DATE: 5 May, 2025 **After Mids** DAY: Monday

OOPS (lec-1)

```
int main ()
{
    student s;
    s++; NOT POSSIBLE
}
```

```
int main ()
{
    int a=0
    a++;
}
```

All the operators in C++ don't works on non-primitive (user-defined) data types.

```
int main ()
{
    Time T;
    T++; NOT POSSIBLE
}
```

```
class Time
{
    int H, M, S;
};
```

```
student student
{
    int id;
};
```

⇒ All the operators in C++ don't works on user-defined type

⇒ If we want to use operators on userdefined types then we would change (modify) their functionality.

⇒ But on built in datatypes the operators would perform their original functionality.

* Operator Overloading:

① ++ , -- Unary Operators

```
class Counter
{
    int count;
    Counter(): count(0)
    {
    }
    int getcount()
    {
        return count;
    }
```

```
void operator ++ ( )
{
    ++count;
}
};
```

Output:
0
0
1
2

```
int main()
{
    counter c1, c2;
    cout << c1.getcout();
    cout << c2.getcount();
    C1++;
    ++C1;
    ++C2;
    ++C2;
    cout << c1.getcount();
    cout << c2.getcount();
```

- For operator overloading we would define a function name "operator".
- Number of arguments always equals to operands - 1

DATE: _____ DAY: _____

No of arguments = No of operands − 1,

For (+) Binary No of arguments = 2 − 1 = $\boxed{1}$.

For (++) Unary No of arguments = 1 − 1 = $\boxed{0}$.

- The statement which invokes the "operator" function become Function call statement
- For unary operators like "++" and "--" there is no return type so we will write "void operator ++ ()"
- What the function calling statement is returning would be the return type of "operator function".

e.g: $\boxed{C3 = C1 + C2}$ → function call

it would decide return type

Assignment operator tb ata jb function return kr rha hota

- In above case the return type would be name of class hai "Counter operator + (1ary argument)"

- For Post Fix Unary Operator:

→ we will pass a dummy argument to the "operator" function so it came to know that the operator is post fix.

```
C1 ++;    void operator ++ (int X)
          {
              ++ count;
```

dummy variable

Binary Operator Overloading:

```
class Time {
int H, M, S;
Public:
Time (): H(0), M(0), S(0)
{ }
Time (int h, int m, int s):
H(h), M(m), S(s)
{ }
```

```
Time operator + (Time T)
{ T.H = H + T.H
  T.M = M + T.M
  T.S = S + T.S
  return T }
void getTime()
{
cout << H << ":" << M << ":" << S;
```

T1

```
int main()
{
Time T1, T2(9, 37, 43), T3;
T3 = T1 + T2;
T3.getTime();
}
```

goes as an argument

calling object

[ ]

DATE: 8 May, 2025 DAY: Thursday

OOPS (lec-2)

```
class Counter
{ public:
  Counter operator ++ ()
  {   ~~Counter C2~~;
      ++ count;
  }   return counter(count);
                  ↑ Nameless object
  private :
  int count;
  public:
  Counter (): count (0) { }
  int getCount()
  {
      return count;
  }
};
```

```
int main ()
{
    Counter c1;
    Counter c;
    c = ++c1;
    c.getcount ();
}
```

Counter (count) //call of the constructor where we are passing integer.

Another Method for Counter operator ++() function

```
Counter operator ++ ()
{
    ++ count;
    Counter C2;
    C2.count = count;
    return C2;
}
```

DATE: 19 May, 2025

DAY: Monday

OOPS (lec - 6)

```
class Base
{
    int a;
    public:
    Base() : a()
    {}
    void get()
    {
        cout<<"Enter no.";
        cin>>a;
    }
virtual void show()
    {
        cout<<a;
    }
};
```

```
class Derive1 : Public Base
{
    int D1;
    Public:
    Derive1() : D1(0), Base().
    {}
    void show()
    {
        cout << D1;
    }
};
```

```
class Derive 2 : Public Base
{
    int D2;
    public:
    Derive2() : D2(0)
    void show()
    {
        cout << D2;
    }
};
```

```
int main()
{
    Derive1 Dr1;
    Derive2 Dr2;
    Base *Ptr;
    Ptr = &Dr1;
    Ptr -> show();
    Ptr = &Dr2;
    Ptr -> show();
}
```

function of Derive 1

int *ptr;

ptr is pointer to an integer or pointing to an integer.

Only parents are allowed to store the address of child, but vice versa is wrong

Pointers only stores address of its own type.

Pointer only stores address, not any other thing

DATE: _____ DAY: _____

① Pointer only stores address, nothing other than that.

② Ye sirf uska address store kr skta hai jis ka taraf point krta

```
Float f;
int a;
int *ptr;
ptr = 10;   // Error
ptr = &a;   ✓
ptr = &f;   // Error
```

(.) jab object call karge access operator for object

(→) access operator for pointer.

③ Pointer hamesha kisi doosre ka address store krta.

we cannot do int *ptr; → ptr = &ptr //Wrong

```
int a = 10;
int * ptr;
ptr = &a;
cout << Ptr;      // The address.
cout << * Ptr;    // The content of that address
```

Compilation:

① Checking Errors

② Translating

③ Memory Allocation.

④ Compile time pr sari function calls bind hoti hain

⑤ Compile time pr content check nhi hota, jo pointer jis type(class) ka hota hai woh uske function ko call krta.

```
Base *Ptr;
Ptr = &D1
Ptr → show();   //Pointer is of type Base so it will point to show() function of Base.
Ptr = &D2;
Ptr → show();
```

output:

DATE: DAY:

Base * Ptr;

Ptr → show();

The pointer ptr of type Base is calling the show() function of class Base.

Derive1 * ptr2;

ptr2 → show()

The pointer ptr2 of type Derive1 is calling the show() function of class Derive1.

DATE: 21 - May - 2025 DAY: Wednesday

OOPs (Lec - 7)

```
class Base
{
  public:
virtual void display() = 0;
/* {  //Pure virtual function
   cout << "I am parent";
  } */
virtual void getdata()
  {
  }
};
class Derive1: public Base
{
  public:
  void display()
  {
    cout << "I am child1";
  }
  void getdata()
  {
  }
};
```

```
class Derive2: public Base
{
  public:
  void display()
  {
  cout << "I, am child2";
  }
  void getdata();
  {
  }
};
int main()
{
  Base *ptr;
  Derive1 D1;
  Derive2 D2;
  ptr = &D1;
  ptr -> display();
  ptr -> getdata();
  ptr = &D2;
  ptr -> display();
  ptr -> getdata();
}
```

- object are always in their specialized form
- Parent class ka object ban skta hai but hum bnate nhi hain.
- Parent class is an Abstract class.
- Abstract class woh class hoti hai jis ka object create nhi hoskta means it cannot be used.

DATE: DAY:

• Kon si class Abstract class ho skti hai? wo class jis ke andar pure virtual function mojood ho wo Abstract class hoti hai. Abstract class is written as:

virtual void display () = 0;

Output without virtualization

I am parent

I am parent

Output with virtualization / not pure

I am child 1

I am child 2

Output with pure virtualized function

I am child 1

I am child 2

// But now the Base class becomes abstract class and we cannot make its object. i.e Base b is wrong

DATE: 26-May-2025 DAY: Monday

OOPS (lec - 9)

TEMPLATE: سانچہ

- Gives shape to anything
- Kisi bhi cheez ko apni shape deta hai
- Gives shape to any object

Anything is object

TEMPLATES

Function

```
int add(int a, int b)
{
    return (a+b);
}
int main()
{
    cout << add(10, 20);
    cout << add('a', 'b');
}
```

```
int main()
{
    int arr[20]
    float arr[20]
    char arr[2]
}
```

```
Template <class T, class TT>
int ftn (T T1 , TT T2)
// For every function template is separated.
{ return T1 + T2
```

class

Syntax for function Template

```
template <class T, class TT>
int ftn (T T1 , TT T2)
{
}
int main ()
{
}
```

DATE: DAY:

Class Template

```
Template <class T>
class stack
{
    T arr[10];
    int top;
    public:
    stack ():top(-1)
    {
    }
    void push (int v)
    {
        arr [++ top] = v;
    }
    T pop ()
    {
        return arr[top--];
    }
    void display ()
    {
        int x= top;
        for (int i=0; i<10; i++)
        {
            int x;
            cout << arr[i];
            x++;
        }
};
```

```
int main ()
{
    stack <int> s;
    s.push (20);
    s.push (30);
    s.push (40);
    cout << s.pop();
    s.push (50);
    s.display;
}
```

Multi-template variables

```
Template <class T, class X>
class stack
{ T arr [10]
};
int main ()
{
    stack <int, float> s;
```

Logical : Runtime (gives wrong output)

Syntax : Compilation time

Runtime error : during execution

eg: Divided by zero or infinite searching

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Runtime errors are most dangerous errors

DATE: 2-June-2025 DAY: Monday

OOPS (lec-10)
Exception Handling

Examples of exception:

① cin >> arr [12]; arr has size of 10 and we are accessing 12th index so it is exception (out of bound)

② f << " "; e.g: we are writing data into a file, may be it is possible that we are writing data into wrong file or the location of file is wrong.

Any statement where it is risk of raising any error, is written in try { } block.

- Generally we write try{} in main().

FLOW OF CONTROL

```
main()                                  input()
{                                       {
// a.input();                              cin >> arr[12];
Try                                        throw;
{                Exception                 Exception is thrown
    a.input();   ①
}
catch()  // immideate statement
{        // after try
```

exception which is thrown, is catched by catch()

- Har function mein jahan exception aane ka risk hoga uski alag throw (class) hogi
- class error { }; is used to throw the object.
- It is always inside general class or the class we are using.
- The object which is thrown is basically the exception or error

```
class test{
    class error{
};          };
```

DATE: _____ DAY: _____

• Exception always comes from function (where there is error) and catch () { } is handler of that ~~functi~~ exception.

PROGRAM EXCEPTION HANDLING

```
class Test
{
  int arr[5];
  Public :
  class Error
  { };
  void input ()
  {
    for (int i=0; i<5; i++)
    ~~cin~~ cin >> arr [i];
    if (i>4 || i<0)
      throw Error ();
  } // All the statements
};  will be skipped if an
    error is occured. /*
```

```
int main()
{
  Test T;
  try
  {
    T.input();
  }
  catch (Test :: Error)
  {
    cout << "array out
              of bond";
  }
```

```
Class stack
{
  int arr [5];
  int top ;
  public:
  class Errorup{ }; class Errorlo{};
  stack () : top(-1) { }
  void push ( int v)
  {
    arr[++top] = v;
    if (top <0 || top > 4)
      Throw Errorup();
  }
  int pop ()
  {
    int temp ;
    temp = arr [top --].
    if (top<0 || top>4)
      Throw Errorlo();
  } return temp;
```

DATE: _____ DAY: _____

• At a time one catch is executed because only one error is thrown at a time.

```
int main()
{
    Stack s;
    try
    {
        s.push(10);
        s.push(20);
        s.push(30);
        s.push(50);
        s.push(100);
        s.push(70);
        cout << s.pop();
        cout << s.pop();
        cout << s.pop();
        cout << s.pop();
        cout << s.pop();
        cout << s.pop();
    }
    catch (stack::Errorup)
    {
        cout << "stack is out of upper bond"; overflow
    }
    catch (stack::Errorlo)
    {
        cout << "stack is out of lower bound"; underflow
```

```
void divide()
{
    cin>> dividend;
    cin>> divisor;
    ~~float~~ if (divisor==0)
        throw Error();
}
```

• After the "catch" the program will be ended.

DATE: 4- June - 2025 DAY: Wednesday

OOP(lec-11)

Exception Handling

```
class calculator
{
    int v1, v2;
    public:
    calculator ( ): v1(0), v2(0)
    { }
    calculator (int va, int vb) : v1(va) , v2(vb)
    { }
    class DivError
    {
        int v;
        string loc;
        public:
        DivError () : v( ) { }
        DivError (int vv, string lo) : v(vv), loc (lo)
        { }
    } i
    void Div()
    {
        if (v2 == 0)
        throw DivError (v2, "Division");
        else
        int D = v1/v2;
    }
};
```

```
int main()
{
    calculator c;
    try
    {
        c.div();
    }
    catch (calculator :: DivError DE)
    {
        cout << DE.v
        cout << DE.loc
    }
}
```

DATE: 11-June-2025 DAY: Wednesday

OOPS (lec - 12)

- We can make our arrays safe by
  1. using subscript operator (operator overloading)
  2. using exception handling

① char Name [20];

② char * name;

③ char Name [ ] = "I am a Pakistani" "\0"

int slen = strlen (Name); <string.h> 16

char * ptr; p = "\0" Space

ptr = New char (slen + 1) p = Name

- Dynamically allocate kro char type ki (slen+1) memory ptr ko.
- "New" is reserved word which is used in dynamic memory allocation
- "New" function is to give the required space
- "Memory which is allocated by "New" would not be free after the scope is ended.
- we use the keyword "delete"

delete ptr; d string name;

- The size of the array must be constant.

getline (cin, name)

char name [10];

cin.getline (name, 10)

DATE: 16- June - 2025 DAY: Monday

OOPS (lec-13)

Insertion and Extraction Operator

1- cout << "Hello" (Write Hello to console output)

use to write

write → output

read → input

2- int a

cin >> a ; (Read into "a" from console Input)

used to read

Q1: Kis ki streaming honi hai?

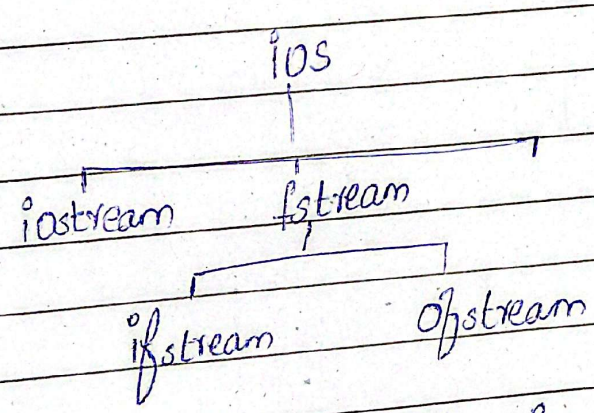
Q2: Kahan par streaming honi hai?

Q3: Kis ne aur kis type streaming honi hai?

cout << "Hello"

Kahan write krna hai

Kya write krna hai



ofstream f("Hello.txt", ios::in)

// we use ios::in to avoid file overriding otherwise

// new file of same name and type will be created

DATE: 23-June-2025 DAY: Monday

OOPS (lec - 14)

```
class Student
{
    string name;
    int RNO;
    public:
    Student ()
    {
      strcpy (Name," ");
      RNO = 123;
    }
    Student (string N, int R)
    {
      strcpy (~~string~~ Name, N);
      RNO = R;
    }
    void display ()
    {
      cout << RNO << " ";
      cout << Name;
    }
    void writedata ()
    {
      ofstream fout;
      fout.open ("data.txt", ios::app);
      fout << RNO << " " << name;
    }
    void readdata ()
    {
      ifstream fin;
      fin.open("data.txt", ios::in);
      string N;
      int R;
      fin >> R >> N;
      cout << R << " " << N;
      if (!fin) cout << "Not found";
      else {
      while (fin >> RNO >> Name)
      {
        display ();
```