Managing Console I/O Operations – Working with Files – Templates – Exception Handling

**MANAGING CONSOLE I/O OPERATIONS**

* A stream is an abstraction. It is a sequence of bytes.
* It represents a device on which input and output operations are performed.
* It can be represented as a source or destination of characters of indefinite length.
* It is generally associated to a physical source or destination of characters like a disk file, keyboard or console.
* C++ provides standard iostream library to operate with streams.
* The iostream is an object-oriented library which provides Input/Output functionality using streams.



In the above figure, ios is the base class. The iostream class is derived from istream and ostreamclasses. The ifstream and ofstream are derived from istream and ostream, respectively. These classes handles input and output with the disk files.

The fstream.h header file contains a declaration of ifstream, ofstream and fstream classes. The iostream.h file contains istream, ostream and iostream classes and included in the program while doing disk I/O operations.

The filebuf class contains input and output operations with files. The streambuf class does not organize streams for input and output operations, only derived classes of streambuf performs I/O operations. These derived classes arranges a space for keeping input data and for sending output data.

The istream and ostream invokes the filebuf functions to perform the insertion or extraction on the streams.

**Advantages of Stream Classes**

* Stream classes have good error handling capabilities.
* These classes work as an abstraction for the user that means the internal operation is encapsulated from the user.
* These classes are buffered and do not uses the memory disk space.
* These classes have various functions that make reading or writing a sequence of bytes easy for the programmer.

**WORKING WITH FILES**

File represents storage medium for storing data or information. Streams refer to sequence of bytes. In Files we store data i.e. text or binary data permanently and use these data to read or write in the form of input output operations by transferring bytes of data. So we use the term File Streams/File handling.

1. **ofstream:** It represents output Stream and this is used for writing in files.
2. **ifstream:** It represents input Stream and this is used for reading from files.
3. **fstream:** It represents both output Stream and input Stream. So it can read from files and write to files.

**Operations in File Handling:**

* Creating a file: open()
* Reading data: read()
* Writing new data: write()
* Closing a file: close()

Many real-life scenarios are there that handle a large number of data, and in such situations, you need to use some secondary storage to store the data. The data are stored in the secondary device using the concept of files. Files are the collection of related data stored in a particular area on the disk. Programs can be written to perform read and write operations on these files.

Working with files generally requires the following kinds of data communication methodologies:

* Data transfer between console units
* Data transfer between the program and the disk file

So far we have learned about *iostream* standard library which provides cin and cout methods for reading from standard input and writing to standard output respectively. In this chapter, you will get to know how files are handled using C++ program and what are the functions and syntax used to handle files in C++.

**TEMPLATES IN C++**

A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

C++ adds two new keywords to support templates: ‘template’ and ‘typename’. The second keyword can always be replaced by keyword ‘class’.

**How templates work?**

Templates are expanded at compiler time. This is like macros. The difference is, compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.

**Function Templates** We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray().

**C++ EXCEPTION HANDLING**

An exception is a problem that arises during the execution of a program. A C++ exception is a 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.

**throw −** A program throws an exception when a problem shows up. This is done using a throw keyword.

**catch −** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

**try −** A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception.

**Syntax**

try {

 // protected code

} catch( ExceptionName e1 ) {

 // catch block

} catch( ExceptionName e2 ) {

 // catch block

} catch( ExceptionName eN ) {

 // catch block

}

**Throwing Exceptions**

Exceptions can be thrown anywhere within a code block using throw statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Following is an example of throwing an exception when dividing by zero condition occurs

double division(int a, int b) {

 if( b == 0 ) {

 throw "Division by zero condition!";

 }

 return (a/b);

}

**Catching Exceptions**

The catch block following the try block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

try {

 // protected code

} catch( ExceptionName e ) {

 // code to handle ExceptionName exception

}