NWEN 241
Systems Programming

Sue Chard

suechard@ecs.vuw.ac.nz

Content

- Command line arguments
- File IO - C
- File IO – C++

C and C++

C

C++

Recap

C and C++

- In C and C++ files are defined as streams of bytes or bits
- The stream interface represents connections to data streams including networks, keyboards and files on disk
- I/O operations are defined in stdio.h or cstdio as the structure FILE that comprises
  - a *file descriptor* and
  - a *file control block*
  - one of the attributes of the file control block is file position

    In sequential files each time a character is read or written, the file position is incremented

    In random access files functions can be used to maniplulate the file position e.g. file position

    is returned by the function ftell and fseek can be used to move to a different position in the file

Recap

C and C++

A file must first be opened properly before it can be accessed for reading or writing

- When a file is opened, a stream is associated with the file
- Pointer to FILE is returned

A file should be closed when the program has finished with it

File Input / Output in C++

- fstream is the header file
- To declare file stream variables use
  - ifstream – input file stream
  - ofstream – output file stream
  - fstream – may be opened input, output or both
- Associate the file stream variables with input /output source
- Open the file
- Use insertion operator << (like cout) , extraction operator >> (like cin) or other input / output functions to read and write to files
- Close the file

Using Input/Output Files

stream - a sequence of characters

- interactive (iostream)
  - **cin** - input stream associated with **keyboard.**
  - **cout** - output stream associated with **display**

- file (fstream)
  - **ifstream** - defines new input stream (normally associated with a file).
  - **ofstream** - defines new output stream (normally associated with a file).
  - **fstream** – defines new input / output stream

Example

C ++

```
//Add additional header files you use
#include <fstream>
int main ()
{ /* Declare file stream variables such as  the following  */
ifstream  fsIn;//input
ofstream fsOut; // output
fstream fsBoth; //input & output
//Open the files
fsIn.open("prog1.txt"); //open the input file
fsOut.open("prog2.txt"); //open the output file
fsBoth.open("prog3.txt"); //open the output file
//Code for data manipulation
fsIn.close(); //Close files
fsOut.close();
fsBoth.close();
return 0; }
```

Open

- Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a disk.
- In the case of an input file:
  - the file must exist before the open statement executes.
  - If the file does not exist, the open statement fails and the input stream enters the fail state
- An output file does not have to exist before it is opened;
  - if the output file does not exist, the computer prepares an empty file for output.
  - If the designated output file already exists, by default, the old contents are erased when the file is opened.

Check for error on the open

C ++

//By checking the stream variable

```
if ( ! fsIn)
{
      cout << " Cannot open 
file.\n ";
}
```

//By using bool is_open() function

```
if ( ! fsIn_open())
{
      cout << " is not 
open.\n ";
}
```

File I/O Example: Open the file with validation

First Method (use the constructor)

```
#include <fstream>
using namespace std;
int main()
{
    //declare and automatically open
    // the file
    ofstream osFile("fout.txt");
    // Open validation
    if(! osFile) {
       cout << “Cannot open file.\n ”;
       return 1;
    }
    return 0;
}
```

Second Method (use Open function)

```
#include <fstream>
using namespace std;
int main()
{
    //declare output file variable
    ofstream osFile;
    // open an existing file fout.txt
    osFile.open("fout.txt");
    // Open validation
    if(! osFile.is_open() ) {
        cout << "Cannot open file.\n ";
        return 1;
    }
    return 0;
}
```

More Output File-Related Functions

ofstream fsOut;

fsOut.open(const char[] fname)

connects stream **fsOut** to the external file ***fname.***

fsOut.put(char character)

inserts character ***character*** to the output stream **fsOut*.***

fsOut.eof()

tests for the end-of-file condition.

File I/O Example: Writing

First Method (use the constructor)

```
#include <fstream>
using namespace std;
int main()
{/* declare and automatically open the
file*/
ofstream outFile("fout.txt");


/*behave just like cout, put the word
into  the file*/
outFile << "Hello World!";


outFile.close();


return 0;
}
```

Second Method (use Open function)

```
#include <fstream>
using namespace std;
int main()
{// declare output file variable
ofstream outFile;
// open an existing file fout.txt
outFile.open("fout.txt”);


//behave just like cout, put the
word into  the file
outFile << "Hello World!";


outFile.close();


return 0;
}
```

More Input File-Related Functions

ifstream fsin;

fsin.open(const char**[ ] fname)**

connects stream **fsin** to the external file ***fname.***

fsin.get(char **character)**

extracts next character from the input stream **fsin** and places it in the character variable ***character.***

fsin.eof()

tests for the end-of-file condition.

File I/O Example: Reading

Read char by char

```
#include <iostream>
#include <fstream>
int main()
{//Declare and open a text file
ifstream openFile("data.txt");
 char ch;
 //do until the end of file
while( !openFile.eof() )
{
   openFile.get(ch);//get one character
   cout << ch;  //display the character
}
openFile.close(); // close the file
return 0;
}
```

Read a line

```
#include <iostream>
#include <fstream>
#include <string>
int main()
{//Declare and open a text file
ifstream openFile("data.txt");
string line;
while(!openFile.eof())
{//fetch line from data.txt and put it in a
string
   getline(openFile, line);
   cout << line;
}
openFile.close(); // close the file
return 0; }
```

File Open Mode

Name	Description
ios::in	Open file to read (default for ifstream)
ios::out	Open file to write (default for ofstream)
ios::app	All the data you write, is put at the end of the file. It calls ios::out
ios::ate	All the data you write, is put at the end of the file. It does not call ios::out
ios::trunc	Deletes all previous content in the file. (empties the file)
ios::nocreate	If the file does not exists, opening it with the open() function gets impossible.
ios::noreplace	If the file exists, trying to open it with the open() function, returns an error.
ios::binary	Opens the file in binary mode.

Note. ios::in |ios::out is the default for fstream

File Open Mode

```
#include <fstream>
using namespace std;
int main(void)
{
   ofstream outFile("file1.txt", ios::out);
   outFile << "That's new!\n";
   outFile.close();
    Return 0;
}
```

To set more than one open mode, just use the **OR** operator- **|**.

ios::ate | ios::binary

Summary of Input File-Related Functions

```
#include <fstream>
ifstream fsIn;
```

`fsIn.open(const char[] fname)`

connects stream **`fsIn`** to the external file ***`fname.`***

`fsIn.get(char& c)`

extracts next character from the input stream **`fsIn`** and places it in the character variable ***`c.`***

`fsIn.eof()`

tests for the end-of-file condition.

`fsIn.close()`

disconnects the stream and associated file.

```
fsIn >> c;  //Behaves just like cin
```

Summary of Output File-Related Functions

```
#include <fstream>
ofstream fsOut;
```

`fsOut.open(const char[] fname)`

connects stream **`fsOut`** to the external file ***`fname`*.**

`fsOut.put(char c)`

inserts character ***`c`*** to the output stream **`fsOut`.**

`fsOut.eof()`

tests for the end-of-file condition.

`fsOut.close()`

disconnects the stream and associated file.

```
fsOut << c;    //Behaves just like cout
```

Example writing numbers to file

```
#include <iostream>
#include <fstream>
using namespace std;
void  main()
{
    ofstream outFile;
    // open an exist file fout.txt
        outFile.open("number.txt",ios::app);

    if (!outFile.is_open())
    { cout << " problem with opening the file ";}
    else
    {outFile <<200 <<endl ;
    cout << "done writing" <<endl;}

    outFile.close();
    return 0;
}
```

Example Reading numbers from file

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;
void main()
{//Declare and open a text file
    ifstream INFile("number.txt");
    string line;
    int total=0;
    while(! INFile.eof(){
        getline(INFile, line); //converting line string to int
        stringstream(line) >> total;
        cout << line  <<endl;
        cout <<total +1<<endl;}
    INFile.close(); // close the file
    return 0;
}
```

Random Access Files

- Use tellg() and putg() to find the current position in file
- Use seekg() for input and seekp for output each have two prototypes

  seekg (position);
  seekp (position); and

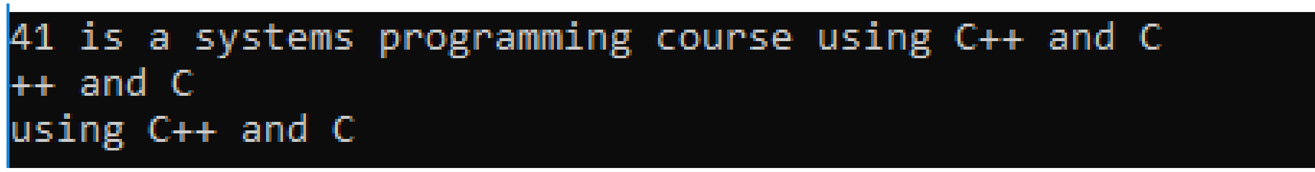
  seekg (offset, direction);
  seekp (offset, direction); direction can have the following values

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

Example

```
ifstream inf("Sample.dat");
if (!inf){         // Print an error and exit
        cerr << "file not opened" << endl;
        return 1;
    }
string strData;
inf.seekg(5); // move to 5th character
getline(inf, strData);//Get the rest of the line c
out << strData << endl; //and print it
inf.seekg(-8, ios::cur);
         //move 8 more bytes into file
getline(inf, strData);// Get rest of the line
cout << strData << endl; // and print it
inf.seekg(-15, ios::end);
         // move 15 bytes before end of file
getline(inf, strData);  // Get rest of the
cout << strData << endl;
```

The output from this code given a file containing the text:
NWEN241 is a systems programming course using C++ and C



```
41 is a systems programming course using C++ and C
++ and C
using C++ and C
```

fstream files opened for input and output

- The fstream class is capable of both reading and writing a file at the same time -- almost!
- The big caveat here is that it is not possible to switch between reading and writing arbitrarily
- Once a read or write has taken place, the only way to switch between the two is to perform an operation that modifies the file position (e.g. a seek)
- If you don’t actually want to move the file pointer (because it’s already in the spot you want), you can always seek to the current position
- Use seekg() for input and seekp for output (each have two prototypes shown on slide 21)

Example

The input file contains the text:

NWEN241 is a systems programming course using C++ and C

After the program runs it contains:

NW#N241 #s # syst#ms pr#gr#mm#ng c##rs# #s#ng C++ #nd C

```
fstream iofile(“test.dat", ios::in | ios::out);
if (!iofile)  { // If we couldn't open iofile, print an error
      cerr << "test.dat could not be opened" << endl;
       return 1;
 }
char chChar; // read character by character
while (iofile.get(chChar))   {
// While there's still data to process
 switch (chChar)  {      // If we find a vowel
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
       iofile.seekg(-1, ios::cur);
       // Because we did a seek, we can now safely do a write, so
       // let's write a # over the vowel
       iofile << '#';
        // Now we want to go back to read mode so the next call
       // to get() will perform correctly.
       //We'll seekg() to the current
       // location because we don't want to move the file pointer.
       iofile.seekg(iofile.tellg(), ios::beg);
        break;
 }
}
```