# CSIS/COMP 1117B
# Computer Programming

## Streams and Text File Input/Output

# Streams and Text File Input/Output

- Introduction
- Output stream
- Input stream
- Character input/output
- File name as a data type
- Classes and objects

# Introduction

- A **file** is a collection of data managed by an operating system for ***long term*** storage.
- A **text file** is a file containing characters.
- In C++, a **stream** is a sequence of data; for example, a stream of integers, a stream of characters, etc.
- Streams are used to establish ***connections*** between a C++ program and some external devices, such as a keyboard, a console window; or some external files, either for input, or for output.
- File streams are provided by the library **fstream**.

# Output Stream

- An **output stream** *flows* data out of a program.
- Recall that cout can be used to send output (which are strings of characters) to the console window.
  - non-characters, such as numbers, are converted into a character string (by the insertion operator <<) before being sent to the console window
- A variable of type **ofstream** can be used to establish an output stream between a C++ program and a text file.

# Writing to a Text File

- Declare a variable of type ofstream:

  ofstream  out_stream;
- Connect the variable to a file in the local folder:

  out_stream.open(“outfile.txt”);
- Output data to out_stream as if it were cout:

  out_stream << “Hello!” << endl;
- Disconnect the variable from the file at the end of program execution:

  out_stream.close();

# A Simple Example

```
//  Text file output demonstration program
#include <fstream>
using namespace std;
ofstream  out_stream;
int main() {
    out_stream.open(“outfile.txt”);
    for (char c = ‘a’; c <= ‘z’; c++)
        out_stream << c << endl;
    out_stream.close();
}
```

# Some Remarks

- The output file contains 52 characters: the 26 alphabets each followed by an '\n'
- When out_stream.open("outfile.txt") is executed
  - if outfile.txt cannot be found in the local folder, a new file with the same name (i.e., outfile.txt) will be created
  - otherwise, the contents of the existing file will be erased
- Output formatting can also be specified as in cout:

```
setw(<width for printing next value>)      // in library iomanip
<ofstream>.setf(ios::scientific);          // in library fstream
<ofstream>.setf(ios::fixed);               // affects subsequent
<ofstream>.setf(ios::showpoint);           // output until reset
<ofstream>.precision(<num of digits>);     // to other values
```

# Example with Error Checking Added

```
//  Text file output demonstration program (with error checking)
#include <fstream>       // library containing ofstream
#include <iostream>      // library containing cout
#include <cstdlib>       // library containing exit
using namespace std;
ofstream  out_stream;
int main() {
    out_stream.open("outfile.txt");        // no need to open cout
    if (out_stream.fail()) {
        cout << "Cannot open \"outfile.txt\".\n";
        exit(-1);              // negative value indicates runtime error
    }
    for (char c = 'a'; c <= 'z'; c++)
        out_stream << c << endl;
    out_stream.close();                    // no need to close cout
    if (out_stream.fail()) {
        cout << "Cannot close \"outfile.txt\".\n";
        exit(-2);              // a different value indicates a different error
    }
    exit(0);                   // normal termination
}
```

# Input Stream

- An **input stream** *flows* data into a program.
- Recall that cin can be used to obtain input (which can be chars, ints, doubles, etc.) from the keyboard
  - values appropriate to the variable to receive the data are extracted from the stream of characters input from the keyboard (by the extraction operator >>)
- A variable of type **ifstream** can be used to establish an input stream between a C++ program and a text file.

# Reading from a Text File

- Declare a variable of type ifstream:

  ifstream  in_stream;
- Connect the variable to a file in the local folder:

  in_stream.open(“infile.txt”);
- Input data from in_stream as if it were from cin:

  in_stream >> a_variable;
- Disconnect the variable from the file at the end of program execution:

  in_stream.close();

# Another Simple Example

```
// Text file input demonstration program
#include  <fstream>
#include  <iostream>
#include  <cstdlib>
using namespace std;
ifstream  in_stream;
bool b; char c; double d; int i; string s;
Int main() {
    in_stream.open(“infile.txt”);
    if (in_stream.fail()) {
        cout << “Cannot open \”infile.txt\”.\n”;
        exit(-1);
    }
    in_stream >> c >> s >> i >> d >> b;
    cout << “c(“ << c << “) s(“ << s << “) i(“ << i << “) d(“
            << d << “) b(“ << b << “)\n”;
    in_stream.close();
    exit(0);
}
```

# A Note on The Extractor

- Suppose infile.txt contains the following:

  \n\b\bHKU\b\b\n\b\b23\b\b\b\b12.0625\n1\b\n
- This is the output from the program:

  c(H) s(KU) i(23) d(12.0625) b(1)
- The extractor (>>) skips leading space characters when looking for the data to extract for the next variable and stops immediately when a space character is encountered
  - *how to input a string that contains blanks?*

# End of File

- Unlike an output file which will be extended indefinitely to accept more output, an input file could be exhausted eventually; hence, input operations should check for the availability of data before attempting to process any data that might have been obtained.
- Recall that input extraction also returns a bool value indicating whether or not the operation has been completed successfully, i.e., a value of the expected type has been obtained.
- An unsuccessful operation could also mean that there is no more data to be extracted, i.e., end of file has been reached.

# Input with Error Checking - Example

```
#include  <fstream>
#include  <iostream>
#include  <cstdlib>
ifstream  in_stream;
// need declaration for a_variable
int main() {
    in_stream.open(“somefile.txt”);
    if (in_stream.fail()) {
        cout << “Cannot open \”somefile.txt\”.\n”;
        exit(-1);
    }
    while (in_stream >> a_variable) {
        // computation involving a_variable
    }
    if (in_stream.eof()) exit(0);    // make sure that input error is caused by EOF
    cout << “Format error on \”somefile.txt\”.\n”;
    exit(-2);
}
```

# More on Text File Streams (1)

- At any moment, a file can only be connected to ***one*** stream, either input or output.
- A file must first be disconnected from a stream (<stream>.close) before it can be re-connected.
- To read a file twice:
  - connect (<stream>.open) it to an input file stream; read its data
  - disconnect (<stream>.close) the file from the stream
  - re-connect (<stream>.open) it to an input stream (a new one or the original one); read again (note that input will start from the beginning of the file)

# More on Text File Streams (2)

- To save some data in a *temporary file* and then read the data:
  - connect a temporary file to an output stream; write data to it
  - disconnect the file from the stream
  - re-connect it to an input stream; read the data (note that input will start from the beginning of the file)
- To open a file for appending, i.e., add new data at the end of an existing file, open it with an additional parameter:

  <ofstream>.open(<file name>, ios::app);

# Character Input/Output (1)

- If all data files contain ***validated*** data, the processing logic will be quite evident in the programs (without being burdened with data validation logic) and will be more ***readable***.
- Separate data validation programs will be used to validate the data
  - ensure that data is well-formed
  - numeric data are within the proper range
  - data validation typically examines the input character by character and performs its own data conversion (e.g., from characters to numeric)

# Character Input/Output (2)

- An input stream has a ***marker*** pointing to the first unread character; when a datum is read, the marker advances forward to the character immediately after the datum.
- To read a single character (whether or not printable) into a variable c from an input stream ***without*** skipping any leading space characters:

  <ifstream>.get(c);

- The marker will be advanced by just 1 character position.
- Similarly, to write a single character to an output stream:

  <ofstream>.put(c);  // equivalent to <ofstream> << c;

# Line Skipping - Example

```
// skip_line set marker to the beginning of the next line
// text input stream passed as reference parameter
// returns false on input error
//              true on successful completion
bool skip_line(ifstream &in_stream) {
    char c;
    do {
        in_stream.get(c);
        if (in_stream.fail()) return false;
    } while (c != '\n');
    return true;
}
```

# File Copying - Example

- Requirement: a program to make a copy of a file.
- Input:
  - name of the source file
  - name of the destination file
- Output:
  - data from source file copied to destination file
  - various error messages
    - source file doesn’t exist
    - I/O errors, . . .
- *If destination file already exists, are existing contents to be preserved?*

# The Main Program

```
#include  <fstream>
#include  <iostream>
#include  <cstdlib>
using namespace std;
int main() {
    ifstream   in;
    ofstream  out;
    setup_source(in);
    setup_destination(out);
    copy(in, out));
    in.close();
    out.close();
    exit(0);
}
```

# Copying is Easy

```
// copy from stream in to stream out
void copy(ifstream &in, ofstream &out) {
    char c;
    for ( ; ; ) {  // an infinite loop
        in.get(c);
        if (in.fail()) return;
        out.put(c);
    }
}
```

# Setup Source

```
// setup source input stream
void setup_source(ifstream &in) {
    file_name f;
    cout << “Please input source file name: “;
    if (get_file_name(cin, f)) {
        in.open(f);
        if (in.fail()) {
            cout << “Cannot open \”” << f << “\”.\n”;
            exit(-1);
        } else return;
    }
    cout << “Bad file name \”” << f << “\”.\n”;
    exit(-2);
}
```

# Setup Destination

```
// setup destination output stream
void setup_destination(ofstream &out) {
    file_name f;
    cout << “Please input destination file name: “;
    if (get_file_name(cin, f)) {
        out.open(f);
        if (out.fail()) {
            cout << “Cannot open \”” << f “\”.\n”;
            exit(-3);
        } else return;
    }
    cout << “Bad file name \”” << f << “\”.\n”;
    exit(-4);
}
```

# Obtain File Name from User

```
// obtain a filename from (istream) in
// return false if no input or input too long
bool get_file_name(istream &in, file_name f) {
    char  c;
    for (int i = 0; i < MAX_FILE_NAME; i++) {
        in.get(c);
        if (in.fail()) return false;
        if (c != '\n') f[i] = c;
        else {
            f[i] = '\0';       // C++ strings terminate with a \0 byte
            return i > 0;  // there must be some input in f
        }
    }
    skip_line(in);   // input too long, skip remaining characters in the input
    return false;
}
```

# File Name as a Data Type

- It is possible to create a *new* type for file names by using a **typedef**:

  typedef <data type> <type name> ‘;’

- For example:

```
// define a literal constant MAX_FILE_NAME for length of file names
// file names can be up to 32 characters long, all occurrences of
// MAX_FILE_NAME in the program will appear as 33 to the compiler

#define  MAX_FILE_NAME  33

// create an alias file_name for char array of size MAX_FILE_NAME

typedef  char  file_name[MAX_FILE_NAME]
```

# Classes and Objects (1)

- Streams are **objects**, which are variables that have functions (also called **methods**, or **member functions**) as well as data (also called **members**) associated with them.
- For example, an ifstream object has a member function open to establish a connection between itself and an external file:

  in.open(“text.file”);  // in is an ifstream object

- Member function calls are similar to an ordinary function call except that the (member) function name has to be ***qualified*** by the name of the object which supplies that function.

# Classes and Objects (2)

- A **class** is a type that can be used to declare variables which are objects.
- An object's class determines what member functions the object has as well as what (data) members it has.
- A class declaration is of the form:

```
class <name> {
    // (data and function) member definitions
}
```

- ifstream is a class and in is an ifstream object:

```
ifstream in;        // compare with int k;
```