Header Files, Stream I/O, and File Processing

Dr. Muhammad Aleem,

Department of Computer Science,

National University of Computer & Emerging Sciences,
Islamabad Campus

Header Files

Header Files

- **Header File:** A C++ **header file** contains definitions of ***Functions*** and ***Variables*.** The header file is imported using "**#include**" statement (pre-processor directive).

- **Header files** has **extension .h**

- **A header file in C/C++ contains:**
  - **Function definitions**
  - **Data-type definitions**
  - **Macros**

Header Files

Header files types:

1. **Standard library header files**: **pre-existing** header files **already available** in the **C/C++ compiler**.
2. **User-defined header files**: designed by the user.

Including header files:

1. Name **enclosed** within **angular brackets**.
2. Name **enclosed** within **double-quotes**.
   - *Preferred way* for ***user-defined files***.

```
#include<iostream>
#include"iostream"
```

Creating Header File - Example

- **Let's create our own useful library**
  - Library **providing factorial function…**

```
int factorial(int number)
{
        int iteration, factorial=1;
        for(iteration=1; iteration<=number; iteration++)
        {
                factorial=factorial*iteration;
        }
        return factorial;
}
```

  - Save it as **myMathLib.h**

Creating Header File - Example

- **Create your C++ application and include this library**

```
#include <iostream>
#include"myMathLib.h"
using namespace std;
int main()
{
    int value;
    cout<<"Welcome to my math library!"<<endl;

    cout<<"Enter a positive integer: "<<endl;
    cin>>value;

    cout<<"The factorial of " << value << " is: “;
    cout<< factorial(value) <<endl;

    return 0;
}
```

Multiple Inclusions

- **Sometime, we can end up including a header file multiple times**:
  - **C++ throws errors** on **re-definition of functions** etc.,
  - let’s see the following demo code:

myMathProgram.cpp

Avoiding Multiple Inclusions

- To **fix this issue**, use the directive **#ifndef**:
  - tells the compiler → ***ignore what follows if it has already seen this stuff before,*** Example:

```
#ifndef MYMATHLIB_H
#define MYMATHLIB_H

int factorial(int number)
{
	int iteration, factorial=1;
	for(iteration=1; iteration<=number; iteration++)
	{
		factorial=factorial*iteration;
	}
	return factorial;
}
#endif
```

Demo: myMathProgram.cpp

Stream I/O and File Processing Part 1

Interactive vs. Batch Processing

- **Interactive Program**
  - the **program halts**, and **waits** for a **user to enter data** from the keyboard, then proceeds...

- **Batch Processing Programs**
  - **Non-interactive input** and **output**
  - **User** and **computer do not interact** while the **program is running**
  - **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**).
- **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** etc.

Type of FILES

- Text type files
  - Text type files are easily ***readable by humans***, Example include (*.txt, *.cpp, *.c, *.h).

- 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:
  1. **Sequential access.**
  2. **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** where **data** are **passed** to **receivers from senders**.
- **Output Stream**
  - a **channel** where **data are sent out** to a **receiver**
  - **cout;** the **standard output stream (to monitor)**
  - the **monitor** is a ***destination*** device
- **Input Stream**
  - a **channel** where **data** are **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**
  1. the **stream must be opened** for use
  2. if it's an **input stream**, ***get*** the **next element**
  3. **detect** the **end** of the **input stream**
  4. if it's an <u>**output stream**</u>, ***put*** the **next element**
  5. **close** the **stream**

File Streams

- **Files**
  - **data structures** that are **stored separately from the program** (using **auxiliary memory**)
  - **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** (output file stream)
    - **ifstream** (input file 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.dat");**
  - **connect** the **output file stream** to a **file** on the disk **in the default directory** named "**myfile**"
  - **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**

Output File Streams

- **General Form for output file stream**

  ofstream <stream variable name>;

  <stream variable name>.open(**<file name>**);

  – the stream variable name can be any valid C++ identifier (**out_file** is a good name to use)

- The **file stream** should be **closed** **when you are finished**. If **out_file** is the **variable name** then:

  – **out_file.close( )** ; //no file name parameter used

Errors Opening & Closing Files

- C++ has a "**fail**" **function for use with file streams**

```
#include <fstream.h>
#include <assert.h>
. . .
ofstream out_file;
. . .
out_file.open("myfile.dat");
assert( ! out_file.fail( ));
//send data to the file
out_file.close( );
assert( ! out_file.fail( ));
```

Output Streams: Point 1

- **Operations on output streams are abstract.**
  - **Abstract:** **hiding the details**
  - **It doesn't matter if the output stream** is a **file** on a disk or the **monitor screen**.
  - We **only need to know** the **name of the stream** to **send the data to** the **stream**
    - use **cout** for the **monitor** or **out_file** for a **data file**

```
out_file<<"This is going to the data file.";
cout<<"This is going to the monitor screen.";
```

Output Streams: Point 2

- **Programs using output streams are portable.**
  - **Portable:** **can be transferred** to **another application** or **computer platform**, and be **recompiled without having to change the code**.
  - This works even if the **different platform uses** a **different method** of **saving files to a disk**.
  - Of course, **different systems may have different requirements** for any **filenames that are used**.

```cpp
// formato.cpp
// writes formatted output to a file, using <<
#include <fstream>                    //for file I/O
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   char ch = 'x';
   int j = 77;
   double d = 6.02;
   string str1 = "Kafka";            //strings without
   string str2 = "Proust";           //   embedded spaces

   ofstream outfile("fdata.txt"); //create ofstream object

   outfile << ch                     //insert (write) data
           << j
           << ' '                    //needs space between numbers
           << d
           << str1
           << ' '                    //needs spaces between strings
           << str2;
   cout << "File written\n";
   return 0;
   }
```

Example

- **One form of data processing:**
  - **inputs data from** the **user** at the **keyboard** (standard input stream)
  - **processes** the **data**
  - **writes** the **results** of the data processing **to a file** (output file stream)

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.dat");**
  - **connect** the **input file stream** to a **file** in the **default directory**
  - if "myfile" **exists**, it is **opened for input**
  - If "myfile" **doesn' t exist**, it is **created**

Input File Streams

- **General Form for input file stream**

  ifstream <stream variable name>;

- The **file stream should be closed** when you are **finished**:
  - **in_file.close( )** //no file name parameter used

Using Input File Streams

- **Input Streams**
  - we use the **>> extractor operator** to **get data from the keyboard**
  - **With the Standard Input Stream**
    - **user enters characters** from the **keyboard followed by a blank space**, **tab**, or **carriage return**
    - The **computer converts** the **characters** into the **data type represented** by the **identifier used**
      - int, double, char

Example

- **Input Streams**
  - we use the **>>** extractor operator to **get data from the data file**
  - With the **Input File Stream**
    - Some user has already **entered characters from the keyboard** followed by a **blank space**, **tab** or **carriage returns into a data file**
    - **We need to know:**
      - **what type of data is stored in the file**
        - » int, double, char, apstring
      - **what order the data is stored**
      - **sometimes even what type of whitespace separates the data**
        - » blank spaces, tabs, carriage returns

Loops And Input File Streams

- **We don't always know precisely how many data values are in a file**
  - **read data while** the **end of the file** has **not been reached** (**eof**)
    - **eof** returns **true** → "**end of file marker**",
    - Otherwise **eof** returns **false**

Input File Stream Example

```
in_file>> data;
while( ! in_file.eof( ))
{
     process (data);
     in_file>> data;
}
```

```
// formati.cpp
// reads formatted output from a file, using >>
#include <fstream>                          //for file I/O
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   char ch;
   int j;
   double d;
   string str1;
   string str2;


   ifstream infile("fdata.txt");    //create ifstream object
                                    //extract (read) data from it
   infile >> ch >> j >> d >> str1 >> str2;

   cout << ch << endl               //display the data
        << j << endl
        << d << endl
        << str1 << endl
        << str2 << endl;
   return 0;
   }
```

Compiler Differences

- **Some compilers** will **indicate** that the **eof** has **been reached** if file stream function **fail ( ) returns true**.
  - **Use this compound Boolean Expression to guard against this:**

```
while((!in_file.fail()) && (!in_file.eof()))
```

Processing string by string

```
//Consider searching through a file of strings looking for a desired
string (word)

void search_for_word(ifstream &in_file, const apstring
   &desired_word,int &position, bool &word_found)
{
     string input_word;
      in_file >> input_word;
     ++position;
     while( (!in_file.eof()) && (input_word != desired_word))
      {
          in_file>> input_word;
          ++position;
      }
      word_found = ! in_file.eof();
}
```

Files and Strings

- **Files containing strings can be processed:**
  1. a string (**word**) at a time **using a string variable**

  2. a **line of strings at a time**
     - using **getline** from the **string library**
     - **getline** is **limited** to **1024 characters per line**
  3. or a **character at a time**
     - necessary to handle white space characters

Processing line by line

- 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 temporary
  - 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

```
while (!in_file.eof() && ! in_file.fail())
{
      getline(in_file, line);
      cout << line << endl;
}
```

Processing string by string

```
string strvar;

in_file >> strvar;
while( (! in_file.fail() ) && ( ! in_file.eof() )
{
      cout << strvar << endl;
      in_file >> strvar;
}
cout << strvar << endl;
```

Processing line by line

```
while (! in_file.eof() && ! in_file.fail())
{
      getline(in_file, line);
      out_file << line << endl;
}
```

Processing Character by Character

- <u>**Some problems call for the input and output of individual characters.**</u>
  - **Counting characters in a file**

- **If** our **data includes white space** (**space**, **tab**, **carriage returns**)
  - The extractor **operator >> treats white space as separators**
  - Therefore, we **cannot 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 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**.
  - **put** is **defined** so that **it can be used** with **any output stream**

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

```
// ochar.cpp
// file output with characters
#include <fstream>                      //for file functions
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   string str = "Time is a great teacher, but unfortunately "
                "it kills all its pupils.  Berlioz";

   ofstream outfile("TEST.TXT");        //create file for output
   for(int j=0; j<str.size(); j++)      //for each character,
      outfile.put( str[j] );            //write it to file
   cout << "File written\n";
   return 0;
   }
```

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)**;

```
int main()
   {
   char ch;                        //character to read
   ifstream infile("TEST.TXT");    //create file for input
   while( infile )                 //read until EOF or error
      {
      infile.get(ch);              //read character
      cout << ch;                  //display it
      }
   cout << endl;
   return 0;
   }
```

Detecting eof at the character level

– A **special character** **marks** the **end of a file**.

- In an **empty file**, this is the **only character present**.
- It is **important to detect this character** and **not try to read any data beyond this marker**.
- **When** "**get**" **reads** the **eof** **function** it **returns** a Boolean value of **true**.
- ***Standard form for processing a file, character by character is:***

```
<input stream name>.get(<character variable>);
while(!<input stream name>.eof( )) {
   process_data(<character variable>);
  <input stream name>.get(<character variable>); }
```

Home Exercise-1

- ***Write a C++ program to count the number of characters in a file.***

Stream I/O and File Processing Part 2

Stream Errors

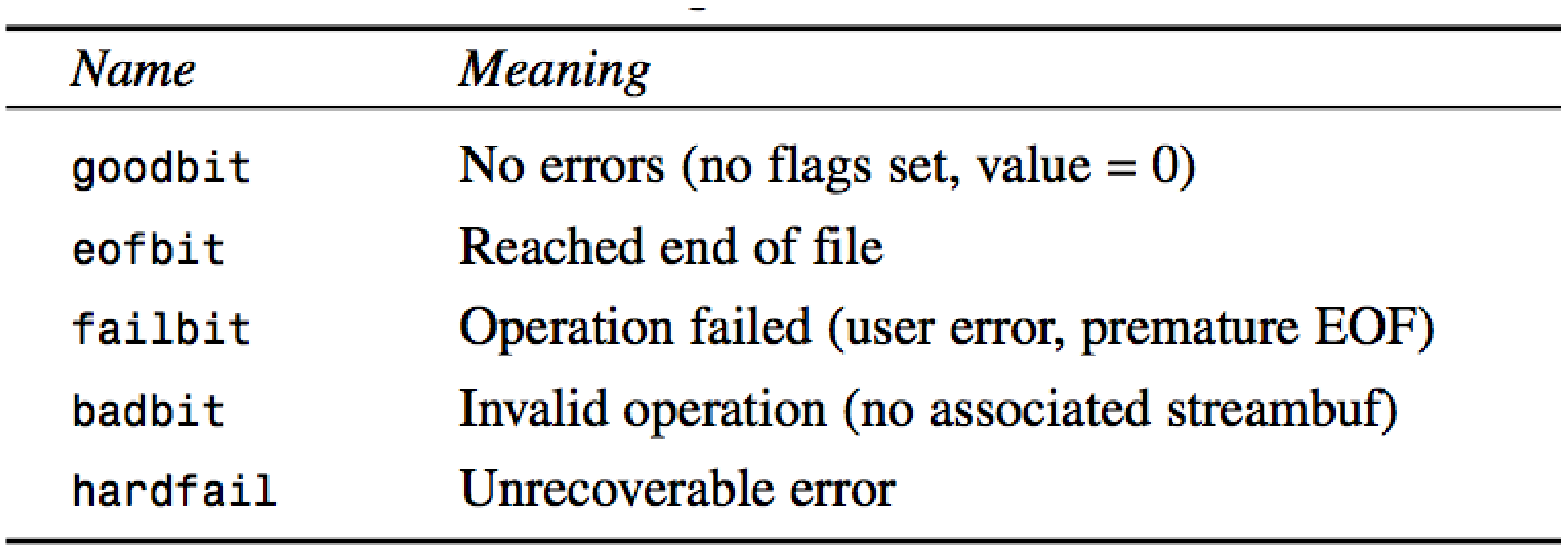
- **We have mostly used** a rather **straightforward approach** to **input** and **output**:

```
cout << "Good morning";
cin >> var;
```

What happens if a user enters the string "nine" instead of the integer 9 ?

Stream Errors

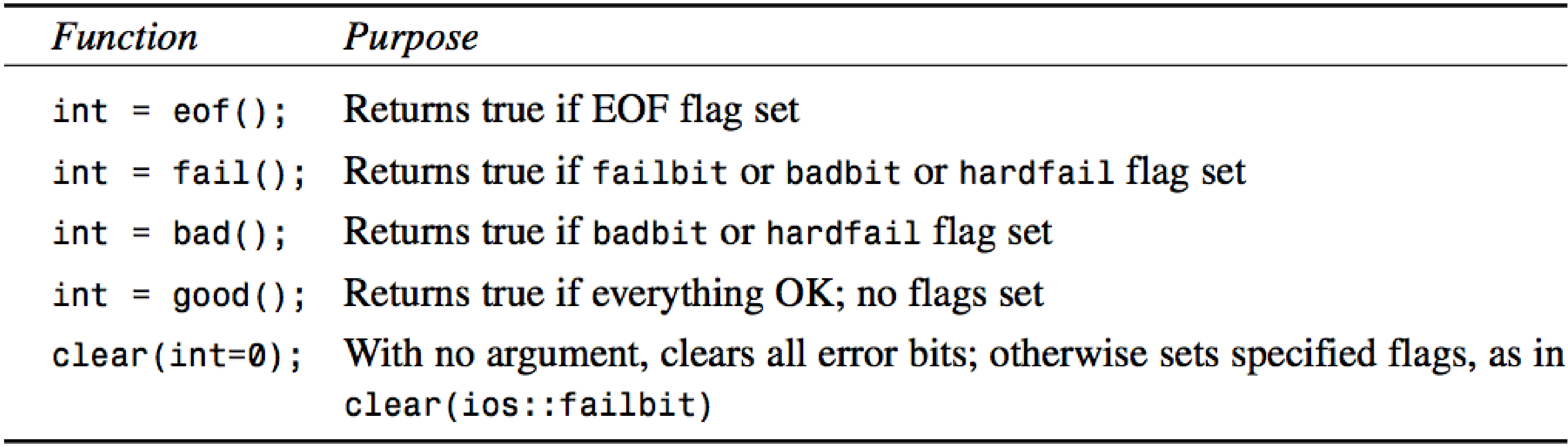
- The **stream error-status flags** constitute an **ios enum member** that **reports errors** that occurred in an **input** or **output operation**.



Name	*Meaning*
`goodbit`	No errors (no flags set, value = 0)
`eofbit`	Reached end of file
`failbit`	Operation failed (user error, premature EOF)
`badbit`	Invalid operation (no associated streambuf)
`hardfail`	Unrecoverable error

Stream Errors

- **Various** **ios functions** can be used to **read** (and even set) these **error flags**,



Function	*Purpose*
`int = eof();`	Returns true if EOF flag set
`int = fail();`	Returns true if `failbit` or `badbit` or `hardfail` flag set
`int = bad();`	Returns true if `badbit` or `hardfail` flag set
`int = good();`	Returns true if everything OK; no flags set
`clear(int=0);`	With no argument, clears all error bits; otherwise sets specified flags, as in `clear(ios::failbit)`

```
cout<<"\nEnter an integer:";
cin>>i;

 if( cin.good() ) {
        //do something
 }
```

Detecting End-of-File

```
while( !infile.eof() )  // until eof encountered

while( infile.good() )  // until any error encountered

while( infile ) // until any error encountered
```

Home Exercise-2

- ***Write a C++ program to count the number of words in a file.***

Home Exercise-3

- ***Write a C++ program to count the number of lines in a file.***

Reading Assignement

Streams and Files – Chapter 12 (Object oriented programming in C++ by *Robert Lafore*)