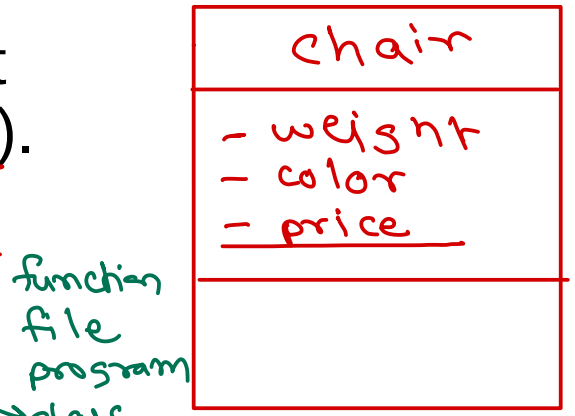
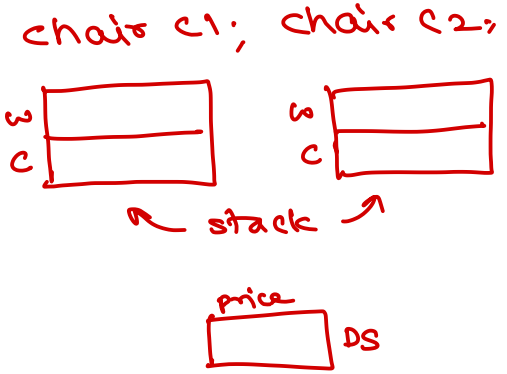
Sunbeam Infotech

Exploring new ideas, Reaching new heights!

static members → shared among all objects of the class.

- Static members belong to class (not to the individual objects of the class).
- Static members are accessed using ClassName::memberName
- Static members are in class scope.
- Even though C++ allows accessing static members on object (using dot operator), it is not good practice (misleading).
- Static data members don't contribute to size of object. They are created in DS of process (independently) and have life throughout the program.





static members

- Static member functions don't receive "this" pointer. Hence they cannot access non-static members of class directly.
- Static member functions are used to
  - access/manipulate static data members
  - perform certain operations on class level (even when its objects are not created).
- Singleton class is object oriented way of making data global (accessible throughout the program).
- Only one object of singleton class can be created. Same object can be referred in entire program.
- Singleton is one of the design pattern.

Static fns are not designed to be called an object. They are called using class name. this pointer keep address of current object. Hence Static fns do not recieve this pointer.

design patterns are solutions to common problem in sw development

problem: access same data throughout the appln in oop project (global is discouraged).
solution: singleton design pattern.

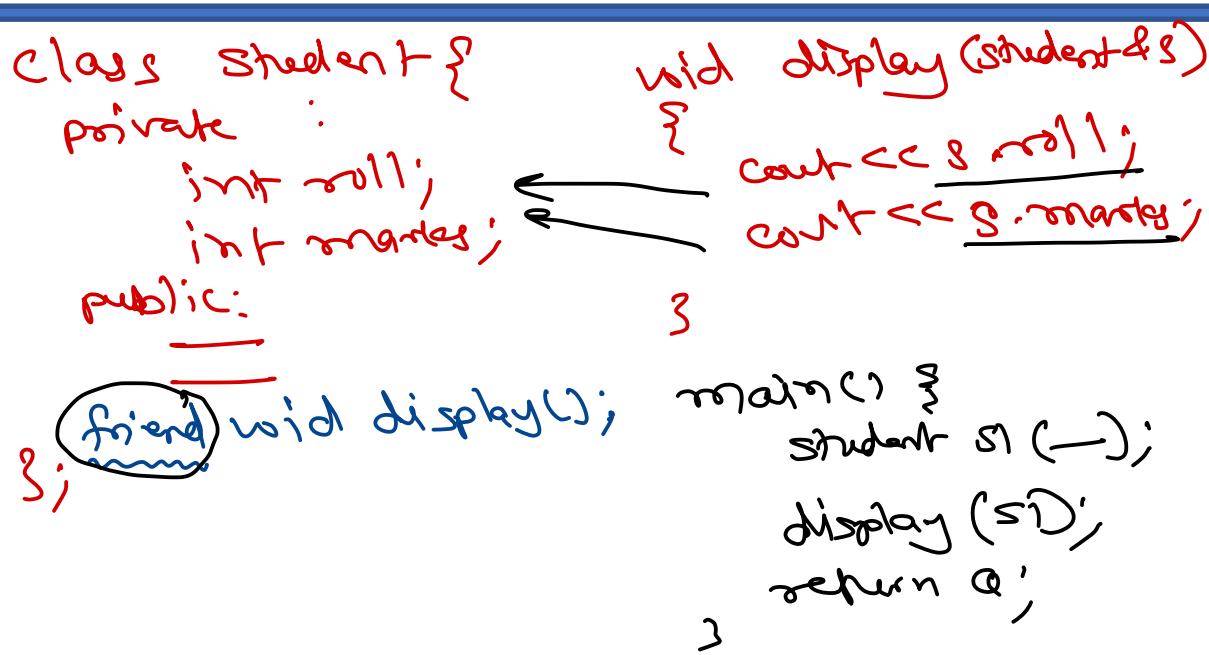
```
class singleton{
private:
    singleton(){}
    static singleton *ptr; //decln
public:
static singleton* get_instance(){
    if(ptr==NULL)
        ptr=new singleton;
    return ptr;
}
};
singleton* singleton::ptr=NULL; //defn
```

private ctor

```
main(){
// singleton obj;
Error:
private ctor
p1=singleton::get_instance()
p2=singleton::get_ptr()
}
```

Friend function & Friend class

- Friend function is non-member function of the class, that can access/modify the private members of the class.
  - It can be a global function.
  - Or member function of another class.
- Friend functions are mostly used in operator overloading.
- If class C1 is declared as friend of class C2, all members of class C1 can access private members of C2.
- Friends are
  - Not symmetric and Not transitive
  - Not inherited
- Friend classes are mostly used to implement data struct like linked lists.



Operator overloading

- By default operators work with built-in types only. → except assignment operator.
- Extending functionality of operator, so that they can be used with user-defined classes, is called as "operator overloading".
- Operators are overloaded to increase readability of the program.
- Operators can be overloaded as special functions (with operator keyword) for binary operator.
  - Member function → 1 arg: first arg is implicit 'this'.
  - Friend function → 2 args: no implicit arg.
- Operators can be unary or binary.

```
c3 = c2 + c1;
  ↳ c3 = c2.op+(c1);

c3 = c2 - c1;
  ↳ c3 = op-(c2, c1);
```

Operator overloading

- Compiler resolves operator use as appropriate operator function call (as per implementation).
- Limitations of operator overloading
  - Cannot change number of args for the operator.
  - Cannot change precedance & associativity of the operator.
  - Cannot create new operator.
  - Cannot overload few operators: sizeof, ?:, ::, dot (.), .*, typeid & casting.
  - Cannot overload few operators as friend functions: =, ->, () & [].
  - We should not change meaning of operator.

member
friend

2 + 3 * 4

$

overloading these operators will change their meaning. And it may be mis-leading. So these operators cannot be overloaded.

obj = ...;
obj -> ...;
obj ( );
obj [-] = x;

these ops are designed to be called on object only. Hence they must be overloaded as member.

Operator overloading

- Following operators can be overloaded
  - Arithmetic + , - , * , / , % ,
  - Relational & Logical < , > , <= , >= , == , != | && , || , !
  - Post/pre increment/decrement ++ , --
  - Insertion/extraction << , >>
  - Assignment operator =
  - Shorthand operators += , -= , *= , --
  - Subscript operator [ ] → object as array
  - Arrow operator -> → smart pointer.
  - Function call operator ( ) → functin object
  - Comma operator ,
  - new & delete operator -

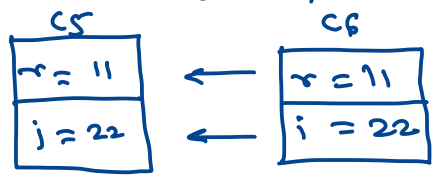
bool return

Operator overloading

deep copy vs shallow copy

copy constructor & assignment operator.

c5 = c6; → c5.op=(c6);



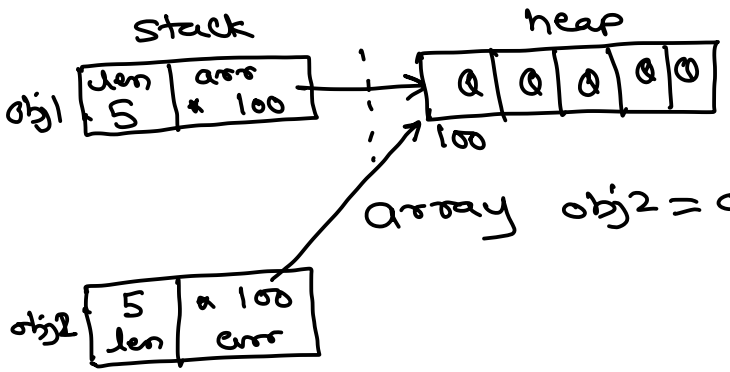
shallow copy (bitwise copy) ← default assign operator

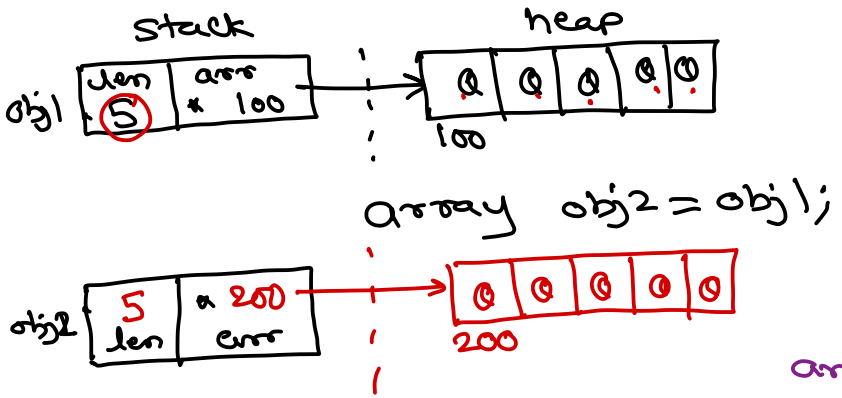
```
class Complex {
    =
Complex& operator=(Complex& other){
        this->real = other.real;
        this->imag = other.imag;
        return *this;
    }
};
```

if copy ctor is private, cannot create copy & cannot pass/return object by value.

if assign op is private, cannot assign object to another obj.

array obj1(5);





OR

```
array obj2(obj1);
array obj2 = obj1;
```

↳ create new obj as copy of existing obj.

② pass by value → formal arg is copy of actual arg.

③ return by value.

```
array(array &other) {
    this->len = other.len;
    this->arr = new int[len];
    for(i=0; i<len; i++)
        this->arr[i] = other.arr[i];
}
```

```
array obj3(3);
obj3 = obj1;
```

↳ assigning one existing obj to another existing obj

```
array& operator=(array &other) {
    delete[] this->arr;
    this->len = other.len;
    this->arr = new int[len];
    for(i=0; i<len; i++)
        this->arr[i] = other.arr[i];
    return *this;
}
```

Operator overloading

c1.display();

no such method present in ostream.

cout << c1;

↳ cout.op<<(c1);

ostream class
↳ op<<()
Complex

no such method can be implemented in ostream class (bcoz, it is predefined class).

op<<(cout, c1);

↘ can implement as friend fn of Complex class.

cout << c1 << c2;

cout

op<<(op<<(cout, c1) , c2);

ostream class.

cout << i;

↳ cout.op<<(i);

cout << f;

↳ cout.op<<(f);

cout << "---";

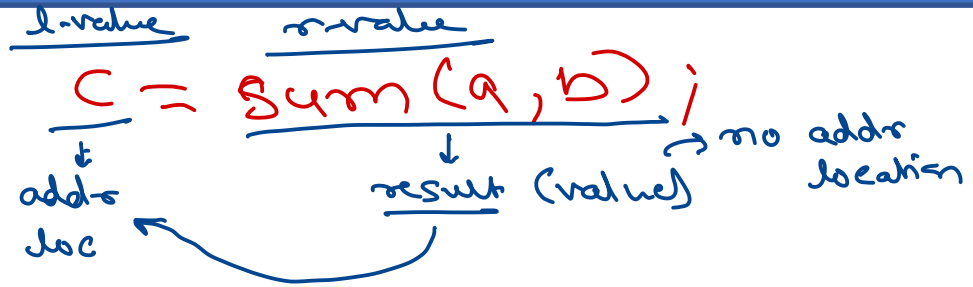
↳ cout.op<<("---");

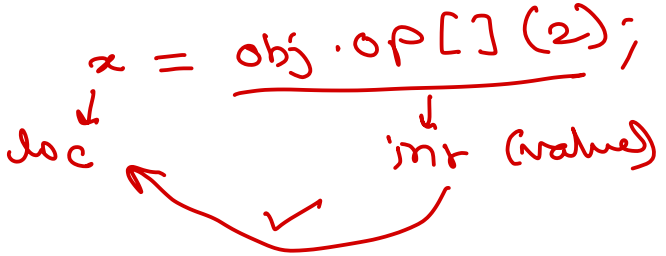
cout << i << f;

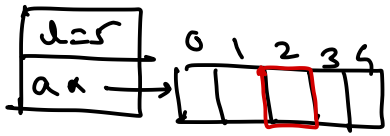
① ② cout.op<<(i).op<<(f);

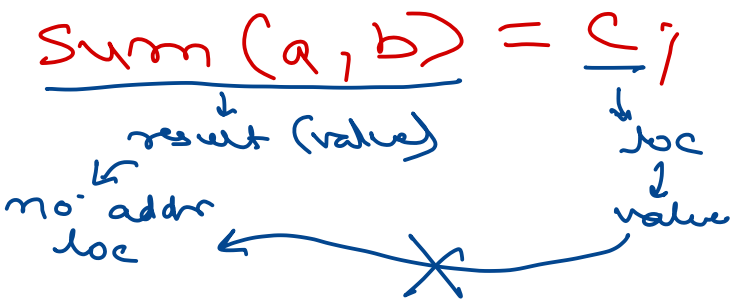
cout (return).

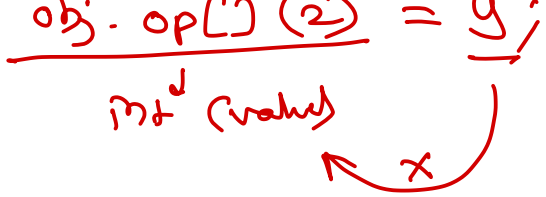
Operator overloading











Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>