CHAPTER 6

Exception Handling and Stream I/O

Er. Ganga Gautam

Outline

- Exception Handling:
  - Error and Exception,
  - Exception Handling Mechanism (try, throw, and catch),
  - Multiple Exception Handling
- File Handling:
  - Stream Class Hierarchy,
  - Opening and Closing a File,
  - Reading and Writing Object

Errors and Exception

Errors can be broadly categorized into two types.

- Compile Time Errors
- Run Time Errors

- **Compile Time Errors** – Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.
- **Run Time Errors** - They are also known as exceptions. An exception caught during run time creates serious issues.

Exception

- An exception is a problem that arises during the execution of a program.
- A C++ exception response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another.
- C++ exception handling is built upon three keywords: try, catch, and throw.

Keyword used for Exception handling

Keywords	Purpose
try	• The code which can throw any exception is kept inside (or enclosed in) a *try* block. • Then, when the code will lead to any error, that error/exception will get caught inside the catch block.
catch	• catch block is intended to catch the error and handle the exception condition. • We can have multiple catch blocks to handle different types of exception and perform different actions when the exceptions occur. • For example, we can display descriptive messages to explain why any particular exception occurred.
throw	• It is used to throw exceptions to exception handler i.e. it is used to communicate information about error. • A throw expression accepts one parameter and that parameter is passed to handler.

Exception Handling

Syntax:

```
try
{
    //code
    throw parameter;
}
catch(ExceptionType Exception_object)
{
    //code to handle exception
}
```

Example:

```
try
{
    if(a/b==0)
      throw "Divide by zero illegal";
}
catch(const char* ex)
{
    cout<<ex;
}
```

Sample Program 6.13

WAP to illustrate exception handling when divide by a zero.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b,c;
6      cout<<"Enter two numbers"<<endl;
7      cin>>a>>b;
8      // try block activates exception handling
9      try
10     {
11         if(b == 0)
12         {
13             // throw custom exception
14             throw "Division by zero not possible";
15         }
16         else
17         {
18          c = a/b;
19          cout<<"Result="<<c;
20
21       }
22     }
```

```
23     catch(const char* ex) // catches exception
24     {
25         cout<<ex;
26     }
27     return 0;
28 }
```

When we enter 10 and 5, result is displayed.

```
Enter two numbers
10
5
Result=2
--------------------
```

When we enter 10 and 0, exception is caught and message is thrown.

```
Enter two numbers
10
0
Division by zero not possible
--------------------------------
```

Sample Program 6.15

WAP to take age of voter from user and check if he/she is eligible to vote or not. The voter should be 18 years or above to vote. Your program should handle the exception case for not being able to vote.

```
#include <iostream>
using namespace std;
main()
{
   int age;
   cout<<"Enter your age"<<endl;
   cin>>age;
   try
      {
      if (age >= 18)
            cout << "Access granted - You can vote.";
      else
            throw 505;
      }
   catch (int myNum)
      {
      cout << "Access denied - You must be at least 18 years old.\n";
      cout << "Error number: " << myNum;
      }
}
```

Output: When we 18 or more.

```
Enter your age
25
Access granted - You can vote.
--------------------------------
```

Output: When we put age less than 18

```
Enter your age
12
Access denied - You must be at least 18 years old.
Error number: 505
--------------------------------
```

Sample Program 6.14

WAP to illustrate exception handling for array index out of size of the array.

```
1  #include <iostream>
2  using namespace std;
3  main()
4  {
5      int id, num[5]={10,20,30,40,50};
6      cout<<"Enter the array index to display the data"<<endl;
7      cin>>id;
8      try
9      {
10       if (id>4)
11          throw "Array index out of range";
12       else
13          cout<<num[id]<<endl;
14     }
15
16     catch(const char* ex) // catches exception
17     {
18         cout<<ex;
19     }
20 }
```

Output: When we put 4 or less.

```
Enter the array index to display the data
4
50
```

Output: When we put 5 or above

```
Enter the array index to display the data
5
Array index out of range
--------------------------------
```

Multiple Catch blocks

- In some cases, more than one exception could be raised by same piece of code in a program.
- To handle different types of situations, we can specify two or more catch blocks, each catching a different type of exception.
- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches with that of exception, is executed.
- After one catch statement executes, the others are bypassed and execution continues after try/catch block.

General Syntax:

```
catch (type1 argument)
    {
        //code
    }
catch(type2 argument)
    {
        //code
    }
    ........
    ........
```

Sample Program 6.16

WAP to illustrate multiple exception handling (divide by zero and arraysize out of range).

```
1  #include <iostream>
2  #include<conio.h>
3  using namespace std;
4  main()
5  {
6      int id, num[5]={10,20,30,40,50};
7      int a,b,c;
8      cout<<"Enter two numbers"<<endl;
9      cin>>a>>b;
10     cout<<"Enter the array index to display the data"<<endl;
11     cin>>id;
12     try
13     {
14       if (b==0)
15          throw "Divide by zero not possible";
16       else if (id>4)
17          throw  400;
18       else
19       {
20          c=a/b;
21          cout<<"Result="<<c<<endl;
22          cout<<num[id]<<endl;
23       }
24     }
```

```
26     catch(cons char   e1) // catches exception1
27     {
28         cout<<e1;
29     }
30
31     catch(const int   e2) // catches exception2
32     {
33         cout<<e2;
34     }
35 }
```

Output: When we input two numbers: 10 and 0 and array index:3

```
Enter two numbers
10
0
Enter the array index to display the data
3
Divide by zero not possible
--------------------------------
```

Output: When we input two numbers: 10 and 5 and array index:5

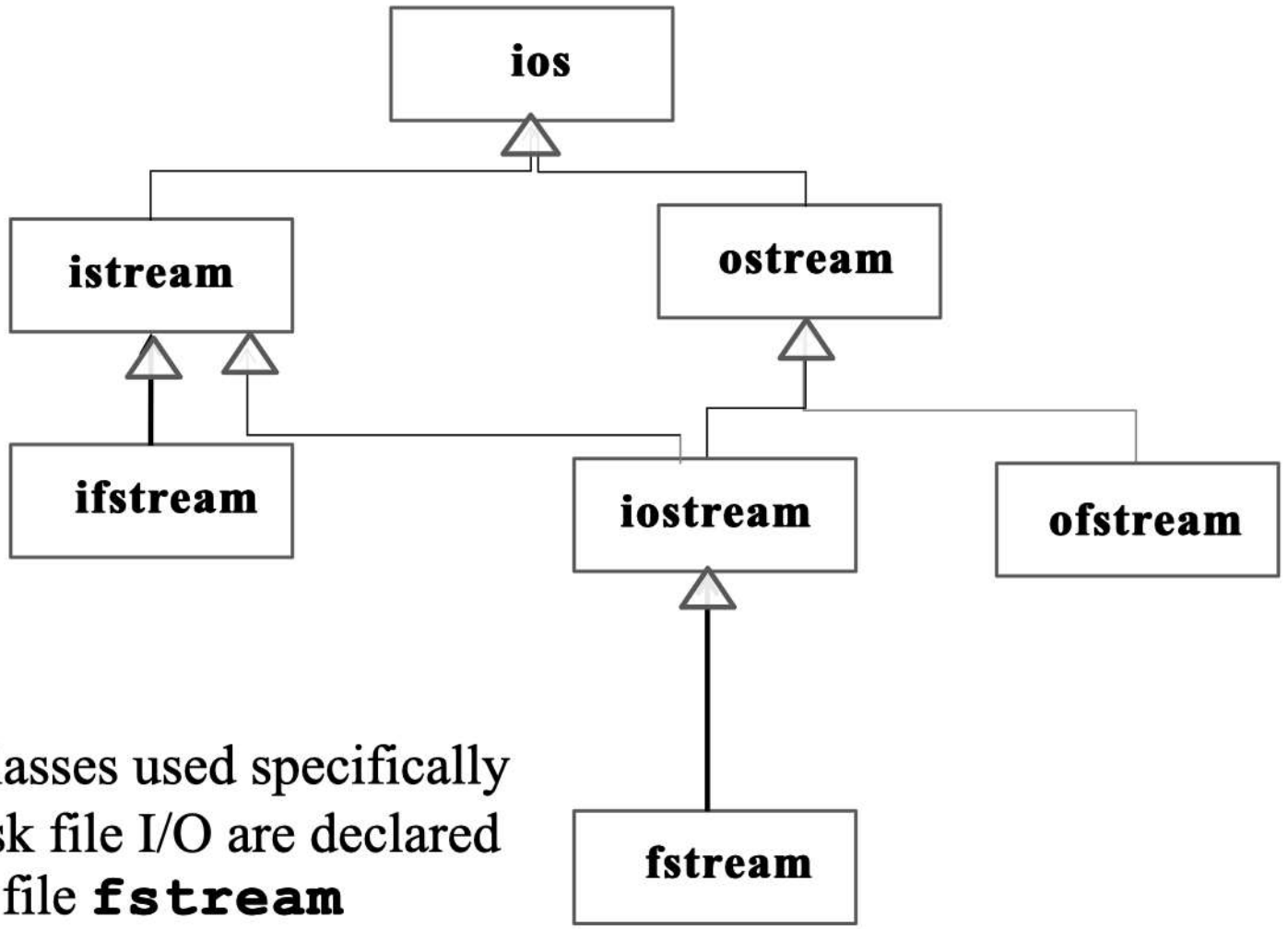
```
Enter two numbers
10
5
Enter the array index to display the data
5
 Array index out of range
--------------------------------
```

File Handling

- File handling is used to store data permanently in a computer.
- Using file handling we can store our data in secondary memory (Hard disk).

Stream class Hierarchy

- Streams in C++ are used for handling input and output operations.
- Stream classes are organized in a hierarchy based on their functionality.
- The two main categories are: **Input Stream** and **Output Stream**.



The classes used specifically for disk file I/O are declared in the file **`fstream`**

Input Stream

- Derived from ios class (base class for I/O streams).
- Used for reading data from a source (e.g., files, keyboard input).
- Examples: istream, ifstream, istringstream.

Output Stream

- Also derived from ios class.
- Used for writing data to a destination (e.g., files, console).
- Examples: ostream, ofstream, ostringstream.

- **ofstream**: This Stream class signifies the output file stream and is applied to create files for writing information to files
- **ifstream**: This Stream class signifies the input file stream and is applied for reading information from files
- **fstream**: This Stream class can be used for both read and write from/to files.

File Handling in C++

- To work with files in C++, include the <fstream> header.
- File handling involves the following steps:
  - **Opening** a file.
  - **Reading/Writing** data to/from the file.
  - **Closing** the file.

Opening a File

We can open a file using any one of the following methods:

1. First is bypassing the file name in constructor at the time of object creation.
2. Second is using the open() function.

Syntax:

void open(const char* file_name,ios::openmode mode);

File opening modes:

Modes	Description
in	Opens the file to read(default for ifstream)
out	Opens the file to write(default for ofstream)
binary	Opens the file in binary mode
app	Opens the file and appends all the outputs at the end
ate	Opens the file and moves the control to the end of the file
trunc	Removes the data in the existing file
nocreate	Opens the file only if it already exists
noreplace	Opens the file only if it does not already exist

Default Open Modes :

- ifstream ios::in
- ofstream ios::out
- fstream ios::in | ios::out

Example:

```
fstream new_file;
new_file.open(“newfile.txt”, ios::out);
```

- Using a stream insertion operator << we can write information to a file
- Using stream extraction operator >> we can easily read information from a file.

Example of opening/creating a file using the open() function

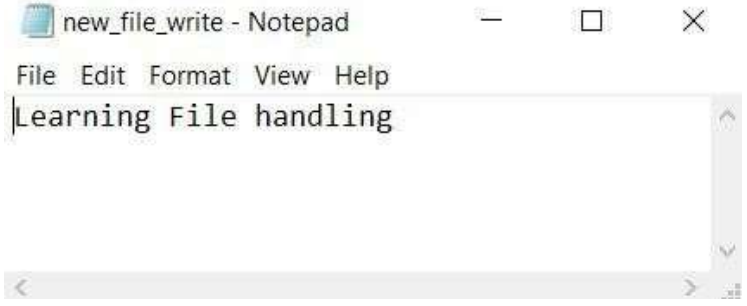
```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
	fstream new_file;
	new_file.open("new_file",ios::out);
	if(!new_file)
	{
		cout<<"File creation failed";
	}
	else
	{
		cout<<"New file created";
		new_file.close();
	}
	return 0;
}
```

Writing to a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	fstream new_file;
	new_file.open("new_file_write.txt",ios::out);
	if(!new_file)
	{
		cout<<"File creation failed";
	}
	else
	{
		cout<<"New file created";
		new_file<<"Learning File handling";    //Writing to file
		new_file.close();
	}
	return 0;
}
```

```
C:\Users\Lenovo\Desktop>g++ new_file.cpp -o new_file.exe

C:\Users\Lenovo\Desktop>new_file.exe
New file created
C:\Users\Lenovo\Desktop>
```



Reading from a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	fstream new_file;
	new_file.open("new_file_write.txt",ios::in);
	if(!new_file)
		cout<<"No such file"; } else { char ch; while 		(!new_file.eof()) { new_file
	>>ch;
		cout << ch;
	}
	new_file.close();
	return 0;
}
```

```
LearningFilehandlingg[Finished in 1.7s]
```

Close a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream new_file;
    new_file.open("new_file.txt",ios::out);
    new_file.close();
    return 0;
}
```

Practise:

- Create a file “Welcome.txt” and “write welcome to file handling in C plus plus” and then read the file content.

Reading and Writing Object

- read data from files into class objects
- write data in class objects to files.

Writing class object to the file

```
class_name obj;
ofstream file1;
file1.open( “file_name”, ios::app );
file1.write( (char*)&obj, sizeof(obj) );
```

Here,

- data present in class object obj is written to file specified file by calling write function.
- (char*)&obj is used to point at the start of an object.
- sizeof(obj) calculates the number of bytes copied in file.

Reading class object to the file

```
class_name obj2;
ifstream file2;
file2.open( “file_name”, ios::in );
file2.read( (char*)&obj2, sizeof(obj2) );
```

Here,

- data present in class object obj2 is read from file specified by calling read function.
- (char*)&obj2 is used to point at the start of an object and
- sizeof(obj2) calculates the number of bytes to read from the file.

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
class Student {
        public:
                char Name[20];
                int Student_ID;
                int age;
        };
        int main(){
                Student std;
                Strcpy(std.Name, "John”);
                std.Student_ID=2121;
                std.age=11000;

                //Writing this data to Student.txt
                ofstream file1;
                file1.open(”Student.txt”, ios::app);
                file1.write((char*)&std,sizeof(std));
                file1.close();

                //Reading data from Student.txt
                Ifstream file2;
                file2.open(”Student.txt",ios::in);
                file2.seekg(0);
                file2.read((char*)&std,sizeof(std));
                cout<<“\nName :”<<std.Name;
                cout<<"\nStudent ID :”<<std.Student_ID;
                cout<<"\nAge:”<<std.age;
                file2.close();
                return 0;
}
```

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
class Person {
        private:
                char Name[20];
                float weight;
                int age;
        Public:
                void get_info()
                {
                        cout<<“Enter name, weight and age: ”;
                        cin>>Name>>weight>>age;
                }
        };
        int main(){
              int i;
                Person p[2];

                //Writing this data to Information.txt
                ofstream file1;
                file1.open(” Information.txt”, ios::app);
                for(i=0;i<2;i++)
                {
                   p[i].get_info();
                   file1.write((char*)&p[i],sizeof(p[i]));
                }
                file1.close();

                //Reading data from Information.txt
                Ifstream file2;
                file2.open(”Information.txt",ios::in);
                file2.seekg(0);
                cout<<“Name\t”<<“ weight\t” <<“ age”;
                for(i=0;i<2;i++)
                {
                        file2.read((char*)&p[i],sizeof(p[i]));
                        cout<<“……”;
                        cout<<p[i].Name<<“\t”<<p[i].weight<<“\t”<<p[i].age;
                }
                file2.close();
                return 0;
}
```

- Assume that you have a class called ”Employee" with the following attributes: name , age , and occupation . Implement the necessary functions to allow the user to input data for multiple Employee objects and store them in a file ’Details.txt’. Additionally, provide functionality to read and display the stored Employee objects from the file.

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
class Employe {
        private:
                char Name[20];
                char occupation[50];
                int age;
        Public:
                void get_info()
                {
                        cout<<“Enter name, occupation and age: ”;
                        cin>>Name>> occupation >>age;
                }
                void set_info()
                {
                        Cout<<Name<<“\t”<< occupation<<“\t” <<age<<endl;
                }
        };
        int main(){
              int i;
                Employe e[2];

                ofstream file1;
                file1.open(” Details.txt”, ios::out);
                for(i=0;i<2;i++)
                {
                   p[i].get_info();
                   file1.write((char*)&e[i],sizeof(e[i]));
                }
                file1.close();

                Ifstream file2;
                file2.open(”Details.txt",ios::in);
                file2.seekg(0);
                cout<<“Name\t”<<“ Occupation\t” <<“ age”<<endl;
                for(i=0;i<2;i++)
                {
                        file2.read((char*)&e[i],sizeof(e[i]));
                        cout<<“……”;
                        p[i].set_info();
                }
                file2.close();

                return 0;
}
```

- **Create a class called book having a member name, price, author and pages. Create a file called "Library.docx" and store record of 500 books. Now, read the file to print the information of the book which is above 300.**

- A class called ”student" has the following attributes: roll number, name , age , and address . Implement the necessary functions to allow the user to display for 100 student objects from file ’StudentInfo.txt’.

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
class Student {
        private:
                char Name[20];
                char add[50];
                int age;
                Int roll;
        public:
                void set_info()
                {
                        Cout<<Name<<“\t”<<add<<“\t”<< age<<“\t” <<roll<<endl;
                }
        };
        int main(){
              int i;
                Student e[100];

                Ifstream file2;
                file2.open(”studentInfo.txt",ios::in);

                if(!new_file)
                {
                        cout<<"No such file";
                }
                else
                {
                        file2.seekg(0);
                        cout<<“Name\t”<<“ address\t” <<“ age”<<“\t”<<roll<<endl;
                        for(i=0;i<100;i++)
                        {
                                file2.read((char*)&e[i],sizeof(e[i]));
                                cout<<“……”;
                                p[i].set_info();
                        }
                }
                file2.close();

                return 0;
}
```

End of chapter 6