OBJECT ORIENTED PROGRAMMING WITH C++

SUB CODE :18BIT23C

UNIT V: Files: File Stream Classes – File Modes – Sequential Read/ Write Operations – Binary and ASCII Files – Random Access Operation – Command Line Arguments - Exception Handlings : Principles of Exception Handling – The Keywords try, Throws and Catch – Exception Handling Mechanism – Multiple Catch Statements – Catching Multiple Exceptions – Re-throwing Exception – Strings: Declaring and Initializing String Objects – Strings Attributes – Miscellaneous Functions.

TEXT BOOK

1. Ashok N Kamthane, “Object Oriented Programming with ANSI and Turbo C++”, Pearson Education Publications, 2006.

Prepared by: Dr. M.Soranamageswari

Applications With Files:

- The file is an accumulation of data stored on the disk created by the user.
- The programmer assigns the file name, the file names are unique and are used identify the file. No two files can have the same name in the same directory.
- There are various types of files such as text file, program file, data file and executable files.
- Data files constrain a combination of number, alphabets, symbols etc.,
- Data communication can be performed between program and output devices or files and programs.
- File stream are used to perform the communication.
- The Stream is nothing but flow of data in bytes in sequence.
- The input stream brings data to the program and the output stream collect data from the program..

5.1 File Stream Classes:

- Stream is nothing but flow of data..In oops the streams are controlled using classes.
- The operations with the files are mainly of two types
- There are read and write.
- The ios class is the base class all other classes are derived from ios class.
- This classes contain several member function to perform input and output operation.
- The istream and ostream classes control input and output functions respectively.
- The function get(), getline(), read() and >> or defined in the istream class.
- The function put(), write() and << are defined in the ostream class.
- The class ifstream and ofstream are derived from istream and ostream classes respectively.
- The header file fstream.h contains the declaration of ifstream and ofstream classes.

File Buffer

- File Buffer used to accomplished input and output operations with files.
- The io functions of the class istream and ostream invoke the file-buf function to perform the insertion and extraction on the streams.

Fstream Base

- It act as a base class for fstream, ifstream and ostream classes.

Ifstream

- It is derived from fstream base class it can access the member function such as get(), getline(), seekg(), tellg() and read().

Ofstream

- This classes derived from fstream base and ostream class.
- It can access the member function such as put(),seekp(), tellp() and write().

Fstream

- It allows simultaneous input and output operations on a file buffer.
- FSTREAM invoke the member function getline() to read() the characters from the file.

Steps of File Operations

- The file operation invokes the following basic activities
  1. File name
  2. Opening file
  3. Reading (or) writing file
  4. Detecting error
  5. Closing the file

File Name:

- The file name can be a sequence of characters called as streams.
- String are always declared with character array using file name a file is recognized.
- The length of file depends on the operating system.
- Normally it is eight characters.
- A file name also contain extension of 3 characters.
- The extension is optional
- Svcc.txt
- Prg.cpp
- Test.tat

Opening file:

- There are two methods for opening a file
- Constructor of the class
- Member function open()

Constructor of the class:

- The file stream object is created using suitable class and initialized with the file name.
- The constructor itself uses the file name as the first argument and opens the file.
- The class of stream creates the output stream object and ifstream creates the input stream object.

5.2 File Opening modes:

- The opening of file involves several modes depending upon the operation to be carried out with the file.
- The open() function has two arguments
  1. File name
  2. File mode

Syntax:

- Object.open(“filename:filemode”);

Example:

- Out.open(“text.dat:ios::out);
- The file can be opened with one or more mode parameters.
- When more than one parameter necessary the bitwise operator separates them. It does not create a new file if the specified file is not present.

Example:

Out.open(“file1”,ios::app::ios::nocreate)

Write a program to open a file in a binary mode write and read the data.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
{
ofstream out;
char name[20];
out.open(“text”,ios::out::ios::binary)
cin.getline(name,20);
out<<name;
out<<close();
ifstream.in;
```

```
in.open(“text”,ios::in::ios::binary);
char ch;
while(in.eof()==0)
{
ch=in.get()
cout<<ch;
}
}
```

Finding End Of A File:

- While reading a data from a file.
- It is necessary to find the end of the file. The programmer predict the end of the file.
- The program does not detect then if will be the infinite loop.
- So avoid this it is necessary to provide to detect the end of file.
- EOF() member function is used for this propose.

- It is an instruction given to the program by the operating system of end of the file is reached.
- The EOF() function returns zero value when end of file is detected. otherwise zero.

Example:

```
#include<iostream.h>
#include<conio.h>
void main()
{
char c;
ifstream in(“test.dat”);
while(in.eof()==0)
{
in.get(c);
cout<<in.eof();
}
}
```

File Pointers And Manipulators

- To file pointers are associated file .
- This to file pointers provide two integer values.
- The read or write operations are carried out using these file pointers.
- The get pointers helps to read the file from the given location.
- The put pointers helps for writing data in file at a specified location.
- While file is open the for reading or writing the file pointers are at the beginning of the file.
- So the reading and writing can be performed from the beginning of the file.
- To explicitly set the file pointer at the specified position the file function are used.

Read Mode:

- When a file is open in read mode the get pointer is set at the beginning of the file.
- It reads the file from the first characters of the file.

Write Mode:

- When a file is open write mode the put pointers is set of the beginning of the file.
- In case the specified file already exists the content of the file will be edited.

Uppend Mode:

- When a file is open in upend mode.
- The output pointers is set at the end of the file.
- When a pre existing file is successfully opened in uppend mode the contains remains safe and new data is upended of the end of the file.

File Pointers Managing Function:

Seekg()

- It is the member of ifstream() class.
- It shift the input pointer to a given location.

Seekp()

- It is the member of ofstream class.
- It shift the output pointer the given location.

Tellg()

- It is the member of ifstream class.
- It provides the current position of the input pointer.

Tellp()

- It is the member of ofstream class.
- It provides the current position of the output pointer.

Manipulators With Arguments

- The seekp() and seekg() function can be used with two arguments.

Example:

- Seekg(offset, preposition)
- Seekp(offset,preposition)
- The first arguments offset specifieds the number of bytes the file pointer is to be shifted from the argument preposition of the pointer. The preposition arguments have the following values.
- The seekg() function shift the associated files input file pointers.
- The seekp() function shift the associated files output pointers.

5.3 Sequential Read And Write

- C++ allows file manipulations command to accessed a file sequentially or randomally. The data of sequential files must be access one characters at a time.
- To access thus specified by all the previous characters are read and ignored.
- There are member of function to perform read and write operations.
- The put() and get() functions are used to read or write a single character. The read() and write() block of binary data.

The Put() And Get() Functions:

- The functions get() reads a single characters from the file.
- Pointered by the get() pointer.
- The put() function writes a character to the specified file.

Example:

```
#include<fstream.h>
#include<conio.h>
#include<string.h>
int main()
{
clrscr();
char text[50];
cout<<"\n enter a text:";
cin.getline(text,50);
int i=0;
fstream io;
```

```
io.open(“data;ios::in/ios::out);
while(1[text]!=’\q’)
io.put(text[1++]);
io.seekg(0);
char c;
cout<<”\n entered text:”;
while(io)
{
io.get(c );
cout<<c;
}
return 0;
}
```

The Read() And Write() Functions:

- The read() and write() function is used binary format of dat during operation.
- In binary format the data representation is same in the file and the system.
- The binary form follows quicks read and write because know conversion is needed during the operations.
- The format a read() and write() function is follows:

Syntax:

- In.read((char *)&p,size of(p));
- Out.write((char *)&p,size of (p));

Example:

#include<fstream.h>

#include<conio.h>

#include<string.h>

```
int main()
{
clrscr();
intnum[]={100,105,110,120,155,250,255};
ofstream out;
out.open(“01.bin”);
out.write((char*)&num,size of (num);
out.close();
for(int i=0;i<7;i++)num[i]=0;
ifstream in;
in.open(“01.bin”);
in.read((char *)&num,size of (num);
for(i=0;i<7;i++)cout<<num[i]<<”\t”;
return 0;
}
```

5.4 Binary And Ascii Files:

- ASCII codes are used by the io devices to share the data to the computer system.
- The CPU can process only the binary numbers(0,1)
- So it is the essential to convert the data while accepting the input device and display the data to the output device.
- Example:
- cin>>a;
- cout<<a;
- The Operator<<converts the integer value into a stream of ASCII characters. The operator>>convert the ASCII value of the binary format and assign it to the variable.

5.5 Random Access Operation

- Random access operation data file always contain large information always changes.
- The changed information should be updated.
- To update a particular record or data file it may be stored any ware in the file but it is necessary identify in the location where the data object is stored.
- The size of() operator determines the size of the object.

Error Handling Functions

- There are many reasons to cause error during reading or writing operation of the program.
- An attempt to read a file which does not exist
- The file name specified for opening a new file may already exist.
- An attempt to read the file when the file pointer is at the end of the file.
- Insufficient disk space

- Invalid file name specified by the programmer.
- An attempt to write data to the file that is opened in read only mode.
- A file opened may already opened by another programs.
- An attempt to opened read only file for writing operation.
- Device error.

5.6 Command Line Arguments

- An executable program that performs a specific task for operating system is called a command .
- The commands are issued from the command prompt of os.
- Some arguments are associated with command called as command line arguments.
- These arguments are passed to the program.
- In c++ every program start with a main() function.
- The main()function does not contain any arguments
- In command line arguments the main() function can receive two arguments.
- 1.argc-argument counter it contains the number of arguments
- 2.argv-argument vector it is an array of character pointers.

Syntax:

Main(int argc,char *argv[])

Example:

```
#include<stdio.h>
#include<fstream.h>
#include<conio.h>
#include<process.h>
main(int argc, char *argv[])
{
fstream out;
ifstream in;
if(argc<3)
{
cout<<:insufficient arguments”;
exist(1);
}
```

```
in.open(argv[1],ios::in,ios::nocreate);
if(in.fail())
{
cout<<”\n file not found;
exist(0);
}
in.close();
out.open(argv[2],ios::in,ios::nocreate);
if(out.fail())
{
rename(argv[1],argv[2]);}
else
cout<<”\n duplicate file name or file is in use”;
return 0;
}
```

Generic Programming With Template:

- The template provides generic programming by defining generic class.
- In template generic data types are used as arguments and they can handle a variety of data type.
- A functions that works for all c++ data types is called as generic function.

Need For Template:

- Template is a technique that allows using a single function or class to work with different data types.
- They can accept data of any type such as int, float,char,double etc.,
- Templates allows better flexibility to the program and overcome the limitation of over loading function.

Definition Of Class Templates:

- The class templates can be defined as follows:
- Syntax:
- Template class<T>
- class classname
- {
- ….
- }
- Template class<T> specifies the compiler the following class declaration use the template data type.
- T is a variable of template type that can be used in the class to define variable of template type.
- One or more variables can be declared by unseparated by comma.
- Templates can not be declared inside classes are functions. They must be global.

Example:

```
#include<iostream.h>
#include<conio.h>
template class<T>
class data
{
public:
data( T C)
{
cout<<”\n”<<”c=”<<c<<”size in bytes:<<size of(c) ;
}};
int main()
{
clrscr();
data<char>h(‘A’);
data<int> i(100);
data<float> j(3.12);
return 0; }
```

Normal Function Template

- A normal function can also used template arguments.
- The difference between normal and member function is that normal function are defined outside the class.
- These are not members of any class.
- They can be accessed directly without using (.) dot operator
- Syntax:
- Template class<T>
- Function name()
- {
- …
- }

Example:

```
#include<iostream.h>
#include<conio.h>
Template class<T>
void show(T x)
{
cout<<x;
}
void main()
{
char c=’A’;
int i=100;
float f=10.5;
show( c);
show(i);
show(f);
}
```

Overloading Of Template Function:

- A Template function also supports overloading mechanism.
- It can be overloaded by normal function or template function
- No implicit conversion is carried out in parameters of template function.
- The rules are,
- Search for accurate match of function, if found it is invoked
- Search for a template function with accurate match can be generated if found it is invoked
- Attempt normal overloading declaration for the function
- Incase of no match found error will be reported

```
# include< iostream.h >
#include < conio.h >
template < class T >
void show (T a)
{
cout << a;
}
void show (int x)
{
cout <<x;
}
void main( )
}
show(‘c’);
show(10);
show(10.5);
}
```

Class Template With Overloaded Operator:-

- The template class permits to declare overloaded operators and member function.
- Creating overloaded operator function is similar to class template members and functions.

```
#include<iostream.h>
#include<conio.h>
#include<class T>
class num
{
private:
tnumber;
public:
num()
{
   number=0; }
```

```
void input()
{
cin>.number;
}
num operator +(num);
void show()
{
cout<<number;
}
};
Template<class T>
num<T> num<T>::Operator+(num<T>)
{
num<T> temp;
Tmp.number=number+c.number;
return(temp);
}
```

- void main()
- {
- num<int>n1,n2,n3;
- N1.input;
- N2.input;
- N3.show();
- }

Class Templates And Inheritance:

- The template class can also act as a base class.
- When inheritance is applied with template class it helps to compose hierarchical data structure known as container class.
- Derive a template class and add new members to it.
- The base class must be of template type.
- Derive a class from non-template class and add new template type member to derived class.
- It is also possible to derive classes from template base class and omit the template features of the derived class.

Syntax:

```
Template< class T>
class XYZ
{
……..
}
Template<class T2,…..)
class ABC::Public XYZ<T2,….>
{
…..
}
```

Guidelines For Templates:

- Templates are applicable when we want to create type secure class that can handle different data type with same member functions.
- The template classes can also be involved in inheritance.

For Example:

```
Template<class T>
class data : public base<T>
```

- Both data and base are template classes. The class data is derived from template class base.
- The template variables also allow us to assign default values.

For Example:

```
Template<class T, int x=20>
class data
{
    Tnum[x];
}
```

- The name of template class is written differently in different situations, while class declaration it is declared as follows:

  class data{….}

  void data<T>:: show(T d) { }
- show() is a member function
- In the example, the template argument T is not used as a parameter and the complier will report an error.

  Template<class T>

  void show(int y)

  { T temp;}
- In the above example, template type arguments is not an argument.possibily the system will crash.

  Template<class T,class s>

  void max(T & K)

  {SP;}
- The template variables S in not used. Therefore, compile time error is generated.

5.7 Exception Handling:

- While writing large programs a programmer makes many mistakes developing an error free program is the objective of the programmer.
- Programmer have to take care to prevent errors.
- Errors can be trapped using exception handling features.
- An Exception is an up normal terminator of a program, which is executed in a program at run-time or it may called at run-time when an error occurs the exception contains warning messages like invalid argument insufficient memory, division by 0…..

5.8 Principles Of Exception Handling:

- Exceptions are two types

  1.synchronous exception

  2.Asynchronous exception
- The goal of exception handling is to create a routine that detect and an exceptional condition is order to execute suitable action.
- The routine carries the following responsibilities:

  1. Detect the problem

  2. Warning message indicates an error.

  3. Accept the error message.

  4. Perform the accurate action.
- An Exception is an object , it is sent from the part of the program where an error occurs to that part which is going to control the error.

5.9 The Keyword try, throw and Catch

The Keyword try

- Exception handling technique possess the control of a program from a location of exception in a program to exception handling routine linked with the key block.
- An Exception handling routine can be called by throw statement.
- Try is a keyword contains series of statement.

The keyword throw

- The function throw statement is to sent the exception found.
- The throw statement can be placed in function or in nested loop and it should be in try block
- After throwing exception control possess to the catch statement.

The Keyword Catch

- Catch block also contains a series of statements.
- It also contains an argument of exception type..

5.10 Exception Handling Mechanism:

- Exception handling Mechanism provides three keyword try, throw and catch.
- The keyword try is used at the starting of the exception.
- The throw block is present inside the try block.
- Immediately after the try block catch block is present.
- When an exception is found the throw statement throws an exception(message) for catch block that an error has occurred in the try block.
- The catch block receives the exception and performs the actions.

Example:

- Write a program to throw exception when j=1 otherwise perform the sub of x and y.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int x,y;
cout<<”enter the value of  x and y”;
cin>>x>>y;
J=x>y?0:1;
try
{
if(j=0)
{
cout<<x<<y;
}
else
{
throw(j); }}
```

```
catch(int k)
{
cout<<”Exception caught”;
}
}
```

5.11 Multiple Catch Statements:

- We can also define multiple catch block in try block.
- It contains multiple throw statements based on certain conditions.

Syntax:

```
try
{
Sts;
}
catch(object 1)
{
catch section 1;
}
catch(object 2)
{
catch section 2;
}
catch(object n)
{
catch section n; }
```

5.13 Catching Multiple Exceptions :

- It is also possible to define single caught block for one or more exception of different types.

Syntax:

```
catch
{
multiple throws sts;
}
```

Example:

```
#include<iostream.h>
#include<conio.h>
void num(int k)
{
Try
{
```

```
if(k==0)
throw k;
else
if(k>0)
throw p;
else
if(k<0)
throw 0;
}
catch(…)
{
cout<<”Caught Exception”;
}
}
```

5.14 Re-throwing Exception

- It is also possible to pass the exception received to another exception handler. That's an exception is thrown from catch block and this is known as re -throwing exception.
- Syntax:
- throw;
- The throw statement is used without any arguments.

Example:

```
#include<iostream.h>
#include<conio.h>
void sub(int j, int k)
{
cout<<"inside function sub()\n;
try
{
```

```
if(j=0)
throw j;
else
cout<<”subtraction =”<<j-k<<”\n”;
}
catch(int)
{
cout<<”caught null value\n”;
throw;
}
cout<<”end of sub()***\n\n”;
}
int main()
{
cout<<”\n inside function main()\n”;
```

```
try
{
sub(8,5);
sub(0,5);
}
catch(int)
{
cout<<”caught null inside main ()\n”;
}
cout<<”end of function main()\n”;
getch();
}
```

5.15 Declaring and Initializing String Objects:

1. Strlen()
2. strcpy()

    Syntax:

    Strcpy(destination ,source)

    Example:

    Strcpy(b,a)

3.strncpy

    Syntax:

    Char*strncpy(char*desk, const char *src, size –t n)

    Example:

    Strncpy(dest,src,10);

4.Strcmp();

Example:

Srtcmp(str1,str2);

5)stricmp();

Example

i=stricmp(Str1,Str2)

6)Strncmp();

Example

Set=strncmp(Str1,Str2,4);

7)Strnicmp();

Example

j=strnicmp(Str1,Str2,5);

8)Strlwr();

Example

Strlwr(Str1);

9)Strupr();

Example

Strupr(Str1);

10)Strdup():-

Example

P2=Strdup(p1);

11) Strchar();

Example

Set=Strchr(Str,ch);

12) Strrchar();

Example

Set=strrchar(Str,ch);

13)StrStr()

Example:

Set=strstr(traystack,needle);

14) strncat()

Example:

Strncat(example, “is over 10 year old”,21)

15) strset()

Example:

Strset(str,’#’);