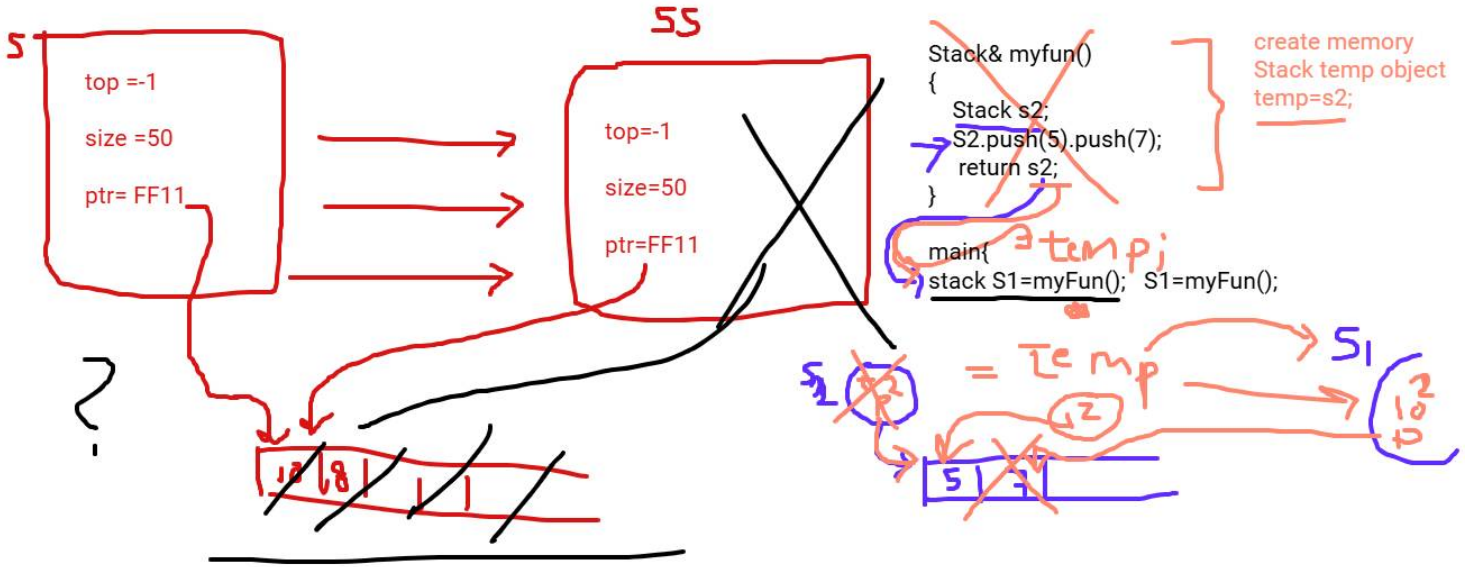
1-Copy Constructor-->(External resource used inside class)
(create new object WRT another,function pass object by Value,function return by Value)
2-Move Constructor

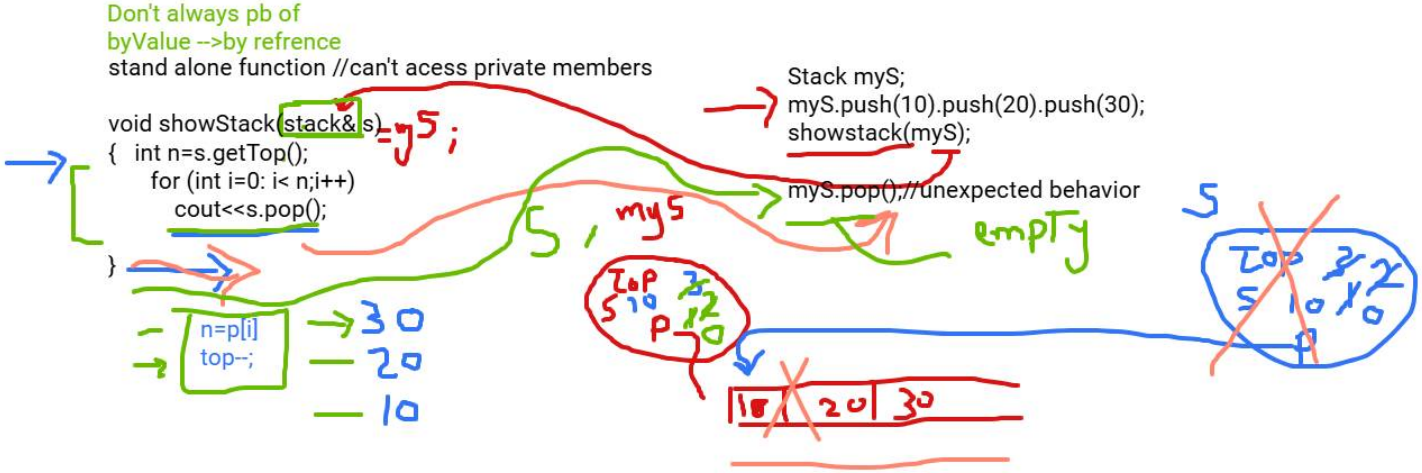
3-operator overloading

4-new features (default,delete)

5-Friend Function

```
Stack s(50);  //call constructors
{Stack ss=s;  // copy ctr.(pb)->implement copy ctr (deep copying)
ss.push(10).push(8).pop();}//call dest. of ss delete memory
//Shallow Copy:(bitwise Copying)
s.show();//10
stack ss1=s;//delete feature -->compile error
complex c1=c;//copy ct. default (no pb)
Deep Copying.
```





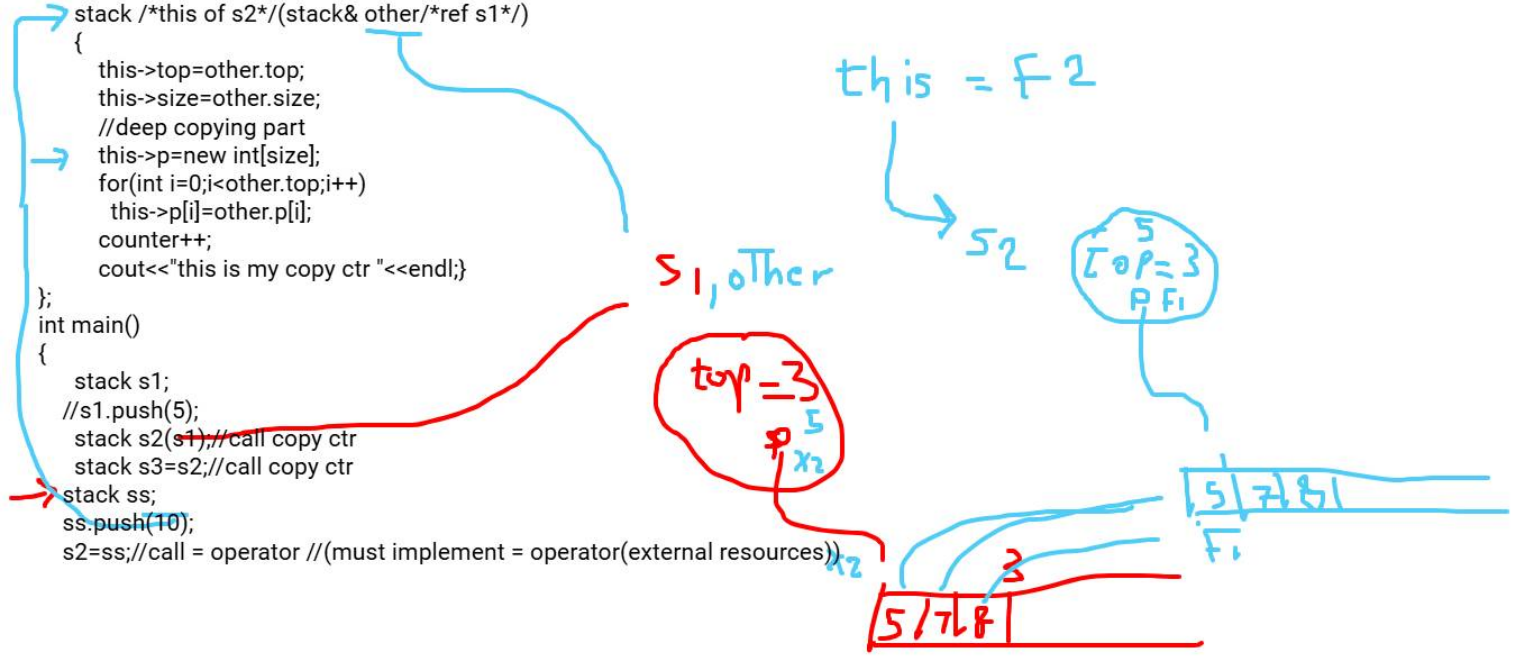
```
Class Stack
{
    int* ptr;
    int top;
    int size;
    inline static int counter=0;
    public:
     stack(int size);
     stack()=delete;
     friend void
ShowStack(stack&);

};
```

Stack s();//Error

```
void showstack(const stack & s)//read only
{
   int n =s.top;//access private member
   for(int i=0;i<n;i++)
     cout<<s.p[i]<<endl;
}
```

```
class stack
{
   int size{5};//int size=5.5;(implicit conversion)
   int top{0};
   int * p;
   inline static int counter{0};
public:
    stack(int size=5):size(size),top(0),p(new int[size]){counter++;}//top=0;this-
>size=size;p=new int[size];}
    stack /*this of s2*/(stack& other/*ref s1*/)
    {
       this->top=other.top;
       this->size=other.size;
       //deep copying part
       this->p=new int[size];
       for(int i=0;i<other.top;i++)
         this->p[i]=other.p[i];
       counter++;
       cout<<"this is my copy ctr "<<endl;}
};
int main()
{
    stack s1;
   //s1.push(5);
    stack s2(s1);//call copy ctr
    stack s3=s2;//call copy ctr
    stack ss;
   ss.push(10);
   s2=ss;//call = operator //(must implement = operator(external resources))
```



Friend Function-->violate oop rule (encapsulation)
but in some cases i shoulf work with this concept

```
class A
{
int n;

public:
friend void B::sayHello();
```

```
class B
{

public :
void sayHello()
{
    cout<<n;
```

```
//Move Constructor

string name{"Ahmed"};

string fullName=name;//copy constructor {deep copying}
string FullName=std::move(name);//create a new object of type string that take a complete
owner ship of name object


cout<<name;//empty
cout<<FullName;


 #include <iostream>

 using namespace std;
 class stack
 {
    int size{5};//int size=5.5;(implicit conversion)
    int top{0};
    int * p;
    inline static int counter{0};
 public:
      stack(int size=5):size(size),top(0),p(new int[size]){counter++
 ;}//top=0;this
 -
 >size=size;p=new int[size];}
      stack /*this of s2*/(stack& other/*ref s1*/)
      {
         this->top=other.top;
         this->size=other.size;
         //deep copying part
         this->p=new int[size];
         for(int i=0;i<other.top;i++)
           this->p[i]=other.p[i];
         counter++;
         cout<<"this is my copy ctr "<<endl;

      }
    //implement Move Constructor
     stack(stack&& other) //noexcept (don't throw an exception)
 //=delete;//disable move feature//-  >rvalue refrence release its
 resources
 after
 copying
     {
        counter++;
        this->top=other.top;   this->size=other.size;  this->p=new
 int[size];
        for(int i=0;i<top;i++)
          this->p[i]=other.p[i];

main()
{
     stack ss(10);
      ss.push(1).push(5).push(7);
      stack s2=std::move(ss);//call move constructor

      s2.showstack();
      ss.showstack();//empty..

      stack myS;//already created
     myS.push(8);
     myS=s2;//call =operator//assign copy from s2 to
mys
     myS=std::move(s2);//move copy from s2 to mys
```

Operator Overloading:

type of polymorphism (with same name but diff parameter list)
same operator but work with different operands (LHS,RHS if exist)

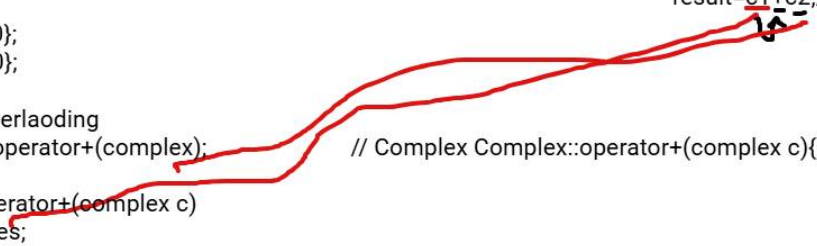
```
class Complex
{
    float real{0.0};
    float img{0.0};
public:
 //operators overlaoding
   //Complex   operator+(complex);          // Complex Complex::operator+(complex c){}//outside
class
  Complex   operator+(complex c)
 {    Complex res;
      res.real=this->real+c.real;
      res.img=this->img+c.img;
      return res;
}

  Complex   operator+(float f)
 {    Complex res;
      res.real=this->real+f;
      res.img=this->img;
      return res;
}

friend Complex operator +(float,Complex);//to access private members
};
//standalone function
Complex operator+(float f,Complex c)
{
     Complex res;
      res.real=f+c.real;
      res.img=c.img;
      return res;
}
```

```
Complex c1{2.1,3.1};
Complex c2{3.0,4.0};

Complex result;
//result=c1;//default copying
result=c1+c2;//operator +
```



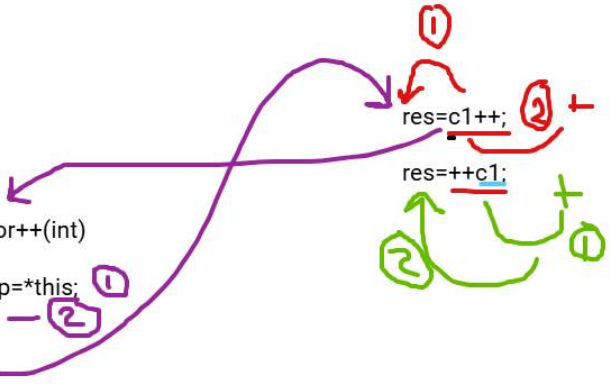
```
result=c1+2.0;
```

Float

```
result=1.5+c2;
```

```
class Complex
{
   //continue ++prefix
   Complex  operator++()
   {
       this->real++;
       return *this;
   }
```

```
//++ postfix
Complex operator++(int)
{
    Complex temp=*this;
    this->real++;
    return temp;
}
```



```
//operator +=
Complex operator+=(Complex c)
{
    real=real+c.real;
    img+=c.img;
    return *this;
}
```

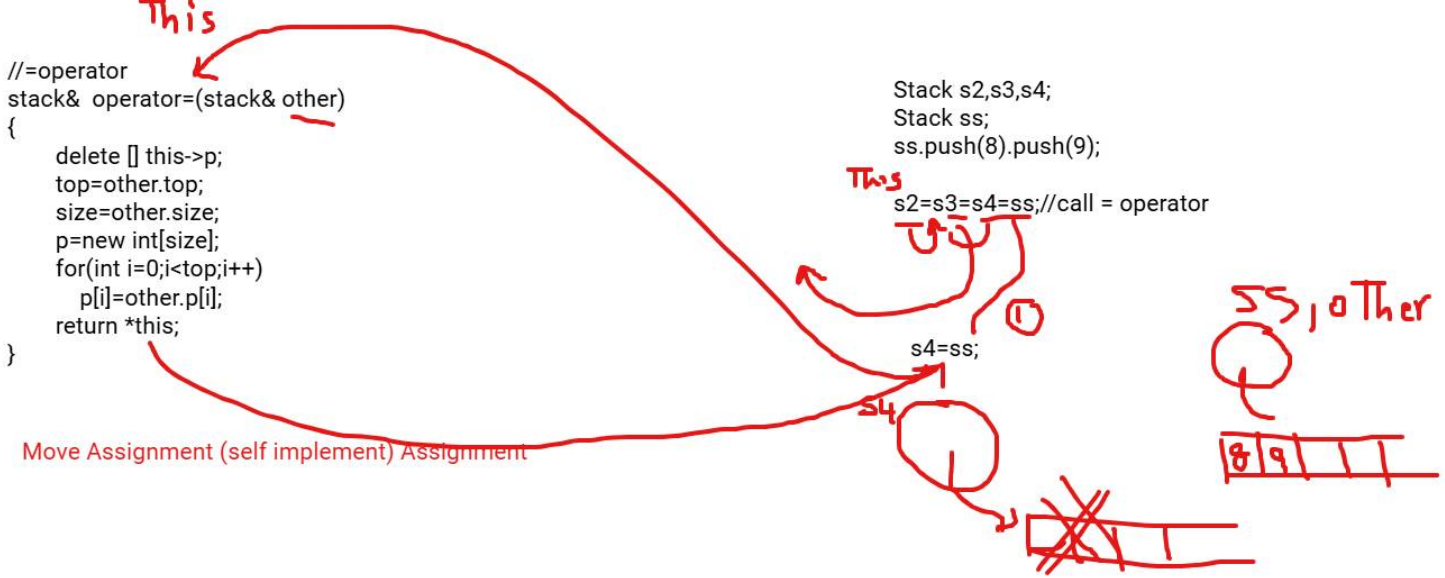
c1+=c2;

```
 int operator==(Complex c)
 {
       return ((real==c.real)&&(img==c.img))
 }

 operator float()
 {
    return  this->real;
 }
```

if(c1==c2)

float f=(float)c1;


this
//=operator
stack& operator=(stack& other)
{
delete [] this->p;
top=other.top;
size=other.size;
p=new int[size];
for(int i=0;i<top;i++)
p[i]=other.p[i];
return *this;
}
Stack s2,s3,s4;
Stack ss;
ss.push(8).push(9);
This
s2=s3=s4=ss;//call = operator
s4=ss;
ss, other
8
9
s4
Move Assignment (self implement) Assignment

1- Complex Example with full overload operators +,-,*,/ realtional < ,>, increment and

decrement operators ,type casting.
- ctr-->using uniform initialization
-ctr --->using feature default//check default const created by compiler
-overload <<, >> cout<<c1; //print content of complex instead of c1.print()(bonus)self cin>>c1;
 study

2-Stack full Example
-ctr default------>//don't allow user to cal default ctr of class
-parametrized ctr with default value of size 5
-copy ctr,Move ctr.
-overload =operator & = Move operator
-overload [ ] --> cout<<s[2];
-destructor--->delete memory ,counter --,show messages

3-Banck Account --->complete with all required ctrs,dont allow use (=operators),but

can do move accounts , overload << ,show data >> enter data