Dr. Ambedkar Institute Of Technology,Bangalore-56

Dept. Of Electronics & Instrumentation Engineering

Pointers, Polymorphism and File Handling

Prepared by,
Shubha P
Assistant Professor,
Dept.of EIE, Dr.AIT

Learning Outcomes

After completing this module, students will be able to:

- Explain pointers to objects.
- Describe runtime polymorphism.
- Implement virtual functions.
- Perform file handling in C++.
- Read and write text and binary files.

Pointers to objects

- A pointer is a variable that stores the memory address of another variable or object.
- Since every object occupies memory, each object has its own address.
- The this pointer is a special pointer available inside a class that points to the address of the current object.
- The this pointer is automatically passed to every non-static member function.
- When accessing the members of an object through a pointer, the arrow operator (->), also called the member access operator, is used.

Example program on pointer object

```
#include<iostream>
class Date
{
   private:
      short int dd, mm, yy;
    public:
      date()
         {
            dd = mm = yy = 0;
         }
       void getdata(int i, int j, int k)
         {
            dd = i;
            mm = j;
            yy = k;
         }
```

Continued..

```
    void prndata(void)
        {
            cout<<"\nData is "<<dd<<"/"<<mm<<"/"<<yy<<"\n";
        }
};
int main()
{
   Date D1; //simple object having type Data:
   Date *dptr; //Pointer Object having type Date:

   cout<<"Initializing data members using the object, with
                values 19, 10, 2016"<<endl;
   D1.getdata(19,10,2016);
```

Continued..

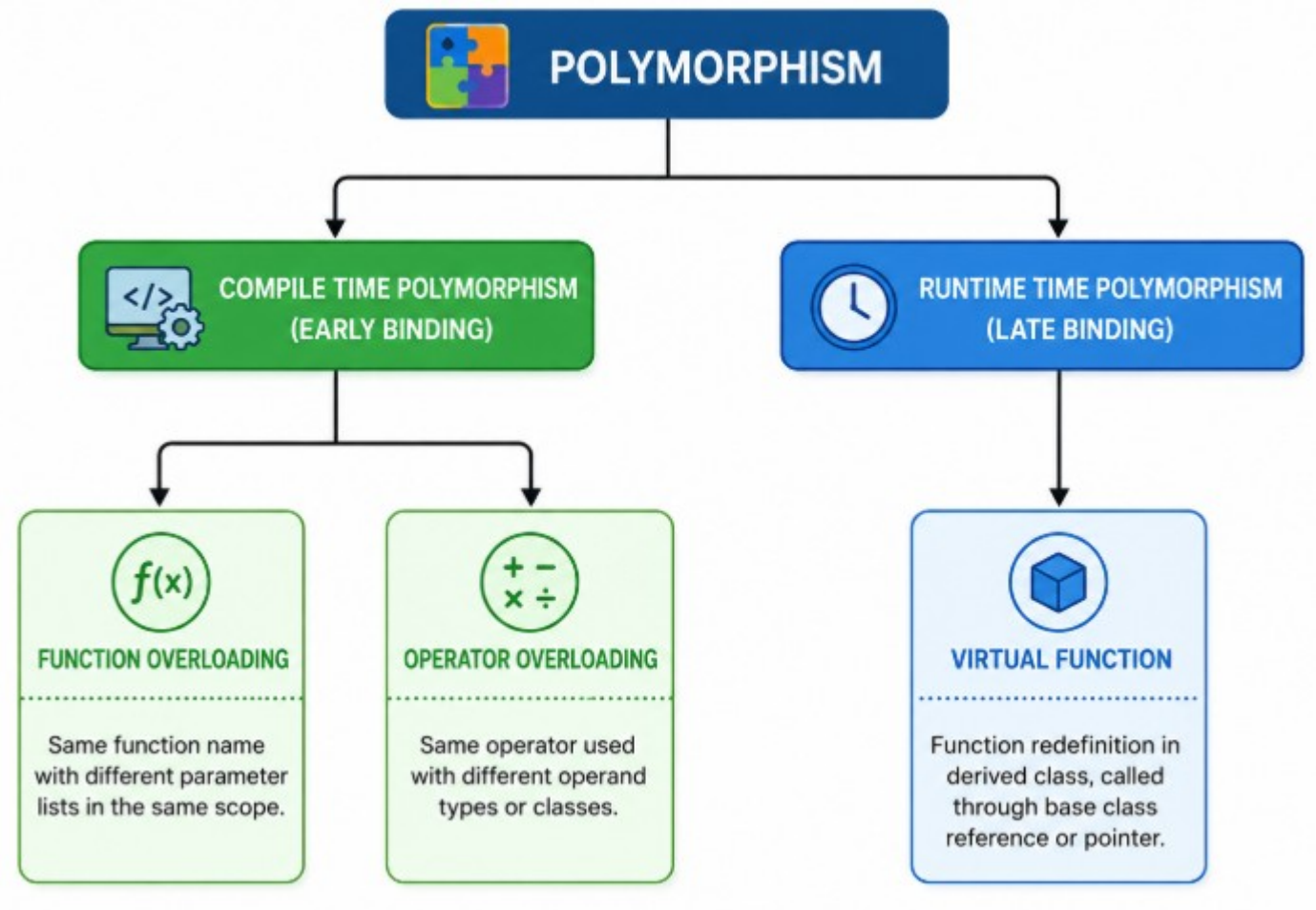
```
cout<<"Printing members using the object ";
    D1.prndata();
     dptr = &D1;
    cout<<"Printing members using the object pointer ";
    dptr->prndata();
    cout<<"\nInitializing data members using the object pointer,
      with values 20, 10, 2016"<<endl;
    dptr->getdata(20, 10, 2016);
    cout<<"printing members using the object ";
    D1.prndata();
    cout<<"Printing members using the object pointer ";
    dptr->prndata();
     return 0;
}
```

Polymorphism

- The term polymorphism means “many forms”.
- Polymorphism allows a single interface to represent different implementations.
- It occurs when multiple classes are related through inheritance.
- In polymorphism, the same function call can produce different behaviors depending on the type of object that invokes the function.
- This enables one interface to perform different operations based on the object being used, improving flexibility and code reusability.

Example: A call to the display() function may execute different implementations in the base class and derived classes, depending on the object that calls it.

Virtual Functions and Polymorphism



Example program…

```
#include <iostream>
using namespace std;

class Shape //base class.
{
    protected:
        int width, height;

    public:
        Shape( int a=0, int b=0) //Constructor.
         {
            width = a;
            height = b;
         }
```

Continued..

```
 int area() //Simple Function to display the result.
        {
          cout << "Shape class area | Base Class | :" <<endl;
          return 0;
        }
};
 class Rectangle: public Shape //Derived Class
{
   public:
       Rectangle( int a=0, int b=0):Shape(a, b) { } //constructor.
       int area () //Simple function to display result.
       {
          cout << "Rectangle class area | Derived Class | :" <<endl;
          return (width * height);
        }
};
```

Continued..

```
class Triangle: public Shape //Derived Class
{
   public:
      Triangle( int a=0, int b=0):Shape(a, b) { }
      int area () //Simple function to display result
       {
         cout << "Triangle class area | Derived Class | :"
<<endl;
         return (width * height / 2);
       }
};
```

Continued..

```
int main( )
{
   Shape *shape;        //pointer Object of type shape.
   Rectangle rec(10,7); //Rectangle Object.
   Triangle tri(10,5);  //Triangle Object.

   shape = &rec;   // store the address of Rectangle
   shape->area();  // call rectangle area using this pointer.

   shape = &tri;  // store the address of Triangle
   shape->area(); // call triangle area using this pointer.

   return 0;
}
```

Contd..

- **o/p**

Shape class area | Base Class | :

Shape class area | Base Class | :

The o/p is incorrect. because that the call of function area() is being set once by the compiler as the version defined in the base class. This is called **static resolution** of the function call. The function call is fixed before the program is executed. Also called **early binding. area()** function is set during compilation of the program.

Example program..

```
#include <iostream>
class Shape //base class
{
    protected:
        int width, height;
     public:
        Shape( int a=0, int b=0) //Constructor.
         {
           width = a;
           height = b;
         }
          virtual int area() //Simple Function to display the result.
         {
           cout << "Shape class area | Base Class | :" <<endl;
          }
};
```

- o/p

Rectangle class area | Derived Class |

Triangle class area | Derived Class |

Virtual Function

- A virtual function is a member function declared in the base class using the virtual keyword and overridden in the derived classes

Abstract class and Pure virtual Function

- An abstract class that cannot be instantiated (cannot create object of that class). However, we can derive a class from it and instantiate object of the derived class.
- Abstract classes are the base class which cannot be instantiated.
- A class containing pure virtual function is known as abstract class.

Contd..

Pure Virtual Function

- A virtual function whose declaration ends with =0 is called a pure virtual function.

Example:

```
class Weapon // abstract class
{
public:
virtual void features() = 0; //Pure virtual function
};
```

Example program on pure virtual function

```
#include <iostream>
class Shape /* Abstract class */
{
  protected:
      float l;
   public:
      void get_data() /*this function is not virtual. */
      {
         cin>>l;
      }
       virtual float area() = 0; /* Pure virtual function */
};
```

Contd..

```
class Square : public Shape //derived class.
{
  public:
      float area()
      {
         return l*l;
      }
};
 class Circle : public Shape //derived class.
{
  public:
      float area()
      {
         return 3.14*l*l;
      }
};
```

Contd..

```
int main()
{
  Square squ; //object of type Square:
  Circle cir; //Object of type Circle:
  shape *sptr;
  sptr=&squ;
  cout<<"Enter length to calculate area of a square: ";
  sptr->get_data();
  cout<<"Area of square: "<<sptr->area();
  sptr=& cir;
  cout<<"\nEnter radius to calculate area of a circle:";
  sptr->get_data();
  cout<<"Area of circle: "<<sptr->area();
   return 0;
}
```

Contd..

- Output:

Enter length to calculate the area of a square: 4 Area of square: 16

Enter radius to calculate the area of a circle: 5 Area of circle: 78.5

Files & Streams

A file is a collection of information stored on a computer's secondary storage that can be saved, accessed, modified, and reused whenever needed.

Each file has a unique name that is used by both the operating system and the user for identification and access.

File Operations

File handling in a program involves three basic steps**.**

Step1:Open the file

- Open an existing file for reading or writing.
- If the file does not exist, it can be created during the opening process.

Step2:Perform file operations

- Read data from the file.
- Write data to the file.
- Or perform both reading and writing, as required.

Step3:Close the file

- Close the file after all operations are completed.
- Closing the file ensures that all changes are saved and system resources are released.

NEED FOR DATA FILES

- Many real-life applications require handling and storing large amounts of data.
- Earlier, arrays were commonly used to store large amounts of data during program execution.
- Arrays store data in Random Access Memory (RAM).
- Data stored in arrays exists only while the program is running.
- Once the program terminates, all data stored in arrays is lost.
- Arrays cannot be used for permanent data storage.
- To store data permanently for future use, files are used.
- Files allow data to be stored on secondary storage devices (such as hard disks or SSDs), making it available even after the program ends.

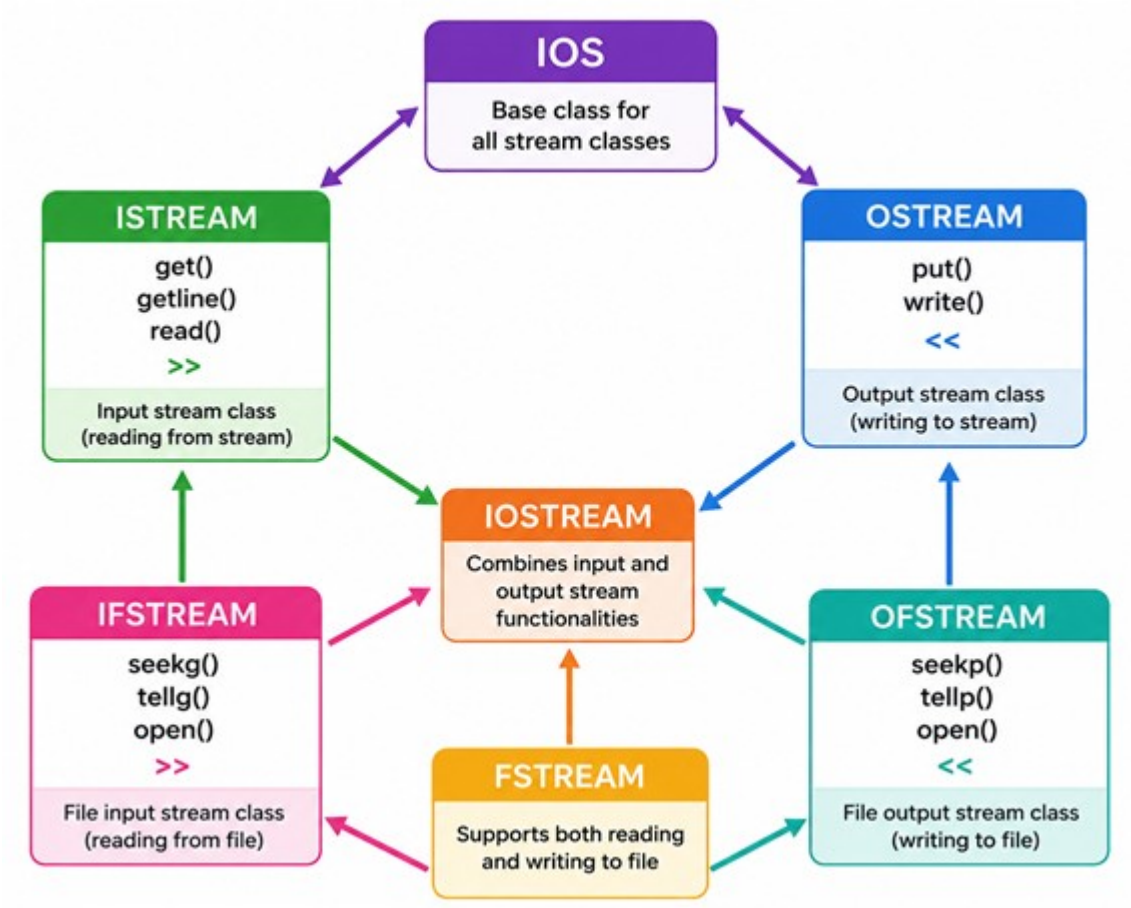
Difference between Arrays and Files

Arrays	Files
Arrays are stored in RAM.	Files are stored on Hard Disk
Data is stored temporarily	Data is stored permanently
Arrays have fixed size	File can have variable size
Arrays can not be used to share data between programs	Files can be used to share data between programs.

File Streams in C++

- C++ uses file streams to provide an interface between the program and the file
- A stream is defined as the flow of data between a program and an input/output device.
- **Types of File Streams**
- **Output Stream**
  - Controls the flow of data from the program to the file.
  - Used to write data into a file.
- **Input Stream**
  - Controls the flow of data **from the file to the program**.
  - Used to **read data** from a file.

Continued..


IOS
Base class for all stream classes
ISTREAM
get()
getline()
read()
>>
Input stream class (reading from stream)
OSTREAM
put()
write()
<<
Output stream class (writing to stream)
IOSTREAM
Combines input and output stream functionalities
IFSTREAM
seekg()
tellg()
open()
>>
File input stream class (reading from file)
FSTREAM
Supports both reading and writing to file
OFSTREAM
seekp()
tellp()
open()
<<
File output stream class (writing to file)

Continued..

- Every stream in C++ is associated with a specific class.
- Each class provides the definitions and member functions (methods) required to perform file operations.
- The commonly used file stream classes are:
  - ifstream Used to read data from a file.
  - ofstream Used to write data to a file.
  - fstream Used for both reading from and writing to a file.
- These classes are declared in the <fstream> header file

Text files and Binary files

1.Text Files

- Store data in the form of ASCII (or Unicode) characters.
- Each line in a text file ends with a special End-of-Line (EOL) character.
- During file operations, the system may perform character translations (such as newline conversion).
- Are human-readable and can be opened using a text editor.

2.Binary Files

- Store data in binary format (0s and 1s).
- Do not use End-of-Line (EOL) characters.
- No character translation occurs during reading or writing.
- Are not human-readable and are primarily used for efficient storage and faster processing of data.

Opening Files

1: Opening a File Using a Constructor

- This method is suitable when only one file is to be opened using a file stream.
- Here, create an object of the required stream class and initialize it with the file name.
- The constructor automatically opens the specified file.

Example1 : Opening a File for Writing

ofstream fout(“student.txt");

Example 2: Opening a File for Reading

ifstream fin(“student.txt");

Continued..

2) **Using open() function**: This method is useful when multiple files are opened sequentially using a single stream object.

eg. ifstream fin; //creates input stream

- fin.open("marks.txt"); // associates marks.txt to this stream
- fin.close(); // closes the file
- fin.open("employee.txt"); // associates the input stream with file employee.txt

Closing Files

- File connections are automatically closed when the corresponding input or output stream object goes out of scope or is destroyed.
- A file can be closed explicitly using close() member function
- fin.close();
- Closing a file flushes the stream buffer, ensuring that any remaining buffered data is transferred to its appropriate destination.
- For an input stream, buffered data is delivered to the program, while for an output stream, buffered data is written to the disk file before the file is closed.

FILE MODES

File Mode	Description
ios::in	Opens a file for reading. The file pointer is positioned at the beginning of the file.
ios::out	Opens a file for writing. If the file already exists, its existing contents are erased by default before writing.
ios::app	Opens a file in append mode. The file pointer is placed at the end of the file, and new data is always added after the existing content.
ios::ate	Opens a file with the file pointer initially positioned at the end of the file. Unlike ios::app, the file pointer can be moved to any location for reading or writing.
ios::binary	Opens a file in binary mode. In this mode, data is stored without any character translation. By default, files are opened in text mode.

Note: Two or more modes can be combined using the bitwise operator |

Text File Functions

Reading/writing a single character from/to file :

- **get() –** read a single character from text file and store in a buffer.

e.g **file.get(ch);**

- **put() -** writing a single character in text file.

e.g. **file.put(ch);**

Continued..

Reading/writing a line from/to file:

- **getline() -** read a line of text from text file stored in a buffer.

e.g **file.getline(s, 80,"\n");**

- **<<(insertion operator) –** write a line to a file.
- **fin<<s;**

Continued..

- **Reading/writing a word from/to file:**

```
char ch[20];
fin.getline(ch,20, ' ');
```

- The >> **(extraction) operator** is used to read a word from a text file.
- The << **(insertion) operator** is used to write a word to a text file.

Note: >> (extraction) reads input only until it encounters a whitespace character. Therefore, it stores only the first word in the variable ch.

Opening a File at Declaration

- **fstream dataFile(“test.txt”, ios::in | ios::out);**

```
//programming example
#include <iostream>
#include <fstream>
int main()
{
    fstream dataFile(“test.txt”, ios::in | ios :: out);
    cout << "The file test.txt was opened.\n";
    return 0;
}
```

- **O/p**

The file test.txt was opened.

Testing for Open Errors

```
dataFile. open(“test.txt”, ios::in);
if (!dataFile)
{
cout << “Error opening file.\n”;
}
```

OR

```
dataFile.open(“test.txt”, ios::in);
if (dataFile.fail())
{
cout << “Error opening file.\n”;
}
```

Closing a File

- **A file should be closed when a program is finished using it.**

```
//example program
#include <fstream>
int main()
{
    fstream dataFile;
    dataFile.open(“testfile.txt”, ios::out);
    if (!dataFile)
    {
    cout << "File open error!" << endl;
    return;
    }
    cout << "File was created successfully.\n";
    cout << "Now closing the file.\n";
    dataFile.close();
    return 0;
}
```

Continued..

Output

File was created successfully.

Now closing the file.

This program uses the << (insertion) operator to write information to a file .

```
#include <iostream>
#include <fstream>
int main()
{
   fstream dataFile;
   dataFile.open(“test.txt”, ios::out);
   if (!dataFile)
   {
   cout << "File open error!" << endl;
   return 0;
}
```

Continued..

```
cout << "File opened successfully.\n";
cout << "Now writing information to the file.";
dataFile << "Jones\n";
dataFile << "Smith\n";
dataFile << "Willis\n";
dataFile << "Davis\n";
dataFile.close();
cout << "Done.\n";
}
```

Continued..

O/P

File opened successfully.

Now writing information to the file.

Done.

Output to File test.txt

Jones

Smith

Willis

Davis

//A Program To Create A Text File

```
#include<fstream>
int main()
{
   ofstream fout(“FV.txt”);
   fout<<“ creating a new text file\n”;
   fout<<“this file contains vegetables and fruits information \n”;
   fout.close();
   return 0;
}
```

The above program will create a text file “FV.txt” and store two lines in it.

//A Program To Read A Text File //Character By Character

```
#include<fstream>
int main()
{
        ifstream fin("students.txt");
        char ch;
        while(!fin.eof())
        {
            fin.get(ch);
            cout<<ch;
        }
      fin.close();
return 0;
}
```

Note:The above program will read a text file "students.txt" one character at a time and display it on the screen.

//A Program To Read A Text //File Word By Word

```
#include<fstream>
int main()
{
        ifstream fin("employee.txt");
       char ch[20];
       while(!fin.eof())
      {
         fin>>ch;
         cout<<ch;
      }
       fin.close();
      return 0;
}
```

//A Program To Read A Text //File Line By Line

```
#include<fstream>
int main()
{
      ifstream fin(“employee.txt”);
      char ch[80];
      while(!fin.eof())
      {
            fin.getline(ch,80,”\n”);
            cout<<ch;
      }
   fin.close();
   return 0;
}
```

// This program writes information to a file, closes the file,
// then reopens it and appends more information.

```
#include <iostream>
#include <fstream>
int main()
{
   fstream dataFile;
   dataFile.open(“test.txt”, ios::out);
   dataFile << "Jones\n";
   dataFile << "Smith\n";
   dataFile.close();
   dataFile.open(“test.txt”, ios::app);
   dataFile << "Willis\n”;
   dataFile << "Davis\n";
   dataFile.close();
   return 0;
}
```

Continued..

- **Output to File test.txt**

Jones

Smith

Willis

Davis

// This program uses the >> operator to read information from a file.

```
#include <iostream>
#include <fstream>
int main()
{
   fstream dataFile;
   dataFile.open(“test.txt”, ios::in);
   if (!dataFile)
   {
   cout << "File open error!" << endl;
   return;
   }
   cout << "File opened successfully.\n";
   cout << "Now reading information from the file.\n\n";
```

Continued..

```
    for (int i= 0; i < 4; i++)
    {
    dataFile >> name;
    cout << name << endl;
    }
    dataFile.close();
    cout << "\nDone.\n";
    return 0;
}
```

Continued..

O/P

File opened successfully.

Now reading information from the file.

Jones

Smith

Willis

Davis

Done.

eof() member function

- The eof() (End-of-File) member function is used to check whether the end of a file has been reached.
- It returns true when there is no more data available to read from the file; otherwise it returns false.

```
if (inFile.eof())
inFile.close();
```

// program using eof

```
#include <iostream>
#include <fstream>
int main()
{
fstream dataFile;
dataFile.open(“test.txt”, ios::in);
if (!dataFile)
{
cout << "File open error!" << endl;
return;
}
cout << "File opened successfully.\n";
cout << "Now reading information from the file.\n\n";
```

Continued..

```
dataFile >> name; // Read first name from the file
while (!dataFile.eof())
{
   cout << name << endl;     //printing name on the
                                          //screen
        dataFile >> name;    //read another name from the
                                          // file
}
dataFile.close();
cout << "\nDone.\n";
return 0;
}
```

Continued..

File opened successfully.
Now reading information from the file.
Jones
Smith
Willis
Davis
Done.

Module Summary

- ✔ Pointer to Objects
- ✔ this Pointer
- ✔ Polymorphism
- ✔ Virtual Function
- ✔ Pure Virtual Function
- ✔ Abstract Class
- ✔ File Handling
- ✔ File Streams
- ✔ File Modes
- ✔ Text Files
- ✔ Binary Files
- ✔ File Functions
- ✔ eof()