DATA FILE HANDLING IN C++

Kripashanker Yadav
Department Of Computer Science and Engineering,
Dronacharya College of Engineering, Gurgaon

ABSTRACT:- **In today's generation, millions are spent on developing softwares, and a number of them are made on c/c++ language. In c/c++, data file handling has a very vital role. This is because files help in storing data permanently as well modifying or deleting data from the files. This paper contains introduction to data files, opening and closing files using constructors and open() function, sequential I/O with files, detecting EOF, file pointers and random access and basic binary operations on files (including searching, appending data, inserting data, deleting data and modifying data) with error handling during file I/O.**

Index Terms:- softwares; data file handling; constructors; pointers; error handling.

I. INTRODUCTION

Files are used widely in computer programs; this is because they help in storing information permanently. Word processors create document files, database programs create files of information and compilers read source files and generate executable files. Here you see, it is the files that are mostly worked out with. A file itself is a bunch of bytes stored on CD's, HDD's etc. In C++, the input/output operations on file are executed with the usage of a header file name fstream.h. At the lowest level, a file in C++ is interpreted simply as a sequence of bytes. The notion of the data is absent at this level. While, at the user level, a file may possibly consists of intermixed data types- arithmetic values, characters, class objects. The fstream.h header file includes the classes and functions to perform all the necessary input and output operations that are performed on files in C++. ifstream is used for input operations and ofstream is used for output operations. We can modify all the contents of the data stored in the file according to the user's need with the help of binary operations.

File: The information/data stored under a specific name on a storage device, is called a file.

Stream: It refers to a sequence of bytes.

Text file: It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.

Binary file: It is a file that contains information in the same format as it is held in memory. In binary files, no delimiters are used for a line and no translations occur here.

II. CLASSES FOR FILE STREAM OPERATIONS

ofstream: Stream class to write on files
ifstream: Stream class to read from files
fstream: Stream class to both read and write from/to files.

III. OPENING A FILE

OPENING FILE USING CONSTRUCTOR

```
ofstream outFile("sample.txt");    //output only
ifstream inFile("sample.txt");  //input only
```

OPENING FILE USING open()

```
Stream-object.open("filename", mode)
ofstream outFile;
outFile.open("sample.txt");
ifstream inFile;
inFile.open("sample.txt");
```

File mode parameter

	Meaning
ios::app	Append to end of file
ios::ate	go to end of file on opening
ios::binary	file open in binary mode
ios::in	open file for reading only
ios::out	open file for writing only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if the file already exist

ios::trunc
delete the contents of the file if it exist

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

```
fstream file;
file.open ("example.bin", ios::out | ios::app | ios::binary);
```

4. Closing File

```
outFile.close();
inFile.close();
```

V. INPUT AND OUTPUT OPERATION

put() and get() function

the function put() writes a single character to the associated stream. Similarly, the function get() reads a single character form the associated stream.

example :

```
file.get(ch);
file.put(ch);
```

write() and read() function

write() and read() functions write and read blocks of binary data.

example:

```
file.read((char *)&obj, sizeof(obj));
file.write((char *)&obj, sizeof(obj));
```

VI. ERROR HANDLING FUNCTION

FUNCTION	RETURN VALUE AND MEANING
eof()	returns true (non-zero) if end of file is encountered while reading; otherwise return false(zero)
fail()	return true when an input or output operation has failed
bad()	returns true if an invalid operation is attempted or any unrecoverable error has occurred.
good()	returns true if no error has occurred.

VII. FILE POINTERS AND THEIR MANIPULATIONS

All i/o streams objects have, at least, one internal stream pointer:

ifstream: like istream, has a pointer known as the get pointer that points to the element to be read in the next input operation.

ofstream: like ostream, has a pointer known as the put pointer that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

seekg()	moves get pointer(input) to a specified location
seekp()	moves put pointer (output) to a specified location
tellg()	gives the current position of the get pointer
tellp()	gives the current position of the put pointer

The other prototype for these functions is:

```
seekg(offset, refposition );
seekp(offset, refposition );
```

The parameter offset represents the number of bytes the file pointer is to be moved from the location specified by the parameter refposition. The reposition takes one of the following three constants defined in the ios class.

ios::beg	start of the file
ios::cur	current position of the pointer
ios::end	end of the file

example:

```
file.seekg(-10, ios::cur);
```

VII. BASIC OPERATIONS ON TEXT FILE IN C++

File I/O is a five-step process:

1. Include the header file fstream in the program.
2. Declare file stream object.
3. Open the file with the file stream object.
4. Use the file stream object with >>, <<, or other input/output functions.
5. Close the files.

IX. BASIC OPERATIONS ON BINARY FILES IN C++

When data is stored in a file in the binary format, reading and writing
data is faster because no time is lost in converting the data from one format to another format. Such files are called binary files. This following program explains how to create binary files and also how to read, write, search, delete and modify data from binary files.

```
#include<iostream>
#include<fstream>
#include<cstdio>
using namespace std;
class Student
{
int admno;
char name[50];
public:
void setData()
{
cout << "\nEnter admission no. ";
cin >> admno;
cout << "Enter name of student ";
cin.getline(name,50);
}
void showData()
{
cout << "\nAdmission no. : " << admno;
cout << "\nStudent Name : " << name;
}
int retAdmno()
{
return admno;
}
};
void write_record()/*function to write in binary file*/
{
ofstream outFile;
outFile.open("student.dat", ios::binary | ios::app);
Student obj;
obj.setData();
outFile.write((char*)&obj, sizeof(obj));
outFile.close();
}
void display() /*function to display records of file*/
{
ifstream inFile;
inFile.open("student.dat", ios::binary);
Student obj;
while(inFile.read((char*)&obj, sizeof(obj)))
{
obj.showData();
}
inFile.close();
}
void search(int n) /* function to search and display from binary file*/
{
ifstream inFile;
inFile.open("student.dat", ios::binary);
Student obj;
while(inFile.read((char*)&obj, sizeof(obj)))
{
if(obj.retAdmno() == n)
{
obj.showData();
}
}
inFile.close();
}
void delete_record(int n) /*function to delete a record*/
{
Student obj;
ifstream inFile;
inFile.open("student.dat", ios::binary);
ofstream outFile;
outFile.open("temp.dat", ios::out | ios::binary);
while(inFile.read((char*)&obj, sizeof(obj)))
{
if(obj.retAdmno() != n)
{
outFile.write((char*)&obj, sizeof(obj));
}
}
inFile.close();
outFile.close();
remove("student.dat");
rename("temp.dat", "student.dat");
}
void modify_record(int n) /*function to modify a record*/
{
fstream file;
file.open("student.dat",ios::in | ios::out);
Student obj;
while(file.read((char*)&obj, sizeof(obj)))
{
if(obj.retAdmno() == n)
{
cout << "\nEnter the new details of student";
obj.setData();
int pos = -1 * sizeof(obj);
file.seekp(pos, ios::cur);
```

```
file.write((char*)&obj, sizeof(obj));
}
}
file.close();
}
int main()
{
for(int i = 1; i <= 4; i++)  //Store 4 records in file
write_record();
cout << "\nList of records";  //Display all records
display();
cout << "\nSearch result";  //Search record
search(100);
delete_record(100);  //Delete record
cout << "\nRecord Deleted";
cout << "\nModify Record 101 ";  //Modify record
modify_record(101);
return 0;
}
```

X. CONCLUSION

Data file handling is a wide concept in c++ and for a good programmer it is necessary to get the sufficient knowledge about handling files in c++. This paper provides sufficient basic knowledge for a programmer at a beginner stage to start with data file handling.

XI. REFERENCES

[1]Assessing programming language: a study on C and C++ by Pamela Bhattacharya and Iulian Neamtiu, University of California, Riverside, CA, USA.
[2]Computer Science with C++ by Sumita Arora.
[3]www.soundfry.com.
[4]www.myplus.com/tutorials/file-handling.com
[5]repository.readscheme.org
[6]www.cs.ucr.edu