06 May
Wednesday

2015
Week 19th Day

Important

8.00

9.00

10.00

11.00

12.00

1.00

2.00

3.00

4.00

5.00

6.00

OOPS

MAY 2015

Monday		4	11	18	25
Tuesday		5	12	19	26
Wednesday		6	13	20	27
Thursday		7	14	21	28
Friday	1	8	15	22	29
Saturday	2	9	16	23	30
Sunday	3	10	17	24	31

Important

Object Oriented programming :-

The main aim of OOPs is to bind together data and functions that operate on them so that no other part of the code can access this data except this fcn.

Class :-

It is user defined data type which holds its own data members and member functions, which can be used or accessed by creating instance of that class

Eg:- human is class ; body parts are its data ; walking is fcn.

Syntax :-

```
class student {
    public:
        int id ; int mobile; string name; // data member
        int add(int x, int y){
            return x+y;          // member fcn
        }
};
```

Object :-

When a class is defined, no memory is allocated but when it is instanced (ie object is created), then memory allocated.

It can operate on both data members and member fcns.

Syntax :-

```
student s = new student();
```

When an object is created using new, then space is allocated for variable in heap and starting address is stored in stack memory. When object is created without new keyword, space is not allocated in heap and object contains null value in stack.

10 May 2015
Sunday 2 Week 19th Day
Important

Encapsulation :
It is the process of combining data and fcns into a single unit called class. Here, the data is not accessed directly; it is accessed through member fcns.
The attributes of class are kept private and public getter and setter fcns are provided to manipulate this data. Thus encapsulation makes data hiding possible ie (protected, private)

Abstraction -
It means displaying only essential information and hiding unnecessary details. With definition of properties of problem including data which are affected and operations which are identified, the model abstracted from problems can be a standard soln to this type of problems.
- Abstraction using classes
- Abstraction using header files (math.h → pow())

Polymorphism :
It is the ability to present same interface for different forms (data types)

Types of polymorphism :
1. Compile Time :- It is implemented at compile time (Static)
a. Function Overloading :- Two or more functions with same name but different parameters. It can be possible on fol. :
- return type of overloaded fcn
- type of parameter passed.
- no. of parameters

Example :

```
class Add {
public:
    int add(int a, int b) {
        return a+b; }
    int add(int a, int b, int c) {
        return a+b+c; }
};
int main() {
    Add obj;
    int res1 = obj.add(2,3); int res2 = obj.add(2,3,4);
    return 0;
}
```

b) Operator overloading : A standard operator can be redefined so that it has different meaning when applied to instance of class.

Example :

```
class Minus
{ private: int a,b,c;
public:
    Minus() {}
    Minus(int A, int B, int C){
        a=A; b=B; c=C;}
    void display(){
        cout << a << b << c; }
    Minus operator - (){
        Minus temp;
        temp.a = -a ; temp.b=-b ; temp.c=-c; }
        return temp; }
};
void main(){
    Minus M1(5,10,-15);
    M1.display; Minus M2 = -M1;
    M2.display;     // -5, -10, 15
```

Using
- unary op.
- binary op
- binary using friend fun

:: . .* sizeof ?: cannot be overloaded

[], (), → cannot be overloaded

12 May 2015
Tuesday 4 Week 20th Day
Important

2. Runtime: Also called dynamic polymorphism
a. Function overriding: It occurs when derived class has a definition of one or more members of base class. It means that child class contains fun which is already in parent class. Hence, child overrides parent.
Here, child and parent both contain same fun with different definition. The call to fun is determined at runtime, thus called runtime polymorphism.
Example:

```
class base{
    public:
        virtual void show(){
            cout<<"Base"<<endl;}
};
class derived: public base{
    public:
        void show(){
            cout<<"Derived"<<endl;}
};
int main(){
    base *b;
    derived d;
    b=&d;
    b->show(); // print content of show() declared in derived.
    return 0;
}
```

Output: Derived

Function overriding is the redefinition of base class fun in its derived class, with same return type & same parameters

class car{};
class sports : public car{};

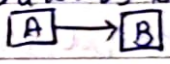
2015
Week 20th Day 133rd
Important

May
Wednesday 13

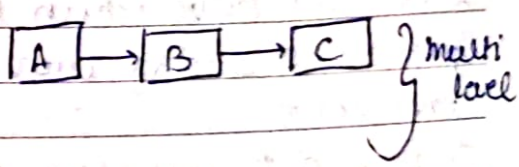
Inheritance :-
The process capacity of the class to derive properties and characteristics from another class ie parent class automatically
Subclass, Superclass, Reusability

Syntax :-
class derived :: visibility mode base_class;
visibility_modes = {private, protected, public}

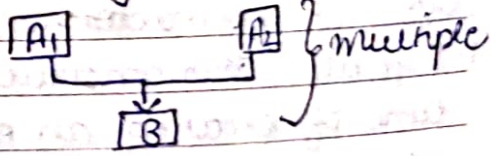
Types of inheritance :-
1. Single inheritance : When one class inherits another class.
class B : public A {};



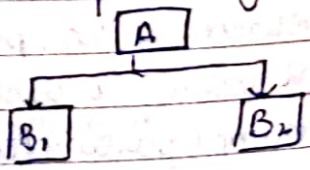
2. Multiple inheritance : It is process of deriving new class that inherits attributes from two or more classes.
class B : public A {};
class C : public B {};



3. Multilevel inheritance : It is process of deriving class from another derived class.
class A1{}; class A2{};
class B : public A1, public A2



4. Hierarchical inheritance : It is the process of deriving more than one class from base class.
class B1 : public A {};
class B2 : public A {};



5. Hybrid inheritance : It is combination of multiple, multiple and hierarchical inheritance.

JUNE 2015

Monday	1	8	15	22	29	
Tuesday	2	9	16	23	30	
Wednesday	3	10	17	24		
Thursday	4	11	18	25		
Friday	5	12	19	26		
Saturday	6	13	20	27		
Sunday	7	14	21	28		

OCT NOV DEC JUN

Dynamic Binding:
The code to be executed in response to function call is decided at run time.

Data Binding:
Binding application UI and business logic: Any change made in business logic will reflect directly to app UI.

Constructors:-
It is member function of a class which initialises objects of a class. In C++, it is automatically called when the object creates.
It has same name as class itself.
It doesn't have return type.
If we do not specify, then C++ compiler generates default constructor.
It is used to initialise data members of new object generally.

Types of Constructors:
1. Default: This constructor has no argument. It is invoked at time of creating an object.
class_name() { a=10; b=20; }

2. Parameterised: It has parameters. It is used to provide different values to distinct objects.
class_name(int x, int y) { a=x; b=y; }

Monday		4	11	18	25
Tuesday		5	12	19	26
Wednesday		6	13	20	27
Thursday		7	14	21	28
Friday	1	8	15	22	29
Saturday	2	9	16	23	30
Sunday	3	10	17	24	31

3. Copy Constructor: It is an overloaded constr. used to declare and initialise an object from another object. It is of two types:

2015
Week 20th Day 135th
May
Friday 15
Important

default copy and used defined copy constructor.
class_name (const class_name &obj) { a = obj. a ; b = obj. b ; }

EXAMPLE :-

```
class go {
        public: int x;
        go(int a) { x = a } // parameterised constructor
        go(go &i) { x = i.x } // copy constructor
};
int main() {
        go a1(20);  // calling parameterised
        go a2(a1);  // calling copy constructor
        cout << a2.x << endl;
}
```

Output :- 20

Destructors :-

It works opposite to constructor ; it destructs objects of classes
It can be defined only once in class.
It is invoked automatically.
It has same name as class, prefixed with tilde sign (~).
Derived class destructor will be invoked first, then base class destructor will be invoked

```
class A {
    public:
        A() { cout << " Constructor " << endl; }
        ~A() { cout << " Destructor " << endl; }
};
int main() {
        A a;
        A b;
        return 0;
}
```

Output :- Constructor
Constructor
Destructor
Destructor

JUNE 2015
Monday Wednesday Friday Sunday
1 8 15 22 29
2 9 16 23 30
3 10 17 24
4 11 18 25
5 12 19 26
6 13 20 27
7 14 21 28

OCT NOV DEC JUN

16 May 2015
Saturday
Week 20th Day 136

Important

'this' pointer :-

'this' refers to current instance of class. 3 main uses of 'this' keyword :

- It is used to pass current object as parameter to another fun
- It can be used to refer to current class instance variable
- It can be used to declare indexers.

Every object in C++ has access to its own address through an important pointer called 'this'.

```
struct node{
    private:
        int data;
        node * next;
    public:
        node(int x){
            this → data = x;
            this → next = NULL;
        }
};
```

Friend function :-

It can access private and protected members of another class in which it is declared as friend.

It is not the member of the class but must be listed in class definition. It is a non member fun that accesses private data.

It cannot access private members directly, it has to use an object name and dot operator with each member name.

Friend fun uses objects as arguments

There can be friend class and friend fun.

```
class A{
    int a=2; int b=4
    public:
        friend int mul(A k){
            return (k.a*k.b);
        }
};

int main(){
    A obj;
    int res = mul(obj);
    cout<<res
    return 0;
}
```

Output = 8

MAY 2015

Monday		4	11	18	25	
Tuesday		5	12	19	26	
Wednesday		6	13	20	27	
Thursday		7				
Friday	1	8	15	22	29	
Saturday	2	9	16	23		
Sunday	3	10	17	24	31	

2015

Week 20th Day 137th

May 17, Sunday

Important

Aggregation :

It is the process in which one class defined another class as any entity reference. It is another way to reuse class. It is a form of association that represents HAS-A relationship

Virtual Function :-

It is member fcn which is declared with virtual keyword in the base class and redeclared (overriden) in derived class.

When you refer to an object of derived class using pointer to a base class, you can call a virtual fcn of that object and execute the derived class's version of fcn.

- They help to achieve run time polymorphism.
- They cannot be static and can't be fcn of another class.
- A class may have virtual destructor but not virtual constructor
- When fcn is made virtual, then C++ determines at run-time which fcn is to be called based on type of object pointed by base class pointer. Thus, by making base class pointer to point to different objects, we can execute different versions of virtual fcns.

```
class base {
    public:
        virtual void print() { cout << "base class" << endl; }
        void show() { cout << "show base" << endl; }
};
class derived : public base {
    public:
        void print() { cout << "derived" << endl; }
        void show() { cout << "show derived" << endl; }
};
int main() {
    base *bptr; derived d; bptr = &d;
    bptr -> print(); // virtual binded at runtime
    bptr -> show(); // non virtual at compile
```

Output :-
derived
show base

Monday 1 8 15 22 29
Tuesday 2 9 16 23 30
Wednesday 3 10 17 24
Thursday 4 11 18 25
Friday 5 12 19 26
Saturday 6 13 20 27
Sunday 7 14 21 28

OCT NOV DEC JUN

18 May Monday 2015

Important

Run time binding (Late Binding) vs Compile time (Early)
As during compile time bptr behaviour judged on basis of which class it belongs, so bptr represents base class.
If the fun is not virtual, then it allows binding at compiler time and print fun of base class will get binded because bptr represents base class.
But at run time bptr points to object of class derived, so it will bind fun of derived at run time.

Pure virtual function :-
A pure virtual fun in C++ is a virtual fun for which we do not have any implementation, we only declare it.
A class containing pure virtual fun cannot be used to declare object of its own, such class is called abstract base class.
The main objective of base class is to provide traits to derived class and to create baseptr used for achieving runtime polymorphism

```
class base{
    public:
        virtual void show() = 0;
};
class derived : public base{
    public:
        void show(){ cout << "You see me!" << endl; }
};
int main(){
    base * bptr; derived d;
    bptr = &d;
    bptr -> show();
    return 0;
}
```

Output :-
You see me!

2015

May 19
Tuesday

Abstract class:

A class is made abstract by declaring atleast one of its func as a pure virtual func. A pure virtual func is specified by placing "=0" in its declaration.

- Its implementation must be provided by derived classes.
- We cannot declare object of abstract class.
  Eg: Base b; will show error.
- We can have one ptr or reference of abstract class.
- We can access the other func except virtual by object of its own derived class.
- If we don't override pure virtual func in derived class, then it becomes abstract.
- It can have constructors.

```
//abstract class
class shape {
    public:
        virtual void draw() = 0;
};
class rectangle : shape {
    public:
        void draw() { cout << "rect" << endl; }
};
class square : shape {
    public:
        void draw() { cout << "sq." << endl; }
};
int main() {
    rectangle rec; square sq;
    rec.draw(); sq.draw();
    return 0;
}
```

Output:
rect
sq.

Namespaces in C++ :

- It is logical division of code which is designed to stop naming conflict. It defines scope where identifiers like variables, class, fxns are declared.
- Its main purpose is to remove ambiguity which occurs when a diff. task happens with same name.
- C++ consists of standard namespace ie std which contains inbuilt classes and fxns.

EXAMPLE :

```
// user defined namespace
namespace Add{
    int a=5, b=5;
    int add(){
        return (a+b);
    }
}
int main()
    int res = Add::add(); //accessing fxn inside namespace
    cout<<res;
}
```

OUTPUT = 10

Access specifiers :-

They specify how fxns and variable can be accessed outside class.

Types :-

1. Private : Can only be accessed within class not outside class
2. Public : They can be accessed from any class.
3. Protected : It is inaccessible outside class but only accessible to subclass and that class. It is used in inheritance.

If we don't specify access modifier then by default it is private for members.

important

Few Terms:-

- Delete: delete is used to release unit of memory, delete[] is used to release array.

Virtual inheritance: It allows you to create only one copy of each object even if the object appears more than once in hierarchy

Overloading & Overriding: Overloading is static binding. The same fun with different argument and may or maynot return same value in same class itself. WHEREAS Overriding is dynamic binding. It is same fun name with same arguments and return type associated with class and its child class.

Advantages of data abstraction
- Avoids code duplication and increases reusability.
- Can change internal implementation of class independently

Structure vs Class
Main diff. is security:
A structure is not secure and cannot hide its member fun and variable while class is secure and can hide its programming and designing details.

Local classes in C++:
A class declared inside fun becomes local to that fun.
All funs of local class must be defined inside class only.

Exception handling in C++:
try: represents block of code that can throw exception.

22 May 2015
Friday Week 21st Day 142
Important

catch: Relevant block of code that gets executed when error is thrown.
throw: used to thrown an exception
There is a special catch block :→ catch(...)
It catches all types of errors.

Inline Function :-
It is a request not command.
It is a fun that is expanded in line when it is called.
When the inline fun is called, whole code gets inserted or substituted at the point of its fun call.
inline returntype func() { }

C	C++
• It supports procedural prog.	C++ is hybrid lang. because it supports both proced. & OOP.
• It doesn't support OOPs so no support for polymorphism, encap., inheritance	It supports them.
• C is subset of C++	It is superset of C.
• It contains 32 keywords.	It contains 52 keywords.
• It is fun driven language	It is Object driven lang.
• Function & operator overloading is not supported.	C++ supports them
• No exception handling supported	It supports exception handling

Structure is collection of dissimilar elements.

2015 May
Week 21st | 143rd Saturday 23

Static members in C++ :-

1. Static variable in fun :- When a variable is declared as static, space for it gets allocated for the lifetime of the program. (default initialised to 0). Even if the function is called multiple times, the space for it is allocated once

2. Static variable in a class :- Declared inside class body. Also known as class member variable. They must be defined outside class. Static variable doesn't belong to any object, but to whole class. There will be only one copy of static member variable for whole class.

Eg:

```
class account {
    private:
        int balance;
        static float roi;
    public:
        void setbalance(int b){ balance = b;}
// initialised outside class
float account :: roi = 3.5f;
void main(){
    Account a1,
}
```

Object can also be declared as static

```
static account a1;
```

Compiler generates 2 constructors by default :-

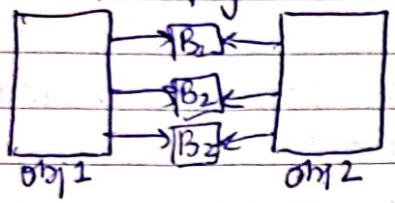
1. Default
2. Copy

But if any of the constructor is created by user, then default constructor will not be created by compiler.

8.00 Constructor overloading can be done like fun overloading

Default (Compiler's) copy constructor can be done only shallow copy.

10.00



(points to same memory location)

11.00

Deep copy is possible only with user defined constructor.

12.00 In user defined copy constructor, we make sure that ptrs of copied object points to new memory location.

1.00

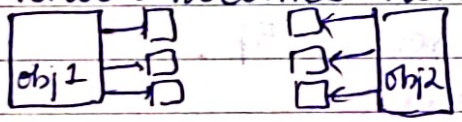
Can you make copy constructor private? Yes

2.00

Why argument to copy constructor must be passed as a reference?

3.00 Because if we pass value, then it would make to call copy constructor which becomes non-terminating.

4.00



deep copy

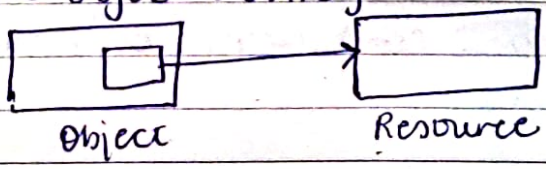
5.00

6.00 Destructor :-

Destructors take no argument and have any return type.

Destructor cannot be static.

Actually destructor doesn't destroy objects; it is last fun that is invoked before object destroy.

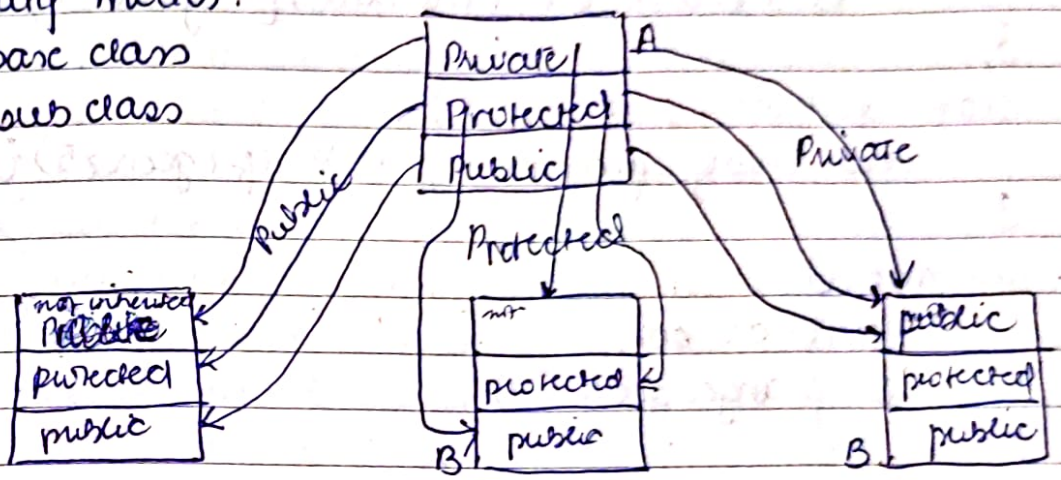


MAY
Monday 4 11 18 25
Tuesday 5 12 19 26
Wednesday 6 13 20 27
[illegible]day 7 14 21 28
1 8 15 22 29
2 9 16 23 30
3 10 17 24 31

Important

It is used so that before deletion of obj we can free space allocated for this resource. Because if obj gets deleted then space allocated for obj will be free but resource doesn't

Visibility modes:-
A = base class
B = sub class



If B is subclass and visibility mode is public, then public member of A will be public in B and protected will be protected.

Is a relationship:
It is always implemented as public inheritance

Constructor and Destructor in inheritance
First child class constructor will run during creation of object of child class, but as soon as obj is created child class constructor run and it will call constructor of its parent class and after the execution of parent class constructors it will resume its constructor execution

B() : A()
child parent constructor call

constructor exec
obj
child const
parent const
complete parent
complete child

While in case of destructors, first child destructor exec, then parent dest. executed

26 May Tuesday 2015
18 Week 22nd Day 146

Important
EXAMPLE :-

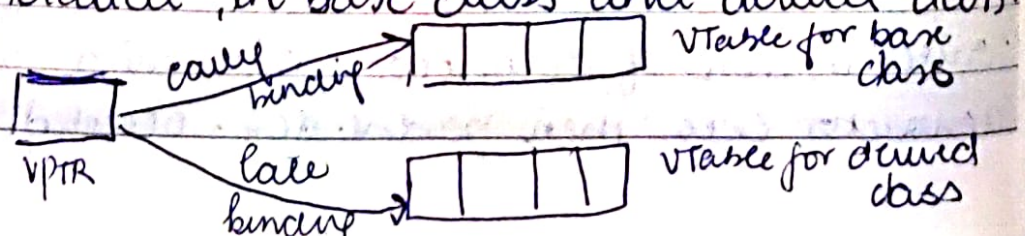
```
class car{
    private:
        int gear_no;
    public:
        void change_gear(int gear){ gear++; }
};
class sportscar : public car{
    void change_gear(int gear){ if(gear>5){gear++;} }
};
int main(){
    sportscar sc;
    sc.change_gear(4);
}
```

8.00 9.00 10.00 11.00 12.00 1.00

fcn of sportscar class will be called.
While calling change_gear(); first it checks if any fcn with this name exists in calling class; otherwise it goes to Base class.
Useful : like we have change_gear for all except one car which has unique method of gearchange.

Working of Virtual fcn (VTable & Vptr)
If a class contains virtual fcn then compiler itself does two things :-

1. A virtual ptr (VPTR) is created everytime obj is created for that class which contains virtual fcn.
2. Irrespective of object is created or not, static array of pointers called VTable where each cell points to each virtual fcn. is created, in base class and derived class.



2015
Week 22nd Day 147th
May
Wednesday 27

Important

Template in C++:

```
template <class x> int check (int a, x b) {
                        if (a > b) return a;
                        else return b;
}
```

It just helps in datatype. So that we can write generic fun that can be used for different datatype

Dynamic Constructor :-

When allocation of memory is done dynamically using dynamic memory allocator 'new' in constructor

```
class geeks {
public:
    void func() { p = new char [6]; }
};
int main() {
    geeks g = new geeks();
```

Virtual Destructor

Deleting a derived class object using a ptr to base class that has non virtual destructor resulting in undefined behaviour ie destructor of base class runs only.

Nested class

It is member and as such has same access rights as any other member

The member of enclosing class have no such access to nested class members

EXAMPLE :

```
class enclosing {
```

Important

```
private :
    int x;
public :
    class nested {
      int y;
      void fun(int a) { x = a }
    };
    void fun1(int b) { y = b.y }
};
```

fun property

error b/c it accnt have access to y

2015

Week 22nd Day 149th

May

Friday 29

Appointments

9.00

10.00

11.00

12.00

1.00

2.00

3.00

4.00

5.00

6.00

JUNE 2015

Monday	1	8	15	22	29	
Tuesday	2	9	16	23	30	
Wednesday	3	10	17	24		
Thursday	4	11	18	25		
Friday	5	12	19	26		
Saturday	6	13	20	27		
Sunday	7	14	21	28		