Writing a Good Program

7. Stream I/O
**Input and Output**

- C++ does not, as a part of the language, define how data are sent out and read into the program.
- The *input and output (I/O)* are handled by the standard C++ library (such as `iostream`).
  - A *library* is a collection of *object code* to be linked to the C++ program to provide additional functionality.
  - For different platforms, different libraries (*that may use the same name*) can be used for I/O functions.
    - This allows the C++ program to be *less platform dependent*.
    - It is because I/O is usually quite different from computers to computers.
Buffering

- **iostream** classes view the flow of data as being a stream of data, one **byte** following another

- **Buffer** is provided in some cases to the stream:
  - The stream of data does not send out if the **buffer** is not full or no flush request is received
  - It is useful in some situations such as written data to **disk** since the **overhead** is very large. It is better to do it in **single lots** rather than byte by byte.
```cpp
char xyz[10];
cin >> xyz;
```

Program

Keyboard

Buffer:
Memory in your keyboard

Chan

```
xyz = "Chan"
```
7. Stream I/O

Standard I/O Objects

- When a C++ program that includes the `iostream` classes starts, **four objects** are created and initialized
  
  - **cin** - handle input **from** the standard input, i.e. keyboard
  - **cout** - handle output **to** the standard output, i.e. display
  - **cerr** - handle **unbuffered** output to the standard error device, i.e. display in a PC
  - **clog** - handle **buffered** error messages that are output to the standard error device, i.e. display in a PC.

Buffered: Display only when a flush command is received, e.g. end of statement
Input using `cin`

- `cin` has been generally used for input data from keyboard
  ```cpp
  int someVariable;
  cin >> someVariable;
  ```

- It is a **global** object – does NOT need to define in our own code

- The operator `>>` is **overloaded** such that different kind of data can be read into the buffer of `cin` and then to some variable.
  ```cpp
  float someVariable1;
  cin >> someVariable1;
  ```
  ```cpp
  double someVariable2;
  cin >> someVariable2;
  ```
  ```cpp
  char someVariable3[100];
  cin >> someVariable3;
  ```
Member functions, e.g. `cin.get()`

- As `cin` is an object, it has its own member functions
  - Helps to obtain the input data in a more **flexible** way.

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

int main() {
    int c;
    while ((c = cin.get()) != EOF) {
        cout << "c: " << (char)c << endl;
    }
    cout << "\nDone!\n";
    return 0;
}
```

- `cin.get()` returns an **integer** from standard input
- The loop will stop if end-of-file (crtl-z) is input
- EOF: **0xFF** is a symbolic `constant`
This is the "Enter" key (newline character)

If we press `crtl-z`, it becomes End-of-file (EOF) condition.
Member functions, e.g. `cin.getline()`

- `cin.getline()` allows the whole line of data to be input to a character array.

```cpp
cin.getline(buffer, MAXSIZE);
```

- The maximum length of data to be read is defined by `MAXSIZE`.
- Unlike `cin`, `getline()` will read in the terminating `newline` character and throw it away.
- Using `cin`, the terminating `newline` is not thrown away, but left in the input buffer.

Include NULL, i.e. `\0` at the end. At most `MAXSIZE-1` real characters will be stored.

`cin` will neglect, though, any preceding space, tab and newline in the buffer.
Why do we use `cin.getline()`?

- `cin >> abc` has been used quite often since the redirection operator `>>` is overloaded, i.e.
  - such statement can be used no matter `abc` is a character array, an integer or a floating-point number
- However if `abc` is a character array, `cin >> abc` allows only string with continuous characters input
  - `cin >> abc` will stop to read in data if it sees, e.g. a space
- `cin.getline()` is different in that `getline()` can only read in string, but not value of other types; however, it will not stop when encountering spaces or tabs.
#include <iostream>
using namespace std;

int main()
{
  char stringOne[256];
  char stringTwo[256];
  char stringThree[256];
  cout << "Enter string one: ";
  cin.getline(stringOne,256);
  cout << "stringOne: " << stringOne << endl;

  cout << "Enter string two: ";
  cin >> stringTwo;
  cout << "stringTwo: "
       << stringTwo << endl;

  cout << "Enter string three: ";
  cin.getline(stringThree,256);
  cout << "stringThree: " << stringThree << endl;

  return 0;
}
7. Stream I/O

```cpp
char abc[10], xyz[10];
cin >> abc; cout << "abc: " << abc << endl;
cin.getline(xyz, 10);
cout << "xyz: " << xyz << endl;
```

Program

```
abc = "Chan"
xyz = '\0'
```

Keyboard

Buffer:

Memory in your keyboard

The "Enter" code is still in the buffer
7. Stream I/O

```cpp
char abc[10], xyz[10];
cin.getline(abc, 10); cout<<"abc: " <<abc<<endl;
cin.getline(xyz, 10); cout << "xyz: " << xyz << endl;
```

Program:

```
abc = "Chan"
```

Keyboard:

Chan

Buffer:
Memory in your keyboard

The "Enter" key has been cleared
Output with `cout`

- Just as `cin`, `cout` is also an object created when the `iostream` class starts.
  - `cout` represents the standard output, i.e. display.
  - The operator `<<` is overloaded such that different kinds of data can be sent out.

- Similar to `cin`, `cout` also has a number of member functions:
  - `cout.put(char a)`
    - Put the value of character `a` to output, and return `cout`.
  - `cout.write(char *buf, int length)`
    - Write `length` characters in `buf` to output device, and return `cout`.

You may write `cout.put(a)<<endl;`.
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    char One[] = "One if by land";
    int fullLength = strlen(One);
    int tooShort = fullLength - 3;
    int tooLong = fullLength + 6;

    cout.write(One,fullLength) << "\n";
    // fullLength=14, return cout after write
    cout.write(One,tooShort) << "\n";
    cout.write(One,tooLong) << "\n";
    return 0;
}
```

write() asks the system to write tooLong characters. Hence something unexpected is displayed.
Other Member Functions

`cout.width(int a)`
- Set the width of the next output field to a's value

`cout.fill(char b)`
- Fill the empty field of the output by b's value

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Start >";
    cout.width(6);
    cout << 123 << "<End\n";

    cout << "Start >";
    cout.width(6);
    cout.fill('*');
    cout << 123 << "<End\n";
    return 0;
}
```

`width(6)` defines the total width of the field to be 6 char.
`fill('*')` fills the empty field with *
Exercise 7.1

Write a program that asks the user to

1. Input his/her surname
2. Input his/her first name

If the user puts a space between the two words of his first name, add a hyphen to replace the space. Skip all other spaces in front of or after the first name or surname.

For example: Chan

Tai Ming ⇒ Chan Tai-Ming

Show the whole name after you read the user inputs.

Hint: A string is stored in a character array. You can use a loop to check the content of every character of a string one by one, and to display the characters out one by one.
File Input and Output

- C++ provides library functions to help us deal with file access

- File access is usually performed in the following:
  - To claim from the system the access of a particular file
  - To perform the read or write instructions
  - To indicate to the system the file access is finished (closed)
    It is not guaranteed that data can be written to file if not closing.
Streams provide a uniform way of dealing with data sending to or reading from files.

The objects that are used to deal with files are called ofstream and ifstream objects.

To create an ofstream or ifstream object, we need to include the fstream header file in our program.

To open a file for write or read, first create an ofstream or ifstream object with the filename as the input parameter.

ofstream fout("myfile.cpp");
ifstream fin("myfile.cpp");

fout and fin are objects but NOT standard objects, unlike cout and cin. Any other names can be used.
Read or Write Files

- After opening files, reading or writing files can be done in a similar way as `cout` or `cin`.

- After writing/reading, the file should be closed by the member function `close()` of the `ofstream/ifstream` objects.

```cpp
#include <iostream>
#include <fstream>

int main() {
    ofstream fout("myfile.cpp");
    fout << "This line is written to file.\n";
    fout.close();
    ifstream fin("myfile.cpp");
    char ch;
    while (fin.get(ch)) //NULL if end of file
        cout << ch;
    fin.close();
    return 0;
}
```

To see the output, look at the content of the file `myfile.cpp`.

If for any reason `myfile.cpp` cannot be opened, `fout` & `fin` will still be created, but `fout.fail()` return `true`.

Similar to `cout` and `cin.get(ch)`.

```cpp
(ch=fin.get())!=EOF
```
Changing Default Behavior

- **Default behavior** of opening a file by `ofstream` is to **create** it if it doesn’t yet exist.
  - If exists, it **deletes all its contents** and overwrite on it (truncate).
  - This default behavior can be changed by providing a second parameter to the constructor.

  ```
  ofstream fout2("myfile.cpp", ios::app);
  ```

- **Open modes** of class `ios`:
  - `ios::app` – File will be opened for appending data to the end of the existing file.
  - `ios::ate` – Place you at the end of the file (either input or output), but write data at the start position, like `trunc`.
  - `ios::trunc` – Default for `ofstream`; truncates if it exists.
  - `ios::in` – The file (if it exists) will **NOT** be truncated, the read/write cursor is positioned at the start of the file.

  `ios::ate|ios::in` is similar to `ios::app`.
#include <fstream>
#include <iostream>
using namespace std;

int main() {
    ofstream f1("myfile.cpp"); //Default trunc
    f1 << "This line is written to file.\n";
    f1.close();
    char fname[]="myfile.cpp";
    ofstream fout2(fname,ios::app);
    fout2 << "This line is appended to the end of the previous line\n";
    fout2.close();
    ifstream fin("myfile.cpp");
    char ch;
    while (fin.get(ch))
        cout << ch;
    fin.close();
    return 0;}

File name can be supplied by a variable

The default behavior is changed

To avoid redeclaring variable, a different name should be provided even for opening the same file
7. Stream I/O

```cpp
//Assume myfile.cpp contains "This line is written to file.\n"
ofstream fout2("myfile.cpp", ios::in);
fout2 << "Beginning of file";
fout2.close();
```

Default behavior is changed to non-truncate. The line will replace the beginning part of the text in the file; but the rest words, if any, will be left in the file.

```cpp
ifstream fin("myfile.cpp");
if (!fin) {cout << "File opening error!\n"; return 0;}
//The operator ! is defined for stream object, if the stream is bad, return NULL (false)
char ch;
while (fin.get(ch))
    cout << ch;
fin.close();
return 0;
```

It is a good practice to check if anything goes wrong when opening a file.
Editing File

- It is often that we need to modify a particular part of a file - editing a file
  - ofstream and ifstream offers 4 member functions that set the current position of an opened file

For ofstream

```cpp
seekp(long pos);
tellp();
```

For ifstream

```cpp
seekg(long pos);
tellg();
```

The first position is position 0.

- `seekp()` sets the current position. `pos` is a byte offset from the file beginning
- `tellp()` tells the current position. Return a byte offset from the file beginning
- `seekg()` and `tellg()` are similar to `seekp` and `tellp` except they are for ifstream
```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main()
{
    ofstream test1file("test1");
    cout << "Text written: \n";
    cout << "This text is written to file!\n"; // write to screen
    test1file << "This text is written to file!\n";
    test1file.close();
    ifstream tin("test1");
    tin.seekg(10);
    cout << "Current position: " << tin.tellg() << "\n";
    cout << "Content of the file after an offset of 10: \n";
    char ch;
    while (tin.get(ch))
    {
        cout << ch;
    } // display file
    tin.close();
    return 0;
}
```
Exercise 7.2

1. Build the program in the last page
2. Run the program in Command Prompt
3. Use the DOS command "dir" to find the file "test1" and display it by using the DOS command "type test1". Can you find the statement as shown in the program?
4. Use `seekp` such that the word "written" in the file is replaced by "*******".
5. Verify the result by using the DOS commands "dir" and "type test1" again.
Slides

- The powerpoint slides are provided by Dr Frank Leung, EIE.