Slot 02 - Binary File

Presenter:

Dr. LE Thanh Tung

Content

1. String
2. Text File
3. Binary File

Exercise 1: Count the number of words in a text string S. Known that words are separated by a space

Exercise 2: A student information includes: student ID, name, final score. Do the following requirements:

- Create the structure for one student
- Design a function to find the student who has the highest score in the list of students

File Introduction

- A file itself is a bunch of bytes stored on a computer's disk
- The process of saving data in a file is known as **writing data** to the file
- The process of retrieving data from a file is known as **reading data** from the file
- C++ file access requires the header file `fstream`
- Type of file:
  - Text format: means storing everything as text, even numbers
  - Binary format: means storing the computer's internal representa1on of a value

Binary Vs Text

- With characters, binary format is the same as text format
  - Content is represented by the binary format of the character's ASCII code (or equivalent)
  - i.e: ABC → 01000001 01000010 01000011
- With numbers, the binary format is much different from the text format
  - 0.375 in text format:

    00110000 00101110 00110011 00110111 00110101
  - 0.375 in binary format:

    0 0111110 11000000000000000000000000

    sign exponent significand/mantissa

Binary Vs Text

	Binary File	Text File
Secure	Less	More
Readable	By normal user	Can't without the knowledge about the structure
Size	Based on the number of characters	Based on the nature of data

- There are three classes included in the `fstream` library, which are used to create, write or read files:

Class	Description
`ofstream`	Creates and writes to files
`ifstream`	Reads from files
`fstream`	A combination of ofstream and ifstream: creates, reads, and writes to files

Text File

- Open a text file:

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

Text File

- Open a text file:

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

- An open file is represented within a program by a stream
- In order to open a file with a stream object, we use its member function

```
open (filename, mode);
```

```
open (filename, mode);
```

Where:

- filename is a string representing the name of the file
- mode is an optional parameter with a combination of the following flags

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

- All these flags can be combined using the bitwise operator OR "|"

```
fs.open("abc.txt", ios::out | ios::in);
fs.open("abc.txt", ios::out | ios::app);
fs.open("abc.bin", ios::out | ios::binary);
fs.open("abc.bin", ios::in | ios::binary);
```

- The default mode of file stream:

Class	Default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::out | ios::in

- Reading from a file can also be performed in the same way that we did with `cin`
- Writing operations on text files are performed in the same way we operated with `cout`
- The following member functions exist to check for specific states of a stream

`bad()`	Returns `true` if a reading or writing operation fails
`fail()`	Returns `true` in the same cases as `bad()`, but also in the case that a format error happens
`eof()`	Returns `true` if a file open for reading has reached the end

- **For input stream:**

```
ifstream ifs;
ifs.open("abc.txt");
```
⇔
```
ifstream ifs("abc.txt");
```

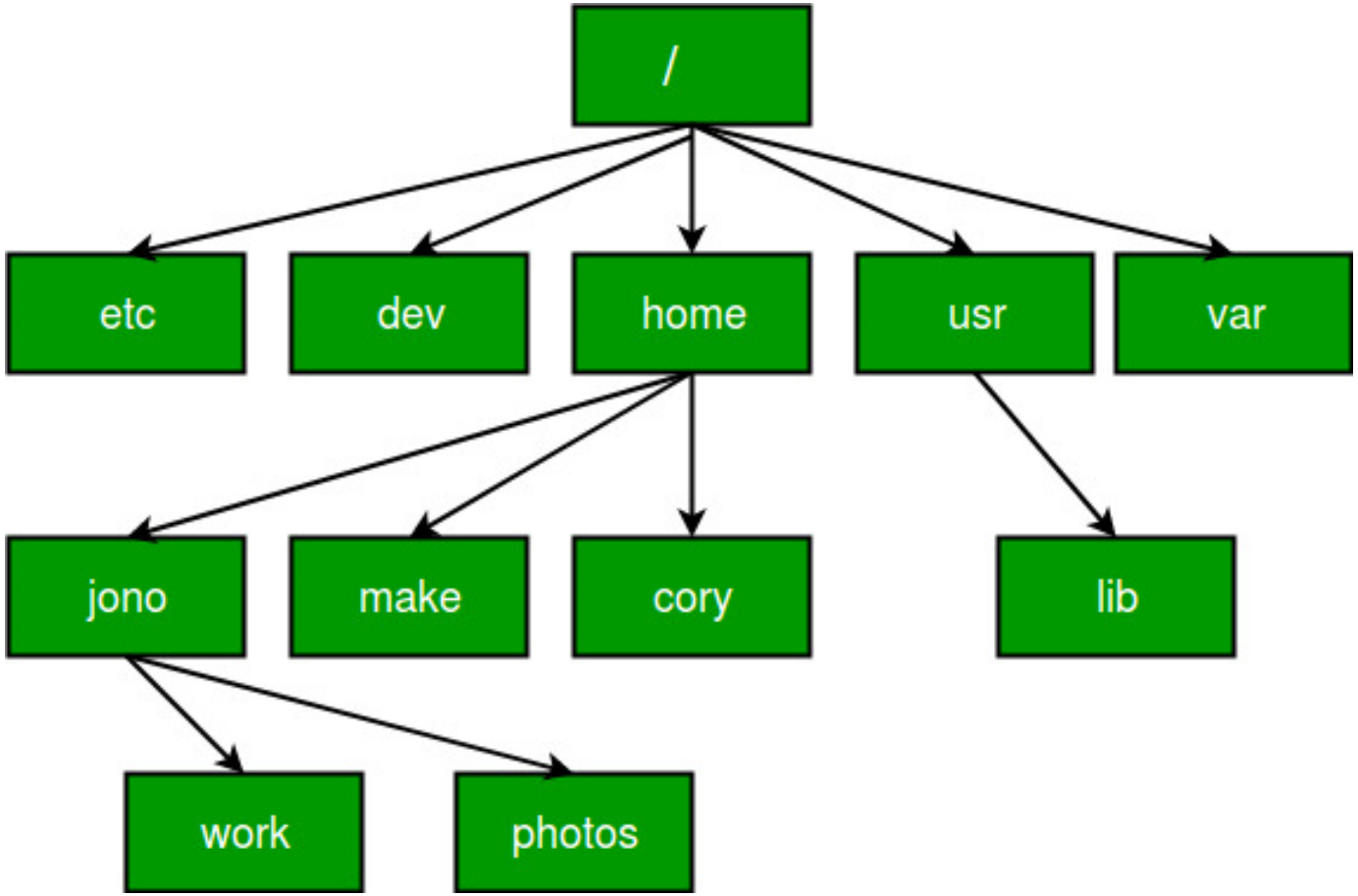
- **For output stream:**

```
ofstream ofs;
ofs.open("abc.txt");
```
⇔
```
ofstream ofs("abc.txt");
```

- **As using the default mode in ofs:**
  - If the specified file already exists, it will be erased, and a new file with the same name will be created

Absolute and Relative Path

Read more: https://www.geeksforgeeks.org/absolute-relative-pathnames-unix/



Open – Close File

- In this course, you must check the condition of file

```
ifs.open("abc.txt");
if (!ifs.fail()) { … }
else
  cout << "Error opening file";
```

```
ifs.open("abc.txt");
if (ifs) { … }
else
  cout << "Error opening file";
```

- Close the file if you do not use it again

```
<fso>.close();
```

<EOF> Character

- EOF> is a character that is automatically written to the end of a file when it is closed

- The actual character used to mark the end of a file depends upon the OS being used
  - Some systems use ^Z

Read position

- When a file has been opened for input, the file stream object internally maintains a special value known as a **read position**
  - the location of the next byte that will be read from the file
- Initially, the read position is set to the first byte in the file
- After each read operation, the read position moves forward, toward the end of the file

- The function reads a line from a file to a `string` object
- It reads a whole line of data, including whitespace characters

```
getline(fso, s);  // string s
```

- **Exercise 3**: Write a program to read the list of students in a text file. After that, display the name of the student who has the highest score.
- Download file from: https://tinyurl.com/ktltslot02

Binary File

- Open the binary file

```
ifstream myFile ("data.bin", ios::in | ios::binary);
```

- **Note**: If you are a GNU g++ user (version 2.7.x or earlier), then do not use i/o mode flags when opening ifstream objects

- Open the binary file

```
ifstream myFile ("data.bin", ios::in | ios::binary);
```

- **Note**: If you are a GNU g++ user (version 2.7.x or earlier), then do not use i/o mode flags when opening ifstream objects

- Instead of using it, fstream is better:

```
fstream myFile;
myFile.open ("data3.bin", ios::in | ios::out | ios::binary);
```

Binary File: Write

- The file stream object's `write` member function is used to write data to a binary file
- The general format of the `write` member function is

```
fs.write(address, size);
```

  - `fs` is the name of a file stream object
  - `address` is the starting address of the block of memory that is to be written to the file. It **must be** the address of a `char`
  - `size` is the number of bytes of memory to write

Binary File: Write - Example

```
fstream fs("abc.txt", ios::out | ios::binary);
char ch = 'A';
fs.write(&ch, sizeof(char));

char st[SIZE];
cin.getline(st, SIZE);
fs.write(st, sizeof(st));
```

Binary File: Read

- The file stream object's `read` member function is used to read data from a binary file into memory
- The general format of the `read` member function is

```
fs.read(address, size);
```

  - `address` is the starting address of the block of memory where the data being read from the file is to be stored. It **must be** the address of a `char`
  - `size` is the number of bytes to read from the file

Binary File: Read - Example

```
fstream fs("abc.txt", ios::in | ios::binary);
char ch;
fs.read(&ch, sizeof(char));
```

```
char st[SIZE];
fs.read(st, sizeof(st));
```

Data Other Than char

- Must use a **type cast** when writing or reading items that are of other data types than char
- To convert a pointer from one type to another you may use
  - C++: `reinterpret_cast<dataType>(val)`
  - C: `(dataType) val`

- Example 1:

```
int n = 100;

fs.write(reinterpret_cast<char *> (&n), sizeof(n));
// fs.write((char *) &n, sizeof(n));
```

- Example 2:

```
const int SIZE = 5;
int a[SIZE] = {0, 1, 2, 3, 4};

fs.write(reinterpret_cast<char *> (a), sizeof(a));
```

Exercise

- Read the following binary file:

https://drive.google.com/file/d/1HVFoGkpzdWUFYOLi92Z0p2RvYQVcg_OG/view?usp=sharing

- The number series in the binary file is a secret message.

Exercise

- Write a program to read the list of students in a text file. After that, write a function to store them into a binary file.
- Download file from: https://tinyurl.com/ktltslot02

Pointer in File

- Pointers cannot be correctly stored to disk because we store the value of variable while value of pointer is “temporary” address instead of data
- String class objects contain implicit pointers
- In these cases, must convert it into data before storing it

Random File Access

- All of the programs created so far have performed **sequential file access**
- What is the problem with sequential file access:
  - In order to read a specific byte from the file, all the bytes that precede it must be read first → slow a program down tremendously
- C++ allows a program to perform **random file access**
  - immediately jump to any byte in the file without first reading the preceding bytes

Random File Access

- These two member functions are used to move the **read/write position** to any byte in the file
  - For **output:** fs.seekp(offset, mode);
  - For **input**: fs.seekg(offset, mode);
    - offset represents an offset into the file
    - mode designates where to calculate the offset from

Random File Access

- If a program has read to the end of a file, a call to the file stream object's `clear` member function is needed before calling `seekg` or `seekp`
  - Otherwise, the `seekg` or `seekp` function will not work

Current Position

- There are two functions to return the current byte number of a file's read/write position
  - `tellp` returns the write position
  - `tellg` returns the read position
- One application of these functions is to determine the size of a file

```
fs.seekg(0L, ios::end);
long numBytes = fs.tellg();
cout << "The file has" << numBytes << "bytes";
```

THANK YOU
for YOUR ATTENTION