Oops

Sivaselvan @
CS24D0002 @

C++ Java

Tue 10
Thurs 11
Fri (lab) 2-5

Deital and Deital
Denis Richie

object oriented programming
↳ Sivaselvan

C - structured programming
Importance to function over data

C++ - object oriented
Importance to data over function

Prog. Paradigms
↳ structured - Imperative "C" (Functions)

Entity < attribute (data) / behaviour (Fns) > "OBJECT"

OOPS
↳ Data Encapsuln } (X)
↳ Data Abstraction }
↳ Op. Overloading
↳ Polymorphism
↳ Inheritance
↳ Templates

main() - Test Driver routine
"Boss" "

Before c++ they used function pointers in c struct

Interface V/S Implementation separation
↳ header file ↳ definition

< > - system defined header file
" " user defined header file

$ cd /usr/include

header file don't contain function definition if only contains function declaration

gcc -c filename.c → checks the syntax of the code without main()

gcc main.c function1.c .....

data + functions = class

↓

Software representation of an abstraction

↓

collection of attributes and behaviour

SRS - System requirement Specification

Data is known as variable in C setup

classes in C++ (Not only data but with function)

class faculty { → group of data and functio

char fname[100]; } data members

int fid;

void coursestaught(); } Member functions

void init();

};

Data encapsuln

g++ filename.cpp

class - doesn't exist in memory

↳ user-defined like struct but includes fns.

Object is a distinct instance of a given class

↓

real life existence

Like a creation of variable in that datatype

Data Encapsulation

Access specifiers

Private Public Protected

(open)

foss - Free and open source software

Linux invented by Linus

Latex -> preparing documents for engineering

#include <iostream.h> -> object way of input and output

cout << " ... \n" ; cout << a ; cout << a << b << c ;

-> ostream

operator cascading

cin

istream

\n - newline

\r - carriage return

<< - stream insertion

>> - stream extraction

private: int a;

private data or fn members cannot be accessed from outside the class

class Example

{ → inside the class

}; → outside the class

If not specified it will automatically private in class but public in C++ struct.

Data members → private } → recommended way
Function members → public }

If once stated public then, till changing to private, it will be considered as public.

:: Scope resolution operator.

```
Void EXAMPLE :: intialize (int val)
```

```
cout << "IIITDM\n";      } → Same
cout << "IIITDM" << endl; }
```

↳ output buffer will be flushed

```
Cin >> a;
Cin >> a >> b >> c;
```

operator cascading.

class is a like a blueprint

```
object.functionname ( )
```

↓ w.r.t object

```
int main( )
{
  return 0; → Sucessfull program termination.
}
```

If you define the function inside the class then it is known as "Inline".

If function contains <10 lines of code, Inline is preffered

* Function not declared in class known as Stand alone function.

```
main()
{
 EXAMPLE 01;
 01. data = 10;  X  -> Access specifier : Private
                       ∴ It cannot be accessed
```

int * a; a points some integer
int c = 80;
a = &c

int & const a; a constantly points some integer

Const int * a; a points to int which is constant

const int * const a; a constantly points to some constant int

Constructors & Destructors

Member functions

Constructor - A special member functions create an object of a class

```
TIME                          main ()
hour                          {
min                             TIME t;
secs                            t.settime (5,6,7)
settime (int, int, int)
```

* Name = className
* init object ("Malloc")

Destructor - Destructing the object, free the memory

If we use the dynamic memory allocation

* For constructor, destructors we can't have a return datatype.

```
class Time {
  int hour, min, sec;
Time ();  // Default constructor
Time (int, int, int);
  Parameterised constructor
```

No return datatype

~Time (); destructor

```
Time :: Time()
{
   hour=0;
   min=0;
   sec=0;
}
Time :: Time (int h, int m, int s)
{
   hour = (h>=0 && h<=23) ? h : 0;
   min  : (m>=0 && m<=59)
   sec  =        ..          ..
}

main()
{
   Time t1;
     t1.Time();

   Time t2(5,6,7);
     t2.Time(5,6,7);

Time :: ~Time();
```

if we didn't define, the compiler will automatically assign 0

~ → Tilda

Time() → destructor

Lab 3 class package Time, 24 to 12 conversion with constructor.

Date package : constructor, whether leap year or not check

Given Date ⇒ Day

Function Pointers

```
void bs ( int work[], int size, int (*compare)(int a, int b)
```

function pointer

Interesting code

```
int a=0
void main()
{
  printf ("Hello :%d \n", a);
   a++;
  if (a<3)
   main();
  p("world :%d \n", a); }
```

o/p

H
H
H
W
W
W

Storage class/scope

Storage classes is a lifetime in memory point of view

Auto static register external

Function scope, Global scope / file scope, Block scope

↓

Inside a for loop, if accessed in a block

Both discuss about lifetime of variable

If the nothing is mentioned then "auto" will be assigned

In "Auto", It will allocate memory each and every time the function called

"static" only once the memory allocation will be happening

Reg -> reserved in the system storage
-> minimal storage
-> Recommended for variable which has more usage
-> reg int i in for loop can be good example

extend -> this variable can be used in different file

-> Global scope is intra file whereas extend is inter file

Class Example {

private : int t;
public : Example (int=10)

↓
combination of default and parametrised constractor

* If this declared there should not be any default constructor

* If nothing assigned by user, the default value 10 be assigned. If we assign then assigned value will be assigned

* Parameterised constructor definition is needed.

if in main
Example O;
O. Example ()
↳ t will be assigned 10

if O. Example (20)
↳ t will be assigned 20

static, Auto, reg, extend can be also used in object.

* object in C++ will be destructed in reverse order of construction.

Example E(1)

```
main() {
  E(2)
  static E(3)
  function();
    E(4)
  }

function()
{
  E(5);
  static E(6);
  E(7);
}
```

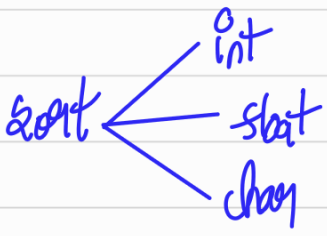
Cons 1
Cons 2
Cons 3
Cons 5
Cons 6
Cons 7
Des 7 (Auto in the function. When the function closed it will be destroyed)
Des 5

Cons 4
Des 4

Now 1, 2, 3, 6
global ↓ Static
Auto En main

Des 2 (static will live longer than auto)
Des 6
Des 3
Des 1 //.

Function Overloading




```
template <class T>

Void sort (T arr[])
{
  - - - .
}
Void main
{
   int iarr[10];
   float farr[10];
   char Chararr[10];
}
```

Class composition

Magic square < generation
checking

Latic square - all the

MACRO in C

```
# define SQ(X) X*X
M()
{
int a=5;
C=SQ(5)
```

SQ(5) will be replaced by X*X

```
d=SQ(a+1);
```

a+1 * a+1

a+a+1 = 2a+1

```
# define SQ(x) (x)*(x)
```

SQ(a+1)

6*6

=36//

emacs - linux editor -> recommended for programming

vi editor VIM

Class Composition

↳ including some other object of different class as a data member

```
class Emp {

    int id;
    char *Name;        Date class should be
    Date dob;  <- already made
    Emp(int, char *, int, int, int)
                        - Date
};
```

bool

function_name const;

↳ There can't be upgradation of variable in the function

Use emacs

P&P - Poinciple of least privelige.

Public member function - Interface
Private " " - Helper function

```
Emp :: Emp( int ei, char *N(50), int d, int m, int y)
        : dob( int d, int m, int y);
{
    if (ei > 0)
    id = ei;
    int length = strlen(N);
    strcpy(Name, N, length);
    Name(Length) = '\0';
}
```

Copy constructor

↳ copying the one constructor from one to another

```
emp e1;
emp e2(e1); -> copy constructor
```

e2 = e1 (Will work
Operation overloading.)
Not recommended

```
class STD
{
    int ssn;
    int age;
    float cgpa;
public:
    STD(int=0, int=0, float=0);   // combination of default and parameterized constructor
    STD(const STD&S);             // copy constructor
};

STD(const STD &S)
{
    ssn = S.ssn;
    age = S.age;
    cgpa = S.cgpa;
}

main()
{
    STD s1(1, 19, 9.4);
    STD s2(s1);
}
```

Arity - How many input does the operator work on

./a.out > filename (write mode)

./a.out >> filename (append mode)

return *this -> address of the object

```
func1
{
  :
  return *this
}
func 2
{
  :
  return *this
}
```

-> address of O1

O1.func1().func2()

O1.func1
O1.func2

friends < fns
class

```
class EX
{  int d;
friend void fn1 (EX &, int);
```

fn1 is friend of class and it is not a member. It is just a standalone function

```
};

void fn1( Example &e, int d)
{
  e.data = d; // this is allowed as result of
              friends feature
}
```

Operator Overloading - functions

```
main()
{
Emp e1, e2
```

[cout << a] << b

cout << b

cout << e user defined

Operator overloading achieved by member function (mostly)

cout << a
↓
operator (keyboard)

operator << (cout, a) -> this is how compiler understands
↓
this will return ostream &

Matrix 3 : Matrix 1 + Matrix 2

To do this we need operator overloading

Matrix1.Opt (Matrix2)

Opt( ) -> We can do it as member func in this case. Because both are same type

cout << M -> this should be as a non member function

friend ostream & operator << (ostream &, Emp &);

```
ostream & operator << (ostream &O, Empge)
{
    O << e.fn;
    O << e.sn;

    return O;
}
```

```
int a[10], b[10], c[10];
```

b = a → assign all then element of a to b
if (a > b) → check all the element

```
My_arr va(10);
```
→ arr[10];

```
My_arr ba(va)
```
→ copy constructor

```
My_arr ba;
ba = va
```
→ operator overloading

Both are of same definition

copy constructor copy it at the point of creation but operator overloading does it after the creation of object.

++a
a.++(int)

a++
a.++()

LC

Backtracking.

new ≈ malloc()
delete ≈ free()

int *aptr

aptr = new int [size] => malloc in C++
key. datatype. Size.

C++ is more type safe than C

class
{
static int a
}

shared to all object

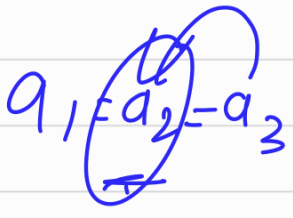
delete [] aptr => free() in C++

Assert
↳ input is the condition

assert (ptr != NULL) -> macro. pre processor directive
further lines of code will execute if the condition is true

Macro ≠ function call

if (right != this)
{ ↳ this will stop A = A



}
return *this

operator != (const Int Array &) const -> No assignment in the function
{
return ! (*this == right)
↓ ↓
not (operator == will be called)

operator << (. . .) const ×

↳ this function writes onto monitor, so const cannot be used.

& ! . ? :

↓

Not overloadable

Overloading of unary ++/-- operator

post increment A & operator ++(int)

```
class A {
  private:
    increment ();
    int n1;
    int n2;
  public
      A & operator ++ ()
      A & operator ++ (int)   post increment
```

```
A & operator ++() {
    increment();
    return *this;
}

A & operator ++(int) {
    A temp = *this;
    increment();
    return temp;
}
```

```
increment()
{
    n1 = n1+1;
    n2 = n2+1;
}
```

Inheritance

Reusability

- Polymorphism
- Class templates } STL → Standard template library
- Vectors
- Exception

Inheritance

```
class Emp
{
    fn
    ln
};
```

class faculty

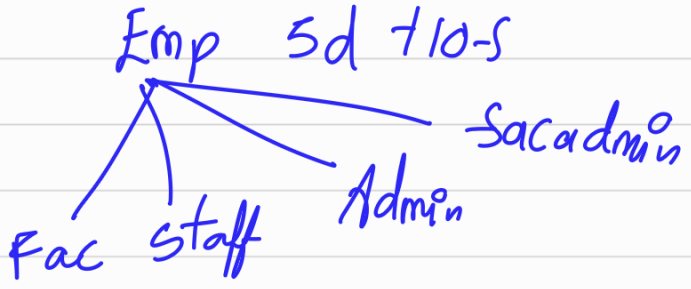
faculty is a employee

Class composition

```
class dob
{
};
```

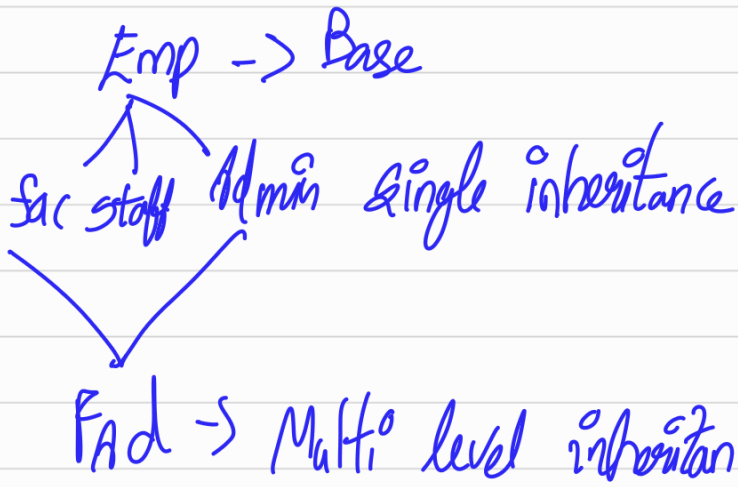
```
class Empdob
{
    ...
};
```

Emp contains dob



≥ 1 dm

Every member for parent will be inheritance, "there is no selective inheritance".



Two member function with same name and same signature is allowed in inheritance

Multiple level composition is not present in C++

iostream Multiple inheritance ✓

Multi level inheritance ✗

Abstract Base class
Virtual Base class

A

↓

B -> All the data members of A
now also the " " of B
+ atleast one data member

Overriding: The member function of A can be used with the object B

Static binding -> Inheritance
Dynamic binding -> Polymorphism (Powerful inheritance)
↳ OOS industry


```
class DerivedClassName : mode BaseclassName
                          ↳ for whole class
{
    derived class attributes (data members)
     "     "   properties (member function)
};
```

Public x Protected -> directly can be accessed by non

Student
↓
Graduate

```
Graduate::Graduate(char *fn, char *ln, char *dg)
                                    : Student(fn, ln)
{
    degree = char new(strlen(dg)+1);
    strcpy(degree, deg);
}

ostream & operator <<(ostream &O, Graduate &g)
{
}
```

↳ function overriding has even though base class, operator << but it can display deg field So. the same function with same signature is written again

```
Student *sptr = 0, s('first_name', 'last_name')
Graduate *gptr = 0, G('  "  ', '  "  ', 'deg')
```

sptr = &G → Property of inheritance It is allowed to do

Only the fn and ln of G will be available

because Graduate obj is mapped to Student *sptr. → Upcasting a pointer

- ↳ Safe, Symantically & syntatically ✓
- ↳ Derived class objects cannot exist without a base class

Printing it only give fname & lname.

Down casting → Syntatically ✓ Symatically ✗
Unsafe.

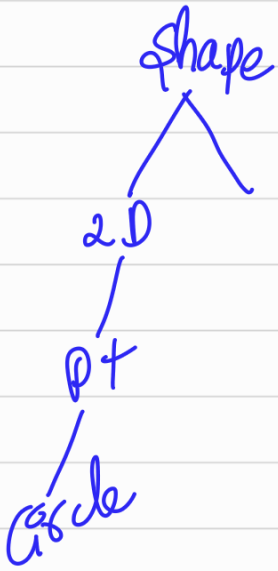
gptr = static_cast<Graduate*>(sptr);

Order of Constructor and Destructor

```
int main()
{
  Student S1("fname", "lname");
  Graduate G1("fname", "lname", "deg");
  {
    Student S2("", "");
  }
  Graduate G2("", "", "");
```

Base class const S1("", "")
" " Const G1("", "", "")
derived " " G2("", "")

Base class const S2 (" ", " ")
" " Dest S2 (" ", " ")
Base class const 6 2
Derived " " 6 2
Derived class destr 6 2
Base " " 6 2
Derived class destr 6 1
Base class " 6 1
Base class destr S1



Polymorphism
Virtual

Pointer => do data manipulation at its address

Multiple inheritance

```
Class A
{
  Protected:
     int val
  Public:
     const
     dest
     display

  Class B
     Protected:
       int b;
     Public:
          :
          :
          ;

Class C : Public A, Public B
{
  private:
     double id ;

  Public:

    C :: C(int i, char j, double k) : A(i), B(j)
    {  id = k;

    }
    ostream operator << (ostream &O, C &obj)
    {  O << obj.a;
       O << obj.b id; }
```

```
int main()
{
  A o1(50) *aptr=0;
  B o2('B') *bptr=0;
  C o3(10,'C',1.1);

o1.print() -> 50
o2.print() -> B
cout << o3; -> 10, C, 1.1

aptr = &o3;

aptr->print(); -> 10 -> Static binding
```

Virtual

```
class A
{
  Protected:
    int i;
};

class B : Public A
{
  Protected:
    int j
};

class C : Public A
{
  Protected:
    int k;
};
```

```
class D : public B, Public C
{
  Protected:
    int l
};
```

Both are derived from A
So, now the problem is both has int i from A
So, which i has to taken?

Error

```
class B: virtual public A        class C: virtual public B
{                                {
};                               };
```

```
class D: Public B, Public C
{
};
```

Now only i will be maintained in class D because of virtual

```
class Shape  -> Base class
{
public:  -> dynamic binding
    virtual double area() { return 0.0; } -> Inline function
    virtual double Volume()
    virtual void printShapeName() const = 0;
    virtual void print() const = 0;
```

↳ No definition in base class

Pure virtual function
No definition in abstract class

```
};
```

```
class point : public shape
{
  public:
        constructor
        destructor
        virtual void printshapeName() const {cout << "Point";}
                                  ↓
                          Now we are
                          defining it
                          so just a virtual
                          function
        virtual void print() const
  private:
        int x, y;
};
```

The class with atleast one Pure Virtual function will be considered as abstract class

```
class Circle : public point
{           point int r;
    constr
    destr.

    double area() const
    { return 3.14 * radius * radius; }

    double print()
    { cout << "Circle" }

};
```

class cylinder

```
double cylinder::Volume() const
{ return circle::area() * height; }

int main {
 Shape * arrayofShapes[5];
 arrayofShapes[0] = &point;
 arrayofShapes[1] = &circle;
 arrayofShapes[2] = &cylindere
 for(i=0; i<3; i++)
 {
    virtualViaPointer(arrayofShapes[i]);
 }
virtualViaPointer(const Shape *baseclassPtr)
{
   baseclassPtr->printShapeName();
   baseclassPtr->area();
   baseclassPtr->Volume();
}
```

object

o/p

Point [4, 11]
area 0.0
Volume 0.0

Circle [22, 8]: Radius = 35
Area 38.48
Volum 0.00

Cylinder [10, 10] Radius 3.30 Height 10.00
Area = 215.77
Volume = 342.12