Object - Oriented Programming (CS2000)

Faculty : Sivaselvan. B sir - sivaselvanb@
T.A. : cs24d0002@

LTPC : 2 0 4 4
↓
Tue (10 am)
Thurs (11 am)

OOP → C++ (85%)
OOP → Java (15%)

1 midsem (for theory & lab)

1 endsem (for theory & lab)

Theory-cum-practical course

better to do : a course project (do it as a 4-5 member team)
↓
build an app/software using C++/Java & OOPs concepts.

refer to Deitel & Deitel for first level intro to C-programming. Then go to Dennis Ritchie.

↳ 10th edition has OOPs. "How to program in C"
↓
start reading on your own.

C++ will be covered.

syntax - rules to implement logic.
semantics - logic/solution

C → structured programming paradigm. (imperative style)
main → user-defined function.

- programmers forced to think wrt functions.
- operations are behaviours. behaviours aren't unique. identity is.
  entities irl → attributes + behaviours (data)
- assigning meaning to data → information.

DATA ——comp. sci——→ INFORMATION

C → gives priority to functions rather than data.
C++ → OOP paradigm. more importance to data over functions.

Prog. Paradigms

① structure - Imperative ; action / function oriented.
like in C, users think in terms of functions.

entities → attributes → data.
entities → behaviour → functions

IRL, attributes / data resonate more.
↓
hence, the programming paradigms shift more

towards Objects / data ⟶ object oriented.
(we still use functions)

OOPs - Overpractices

1. data encapsulation
2. data abstraction.
3. polymorphism
4. Operating overloading.
5. Inheritance.
6. Templates

$\{data + fn\} \rightarrow$ OOPs.

main fn. ⟶ Also called as Test / Boss Driver Routine.
↳ fns. I develop can only be tested iff they are called in main.

fn
- declaration - return type, arguments & fn. name
- definition - what does it do with the args. to return something.

fn (int a, int b) ; fn (x, y)

formal param. actual param.

OOP ⟶ steps in for data security.
↳ enforce permissions for functions to operate ONLY on certain type of data.
e.g. ⟶ only faculties can access students' data.
↳ data encapsulation is done - cover both data & functions in one place (classes?)

C++ and not ++C ⟶ can run 'C' code also

x++ → assignment, then increment.

(incremented features will work, along with older version – "C")

C → building blocks of C++.

in C → we use structures for defining user-specific data types.

Can we have fns. in structures? wouldn't this encapsulate attr. & behaviours?

Not directly, but can use fn. pointers.

interface vs implementation

#include <____> ⟶ system defined header files.

#include "____" ⟶ user defined header files

interface | $ cd /usr/include → and ls to see the content.

header files only have fn. declaration (not defn.)

↳ header files help users focus only on interface. (unwanted info/ data filtered).

example ⟶ do rational no. operations.

→ create "rational.h"

→ declare functions here like ratadd, ratsubt, etc----

→ define functions like ratadd (struct rat a,

struct rat b);

in files like <function>.c

→ compile these files with:

gcc -c ratadd.c.

(or) do

gcc main.c ratadd.c ratmult.c

and so on...

Larger Idea → separate interface from implementation.
(header, fn. files, main file)

OOP ≡ group data & behaviour — into classes - whose instances are objects.

Class - named software repn. for abstraction

⇓

abstraction - named collection of data & behaviour relevant to modelling something for some purpose.

⇓

Object - distinct instance of a class - structurally identical to other instances of the class.

⇓

code to → define classes, instantiate objects & manipulate objects.

SRS → Software / System Required Specifications

↳ Spelling out requirements from a software.

↳ An english Text (like us giving to ChatGPT LOL)

↳ nouns ≡ data → CLASSES

Verbs ≡ functions.

```
class faculty
{
   char name[100];
   int fac_id;
   void coursestaught();
   void init();
}
```

→ probably an Andhra faculty - Sir, 11:35 AM, 20th Aug.

↳ DATA ENCAPSULATION.

DON'T define fns. in the class → keep it secretive
↳ ಮುಚ್ಚಿಟ್ಟಿರುವ ಕೆಲಸಗಳನ್ನು ಮಾಡುವುದಕ್ಕಾಗಿ / ಅಂತ

once the class is defined, u needn't tell "class" again while declaring an object, unlike structs. just say: Faculty f;

size of object = size of {data + fn.}

Classes → data members
Classes → member functions. (methods in Java)

member access specifiers → Private / Public.

DATA Encapsulation

Access Specifiers

* Private * Protected * Public

(Linux, e.g.) Free-end Open Source Software (FOSS)

(open access)

Linux : built on UNIX, by Linus.
development → best in Linux as its Open source

LaTeX → used to type formal scientific / engineering documents.
↳ FOSS, again. (Donald E Auth ?)

Windows (Ctrl+C Ctrl+V)'d Task Manager from Linux's Ctrl+Alt+Del. LOL.

Access specifiers → tell which data can be accessed by which fn. and so on...

output : { cout << " iiitdm \n " << endl;
{ cout << data;

OO. way of outputting.

cout → object of class ostream
cin → object of class istream
} in <iostream.h>

\n → positions the cursor to the home of next line.

\r → " " " " " " } → in Linux, if we do \r & type more, chars will be overwritten.
" current line.

cout << a << b << c ;
↳ stream direction in C++.

{operational overloading ?}

(usually left shift, but here this!!)

```
class Example
{
   private: int data;
   public : void initialize (int val);
   void increment ();
};
```

can't access outside the class ↑

int data member is private

→ assigns val to data part of object.

```
void Example :: initialize (int val);
{
   if (val >=1)
   {
     data = val;
   }
   else
   {
     cout << "Invalid" >> endl;
   }
}
```

↳ member functions public
↳ data members private.

- Scope resolution operator.
- define member fn. to class Example, OUTSIDE the class.

↳ OO way of ending line

```
void Example :: increment ()
{
  data += 1;
  cout << "Incremented Val is: " << data << endl;
}
```

```
int main ()
{
   Example O1;
   O1.initialize ();
   O1.increment ();
```

obj.fn()
↳ call fn. wrt obj.

x needs to be output, so

```
    return 0;
}
```

push from x, to cout.

cout << x;

cin >> a;

input needs to be saved to var a, so push from cin to a.

./a.out → CALLS the main fn.

CONSTRUCTORS / DESTRUCTORS

↳ major part of data - encapsulation.
↳ they are ALSO member functions.

C++ Class → members are private by default.
C++ Structure → members are public by default.

member functions defined in class: INLINE.

if not INLINE, control transfer takes place to a diff world. we pass arguments to that world & get the return value back from it. (internally, a stack works) → This is better, as the function defn. DOESN'T get pasted in main → in cases of BIG defns.

if fn. defns. are BIG ⟶ not INLINE.
" " " are SMALL ⟶ INLINE.

```
main ( )
{
   Example 01;
   01. data = 10; ⟶ WON'T WORK . . . .
}
```

↓ (data is private, & it's not safe)

BETTER define fns.

What can be done: define member fns. that'll get obj. data & return its location or something? (Will it work?)

```
int *a;
int c = 80;
  a = &c;
```

pointers → to manipulate data directly at its location.

const → ALSO a secure access.

int * a; → Assign many times

int * const a; → Assign only once.

const int * a; → Assign many times only to const ints.

const int * const a; Assign only once, only to const int.

(read from R → L)

(ADA used in ISRO)

MOST SECURE! EXPLOIT in STANDALONE fns.

CONSTRUCTORS / DESTRUCTORS

* Are also member functions.
* constructors → create / initialize an object of a class : assign values to data members. (Malloc will be done here, if dynamic members are present)
* destructors → free the memory used after end of program.

* name of the constructor / destructor = SAME as the className.
* default constructors & destructors are available, but it's better to define on our own.
  ↓
  - doesn't need arguments to pass!
  - WE can't specify a return Type for them.

```
Class Test()
{
  private: int a;
  private: float b;

  public: Test();
  public: test(int a, float b);
  public: display();
}
```

```
#include "test.h"
```

```
Test :: Test()
{
    a=0;
    b=0.0;
}
```

→ * default
* no return Type

```
Test :: test(int v1, float v2)
{
    a = ((v1>=0) ? v1 : 0);
    b = ((v2>=0) ? v2 : 0);
}
```

→ * parameterised

ternary :
* $e_1 \,?\, e_2 : e_3$
* eval e_1. if true, eval e_2 & return result & assign. else, eval e_3 & return result & assign.

Time t1; }

This AUTOMATICALLY calls the constructor & initializes t1.
[t1.Time() is done automatically - even if it has mAlloc & stuff]

Time t2.Time(5,6,7)

↳ compiler knows this is parameterized, hence user defined constructor.

function differentiated by
- parameters.
- name.

everything told for constructors holds good for destructors → but declare destructors using:

~className();

↳ negated operation of constructor.

① create Time Package – with constructors / destructors · check for data validity · convert into AM/PM format · do formatting properly.

② similarly create a date package · given a date, find day.

FUNCTION POINTERS

* bubbleSort in ascending & descending are the same, EXCEPT for the comparison! so, have a generic bubbleSort fn, & write comparison logic separately.

```
void bubbleSort (int work[], int size,
                 int (*compare)(int a, int b));
int ascending (int a, int b);
int descending (int a, int b);

scanf ("%d", order);

if (order == 1)
    bubbleSort (a, size, ascending);
```

↓

this will return truth value of arr[i], arr[i+1].

if you NEED ascending, then if arr[i+1] > arr[i], swap ISNT reqd, so return 0.

else, SWAP is reqd, so return 1.

if DESCENDING, exact opp. logic.

(3) run fn. ptr. logic for bubblesort.

(4) write fn. ptr logic for ANY one fn. in number operation package, IN C.

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

→ run & test

```
Hello 0
Hello 1
Hello 2
World 3
World 3
World 3
```

STORAGE CLASS / SCOPE

* auto, static, register, external
* function Scope, file Scope, global Scope, block Scope.

* Scope ≡ Where all in the source code can I access a variable is Scope. lifetime wrt code.

* Storage Class ≡ how long will a variable be accessible, from a MEMORY pt. of view.

```
int a = 1;
main() {
  int a=5;
  printf("%d\n", a);  ⟶ 5
  {
    int a=6;
    printf("%d\n",a);  ⟶ 8
  }
  printf("%d\n", a);  ⟶ 5
  fn1();  →  (6,7)
  fn2();  →  (40,41)
  fn3();  →  (1,20)
  fn1();  →  (6,7)
  fn2();  →  (41,42)
  fn3();  →  (20,400)
  return 0;
}
fn1() {
  int a = 6;
  printf("%d \n", a);
  a++;
```

```
    printf("%d\n", a);
}
fn2(){
    static int a = 40;
    printf("%d\n", a);
    a++;
    printf("%d\n", a);
}
fn3(){
    printf("%d\n", a);
    a *= 20;
    printf("%d\n", a);
}
```

register → faster memory access, but restricted in storage
→ use loop variables : as access is quick
→ if registers run out, data converted to auto.

extern → inter file variable access.

Bhadra

CLASS - COMPOSITION

↳ can have an instance of another class as its data-member.

```
Example :: Example(void) {
    if (val >= 0) {
        i = val;
```

Example
↓

```
}
else {
    cout << "Invalid" << endl;
}
```

```
class Employee {
private:
    int Aadhar;
    char* Fname;
    char* Lname;
    Date dob;
    Time tob;
}
```

```
Example :: ~Example (void) {
    cout << "Destructor called" << endl;
}
```

functions are allowed to default their variables to some values.

Example (int=0)

↓

both default & parameterised constructors.

We can't do

Example (int=0)

≠

Example ()

```
Example One (1); ← 1
int main () {
    Example two(2); ← 2
    static Example three(3); ← 3
    standalone fn();
    Example four(4); ← 7
    return 0;
}
```

```
void standalonefn() {
    Example five(5); ← 4
    Static Example six(6); ← 5
    Example seven(7); ← 6
}
```

democratic INDIAN programm-ers can dck off.

- if you default, default values to ALL Arguments.

else, dont.

- if you default, DEFINE a parameterised constructor...

01, 02, 03, 05, 06, 07, 07, 05,
04, 04, 06, 03, 02, 01.

↓

live longer.

04, 02, 01, 06, 03 → ACTUAL ORDER.

- execute storage classes & scopes for obj creation & destruction programs.

FUNCTION OVERLOADING

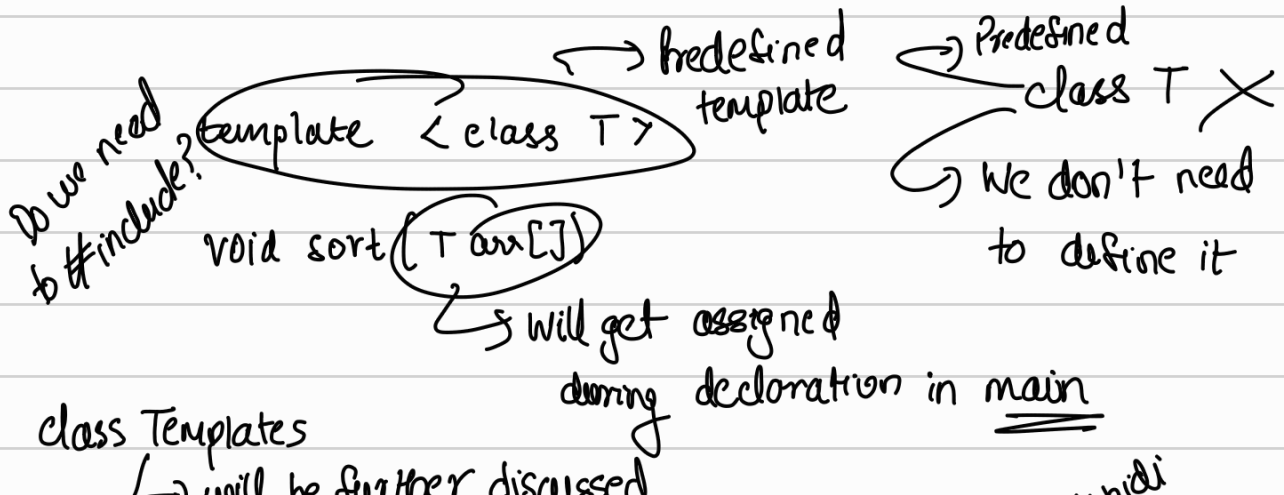
↳ MORE THAN 1 definition for the same function.
(e.g. same sort fn., but with diff. parameters)
↳ diff. signatures.

↳ execute fn. overloading to sort arr of chars, ints & floats. → once with 3 fns, & once with template.

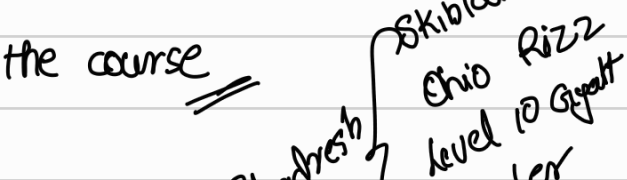
↳ check magic square.

↳ Optional: generate a magic square for a given value of n (nxn)

↳ Latin Square: generate permutations



class Templates
↳ will be further discussed towards the ending of the course



#DEFINE SQ(x) x*x → MACRO.

→ STRING REPLACEMENT - - - -

↳ NOT FUNCTIONS, but like functions.

```
m() {
  int a=5;
  SQ(a);
}
```

↳ if I put a+1,

a+1 * a+1 = 2a+1 happens.

COMPOSITION

use this...

editors: • emacs
• VI editor (VIM)

```
Class Date {

private:
        int month;
        int day;
        int year;
        int check_day(int);
public :
        Date(int=1, int=1, int=2003);
        Void display() const;
};
```

C++ → has bool data-type (boolean)
↳ C doesn't have it.

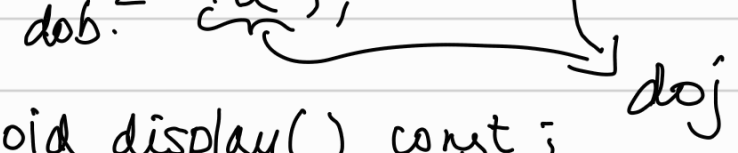
↳ gives only READ access to data members.

```
class Employee {
```

follow:

```
private:
        char fname[25];
        char lname[25];
        const Date dob;
        const Date doj;

public:
        Employee(char*, char*,
                 int, int, int, int, int,
                 int);
        void display() const;
        ~Employee();
};
```



principle of least priviledge (PLP)

public fns.
↳ interface.
private fns.
↳ helper fns.

```
Employee::Employee(char *fn, char *ln, int bm,
                   int bd, int by, int jm, int jd,
                   int jy): Birth_Date(bm, bd, by),
                            Join_Date(jm, jd, jy)
{
    int length = strlen(fn);
    strncpy(fname, fn, length);
    fn[length] = '\0';
        length = strlen(ln);
    strncpy(lname, ln, length);
    ln[length] = '\0';
}

void Employee::display(){

    cout << fname << " " << lname;
    id.print();
```

"%[^\n]s"

```
    bd. print();
    cout << endl;
}
```

COPY CONSTRUCTOR

```
Employee e1;
Employee e2(e1);
```

→ Copy e1's state into e2.

→ CALLS copy constructor.
(series of values to be assigned)

→ in single go, create & copy state of another object into this.

```
Class Student { int ssn;
                int age;
                float cgpa;
```

Syntax of cpyconstr: `Student (const Student S)`

```
Student s1(x, y, z);
Student s2(s1)
```

→ inside fn:

```
ssn = s.ssn;
age = s.age;
cgpa = s.cgpa;
```

pass by value → operates on the copy of an

actual value
→ ensures data security.
→ wastage of memory.

pass by reference → operates directly on the actual value (usage of pointers)
→ data - vulnerable.
→ No wastage.

SAFEST & BEST → const int * const e
↳ passed by reference
↳ const ptr.
↳ ptng. to const int.

FOR COPY CONSTRUCTOR, we can't do PASS BY VALUE, as PASS BY VALUE creates a copy of the value of a certain datatype, but we are LITERALLY JUST creating the copy fn. for our datatype (TWIST!!!)

defn → Student (Student * const s)
call → Student (&s)

OPERATOR OVERLOADING

unary, binary & ternary.

binary operators.

ARITY - no of operands does the operator work on.

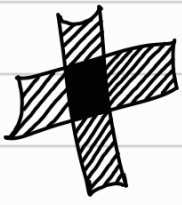
$ ./a.out >> op → push output into the file, rather than displaying on the screen. (this will append) → SAFER.

(out-direction)

Why not ./a.out > op ? → this REWRITES from the first bit.

(in-direction) (write)

./a.out << ip → input the params. from a file into the executable during runtime.



I want to do cout << e, where e is probably an instance of some class with data members.

↓ it should display all the data members.

currently, it will show SYNTAX ERROR (as the class might have ≥ 1 data-member)

↳ ASSIGN a NEW MEANING to the << operator s.t. it will display all the data members.

|| if A & B → 2 matrices, A + B SHOULD do $A_{ij} + B_{ij}$ $\forall i<n$ & $\forall i<m$ ||

```
class Test {

    private:
            int a;
            int b;
    public:
            Test (int =0, int =0);
            void setValue (int, int);
            int getVal();
            int &setValue (int);
};

Test :: Test (int V1, int V2) {
    a = V1;
    b = V2;
}

int &setValue (int V3) {
    a = V3;
    return a;
}

int main() {

    Test 01;
    int &aref = 01.setValue (50);
    aref = 30; //undesired modif.
    01.setValue(50) = 70;
    return 0;
}
```

returned reference used as lvalue

```
Test {
    mem1, 2, 3;
  &set1();
  &set2();
  &set3();
  &setall();
}
```

```
Test & setall(){
    set1(v1);
    set2(v2);
    set3(v3);
    return *this;
}
```

```
Test & set1(){
    mem1 = v1;
    return *this;
}
```

```
Test & set2(){
    mem2 = v2;
    return *this;
}
```

```
Test & set3(){
    mem3 = v3;
    return *this;
}
```

in main,

$o1.set1(v_1).set2(v_2).set3(v_3)$

(1)

$\Rightarrow$

$o1'.set2(v_2).set3(v_3)$

(2)

$\Rightarrow$

$o_1''.set3(v_3)$

(3)

$\Rightarrow$

o_1^m

o_1^m $\equiv$ location of o1 with the latest changes

CASCADING WAY.
(Better Approach than

setall)

$m_4 = \underbrace{m_1 + m_2}_{①} + m_3$ { m_is → matrices }

= $\underbrace{m_{12} + m_3}_{②}$ returns address of resultant matrix

$m_4 = m_{123}$ → returns address of final matrix.

OPERATOR OVERLOADING X THIS *

FRIEND → functions
FRIEND → classes

```
class Example {
    friend void fn1 (Example &, int);
    void display(){
      cout << value;
    }
    Example () {
       data = 0;
    }
    private:
          int data;
};

void fn1(Example &e, int d) {
    e.data = d;
```

ALLOWED AS A RESULT of friends feature.

`cout << a << b` → `cout << b`

ALSO CASCADING EFFECT!

cout is an object of class ostream

cout available for b, AFTER a.

OPERATOR OVERLOADING is acheived by functions:

↳ primarily by MEMBER functions.

↳ if not, make the standAlone fn. a friend of the class.

HOW DO I ACHEIVE addn. of 2 matrices m_1 & m_2, and return result to m_3?

if LHS of operand is of the same class of reference, then use member fns. else, non-member fns.

EXAMPLE:

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

- friend → allow non-member fn. to access private data members
- ostream& → for cascading
- operator → keyword
- << → the operator
- (ostream&, Emp&) → operating object (cout), operand object
- Emp& → input object's data members
- ostream& → to output data members.

member fns.
to operate on
private data members

definition:

```
ostream & operator << (ostream & O, Emp & e) {
    O << e.fn << endl;
    O << e.ln << endl;
    return O;
}
```

CASCADING STILL WORKS for O — as its predefined .....

COPY CONSTRUCTOR

↓

similar to obj1 = obj2

↓

but cpy happens in same line of construction

if A → array,
will ++A work? yes, if in defn. we add a dummy int in the LHS of ++. Duh.

```
class IntArray {
friend ostream & operator << (ostream &, IntArray &);
friend istream & operator >> (istream &, IntArray &);
public:
```

```
IntArray(int = 20);
IntArray(const IntArray &);
~IntArray();
const IntArray & operator = (const IntArray &);
bool operator == (const IntArray &) const;
bool operator != (const IntArray &) const;
int & operator [] (int);
const int & operator [] (int) const;
```

[] ⟶ array indexing operator.