C/C++ Program Design

LAB 5

CONTENTS

- Learn Relational Expressions
- Master while, do-while and for loops
- Learn logical operators
- Master branching statements
- Master switch multi-branch statement
- Master the use of break and continue statements
- File operation

2 Knowledge Points

2.1 Relational Operators

2.2 Repetition Control Structure

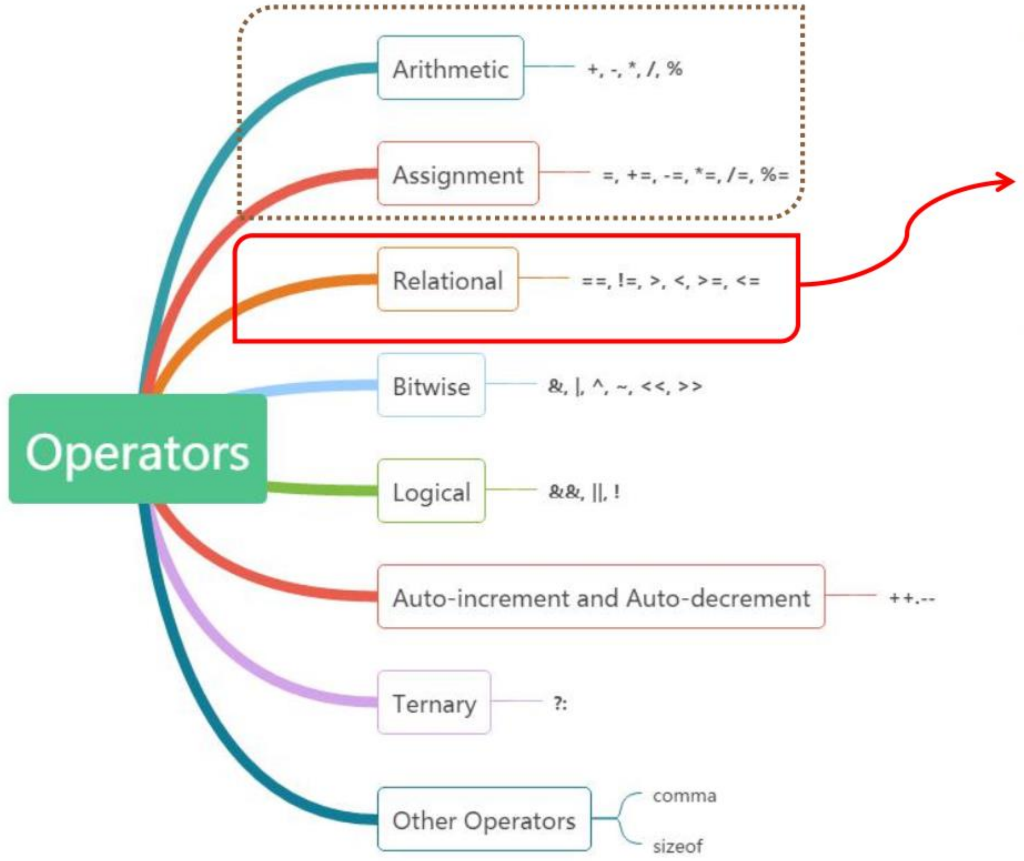
2.3 Logical Operators

2.4 Selection Control Structure

2.5 continue and break statement

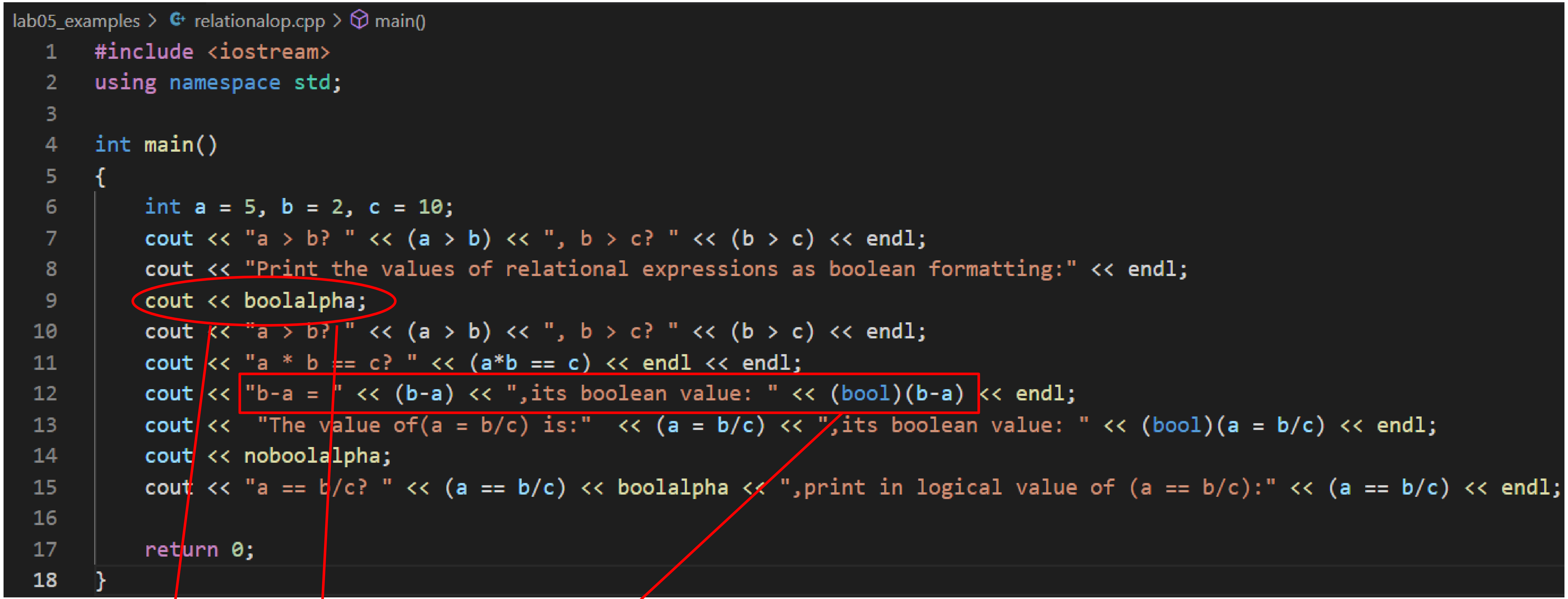
2.6 File input/output

2.1 Relational operators

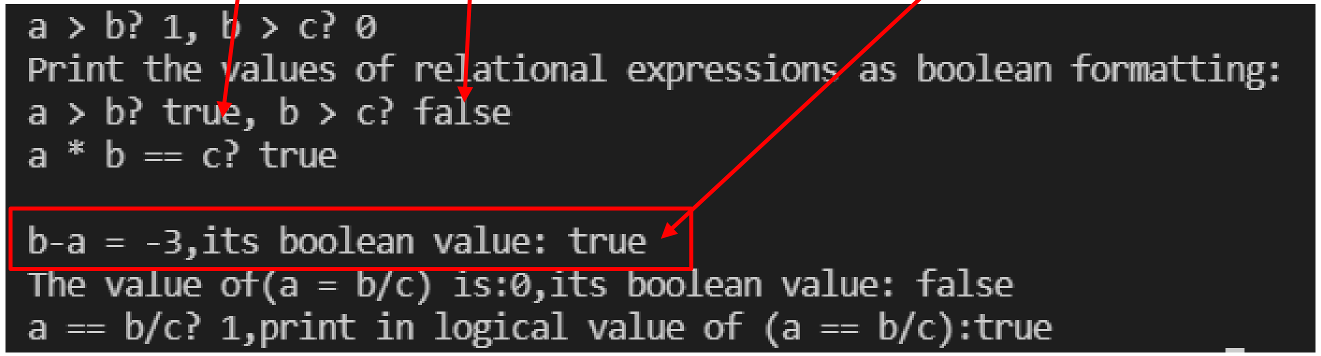


operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

The values of relational expressions is **0 for false** or **1 for true** by default. You can set the formatting of the output using **boolalpha** manipulator or setf().
setf(ios_base::boolalpha);


lab05_examples > relationalop.cpp > main()

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

int main()
{
    int a = 5, b = 2, c = 10;
    cout << "a > b? " << (a > b) << ", b > c? " << (b > c) << endl;
    cout << "Print the values of relational expressions as boolean formatting:" << endl;
    cout << boolalpha;
    cout << "a > b? " << (a > b) << ", b > c? " << (b > c) << endl;
    cout << "a * b == c? " << (a*b == c) << endl << endl;
    cout << "b-a = " << (b-a) << ",its boolean value: " << (bool)(b-a) << endl;
    cout <<  "The value of(a = b/c) is:"  << (a = b/c) << ",its boolean value: " << (bool)(a = b/c) << endl;
    cout << noboolalpha;
    cout << "a == b/c? " << (a == b/c) << boolalpha << ",print in logical value of (a == b/c):" << (a == b/c) << endl;

    return 0;
}
```



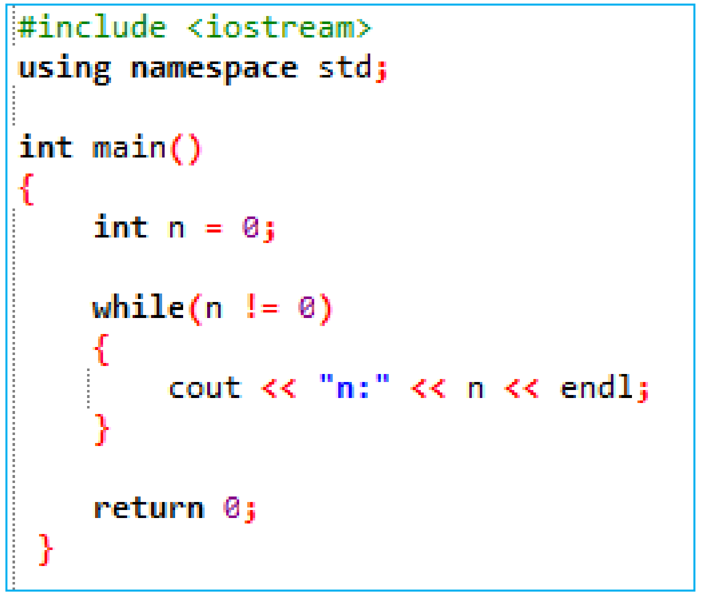
```
a > b? 1, b > c? 0
Print the values of relational expressions as boolean formatting:
a > b? true, b > c? false
a * b == c? true

b-a = -3,its boolean value: true
The value of(a = b/c) is:0,its boolean value: false
a == b/c? 1,print in logical value of (a == b/c):true
```

You can convert the values of arithmetic expressions to bool type explicitly. 0 for false and non-zero for true.

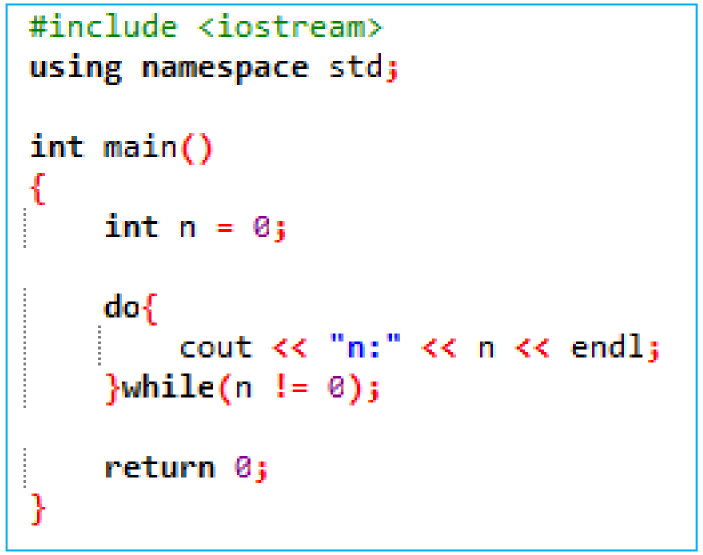
2.2 Repetition Control Structure

Difference between while and do-while loop

- **while:** The loop condition is tested at the beginning of the loop before the loop is performed.
- **do-while:** The loop condition is tested after the loop body is performed. Therefore, the loop body will always execute at least once.



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

int main()
{
    int n = 0;

    while(n != 0)
    {
        cout << "n:" << n << endl;
    }

    return 0;
}
```

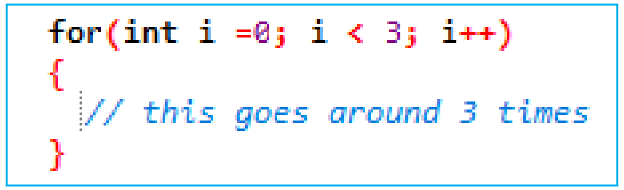


```
#include <iostream>
using namespace std;

int main()
{
    int n = 0;

    do{
        cout << "n:" << n << endl;
    }while(n != 0);

    return 0;
}
```

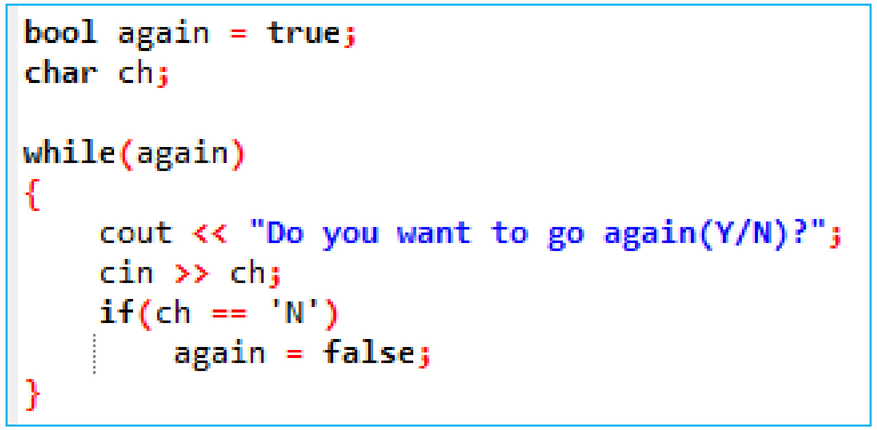
Compare these two outputs

Difference between for and while loop

They can both do the same things, but in general if you know how many times you will loop, use a for, otherwise, use a while.

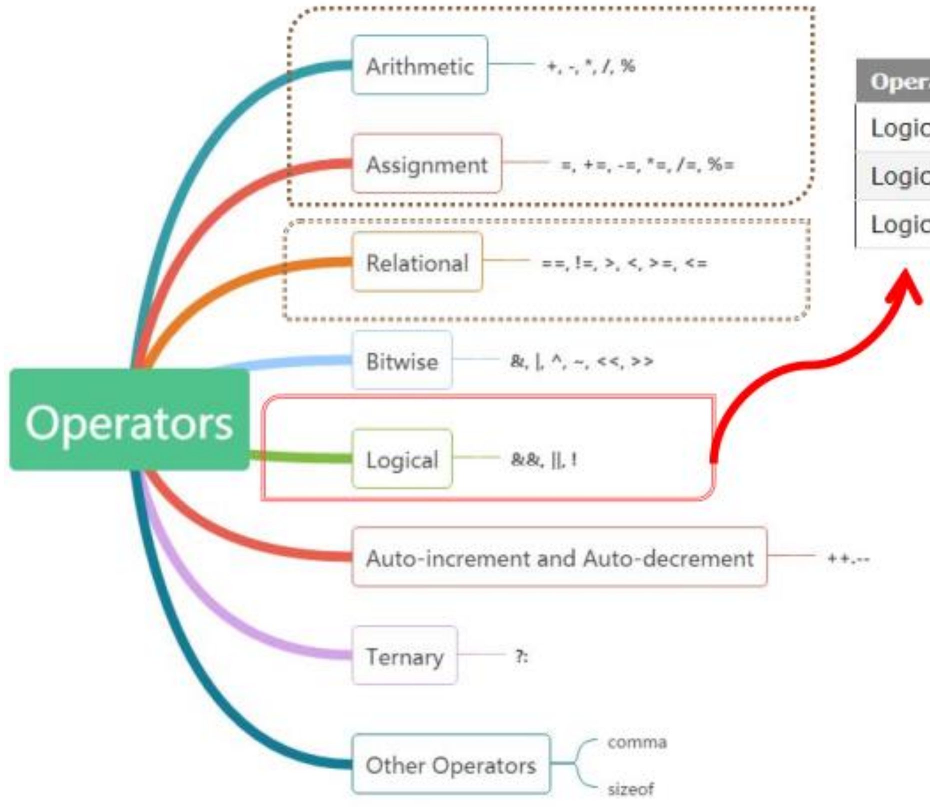


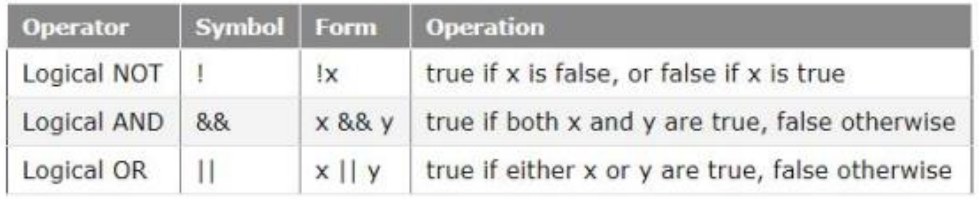
```
for(int i =0; i < 3; i++)
{
    // this goes around 3 times
}
```



```
bool again = true;
char ch;

while(again)
{
    cout << "Do you want to go again(Y/N)?";
    cin >> ch;
    if(ch == 'N')
        again = false;
}
```

2.3 Logical Operator



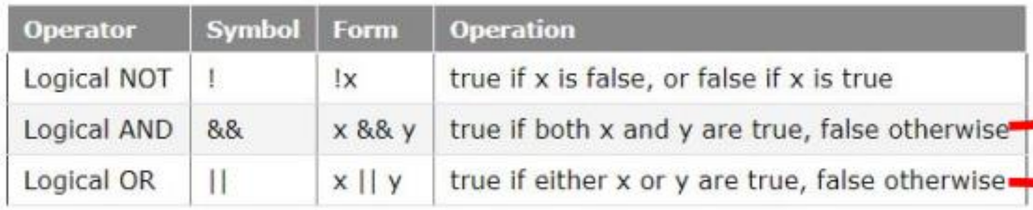


Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR	||	x || y	true if either x or y are true, false otherwise

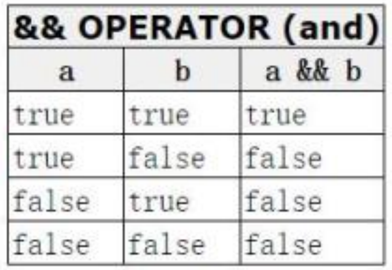
The values of logical expressions is **0 for false** or **1 for true** by default. You can set the formatting of the output using **boolalpha** manipulator or setf().
setf(ios_base::boolalpha);

Logical Operators (!, &&, ||)

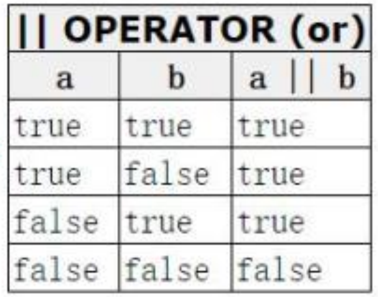
The logical operators are:



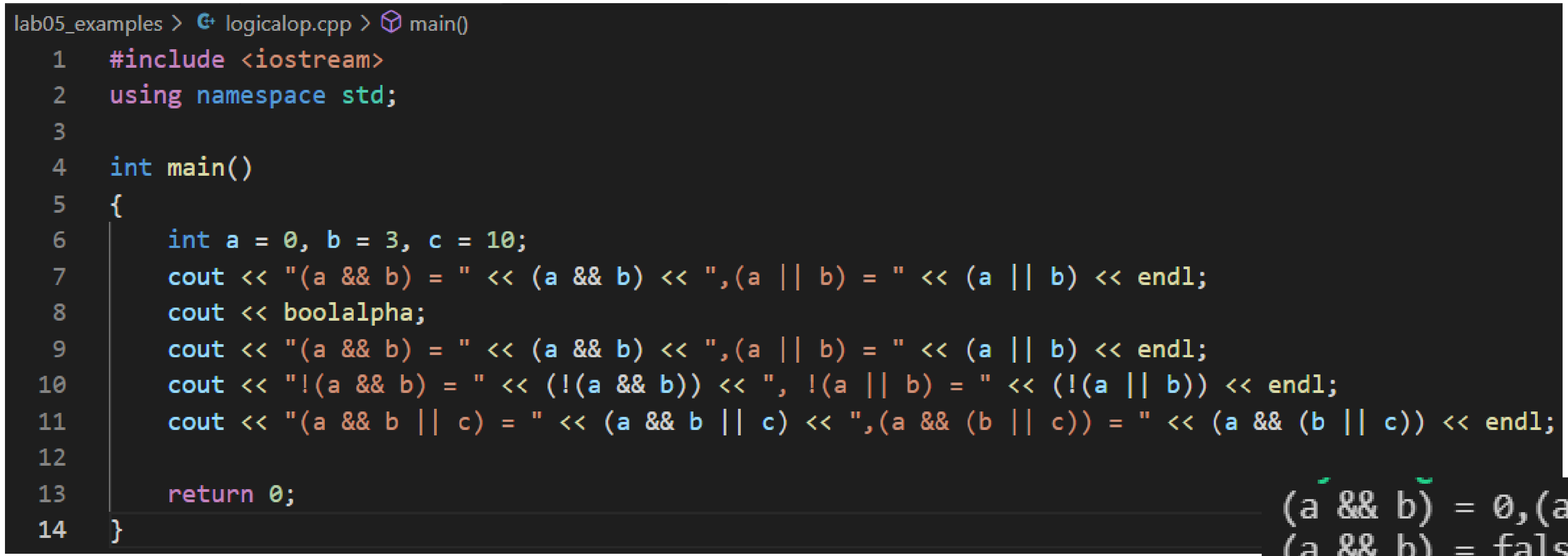
Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR	||	x || y	true if either x or y are true, false otherwise



&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

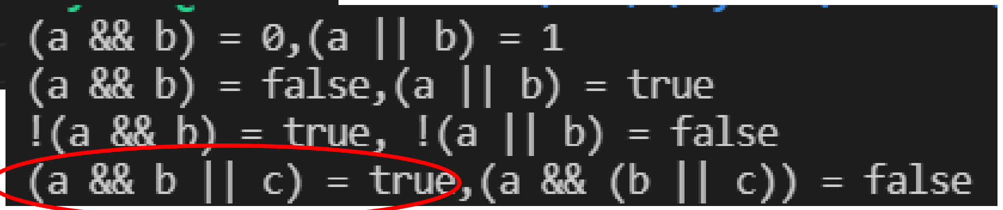


|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false



lab05_examples > logicalop.cpp > main()

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

int main()
{
    int a = 0, b = 3, c = 10;
    cout << "(a && b) = " << (a && b) << ",(a || b) = " << (a || b) << endl;
    cout << boolalpha;
    cout << "(a && b) = " << (a && b) << ",(a || b) = " << (a || b) << endl;
    cout << "!(a && b) = " << (!(a && b)) << ", !(a || b) = " << (!(a || b)) << endl;
    cout << "(a && b || c) = " << (a && b || c) << ",(a && (b || c)) = " << (a && (b || c)) << endl;

    return 0;
}
```

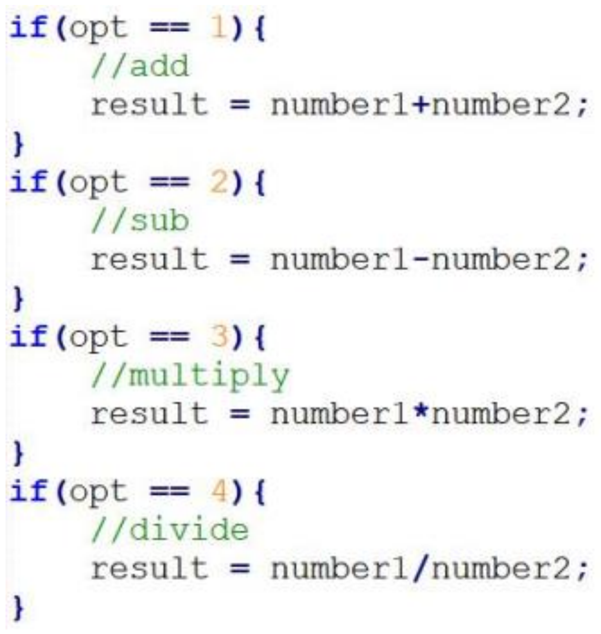


```
(a && b) = 0,(a || b) = 1
(a && b) = false,(a || b) = true
!(a && b) = true, !(a || b) = false
(a && b || c) = true,(a && (b || c)) = false
```

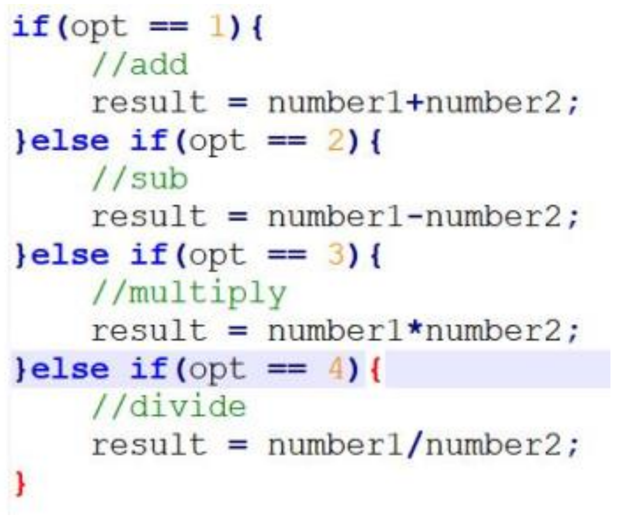
&& has higher precedence than ||.

2.4 Selection Control Structure

if and if-else statement



```
if(opt == 1){
    //add
    result = number1+number2;
}
if(opt == 2){
    //sub
    result = number1-number2;
}
if(opt == 3){
    //multiply
    result = number1*number2;
}
if(opt == 4){
    //divide
    result = number1/number2;
}
```



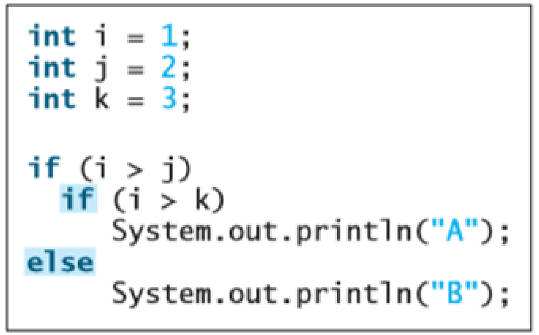
```
if(opt == 1){
    //add
    result = number1+number2;
}else if(opt == 2){
    //sub
    result = number1-number2;
}else if(opt == 3){
    //multiply
    result = number1*number2;
}else if(opt == 4){
    //divide
    result = number1/number2;
}
```

It's logical fine, but it doesn't work very efficiently.

It's more efficient. Because if opt==1, then the addition is performed, but the rest of the operation are definitely not to be look at.

The Dangling else problem

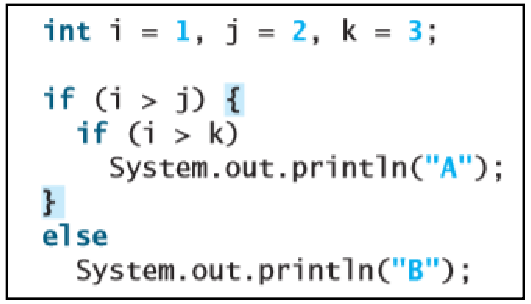
When an if statement is nested inside another if statement, the *else* clause always matches the most recent unmatched **if** clause in the same block.



```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

The complier ignores all indentation and matches the else with the preceding **if**.

To force the **else** clause to match the first *if* clause, you must add a pair of braces.



```
int i = 1, j = 2, k = 3;

if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

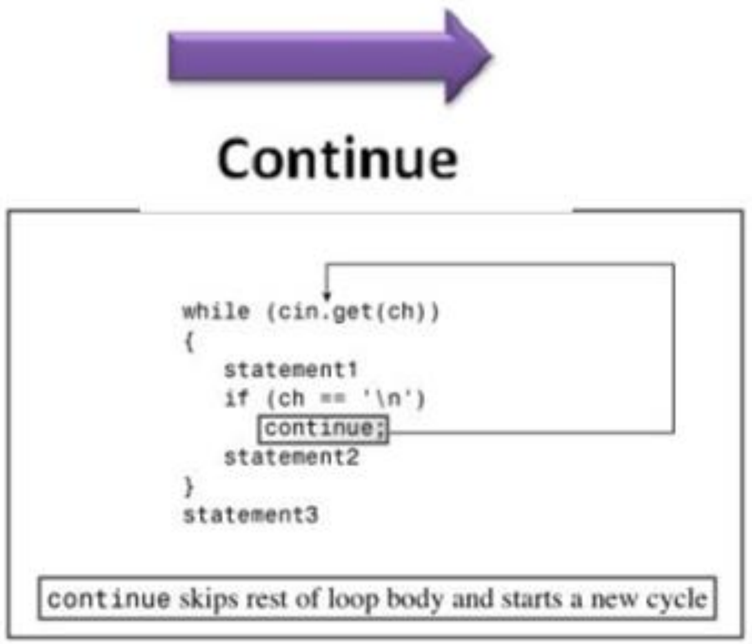
Difference between if and switch

- Check the Expression: An **if-else-if** statement can test boolean-expressions based on ranges of values or conditions, whereas a **switch** statement tests switch-expressions based only on a single int, enumerated value, byte, short, char. The **switch…case** can only judge the condition of equality, and **if** can judge any condition, such as equal, not equal, greater, less, etc.. If your alternatives involve ranges or floating-point tests or comparing two variables, you should use **if else**.

- switch case is faster than if-else: When the number of branches is large (generally larger than 5), **switch-case** is faster than **if-else-if**.

- Clarity in readability: A **switch-case** looks much cleaner than **if-else-if**.

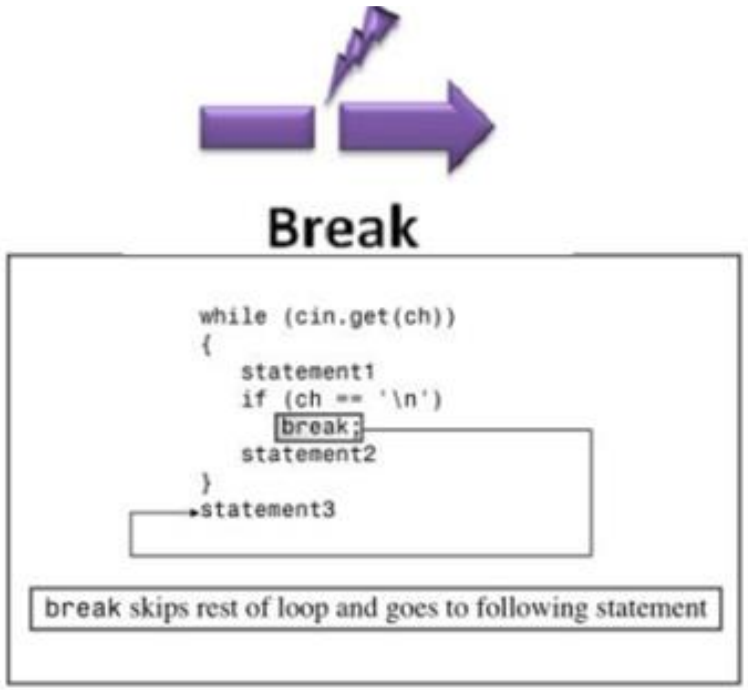
2.5 Difference between continue and break



```
while (cin.get(ch))
{
   statement1
   if (ch == '\n')
    continue;
   statement2
}
statement3
```

continue skips rest of loop body and starts a new cycle

The structure of continue statement



```
while (cin.get(ch))
{
   statement1
   if (ch == '\n')
    break;
   statement2
}
statement3
```

break skips rest of loop and goes to following statement

The structure of break statement

The main difference is as follows:

- **break** is used for immediate termination of loop
- **continue** terminate current iteration and resumes the control to the next iteration of the loop

2.6 Simple File Input and Output

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files

```
ofstream outClientFile;
```

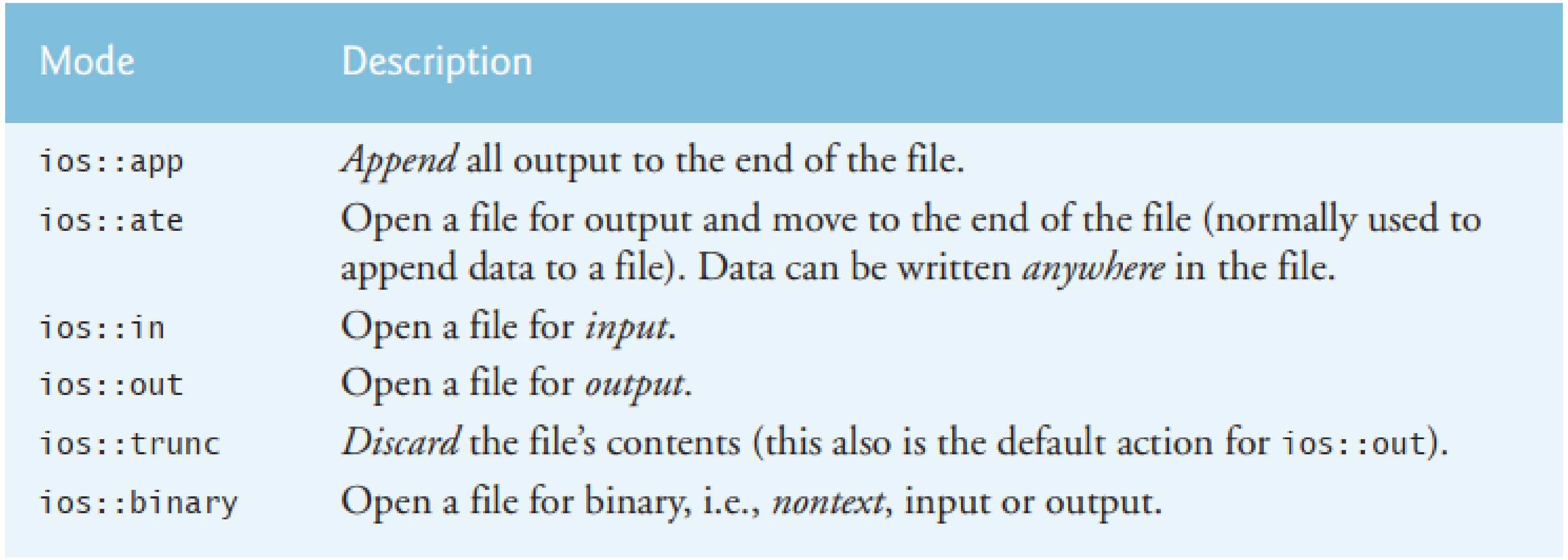
Create an ofstream object

```
outClientFile.open("clients.txt", ios::out);
```

The ofstream member function **open** opens a file and attaches it to an existing ofstream object. **ios::out** is the default value for the second argument.

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

File Open Modes



Mode	Description
`ios::app`	*Append* all output to the end of the file.
`ios::ate`	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written *anywhere* in the file.
`ios::in`	Open a file for *input*.
`ios::out`	Open a file for *output*.
`ios::trunc`	*Discard* the file's contents (this also is the default action for `ios::out`).
`ios::binary`	Open a file for binary, i.e., *nontext*, input or output.

Checking state flags

bad()	Returns true if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.
fail()	Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.
eof()	Returns true if a file open for reading has reached the end.
good()	It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true. Note that good and bad are not exact opposites (good checks more state flags at once).

It's simpler to use the **good()** method, which returns true if nothing when wrong.

```
while (inFile.good())   // while input good and not at EOF
{
    ...
}
```

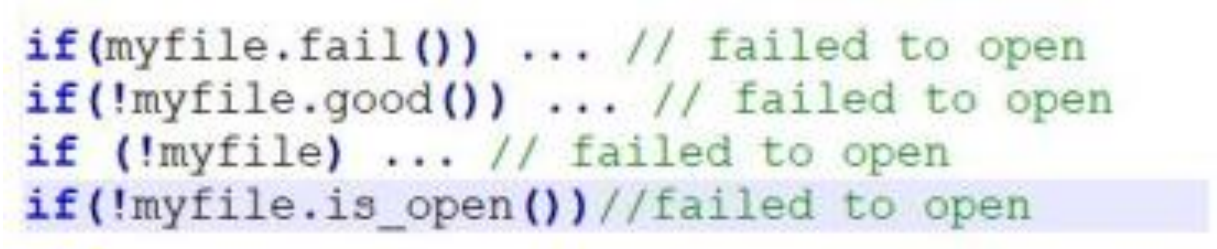
You can use the other methods to determine exactly why the loop terminated:

```
if (inFile.eof())
    cout << "End of file reached.\n";
else if (inFile.fail())
    cout << "Input terminated by data mismatch.\n";
else
    cout << "Input terminated for unknown reason.\n";
```

Checking state flags

is_open(): tests for some subtle problems that the other forms miss, such as attempting to open a file by using an inappropriate file mode.

The usual tests for successful opening of a file were the following:

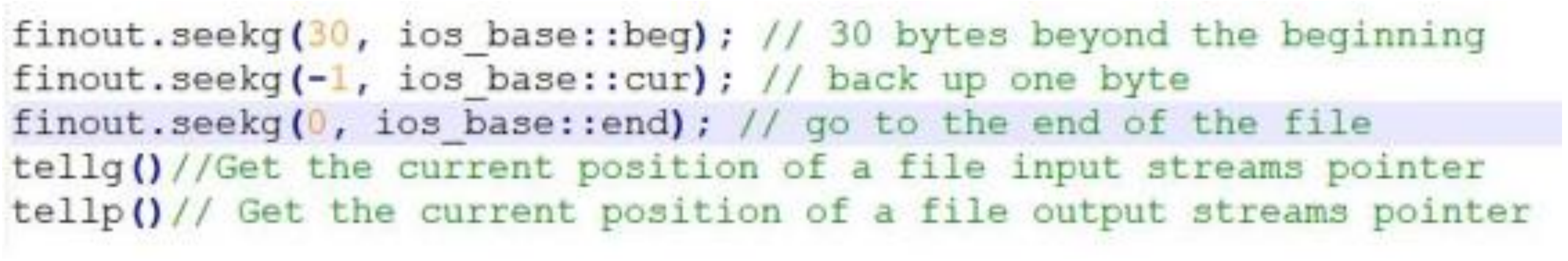


```
if(myfile.fail()) ... // failed to open
if(!myfile.good()) ... // failed to open
if (!myfile) ... // failed to open
if(!myfile.is_open())//failed to open
```

File position pointer:

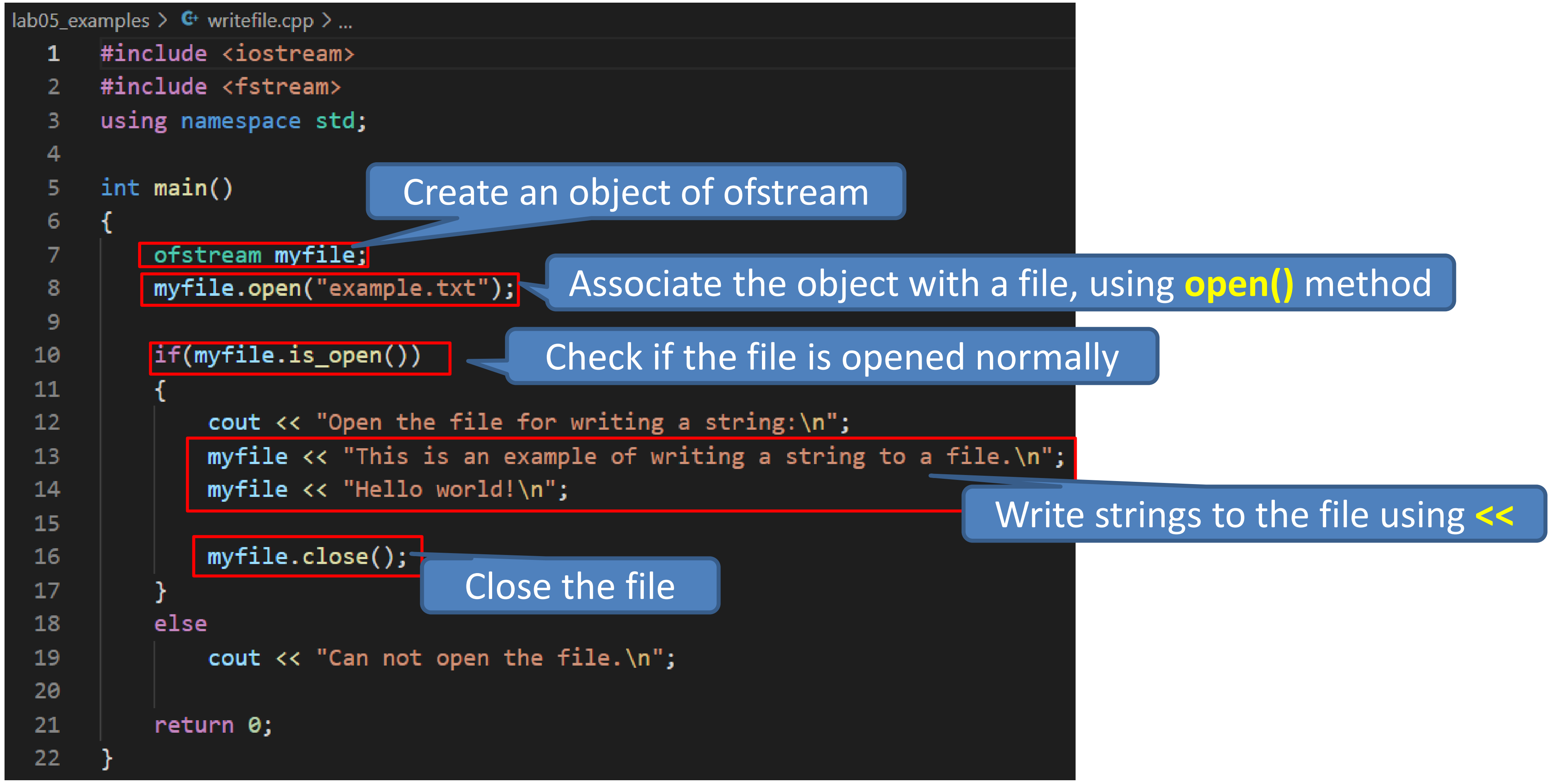
seekg(): moves the **input** pointer to a given file location

seekp(): moves the **output** pointer to a given file location.



```
finout.seekg(30, ios_base::beg); // 30 bytes beyond the beginning
finout.seekg(-1, ios_base::cur); // back up one byte
finout.seekg(0, ios_base::end); // go to the end of the file
tellg()//Get the current position of a file input streams pointer
tellp()// Get the current position of a file output streams pointer
```

Writing to a text file:


lab05_examples > writefile.cpp > ...

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

int main()
{
    ofstream myfile;
    myfile.open("example.txt");

    if(myfile.is_open())
    {
        cout << "Open the file for writing a string:\n";
        myfile << "This is an example of writing a string to a file.\n";
        myfile << "Hello world!\n";

        myfile.close();
    }
    else
        cout << "Can not open the file.\n";

    return 0;
}
```

Create an object of ofstream

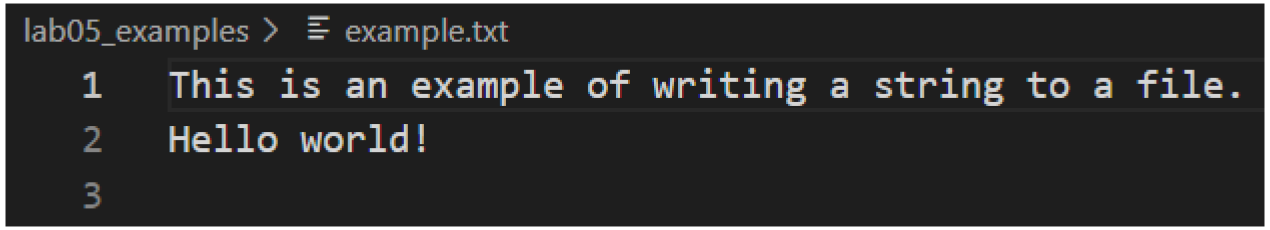
Associate the object with a file, using **open()** method

Check if the file is opened normally

Write strings to the file using **<<**

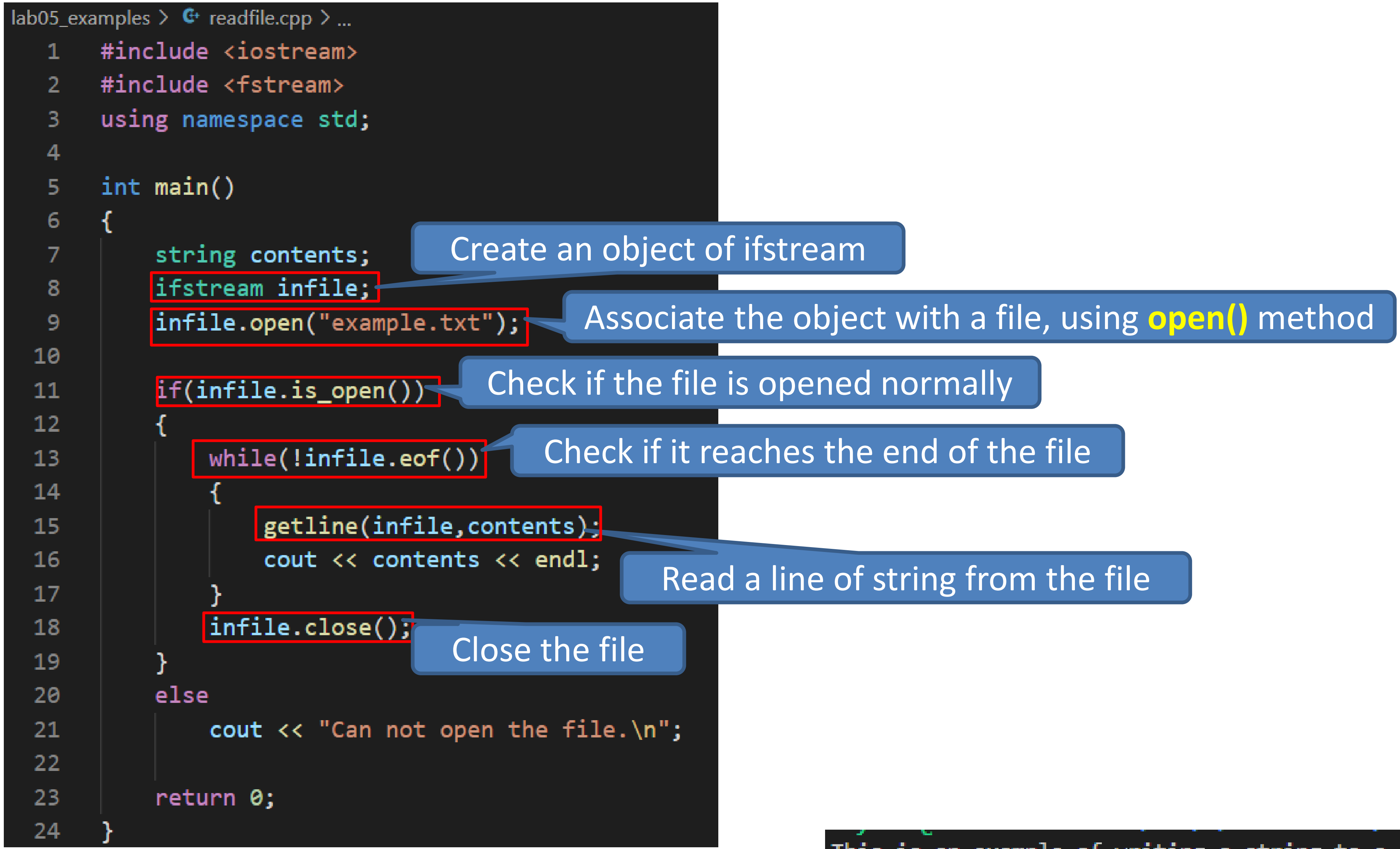
Close the file

The contents of the file:


lab05_examples > example.txt

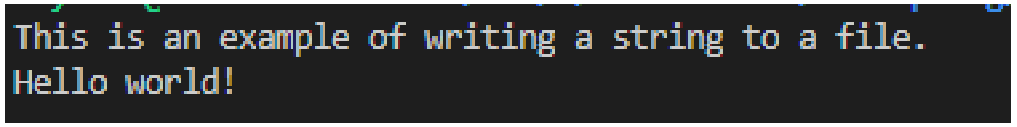
```
This is an example of writing a string to a file.
Hello world!

```

Reading from a text file:


lab05_examples > readfile.cpp > ...

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

int main()
{
    string contents;
    ifstream infile;
    infile.open("example.txt");

    if(infile.is_open())
    {
        while(!infile.eof())
        {
            getline(infile,contents);
            cout << contents << endl;
        }
        infile.close();
    }
    else
        cout << "Can not open the file.\n";

    return 0;
}
```

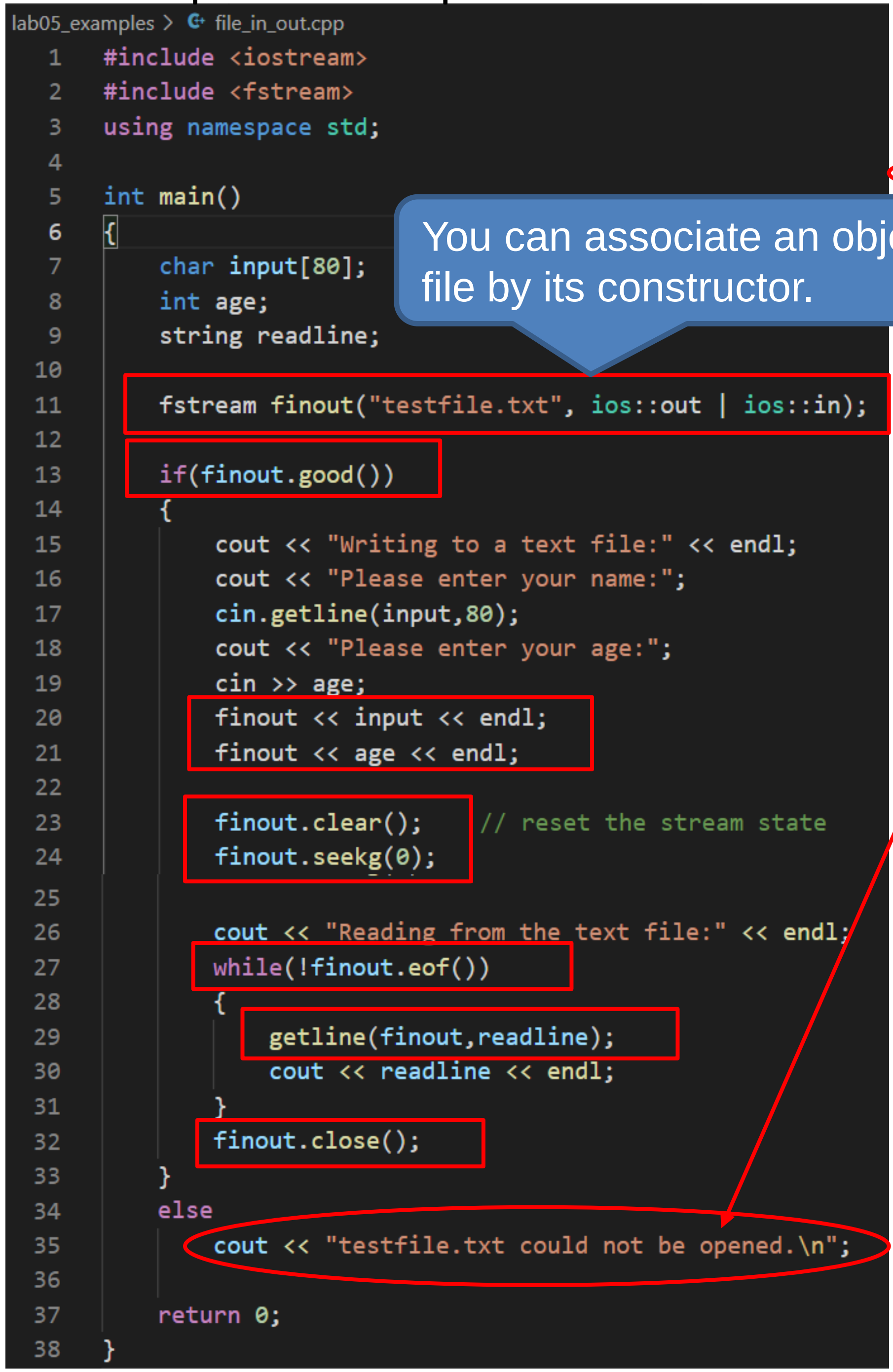
Create an object of ifstream
Associate the object with a file, using **open()** method
Check if the file is opened normally
Check if it reaches the end of the file
Read a line of string from the file
Close the file



```
This is an example of writing a string to a file.
Hello world!
```

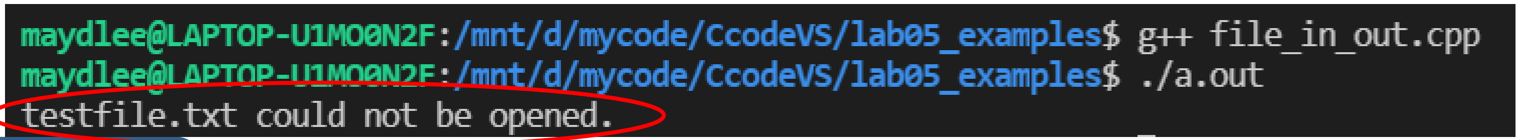
File input and output

testfile.txt must be existed, otherwise, the file cannot be opened.

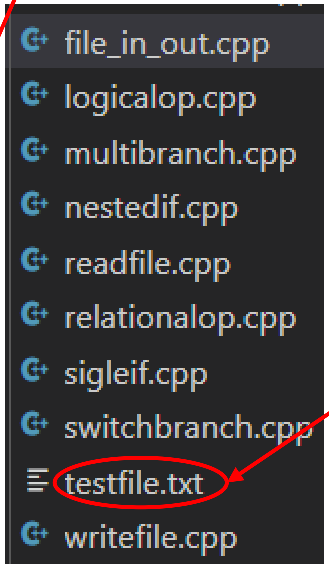

lab05_examples > file_in_out.cpp

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

int main()
{
    char input[80];
    int age;
    string readline;

    fstream finout("testfile.txt", ios::out | ios::in);

    if(finout.good())
    {
        cout << "Writing to a text file:" << endl;
        cout << "Please enter your name:";
        cin.getline(input,80);
        cout << "Please enter your age:";
        cin >> age;
        finout << input << endl;
        finout << age << endl;

        finout.clear();  // reset the stream state
        finout.seekg(0);

        cout << "Reading from the text file:" << endl;
        while(!finout.eof())
        {
            getline(finout,readline);
            cout << readline << endl;
        }
        finout.close();
    }
    else
        cout << "testfile.txt could not be opened.\n";

    return 0;
}
```

You can associate an object with a file by its constructor.

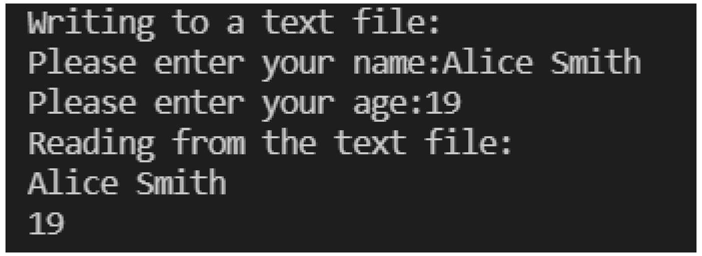


```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab05_examples$ g++ file_in_out.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab05_examples$ ./a.out
testfile.txt could not be opened.
```



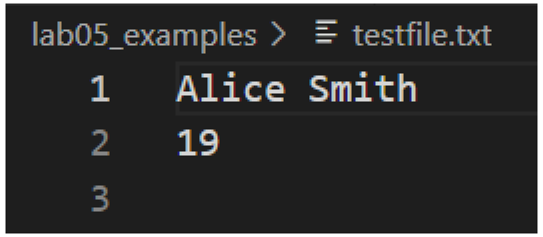
Create an empty file named testfile.txt in the current folder.

Run the program again:



```
Writing to a text file:
Please enter your name:Alice Smith
Please enter your age:19
Reading from the text file:
Alice Smith
19
```

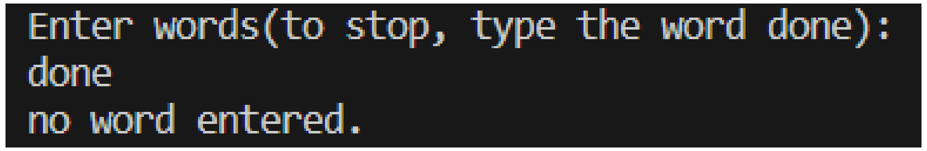
The contents of the file:


lab05_examples > testfile.txt

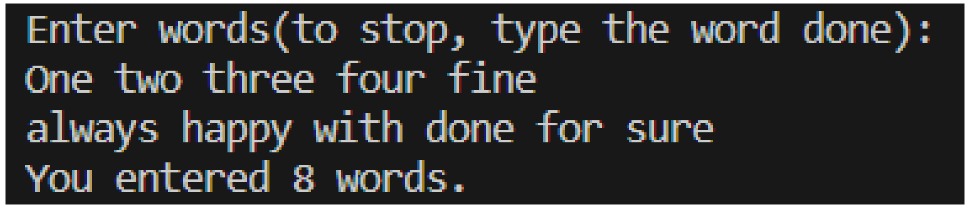
```
Alice Smith
19

```

3 Exercises

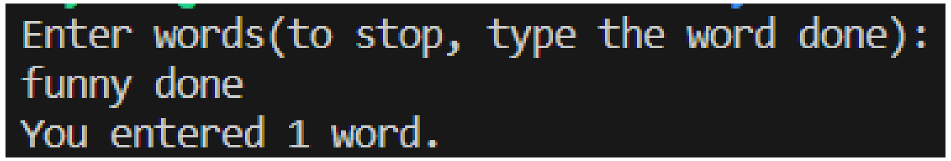
1. Write a program that uses a loop to read one word at a time until the word **done** is entered. The program should then report the number of words entered(not counting done). A sample run could look like this:



```
Enter words(to stop, type the word done):
done
no word entered.
```



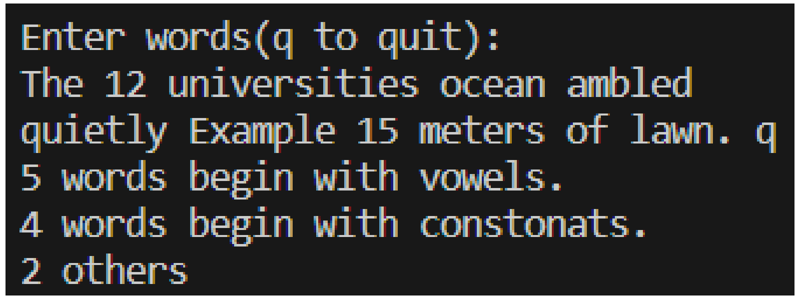
```
Enter words(to stop, type the word done):
One two three four fine
always happy with done for sure
You entered 8 words.
```



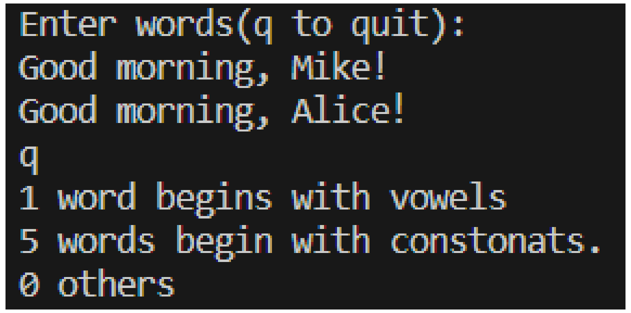
```
Enter words(to stop, type the word done):
funny done
You entered 1 word.
```

2. Write a program that reads input a word at a time until a lone **q** is entered. The program should then report the number of words that began with vowels, the number that began with consonants, and the number that fit neither of those categories. A sample run might look like this:

Hint: One approach is to use **isalpha(char)** to discriminate between words beginning with letters and those that don't.



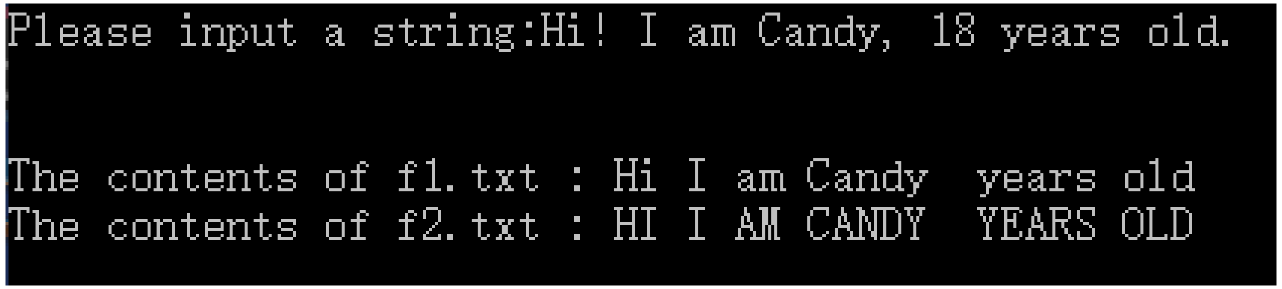
```
Enter words(q to quit):
The 12 universities ocean ambled
quietly Example 15 meters of lawn. q
5 words begin with vowels.
4 words begin with constonats.
2 others
```



```
Enter words(q to quit):
Good morning, Mike!
Good morning, Alice!
q
1 word begins with vowels
5 words begin with constonats.
0 others
```

3. Write a program that asks user to input a string by keyboard, save the letters and blanks of the string to a file named f1.txt. Convert the lower case letters into the upper case letters and save to another file named f2.txt. Show the contents of f1.txt and f2.txt on the screen respectively.

Sample output:



```
Please input a string:Hi! I am Candy, 18 years old.


The contents of f1.txt : Hi I am Candy  years old
The contents of f2.txt : HI I AM CANDY  YEARS OLD
```