National University of Computer & Emerging Sciences

Lecture 1
File Handling

Dr.Kifayat Alizai

Interactive vs. Batch Processing

- Interactive Program
  - what we have used to date
  - the program halts, and waits for a user to enter data from the keyboard, then proceeds...

- Batch Processing Programs
  - Non-interactive input and output
  - the user and computer do not interact while the program is running
  - the data is stored as a separate file on a disk or HD

What are FILEs in C/C++

- Storage of information in variables/arrays or pointers is (**temporary**).
- The data in these variables is lost when we shut down or switch off our system.
- Files are used for **permanent retention** of data. (Never lost during shut down or power failure).
- **Permanent retention** can be Hard Disk, Flash Drives, Floppy Disks, CD ROMs, Cloud storage etc.

Type of FILES

- **Text type files**
  - Text type files are easily ***readable by humans***, Example include (*.txt, *.cpp, *.c, *.h).
  - We can read these FILES easily by opening them in either Notepad or any other word processing system.

- **Binary type files**
  - Binary type files can't be ***readable or modify able*** by the humans.
  - Only a ***particular software*** can open and edit these FILES.
  - Example include (*.gif, *.bmp, *.jpeg, *.exe, *.obj, *.dll).

FILE accessing in Computer Science

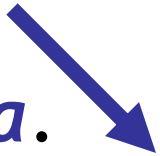
- We can access the contents of any FILE with the help of two types
  - ***Sequential access.***
  - ***Direct access (Random access).***

- **Sequential Access**
  - Files contents are accessed sequentially ***(from first to desired content)***.
  - In order to access the location ***101th***, we must have to first traverse all contents from ***0 to 100***. Then the ***101th*** location could be accessed.
  - (Normally slow if we want to access random contents in a file). ***Random access is especially used in Databases***.

Sequential Access FILE (Example)

- To access the data of the candidate of NIC# 5.
- We have to follow the following 3 steps.

1. ***First open the file.***
2. Must have to traverse all contents of ***NIC# 1, 2, 3 and 4***. Then finally we can access ***NIC# 5 data***.
3. *Close the file.*



NIC#	Candidate Name	Age
1	R. Agrawal	32
2	R. Srikant	25
3	C. Bettini	23
4	D. Burdick	40
5	R. Zaki	21
6	….	26
7	….	35

Streams

- Stream
  - a channel from where data is passed to receivers from senders.
- Output Stream
  - a channel where data is sent out to a receiver
  - cout; the standard output stream (to monitor)
  - the monitor is a *destination* device
- Input Stream
  - a channel where data is received from a sender
  - cin; the standard input stream (from the keyboard)
  - the keyboard is a *source* device

Stream Processing

- Five operations necessary for stream processing
  - the stream must be opened for use
  - if it's an input stream, *get* the next element
  - detect the end of the input stream
  - if it's an output stream, *put* the next element
  - close the stream

Standard Input / Output Streams

- Standard Streams use #include <iostream.h>
  - Standard Input Stream
    - cin names the stream
    - the extractor operator >> extracts, gets, or receives the next element from the stream
  - Standard Output Stream
    - cout names the stream
    - the inserter operator << inserts, puts, or sends the next element to the stream
  - Opening & closing of the standard streams occur automatically when the program begins & ends.

File Streams

- Files
  - data structures that are stored separately from the program (using auxiliary memory)
  - streams must be connected to these files
  - Input File Stream
    - extracts, receives, or gets data from the file
  - Output File Stream
    - inserts, sends, or puts data to the file
  - #include <fstream.h> creates two new classes
    - ofstream (**o**utput **f**ile stream)
    - ifstream (**i**nput **f**ile stream)

Output File Streams

- #include <fstream.h>
  - allows use of the two classes: ofstream, ifstream
- ofstream out_file;
  - a variable or object (out_file) is declared to be of type or of the class ofstream
- out_file.open("myfile.txt");
  - connect the output file stream to a file on the disk **in the default directory** named "myfile"
  - dot notation calls the member function "open"
  - if "myfile" exists it is opened for output & connected to the data stream out_file. If data is there, it is erased!
  - If "myfile" doesn't exist, it is created & connected to the data stream out_file

Input File Streams

- #include <fstream.h>
  - allows use of the two classes: ofstream, ifstream
- ifstream in_file;
  - a variable or object (in_file) is declared to be of type or of the class ifstream
- in_file.open("myfile.txt");
  - connect the input file stream to a file on the disk in the default directory named "myfile"
  - dot notation calls the member function "open"
  - if "myfile" exists it is opened for input & connected to the input data stream in_file.
  - If "myfile" doesn't exist …

Syntax

```
fstream dataFile;

dataFile.open("file_path", File Access Flag);

For example:
dataFile.open("info.txt", ios::out );
dataFile.open("info.txt", ios::in);
```

File Access Flags

File Access Flag	Meaning
`ios::app`	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
`ios::ate`	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
`ios::binary`	Binary mode. When a file is opened in binary mode, data are written to or read from it in pure binary format. (The default mode is text.)
`ios::in`	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
`ios::out`	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
`ios::trunc`	If the file already exists, its contents will be deleted (truncated). This is the default mode used by `ios::out`.

Example - Writing in a Text File - code # 1

```
#include <fstream>
#include <iostream>
using namespace std;

int main () {
  ofstream outputFile;
  int id;
  int marks;

  outputFile.open("/demo-data/myDemo.txt");

  cout << "Please enter your id: ";
  cin >> id;

  cout << "\nPlease enter your marks: ";
  cin >> marks;

  cout << "\nNow writing data to the file.\n";

  outputFile << "Your id is: " << id;
  outputFile << "\nYour marks are: " << marks;

  outputFile.close();

  return 0;
}
```

Example - app mode of file access - Code # 2

```
#include <fstream>
#include <iostream>
using namespace std;

int main () {
  ofstream outputFile;
  int id;
  int marks;

  outputFile.open("/demo-data/myDemo.txt", ios::out|ios::app);

  cout << "Please enter your id: ";
  cin >> id;

  cout << "\nPlease enter your marks: ";
  cin >> marks;

  cout << "\nNow writing data to the file.\n";

  outputFile << "Your id is: " << id;
  outputFile << "\nYour marks are: " << marks;

  outputFile.close();

  return 0;
}
```

Example Code # 3 - Reading from Text File

Loops And Input File Streams

- We don't always know precisely how many data values are in a file, or if any values are in a data file.
  - We do need to know the general format of the file:
    - what type of data it contains
  - We use a loop to read data **while** the end of the file has **not** been reached (**eof**)
    - **eof** returns **true** when it reaches the "end of file marker", and there is no more data to be read.
    - Otherwise **eof** returns **false**

Files and Strings

- String processing is an important part of word processing and data base applications.
- Files containing strings can be processed:
  1. a word at a time using a string variable
  2. a line at a time
     1. using getline from the string library
     2. getline is limited to 1024 characters per line
  3. a character at a time
     - necessary to handle white space characters

Processing word by word

Example code # 4

Searching for a word in a file

Example code # 4-1

Processing line by line

- **getline()** reads characters and stores them into a string variable until the end of line character is reached or until the getline limit of 1024 characters have been read.

- getline supports “buffered file input”
  - the input of large blocks of data from a file into a “buffer”
  - a “buffer” is a block of memory of a definite size where data is placed while waiting to be sent to the program.
  - The main advantages are efficiency and speed
  - The main disadvantage is that some data may exceed the limits of the buffer (1024 characters per line)

Example code # 5

Processing Character by Character

- Some problems call for the input and output of individual characters.
  - Counting lines in a file
  - Counting characters in a file
- If our data includes white space (space, tab, carriage returns) we can't use the >> extractor operator.
  - The extractor operator >> treats white space as separators between data values in an input stream.
  - Therefore, we can't use >> to input or process white space characters as their own data values.
  - C++ includes two commands to process data a character at a time including processing the white space characters.

Character Input with "get"

- General format for "get" statement
  - <input file stream>**.get**(<character value>);
  - get is called as a function with a character value as its parameter.
  - The dot notation associates the member function call with the input stream
  - get is defined so that it will treat a blank space, tab, or carriage return as valid character data
  - Whether a white space character or other character, the following will place the first character of a file into the input stream: in_file**.get**(ch);

Character Input with “get”

- Example Code # 6

Character Output with "put"

- General format for "put" statement
  - <output file stream>**.put**(<character value>);
  - put is called as a function with a character value as its parameter.
  - The dot notation associates the member function call with the output stream
  - put is defined so that it can be used with any output stream

```
for(char ch = 'a'; ch <= 'd'; ++ch)
   cout .put(ch);
```