ST. CHARLES COMMUNITY COLLEGE

CPT-182 - Evening - Programming in C++

Lecture 2

File I/O Streams, Branches, Loops

Dayu Wang

- **File Input and Output**

→ C++ uses streams to handle input and output.

→ To use file input, you need to open an input file stream.

Step 1 - "include" statements

```
#include <fstream>   // Use file input/output streams.
#include <iostream>  // Use standard input/output streams.
```

Step 2 - Create an "ifstream" object.

```
ifstream fin;  // Variable "fin" is an input file stream.
```

Step 3 - Open a file.

```
fin.open("input.txt");
```

C++ supports both absolute pathname and relative pathname.

Visual Studio's default directory is the (project folder)\(project folder).

You can also open a file when you create "fin".

```
ifstream fin("input.txt");  // Create "fin" and open the file.
```

If you use C++ version older than C++ 98, then you have to use below.

```
string filename = "input.txt";
fin.open(filename.c_str());
```

Step 4 - Check whether the file was successfully opened.

In Python, if you try to open a nonexistent file, runtime error will occur.

In C++, there will be no ~~error~~ when you open a nonexistent file.

Later, when you try to read the input file, you program will crash.

[Requirement] After opening an input file, you must check whether the file was successfully opened.

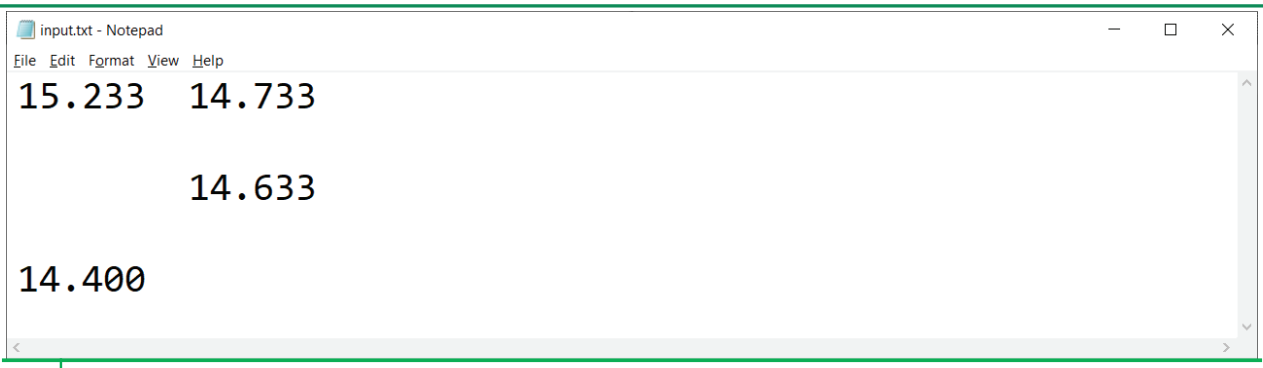
```
int main() {
    ifstream fin("input.txt");
    if (!fin.good()) {  // If the input file stream's state is not good.
        cout << "[Error] Nonexistent input file" << endl;
        system("pause");
        return -1;  // Error exit
    }
    // Read the data in the input file.
    return 0;  // Normal exit
}
```

"!fin.good()" can be replaced with "!fin".

```
if (!fin) {
```

Step 5 - Use ">>" to read the data from the file, just like using "cin".

```
int next;
fin >> next;
// Read the next integer in the file and store it in variable "next".
```


```
int main() {
    ifstream fin("input.txt");
    if (!fin) {
        cout << "[Error] Nonexistent input file" << endl;
        system("pause");
        return -1;
    }
    double d1, d2, d3, d4;
    fin >> d1;
    fin >> d2;
    fin >> d3;
    fin >> d4;
    cout << d1 << endl << d2 << endl << d3 << endl << d4 << endl;
    system("pause");
    return 0;
}
```

Console

```
15.233
14.733
14.633
14.4
```

Whitespaces are automatically ignored.

If you do not want whitespaces to be ignored...

The `getline()` function will read the "entire line" (subsequent text after the cursor in the same line) as a string.

```
int main() {
    ifstream fin("input.txt");
    if (!fin) {
        cout << "[Error] Nonexistent input file" << endl;
        system("pause");
        return -1;  // Error exit
    }
    string l1, l2, l3, l4;
    getline(fin, l1);
    getline(fin, l2);
    getline(fin, l3);
    getline(fin, l4);
    cout << l1 << endl << l2 << endl << l3 << endl << l4 << endl;
    system("pause");
    return 0;
}
```

Console

```
15.233  14.733

        14.633

14.400
```

If you want to read multiple numbers at once...

```
double d1, d2, d3, d4;
fin >> d1 >> d2 >> d3 >> d4;  // Read 4 values (store them in d1 to d4).
cout << d1 << endl << d2 << endl << d3 << endl << d4 << endl;
```


What if you write "`fin >> d1 >> d2 >> d3 >> d4;`" but there are only 3 numbers in the input file after the cursor?

"`fin >> d1;`" but there are no ~~numbers~~ after the cursor.

"`fin >> d1 >> d2;`" but there are only 0 or 1 number after the cursor.

"`fin >> d1 >> d2 >> d3;`" but there are only 0, 1, or 2 numbers after the cursor.

In all cases above, <u>program will crash</u>.

How to test whether "`fin >> d1 >> d2 >> d3 >> d4;`" succeeded?

```
double d1, d2, d3, d4;
bool success = (bool)(fin >> d1 >> d2 >> d3 >> d4);
cout << success << endl;
```

If "success" is `true`, that means we have successfully read 4 numbers from the input file, and store them in `d1` to `d4`.

If "success" is `false`, that means there are not ~~4 numbers~~ after the cursor (could have 0, 1, 2, or 3 numbers). Trying to read 4 numbers from the input file resulted in a failure.

Step 5 - Never forget to <u>close the input file</u> after reading the data!

```
fin.close();
```

[Requirement] Any file opened must be closed!

→ **To write data to an output file, you need to use output file stream.**

Step 1 - "include" statements

```
#include <fstream>   // Use input/output file streams.
#include <iostream>  // Use standard input/output streams.
```

Step 2 - Create an ofstream.

```
ofstream fout;  // Create an output file stream "fout".
```

Step 3 - Open the output file.

```
fout.open("output.txt");
```

```
ofstream fout("output.txt");
// Create output file stream "fout" and open output file "output.txt".
```

```
ofstream fout;
string filename = "output.txt";
fout.open(filename.c_str());  // Old C++ (earlier than 98).
```

If you attempt to open a file using ofstream but the file does not exist, your program will create a new file.

If you attempt to open a file using ofstream and the file does exist, the file will be overwritten.

All contents in the "old" file will disappear.

Step 4 - Use "<<" to write data to the output file, just like using "cout".

```
int main() {
    ifstream fin("input.txt");
    if (!fin) {
        cout << "[Error] Nonexistent input file" << endl;
        system("pause");
        return -1;  // Error exit
    }
    double d1, d2, d3, d4;
    fin >> d1 >> d2 >> d3 >> d4;
    ofstream fout("output.txt");
    fout << d1 << endl << d2 << endl << d3 << endl << d4 << endl;
    fin.close();
    fout.close();
    return 0;
}
```

Step 5 - Never forget to close the input and output files at the end of the program!

output.txt

```
15.233
14.733
14.633
14.4
```

- **if-else Branches**

Syntax of if branch

```
if (boolean_expression) {
    // Statements if boolean_expression is true.
}
```

Syntax of if-else branch

```
if (boolean_expression) {
    // Statements if boolean_expression is true.
} else {
    // Statements if boolean_expression is false.
}
```

Syntax of multiple if-else branches

```
if (exp_1) {
    // Statements if exp_1 is true.
} else if (exp_2) {
    // Statements if exp_1 is false but exp_2 is true.
} else if (exp_3) {
    // Statements if exp_1 and exp_2 are false but exp_3 is true.
} else {
    // Statements if exp_1, exp_2 and exp_3 are all false.
}
```


→ **If there is only one statement in an `if` branch, then {} is unnecessary.**

```
if (x1 < x2)
    cout << "x1 is smaller." << endl;
else
    cout << "x2 is smaller." << endl;
```

```
// You can also write in the same line.
if (x1 < x2) cout << "x1 is smaller." << endl;
else cout << "x2 is smaller." << endl;
```

→ **[Exercise] What is the output value?**

```
int main() {
    const double PI = 3.1416;
    int a = 3, b = 0;
    if (a > PI)
        if (a - PI >= PI)
            b = 1;
    else b = 2;
    cout << b << endl;
    system("pause");
    return 0;
}
```

[Good Habit] Please always add {}, even if there is only one statement.

- **Comparison Operators**

→ ">", ">=", "<", "<=", "==", "!=" (Skipped)

→ [Exercise] true or false

```
int x = 3, y = 4;
cout << (++x < y) << endl;
```

```
int x = -3, y = -2;
cout << ((x++) == y) << endl;
```

- **Logical Operators**

→ Logic AND

In C++, "&&" is the logic AND operator.

→ Logic OR

In C++, "||" is the logic OR operator.

→ Logic NOT

In C++, "!" is the logic NOT operator.

→ [Example] Max of Three Integers

```
int main() {
    // Let the user input three integers, find the max of the three.
    int x1, x2, x3;
    cout << "Enter three integers: ";
    cin >> x1 >> x2 >> x3;  // Let the user enter 3 integers at once.
    if (x1 >= x2 && x1 >= x3) { cout << "Max: " << x1 << endl; }
    else if (x2 >= x1 && x2 >= x3) { cout << "Max: " << x2 << endl; }
    else { cout << "Max: " << x3 << endl; }
    system("pause");
    return 0;
}
```

- **Order of Evaluation (Precedence Rule)**

→ "*" == "/" == "%" > "+" (addition) == "-" (subtraction)

→ Arithmetic operators > comparison operators

→ Comparison operators > logical operators

→ What is the "best precedence rule"?

If you are not sure, use ().

- **Do not use ~~comparison chaining~~.**

```
int main() {
    int x = -3, y = -2, z = -1;
    bool compare = x < y < z;
    cout << compare << endl;
    system("pause");
    return 0;
}
```

Is compare true or false?

Console
```
0
```

First, "x < y" is evaluated, which is true.

Then, true is converted to 1.

Then, "1 < z" is evaluated, which is false.

So, the final result is false.

→ [Good Habit] Do not use ~~comparison chaining~~ in C++.

```
int main() {
    int x = -3, y = -2, z = -1;
    bool compare = x < y && y < z;
    cout << compare << endl;
    system("pause");
    return 0;
}
```

Console
```
1
```


- **switch Statements**

Syntax of switch block

```
switch (var) {
case 0:
    // Statements if var == 0
    break;
case 3:
    // Statements if var == 3
    break;
case 'a':
    // Statements if var == 97
    break;
default:
    // Statements if none of the above cases is true.
}
```

→ A switch statement can more clearly represent multi-branch behavior involving a variable being compared to constant values.

The program executes the first case whose constant expression matches the value of the switch expression, executes that case's statements, and then jumps to the end.

If no case matches, then the default case statements are executed.

→ **The switch variable var must be an integer type.**

`short`, `int`, `long long`, `char`

It cannot be ~~a string type~~.

→ **Each case variable must be a literal or `const` variable.**

→ **In a `switch` block, a `break` statement stops executing the following statements and jumps out of the `switch` block immediately.**

Omitting the `break` statement for a case will cause the statements within the next case to be executed.

Such "falling through" to the next case can be useful when multiple cases, such as cases 0, 1, and 2, should execute the same statements.

→ **Any `switch` block can be rewritten using multiple `if`-`else` branches.**

```
int main() {
    int x;
    cout << "Enter an integer: ";
    cin >> x;
    switch (x) {
    case 0:
    case 1:
        x++;
    case 2:
        x += 3;
        break;
    case 3:
        x = -x;
        break;
    default:
        x = 0;
    }
}
```

```
int main() {
    int x;
    cout << "Enter an integer: ";
    cin >> x;
    if (x == 0 || x == 1) {
        x += 4;
    } else if (x == 2) {
        x += 3;
    } else if (x == 3) {
        x = -x;
    } else {
        x = 0;
    }
}
```

→ **[Application of `switch` blocks] Menu-based system**

See sample code.

- **Conditional Statement**

→ In C++, the "?:" is the conditional operator.

```
A ? B : C
```

1) Statement A is evaluated first.
2) If A is true, then B is evaluated.
3) If A is false, then C is evaluated.

→ This is a ternary operator (3 operands).

```
int main() {
    /* Find the absolute value. */
    int x;
    cout << "Enter an integer: ";
    cin >> x;
    int abs_val = x >= 0 ? x : -x;
    cout << "Absolute value: " << abs_val << endl;
    system("pause");
    return 0;
}
```

"?:" can always be rewritten as "if...else..." statements.

- **do-while Loop**

→ A do-while loop is a loop construct that first executes the loop body's statements, then checks the loop condition.

Syntax of do-while loop

```
do {
    // Statements in the loop (loop body)
} while (condition);
```

→ A do-while loop uses the "repeat ... until ..." logic.

Repeat doing something until the condition is no longer true.

```
int main() {
    /* Calculate 1 + 2 + 3 + ... + 100 = ? */
    unsigned int result = 0, current = 1;
    do {
        result += current++;
    } while (current <= 100);
    cout << result << endl;
    system("pause");
    return 0;
}
```

Console: 5050

- **while Loop**

→ A while loop repeatedly executes the statements in the loop body while the loop's expression evaluates to true.

Syntax of while loop

```
while (condition) {
    // Statements in the loop (loop body)
}
```

```
int main() {
    /* Calculate 1 + 2 + 3 + ... + 100 = ? */
    unsigned int result = 0, current = 1;
    while (current <= 100) {
        result += current++;
    }
    cout << result << endl;
    system("pause");
    return 0;
}
```

Console: 5050

	do-while Loop	while Loop
Checking condition	At least once	At least once
Executing loop body	At least once	0 or more times

- **[Demo] While there are more input values...**

→ A rectangle has a width and a height.

→ The area of a rectangle is width × height.

→ The input file stores several rectangles.

Each line stores a rectangle.

Each line contains 2 positive integers, width and height.

Rectangles.txt - Notepad

```
4    8  → Rectangle 1
10   5  → Rectangle 2
20   2  → Rectangle 3
5    5  → Rectangle 4
↑    ↑
Width Height
```

Ln 4, Col 7 | 100% | Windows (CRLF) | UTF-8

See solution in sample code.

→ Calculate the areas of all the rectangles in the input file.

You cannot assume how many rectangles are in the input file. In other words, no matter how many rectangles are stored in the input file, your program should correctly process all of them.

→ Write your results to the output file (one result per line).

- **for Loop**

Syntax of for loop

```
for (statement_1; statement_2; statement_3) {
    // Statements in the loop (loop body)
}
```

→ `statement_1` is the loop variable initialization.

You can declare and initialize the loop variable here.

→ `statement_2` is the loop expression (evaluated to `true` or `false`).

→ `statement_3` is the loop variable update.

→ Sequence of execution

1) `statement_1` executes.
2) `statement_2` is checked.
3) If `statement_2` is `false`, then jump out of the loop immediately (terminal).
4) If `statement_2` is `true`, then the loop body is executed.
5) After the loop body is executed, `statement_3` is executed.
6) After `statement_3` is executed, `statement_2` is checked again.
7) Repeat 3) to 6).

- **[Example] Calculate 1 + 2 + 3 + ··· + 100.**

```
1  int main() {
2      /** Calculate 1 + 2 + 3 + ... + 100 = ? */
3      unsigned int result = 0;
4      for (unsigned int current = 1; current <= 100; current++) {
5          result += current;
6      }
7      cout << result << endl;
8      system("pause");
9      return 0;
10 }
```

Console: 5050

→ `statement_1` can be absent.

```
1  int result = 0, current = 0;
2  for (; current <= 100; current++) { result += current; }
```

→ `statement_2` can be absent.

```
1  int result = 0, current = 0;
2  for (;; current++) {
3      result += current;
4      if (current == 100) { break; }
5  }
```

→ `statement_3` can be absent.

```
1  int result = 0, current = 0;
2  for (;;) {
3      result += current++;
4      if (current > 100) { break; }
5  }
```

- **Programming Habit**

```
for (int i = 0; i < 100; i++) {
```

→ Can `i++` be replaced by `++i`?

→ Can `i++` be replaced by `i += 1`?

→ Can `i++` be replaced by `i = i + 1`?

→ [Requirement] In a `for` loop, use `i++` or `++i` and do **not** use ~~`i += 1`~~ or ~~`i = i + 1`~~.

- **Break and Continue (Skipped)**