ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming

First published on 3 July 2012
This is the 8th Revised edition

Updated on:
12th April 2020

DISCLAIMER

- The data in the tutorials is supposed to be one for reference.
- We have made sure that maximum errors have been rectified. Inspite of that, we (ECTI and the authors) take no responsibility in any errors in the data.
- The programs given in the tutorials have been prepared on, and for the IDE Dev-C++.
- To use the programs on any other IDE, some changes might be required.
- The student must take note of that.

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 1

Revision of C Programming

- Variables
- Conditional statements
- Switches
- Loops
- Functions
- Arrays
- Strings
- Structures
- Pointers

Basic Programming Skills

- Indentation is a MUST
  - Use “Tabs” not spaces
- Program names should make sense
  - A program for array operations should be something like **arrays.cpp** and not **untitled23.cpp**
- Comments are mandatory
  - Not for every statement
  - For every block of **Business Logic**
- Header for each program is a very good habit
  - This includes Program title, Problem Statement, Exam Seat Number, Assumptions (if any), Code revision history (if any)

Procedure-Oriented Programming

- The procedure oriented language deals with a sequence of processes to be done such as reading, calculating and printing.
- The **program is divided into functions** to achieve these tasks.
- In procedure oriented languages the main attention is given to how to achieve a particular task and very **less attention is given to the data** which is used by the functions.
- If the data is required to be accessed by many functions then it is declared as **global**. Global declarations are more vulnerable as data can be changed by all the functions.

Object Oriented Programming

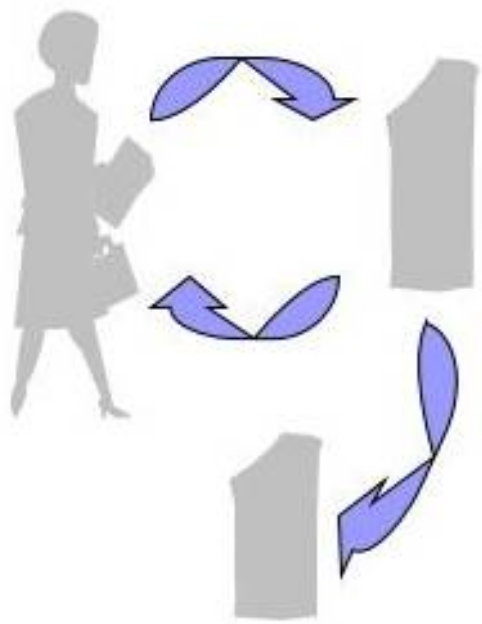
- More emphasis is given on data rather than procedure
- Programs are divided into objects
- Classes are designed such that they characterize the objects
- The data and the functions that can operate on the data are tied together
- Data is hidden and cannot be accessed by external functions
- New data and functions can be added whenever required
- Follows bottom-up Design approach

…continued

- Object-oriented programming is a programming methodology that associates data structures with a set of operators which act upon it
- It gives importance to relationships between objects rather than implementation details
- Hiding the implementation details within an object results in the user being more concerned with an objects relationship to the rest of the system, than the implementation of the object's behaviour

The OOP Paradigm

Procedure Oriented

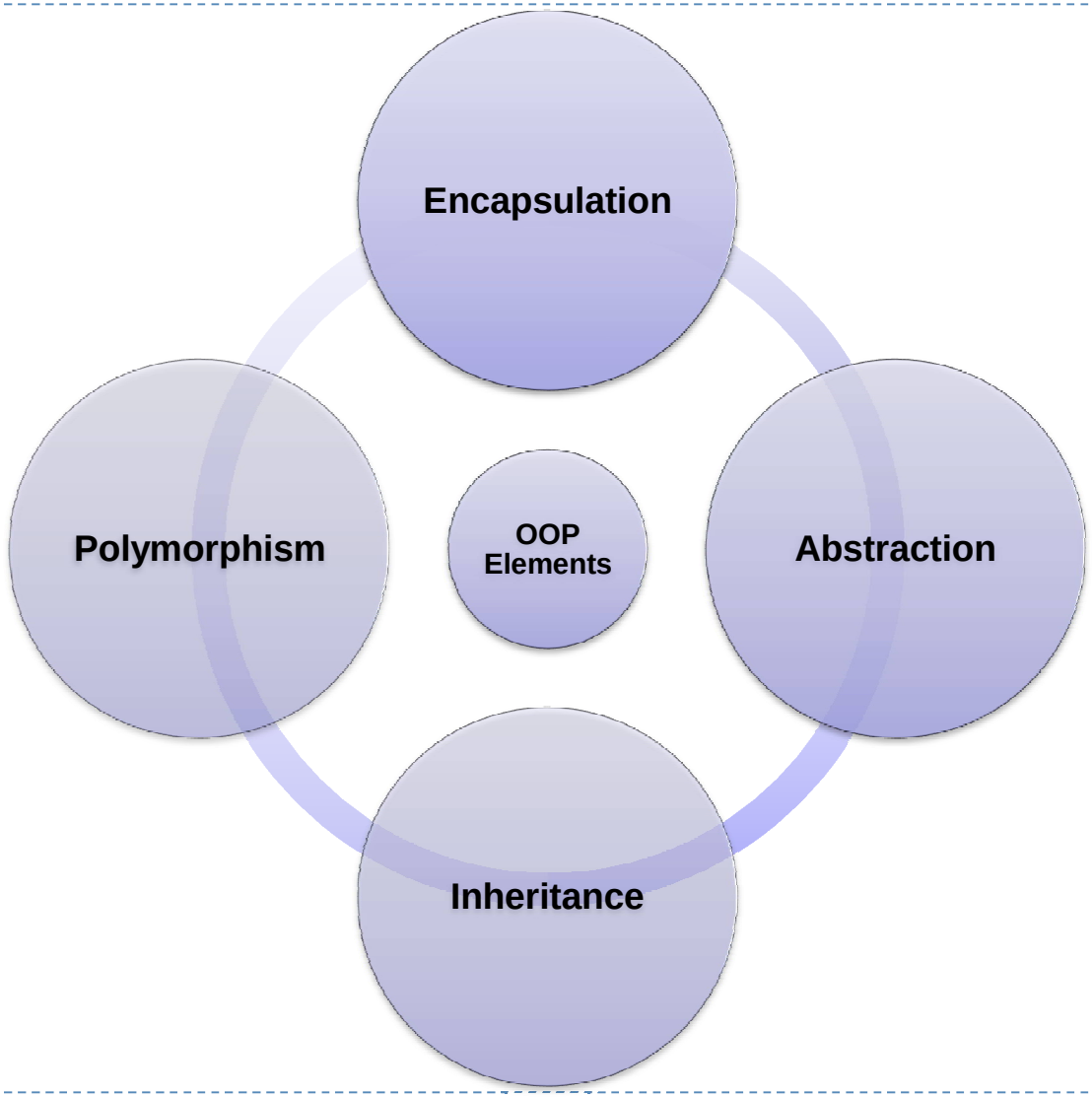


Withdraw, deposit, transfer

Object Oriented

Customer, money, account

Elements of OOP



Introduction to C++

- C++ language was developed by **Bjarne Strousstrup** at AT & T Bells Laboratories in early **1980's**.
- He thought of a language which can have object oriented features as well as it can retain the simplicity of C language.
- Initially the language was names as **'C with classes'**. However, later in **1983** the name was changed to **C++**.
- C++ almost supports all the C functionalities with some new functionalities like classes, inheritance, function overloading, operator overloading. These features of C++ enable creating abstract data types, inherit properties from existing data types etc.

First C++ Program

```
#include<iostream>

using namespace std;
int main()
{
   cout << "Hello World!!!";
   return 0;
}
```

Output:

Hello World!!!

- **#include<iostream>** instruction causes to add **iostream** file to program which contains the declarations of the identifier **cout**.
- **cout (<<)** is called as a Output Operator which prints string to the console.
- **Namespace** is a collection of identifiers i.e. name of classes, functions, variables etc. Namespace is used to name collisions.

Note: The header files should now be added by the student. Henceforth, no headers are mentioned in the sample programs.

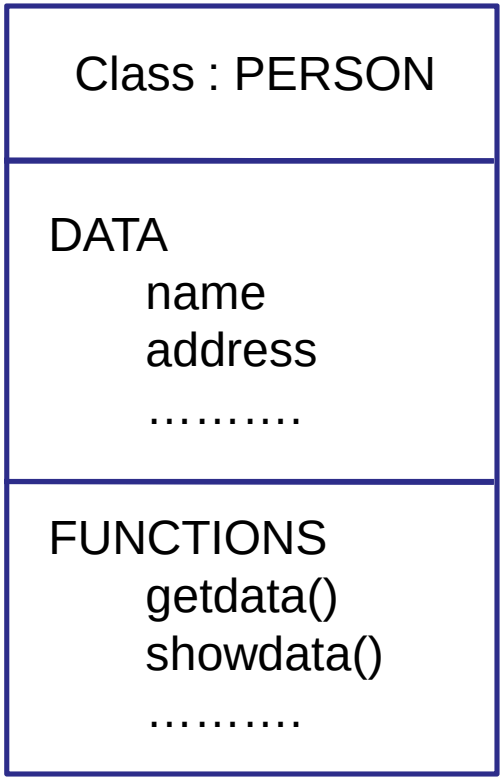
Class

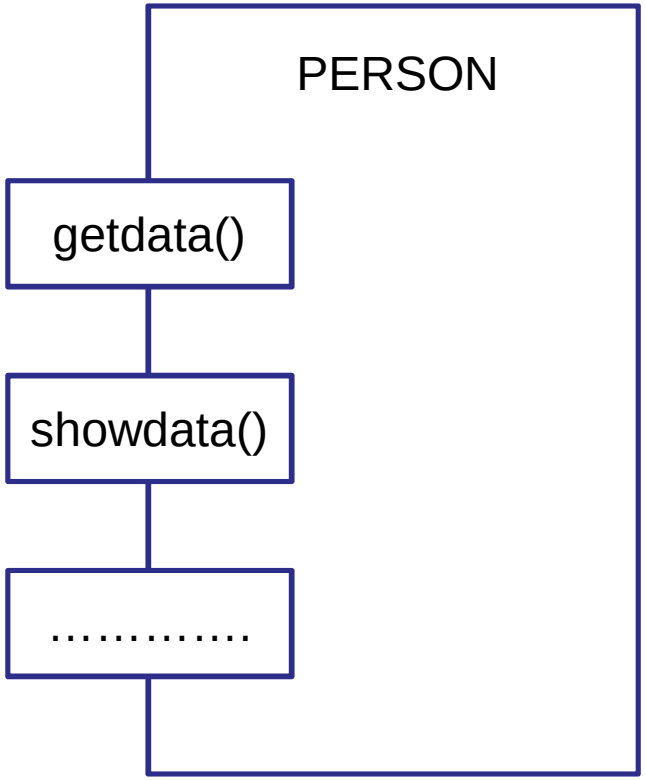
```
class class_name
{
    private:
        variables;
        functions;
    public:
        variables;
        functions;
};
```

- The body of a class contains **variables and functions**.

- A class defines the structure and behavior (data and code) that will be shared by a set of objects.
- The purpose of class is to encapsulate complexity, there are mechanisms for hiding the complexity of the implementation inside the class.
- The words **private** and **public** are called as **access specifiers**. The default access specifier is **private** in C++.

Class Representation





Objects

```
class Addition
{
   int no1, no2, sum;
   public:
   void getData()
   {
        cout << "Enter 2 nos: " ;
        cin >> no1 >> no2;
   }
   void calSum()
   {
        sum = no1 + no2;
   }
   void showData()
   {
        cout << "Sum is: " << sum;
   }
};
```

```
int main()
{
   Addition obj;
   obj.getData();
   obj.calSum();
   obj.showData();
   return 0;
}
```

- Each object of a given class contains the structure and behavior defined by the class.
- Thus **class is a logical representation**, where as **object has physical representation**.
- The memory is always assigned to object and not to a class.

END OF DAY 1

- Thus, Day 1 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 2

Revision of Day 1

- Elements of Object Oriented Programming
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
- Class
- Object

Member Functions

Defining Function inside Class

```
class Addition
{
   int no1, no2, sum;
   public:
   //inline function declared
   inside class
   void getData()
   {
        cout << "Enter 2 nos:" ;
        cin >> no1 >> no2;
   }
   void showData()
   {
        cout << "Sum: " << sum;
   }
};
```

Defining Function outside Class

```
class Addition
{
   int no1, no2, sum;
   public:
   void getData();  //prototype
   declaration inside class
   void showData()
   {
        cout << "Sum: " << sum;
   }
};

//function defined outside class
void Addition :: getData()
{
   cout << "Enter 2 nos: " ;
   cin >> no1 >> no2;
}
```

Defining Functions Outside Class

- When you are defining a function outside a class you have to incorporate **membership 'identity label'** in the header. The 'identity label' tells the compiler which class the function belongs to.

```
return_type class_name :: function_name (arguments)
{
        ..............;
        ..............;
}
```

- Due to **:: (scope resolution operator)** the compiler understands the function **function_name** is restricted to **class_name**.
- Due to membership label many classes can use same function name.
- Member functions can access the private data of the class.
- Member function can call another member function without using dot operator.

Inline Functions

- When we call a function it takes extra time to execute.
- To eliminate the cost of calls to small functions **inline functions** are used.
- **An inline function is a function which is expanded in line when it is invoked**.
- The compiler replaces the function call with the corresponding function code.

```
inline function_header
{
        ..............;
        ..............;
}
```

- The faster execution of inline function diminishes when the function grows in size.
- Inline functions should be between 1 to 2 lines.

Default Arguments to a Function

- A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument.

```
class Addition
{
   int no1, no2, sum;
   public:
   void getData(int n1,int =10);
   //default argument =10
   void calSum();
   void showData()
   {
        cout << "Sum: " << sum;
   }
};
```

```
void Addition::getData(int n1,
   int n2)
{
   no1 = n1;
   no2 = n2;
}
void Addition :: calSum()
{
   sum = no1 + no2;
}
int main()
{
   Addition obj;
   obj.getData(100,200);
   obj.calSum();
   obj.showData();
   obj.getData(50);
   obj.calSum();
   obj.showData();
   return 0;
}
```

END OF DAY 2

- Thus, Day 2 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 3

Revision of Day 2

- Defining Member Functions outside the class
- Inline functions
- Default arguments to a Function

Passing Object as a Parameter to Function

- You can pass object as a function argument.
- If you pass entire object to function then it is called as **pass-by-value** and if only you pass addresses of object then it is called as **pass-by-reference**.
- **obj1.calsum(obj2),** in this function call obj2 is passed by value and hence the function prototype will be as follows – **void calsum (class_name o).** This means if we change variable values of **o** they will not affect values in obj2.
- **obj1.calsum(obj2),** in this function obj2 is passed by reference and hence the function prototype will be as follows – **void calsum (class_name &o).** This means if we change variable values of o it will affect values in obj2.
- **Object to Pointers**-

  obj1.calsum(obj2) will be passed to function as follows – **void calsum(class_name *o).** This means o will point to the memory of obj2.

Returning Objects

```
class Maths
{
   int no;
   public:
   void getData(int);
   void showData();
   Maths calSum(Maths &);
};
void Maths :: getData(int num)
{
   no = num;
}
void Maths :: showData()
{
   cout << "The number is:" <<
   no;
}
```

```
Maths Maths :: calSum(Maths &o)
{
   Maths temp;
   temp.no = no + o.no;
   return temp;
}
int main()
{
   Maths obj1, obj2, obj3;
   obj1.getData(10);
   obj2.getData(20);
   obj3 = obj1.calSum(obj2);
   obj1.showData();
   obj2.showData();
   obj3.showData();
   return 0;
}
```

- Here **obj1** is called as **invoking object** and the default memory will be of obj1.

Array of Objects

```
class Student
{
    int roll_no;
    char name[50], address[50];

    public:
    void getData();
    void showData();
};

void Student :: getData()
{
    cout << "Enter roll no.: ";
    cin >> roll_no;
    cout << "Enter name: ";
    cin.ignore();
    cin.getline(name,50);
    cout << "Enter address: ";
    cin.getline(address);
    cout << endl << endl;
}
```

```
void Student :: showData()
{
    cout << "Roll no.: " << roll_no;
    cout << "Name: " << name;
    cout << "Address: " << address;
    cout << endl << endl;
}

int main()
{
    Student s[20]; //array of objects
    int size;
    cout << "Enter array size : ";
    cin >> size;
    cout << "Enter data of " << size
    << " students: ";
    for(int i=0; i<size; i++)
         s[i].getData();
    cout << "The information is: ";
    for(int i=0; i<size; i++)
         s[i].showData();
    return 0;
}
```

END OF DAY 3

- Thus, Day 3 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 4

Revision of Day 3

- Passing Object as a Parameter to Function
  - Pass By Value
  - Pass By Reference (Object Referencing)
  - Pass By Pointers
- Returning Objects
- Array of Objects

Array of Objects to Functions

```
class Student
{
    int roll_no;
    char name[50], address[50];
    public:
    void getData();
    void showData(Student [], int);
};
void Student :: getData()
{
    cout << "Enter roll no.: ";
    cin >> roll_no;
    cout << "Enter name: ";
    cin.ignore();
    cin.getline(name,50);
    cout << "Enter address: ";
    cin.getline(address,50);
    cout << endl << endl;
}
```

```
void Student :: showData(Student s[],int n)
{
    for(int i=0;i<n;i++)
    {
      cout << "Roll no.: "<< s[i].roll_no;
      cout << "Name: " << s[i].name;
      cout << "Address: " << s[i].address;
      cout << endl << endl;
    }
}
int main()
{
    Student s[3], s1;
    cout << "Enter data of 3 students: ";
    for(int i=0; i<3; i++)
        s[i].getData();
    cout << "The information is: ";
    s1.showData(s,3); // array to function
    return 0;
}
```

Array of Objects to Pointers

```
class Student
{
    int roll_no;
    char name[50], address[50];
    public:
    void getData();
    void showData(Student *, int);
};
void Student :: getData()
{
    cout << "Enter roll no.: ";
    cin >> roll_no;
    cout << "Enter name: ";
    cin.ignore();
    cin.getline(name,50);
    cout << "Enter address: ";
    cin.getline(address,50);
    cout << endl << endl;
}
```

```
void Student :: showData(Student *s,int n)
{
    for(int i=0; i<n; i++)
    {
      cout << "Roll no.: "<< (s+i)->roll_no;
      cout << "Name: " << (s+i)->name;
      cout << "Address: " << (s+i)-> address;
      cout << endl << endl;
    }
}
int main()
{
    Student s[3], s1;
    cout << "Enter data of 3 students: ";
    for(int i=0; i<3; i++)
        s[i].getData();
    cout << "The information is: ";
    s1.showData(s,3); // array to pointer
    return 0;
}
```

END OF DAY 4

- Thus, Day 4 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 5

Revision of Day 4

- Passing Array of Objects to Functions
- Passing Array of Objects to Pointers

Polymorphism

- The word **polymorphism** means having many forms
- **C++ polymorphism** means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function

Function Overloading

- You can define **more than two functions having same name** as long as their parameters are different.
- Function overloading is one of the ways that C++ supports **polymorphism**.
- The **type and/or number of arguments** determine which overloaded version is to be called.
- C++ does not allow overloaded functions with same parameters but different return types.

```
class Maths
{
    void sum()
    {
        ……
    }
    int sum(int x, int y)
    {
        …….
    }
    void sum(int x)
    {
        …….
    }
    void sum(int x, float y)
    {
        ……
    }
};
```

```
void sum(int x,int y)
{
    ……
}
```

END OF DAY 5

- Thus, Day 5 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 6

Revision of Day 5

- Function Overloading

Constructors

- In C++ you can initialize the class variables at the time of creation of objects using constructors.
- **Constructor has same name as the class** in which it resides and syntactically similar to a method.
- **Constructors do not have return type nor they are void**.
- The constructor is automatically called immediately after the object is created.

```
class Demo
{
   int a, b;
   public:
   Demo()
   {
        a = 10; b = 20;
   }
   void showData()
   {
        cout << "A=" <<  a <<
   "B=" << b;
   }
};

int main()
{
   Demo obj1;
   obj1.showData();
   return 0;
}
```

Constructor will get automatically called when the object obj1 is created

Parameterized Constructors

- In the example seen before variables of all the objects created will get initialized to 10 and 20. If we want to initialize them to different values you have to use parameterized constructors.
- The constructors with arguments is called as parameterized constructors.

```
class Demo
{
   int a, b;
   public:
   Demo(){}
   //parameterized constructor
   Demo(int x, int y);
   …….;
   …….;
};

Demo :: Demo (int x, int y)
{
   a = x; b = y;
}
```

- You can call the parameterized constructor implicitly or explicitly
- **Implicit call**: Demo obj(10, 20);
- **Explicit call**: Demo obj = Demo(10,20);

Constructor Overloading

```
class maths
{
        int x, y;
   public:
        maths()
        { x = 0; y = 0}
        maths(int a, int b)
        { x = a; y =  b;}
        maths(maths &o)
        { x = o.x, y = o.y}
};
```

- In the above class we have overloaded three constructors.
- **maths obj** declaration will invoke first constructor where x and y will be initialized to 0.
- **maths obj1(100, 200)** will invoke second constructor where x and y will be initialized to 100 & 200 respectively.
- **maths obj2(obj1)** will invoke third constructor and will copy the values of obj1 to obj2. Hence, it is called as **copy constructor**.
- Some other examples of copy constructor –
- **maths obj2 = obj1** will create a new object obj2 and same time will initialize the values to that of obj1.
- **obj2 = obj1** will not invoke copy constructor but due to **= operator overloaded** it will initialize the values of obj2 to obj1 member by member.
- The parameter to a constructor can be any type except the class it belongs to.
- Hence maths(maths o) will not work but **maths(maths &o)** will work because we have passed the reference of it's own class.

Destructor

- Destructor is used to destroy the objects created by constructors.
- Same as constructor destructor name is also same as that of class name preceded by tilde (~) character.
- Destructor does not take any argument nor returns any value.
- Destructor is automatically invoked by the compiler on exit of a program or a block of code in which the objects are declared.
- Destructor releases the memory for further use.

```
~Student() //Destructor
{
   cout << "In Destructor";
}
```

END OF DAY 6

- Thus, Day 6 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 7

Revision of Day 6

- Constructor
  - Implicit Call
  - Explicit Call
- Parameterized Constructors
- Constructor Overloading
- Destructor

this Pointer

- This pointer is used to represent an object that invokes a member function.
- This pointer points to the object on which a function is called e.g. **obj.sum()** will set **this pointer to the address of object obj**.

Program with ambiguity issue

```
#include<iostream>

Using namespace std;

class Maths
{
   int a, b;
   public:
   Maths(int a, int b)
   {
        a=a; b=b; //both a & b
                  are locals
   }
   void display()
   {
     cout << a << ", " << b;
   }
};
```

```
int main()
{
     Maths obj1(10, 20);
     obj1.display();
     return 0;
}
```

- Here we are expecting the output would be **10, 20**. But the output would be some garbage values. Because a and b are the local variables. Hence the statement a = a and b = b assigns the values of local variables a and b to itself.

Use of this keyword

```
#include<iostream>

Using namespace std;

class Maths
{
   int a, b;
   public:
   Maths(int a, int b)
   {
        this->a = a;
        this->b = b;
   }
   void display()
   {
     cout << a << ", " << b;
   }
};
```

```
int main()
{
     maths obj1(10, 20);
     obj1.display();
     return 0;
}
```

The program output is as follows:-

10, 20

new, delete Keywords (Dynamic Memory Allocation)

- Allocating memory to the objects at the time of their construction is called as Dynamic Memory Allocation. The memory is allocated with the help of **new keyword**.
- Dynamic memory allocation enables programmer to allocate right amount of memory when the objects are not of the same size.
- **delete keyword** is used to destroy the memory created with the new operator.

Use of new keyword

```
#include<iostream>
#include<string.h>

using namespace std;

class Str
{
   char *name;
   int length;
   public:
   Str()
   {
      length = 0;
      name = new char[length+1];
   }
   Str(const char *s)
   {
      length = strlen(s);
      name = new char[length+ 1];
      strcpy(name, s);
   }
   void display()
   {
        cout << name << "\n";
   }
   void join (Str &, Str &);
};

void Str :: join (Str &a,Str &b)
{
   length = strlen(a.name) +
   strlen(b.name);
   delete name;
   name = new char [length + 1];
   strcpy(name, a.name);
   strcat(name, b.name);
}
```

continued on next slide….

Use of new keyword

```
int main()
{
   const char *first = "Sapna ";

   Str name1(first),
   name2("Prajakt "),
   name3("Shravani "), s1, s2;

   s1.join(name1, name2);
   s2.join(s1, name3);

   name1.display();
   name2.display();
   name3.display();
   s1.display();
   s2.display();
   return 0;
}
```

The output of the program is –

Sapna

Prajakt

Shravani

Sapna Prajakt

Sapna Prajakt Shravani

Shallow Copy Vs. Deep Copy

- The process of copying data from one object to other object can be achieved by **copy constructor**. There are two types of copy constructors –
  - Implicit Copy Constructor
  - User defined Copy Constructor
- Copying data using Implicit Copy Constructor is called as Shallow Copy.
- Copying data using User defined Copy Constructor is called as Deep Copy.

```
class Student
{
   int age;
   int *rollNo;
   public:
   Student()
   {
      this->rollNo = new int;
   }
   void setRollNo(int rno)
   {
      *this->rollNo = rno;
   }
```

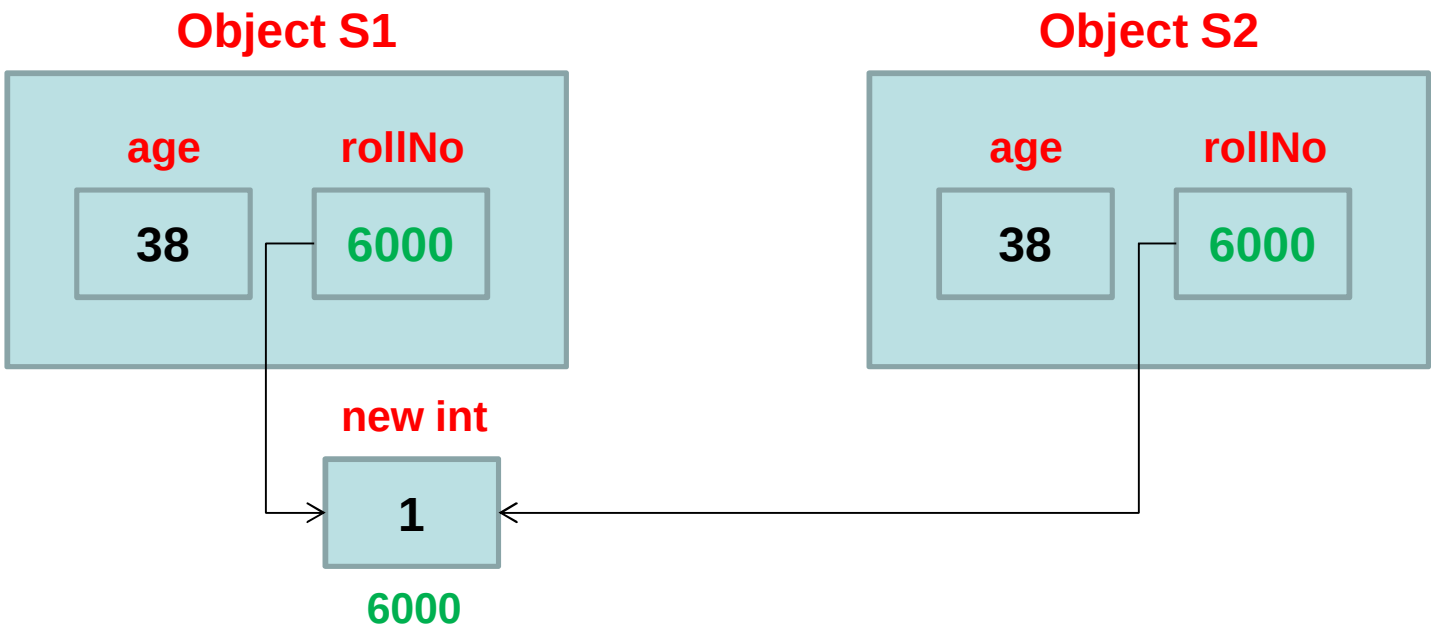
Continued…

...continued

```
    void setAge(int age)
    {
        this->age = age;
    }
    Student(Student &o)
    {
        cout<<"Deep Copy"<<endl;
        this->age = o.age;
        this->rollNo = new int;
        *this->rollNo=*o.rollNo;
    }
    void show()
    {
        cout << "Hi " <<
        *this->rollNo << ", your
        age is: " << this->age
        << endl << endl << endl;
    }
};
```

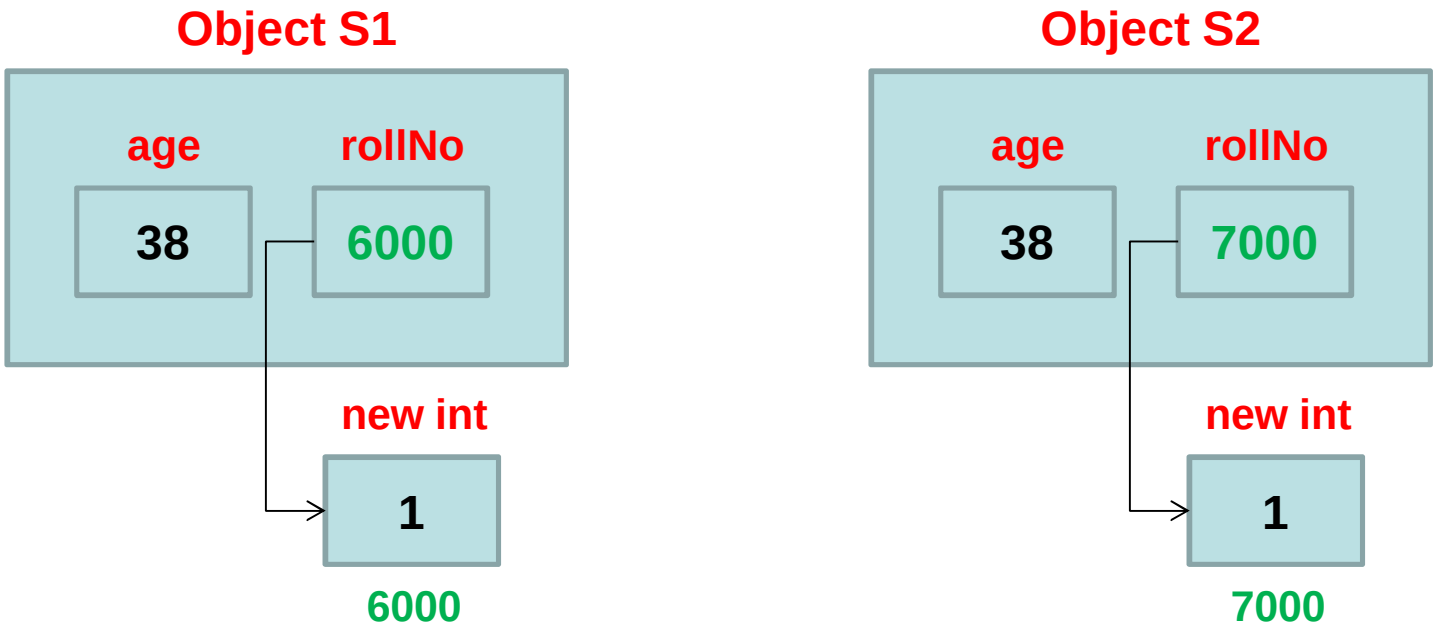
```
int main()
{
    Student s1;
    s1.setAge(38);
    s1.setRollNo(1);
    //will call copy constructor
    Student s2(s1);
    s2.setRollNo(2);
    //will call copy constructor
    Student s3 = s2;
    Student s4;
    //implicit = operator will
    copy data from s3 to s4
    s4 = s3;
    s1.show();
    s2.show();
    s3.show();
    s4.show();
    return 0;
}
```

Shallow Copy (Default Copy Constructor)



Hence, any changes made in rollNo of Object s2 will affect rollNo of s1 Object.

Deep Copy (User Defined Copy Constructor)



Hence, any changes made in rollNo of Object s2 will not affect rollNo of s1 Object.

END OF DAY 7

- Thus, Day 7 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 8

Revision of Day 7

- this Pointer
- Dynamic Memory Allocation
  - new keyword
  - delete keyword
- Copy Constructor
  - Shallow Copy
  - Deep Copy

Static Data Members

- We can declare any data member as **Static**.
- Static data member is **initialized to zero** when the first object of the class is created.
- When we define a data member as Static **only one copy of that member is created** for all the objects.
- It is visible only within the class, but its **lifetime is for the entire program**.

```
class Stat
{
   int no;
   //static data member scount
   static int scount;
   public:
      void getData(int x)
      {
         no= x;
         scount++;
      }
      void getCount()
      {
         cout<<"Count: "<<scount;
      }
};
```

continued on next slide….

Static Data Members

```
int Stat :: scount;
int main()
{
   Stat a, b, c;
   a.getCount();
   b.getCount();
   c.getCount();
   a.getData(100);
   cout << "After
   initialization: " << endl;
   a.getCount();
   cout << "After
   initialization : " << endl;
   b.getData(200);
   b.getCount();
   c.getData(300);
   cout << "After
   initialization : " << endl;
   c.getCount();
   return 0;
}
```

The output of the program is –

Count: 0

Count: 0

Count: 0

After initialization:

Count: 1

After initialization:

Count: 2

After initialization:

Count: 3

Static Member Functions

- We can declare static member functions as we define a data member as static.
- A static function can access other static member functions or static data members.
- Static functions are **called using the class name** and cannot be called using the objects of the class.

```
class Stat
{
   int no;
   static int scount;
   public:
     void setNo()
     {
       no = ++scount;
     }
     void showNo()
     {
       cout << "\nObject
       Number:" << no;
     }
```

continued on next slide….

Static Member Functions

```
    static void showCount()
    {
        cout<<"\nCount: " <<
        scount;
    }
}; //end of class
int Stat :: scount;
int main()
{
    Stat s1, s2;
    s1.setNo();
    s2.setNo();
    //calling static function
    Stat :: showCount();
    Stat s3;
    s3.setNo();
    Stat :: showCount();
    s1.showNo();
    s2.showNo();
    s3.showNo();
    return 0;
} //end of main
```

The output of the program is:-

Count: 2

Count: 3

Object Number: 1

Object Number: 2

Object Number: 3

Friend Functions

- The private members of a class cannot be accessed from outside the class.
- There might be situation in which the two classes need to share a particular function, we need to make that function as a friend function of both the class.
- So using friend function we can access private data of both the classes.
- You cannot call a friend function using an object of a class, it needs to be called as normal function.

```
class ABC
{
    ……. …….
public:
    ……. …….
    friend void name(ABC);
};
```

- If a function is a **friend function of a particular class** then you **need to pass at least one object** of the that particular class as an argument.

Friend Function of Single Class

```
class Math
{
   int a, b;
   public:
   void setData(int x, int y)
   {
      a = x;
      b = y;
   }
   friend float mean(Math &);
};
float mean (Math &m)
{
    float x = (m.a + m.b)/2.0;
    return x;
}
```

```
int main()
{
   Math mobj;
   mobj.setData(22,19);
   cout<<"Mean = "<<mean(mobj);
   return 0;
}
```

The output of the program is:-

Mean = 20.5

Friend Function to Two Classes

```
class PQR; //forward declaration
class LMN
{
   int a;
   public:
    void setData(int x)
    {
       a = x;
    }
    friend void max(LMN, PQR);
};
class PQR
{
    int b;
    public:
     void setValue(int y)
     {
        b = y;
     }
     friend void max(LMN, PQR);
};
```

```
void max(LMN l, PQR p)
{
   if(l.a > p.b)
     cout<<"Max No. = "<<l.a;
   else
     cout<<"Max No. = "<<p.b;
}
int main()
{
   LMN lmn;
   lmn.setData(100);
   PQR pqr;
   pqr.setValue(200);
   max(lmn, pqr);
   return 0;
}
```

The output of the program is:-

Max No. = 200

END OF DAY 8

- Thus, Day 8 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 9

Revision of Day 8

- Static Data Members
- Static Member Functions
- Friend Functions
  - Friend Function of a Single Class
  - Friend Function of Two Classes

Operator Overloading

- In C++ you can give **special meaning to the operators**. This process is called as operator overloading.
- In C++ all the operators except the following operators can be overloaded –
  - Class member operators (. .*)
  - Scope resolution Operator (::)
  - Size operator (sizeof)
  - Conditional operators (?:)

```
return_type class_name ::
operator op(Argument List)
{
        ..............;
        ..............;
}
```

- Operator functions must be either member functions (non-static) or friend functions.
- Friend function will only have one argument for unary operators and two for binary operators.
- Member function has no arguments for unary operators and only one for binary operators.

Overloading Unary Operators

```
class Opunary
{
   int no;
   public:
   void getData(int );
   void showData();
   void operator –();
};

void Opunary :: getData(int a)
{ no = a; }
void Opunary :: showData()
{cout << no; }
void Opunary :: operator –()
{no = -no; }
```

```
int main()
{
   Opunary obj;
   obj.getData(10);
   cout << "Value is: ";
   obj.showData();
   obj;
   cout << endl<< "Value is: ";
   obj.showData();
   return 0;
}
```

The output of the program is:-

Value is: 10

Value is: -10

Overloading Binary Operators

```
class Complex
{
   double real, img;
   public:
   Complex(){ }
   Complex(double r, double i)
   {
      real = r; img = i;
   }
   Complex operator +(Complex &);
   void showComplex();
};
Complex Complex :: operator
+(Complex &c)
{
  Complex temp;
  temp.real = real + c.real;
  temp.img = img + c.img;
  return temp;
}
```

```
void Complex :: showComplex()
{
  cout<<real<<"+"<<img<<"i";
}

int main()
{
   Complex c1, c2, c3;
   c1 = Complex(3.2, 1.8);
   c2 = Complex(2.3, 5.3);
   c3 = c1 + c2;
   cout << "C1 = ";
   c1.showComplex();
   cout << "C2 = ";
   c2.showComplex();
   cout << "C3 = ";
   c3.showComplex();
   return 0;
}
```

Pitfalls of Operator Overloading

- Programmers most of the time are unaware of coding semantics
- This makes debugging or maintaining the code a disaster in making
- Technically, the Operators can be overloaded but their presidency cannot be overloaded. This may create horrible situations thus proving functions (overloading) being better
- An operator does not automatically invoke the operator of the base class in case of inheritance*

END OF DAY 9

- Thus, Day 9 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 10

Revision of Day 9

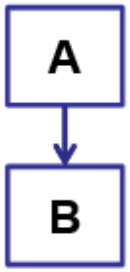
- Operator Overloading
  - Overloading Unary Operators
  - Overloading Binary Operators
- Pitfalls of Operator Overloading

Inheritance

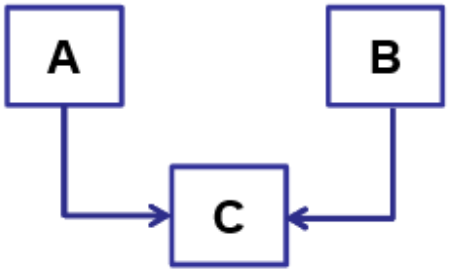
- Inheritance is a feature of OOP which enables user for reusability of the code.
- You can create new classes which can use properties of existing classes using inheritance and we can add new features in derived class.
- In inheritance the old class is called as base class and the new class is called as derived class.
- The derived class inherits some or all features from base class.
- One class can be derived from many base classes or many classes can be derived from one base class.

```
class derived : visibility_mode base {
……;
……;
}
```

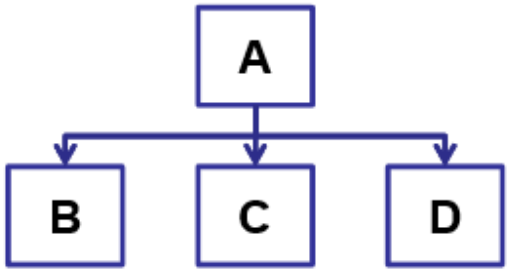
Types of Inheritance



Single Inheritance

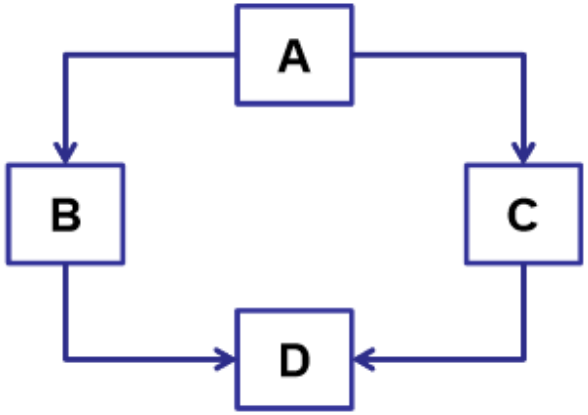


Multiple Inheritance



Hierarchical Inheritance

Multilevel Inheritance



Hybrid Inheritance

Deriving a Class from Base Class

Default Private Derivation:

```
class derived : base
{
   members of derived class;
}
```

Private Derivation:

```
class derived : private base
{
   members of derived class;
}
```

Public Derivation:

```
class derived : public base
{
   members of derived class;
}
```

- The : indicates derivation of derived from base class. Private or Public indicate the visibility of the features of base class into derived class.
- If **privately derived**, the public members of base class become private members of derived class and if **publically derived** then public members of base class become public members of derived class. In private derivation we cannot access public member functions of base class from derived class object but in public derivation we can access public member functions of base class from derived class object.

Protected Access Specifier

- If we require the private data to be visible in the derived class then C++ provides the protected visibility modifier.
- Protected member inherited in public mode becomes protected in derived class, hence it is accessible to the member function of derived class and ready for further inheritance.
- Protected member inherited in private mode becomes private in derived class, hence it is accessible to the member functions of derived class but not available for further inheritance.

```
class ABC
{
   private:
   //visible to member functions
   within the class
    ……….;
    ……….;
   protected:
   //visible to member functions
   of its own and derived class
    ……….;
    ……….;
   public:
   //visible to all the functions
   & objects of the derived class
    ……….;
    ……….;
};
```

Access Specifiers Reloaded

Accessibility	Private	Protected	Public
From Base Class	Yes	Yes	Yes
From Child Class	No	Yes	Yes
Outside any related Class	No	No	Yes

END OF DAY 10

- Thus, Day 10 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 11

Revision of Day 10

- Inheritance
  - Single Inheritance
  - Multilevel Inheritance
  - Multiple Inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance
- Protected Access Specifier

Function Overriding

- When we define a function with same name and arguments in both base class and derived class, then the function is called as **overridden function** and this process is called as **function overriding**.
- Function overriding enables programmer **to change the behaviour of any base class functionality in the derived class**.
- When you call the overridden function by child object the child version is called and when you call the overridden using base object the base version is called.

```
class A
{
    ……….;
    public:
      void getdata();
};
class B : public A
{
    ……….;
    public:
    //overridden function
      void getdata();
      void showdata();
};
```

Function Overriding Example

```
class Base
{
   public:
   void show()
   {
    cout << "Base class\n";
   }
};
class Derived : public Base
{
    public:
    void show()
    {
     cout << "Derived class";
    }
};
```

```
int main()
{
    Base b;
    Derived d;
    b.show();
    d.show();
    return 0;
}
```

The output of the program is:-

Base class

Derived class

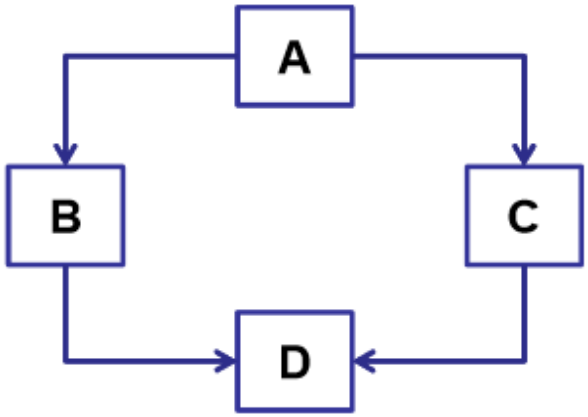
Ambiguity Resolution in Inheritance

```
class ABC
{
    public:
   void display(void)
   {
     cout<<"I am in Class ABC";
   }
};
class XYZ
{
    public:
   void display(void)
   {
     cout<<"I am in Class XYZ";
   }
};
class LMN : public ABC, public XYZ
{
    public:
       void show(void)
       {
         cout<<"I am in class LMN";
       }
};
```

```
int main()
{
    LMN o;
    o.display();//ambiguity error
    //call display function of ABC
    o.ABC::display();
    //call display function of XYZ
    o.XYZ::display();
    o.show();
    return 0;
}
```

- In above example, the class LMN is derived from both class ABC and class XYZ. Both the class ABC and XYZ have same method named display() in them which is inherited by class LMN. Hence when we call display function using object class LMN, the **ambiguity error is displayed**.

Virtual Base Class



- In above inheritance, B and C classes inherit features of A and then as D is derived from B and C, there is **an ambiguity error** because the **compiler does not understand how to pass features of Class A to Class D either using B class or C Class.**

- So to remove the ambiguity you need to make the class B and class C as virtual base classes.

  Class B : public **virtual** A
  Class C : **virtual** public A

- When the class is made as virtual base class, the compiler takes **necessary care to pass only one copy of the members in inherited class** regardless of many inheritance paths exist between the virtual base class and derived class.

Virtual Base Class Example

```
class Student
{
protected:
   int roll_number;
public:
   void get_number(int a)
   {
     roll_number = a;
   }
   void put_number(void)
   {
     cout << "Roll Number: "
     <<roll_number << endl;
   }
};
```

```
class Test : public virtual Student
{
   protected:
   float part1, part2;
   public:
   void get_marks(float x, float y)
   {
     part1 = x;
     part2 = y;
   }
   void put_marks(void)
   {
     cout<<"Markts Obtained: "
     <<endl<< "Part1 = "<<part1<<
     endl<<"Part2 = "<<part2;
   }
};
```

continued on next slide….

```
class Sports : virtual public
Student
{
protected:
   float score;
public:
   void get_score(float s)
   {  score = s;   }
   void put_score(void)
   {
     cout << "Sports Marks = "
     << score << endl;
   }
};
class Result : public Test,
public Sports
{
   float total;
   public:
   void display(void);
};
```

```
void Result :: display(void)
{
   total = part1+part2+score;
   put_number();
   put_marks();
   put_score();
   cout<<"Total Score: "<<total
   <<endl;
}
int main()
{
   Result student1;
   student1.get_number(10);
   student1.get_marks(75.25,82.90);
   student1.get_score(92.80);
   student1.display();
   return 0;
}
```

Behaviour of Constructors in Derived Class

- When we create the object of derived class the default constructor of base class gets called and then default constructor of derived class gets called.
- When you create object of derived class with parameters, still default constructor of base class gets called and parameterized constructor of child class gets called.
- If base class contains constructor more than one argument, then it is compulsory for the derived class to have a constructor and pass the arguments to base class constructor.

Constructors in Single Inheritance (Example)

```
class Base
{
   int a;
    public:
   Base()
   {
     cout << "I am into base's
     default constructor";
   }
   Base(int x)
   {
     cout << "I am into base's
     parameterized constructor";
     this->a = x;
   }
   void show_a()
   {
     cout << endl << "The value of a
     is: " << this->a;
   }
};
```

```
class Derived : public Base
{
   int b;
    public:
   Derived()
   {
     cout <<  endl << "I am in
     child's default constructor";
   }
   Derived(int y, int z):Base(y)
   {
     cout << endl << "I am in child's
     parameterized constructor";
     this->b = z;
   }
   void show_b()
   {
     cout << endl << "The value of b
     is: " << this->b;
   }
};
```

continued on next slide….

```
int main()
{
    Derived d;
    Derived d1(10,20);
    d1.show_b();
    d1.show_a();
    return 0;
}
```

Output:

I am into base's default constructor

I am into child's default constructor

I am into base's parameterized constructor

I am into child's parameterized constructor

The value of b is: 20

The value of a is: 10

Constructors in Multilevel Inheritance (Example)

```
class A
{
   int num1;
   public:
   A()
   {
     cout << "Into Class A's
      default constructor";
   }
   A(int x)
   {
     this->num1= x;
     cout << endl << "Into Class
      A's parameterized
      constructor“ << endl;
     cout << "A= " << this->num1;
   }
};
```

```
class B:public A
{
   int num2;
   public:
   B()
   {
      cout << endl << "Into Class
      B's default constructor";
   }
   B(int l, int m):A(l)
   {
      this->num2= m;
      cout << endl << "In Class
      B's parameterized
      constructor" << endl;
     cout << "B = " << this->num2;
   }
};
```

continued on next slide….

```
class C:public B
{
   int num3;
   public:
   C()
   {
      cout << endl << "Into Class
      C's default constructor";
   }
   C(int p, int q, int r):B(p,q)
   {
      this->num3= r;
      cout << "\nInto Class C's
      parameterized constructor";
      cout << endl;
     cout << "C = " << this->num3;
   }
};
```

```
int main()
{
    C obj1;
    C obj2(10,20,30);
    return 0;
}
```

Output:

Into Class A's default constructor
Into Class B's default constructor
Into Class C's default constructor
Into Class A's parameterized constructor
A = 10
Into Class B's parameterized constructor
B = 20
Into Class C's parameterized constructor
C = 30

END OF DAY 11

- Thus, Day 11 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 12

Revision of Day 11

- Function Overriding
- Ambiguity resolution in function overriding (The diamond problem)
- Behaviour of Constructors in Inheritance
  - In single inheritance
  - In multilevel inheritance

Nested Classes

- When one class is declared inside the other class then the **inner class is called as a nested class**.
- The nested class is member of the outer class.
- The rules of accessing the nested class is same as that of other members of the class.
- The members of outer class does not have any special permission to access the data of the inner class.
- If the inner class is declared as private, then you cannot create object of the inner class outside the outer class.
- If the inner class is declared as public, then you can create the object of the class inside main as below

  Outer :: Inner obj;
- The inner class can have access to the private data members of outer class. To achieve this, you need to pass an outer class object to inner class function.

Nested Classes Example

```
class Outer
{
    public:
    class Inner //nested class
    {
        private:
        int num;

        public:
        void getData(int no)
        {
          this->num = no;
        }

        void showData()
        {
          cout << "The number is:
          " << this->num;
        }
   };
};
```

```
int main()
{
   cout << "Implementation of
   Nested Classes" << endl;
   Outer :: Inner obj;
   obj.getData(14);
   obj.showData();
   return 0;
}
```

- As the Inner class was declared publically inside Outer class, we were able to define it's object using Outer class, inside main function.

END OF DAY 12

- Thus, Day 12 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

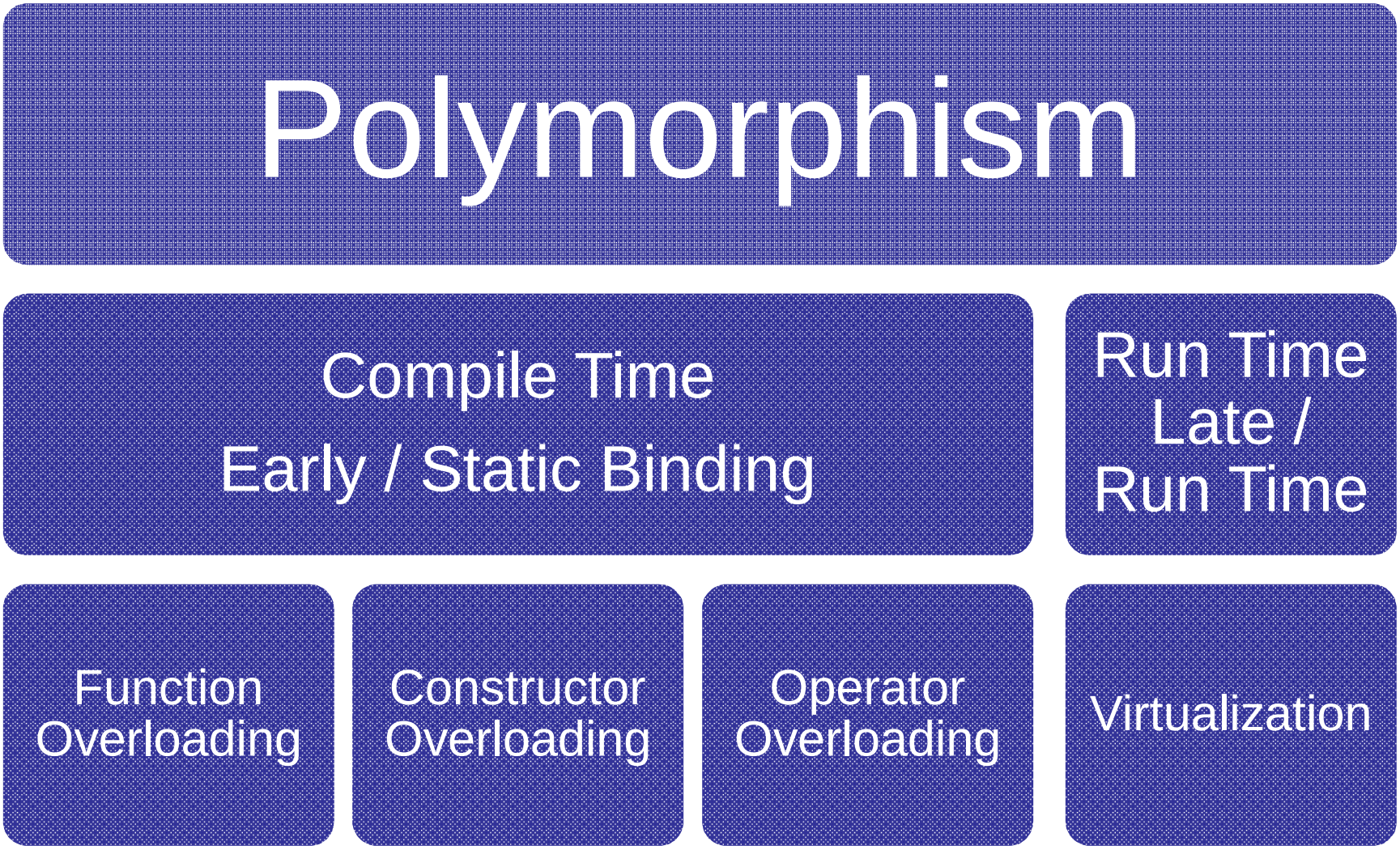
ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 13

Revision of Day 12

- Nested Classes
- Access Outer class data inside Inner class.

Run Time Polymorphism (Virtualization)



Run Time Polymorphism (Virtualization)

- In C++, a **base class pointer** can also point to **derived class object**.
- But the problem while using base class pointer with derived class object is, it can access only those members which are inherited from base class and not the members of derived class.
- If the derived class has a overridden method and if you try to call the derived class method then as the pointer belongs to base class, hence it calls the base version of the overridden function.
- If you want to call child class method in such case, you either need to use **typecasting** or **virtual functions**.
- Runtime polymorphism is also known as **Dynamic Polymorphism** or **Late Binding**. In case of run time polymorphism, the compiler decides which function to call at runtime.
- To achieve run time polymorphism you need to **decalre the function in the base class as virtual function** and the **same function has to be overridden in the derived class**.
- The concept of runtime polymorphism is implemented in c++ using a concept of vTable and *vptr pointer. The vTable contains a *vptr for each base class and the derived classes for virtual function into it. The *vptr points to the overridden function for that particular class.

Virtual Function

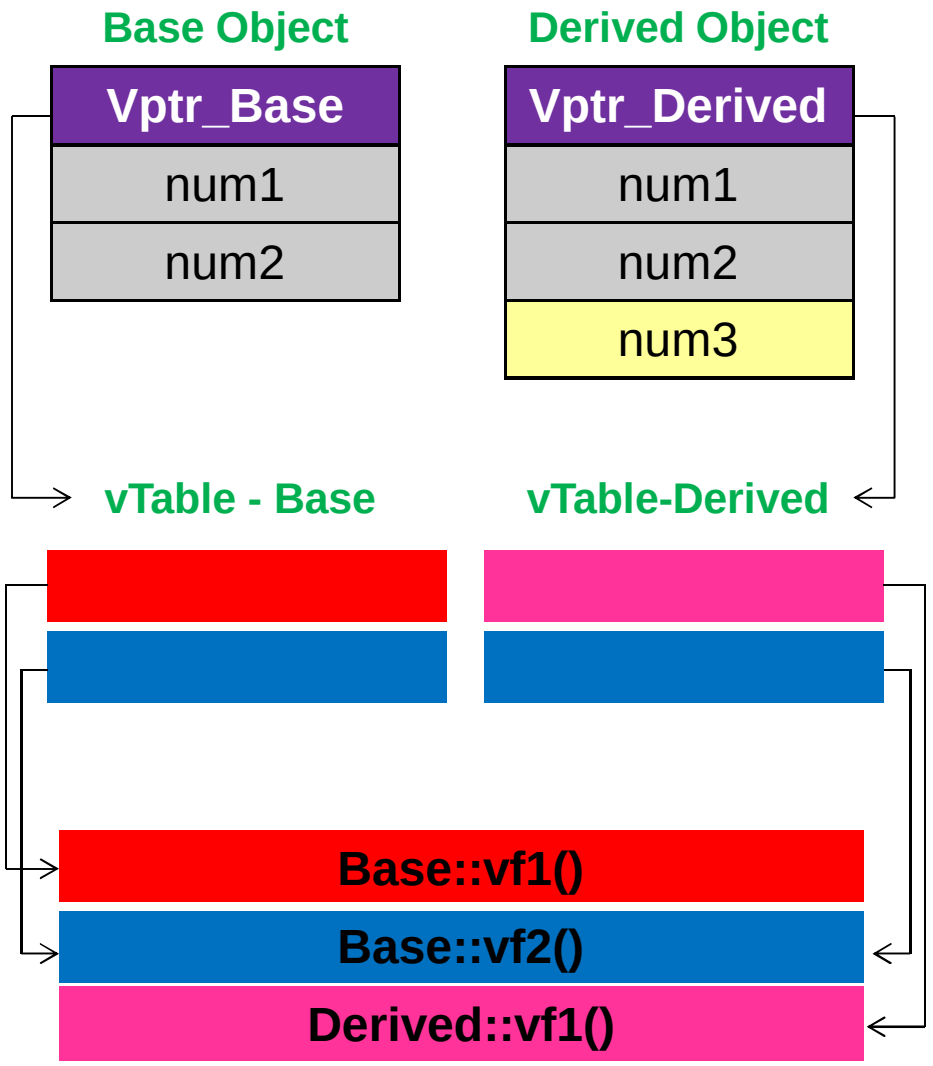
```
class Base
{
    public:
    virtual void show()
    {
       cout << "Base class";
    }
};
class Derived : public Base
{
    public:
    void show()
    {
       cout << "Derived Class";
    }
}
```

```
int main()
{
   //Base class pointer
   Base *b;
   Base bObj;
   b = &bObj;
   b->show();

   Derived dObj;
   b = &dObj;
   b->show(); //Late Binding
   return 0;
}
```

Working of - vptr and vTable

```
class Base
{
   int num1,num2;
   public:
   virtual void vf1();
   virtual void vf2();
   void f1();
   void f2();
   void
};
class Derived : public Base
{
   int num3;
   public:
   void vf1(); //overridden fun
   Void f1(); // overridden fun
   void f3();
};
```



END OF DAY 13

- Thus, Day 13 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 14

Revision of Day 13

- Difference between Compile Time and Run Time Polymorphism
- Working of vptr and vTable.

Abstract Class

- Abstract class is designed to act as base class for other classes.
- It is a design concept in a program development and provides a base upon which other classes may be built.
- You cannot create object of an Abstract class.
- Abstract class is used only to derive other child classes.
- A class with a Pure Virtual Function is called as an Abstract Class

```
Class A
{
   int num1, num2;

   public:
   //pure virtual function
   virtual void add()=0;
   void show();
   void get();
}
```

- It is mandatory to override all the pure virtual functions of base class in the derived class.

Virtual Destructor

```
class Base
{
    public:
    ~Base()
    {
      cout << "Calling ~Base()"
      << endl;
    }
};

class Derived: public Base
{
    private:
    int* m_pnArray;

    public:
```

```
Derived(int nLength)
{
  m_pnArray = new int[nLength];
}
~Derived()
{
   cout<<"Calling ~Derived()”;
   delete[] m_pnArray;
 }
};
int main()
{
   Derived *pDerived = new
   Derived(5);
   Base *pBase = pDerived;
   delete pBase;
   return 0;
}
```

...continued

- Output:

```
Calling ~Base()
```

This is because the pointer created belongs to the base class

...continued

```
class Base
{
    public:
    virtual ~Base()
    {
      cout << "Calling ~Base()"
      << endl;
    }
};

class Derived: public Base
{
    private:
    int* m_pnArray;

    public:
```

```
 Derived(int nLength)
 {
    m_pnArray = new int[nLength];
 }
 virtual ~Derived()
 {
    cout << "Calling ~Derived()";
    delete[] m_pnArray;
 }
};
int main()
{
    Derived *pDerived = new Deri
    ved(5);
    Base *pBase = pDerived;
    delete pBase;
    return 0;
}
```

...continued

- Output:

```
Calling ~Derived()
Calling ~Base()
```

This will happen as the derived destructor would in turn call the base destructor

END OF DAY 14

- Thus, Day 14 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

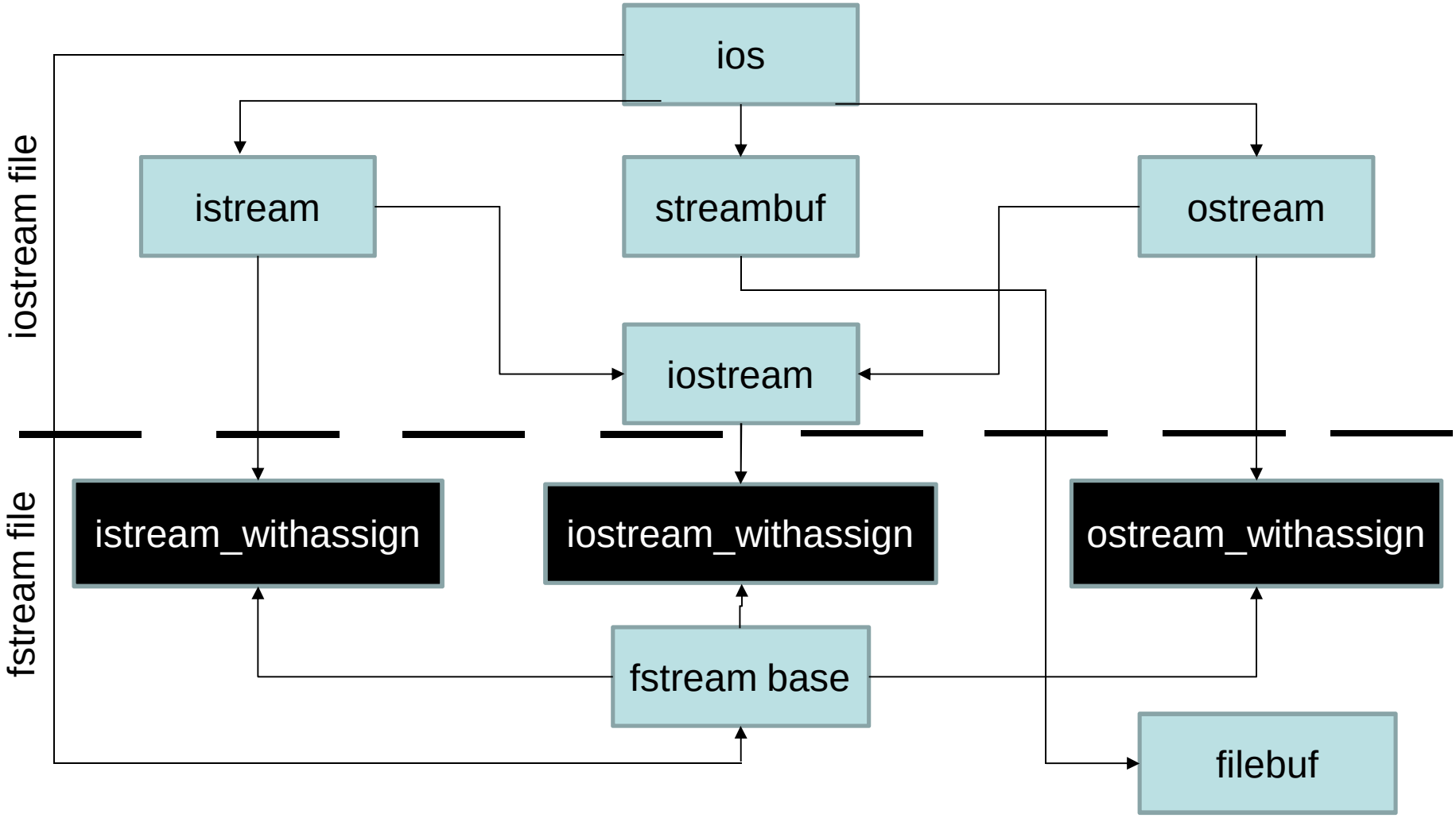
ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 15

Revision of Day 14

- Abstract Class
  - Pure virtual function
- Virtual Destructor

Working on Files



Opening and Closing a File

- There are input and output stream classes for files similar to those for consoles.
- These are ifstream and ofstream respectively.
- To open a file in WRITE mode, we need to an object of the output stream.
- Thus,

```
ofstream
outfile("results.txt");
```

  is the desired statement.

- Similarly, to open a file in READ mode, the syntax is,

```
ifstream
infile("results.txt");
```

- A file can also be opened through a function,

```
infile.open("results.txt");
```

- To close the file, we simply call the close function through the object:

```
infile.close();
```

Writing and Reading from a File

- Writing to a file is done through the overloaded operator (<<).
- e.g.:

```
ofstream outf;
outf.open("Results.txt");
outf << "Hello World! ";
```

- Similarly, reading is done through the overloaded operator (>>).
- e.g.:

```
ifstream inf;
char name[30];
inf.open("Results.txt");
inf >> name;
```

The getline() Function and eof() Function

- File contents can also be read through a member function getline().
- It takes 2 parameters:
  - a string
  - number of characters

```
ifstream inf;
char name[30];
inf.open("results.txt");
inf.getline(name,20);
```

- When reading content from the file, we need to make sure that the End-of-File has not been encountered.
- To make sure we get this, a member function eof() is called to check if the end of file is encountered.
- The eof() returns a 0 if false.

```
ifstream inf("results.txt");
if(inf.eof()) { exit(0); }
```

Opening File in a Specific Mode

streamObj.open("FileName", mode);

Parameter	Meaning
ios::in	Opens the file for Read purpose only
ios::out	Opens file for Write purpose only
ios::app	Appends data to the end of the file
ios::ate	Go to the end of the file after opening. But allows to edit the data anywhere in the file
ios::binary	Opens the file in the Binary Mode
ios::trunc	Deletes the contents of the file if it exists

Accessing the File Pointers

Function	Meaning
seekg()	Moves input pointer to specified location
seekp()	Moves output pointer to specified location
tellg()	Gives current position of input pointer
tellp()	Gives current position of output pointer

seekg(Offset, ReferencePosition);

seekp(Offset, ReferencePosition);

Offset is number of bytes the pointer should move.

ReferencePosition	Meaning
ios::beg	Beginning of the file
Ios::end	End of the file
Ios::cur	Current Position of the pointer

Examples of seekg Function Calls

seekg function call	Meaning
streamObj.seekg(0, ios::beg)	Moves pointer to start of the file
streamObj.seekg(0, ios::cur)	Stays at current position
streamObj.seekg(0, ios::end)	Go to the end of file
streamObj.seekg(m, ios::beg)	Moves to the m+1th byte in the file
streamObj.seekg(m, ios::cur)	Go forward by m bytes from current position
streamObj.seekg(-m, ios::cur)	Go backward by m bytes from current position
streamObj.seekg(-m, ios::end)	Go backword by m bytes from end poistion

END OF DAY 15

- Thus, Day 15 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 16

Revision of Day 15

- Opening the file with constructor
- Opening the file with open() function
- Closing the file with close() function
- Writing the data with << Operator
- Reading the data with >> Operator
- getline() and eof() function
- Opening files with specific modes
- Moving read and write pointer using seek and tell functions.

put() and get() Functions

- Function **put()** writes a single character to the associated stream.
- Function **get()** reads a single character from the associated stream.

```
int main()
{
   char str[50], ch;
   cout<<"Enter any string:";
   cin.getline(str,50);

   int len = strlen(str);
```

```
   fstream file;
   file.open("Data.txt",
   ios::in | ios::out);

   //write data to the file
   for(int i=0; i<len; i++)
        file.put(str[i]);
   //go to the start of the file
   file.seeg(0, ios::beg);
   while(1)
   {
        //read a character from file
        file.get(ch);
        if(file.eof())
            break;
        cout << ch;
   }
   file.close();
   return 0;
}
```

write() and read() Functions

- **write()** and **read()** functions handle the data in the binary format.

```
infile.read((char *) &Variable,
sizeof(Variable));

outfile.write((char *)
&Variable, sizeof(Variable));
```

```
int main()
{
   int weight[]={50,55,105,80};
   int i;

   fstream file;
   file.open("Weight.txt",
   ios::in | ios::out);
   if(file)
      file.write((char *)
      &weight, sizeof(weight));
   else
   {
      cout << "File Does not
      exist";
      exit(0);
   }
   file.seekg(0, ios::beg);
   for(i=0; i<4; i++)
      weight[i] = 0;
   file.read((char *) &weight,
   sizeof(weight));
   for(i=0; i<4; i++)
   {
      cout<<weight[i]<<" ";
   }
   return 0;
}
```

Command Line Arguments (CLA)

- In C++, it is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems and are passed in to the program from the operating system. To use command line arguments in a program, we must see the full declaration of the main function, which previously has accepted no arguments.

  void main(int argc, char *argv[])

- The integer, argc is the **ARG**ument **C**ount. It is the number of arguments passed into the program from the command line, including the name of the program.
- The character pointer, *argv[] is the **ARG**ument **V**alues list. It has elements specified in the previous argument, argc.

Writing from one file to another using CLA

```
int main(int argc, char *argv[])
{
   ifstream fileR;
   ofstream fileW;
   char ch;
   if(argc != 3)
   {
      cout << "Improper no. of
      arguments";
      exit(0);
   }
   fileR.open(argv[1]);
   if(!fileR)
   {
      cout << "Cannot open
      source file";
      exit(0);
   }
   fileW.open(argv[2]);
   if(!fileW)
   {
      cout << "Cannot open
      target file";
      fileR.close();
      exit(0);
   }
   while(!fileR.eof())
   {
      fileR.get(ch);
      fileW.put(ch);
   }
   fileR.close();
   fileW.close();
   cout << "File copied
   successfully";
   return 0;
}
```

How to Run CLA Program in Dev-C++

- Let us consider that you have written a program and saved the C++ file in e:\Sapna\Cpp**Filecopy.cpp**
- Hence, it is recommended that you save both the files, i.e. the file to read from and the file to write to in the same folder. e.g.

  e:\Sapna\Cpp**Source.txt**

  e:\Sapna\Cpp**Target.txt**
- Just compile the program and do not run it
- Press **Win + S**, to open windows search box
- Type **cmd** and press enter
- You will land on to the command prompt.
- Type **cd** and press enter to come to root folder.
- Type **e:** and press enter
- Type **cd Sapna\Cpp** and press enter
- Type, **Filecopy Source.txt Target.txt** and press enter to run the program

END OF DAY 16

- Thus, Day 16 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

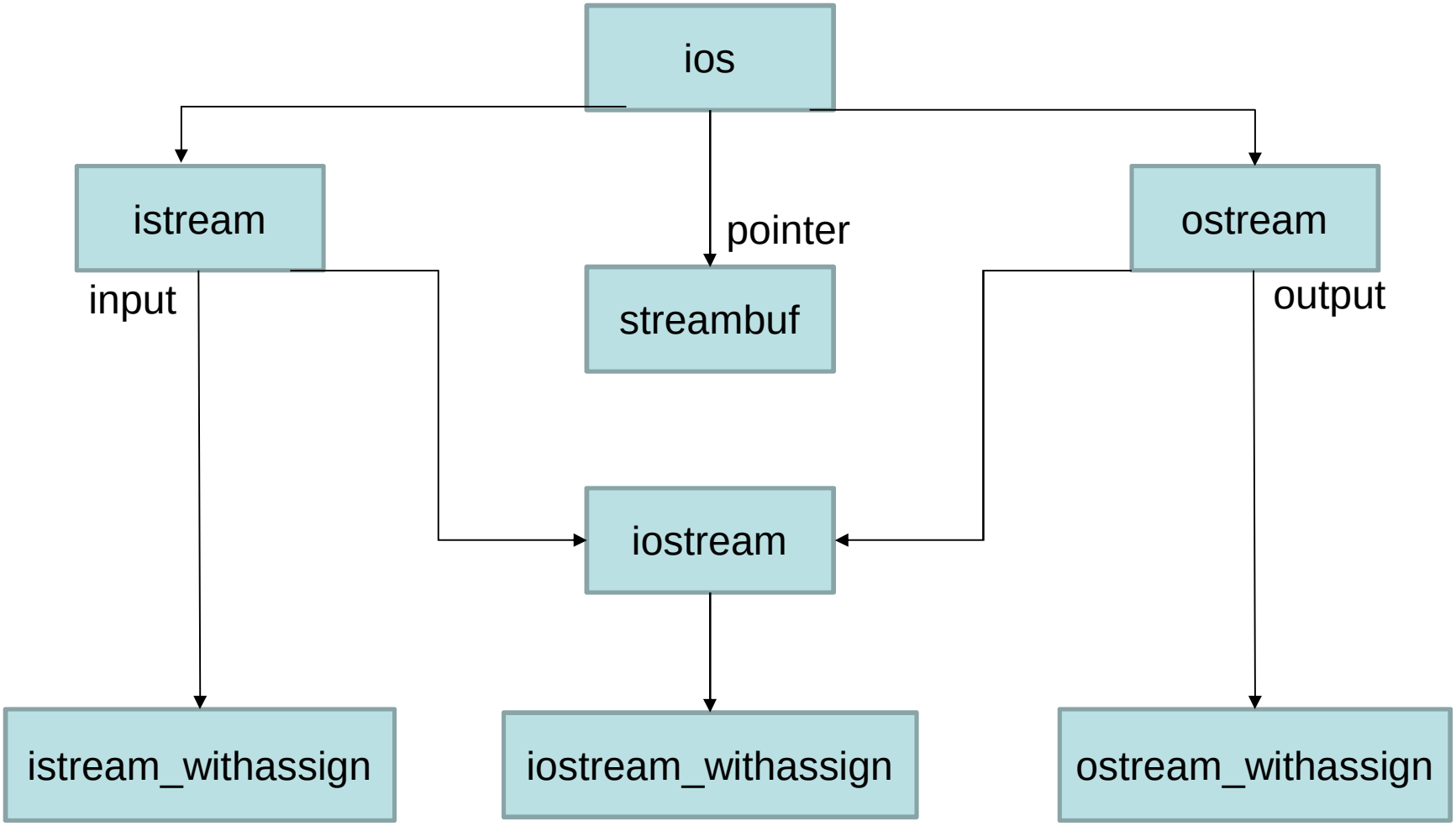
ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 17

Revision of Day 16

- put() and get() Functions
- write() and read() Functions
- Command Line Arguments

Console I/O Operations



put() and get() Functions

- **get()** function is defined in the istream class in order to handle single character inputs.
- It can be implemented as:
  - `cin.get(ch);`

The above method calls the get() function with ch as a parameter.

  - `ch=cin.get();`

The above method calls the get() function without a parameter returning the value to the variable ch.

- **put()** function is defined in the class ostream in order to handle single character outputs.
- It can be implemented as:
  - `cout.put('x');`

The above method calls the put function with a character value 'x' as a parameter.

  - `cout.put(ch);`

The above method calls the put() function with a variable ch.

get() and put() function

```
int main()
{
   char ch;
   cout<<"Enter character:";
   cin.get(ch);
   cout<< "You entered:" <<ch;
   return 0;
}
```

Output:

Enter character: h

You entered: h

```
int main()
{
   char ch;
   cout<<"Enter character:";
   cin.get(ch);
   cout<<"You entered:";
   cout.put(ch);
   return 0;
}
```

Output:

Enter character: h

You entered: h

getline() and write() Functions

- getline() is a function defined to handle strings in the istream class.
- It takes two parameters:
  - A string variable that takes the string
  - Number of characters to be accepted as a string (line).
- write() is a similar function used to write strings onto files. It is defined in the ostream class.
- It takes two parameters:
  - A string that is to be displayed on the console
  - Number of characters to be written.

getline() and write() Functions

```
int main()
{
   char ch[20];
   cout<<"Enter a string:";
   cin.getline(ch,20);
   cout<<"You entered:"<<ch;
   return 0;
}
```

Output:

Enter a string: Envision

You entered: Envision

```
int main()
{
   char ch[20];
   cout<<"Enter a string:";
   cin.getline(ch,20);
   cout<<"You entered:";
   cout.write(ch,20);
   return 0;
}
```

Output:

Enter a string: Envision

You entered: Envision

Formatted Console I/O Operators

- Unformatted console operators are used in order to manipulate or interpret the input and output.
- On the other hand, formatted operators are used in order to give formatting to the output of your program.
- These operators are used just to give a formatting to the input / output.

ios Format Functions

Function	Description
width()	To specify the required field size for displaying an output value
precision()	To specify the number of digits to be displayed after the decimal point of a float value
fill()	To specify a character that is used to fill the unused portion of a field
setf()	To specify format flags that can control the form of output display
unsetf()	To clear the flags specified

Defining Field Width: width()

- We can use the width() function to define the width of a field necessary for the output of an item. Since it is defined within the class ostream, we call it as:

```
cout.width(w);
```

- e.g.:

```
cout.width(5);
cout << 543 << 12 << "\n";
```


```
cout.width(5);
cout << 543;
cout.width(5);
cout << 12;
```

Setting precision: precision()

- By default, the floating point values are printed with six digits after the decimal. However, we can specify the number of digits to be displayed after the decimal through precision();

- e.g.:

```
cout.precision(3);
cout << 3.14159;
cout << 2.50062;
```

- Output:

3.142
2.5 (no trailing zeros)

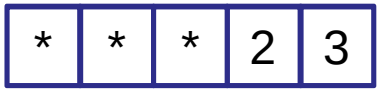
Filling & Padding: fill()

- We have been printing values using larger spaces than required. These are generally blank spaces. With the help of fill(), we can fill the unused positions with the desired character.

- e.g.:

```
cout.fill('*');
cout.width(5);
cout << 23;
```

- Output:



Formatting flags, bit-fields & setf()

- When we use width(), precision() and fill() functions, the data is put by default in right justification.
- We can manage this using this formatting.
- The setf() is a member function of the ios class and can set various flags in order to give proper formatting.

```
cout.setf(arg1, arg2);
```

- Here, `arg1` is one of the formatting flags defined in the class ios. It specifies the format action required for the output.
- Also present is, `arg2`. This specifies which group does the formatting flag belong to. It is called as the bit-field

...continued

Format Required	Flag (arg1)	Bit-field (arg2)
Left-justification	ios::left	ios::adjustfield
Right-justification	ios::right	ios::adjustfield
padding after sign or base indicator	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
fixed point notation	ios::fixed	ios::floatfield
decimal base	ios::doc	ios::basefield
octal base	ios::oct	ios::basefield
hexadecimal base	ios::hex	ios::basefield

END OF DAY 17

- Thus, Day 17 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 18

Revision of Day 17

- get() and put() Functions
- getline() and write() Functions
- Formatted I/O Functions
  - width()
  - precision()
  - fill()
  - setf()
  - unsetf()

Templates

- Templates is a feature in C++ which enables us to define generic classes and functions and thus provides support for generic programming.
- A template can be used to create a family of classes or functions.
- We can template for both:
  - Function Template
  - Class Template

Function Templates

- Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
- A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.
- You can use templates to define functions as well as classes
- Let us see an example:

Example

```
template<class t>
void swap(t &x, t &y)
{
   t temp=x;
   x=y;
   y=temp;
}
int main()
{
   int a, b;
   float c, d;
   cout<<"Enter A, B
   values(integer):";
   cin>>a>>b;
   cout<<"\nA and B before
   swapping :"<<a<<"\t"<<b;
   swap(a,b);
   cout<<"\nA and B after
   swapping  :"<<a<<"\t"<<b;

   cout<<"Enter C, D
   values(float):";
   cin>>c>>d;
   cout<<"\nC and D before
   swapping :"<<c<< "\t"<<d;
   swap(c,d);
   cout<<"\nC and D after
   swapping: "<<c<<"\t"<<d;

   return 0;
}
```

...continued

- Output:

```
Enter A, B values (integer): 10  20
Enter C, D values (float): 2.50  10.80

A and B before swapping: 10 20
A and B after swapping: 20 10

C and D before swapping: 2.50  10.80
C and D after swapping: 10.80  2.50
```

Overloaded Function Templates

```
template<class T>
void f(T x, T y)
{
   cout << "Template";
}
void f(int w, int z)
{
   cout << "Non-template";
}
int main()
{
   f( 1,  2 );
   f('a', 'b');
   f( 1, 4);
   return 0;
}
```

Output:

```
Non-template
Template
Non-template
```

Class Templates

```
template <class T>
class vector
{
   T a;
   int size;
public:
   vector() {  }
   vector(T b)
   {   a=b;  }
};
```

- The class above is defined as a generic class and can take any data type specified.

- In order to define an object of the said class with the variable 'a' as an integer, we write,

```
vector <int> v1;
```

- Similarly, for a character, we say,

```
vector <char> v2;
```

Example

```
template <class T>
class Stack
{
   public:
        Stack();
        void push(T i);
        T pop();
   private:
        int top;
        T st[100];
};
template <class T>
Stack<T>::Stack()  {
   top = -1;
}
template <class T>
void Stack<T>::push(T i)  {
   st[++top] = i;
}
```

```
template <class T>
T Stack<T>::pop()
{
    return st[top--];
}

int main ()
{
    Stack<int> int_stack;
    Stack<string> str_stack;
    int_stack.push(10);
    str_stack.push("Hello");
    str_stack.push("World");
    cout<<int_stack.pop()<<endl;
    cout<<str_stack.pop()<<endl;
    cout<<str_stack.pop()<<endl;
    return 0;
}
```

...continued

- Output:

```
10
World
Hello
```

END OF DAY 18

- Thus, Day 18 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

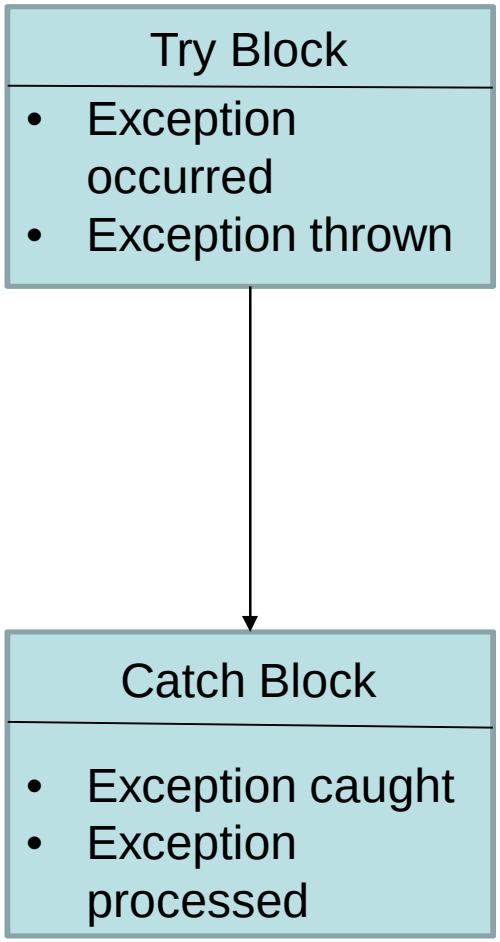
C++ Programming – Day 19

Revision of Day 18

- Templates
  - Function Templates
  - Class Templates

Exception Handling - Concept

- The problems which are not syntax errors nor logical errors are called as Exceptions.
- When these are encountered, the program terminates unexpectedly.
- To avoid this, we use exception handling.



Exception Handling

- try block throwing an exception
- Invoking function throwing an exception
- Multiple catch statements
- catch-all
- Rethrowing an exception

END OF DAY 19

- Thus, Day 19 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 20

Revision of Day 19

- Exception Handling
- try-catch block
- Throwing exception from outside try block
- Catching multiple exceptions
- Catching all exceptions
- Rethrowing exceptions

String Class

- The strings which we have learned till date are array of characters which are terminated by a null character. In c++ the sequence of characters can be represented by an object of a class.
- There is a limititation to array of characters as its size is allocated statically. Hence you cannot increase the memory if required or if you use less amount of memory then the unused memory is wasted. In case of string object, the memory is allocated dynamically and hence no wastage of memory happens.

String Class Functions

Function	Use of the Function
getline()	Stores a stream of characters as entered by the user in the object memory
push_back()	Push the character at the end of the string
pop_back()	Remove last character from the string
capacity()	Returns the capacity allocated to the string, which can be equal to or more than the size of the string. Additional space is allocated so that when the new characters are added to the string, the operations can be done efficiently
resize()	Changes the size of string, the size can be increased or decreased
length()	Finds the length of the string
shrink_to_fit()	Decreases the capacity of the string and makes it equal to its size. This operation is useful to save additional memory if we are sure that no further addition of characters have to be made

String Class Functions

Function	Use of the Function
begin()	Returns an iterator to beginning of the string
end()	Returns an iterator to end of the string
rbegin()	Returns a reverse iterator pointing at the end of string
rend()	Returns a reverse iterator pointing at beginning of string
copy(character array, len, pos)	Copies the substring in target character array mentioned in its arguments. It takes 3 arguments, target char array, length to be copied and starting position in string to start copying
swap()	This function swaps one string with other

Note: Some of the functions do not work with certain IDEs. In Dev C++ following functions do not work – pop_back(), shrink_to_fit()

END OF DAY 20

- Thus, Day 20 of C++ programming ends here.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

- Call us on: +91 8485812611
- E-mail us at: info@ecti.co.in
- Web URL: www.ecti.co.in

- For any suggestions of complaints regarding the theory or programs in the tutorial, write us at: complaints@ecti.co.in

ENVISION®
Computer Training Institute
An **ISO 9001:2015** Certified Company

C++ Programming – Day 21

Revision of Day 20

- String Class
- String Class Functions

Standard Template Library (STL)

- STL means Standard Template Library.
- This programming approach was built by Alexander Stepanov and Meng Lee.
- The approach was developed to store and manipulate/process the data.
- Collection of Some generic classes (data structures) and functions (algorithms) are called as STL.

Components of STL

- Containers - A container is an object that actually stores the data.
- Algorithm - An algorithm is procedure that is used to process the data inside the containers.
- Iterators - Iterator is an object that points to an element in a container. Iterators act as an interface between containers and algorithms.

Types of STL Containers

- **Sequence Containers** - The data is stored in sequence as inserted.
  - Vector
  - list
  - deque
  - forward_list
- **Associative Containers** - The data is stored as key-value pair.
  - Map
  - multimap
  - Set
  - multiset
  - unordered_set
  - unordered_multiset
  - unordered_map
  - unordered_multimap

END OF BASICS IN C++

- Thus, the basics of C++ programming ends here.
- We hope you are satisfied with the theory provided.
- Feel free to share, distribute or use it in any form you wish to. **IT IS FOR YOU**. ☺

END OF BASICS IN C++

For advance C++ programming course or for any doubts in this tutorial, please contact us on any of the following details:

Call us on: 8485812611

E-mail us at: prog@ecti.co.in

Web URL: www.ecti.co.in

For any suggestions of complaints regarding the theory or programs in the tutorial, write us at:

complaints@ecti.co.in