Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 11 Working with Files

- **File（文件）**
  - Files in C++ are interpreted as **a sequence of bytes** stored on some storage media.
  - Each file ends with an **end-of-file (EOF)** marker.
- **Using Input/Output Files（文件输入输出）**：
  - File $\longrightarrow$ Program (Input stream) - **reads**
  - Program $\longrightarrow$ File (Output stream) – **write**
  - File streams act as an interface between files and programs, **stream classes for file operations** are declared in the header file `fstream`

$$\begin{tikzcd}
\text{iostream} \arrow[r] & \text{fstream} & \text{iostream} \arrow[l] \\
\text{istream} \arrow[u] \arrow[d] & \text{ios} \arrow[l] \arrow[r] \arrow[d] & \text{ostream} \arrow[u] \arrow[d] \\
\text{ifstream} & \text{streambuffer} \arrow[d] & \text{ofstream} \\
& \text{filebuf} &
\end{tikzcd}$$

  cont.

$$\begin{tikzcd}
& \text{ios} \arrow[d] & \\
\text{ifstream} & \text{fstreambase} \arrow[l] \arrow[r] \arrow[d] & \text{ofstream} \\
& \text{fstream} &
\end{tikzcd}$$

- **The Process of Using a File**
  - Three-step process:
    - The file must be **opened**. If the file does not yet exists, opening it means **creating** it.
    - Information is then **saved** to the file, **read** from the file, or **both**.
    - When the program is **finished using the file**, the file must be **closed**.
- **How to open a file in C++**:
  - The following classes can be used:
    - `ofstream` – writing to a file
    - `ifstream` – reading from a file
    - `fstream` – reading or writing
  - A file can be opened in two ways:
    - Using the **constructor function** of the file stream class:

```
ofstream out_file("client.dat", ios::out);
```

        - Using the **member function** `open()` of the file stream class:

          ```
          ofstream out_file;
          out_file.open("client.dat", ios::out);
          ```

- **File Open Modes（文件打开模式）**

Mode	Description
`ios::in`	Open a file for **input**
`ios::out`	Open a file for **output**, **discard** the contents **if it exists**
`ios::trunc`	**Discard** the contents **if it exists**
`ios::app`	Write all output to the end of file **(append)**
`ios::out|ios::app`	Open a file for **output**，**keep the contents** and write to the end of the file
`ios::ate`	**File pointer** is positioned **at the end of the file**
`ios::out|ios::ate`	Open a file for **output**, discard the contents if it exists
`ios::in|ios::ate`	Open a file for **input**, **file pointer** is positioned **at the end of the file**
`ios::binary`	Read/write data in **binary format**
`ios::nocreate`	If the file does **NOT exists**, the open operation **fails**
`ios::noreplace`	If the file **exists**, the open operation **fails**

File Type	Default Open Mode
`ofstream`	The file is opened for **output only**. If the file does **not exist**, it is **created**. If the file **already exists**, its contents are **deleted**.
`ifstream`	The file is opened for **input only**. The file's contents will be **read from its beginning**. If the file does **not exist**, the open function **fails**.

- **Testing for Open Errors（文件打开错误检测）**：

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

  using namespace std;

  int main(void) {
    fstream data_file("names.dat", ios::in | ios::out);
    // Stream object returns a value of 0 if any error occurs in the file operations
    if (!data_file) {
      cout << "Error opening file.\n";
    }

    // Another way to test for open errors
    if (data_file.fail()) {
      cout << "Error opening file.\n";
    }
    return 0;
  }
  ```

- **Closing a File（关闭文件）**
  - A file should be closed **when a program is finished using it**:

    ```
    stream_object.close();
    ```

- **Using `<<` to Write Information to a File**
  - The stream **insertion operator（插入运算符）** `<<` can be used to write information to a file.
  - File output can be **formatted the same way as screen output**.
- **Using `>>` to Read Information from a File**
  - The stream **extraction operator（提取运算符）** `>>` may be used to read information from a file.
- **Detecting the End of a File（检测文件结束）**
  - The `eof()` **member function** reports when the end of a file has been encountered:

    ```
    if (in_file.eof())
      in_file.close();
    ```

  - `while(in_file.eof())` is Equivalent to `while(in_file)`
- **More Detailed Error Testing（更多关于错误检测的细节）**：
  - A file which we are attempt to open for reading does not exist. **（文件不存在）**
  - We may attempt an invalide operation such as reading past the end-of-file. **（执行无效操作，如读取EOF后的内容）**
  - There may not be any space in the disk for storing more data. **（磁盘空间不足）**
  - We may use an invalid file name. **（使用非法文件名）**
  - We may attempt to perform an operation when the file is not opened for that purpose **（尝试执行与文件打开模式不对应的操作）**
- **Error State Bits（错误状态位）**
  - All stream objects have error state bits that **indicate the condition of the stream**:

Bit	Description
`ios::eofbit`	Set when the **end** of an **input stream** is encountered
`ios::failbit`	Set when an attempted **operation has failed**
`ios::hardfail`	Set when an **unrecoverable error** has occurred
`ios::badbit`	Set when an **invalid operation** has been attempted
`ios::goodbit`	Set when **all the flags above are not set**. Indicates the stream is in good condition

  - The class `ios` supports several member functions to read the status recorded in a file stream:

Function	Description
`eof()`	Returns **true (non-zero)** if the `eofbit` flag is set, otherwise returns **false**
`fail()`	Returns **true (non-zero)** if the `failbit` or hardfail flags are set, otherwise returns **false**
`bad()`	Returns **true (non-zero)** if the `badbit` flag is set, otherwise returns **false**
`good()`	Returns **true (non-zero)** if the `goodbit` flag is set, otherwise returns **false**
`clear()`	When called with no arguments, **clears all the flags** listed above. Can also be **called with a specific flag** as an **argument**

- **Member Functions for Reading and Writing Files（读写文件的成员函数）**
  - Read and write to a file in **text form（文本形式）**：

    ```
    getline()
    get()
    put()
    ```

  - Read and write to a file in **binary form（二进制形式）**：

    ```
    write()
    read()
    ```

- The `getline` Member Function:

  ```
  getline(str, 81, '\n');
  ```

  - `str`: This is the **name of a character array（数组名）**, or a **pointer to a section of memory（指向内存节的指针）**. The information read from the file will be stored here.
  - `81`: This number is **one greater than the maximum number（最后有一个`'\0'`，所以需要比输入的最大80个字符多1个字符）** of characters to be read. In this example, a maximum of 80 characters will be read.
  - `'\n'`: This is a **delimiter（分隔符）** character of your choice. **If this delimiter is encountered, it will cause the function to stop reading（遇到就停止读取了）** before it has read the maximum number of characters. (This argument is **optional（可选参数）**. If it's left, `'\n'` is the default.)
- The `put` and `get` Member Functions:
  - `get()` is used to **read an alphanumeric character（读取一个字母数字字符？似乎也不太准确，可以读到非操作字符）** from a file.
  - `put()` is used to write a character to a specified file or a specified output stream

* **Binary Files（二进制文件）**:
  - Binary files contain data that is **unformatted（未格式化的）**, and not stored as ASCII text.
  - Example:
    - Text file of `1297` (expressed in ASCII):

49	50	57	55	<EOF>

    - Binary file of `1297` (An integer):

00000101	00010001

  - To open a binary file, use:

    ```
    file.open("stuff.dat", ios::out|ios::binary);
    ```

    - Commonly use `.dat` as binary file extension name.
  - Binary **input** function:

    ```
    read(unsigned char* buffer, int n);
    ```

    - The `<istream>` function `read` **inputs a specified number of bytes** from the current position of the specified stream into an object.
    - Usage:

      ```
      inflie.read((char*) & V, sizeof(V));
      ```

  - Binary **output** function:

    ```
    write(const unsigned char* buffer, int n);
    ```

    - The `<ostream>` member function `write` **outputs a fixed number of bytes** beginning at a specific location in memory to the specific stream.
    - Usage:

      ```
      outfile.write((char*) & V, sizeof (V));
      ```

* **Reading and writing a class object（类对象读写）**
  - **Structures（结构体）** may be used to store **fixed-length** records to a file:

```
struct info {
  char name[51];
  int age;
  char address[51];
  char phone[51];
};
```

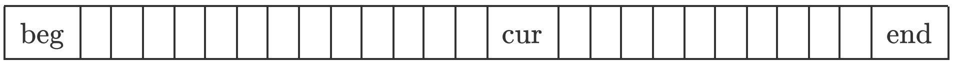
  - Since structures can **contain a mixture of data types（包含混合的数据类型）**, you should **always use the** `ios::binary` **mode（总是需要以二进制模式读写）** when opening a file to store objects.

- **Random Access Files（随机文件访问）**
  - It means non-sequentially accessing informaiton in a file
  - Every file maintains **two internal pointers（两个内部指针）** known as **file pointers**:
    - **get_pointer**
    - **put_pointer**
  - They enable to **attain the random access（实现随机访问）** in file otherwise which is **sequential in nature（本质上是顺序访问）**.
  - In C++ randomness is achieved by manipulating certain functions.
  - **Default Actions（默认行为）**:
    - When we open a file in **read-only mode（只读）**, the **input pointer(get_pointer)** is automatically set at the **beginning（输入指针设置在文件开头位置）**.
    - When we open a file in **write-only mode（只写）**, the **output pointer(put_pointer)** is automatically set at the **beginning** and **the contents are deleted（原有内容被删除，输出指针也设置在文件开头位置）**
    - When we open a file in **'append' mode(** `ios::app` **)**, the **output pointer(put_pointer)** is moved to the **end** of the file
  - **Moving within the File（在文件内移动）**:
    - `seekg()` / `seekp()` – moving the **get** / **put pointer** to specified situation

      ```
      seekg(offset, refposition);
      seekp(offset, refposition);
      ```

      - Need two parameters: **offset（偏移量）（长整型参数）** and **refposition（参照位置）**
        - Parameter **offset** represents **the number of bytes** the file pointer to be moved from the location specified by the parameter **refposition**.
    - `tellg()` / `tellp()` – getting the position of the **get** / **put pointer**
      - `tellp` returns a **long integer** that is the **current byte number** of the file's **write** position.
      - `tellg` returns a **long integer** that is the **current byte number** of the file's **read** position.
    -

Mode Flag	Description
`ios::beg`	The offset is calculated from the **beginning** of the file.
`ios::end`	The offset is calculated from the **end** of the file.
`ios::cur`	The offset is calculated from the **current** position.



```
fstream file("d:\\file1", ios::in);
// get_pointer move 10 bytes backward from the current position
// Move to Left
file.seekg(-10L, ios::cur);
// get_pointer move 10 bytes forward from the beginning of the stream
// Move to Right
file.seekg(10L, ios::beg); // = file.seekg(10L);
```

    - Example:

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

using namespace std;

int main(void) {
  fstream file("letters.txt", ios::out | ios::in);

  if (!file) {
    cout << "the file is not opened" << endl;
    abort();
  }

  file << "abcdefghijklmnopqrstuvwxyz";

  char ch;

  file.seekg(5L, ios::beg);
  file.get(ch);
  cout << "Byte 5 from beginning: " << ch << endl;

  file.seekg(-10L, ios::end);
  file.get(ch);
  cout << "Byte 10 from end: " << ch << endl;

  file.seekg(3L, ios::cur);
  file.get(ch);
  cout << "Byte 3 from current: " << ch << endl;

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

Output:

```
Byte 5 from beginning: f
Byte 10 from end: q
Byte 3 from current: u
```