Object Oriented Concept and Programming Unit –4

–Madhavi Dave

Runtime Polymorphism by Virtual Functions

Introduction

- Polymorphism : It is one of the crucial features of OOP. It simply means ‘one name, multiple forms’.
- The overloaded member functions are selected for invoking by matching the arguments, both type and number.
- The information is known to the compiler at compile time.
- This is called early binding or static binding or static linking and polymorphism.

- When two functions with the same name are used in two different classes, they can be defined using class resolution operator.
- Here the function is not overloaded and so static binding does not apply.
- If we want a member function could be selected while the program is running , then we have to use virtual function.
- This is known as run time polymorphism.

```
// Virtual Functions and // Run-time Polymorphism
#include <iostream.h>
    // base class
class base
{ public: int a; };
    // derived class
class derived:public base
{ public: int b; };
    // main
void main()
{ base b; derived d;
    // base class pointer
  base *bptr;
  // pointer pointing  to base's object
  bptr=&b;
  bptr->a=10;
// pointer pointing  to derived's object
   bptr=&d; // still is able to access  the members of the base  class
   bptr->a=100; }
```

```
// Using Virtual functions to  achieve run-time Polymorphism
#include <iostream.h>
// base class
class base
 {
        public:
       virtual void func( )
       { cout<<‘Base’s func( )\n‘; }
 };
// derived class
class derived:public base
{
   public:
    void func( )
   {
          cout<<‘Derived’s func( )\n‘; }
   };
```

```
// main
void main()
{
  int ch=0;
  base b;
 derived d;
// base class pointer
  base *bptr;
 while(ch!=3)
 {
    cout<<'1| Call Base's func\n';
    cout<<'2| Call Derived's func\n';
   cout<<'3| Quit\n'; cin||ch;
    switch(ch)
    {
        case 1: // point to base's object
        bptr=&b; break;
        case 2: // point tp derived's object
        bptr=&d; break;
       default: bptr=&b;
    } // call whichever function // user has chosen to call
bptr-|func(); } }
```

Pointers

- To achieve the dynamic binding, it is required to use the pointer to objects and virtual functions.
- Pointer is a derived data type that refers to another data variable by storing the variable's memory address rather than the data.
- Like C, a pointer variable can also be used to refer another pointer in c++.
- Pointers provide an alternative approach to access other data objects.

Pointer to Object

- Normal Objects cannot help in achieving runtime polymorphism. It is possible to achieve it only by setting or defining pointer to objects,
- Pointer to object is a variable containing an address of an object.
- It is similar to a pointer to any other variable. We can use normal address–of operator to get the address of an object.
- We can define a pointer to an object and can assign it the address of an object.

Declaring Pointers

- The declaration of a pointer is based on the data type of the variables it points to.
- Syntax : – datat–type *pointerVariable;
- A pointer is able to point to only one data type at a specific time.
- Eg :– int *ptr; // declare
  ptr = &a; // initialize

```
Class Demo{ ----}
Demo objDemo;
Demo *ptrObjDemo;
ptrObjDemo = &objDemo;
```

this pointer

- The this pointer is a pointer that represent an object that invokes a member function.
- This is a pointer that points to an object for which this function was called.
- This pointer is automatically passed to a member function when it is called.
- This pointer acts as an implicit argument to all the member functions.
- Eg: objDemo.Display( ) function call, the value of this pointer contains the address of objDemo.
- this pointer is poiner to objDemo.

Virtual Functions

- Virtual functions are special.
- Using virtual functions we are able to point to any object of a derived class using a base pointer and can manipulate that object.
- The class has an additional storage requirement when at least one virtual function is defined.
- A table is created additionally to store pointers to all virtual functions available to all objects of class.
- This is known to a virtual table.
- Syntax Constraints on Virtual functions:
- 1) Function should precede virtual keyword in the base class
- .

2) The function name in derived class must have same name as of virtual function defined in base class and same prototype

3) The function in derived class need not to be preceded by virtual keyword.

4) If function is not defined with same name then base class function will be called.

5) The virtual function must be defined in the base class, it may have an empty body though.

6) Polymorphism is achieved (executing the function of the object which is pointed to) only using pointers to the base class. It is not possible using objects.

7) Virtual Constructors are not possible.

Virtual Destructors

- Virtual destructors are needed for proper deletion of objects of derived class, when pointed to by a base class pointer.
- If do not define virtual destructors, only the base class subobject is deleted, and the remaining portion of the derived class object is not deleted.
- We know destructor name cannot be same in base and derived class as it should same as class name.
- A class can have only one destructor and the derived class destructor is the function called when delete is invoked with a base class pointer.

Pure Virtual Functions

- When a class does not have any object, there is no need to have functions for it as there is no object to utilize those functions.
- When we write "$=0$" in place of the function body after function header, the function is said to be pure virtual function.
- In this function need not have any body.
- So instead of defining the function with any empty body, if we define it as pure virtual function, the function forces the base class to be abstract and it has to be derived.
- And abstract classes do not have object.

Static Invocation of virtual Functions

- Virtual functions can also be invoked statically.
- When we use a class name :: ptr–ا virtual function mechanism, they are called statically.

Pointers expressions and Arithmetic

- C++ allows pointers to perform the following arithmetic operations:
  - A pointer can be incremented (++) or decremented (--).
  - Any integer can be added to or subtracted from a pointer
  - One pointer can be subtracted from another.

Eg:
```
int a[10];
int *aptr;
aptr = &a[0];
```

We cn do aptr++ or aptr– to increment or decrement a pointer and it moves to the next memory address.

Pointers with Arrays and Strings

- Accessing an array with pointers is simpler than accessing the array index.
- Arrays refer to a block of memory space, but pointers do not refer to any section of memory.
- The memory addresses of arrays cannot be changed, whereas the content of the pointer variables, such as the memory addresses that it refer to can be changed.
- Eg :– Pointer to array

```
int *nptr;
nptr = number[0];
```

Nptr points to the 1^{st} element of an array.

Array of pointers

- An array of pointers represent the collection of addresses.
- An array of pointers point to an array of data items.
- Each element of the pointer array points to an item of the data array.
- Data items can be accessed either directly or by dereferencing the elements of pointer array

Pointers to Functions

- The pointer to function is known as callback function. We can use these function pointers to refer to a function.
- Using function pointers, we can allow a c++ program to select a function dynamically at run time.
- We can also pass a function as an argument to another function (as pointer).
- There are two types of function pointers, function that points to static member functions and function pointers that point to non-static member functions
- For non-static member function requires hidden argument.

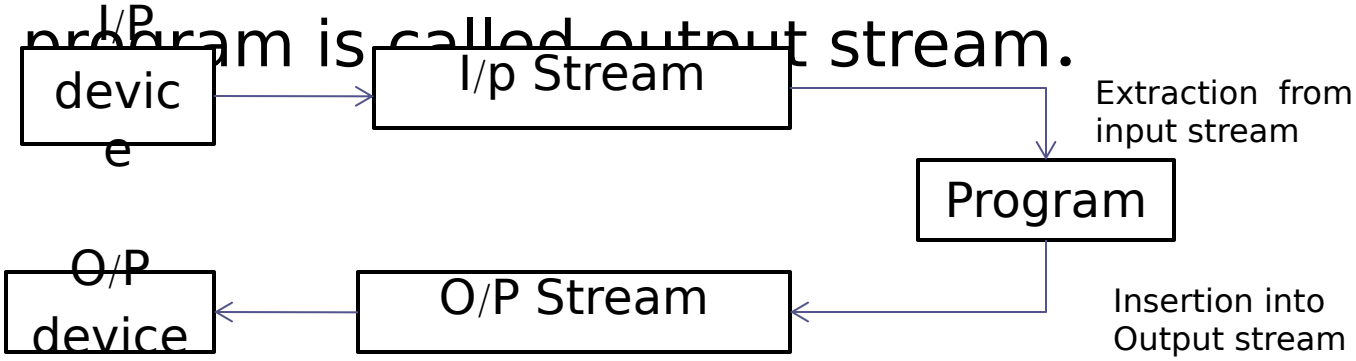
Managing Console i/o operations and Working with files

Introduction

- C++ has rich set of I/o functions and operations to format the o/p and print in the desired form.
- C++ uses the concept of stream and stream classes to implement its I/O operations with console and files.

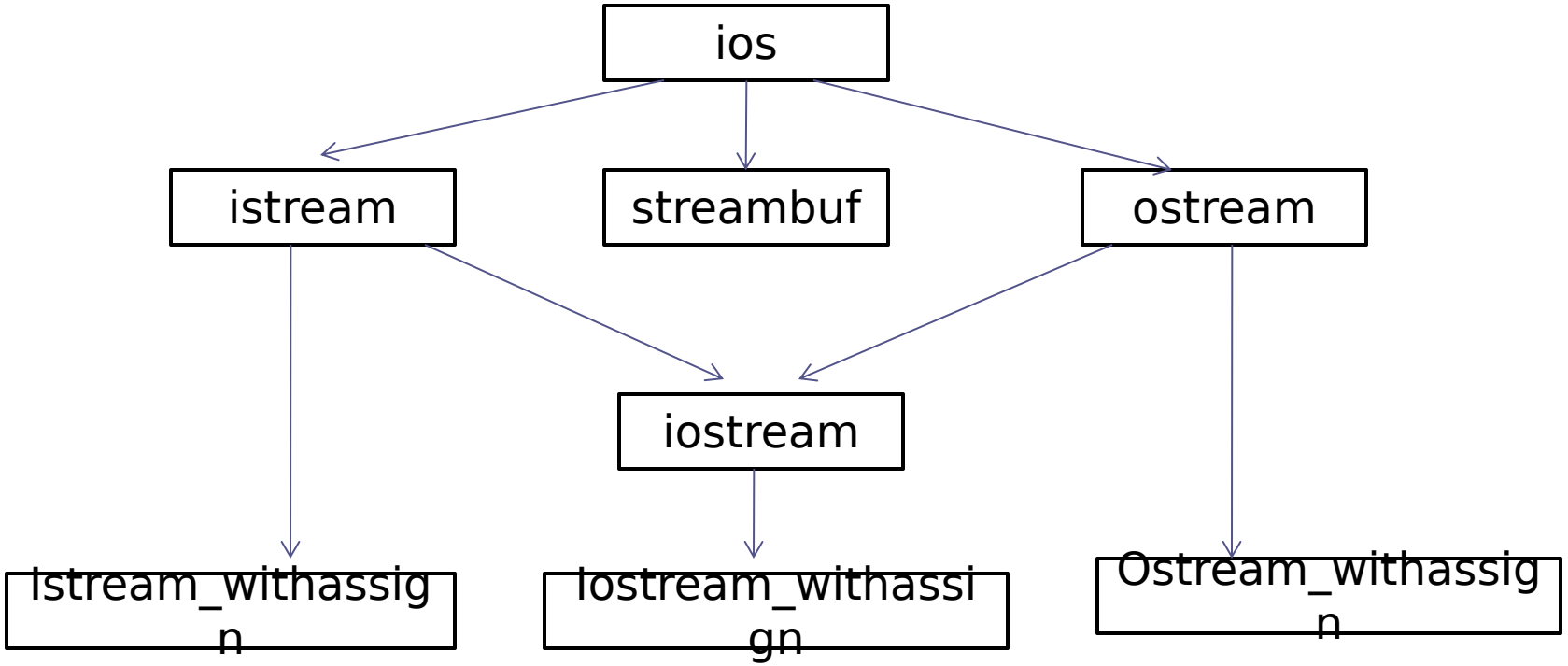
C++ Streams

- A stream is a sequence of bytes. It acts as a interface.
- It acts as a source from which the input data can be obtained or as a destination to which the output can be sent.
- The source stream that provides data to the program is called the input stream and the destination stream that receives output from program is called output stream.



C++ Stream Classes

- The c++ I/O system contains a hierarchy of classes that are used to define various streams to deal with console and files.
- These classes are called stream classes.



Put( ) and get( ) functions

- Get( ) and put( ) are the member functions of istream and ostream to handle the single character input/output operations.
- Two types of get( ) functions are get(char *) and get(void).
- They both can be used to fetch a character including the blank space, tab and the newline character.
- The get(char *) version assigns the input character to its argument and the get(void) version returns the input character. Eg:

```
Char ch;
Cin.get(ch);                    // get character from keyboard and assign to ch
While (ch != '\n')
{
Cout << ch;                     // display character on screen and get
```

- The get(void) version returns the input character.
- Eg
  - Char ch;
  - Ch = cin.get( ) // value assigned to ch.

The function put( ) is a member of ostream class and can be used to output a line of text, character by character.

Eg:

```
cout.put('c');        //displays c
cout.put (c );        // display value in c.
```

Getline( ) and write( ) functions

- The getline( ) function reads a whole line of text that ends with a newline character.
- This function can be invoked by using the object cin as follows:
- Cin.getline(line,size);
- This function reads character input into the variable line.
- The reading is terminated as soon as the newline character '\n' is encountered or size–1 characters are read. The newline character is replaced by null.
- Eg:
  - Char name[20];
  - Cin.getline(name,20);

- The write( ) function displays an entire line and has the following form.

- Cout.write(line,size)

- The first argument line is the name of the string to be displayed and the second indicates the size (no of characters to display).
- But it does not stop displaying the characters automatically when the null character is encountered.
- If the size is greater than the length of line, then it displays beyond the bounds of line.

Formatted console I/O operations

- C++ supports a number of features that could be used for formatting the output.
- Ios class functions and flags.
- Manipulators.
- User–defined output functions

Function	Task
Width( )	To specify the required field size for displaying and output value
Precision( )	To specify the no of digits to be displayed after decimal point
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 o/p display like left or right justification
Unsetf( )	To clear the flags specified.

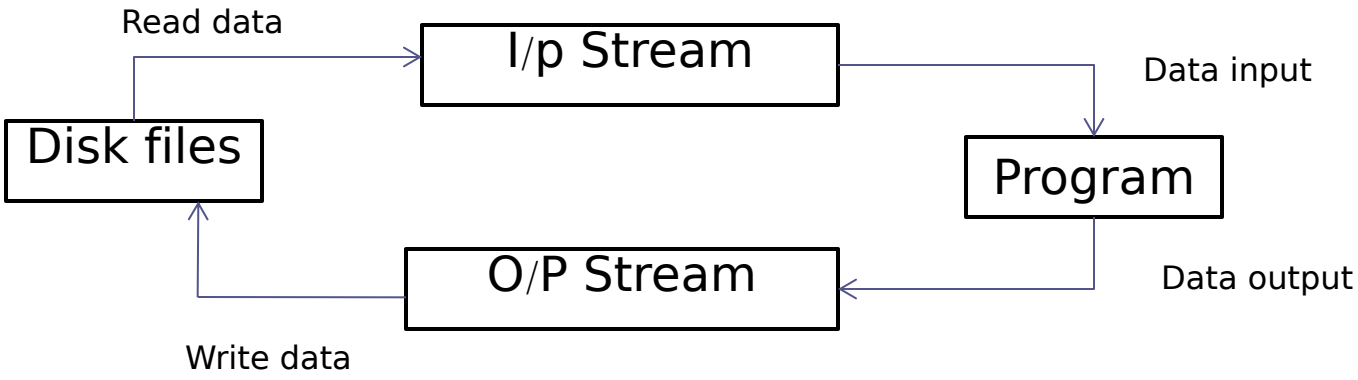
- Manipulators : They are special functions that can be included in the I/O statements to alter the format parameters of a stream.
- File iomap should be included to acces the manipulators.

Manipulators	Ios Function
Setw( )	Width( )
Setprecision	Precision( )
Setfill( )	Fill( )
Setiosflags( )	Setf( )
Resetiosflags( )	Unsetf( )

Files

- A file is a collection of related data stored in a particular area on the disk.
- Programs can be designed to perform the read and write operations on these files.
- Program can involve
  - Data transfer between the console unit and the program.
  - Data transfer between the program and disk file.

- File streams is used as an interface between the programs and the files.
- The stream that supplies data to the program is known as input stream and the one that receives data from the program is known as output stream.
- Inputs stream extracts data from file and output stream inserts data to the file.



Classes for File Stream operations

- The set of classes that define the file handling methods are ifstream, ofstream and fstream.
- These classes are derived from fstreambase and from the corresponding iostream class.
- These classes, designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.

Opening and closing files

- Before using any file we need to decide the following things :
  - Proper name for the file.
  - Data type and structure.
  - Purpose.
  - Opening method.

A file can be opened in two ways :

1) Using the constructor function of the class.
2) Using the member function open() of the class.

Using Constructor

- Using constructor involves the following steps:

1) Create a file stream object to manage the stream using the appropriate class.

   i.e Class ofstream is used to create the output stream and class ifstream is used to create the input stream.

2) Initialize the file object with the desired filename.

   Eg: ofstream outputFile("test");

- Similarly to open for reading
  - Eg: ifstream inputFile(“data”)
- This statement declares inputFile as an ifstream object and attaches it to the file data for reading input.
- We can also use the same file for both output and input or for reading and writing data.
- The connection with a file is automatically closed when stream object expires or use filenameObject.close( ).

```
Ofstream outputfile (“data”);
Ifstream inputfile(“data”);
```

- // program to write into file.

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

int main ()
{
    ofstream myfile;
   myfile.open ('example.txt');
   myfile << 'Writing this to a file.\n';
   myfile.close();
   return 0;
}
```

```
#include <iostream>
 #include <fstream>
int main ()
{
   ofstream myfile ('example.txt');
   If (myfile.is_open())
   {
          myfile << 'This is a line.\n';
          myfile << 'This is another line.\n';
         myfile.close();
    }
   else cout << 'Unable to open file';
    return 0;
 }
```

- Detection of end of file is necessary for preventing any attempt to read data from the file.
- If we use while(fin), an ifstream object returns a value 0 if any error occurs in the file operation including the end-of-file condition. And the while loop terminates when fin returns value 0.
- Eof( ) is a member function of ios class which can also be used to detect the end of file condition.
- It returns a non-zero value if the end-of-file condition is encountered, and 0 otherwise.
- Eg:

```
if(fin.eof( ) !=0 )
{
   Exit( 1 );
}
```

File Modes

Parameter	Meaning
ios::app	Append
ios::ate	Go to end of file for opening
ios::binary	Binary file
ios::in	Open file for reading only
ios::nocreate	Open fails if file does not exist
ios::noreplace	Open file if file already exist
ios::out	Open file for writing only
ios::trunc	Delete contents if file exists

Manipulation of File pointer

Standard Template Library

Introduction

- Standard Template library is a collection of generic software components(containers) and generic algorithms and objects called iterators.
- STL has large number of non-member functions designed to work on multiple classes of container types.
- There are three main components of STL.

1) Containers
2) Algorithms
3) Iterators.

These three components work in conjunction with one another to give the support to a different variety of programming solutions.

Algorithm employ iterators to perform operations stored in containers.

Containers

- Containers is an object that actually stores the data of same type.
- It is a way data is organized in memory. The STL containers are implemented by template classes and so they can be easily customized to hold different types of data.
- The STL , defines ten containers which are grouped into three categories
- 1) Sequence containers
  - vector
  - deque
  - list
- 2) Associative containers
  - set
  - multiset
  - map
  - multimap
- 3) Derived containers
  - stack
  - Queue

Advantages of containers

1) Small in number : The software components are few in number and so are easy to master, they are extremely useful in solving problems.
2) Generality : They are general in nature, it is possible to use them at various places without much trouble.
3) Efficient, Tested, Debugged and Standardized: The program written using STL components are easier to program and also easier to read.
4) Portability and Reusability : Program that uses STL becomes more portable because the vectors, queues are available.

Algorithms

- An algorithm is a procedure that is used to process the data contained in the containers. The STL includes many different kinds of algorithms to provide support to tasks such as initializing, searching, copying, sorting and merging.
- Algorithms are implemented using template functions.
- Algorithms are functions that can be used generally across a variety of containers for processing their contents.
- STL provides more than sixty standard algorithms to support more complex operations.
- STL Algorithms, based on the nature of operations they perform, may be categorized as under:
- Retrieve or non–mutating algorithms
- Mutating algorithms
- Sorting algorithms
- Set algorithms
- Relational algorithms vc

Advantages of Algorithms

- Programmers are free from writing routines like sort( ), merge( ), binary_search( ) and so on with different variations in each.
- The algorithms use the best mechanisms to be as efficient as possible , and designing which may not be possible for most programmers.
- Generic algorithms are standardized and have more acceptability.

Iterators

- An iterator is an object like a pointer that points to an element in a container. We can use iterators to move through the contents of containers.
- They are often used to traverse from one element to another.
- Iterators are handled just like pointers, as we can increment and decrement them.
- Iterators connect algorithms with containers and play a key role in the manipulation of data stored in containers.
- There are five types of iterators :
- 1) Input : Only for traverse in container
- 2) Output : Only for traverse in container
- 3)Forward : it supports all operations of i/o & also retains position
- 4) Bidirectional : it has ability to move backward in container/
- 5) Random : it combines the functionality of bidirectional and jump to any location.
- Different types of iterators must be used with different types of