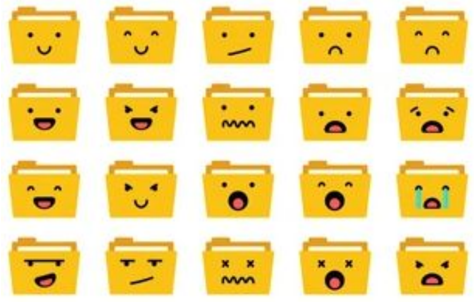


Filing

BIZARRO.COM
©2019 BIZARRO STUDIOS
Facebook.com/BizarroComics
Dist. by King Features
Wayno® & 2 Piraro. 11-2-19
NOV
BLUE BERRY
APPLE
PUMPKIN
CHERRY
FILLING CABINET

stream

/striːm/

noun

1. a small, narrow river.
   "a perfect trout stream"
   Similar: brook rivulet rill runnel streamlet freshet river

2. a continuous flow of liquid, air, or gas.
   "Frank blew out **a stream of** smoke"
   Similar: jet flow rush gush surge spurt spout torrent flood

verb

1. (of liquid, air, gas, etc.) run or flow in a continuous current in a specified direction.
   "she sat with tears **streaming down** her face"
   Similar: flow pour course run gush surge spurt flood
2. transmit or receive (data, especially video and audio material) over the internet as a steady, continuous flow.

Dictionary

Definitions from Oxford Languages · Learn more

live stream

noun

noun: **livestream**

a live transmission of an event over the internet.
"a live stream of Saturday's concert will run from 6:30 p.m."

verb

verb: **livestream**

transmit or receive live video and audio coverage of (an event) over the internet.
"you can live-stream the performance from your computer"

Input Output Stream

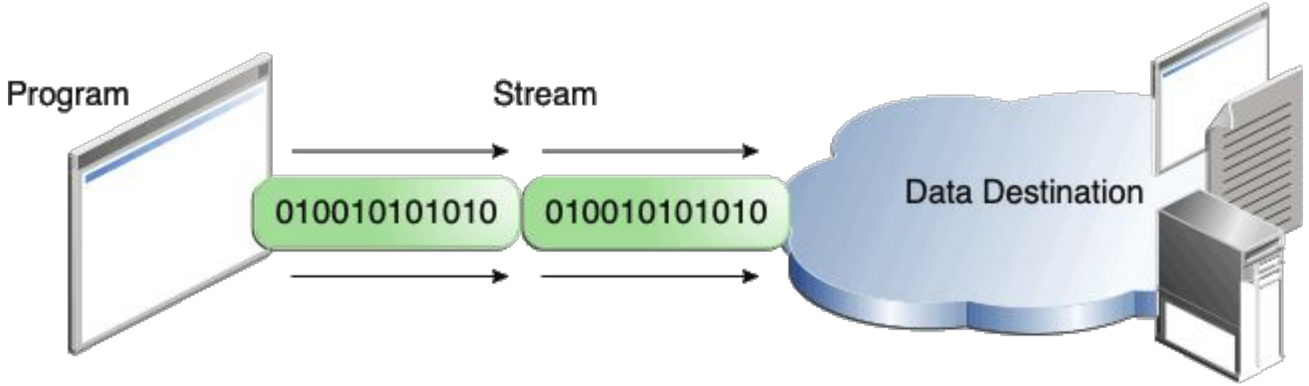
Stream is the sequences of bytes or flow of data , which acts as a source from which the input data can be obtained by a program or a destination to which the output data can be sent by the program.

Input Output Stream

A stream is an abstraction of a device where input/output operations are performed.

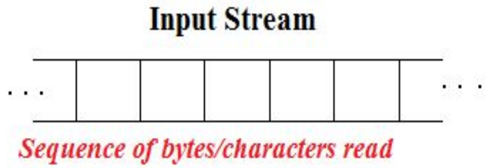
You can represent a stream as either a destination or a source of characters of indefinite length.

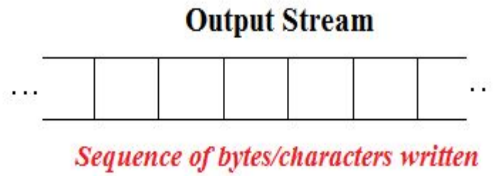
When we open a stream, one end is always attached to the program that opens the stream and the other end is attached to a file, which is accessed by its name.



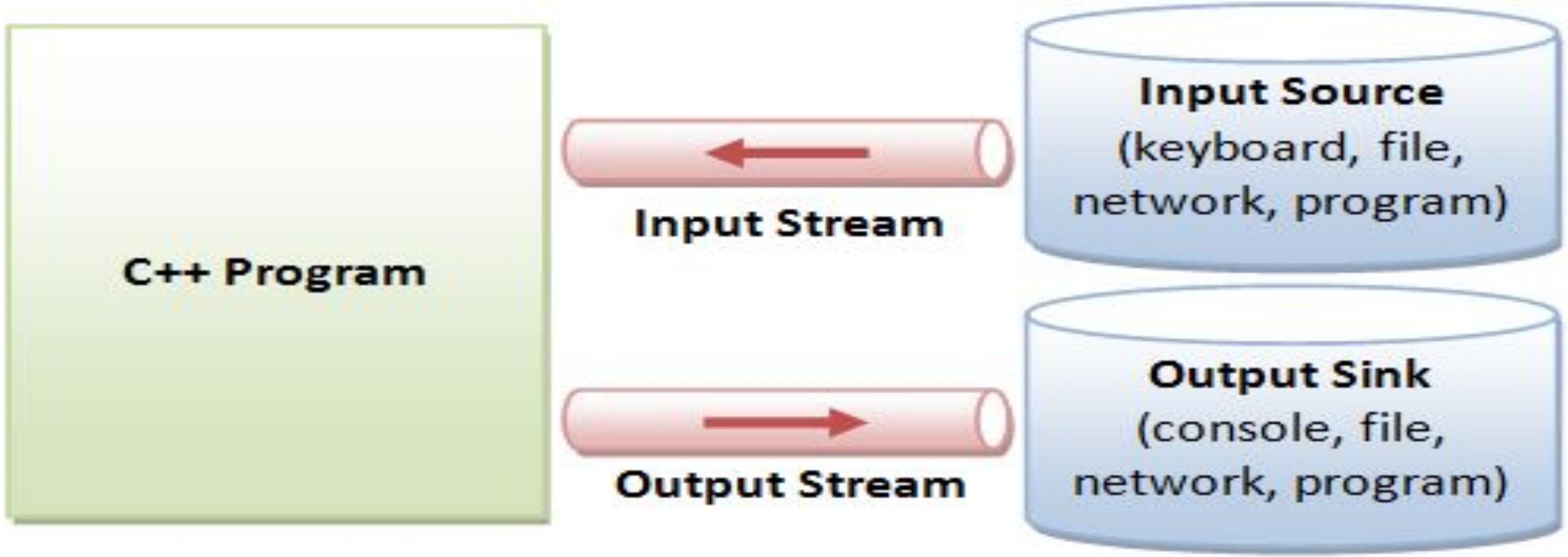
Input Output Stream

Input Stream : It is flow of data bytes from a device (e.g Keyboard , disk drive) to main memory (when we read/take file's data into a program variable)





Output Stream : It is flow of data bytes from main memory (i.e program) to a device(when we store/write variable's data into a file)


C++ Program
Input Stream
Output Stream
Input Source
(keyboard, file, network, program)
Output Sink
(console, file, network, program)

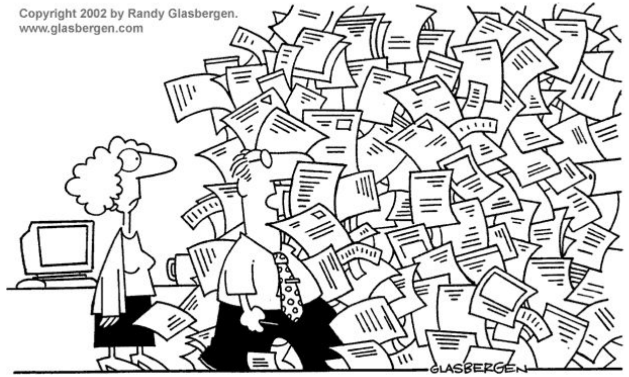
Internal Data Formats:

- Text: char, wchar_t
- int, float, double, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

FILE HANDLING



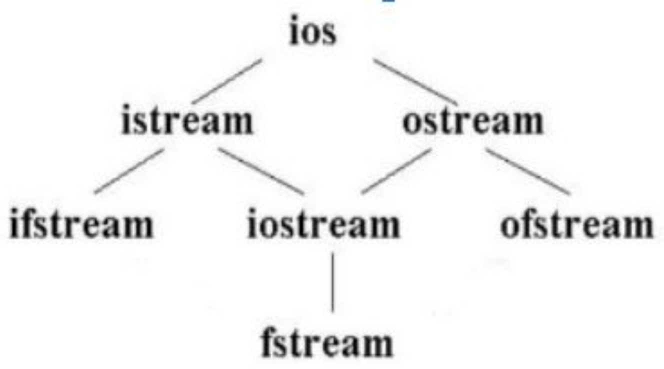
"I have some paperwork to catch up. If I'm not back in two days, organize a search and rescue team!"

Files store data permanently in a storage device. With file handling, the output from a program can be stored in a file.

Using file handling we can store our data in Secondary memory (Hard disk).

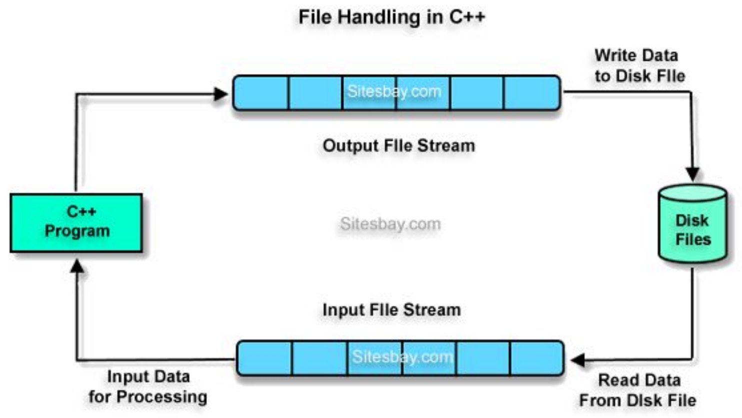
The transfer of input - data or output - data from one computer to another can be easily done by using files.

Classes for stream input and output



- ☐ ios is the base class.
- ☐ istream and ostream inherit from ios
- ☐ ifstream inherits from istream (and ios)
- ☐ ofstream inherits from ostream (and ios)
- ☐ iostream inherits from istream and ostream (& ios)

File I/O

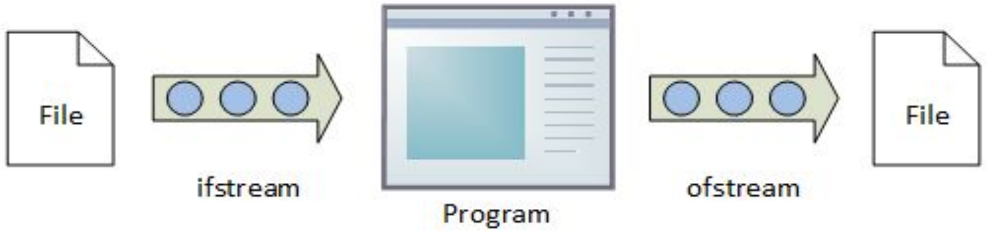


To perform file I/O, you must include the header <fstream> in your program.

It defines several classes, including ifstream, ofstream, and fstream.

C++ views each file as a sequence of bytes

File Stream



There are three types of file streams: input, output, and input/output

```
ifstream    in;     // input stream object
ofstream  out;     // output stream object
fstream     io;    // input/output stream object
```

Ofstream predefined methods

This class is used to prepare an object which performs write operations on file.

open()

write()

put()

seekp()

tellp()

close()

IFSTREAM PREDEFINED METHODS

It is used to prepare an object which performs read operations on file.

open()

read()

get()

getline()

seekg()

tellg()

close()

FSTREAM PREDEFINED METHODS

It performs both read and write operations of file.

Reading from a file

```
//open a file for reading using constructor
ifstream    in(“myfile”);


//open a text file for reading
ifstream in(“myfile.txt”);
```

If for some reason, file cannot be opened then ifstream object has the value false

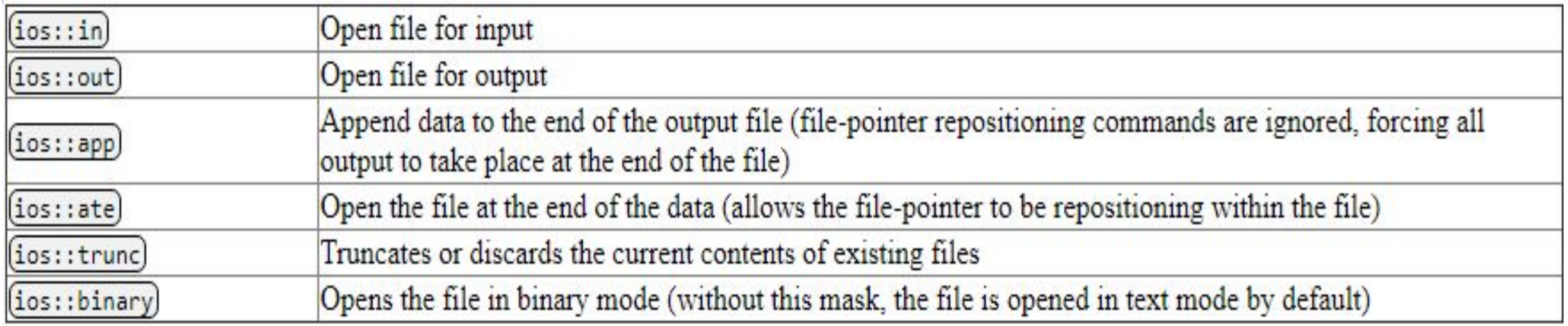
Alternate Syntax

An alternate way to open a file for read/write is by using open() function

```
ofstream out;
out.open("test", ios::out);
```

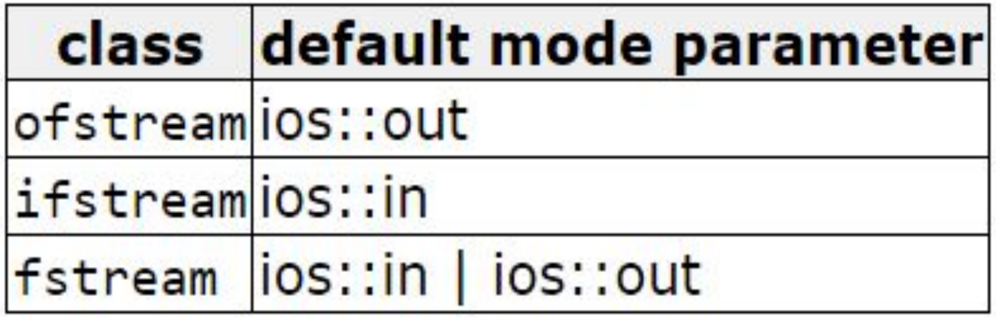
IOS MODES

“We have a VP of Records Management, but we don’t know who it is because nobody can locate the file.”



`ios::in`	Open file for input
`ios::out`	Open file for output
`ios::app`	Append data to the end of the output file (file-pointer repositioning commands are ignored, forcing all output to take place at the end of the file)
`ios::ate`	Open the file at the end of the data (allows the file-pointer to be repositioning within the file)
`ios::trunc`	Truncates or discards the current contents of existing files
`ios::binary`	Opens the file in binary mode (without this mask, the file is opened in text mode by default)

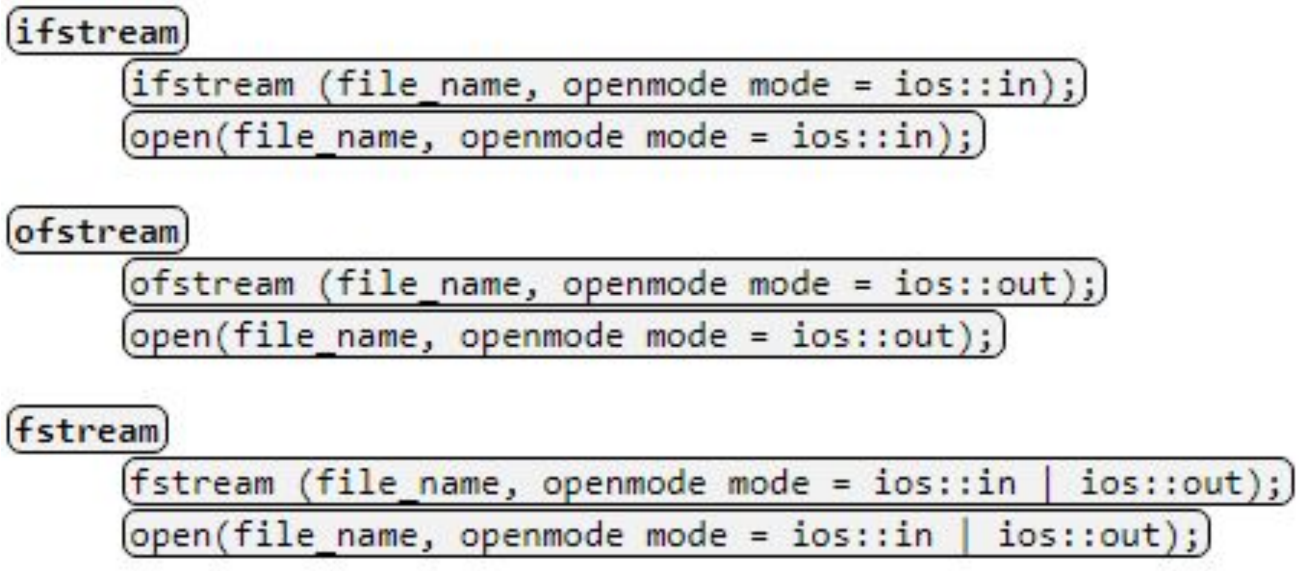
Default Modes



class	default mode parameter
`ofstream`	`ios::out`
`ifstream`	`ios::in`
`fstream`	`ios::in | ios::out`

© Randy Glasbergen.
www.glasbergen.com
GLASBERGEN
"Organized people are just people who are too lazy to look for things!"

Prototypes for the stream constructors and open functions



```
ifstream
        ifstream (file_name, openmode mode = ios::in);
        open(file_name, openmode mode = ios::in);

ofstream
        ofstream (file_name, openmode mode = ios::out);
        open(file_name, openmode mode = ios::out);

fstream
        fstream (file_name, openmode mode = ios::in | ios::out);
        open(file_name, openmode mode = ios::in | ios::out);
```

ORING

All these flags can be combined using the bitwise operator OR (|).

```
ofstream myfile;

myfile.open ("example.bin", ios::out | ios::app );
```

Closing a file

To close a file, we can use the member function close( )

```
mystream.close();
```

File open using constructor method

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    ofstream f("XYZ");
    f << "hello";
    f.close();
}
```

Multiple File open using open method with same filestream

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    ofstream f;
    f.open("file1");
    f << "hello";
    f.close();
      //    second file through the same file stream
    f.open("file2");
    f<<"i am a student";
    f.close();
}
```

Types of I/O

C++ provides both the formatted and unformatted IO functions.

In formatted or high-level IO, bytes are grouped and converted to types such as int, double, string or user-defined types.

In unformatted or low-level IO, bytes are treated as raw bytes and unconverted.

Formatted IO operations are supported via overloading the stream insertion (<<) and stream extraction (>>) operators

Types of I/O

We can perform either formatted or unformatted I/O with file stream

Formatted output is carried out on streams via the stream insertion << and stream extraction >> operators

Character translations are performed between console window and files

FORMATTED OUTPUT AND UNFORMATTED OUTPUT

The formatted output functions (via overloaded stream insertion operator <<) convert numeric values (such as int, double) from their internal representations (e.g., 16-/32-bit int, 64-bit double) to a stream of characters that representing the numeric values in text form.

The unformatted output functions (e.g., put(), write()) outputs the bytes as they are, without format conversion.

FORMATTED INPUT AND UNFORMATTED INPUT

In formatting input, via overloading the >> extraction operator, it converts the text form (a stream of character) into internal representation (such as 16-/32-bit int, 64-byte double).

In unformatting input, such as get(), getlin(), read(), it reads the characters as they are, without conversion.

FORMATTED INPUT AND UNFORMATTED INPUT

Formatted I/O can be performed by using extraction >> and insertion << operators

All information is stored in the file in the same format as it would be displayed on the screen

When reading text files using the >> operator, certain character translations occur. For example, white-space characters are omitted

Formatted I/O

Formatted output converts the internal binary representation of the data to ASCII characters which are written to the output file.

Formatted input reads characters from the input file and converts them to internal form.

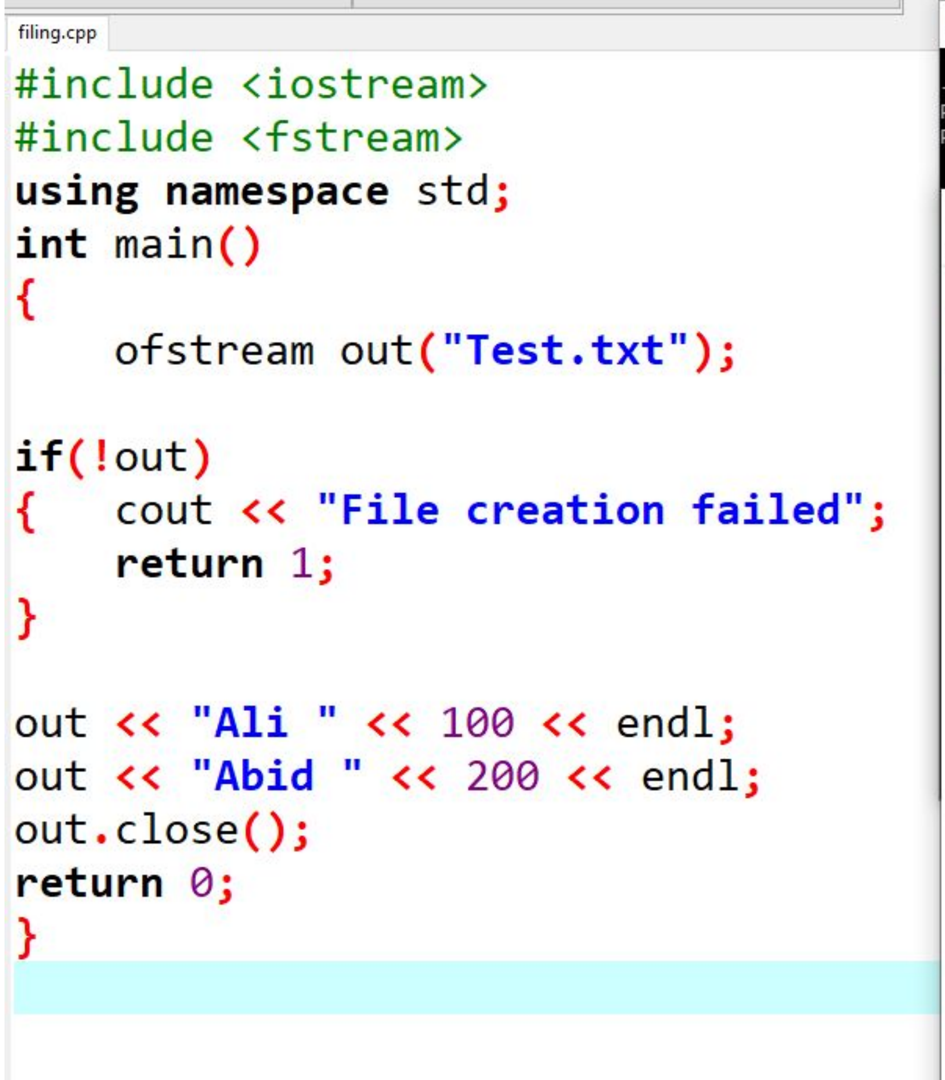
Advantages and Disadvantages of Formatted I/O

Formatted input/output is very portable. It is a simple process to move formatted data files to various computers, even computers running different operating systems, as long as they all use the ASCII character set.

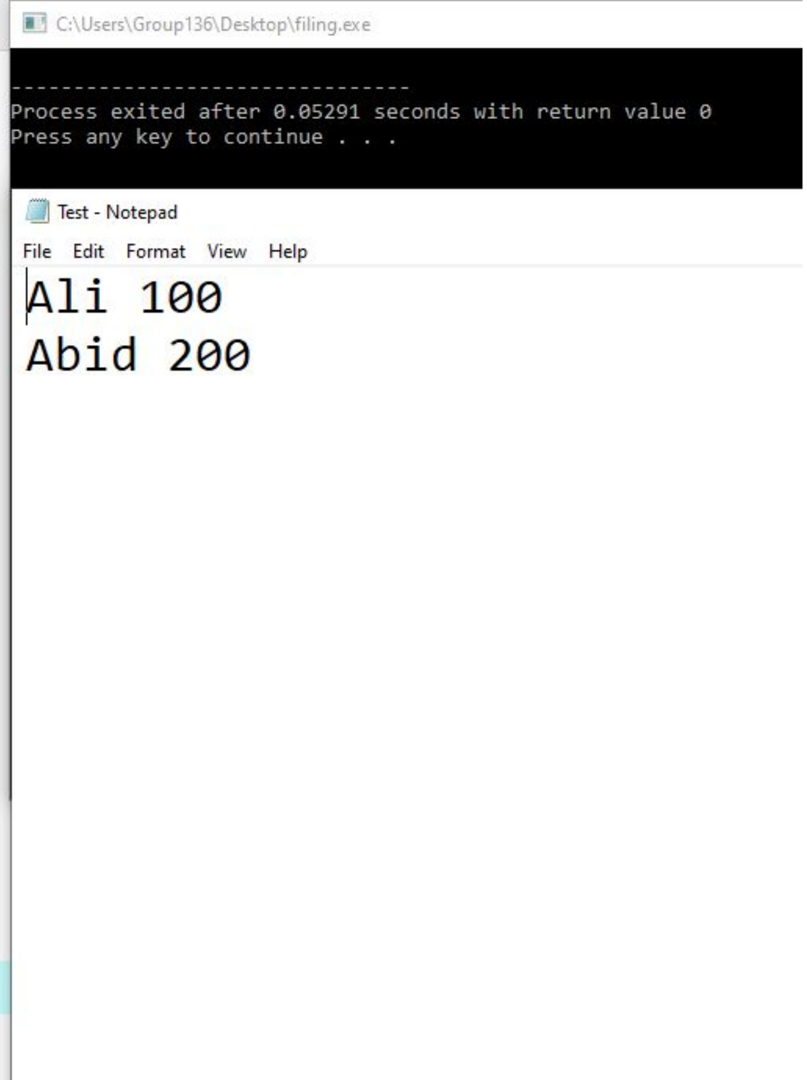
Formatted files are human readable and can be typed to the terminal screen or edited with a text editor.

To write to the file, use the insertion operator (<<)

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

int main() {
  // Create and open a text file
  ofstream MyFile("filename.txt");

  // Write to the file using insertion operator
  MyFile << "Files can be tricky, but it is fun enough!";

  // Close the file
  MyFile.close();
}
```



```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream out("Test.txt");

if(!out)
{   cout << "File creation failed";
    return 1;
}

out << "Ali " << 100 << endl;
out << "Abid " << 200 << endl;
out.close();
return 0;
}
```


C:\Users\Group136\Desktop\filing.exe

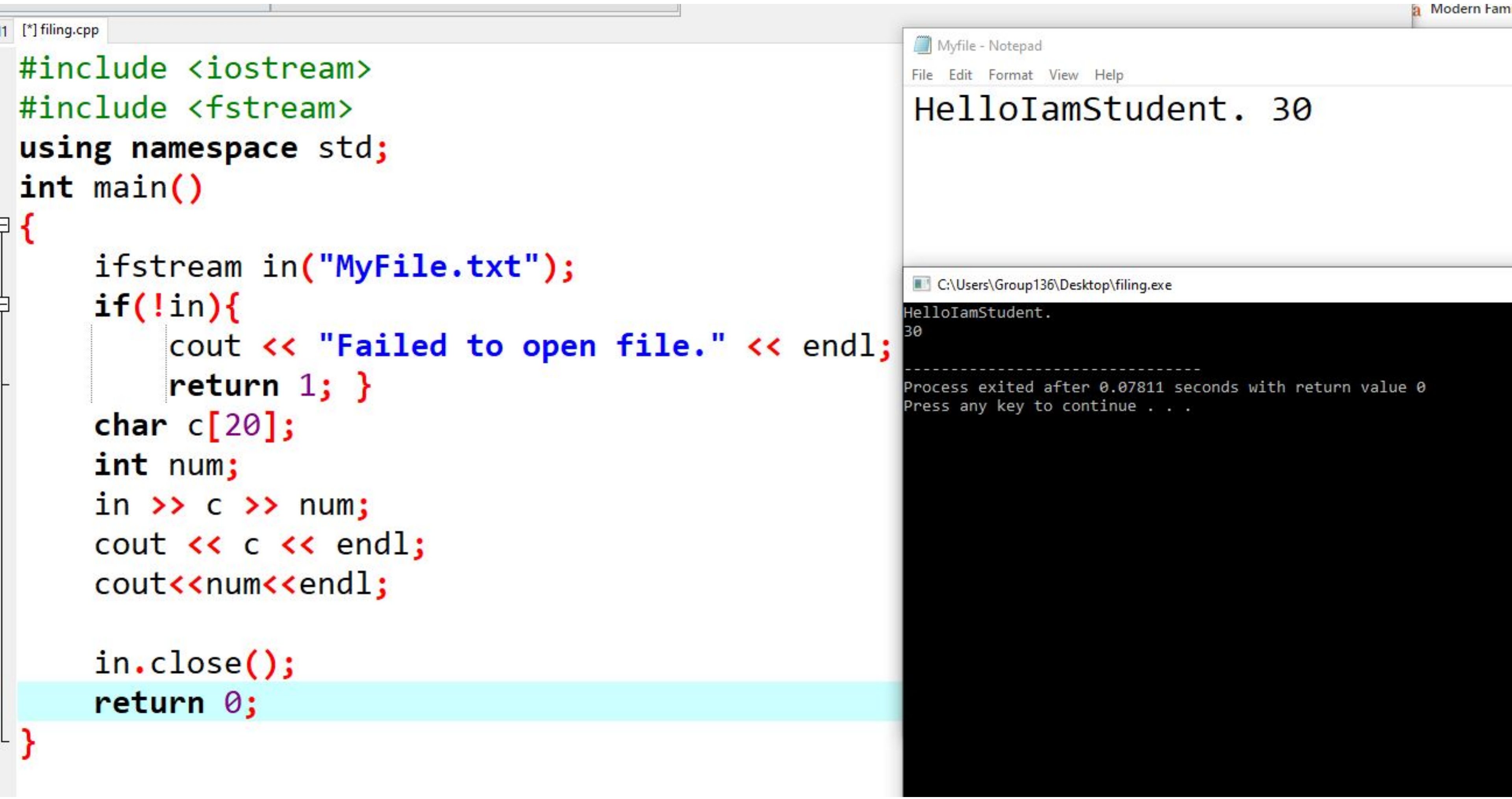
```
--------------------------------
Process exited after 0.05291 seconds with return value 0
Press any key to continue . . .
```

Test - Notepad
File Edit Format View Help

```
Ali 100
Abid 200
```

Code

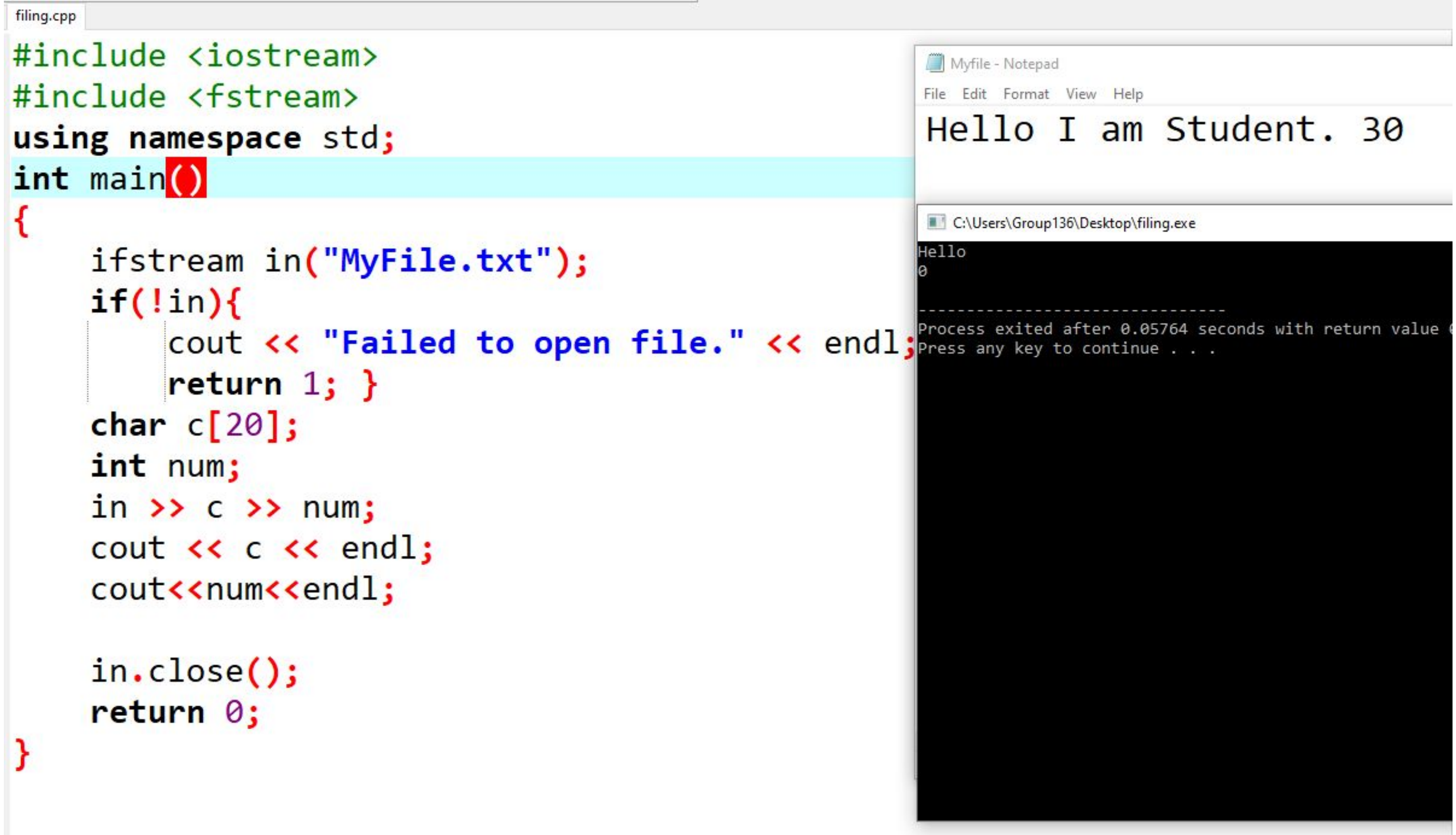
```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
     ofstream out("Test.txt");

if(!out)
{    cout << "File creation failed";
     return 1;
}

out << "Ali " << 100 << endl;
out << "Abid " << 200 << endl;
out.close();
return 0;
}
```


[*] filing.cpp
Modern Fam
Myfile - Notepad
File Edit Format View Help
C:\Users\Group136\Desktop\filing.exe

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("MyFile.txt");
    if(!in){
        cout << "Failed to open file." << endl;
        return 1; }
    char c[20];
    int num;
    in >> c >> num;
    cout << c << endl;
    cout<<num<<endl;

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

```
HelloIamStudent. 30
```

```
HelloIamStudent.
30

--------------------------------
Process exited after 0.07811 seconds with return value 0
Press any key to continue . . .
```



filing.cpp

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("MyFile.txt");
    if(!in){
        cout << "Failed to open file." << endl;
        return 1; }
    char c[20];
    int num;
    in >> c >> num;
    cout << c << endl;
    cout<<num<<endl;

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

Myfile - Notepad

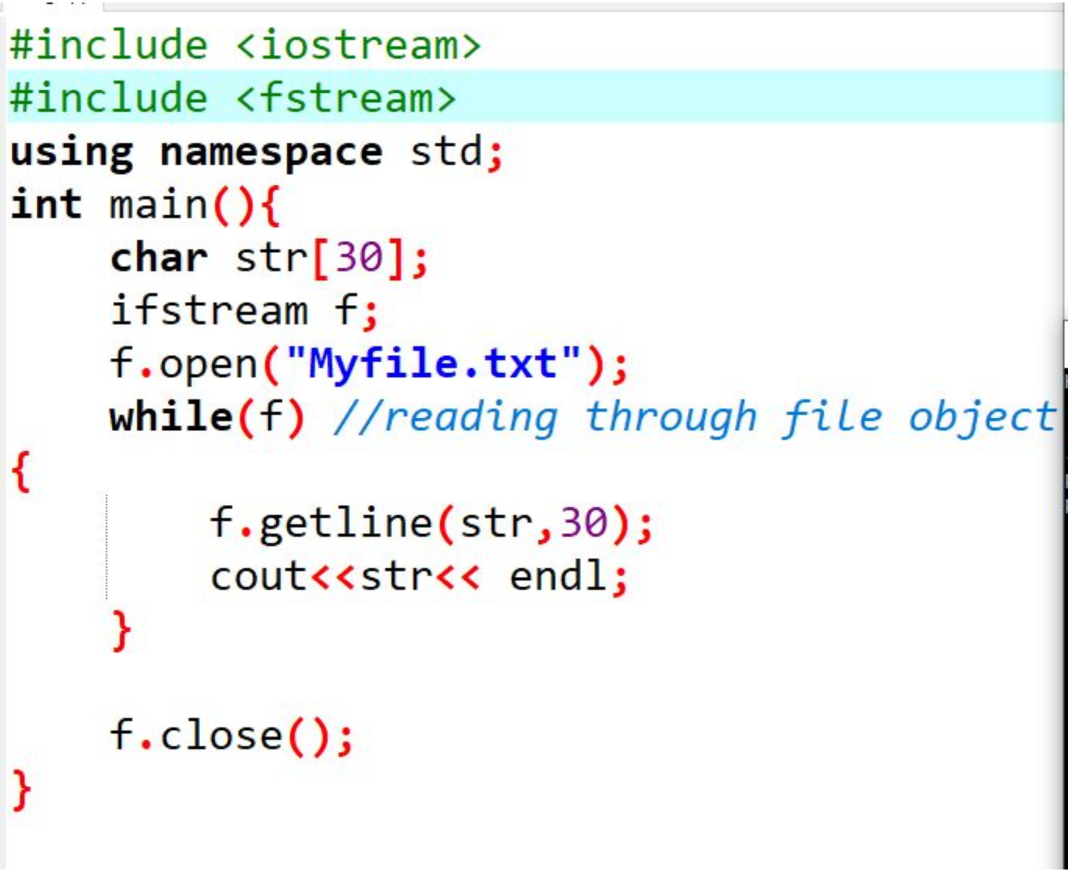
```
Hello I am Student. 30
```

C:\Users\Group136\Desktop\filing.exe

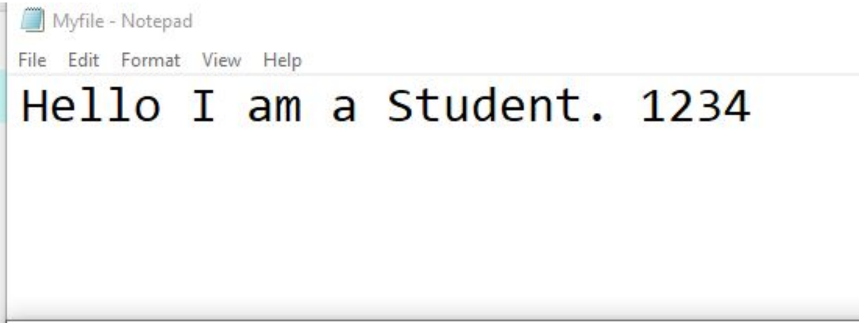
```
Hello
0

--------------------------------
Process exited after 0.05764 seconds with return value
Press any key to continue . . .
```

Code - File Read through char array

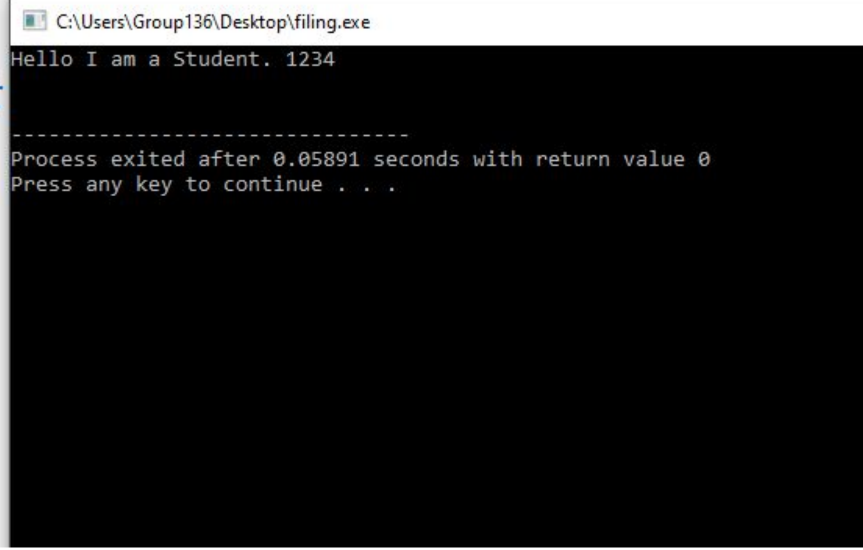
```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    char str[12];
    ifstream f;
    f.open("MyFile.txt");
    while(f) //reading through file object
{
        f.getline(str,10);
        cout<<str<< endl;
    }

    f.close();
}
```



```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    char str[30];
    ifstream f;
    f.open("Myfile.txt");
    while(f) //reading through file object
{
        f.getline(str,30);
        cout<<str<< endl;
    }

    f.close();
}
```



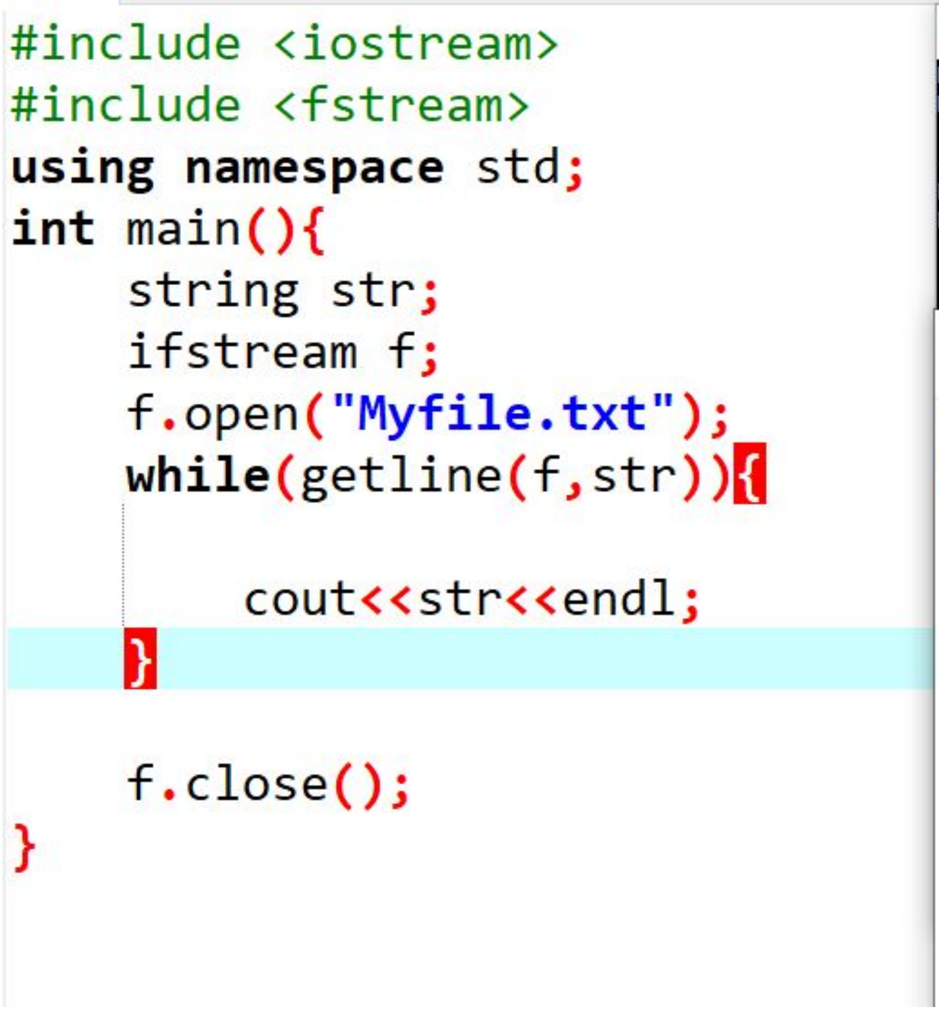
```
Hello I am a Student. 1234
```



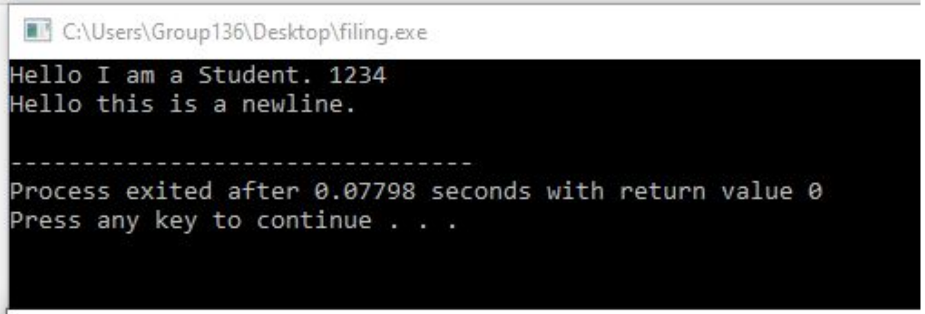
```
Hello I am a Student. 1234


--------------------------------
Process exited after 0.05891 seconds with return value 0
Press any key to continue . . .
```

Code - File Read through string object

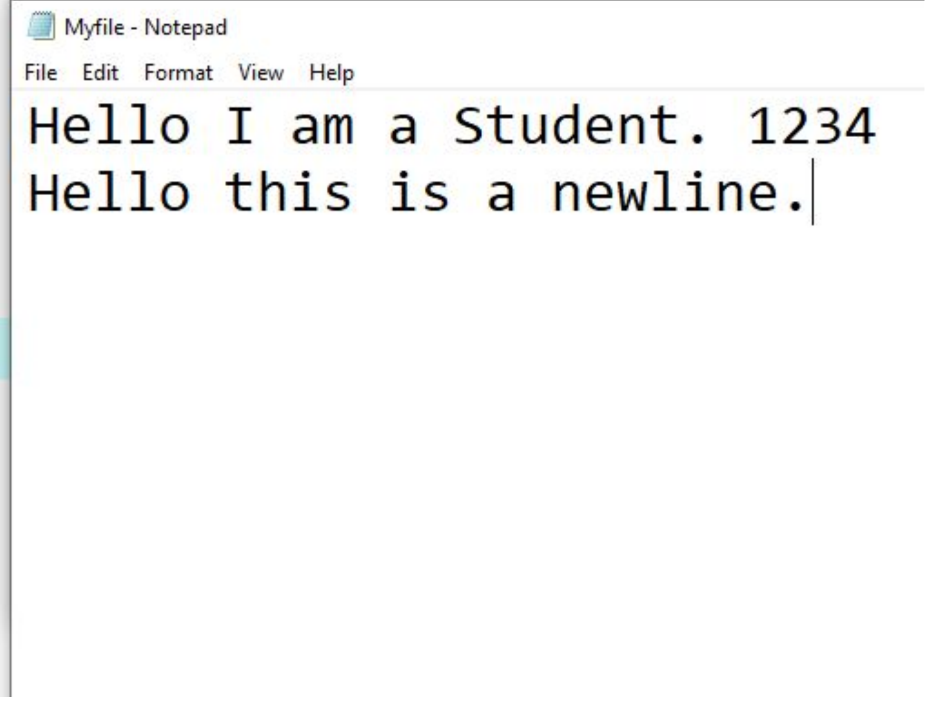
```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    string str;
    ifstream f;
    f.open("Myfile.txt");
    while(getline(f,str)){

        cout<<str<<endl;
    }

    f.close();
}
```



```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    string str;
    ifstream f;
    f.open("Myfile.txt");
    while(getline(f,str)){

        cout<<str<<endl;
    }

    f.close();
}
```



```
Hello I am a Student. 1234
Hello this is a newline.

--------------------------------
Process exited after 0.07798 seconds with return value 0
Press any key to continue . . .
```



```
Hello I am a Student. 1234
Hello this is a newline.
```

Code - File read and write

```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
    fstream file;
    file.open("smple.txt",ios::out);

    if(!file){
        cout<<"error";
        return 0;
    }
    cout <<"File created" << endl;
    file << "hi i am";
    file.close();
```

Code - File read and write

```
file.open("smple.txt",ios::in);
     if(!file){
          cout<<"error";
          return 0;
     }
     char ch[40];
     cout <<"contents are: ";
     while(!file.eof()) //reading through eof
{
          //file>>ch;
          file.getline(ch,40);
          cout<<ch ;
     }
     file.close();
     return 0;
}
```

Unformatted

Unformatted I/O

When we need to store unformatted (raw) binary data (not text) in a file, we can make use of the following set of functions

When performing binary operations on a file, we open it using the **ios::binary** mode specifier

Although unformatted file functions can work on text files, some character translations may still occur

Unformatted I/O

Unformatted Input/Output is the most basic form of input/output. Unformatted input/output transfers the internal binary representation of the data directly between memory and the file

Unformatted I/O

The unformatted output functions (e.g., put(), write()) outputs the bytes as they are, without format conversion.

In unformatting input, such as get(), getline(), read(), it reads the characters as they are, without conversion.

Advantages and Disadvantages of Unformatted I/O

- Unformatted input/output is the simplest and most efficient form of input/output. It is usually the most compact way to store data. Unformatted input/output is the least portable form of input/output. Unformatted data files can only be moved easily to and from computers that share the same internal data representation.
- Unformatted input/output is not directly human readable, so you cannot type it out on a terminal screen or edit it with a text editor.

Get/Put Functions

The functions get() and put() reads and writes a single character to a file, respectively

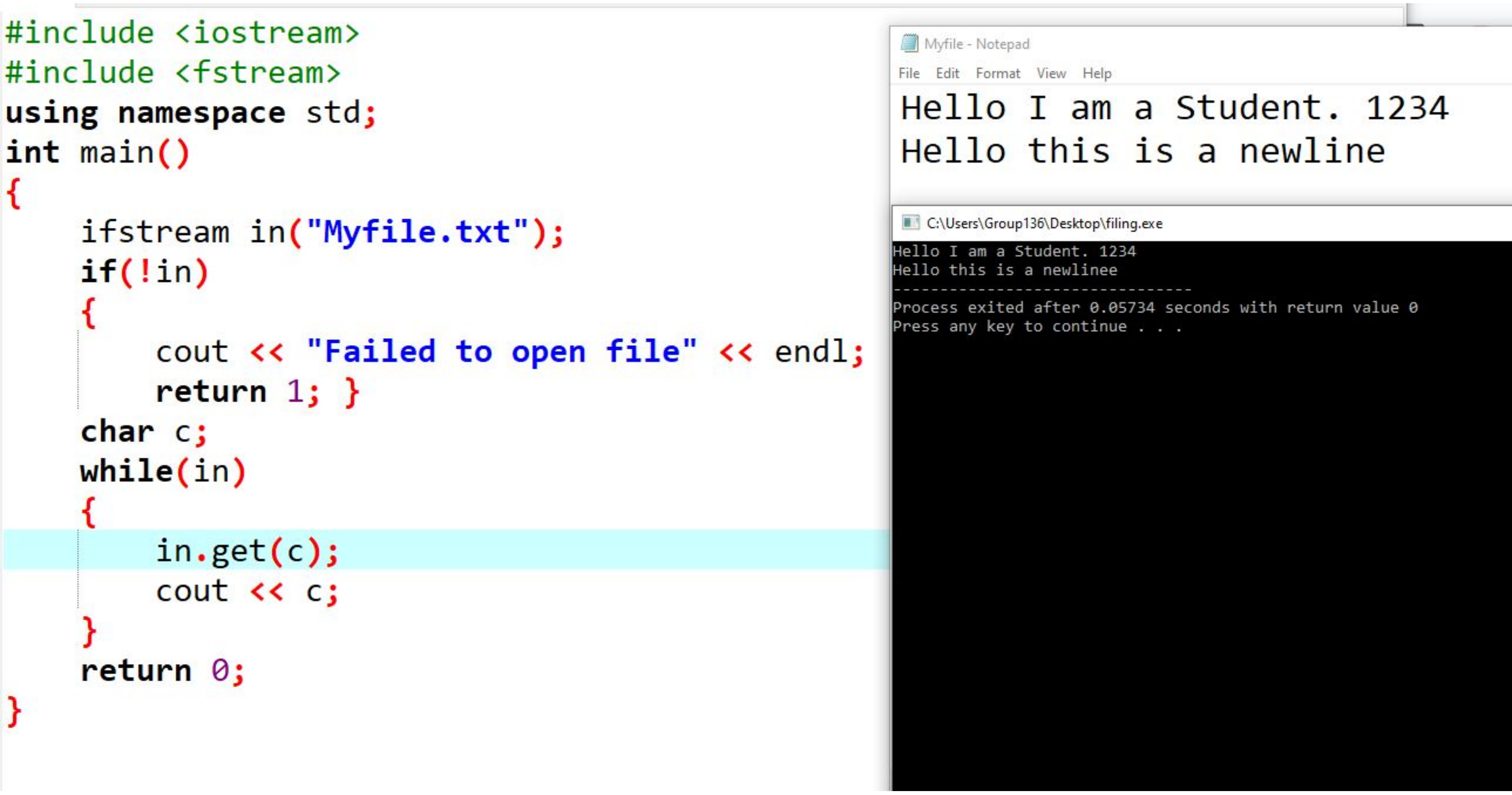
get() Function

- istream &get(char &*ch*);

- The **get( ) function reads a single character from the invoking stream and puts that** value in *ch*. *It returns a reference to the stream*

Code

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
     ifstream in("Myfile.txt");
     if(!in)
     {
          cout << "Failed to open file" << endl;
          return 1; }
     char c;
     while(in)
     {
          in.get(c);
          cout << c;
     }
     return 0;
}
```



```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("Myfile.txt");
    if(!in)
    {
        cout << "Failed to open file" << endl;
        return 1; }
    char c;
    while(in)
    {
        in.get(c);
        cout << c;
    }
    return 0;
}
```

Myfile - Notepad
File Edit Format View Help

```
Hello I am a Student. 1234
Hello this is a newline
```

C:\Users\Group136\Desktop\filing.exe

```
Hello I am a Student. 1234
Hello this is a newlinee
--------------------------------
Process exited after 0.05734 seconds with return value 0
Press any key to continue . . .
```

eof() Function

bool eof( );

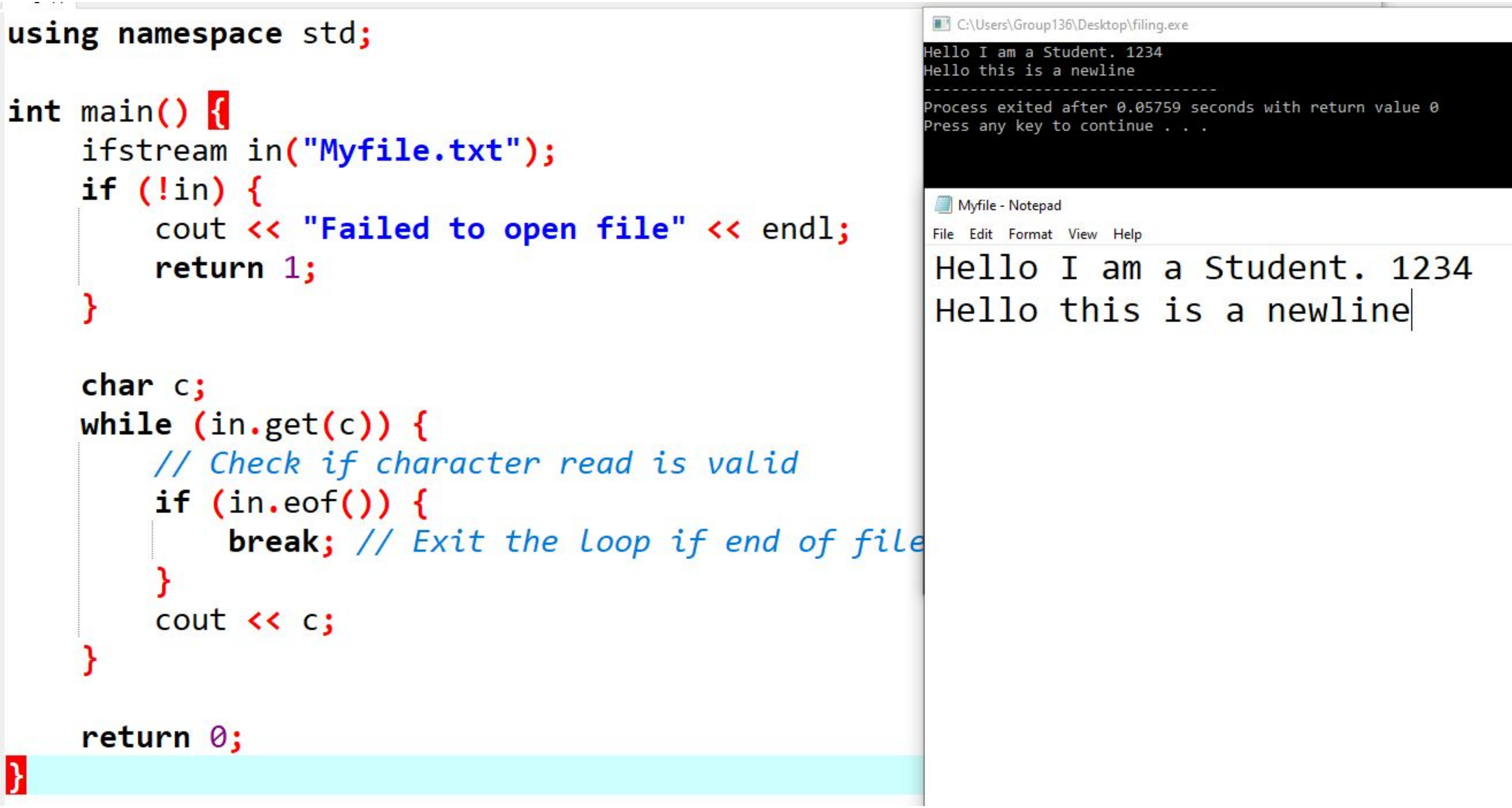
- You can detect when the end of the file is reached by using the member function **eof( )**
- It returns *true* when the end of the file has been reached; otherwise it returns *false*

FLAWED BEHAVIOR

Unfortunately, the behavior of the eof( ) function is not as straightforward as we might expect. The function does not actually test the file to see if there is more data to read. Instead, the eof( ) function returns the current value of the eof-bit (one of the stream's state flags), which is set by the last *read* function to run. That means that the value that eof( ) returns depends on the outcome of a different, previously run, function. This unexpected behavior is usually only a problem when reading single characters from a file - functions reading more complex data generally set the eof-bit as a part of the read operation.

Code

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream in("Myfile.txt");
    if (!in) {
        cout << "Failed to open file" << endl;
        return 1;
    }
    char c;
    while (in.get(c)) {
        // Check if character read is valid
        if (in.eof()) {
            break; // Exit the loop if end of file is reached
        }
        cout << c;
    }

    return 0;
}
```



```cpp
using namespace std;

int main() {
    ifstream in("Myfile.txt");
    if (!in) {
        cout << "Failed to open file" << endl;
        return 1;
    }

    char c;
    while (in.get(c)) {
        // Check if character read is valid
        if (in.eof()) {
            break; // Exit the loop if end of file
        }
        cout << c;
    }

    return 0;
}
```

C:\Users\Group136\Desktop\filing.exe

```
Hello I am a Student. 1234
Hello this is a newline
--------------------------------
Process exited after 0.05759 seconds with return value 0
Press any key to continue . . .
```

Myfile - Notepad

File Edit Format View Help

```
Hello I am a Student. 1234
Hello this is a newline
```

put() Function

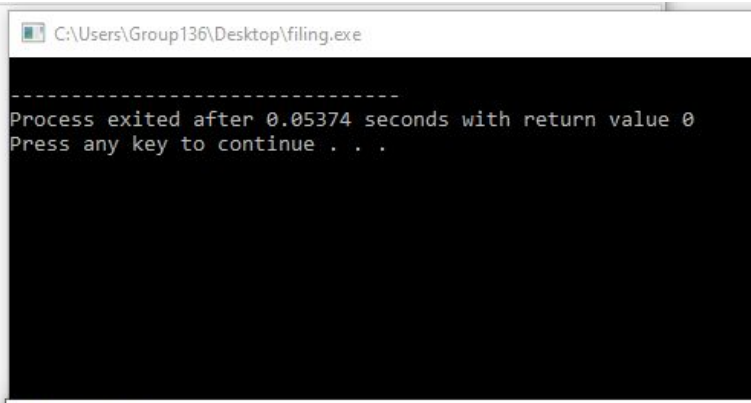
- `ostream &put(char ch);`

- *The* ***put( ) function writes ch to the*** stream and returns a reference to the stream

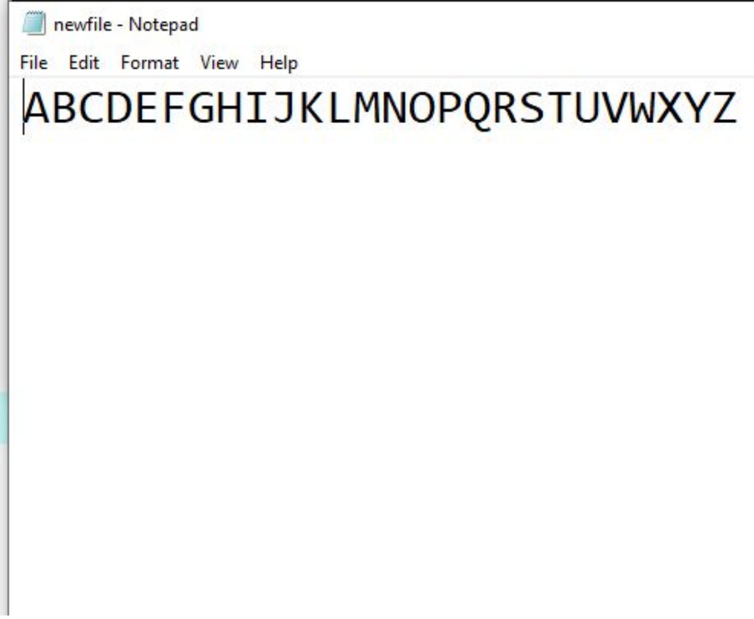
Code

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
     ofstream o("newfile.txt");
     if(!o)
     {
          cout << "Failed to open file" << endl;
          return 1;
     }
     for(int i = 65; i <= 90; i++)
     {
          o.put(i);
     }
     o.close();
     return 0;
}
```

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

int main()
{
    ofstream o("newfile.txt");
    if(!o)
    {
        cout << "Failed to open file" << endl;
        return 1;
    }
    for(int i = 65; i <= 90; i++)
    {
        o.put(i);
    }
    o.close();
    return 0;
}
```


C:\Users\Group136\Desktop\filing.exe

```
--------------------------------
Process exited after 0.05374 seconds with return value 0
Press any key to continue . . .
```


newfile - Notepad

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Read/Write Functions

- The functions **read()** and **write()** are similar to get() and put() except that we can read and write entire blocks of bytes (e.g. character array) in a file

```
istream &read(char *buf, int num);
ostream &write(const char *buf, int num);
```

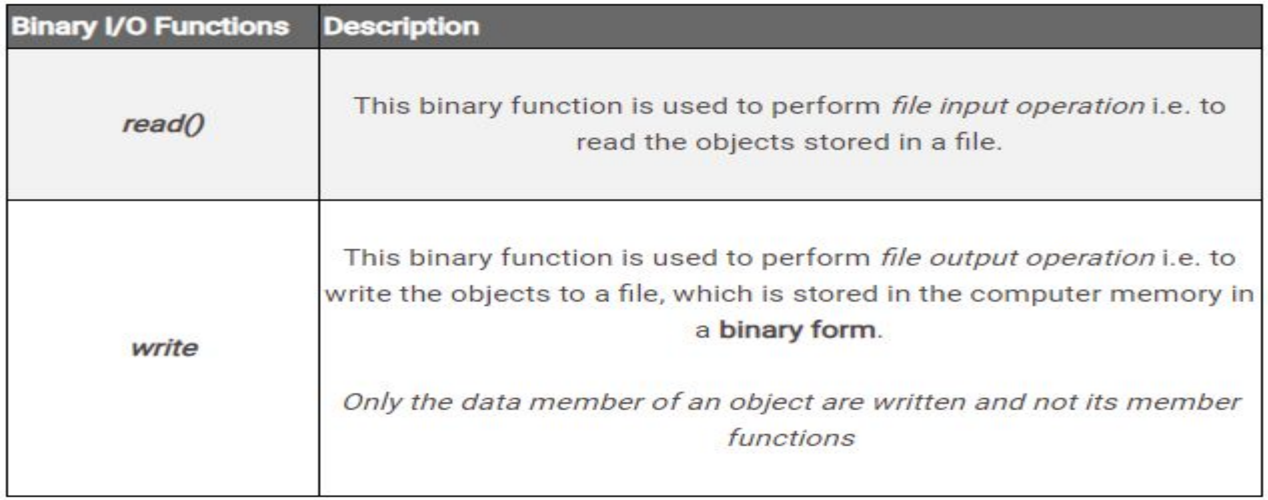
Example

```
// Writing block (array) of characters
char chW[20] = {'T', 'h', 'i', 's', 'i', 's', 'a',
't','e','s','t'};
o.write(chW, 20);

// Reading block (array) of characters
char chR[20];
i.read(chR, 20);
for(int i = 0; i < 20; i++)
{
 cout << chR[i];
}
```

C++
Binary *read()* and *write()* Functions

- Write an object of a class to this file, by using the ***write()*** function.
- **write((char *) & ob, sizeof(ob));**
- Read the stored object from the file, by using the ***read()*** function.
- **read((char *) & ob, sizeof(ob));**

C++
Binary *read()* and *write()* Functions



Binary I/O Functions	Description
read()	This binary function is used to perform *file input operation* i.e. to read the objects stored in a file.
write	This binary function is used to perform *file output operation* i.e. to write the objects to a file, which is stored in the computer memory in a **binary form**. *Only the data member of an object are written and not its member functions*

Code - Object Read & Write

.cpp file

GOOD LUCK

FOR YOUR

PROJECTS

File pointers

- Every file maintains two pointers called get_pointer (in input mode file) and put_pointer (in output mode file) which tells the current position in the file where reading or writing will takes place. These pointers help attain random access in file. That means moving directly to any location in the file instead of moving through it sequentially.

- There may be situations where random access in the best choice. For example, if you have to modify a value in record no 21, then using random access techniques, you can place the file pointer at the beginning of record 21 and then straight-way process the record. If sequential access is used, then you'll have to unnecessarily go through first twenty records in order to reach at record 21.

The seekg(), seekp(), tellg() and tellp() Functions

- random access is achieved by manipulating seekg(), seekp(), tellg() and tellp() functions. The seekg() and tellg() functions allow you to set and examine the get_pointer, and the seekp() and tellp() functions perform these operations on the put_pointer.
- The seekg() and tellg() functions are for input streams (ifstream) and seekp() and tellp() functions are for output streams (ofstream).

Random Access

- We can move the pointer (while reading or writing) to a specific position in a file

```
istream &seekg(int offset, origin);
ostream &seekp(int offset, origin);
```

Where origin can be any of the three following options

Origin Options

- `ios::beg        // Beginning-of-file position`
- `ios::cur        // Current location position`
- `ios::end        // End-of-file position`

- fin.seekg(30); // will move the get_pointer (in ifstream) to byte number 30 in the file
- fout.seekp(30);// will move the put_pointer (in ofstream) to byte number 30 in the file
- It automatically points at the beginning of file, allowing us to read the file from the beginning.
- fin.seekg(30, ios::beg);   // go to byte no. 30 from beginning of file
- fin.seekg(-2, ios::cur);   // back up 2 bytes from the current position
- fin.seekg(0, ios::end);   // go to the end of the file
- fin.seekg(-4, ios::end);   // backup 4 bytes from the end of the file
-

tellg() and tellp()

- The functions tellg() and tellp() return the position, in terms of byte number, of put_pointer and get_pointer respectively, in an output file and input file.

seekg() Function

- The **seekg( )** function moves the associated file's current pointer *offset the number* of characters from the specified *origin*
- The **seekg()** function is member of the *ifstream* class and is called through an *ifstream* object
- The function only moves the pointer ahead, the reading operation should then be performed through some other function

Example

```
// Starting from the beginning, move position pointer five
   characters further

i.seekg(5, ios::beg);
char chR[20];
i.read(chR, 20);
for(int i = 0; i < 20; i++)
{
    cout << chR[i];
}
```

seekp() Function

- The **seekp()** function moves the associated file's current pointer *offset the number* of characters from the specified *origin*

- The **seekp()** function is member of the *ofstream* class and is called through an *ofstream* object

- The function only moves the pointer ahead, the writing operation should then be performed through some other function

Example 1

```
// Starting from the beginning, move position pointer two
   characters forward… then perform writing with put()

char ch = ‘K’;
i.seekp(2, ios::beg);
i.put(ch);// writes K at third position in the file
```

Example 2

```
// Starting from the end, move position pointer three
   characters backwards… then perform writing with put()

char ch = ‘J’;
i.seekp(-3, ios::end);
i.put(ch);// writes J at fourth from last position
```

GooD Luck

for your

Projects