OOP Course

Created by : Israa Abdelghany
LinkedIn : www.linkedin.com/in/israa-abdelghany-4872b0222
GitHub : https://github.com/IsraaAbdelghany9

Lecture 1

```
void setId ( int _id)
{  id = _id; }

int getAge ()
{return age;}
```

or

```
void setId (int id)
{ this -> id = id; }
```

Picture in iPhone

Lecture 2

struct in c & c++ ⇒ functions & access modifier (default public)

class ⇒ default private ⇒ till now its the only diffrence

void print (emp e) ⇒ called stand alone function → can be called directly

{ cout << e.getId() << ":" << e.getName() ;

e.setId (9000) ; ⇒ call by value

}

functions must be after class &struct & include

- instant member or function member ⇒ functions of classes

```
*main.cpp
40  void print(emp e);
41
42  int main()
43  {
44      emp e1;
45      e1.setId(10);
46      e1.setName("aly");
47      e1.setAge(30);
48      print2(&e1);
49      print(e1); ⇒ creates a copy in function
50      return 0;
51  }
52  //stand-alone function
53  void print(emp e)//
54  {
55      cout<<e.getId()<<":"<<e.getName()<<":"<<e.getAge();
56      e.setId(9000); ⇒ will be deleted after exiting the function
57  }                          because it's a copy
58  void print2(emp* e)
59  {
60      cout<<e->getId()<<":"<<e->getName()<<":"<<e->getAge();
61      e -> setId (9000); ⇒ will save it even after exiting
                                function
Logs & others
```

* any member function has a hidden parameter called this pointing to its object

* if you debug the code you will find this

* this -> id because this a pointer

```
class Complex
{
    int real;
    int img;

    Public :
        //setters & getters   (void)

        void setReal ( int _real)
          { real = _real ; }


        void set img (int _img)
          { img = _ img ; }

        int get Real ()
        { return real ; }

        int get img ()
          {return img ; }

          void print ()
          {
              if (img >0)

                cout << real << "+" << img << "j" ;

            else if (img <0)

                cout << real << img << "j" ;

              else

                Cout << real ;

          }

          void setComplex (int _real , int _ img)

              { real = _real ;
                img = _ img ;
              }
```

```
Complex Add (Complex c)
{
    Complex res;
    res.real = c.real + real;
    res.img = c.img + img;
    return res;

    // if I typed img = 9000; it will assign this value to c2
}
};

int main()
{ complex c1, c2, c3;
  c3 = c2.Add(c1);
}
```

default by it's this →

calling by value

calling by address

in function its called by this

*Note :-

polymorphisum ⇒ differ different functions by number of arguments or type of them (overloading)

* over loading depends on arguments not the return type.

* Constructor is called directly after the object creation

it is used if we need to do something right after the object created

* Constractor [1] is called once

[2] has same name as the class

[3] no return type

```
emp () { age = 30;
         id = 0;
         strcpy(name, "no name");
```

* constructor is called parameterless constructor because it don't have parameters to get emp()

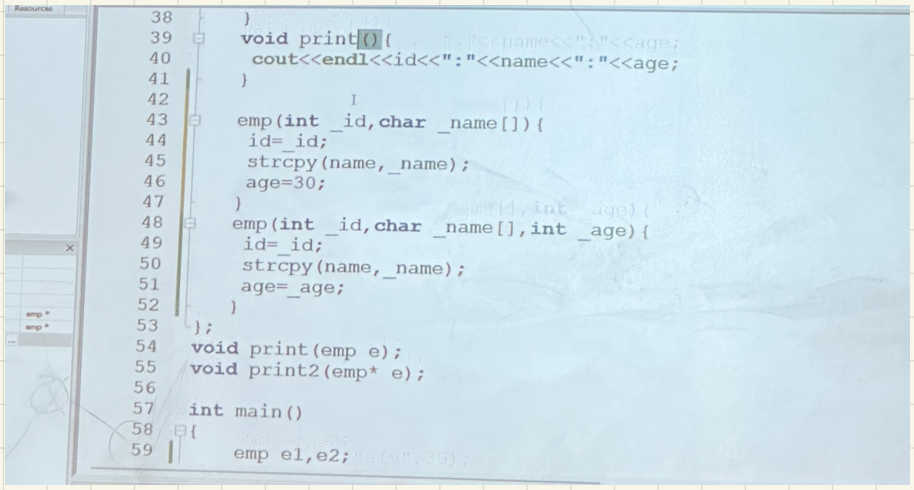
* we can overload the constructor.

* emp e1, e2; ⟹ will call the parameterless constructor

emp e3 (10, "aly", 30); ⟹ emp (int _id , char _name[], int _age)

{ id = _id;
age = _age;
strcpy (name, _name);
}

- if I typed just emp e1, e2; ⟹ No error
- if I typed emp e3 (10, "aly", 30); ⟹ with



```
38      }
39      void print(){
40       cout<<endl<<id<<":"<<name<<":"<<age;
41      }
42
43      emp(int _id,char _name[]){
44       id=_id;
45       strcpy(name,_name);
46       age=30;
47      }
48      emp(int _id,char _name[],int _age){
49       id=_id;
50       strcpy(name,_name);
51       age=_age;
52      }
53  };
54  void print(emp e);
55  void print2(emp* e);
56
57  int main()
58  {
59      emp e1,e2;
```

*Constructor can be private

* when you create any constructors the default const. needs to be created too or error if you call it or emp e1;

* destructor is ~name_of_class

* destructor is called when object is deleted or code is finished

* if the function the objects created in is not ended the object can't be deleted because it's saved in stack

```
int main()
{  emp* e1;        ⇒not object
   emp * e2= new emp (10, "ahmed");
   e2 → print ();
   delete e2;
}
```

calling instructor

I have created pointer that created an object (place to save the object)

delete e2; ⟹ to delete it from the heap it will call destructor to delete its object

* Copy Constructor

used in 4 cases

#search

* Creating object from object call the copy Constructor.

Exam :- → mcq.

→ 2 questions code typing.

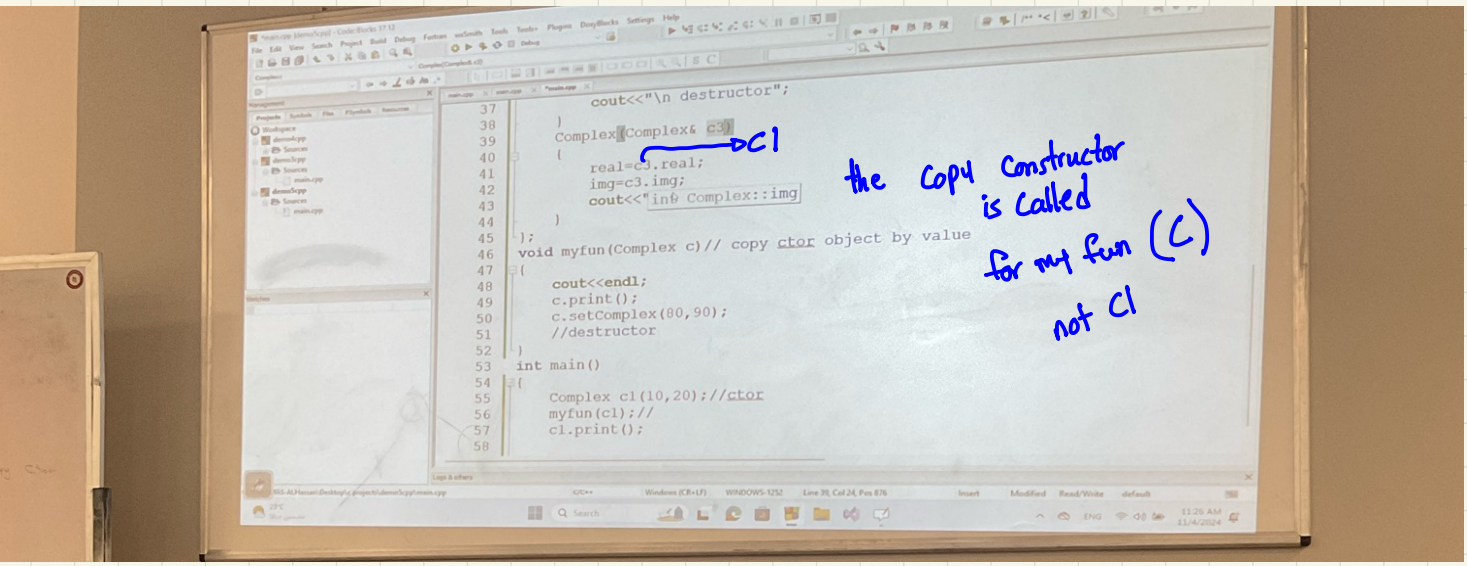
LAB

1. Complex → setter,
   getter
   print
   constractor
   Add (member function & stand alone)

2. emp class → Constructor
   destractor
   trace

Lecture 3

- complex class
  - setters
    - img
    - real
    - complex
  - getters
    - Real
    - Img
  - print
  - Constructor
    - set one param (real)
    - set both img & real
  - Copy Constructor
  - destructor



c1 is c3

this → object member
↳ instance member

```
    ~Complex(){
        cout<<"\n destructor";
    }
    Complex(Complex& c3)
    {
        real=c3.real;
        img=c3.img;
        C3. img= 7000; C3. real= 5000;
        cout<<"\n copy ctor";
    }
};
void myfun(Complex c)// copy ctor object by value
{
    cout<<endl;
    c.print();
    c.setComplex(80,90);
    //destructor
}
int main()
{
    Complex c1(10,20);//ctor
    cout<<"\n address of c1"<<&c1;
```

```
45          c3.img=8000;
46          cout<<"\n copy ctor";
47      }
48  };
49  void myfun(Complex c)// copy ctor object by value
50  {
51      cout<<endl;
52      c.print();
53      c.setComplex(80,90);
54      //destructor
55  }
56  int main()
57  {
58      Complex c1(10,20);//ctor
59      cout<<"\n address of c1"<<&c1;
60      myfun(c1);//
61
62      c1.print();   => 5000 + 7000 j
63
64      return 0;
65  }
66
```

my fun create object by value so to initialize the copy of the function ⇒ copy constructor is called

creating object ⇒ copy constructor is used

after finishing myfun will call the destructor to destroy the copy which is c

if myfun (Complex & c) ⇒ no copy will be called
- alias created
- alias deleted after finishing myfun

myfun (c1)

if myfun (Complex * c)

```
c -> print();
c -> set Complex (2,3);
```

my fun (&c1)

Case 2 of Copy Constructor

```
int main()
{
    Complex c1(10,20);
    Complex c2(c1);
```

⇒ will call Copy Constructor

OR

```
    Complex c4 = c1;
```

⇒ will call Copy Constructor

Not like ⇒ `C4 = C1;` ⇒ no Copy Called

```
}
```

if my_fun (Complex* c3) ⇒ no destructor is called

```
Complex(Complex& c3)
{
    real=c3.real;
    img=c3.img;

    cout<<"\n copy ctor";
}
};
void myfun(Complex c,Complex r)// copy ctor object by value
{
  cout<<"\n myfun\n";
   //destructor
}
int main()
{
    Complex c1(10,20);
    Complex c4(80,90);
    myfun(c1,c4);

    return 0;
}
```

```
Complex::Complex()
Complex::Complex(Complex& c3)
Complex::Complex(int _real)
Complex::Complex(int _real, int _img)
class Complex {...}
```

2 Constructor
2 Copy Constructor
4 destructors

if I defined the Copy constructor I need to define its function myself or it will return garbage data to the object is copying data

```
Complex (Complex & c3)
{
    this->real = c3.real;
    img = c3.img
    cout << " in copy constructor " << endl;
}
```

Some notes: any thing in gray is a note not part of the code.

1. real = real of the c (the object copying the data)
2. c3 is c1 which is sent to the function.
3. this ⟹ is an instance pointer for c not for c1 or c3
4. if I commented real=real and img = img this will not affect the values of c3 but the copy created from c3 which is c will have garbage data.
5. I can access and change the data of c3 and change it like c3.setimg(3); c3.setreal(2); and this will also affect the c1 object

* Cases where Copy Constructor is called :-

1. object by value to function (not ref or *)
2. Complex C4(C1)
3. complex C4 = C1
4. Return object by value

```
C5 = myfun(C1, C2);
```

↳ after exiting myfun it's destroyed but saving the value in temp using Copy Constructor then saved in C5. then temp is deleted.

* all return save the value in a temp (work in a same way)

* I don't need to type copy constructor my self but I need to define it if there is a dynamic

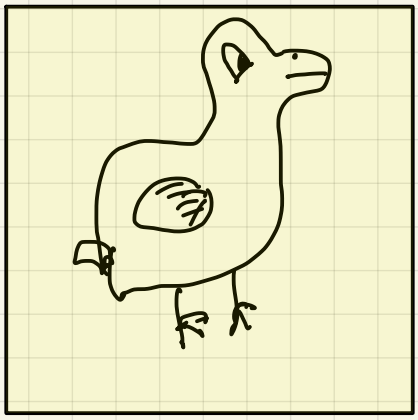
---

```
complex myfun(Complex C3)
{ Complex res;
} return &res ⟹ error
```

But if

```
Complex* myfun(Complex C3)
{ Complex * res = new Complex();
} return &res; ⟹ will not Call Copy Constructor and no error
                but not the value of res
```

Stack



LIFO

(Last in first out)

* to create that I need to use OOP to create restrictions on the usage and hide my array

* I need variable to keep tracking the current (last used) index tos (top of stack) initialized by -1
* pop will decrease the index and push will increase it and enter data
* tos range is -1 : (range - 1)
  - -1 : empty
  - (range - 1) : full

```
Class stack
{
    int arr[5];
    int size;
    int tos;
  Public:
    stack ()
    { tos = -1;
      size = 5;
    }
```

```
void Push (int d)
{
  if(tos < size -1)
    { tos++;
      arr[tos]=d;
    }
  else cout<<" stack is full ";
}

int pop ( int & d)
{  if (tos != -1)
    {
      d = arr[tos];
      tos --;
      return 1;
    }
  else
    {cout <<"stack is empty";
     return0;
    }
}
```

check again

```
Stack s1;
s1.Push(30);
s1.Push(40);
s1.Push(50);
s1.Push(50);
s1.Push(50);

int x=0;;
if(s1.pop(x)==1)
    cout<<endl<<x;//50
if(s1.pop(x)==1)
    cout<<endl<<x;//40
if(s1.pop(x)==1)
    cout<<endl<<x;//30
if(s1.pop(x)==1)
```

```
Class stack
{
    int size;
    int tos = -1;
    int arr[5];

  Public:
    stack ()
    { tos=-1;
      size = 5;
          arr = new int[size];
    }

   ~stack ()
    { cout << "\n destructor";
        delete [] arr;
    }
    stack (int _size)
     {  size = _size;
        tos = -1 ;
        arr = new int [size];
     }
```

S1 = S2 ⇒ donot produce errors ⇒ but tos = tos which is wrong

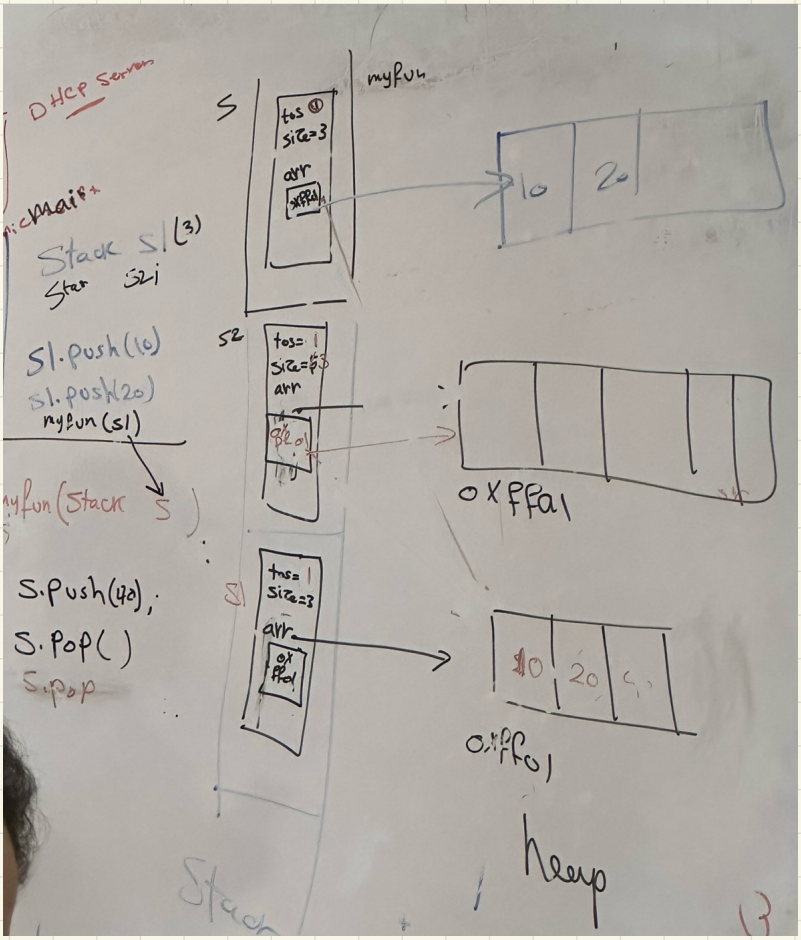
I need to define the copy constructor if I have any dynamic allocated data.

Because if I passed the data to function by value (shallow copy) it will delete the memory for the dynamic allocated data after exiting the function so I can't use it in main function

The solution is:- difining the copy constructor by myself ★

```
            cout<<"\n stack is empty";
            return 0;
        }
    }
    Stack(Stack& s){
        tos=s.tos;
        size=s.size;
        arr=new int[size];
        for(int i=0;i<=tos;i++)
            arr[i]=s.arr[i];
    }
    ~Stack(){
```

After modification


DHCP Server
Stack S1(3)
S1.push(10)
S1.push(20)
myfun(s1)
myfun(Stack S)
S.push(40);
S.Pop()
S.pop
S
myfun
tos 1
size=3
arr
10
20
S2
tos=1
size=3
arr
0xffa1
S1
tos=1
size=3
arr
0xffa1
10
20
0xffa1
Stack
Heap

Example of copy constructor

```
myfun(Stack s)
{
    int x;
    s.pop(x);
    cout<<endl<<x;
    s.pop(x);
        cout<<endl<<x;
s.Push(30);
s.Push(40);
}
int main()
{
    Stack s2(2);
    s2.Push(1);
    s2.Push(2);
    myfun(s2);
    int x;
    s2.pop(x);
    cout<<endl<<x;
```

it's undetermined behaviour so runtime error

```
2
1
 destructor
2
 destructor
Process returned 0 (0x0)   execution time : 0.230 s
Press any key to continue.
```

if I didn't create a copy constructor it will be shallow copy so reffer to same array and delete it after finishing.

LAB

* stack static

// dynamic

random notes

taken in my note book

```
Complex *ptr = new Complex
();
```

NOTE:-

1. ⇒ new creates an object and point at it using the pointer.

1

```
void myfun (Complex C)
{ C.print(); }

int main()
{ Complex C1(10,20);
  myfun(C1);
  return 0
}
```

```
void myfun (Complex &c)
{ c.print(); }
  c.setComplex (80,70);
}
```

the only change

* alias is deleted when exit function

```
int main()
{ Complex C1(20,30);
  myfun (C1);
  C1.print(); return 0; }
```

3

```
Complex C2(10,50);
Complex & ptr = C2;
ptr.print();
```

Refrence

```
Complex *ptr = new Complex(10,20)
ptr -> print();
delete (ptr);
```

Heap

```
complex & ptr = C1;
```

✓

* Complex &ptr → Compilation error

Alias

2

Copy Constractor

* Creation in Class :-

```
Complex (Complex& C3)
{
  real = C3.real;
  img = C3.img;
  Cout << "Copy Const";
}
```

alias not Creating new obj.

5

In alias example :- (Copy Constructor) if object is sent to function by value a copy constructor created by default is copying the data from the main to function using bit wise operation (bit wise copy)

In the function it's working with a copy if by value and destroying the object after that during exiting the function

4

the C3 is Same as C1

So if

Cout << "the address of"

6

Lecture 4

Online

- the default copy constructor creates shallow copy
- friend function can access private members
- friend & stand alone function violates the rules of OOP
- friend name of stand alone function

1-send object by value to function
2-return object by value
3- stack s1(s2);
4- Stack s2=s1;
// complex copy ctor can not be implemented
// stack static allocated array we can not implement copy ctor
// stack : dynamic allocated array shallow copy bitwise copy
we must implement copy ctor

operator + ⟹ overload // Complex + Complex

```
        img= _img;
    }
    //complex.opertaor+(complex)
    // complex + complex
    Complex operator+ (Complex c)
    {
        Complex res;
        res.real=real+c.real;
        res.img=img+c.img;
        return res;
    }
};
int main()
```

if I added real= 9000; will change c1

```
int main()
{
    Complex c1(10,20);

    Complex c2(5,6);
    Complex c4(8,9);
    Complex c3;
    //c3=c1.operator+(c2);
    c3=c1+c2;
    c3.print();

    return 0;
}
```

I can use + operator now

c1 + c2

c1 caller object

By value

```
    }
    //complex.operator+(complex)
    // complex + complex
    //c1+c2
    Complex operator+ (Complex& c)
    {
        Complex res;
        res.real=real+c.real;
        res.img=img+c.img;
        return res;
    }
};
```

const (better to add it)

```
Complex::Complex()
Complex::Complex(int _real, int _img)
class Complex {...}
```

this will not call copy Const (preffered)

```
// complex + complex
//c1+c2
Complex operator+ (Complex& c)
{
    Complex res(real+c.real,img+c.img);
    //res.real=real+c.real;
    //res.img=img+c.img;
    return res;
}
// complex +int
//c1+3
Complex operator+(int x){
    Complex res;
    res.real=real+x;
    res.img=img;
    return res;
}
```

اختصار للكود

if 3+C1 → Compilation error

```
54
55  friend Complex operator+(int x, Complex c);
56
57  };
58
59  Complex operator+(int x, Complex c)
60  {
66  int main()
67  {
68      Complex c1(3,4);
69      Complex c2(30,4);
70      if(c1==c2)
71          cout<<"equal\n";
72      else
73          cout<<"not equal\n";
74
75      Complex c4(8,9);
```

binary will print not equal

```
    int operator == (Complex c){
        return real==c.real && img==c.img;
    }
    int operator!=(Complex c){
        return !(*this==c);
    }

friend Complex operator+(int x, Complex c);

};
```

→ Solution

→ Implementation of == & !=

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

→ Implementation of +=

c6 = c3+=c1 this type need return

Unary operators

```
Complex operator++()//prefix
{
    real=real+1;
    return *this;
}
```

→ Implementation of ++ pre fix

```
Complex operator++(int)//postfix
{
    Complex temp=*this; // temp 3,5
    real=real+1; c1 ,real=4,5
    return temp;//3,5
}
```

→ Implementation of ++ post fix

```
4+5J
3+5J
4+5J07+5J
Process returned 0 (0x0)   execution time : 3.889 s
Press any key to continue.
```

Rules of operator overload.

```
}
explicit operator  int(){
    return real;
}
```

→ Cast

```
x=(int)c1;
```

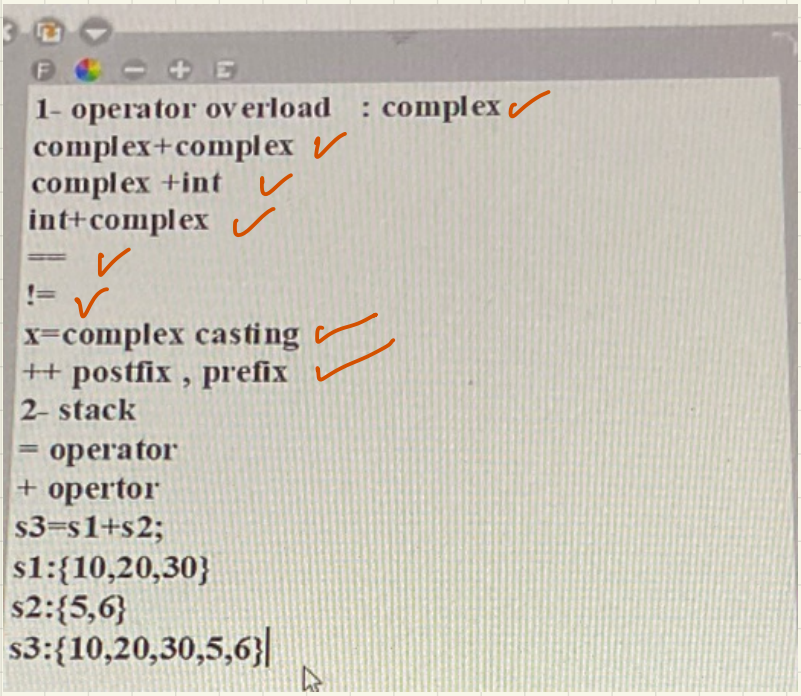
```
Complex operator=(Complex c){
    real=c.real;
    img=c.img;
    return *this;
}
```

```
    cout<<endl;

    Complex c6;
    c1.print();
    c6.print();

    Complex c4(8,9);
    //Complex c3;
    //c3=c1.operator+(c2);
   // c3=c1+c2;
    c3=c1+3;
    c3=3+c1;
    c3.print();
    // complex + complex member function , sta
    //complex + int member function , stand-al
    //int + complex , stand alone function
```

```
Complex c1(3,5);
Complex c2(30,4);
Complex c3;
c3=c2=c1;
c2.print();
cout<<endl;
//c2=c1++;
int x;
x=(int)c1;
c1=x;// constructor casting
cou int main::x
cout<<endl;
c1++.print();
c1.print();
cout<<endl;
c2.print();
cout<<endl;
```

```
Complex (int real)
{
    real = -real ;
    img = 0 ;
}
```

LAB



3-fraction number
num/den data, attributes
ctor , paramless, one param , two param
setters, getters
print
fraction+fraction

Day 5 (online)

to day schedule :-

- static members , object members
- array of objects
- array of pointers to objects
- object relations (Composition , aggregation , association , inheritance)

- static → make the member → class member instead of object member

---

- static function inside class → can call it without object
  → cout << Complex :: getCounter();
  and if you tried to print any other member of object
  └ compilation error because static functim doesn't have this pointer
  Ex:- static int getCounter() return Counter;
  if trying to add cout <<real ⟹ Compilation error

- Even if counter is static it's still private so you can't access it directly from main function ⟹ Compilation Error
  you need getCounter functim. → this function need to be static to be able to call it without object

```
int main ()
{
    {
        Complex S1 (30);   ⟶ this object will be created here
    }   ⟶ deleted when exit this {
    return 0;
}
Public:
  static getCounter ()
  {
      real = 60;   ⟶ Compilation error (static fun)
      return Counter;
  }
```

- any function can
- :: ⇒ scope operator (scope resolution)

  it will not be available in higher languages

- if C4.getCounter ⇒ Compilation error

because the higher languages consider it as compilation error

- Array of objects in stack :-

Complex arr[3];

Coding Example :-

```
int main()
{

    Complex arr[3]={Complex(10,20)};
    for(int i=0;i<3;i++)
        arr[i].print();

    cout<<Complex::getCounter();

    return 0;
}
```

⟹ initialized the first obj.

OR →

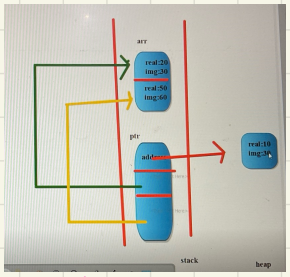
```
Complex arr[3];
arr[0].setComplex(20,30);
arr[1].setComplex(50,60);
arr[2].setComplex(70,80);

for(int i=0;i<3;i++)
    arr[i].print();

cout<<Complex::getCounter();
```

- Note Complex * ptr ; it's not an object it's pointer of type Complex (Counter still 0)
- Same if Complex * ptr[3]; it is array of pointers No objects (Counter = 0)

```
int main()
{
    Complex * ptr[3];
    ptr[0]=new Complex(10,30);
    ptr[0]->print();
```

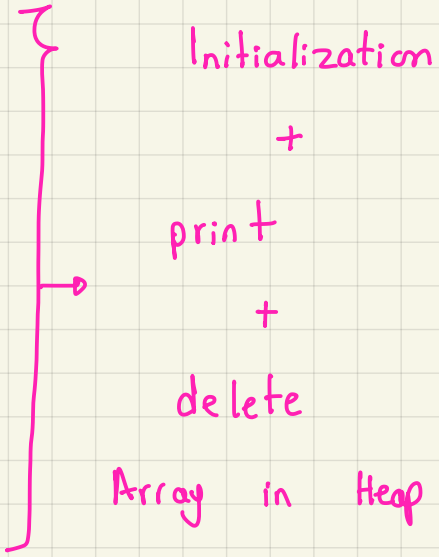
```
int main()
{
    Complex * ptr[3];
    ptr[0]=new Complex(10,30);
    ptr[0]->print();

    Complex arr[2];
    arr[0].setComplex(20,30);
    arr[1].setComplex(50,60);
                = arr
                = arr+1
    ptr[1]=&arr[0];
    ptr[2]=&arr[1];
    ptr[1]->print();
    ptr[2]->print();
    delete ptr[0];
```

```
"C:\Users\MAS-ALHassan\Des
10+30J
20+30J
50+60J
3
 object deleted
 object deleted
Process returned 0 (0x0)   execution time : 0.225 s
Press any key to continue.
```



↳ • here first Complex is in the heap but the other 2 are in the stack

```
int Complex::counter=0;
int main()
{
    Complex * ptr[3]={
        new Complex(3,4),
        new Complex(5,6),
        new Complex(7,8)
    };
  for(int i=0;i<3;i++)
        ptr[i]->print();
  for(int i=0;i<3;i++)
    delete ptr[i];
```



delete [] ptr ; ⇒ will not delete the array

• Relations

we have 4 relations (part , whole) (جزء , كل)

1. wall _ room : Composition : strong agregation :Consist of

• the part may belong to one whole , the life time of part &whole is the same (علاقة القلب بالإنسان) (الجدار والغرفة) (الاثنين اعتماد كامل على بعض)

2. team _ match : agregation : weak agregation

• if match is ended team will remain but the match depend on team. (احد الأطراف يعتمد على الاخر)

• Uni -direction

3. student _ instructor : association : deal with

• don't depend on each other and not part of

• this relation performed using pointers reffering to the other one

• Bi_direction

4. Inheritance : kind of (is a)

```
class A
{
}
Class B// whole
{
        int x
        A a1; //part

}

class match{
  team* a;

}
c c1
```

- in Rectangle Class ⇒ points are created first then Rect object

- But in destroying the Rect obj ⇒ Rect is deleted first then points

  - Rect (int x1, int y1, int x2, int y2) : ul (x1, y1), rl (x2, y2)

Complex * ptr OR Complex * ptr[3]

in both Counter will remain 0 because it's pointers not objects

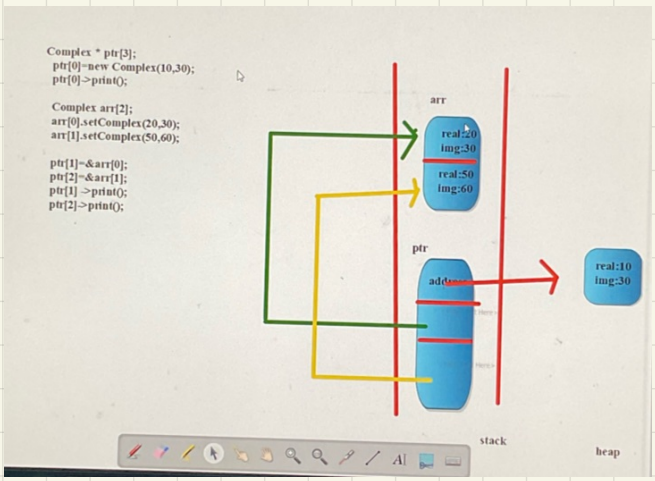
```
Complex * ptr[3];
  ptr[0] = new Complex(10,30);
  ptr[0] -> print();
```

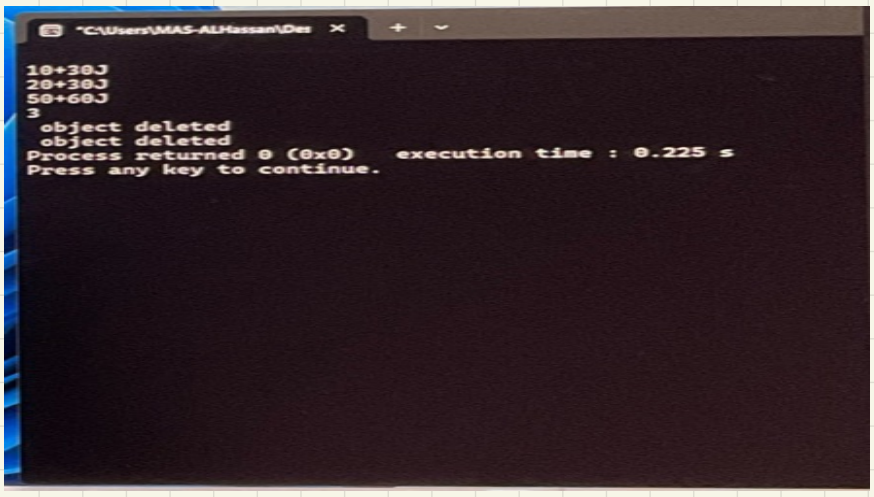
* Counter = 1

```
Complex arr[2];
arr[0] = setComplex(20,30);
arr[1] = setComplex(50,60);
ptr[1] = &arr[0]; //=arr
ptr[2] = &arr[1]; //arr+1
  ptr[1] -> print();
  ptr[2] -> print();
```

* Counter 3 #

```
delete ptr[0];
```





```
int Complex::counter=0;
int main()
{
    Complex * ptr[3]={
        new Complex(3,4),
        new Complex(5,6),
        new Complex(7,8)
    };
 for(int i=0;i<3;i++)
        ptr[i]->print();
 for(int i=0;i<3;i++)
   delete ptr[i];
```

```
int Complex::counter=0;
int main()
{
    Complex * ptr[3]={
        new Complex(3,4),
        new Complex(5,6),
        new Complex(7,8)
    };
 for(int i=0;i<3;i++)
        ptr[i]->print();
 //for(int i=0;i<3;i++)
   delete[] ptr;
```

don't delete

```
"C:\Users\MAS-ALHassan\Des
3+4J
5+6J
7+8J
Process returned 0 (0x0)   execution time : 0.221 s
Press any key to continue.
```

4

Day _ 6

is_a >> Inheritance << kind of

الوراثة

State Vs Identity.

Point p1 (20, 30)

Point P2(40, 50)

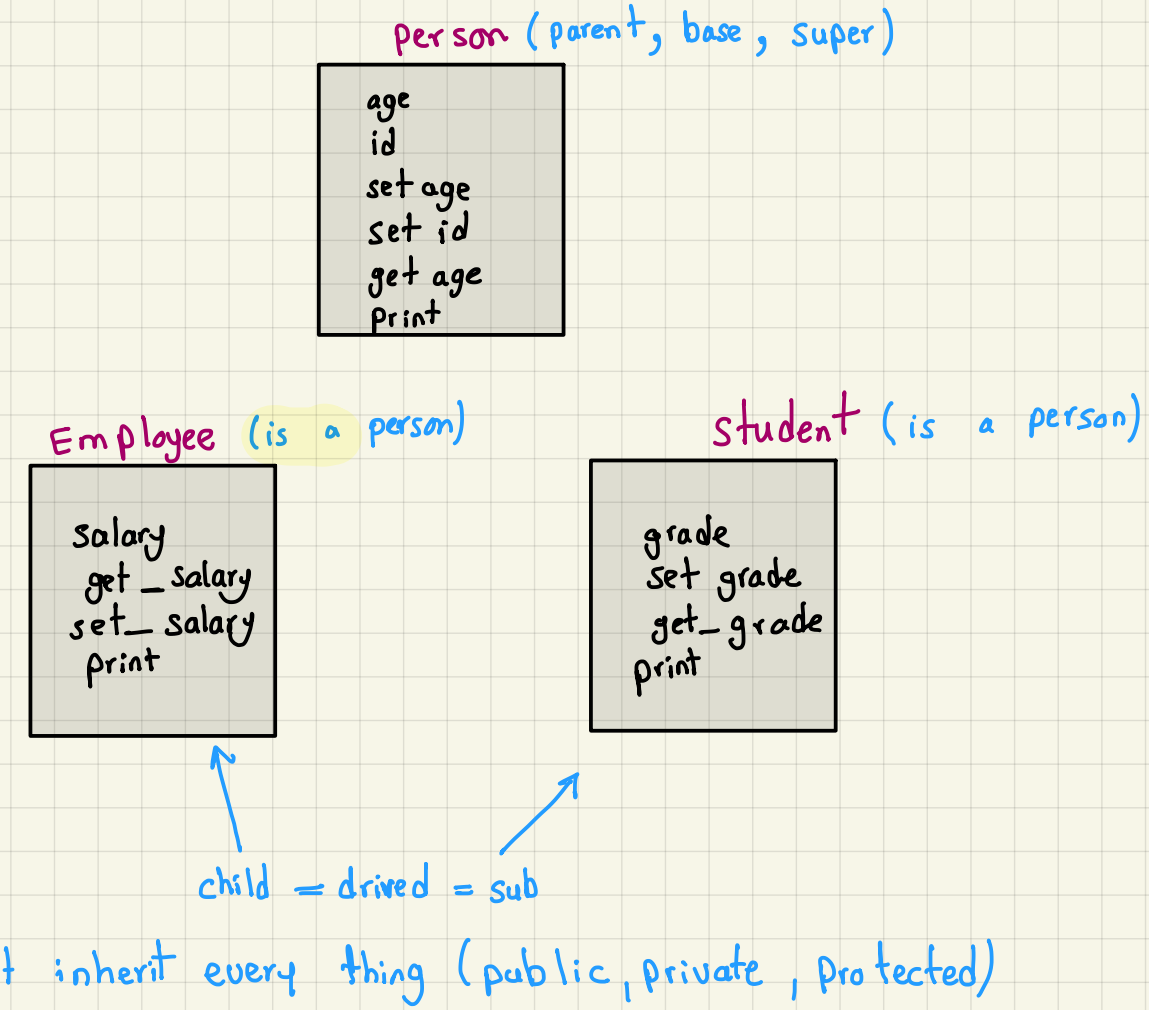
State (values)

Location in the memory is the Identity

Point * p = &p1;

Point * p3 = &p1;

} reffer to same address so they have same identity



it inherit every thing (public, private, protected)

- if I have a dog class I can not inherit it from Person (logical error)
- private members of the parent are not accessable by childs but it can be accessible if protected instead of private
- Point *p is not an object
- if member is protected I can access it in the child using this ⟹ cout << this -> ID << endl ;

```
// the problem here is that the parent should be created before the child so when I call this constructor
// it calls the parent constructor(parameterless at first then call this constructor)
// so it is reassigning the values for the same object twice
//  and if the parent do not have a parameterless constructor it will give compilation error
   emp(int _ID , int _Age , char *_name )
   {
       setID( _ID ) ;
       setAge( _Age ) ;
       setName( _name ) ;
       salary = 3000;
   }
```

TABLE 9.1 Inheritance and Accessibility

Access Specifier	*Accessible from Own Class*	*Accessible from Derived Class*	*Accessible from Objects Outside Class*
public	yes	yes	yes
protected	yes	yes	no
private	yes	no	no

when to use Inheritance.

- if I don't have the code only object file I can create a child of class of the obj
- if I have good class that I want to add new feature it is preffered to create a child of it to save

adv :- . reusability
- take data from code and build on it in another class
- generalization myfun (Person s) ⇒ can take emp or student too
- create child for the object file and can override later

protected member let child access the data of the parent.

private members can be توريث

constructor يورث ✗ ⇒ compilation error

access modifiers:-

public
private
protected

* if parameterless constructor of the parent is not availble ⟹ Compilation error

* parent is called at first then child

* Composition relation ⟹ part then whole.

* Constructor & destructor ⇒ opposite to construction child then parent

Code example :-

```
Class Person
{
    int id;
    char name[20];
    int age;
  Public:
    void setId(int _id) {id = _id;}
    void set name(char *_name) { strcpy(name, _name);}
    void setAge (int _Age) { age = _Age;}
    int get id(){ return id;}
    char* get name () { return name;}
    int getage() { return Age;}
    print
```

Lecture
Day - 7

* very important
* assign by value ⇒ normal =
* in intialization ⇒ e1=2 ⇒ copy Constructor
* Constructor can be private
* e1 = e2 ⇒ Copy Constructor
* function returning value has a temp object deleted after after the return. and the objects are deleted after exit the Function
* sending object to function ⇒ Composion relation

base = child ⇒ No error ⇒ object of parent = object of child
will take a copy of parent from the child and assign it in the <base>

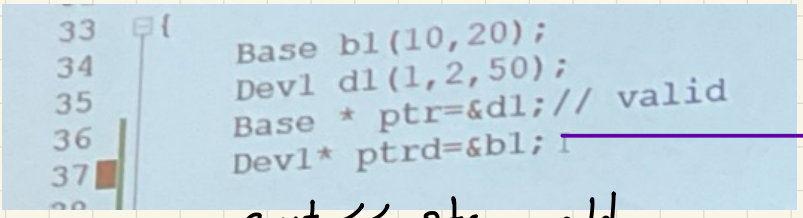
child = base ⇒ will give error <variable will be missed>
even if there's no missed variables

pointer of base point to object of base

```
Base * ptr = &b1;
cout << ptr->add();
```

pointer of child point to object of child

```
Base * ptr = &d1;
cout << ptr->add();
```



→ Compilation error

step of pointer is as child not parent

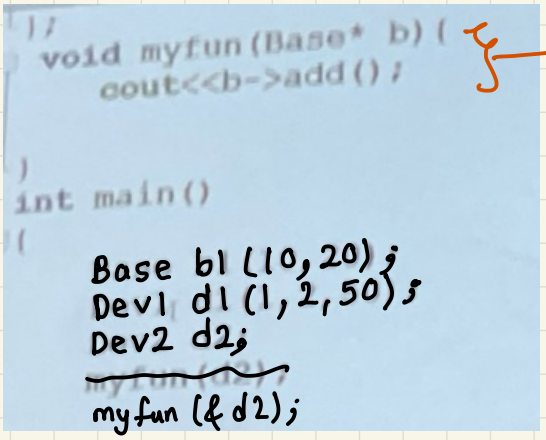
cout << ptr -> add

call add of the pointer type

★ Virtual :- الباطن مش الظاهر

- will look at what is the type of the object the pointer is pointing to and call its function
  - this is a type of polymorphism
  - virtual for the parent.
  - b1 = d1

    cout << b1.add() ⟹ will look at the drieved value not actual one which is the child here
  - if I deleted virtual no error but will call parent function each time.
  - if the child don't have function override it will call parent function ⟹ no error



→ general function

→ function of pointer

```
Base* arr[3] = {&b1, &d1, &d2};
for (int i=0; i<3; i++)
{ cout << arr[i]->add(); }
```

OR

```
Base& arr[3] = {b1, d1, d2};
for (int i=0; i<3; i++)
{ cout << arr[i].add(); }
```

error arr decleration

- virtuality can't be stoped in C++ but other higher languages can
- abstract class must have at least one pure virtual class

  void add() =0
- if the child class don't have override of the parent virtual function it will be considered as abstract class ⇒ No objects can be created from it

- Base * ptr ; (valid)
  ptr = new Dev2();

  cout << ptr → add()

Base* ptr = new Base() ⟹ Compilation error Because of abstract

```
// without virtual
//static binding - early binding
Base* ptr;//=new Base();
ptr->add();
ptr=&d1;
// with virtual , pointer of parent -> child
//dynamic binding , late binding
```

during runtime

interview question what are the diffrences between static and dynamic binding?

Static binding (VS)	dynamic binding
early binding	late binding
before	during run time
calling parent function	child function

virtual function

virtual int fun() { return 0; }

1- not pure but will do nothing

2- Can create object from its class

→ since it is virtual if pointer or object will not be diffrent

→ if not virtual will return value zero.

virsual int fun() = 0;

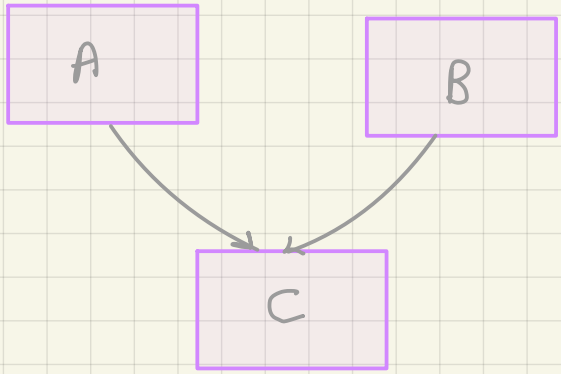
1 pure virtual function

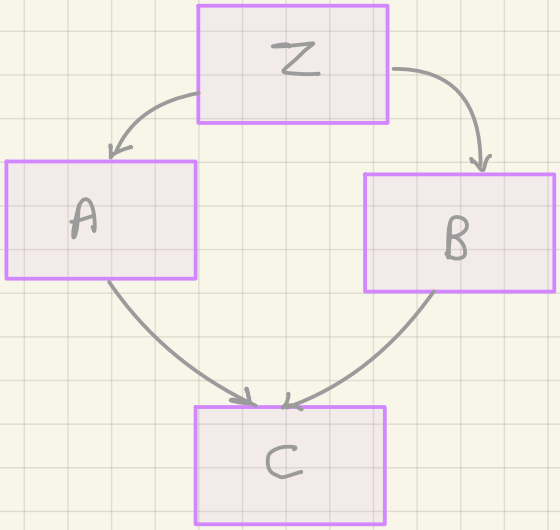
2 can not create objects from its function

↳ Compilation error

So to solve that create array pointer containing its childeren ⇒ shape *arr [3] = {c1, r1, t1};

multi level inheritance:- class C : Public A, Public B





multiple Inheritance had problems
So removed from other higher languages
(will get values of z once by A and
, z, y, c, another by B

what happen if virtual function is in one of the children not the parent

```
class Base
{
    int x;
    int y;
  Public:
    void print() {}
};
```

```
class A : public Base
{

}
```

A will inherit Both Private and public members as they are and inherit them to any child of them and main can see and access the public members

```
class B : protected Base
{

}
```

B will inherit Both Private and public members but Public members as protected which means inherit them to any child of them and main can not access the public members (now they are protected)

```
class C : private Base
{

}
```

B will inherit Both Private and public members but Public members as private which means can't inherit them to any child of them and main can not access the public members (now they are protected)

these access modifiers works mainly on Public members of the parent which affect children & main function (outside the inherited classes)