Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 10 Console I/O

- C++ provides an alternative with the new **stream** input/output features for two reasons:
  - I/O methods in C++ **support the concept of oop（支持面向对象）**
  - I/O methods in C **cannot handle the user-defined data types（无法处理用户自定数据类型）**
- **C++ Streams（C++流）**
  - A transfer of information in the form of **a sequence of bytes（以字节序列传输数据）**
  - C++ Stream Classes:

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

  - **Pre-defined Streams（预定义流）**：

Object Name	Class	Device
`cin`	`istream_withassign`	standard input device(**keyboard**, can be redirected)
`cout`	`ostream_withassign`	standard output device(**screen**, can be redirected)
`cerr`	`ostream_withassign`	standard error output device(**screen**, **can not** be redirected)
`clog`	`ostream_withassign`	standard error output device(**screen**, **can not** be redirected)

    - `cin` 是 `istream` 的**派生类** `istream_withassign` 的对象
    - `cout` 是 `ostream` 的**派生类** `ostream_withassign` 的对象
  - Using the **header file** `iostream`:
    - Include `<iostream>` instead of `<stdio.h>`
    - Standard iostream objects:
      - `cout` - object providing a connection to the **monitor**
      - `cin` - object providing a connection to the **keyboard**
      - `cerr` - object providing a connection to **error stream**
  - The **Insertion** Operator (`<<`):
    - To send output to the **screen**
    - Format: `cout << Expression;`
    - The compiler figures out the type of the object and prints it out appropriately
  - The **Extraction** Operator (`>>`):
    - To get input from the **keyboard**
    - Format: `cin >> Variable;`
    - The compiler figures out the type of the variable and reads in the appropriate type
    - `cin` **ignores whitespaces（无视空白字符）**（spaces, tabs, newlines)
    - Returns **zero (** `false` **) when** `EOF` **is encountered（如果遇到** `EOF` **则返回** `false` **）**，otherwise **returns reference to the object from which it was invoked（否则返回自身的引用，实现连续调用）**（i.e. `cin`)

- `get` and `getline` Member Functions
    - `cin.get(array, size, delimiter)`
        - Accepts 3 arguments: array of characters, the size limit (**character count（字符数）**), and a **delimiter（分隔符）** (default of `'\n'`)
        - Input a sequence of characters from stream till the delimiter or `EOF` is encountered or `size-1` characters are read.
        - Uses the array as a **buffer（缓冲区）**
        - When the delimiter is encountered, it remains in the input stream, unless delimiter flushed from stream, it will stay there **（遇到分隔符时，分隔符会被保留在输入流中）**

          ```
          #include <iostream>

          using namespace std;

          int main() {
            char *str1 = new char[100];
            char *str2 = new char[100];
            cin.get(str1, 10, ' ');
            cout << str1 << endl;
            // cin.get();
            cin.get(str2, 10, ' ');
            cout << str2 << endl;
            return 0;
          }
          ```

          Input & Output:

          ```
          h3art hello // input
          h3art       // output
                      // output
          ```

        - **Null character (** `'\0'` **)** is inserted into the array **at the end of the characters**
    - `cin.getline(array, size)`
        - Operates like `cin.get(buffer, size)` but it **discards the delimiter from the stream（读取一行直到碰到分隔符或行结束，但此时会将分隔符丢弃，不再保留在输入流中）** and does not store it in array
        - Null character inserted into array
- `put` and `write` Member Functions
    - `cout.put(char)`
        - Outputs one character to specified stream
        - **Returns a reference to the object** that called it, so may be **cascaded（连续调用）**:

          ```
          cout.put('A').put('\n');
          ```

        - May be called with an **ASCII-valued** expression:

          ```
          cout.put(65); // = cout.put('A');
          ```

    - `cout.write(line, size)`
        - Outputs the entire line till size characters are displayed
        - the functions **will not terminate** at a **newline** character
        - the functions **will not terminate** at a **null** character **（输出流里存在** `'\n'` **或** `'\0'` **都不会终止其输出）**

- **Formatted Console I/O Operations（格式化控制台输出输出操作）**
    - `ios` class constains a large number of **member functions** to format the output:

Function	Description
`width()`	Specify the required field size

Function	Description
`precision()`	Specify the number of digits to be displayed after the decimal point
`fill()`	Specify a character that is used to fill the unused portion of field
`setf()`	Specify format flags
`unsetf()`	Clear the flags specified

- Setting the **Width**:
  - Use the `width(int)` function to set the width for printing a value, but it **only works for the next output command （只对下一个输出有效，默认为右对齐）**
  - Example:

    ```
    int x = 42;
    cout.width(5);
    cout << x << '\n';
    cout << x << '\n';
    ```

    Output:

    ```
       42
    42
    ```

- Setting **Precision**:
  - By default, the floating numbers are printed with **six** digits **（默认精度为6）**
  - Use the `precision(int)` function to specify the number of digits to be displayed
  - The setting **stays in effect** until it is **reset （持续有效）**
- Setting the **Fill** Character:
  - Use the `fill(char)` function to set the fill character.
  - The character **remains** as the fill character until set again. **（持续有效）**
  - Example:

    ```
    int x = 42;
    cout.width(5);
    cout.fill('*');
    cout << x << '\n';
    ```

    Output:

    ```
    ***42
    ```

- **Flags（标记位）**：
  - `ios` defined a **word (16 bits)** to control I/O format
  - Each bit represent one format:

Constant	Value	Meaning	I/O	Default	Bit-field
`ios::skipws`	0x0001	Skip white spaces	I		No
`ios::left`	0x0002	Left adjustfied	O	Not set	`ios::adjustfield`
`ios::right`	0x0004	Right adjustfied	O	Set	`ios::adjustfield`
`ios::internal`	0x0008	Internal adjustfied	O		`ios::adjustfield`
`ios::dec`	0x0010	Decimal base	I/O	Set	`ios::basefield`
`ios::oct`	0x0020	Octal base	I/O	Not set	`ios::basefield`

Constant	Value	Meaning	I/O	Default	Bit-field
`ios::hex`	0x0040	Hexadecimal base	I/O	Not set	`ios::basefield`
`ios::showbase`	0x0080	Show the base of an output number	O	Not set	No
`ios::showpoint`	0x0100	Show point	O	Not set	No
`ios::uppercase`	0x0200	Uppercase	O	Not set	No
`ios::showpos`	0x0400	Show positive	O	Not set	No
`ios::scientific`	0x0800	Scientific	O	Not set	`ios::floatfield`
`ios::fixed`	0x1000	Fixed	O	Not set	`ios::floatfield`
`ios::unitbuf`	0x2000	Flush stream after output	O		No
`ios::stdio`	0x4000	Flush stdout and stderr after output	O		No

- To set a flag(s) we use the `setf` function:

  ```
  cout.setf(0x0001);
  cout.setf(ios::skipws);
  ```

- Set more than one bits simutaniously:

  ```
  cout.setf(0x0001|0x0002);
  cout.setf(ios::skipws|ios::left);
  ```

- To unset other flags, we use the `unsetf` function:

  ```
  cout.unsetf(flags);
  ```

- C++ also provides a short-hand to combine both operations:

  ```
  cout.setf(on_flags, off_flags);
  ```

  - First turns off the flags `off_flags`
  - Then turns on the flags `on_flags`
- Integer Base Example:

  ```
  int x = 42;

  cout.setf(ios::oct, ios::basefield);
  cout << x << '\n'; // Outputs 52\n
  cout.setf(ios::hex, ios::basefield);
  cout << x << '\n'; // Outputs 2a\n
  cout.setf(ios::dec, ios::basefield);
  cout << x << '\n'; // Outputs 42\n
  ```

- Floating Point Format:

  ```
  cout.setf(ios::scientific, ios::floatfield);
  cout << 123.45 << '\n'; // Outputs 1.2345e+02
  cout.setf(ios::fixed,ios::floatfield);
  cout << 5.67E1 << '\n'; // Outputs 56.7
  ```

  - Use function `precision(int)` to set the number of significant digits printed (may convert from fixed to scientific to print)
  - Effect of precision **depends on format**
    - scientific **(total significant digits总有效位数)**

```
float y = 23.1415;
cout.precision(1);
cout << y << '\n';
// Outputs 2e+01
cout.precision(2);
cout << y << '\n';
// Outputs 23
cout.precision(3);
cout << y << '\n';
// Outputs 23.1
```

- fixed **(how many digits after decimal point小数点后位数)**

```
cout.setf(ios::fixed,ios::floatfield);
cout.precision(1);
cout << y << '\n';
// Outputs 23.1
cout.precision(2);
cout << y << '\n';
// Outputs 23.14
cout.precision(3);
cout << y << '\n';
// Outputs 23.142
```

- Showing the Base:
  - The flag `ios::showbase` can be set (its **default is off**), it results in integers being printed in a way that demonstrates their base:
    - decimal - no change
    - **octal（八进制）** - leading `0`
    - **hexadecimal（十六进制）** - leading `0x`
- Showing the Plus Sign:
  - The flag `ios::showpos` can be set (its **default is off**) to print a `+` sign when a **positive integer or floating point** value is printed
- Showing Upper Case Hex Ints:
  - The flag `ios::uppercase` (**default off**) can be used to indicate that the letters making up **hexadecimal numbers should be shown as upper case（十六进制数字的字母会被大写展示）**
- Decimal Points in Floats:
  - Set flag `ios::showpoint` to make sure decimal point appears in output **(C++ only shows significant digits in default默认只展示有效的位)**
  - Example:

```
float y = 3.0;
cout << y << '\n'; // Outputs 3
cout.setf(ios::showpoint);
cout << y << '\n'; // Outputs 3.00000
```

- Displaying bools:
  - Variables of type bool print out as `0` (`false`) or `1` (`true`), to print out words (`false`, `true`) use flag `ios::boolalpha` **（使用该标志符输出字符串形式下的布尔字面量）**

- **`ios` member functions（`ios` 成员函数）**
  - `width()`
  - `fill()`
  - `precision()`
  - `setf()`
  - `unsetf()`
- **Manipulators（操纵符）**
  - A manipulator is a simple function that can be included in an insertion or extraction chain:

```
cout << manip1 << manip2 << manip3 << item;
```

- C++ manipulators:
  - must **include** `<iomanip>` to use
  - several are provided to do useful things
  - you can also **create your own manipulators**
- Manipulators **without arguments**:

Name	Description
`endl`	Outputs a newline character, **flushes** output
`dec`	Sets the base of int output to decimal
`hex`	Sets the base of int output to hexadecimal
`oct`	Sets the base of int output to octal

- Manipulators taking 1 argument:

Name	Description	Corresponding Member Function
`setw(int)`	Sets the width to `int` value	`width(int)`
`setfill(char)`	Sets fill char to `char` value	`fill(char)`
`setprecision(int)`	Sets precision to `int` value	`precision(int)`
`setbase(int)`	Sets int output to hex if `int` is `16`, oct if `int` is `8`, dec if `int` is `0` or `10`	
`setiosflags(flags)`	Set flags on	`setf(flags)`
`resetiosflag(flags)`	Set flags off	`unsetf(flags)`