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

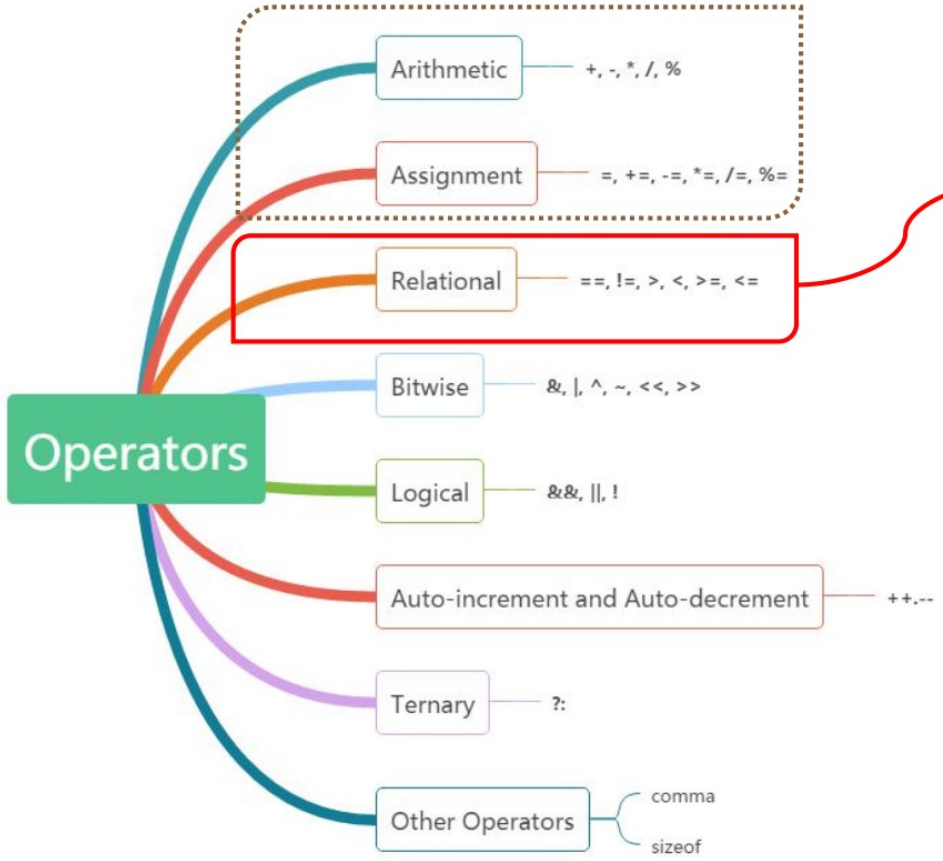
2.3 Logical Operators

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

2.2.1 while loop

The syntax of a **while** loop is:

```
while (testExpression)
{
      // codes
}
```

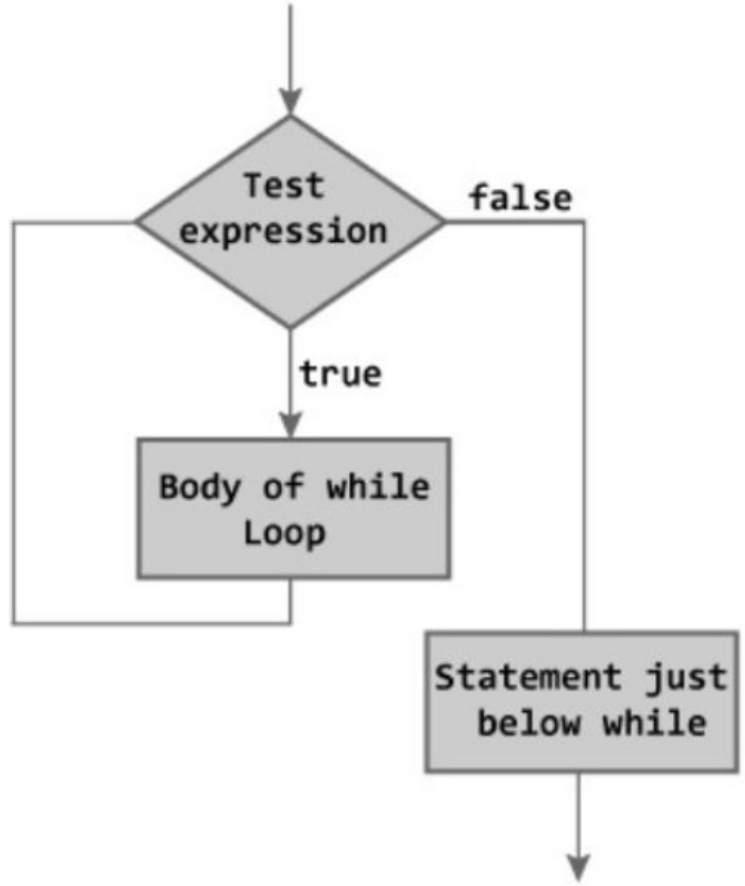


Figure: Flowchart of while Loop

Example: Compute factorial of a number.

Factorial of n = 1*2*3….*n

factorialwhile.cpp > main()

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

int main()
{
    int i, n;
    long factorial = 1;

    cout << "Enter a postive integer:";
    cin >> n;

    i = 1;
    while(i <= n)
    {
        factorial *= i;
        i++;
    }

    cout <<"Factorial of " << n << " = " << factorial << endl;

    return 0;
}
```

Sample output:

```
Enter a postive integer:5
Factorial of 5 = 120
```

2.2.2 do...while loop

The syntax of a **do...while** loop is:

```
do {
    // codes;
}
while (testExpression);
```

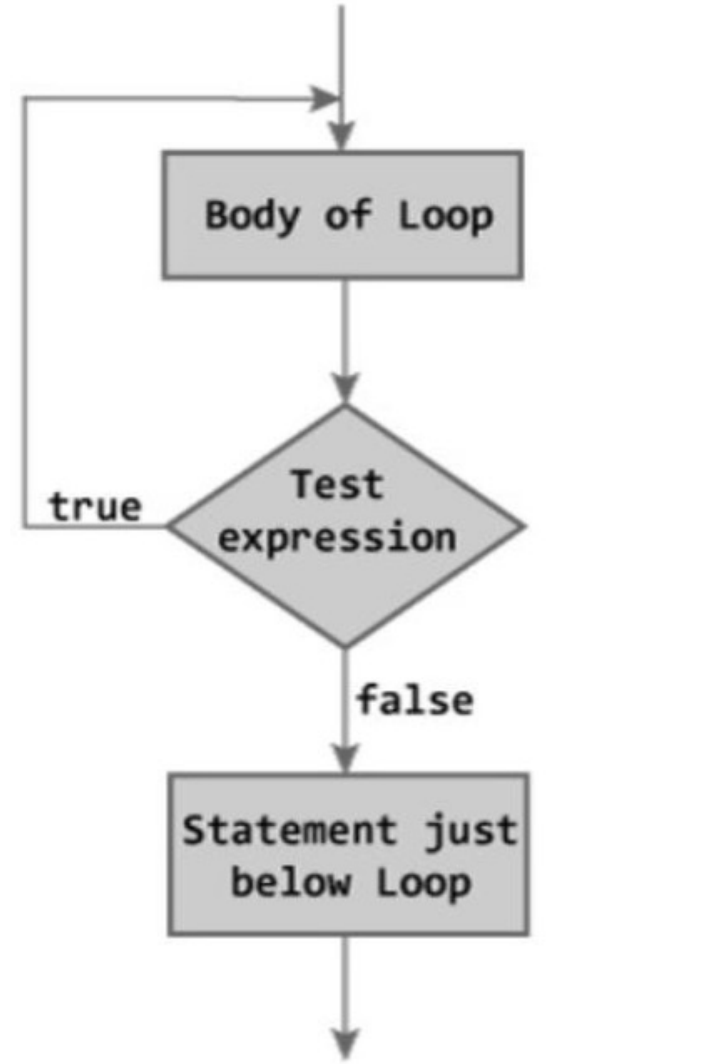


Figure: Flowchart of do...while Loop

Example: add numbers until user enters 0.

dowhile.cpp > ...

```cpp
using namespace std;

int main()
{
    float number, sum = 0.0;

    do{
        cout << "Enter a number(0 to teminate):";
        cin >> number;
        sum += number;
    }while(number != 0);

    cout << "Total sum = " << sum << endl;

    return 0;
}
```

Sample output:

```
Enter a number(0 to teminate):3.7
Enter a number(0 to teminate):-2
Enter a number(0 to teminate):9.8
Enter a number(0 to teminate):4.5
Enter a number(0 to teminate):0
Total sum = 16
```

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

2.2.3 for loop

The syntax of a **for** loop is:

```
for(initializationStatement; testExpression; updateStatement) {
     // codes
}
```

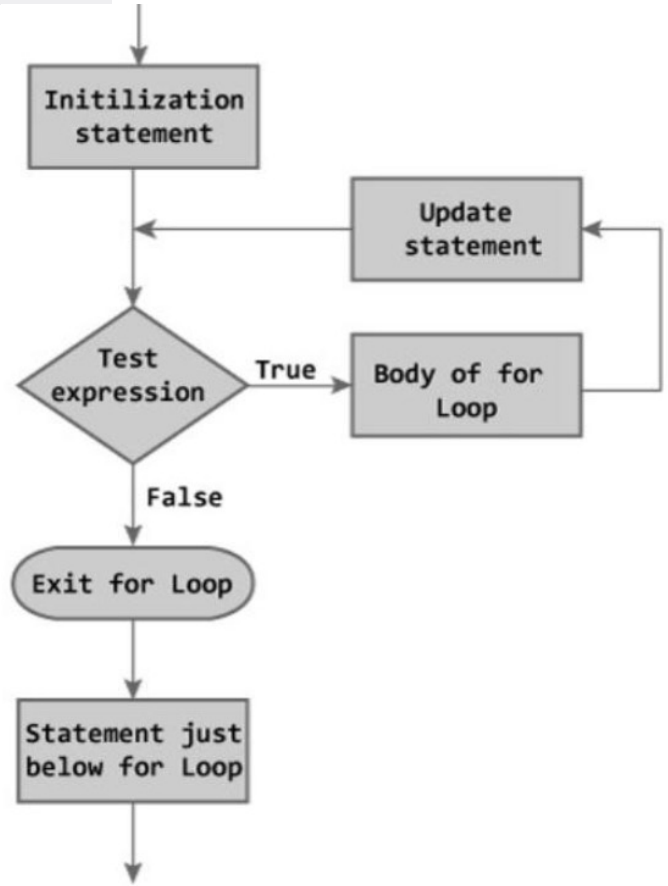


Figure: Flowchart of for Loop

Example: Compute factorial of a number.

Factorial of n = 1*2*3….*n

factorialfor.cpp > ...

```cpp

int main()
{
    int i,n;
    long factorial = 1;

    cout << "Enter a positive integer:";
    cin >> n;

    for(i = 1; i <= n; i++)
        factorial *= i;

    cout << "Factorial of " << n << " = " << factorial << endl;

    return 0;
}
```

Sample output:

```
Enter a positive integer:5
Factorial of 5 = 120
```

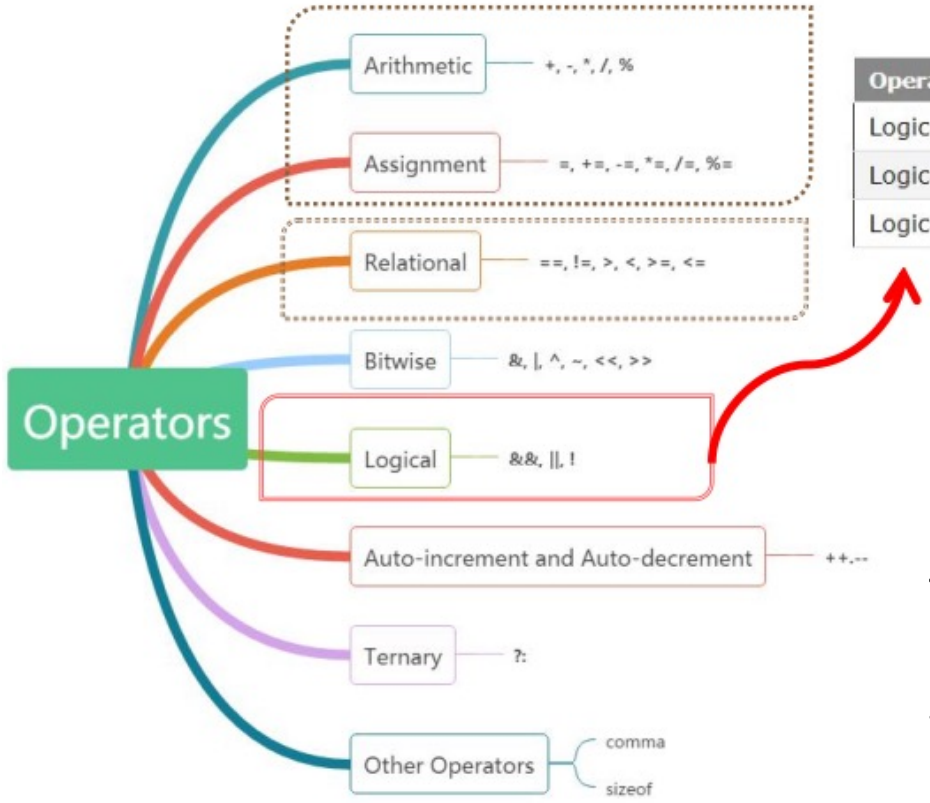
Difference between for and while loop

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;
}
```

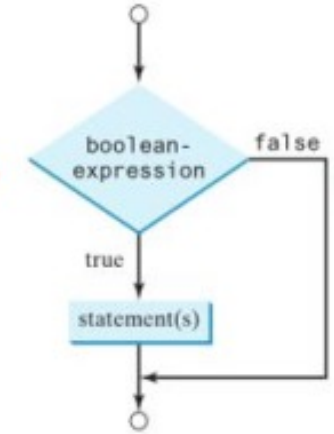
```
(a && b) = 0,(a || b) = 1
(a && b) = false,(a || b) = true
!(a && b) = true, !(a || b) = false
(a && b || c) = true,(a && (b || c)) = false
```

&& has higher precedence than ||.

2.4 Selection Control Structure

1. The syntax of the if statement

```
if(boolean-expression){
    statement(s);
}
```



lab05_examples > sigleif.cpp > ...

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

int main()
{
    int n ;

    cout << "Please input an integer:";
    cin >> n;

    if(n < 100)
        cout << n << " is less than 100." << endl;

    if(n > 100)
        cout << n << " is greater than 100." << endl;

    if(n == 100)
        cout << n << " is equal to 100." << endl;

    return 0;
}
```

```
Please input an integer:45
45 is less than 100.
```

```
Please input an integer:100
100 is equal to 100.
```

```
Please input an integer:231
231 is greater than 100.
```

2. The syntax of the Nested if statement

```
if(boolean-expression1)
{
     statement1;
     if(boolean-expression2)
     {
       statement2;
     }
}
```

nestedif.cpp > ...

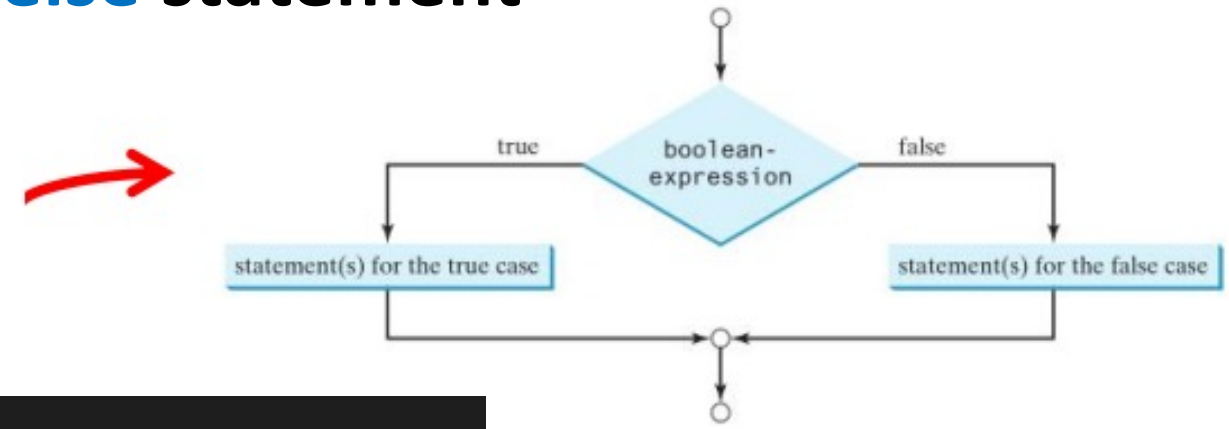
```
5   {
6       int n;
7
8       cout << "Please input an integer:";
9       cin >> n;
10
11      if(n < 100)
12      {
13          cout << n << " is less than 100, ";
14          if(n > 50)
15              cout << "but it is greater than 50." << endl;
16          if(n < 50)
17              cout << "and it is less than 50." << endl;
18      }
19
20      return 0;
21  }
```

```
Please input an integer:67
67 is less than 100, but it is greater than 50.
```

```
Please input an integer:23
23 is less than 100, and it is less than 50.
```

3. The syntax of the if-else statement

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



lab05_examples > doublebranch.cpp > ...

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

int main()
{
    int n;

    cout << "Please input an integer:";
    cin >> n;

    if(n > 100)
        cout << n << " is greater than 100." << endl;
    else
        cout << n << " is equal to or less than 100." << endl;

    return 0;
}
```

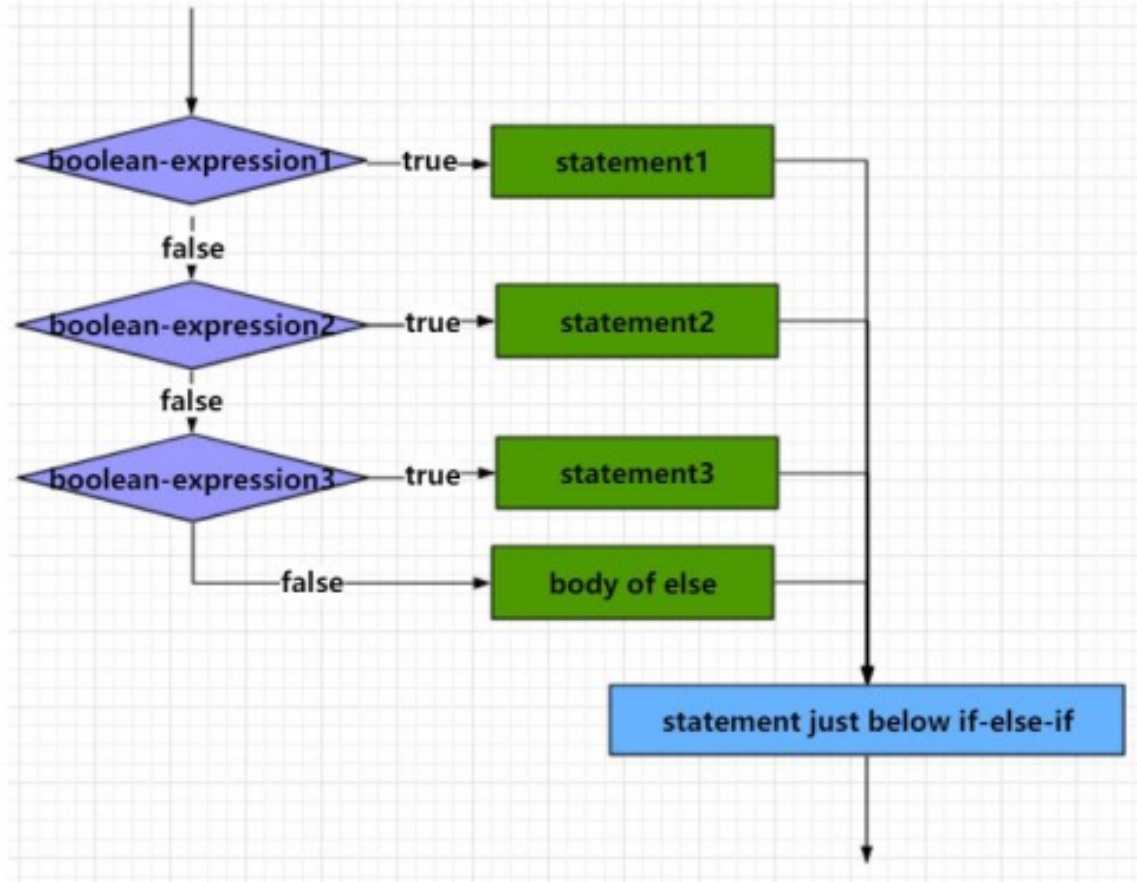
```
Please input an integer:123
123 is greater than 100.
```

```
Please input an integer:65
65 is equal to or less than 100.
```

```
Please input an integer:100
100 is equal to or less than 100.
```

4. The syntax of the if-else-if statement

```
//An if-else-if  statement can test boolean-expressions
//based on ranges of values or conditions
if (boolean-expression)
    //execute statement1
    statement1;
else if (boolean-expression)
    //execute statement2
    statement2;
...
else
    statement;
```



lab05_examples > multibranch.cpp > ...

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

int main()
{
    int n;

    cout << "Please input an integer bwteen 1 and 99999:";
    cin >> n;

    if(n < 10 && n >=1)
        cout << n << " is a one digit number." << endl;
    else if(n <100 && n >= 10)
        cout << n << " is a two digit number." << endl;
    else if(n < 1000 && n >= 100)
        cout << n << " is a three digit number." << endl;
    else if(n < 10000 && n >= 1000)
        cout << n << " is a four digit number." << endl;
    else if(n <100000 && n >= 10000)
        cout << n << " is a five digit number." << endl;
    else
        cout << n << " is not between 1 and 99999." << endl;

    return 0;
}
```

```
Please input an integer bwteen 1 and 99999:23
23 is a two digit number.
```

```
Please input an integer bwteen 1 and 99999:135
135 is a three digit number.
```

```
Please input an integer bwteen 1 and 99999:4567
4567 is a four digit number.
```

```
Please input an integer bwteen 1 and 99999:23456
23456 is a five digit number.
```

Example: The Richter scale is a measurement of the strength of an earthquake.

Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

In this case, there are five branches: one each for the four descriptions of damage, and one for no destruction.

Use **if-else-if** statement

lab05_examples > earthquake.cpp > ...

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

int main()
{
    float richter;

    cout << "Please input the richter:";
    cin >> richter;

    if(richter >= 8.0)
        cout << "Most structures fall." << endl;
    else if(richter >= 7.0)
        cout << "Many building destroyed." << endl;
    else if(richter >= 6.0)
        cout << "Many building considerably damaged, some collapse." << endl;
    else if(richter >= 4.5)
        cout << "Damage to poorly constructed buildings" << endl;
    else
        cout << "No destruction of buildings." << endl;

    return 0;
}
```

How about this program? If the input is 7, what is the output?

lab05_examples > earthquake2.cpp > ...

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

int main()
{
    float richter;

    cout << "Please input the richter:";
    cin >> richter;

    if(richter >= 4.5)
        cout << "Damage to poorly constructed buildings." << endl;
    else if(richter >= 6.0)
        cout << "Many building considerably damaged, some collapse." << endl;
    else if(richter >= 7.0)
        cout << "Many building destroyed." << endl;
    else if(richter >= 8.0)
        cout << "Most structures fall." << endl;
    else
        cout << "No destruction of buildings." << endl;

    return 0;
}
```

If 7 is input to the program, what it prints?

lab05_examples > earthquake3.cpp > ...

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

int main()
{
    float richter;

    cout << "Please input the richter:";
    cin >> richter;

    if(richter >= 8.0)
        cout << "Most structures fall." << endl;
    if(richter >= 7.0)
        cout << "Many building destroyed." << endl;
    if(richter >= 6.0)
        cout << "Many building considerably damaged, some collapse." << endl;
    if(richter >= 4.5)
        cout << "Damage to poorly constructed buildings" << endl;

    return 0;
}
```

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;
}
```

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

```
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 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");
```

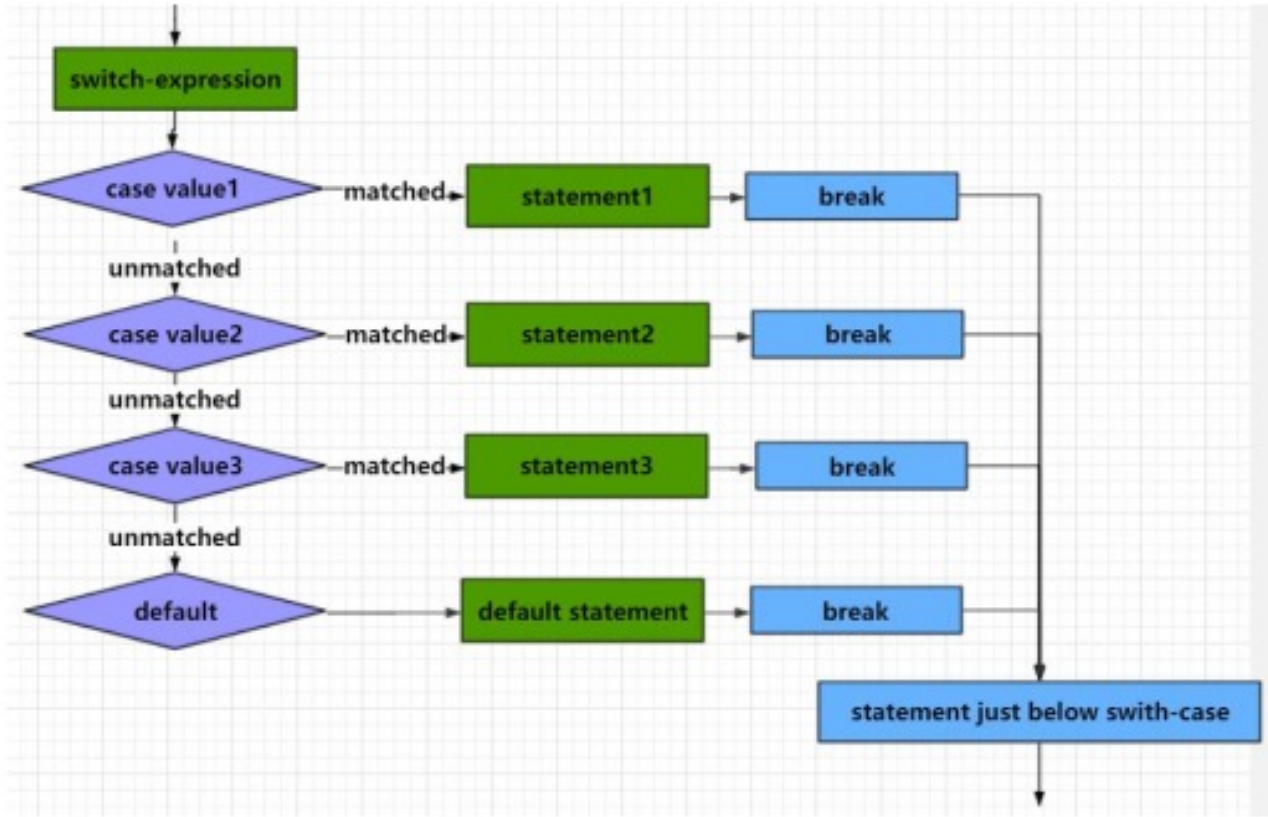
The complier ignores all indentation and matches the else with the preceding **if**.

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");
```

5. The switch statement

```
// The switch-expression must yield a value
// of char,byte,short,int,or String type
switch (switch-expression)
{
  case value1:
    //execute statement1
    statement1;
    break;
  case value2:
    //execute statement1
    statement2;
    break;
  ...
  case valueN:
   //execute statementN
    statementN;
    break;
  default:
    //execute statementDefault
    statementDefault;
}
```



lab05_examples > switchbranch.cpp > ...

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

int main()
{
    int num;

    cout << "Enter an integer between 1 and 3:";
    cin >> num;

    switch(num)
    {
        case 1: cout << "Case 1." << endl;
                break;
        case 2: cout << "Case 2." << endl;
                break;
        case 3: cout << "Case 3." << endl;
                break;
        default: cout << "Default." << endl;
    }

    return 0;
}
```

```
Enter an integer between 1 and 3:1
Case 1.
```

```
Enter an integer between 1 and 3:2
Case 2.
```

```
Enter an integer between 1 and 3:3
Case 3.
```

```
Enter an integer between 1 and 3:5
Default.
```

If there is no break in the switch statement, what will happen?

lab05_examples > switchbranch.cpp > ...

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

int main()
{
    int num;

    cout << "Enter an integer between 1 and 3:";
    cin >> num;

    switch(num)
    {
        case 1: cout << "Case 1." << endl;

        case 2: cout << "Case 2." << endl;

        case 3: cout << "Case 3." << endl;

        default: cout << "Default." << endl;
    }

    return 0;
}
```

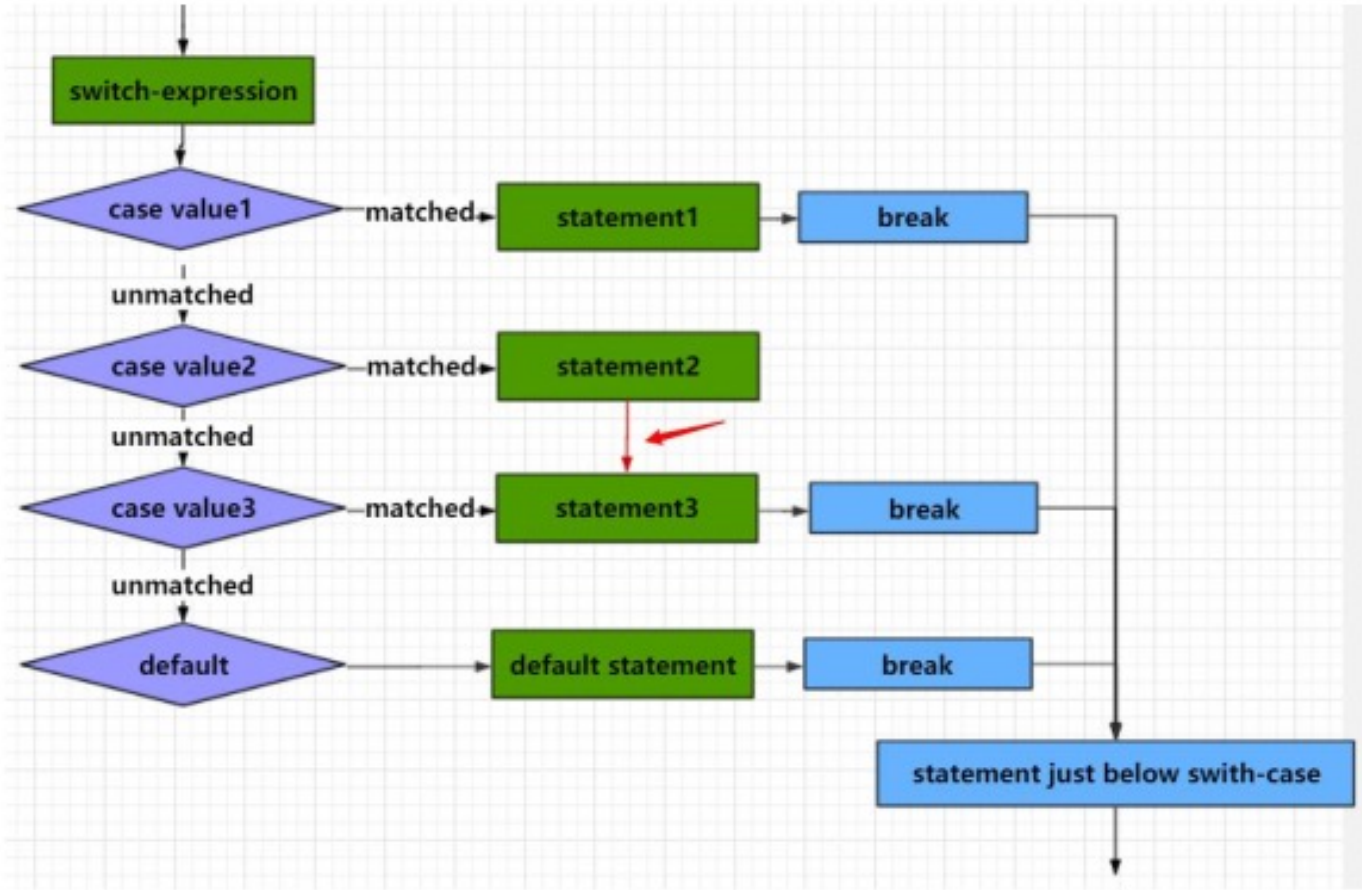
```
Enter an integer between 1 and 3:1
Case 1.
Case 2.
Case 3.
Default.
```

If the **break** statement is omitted, the output will not exit the switch

Note: switch case statement is mostly used with **break** statement.

```
switch (switch-expression)
{
  case value1:
    //execute statement1
    statement1;
    break;
  case value2:
    //execute statement1
    statement2;
    //break;
  ...
  case valueN:
   //execute statementN
    statementN;
    break;
  default:
    //execute statementDefault
    statementDefault;
}
```

If the break statement is omitted



Difference between if and switch

Piece #1

```
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;
}
```

Piece #2

```
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;
}
```

Piece #3

```
switch(opt){
    case 1:
        //add
        result = number1+number2;
        break;
    case 2:
        //sub
        result = number1-number2;
        break;
    case 3:
        //multiply
        result = number1*number2;
        break;
    case 4:
        //divide
        result = number1/number2;
        break;
    default:
        printf("The operator must be one of 1,2,3, and 4\n");
        return; //退出
}
```

use switch if you have three or more alternatives

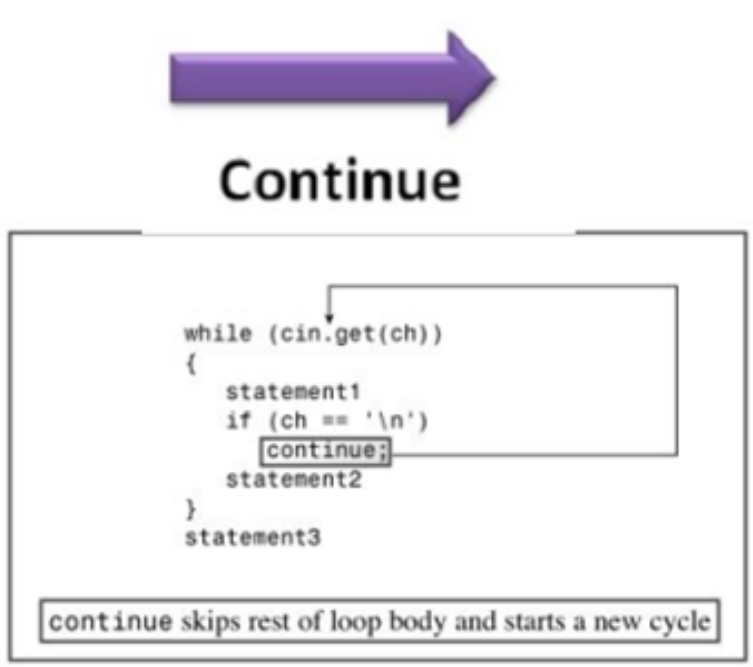
Difference between if and switch

- 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**.

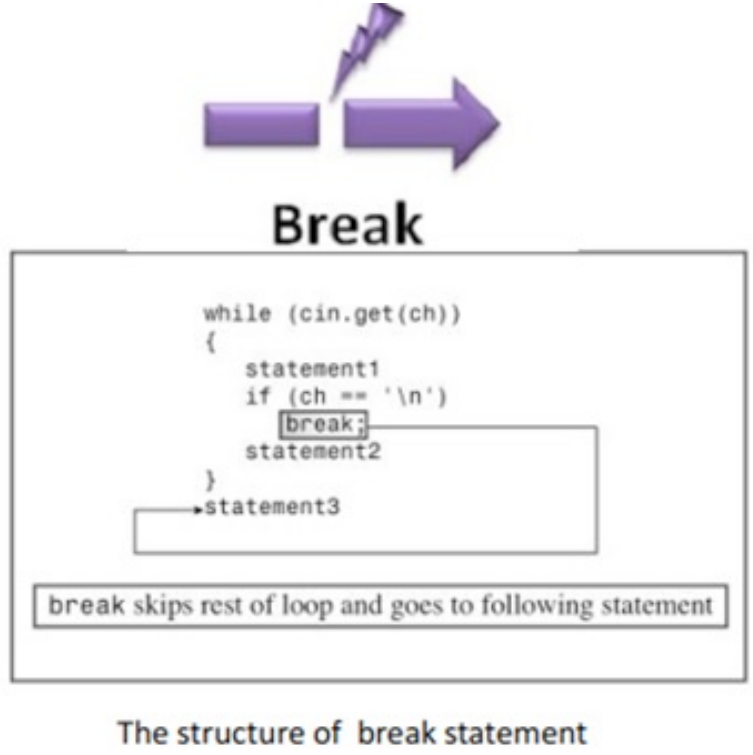
- switch case is faster than if-else: When the number of branches is large (generally larger than 5), **switch-case** is faster than **if-else-if**.

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

2.5 Difference between continue and break



The structure of continue 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;
outClientFile.open("clients.txt", ios::out);
```

Create an ofstream object

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

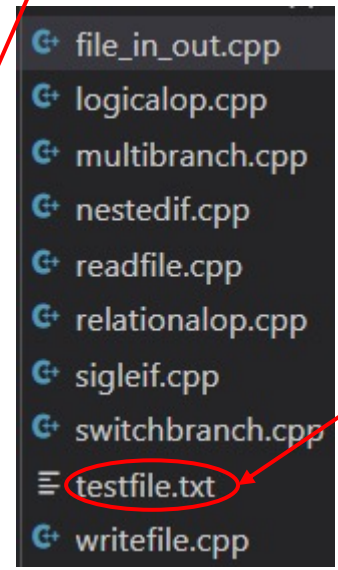
testfile.txt must be existed, otherwise, the filecannot be opened.

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
```

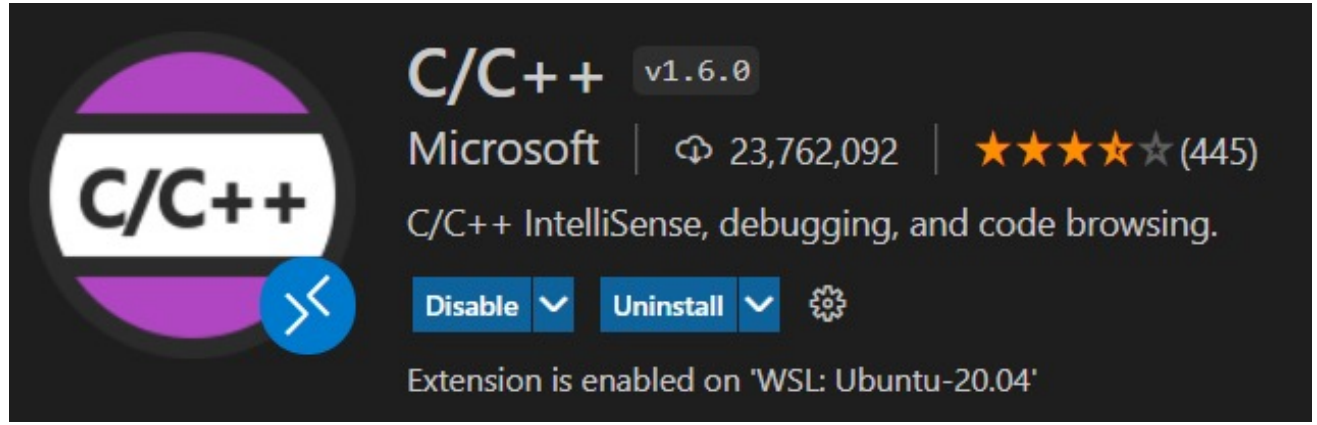
The contents of the file:

lab05_examples > testfile.txt

```
1   Alice Smith
2   19
3
```

Debugging with VSCode

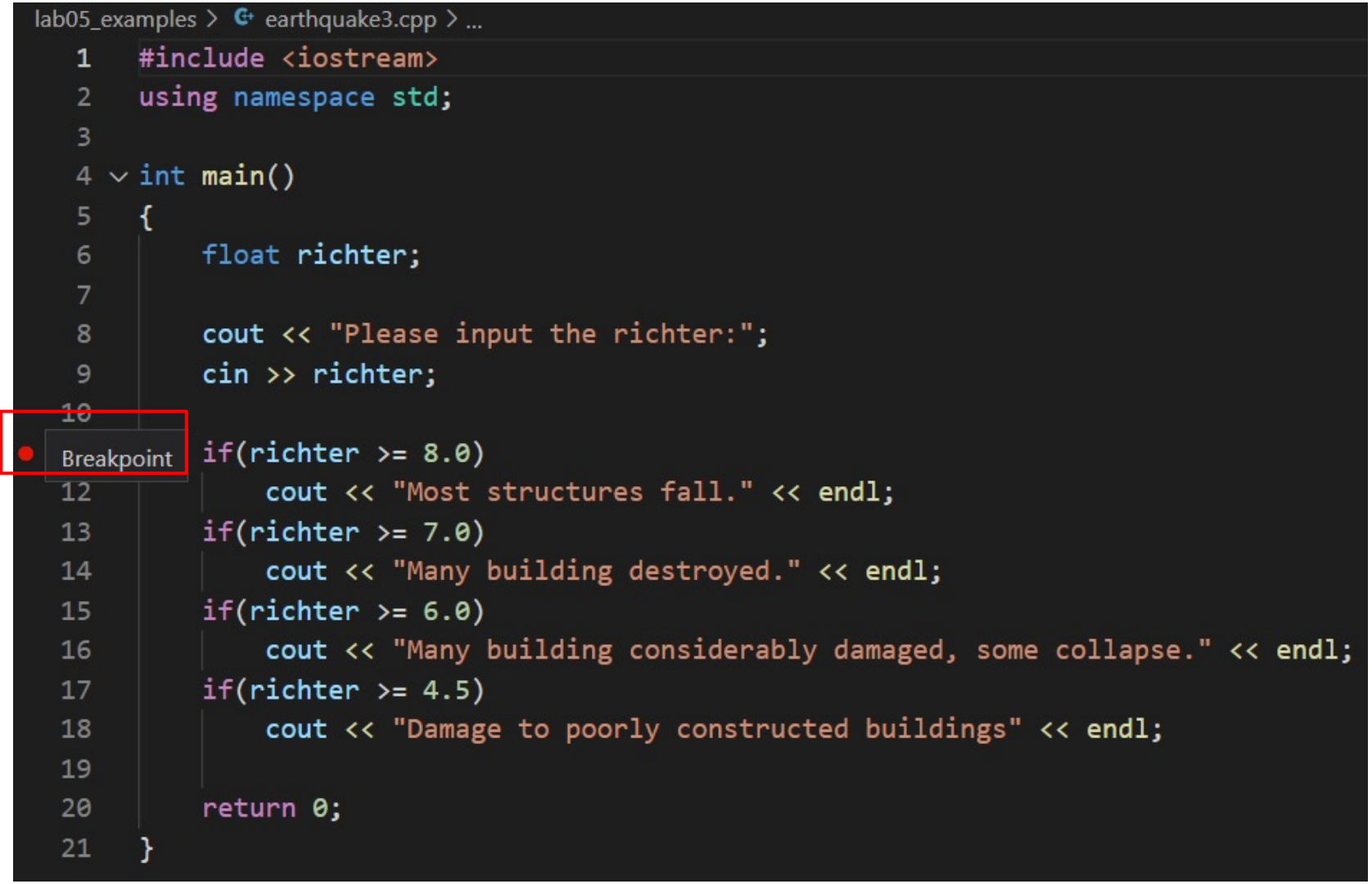
- VS Code's built-in debugger helps accelerate your edit, compile, and debug loop.
- Suppose you have probably installed the C/C++ extension, if not, install it:



Debugging with VSCode

- Open the program you want to debug, and set breakpoint with clicking on the left edge of the code at the certain line.

Set a breakpoint at line 11

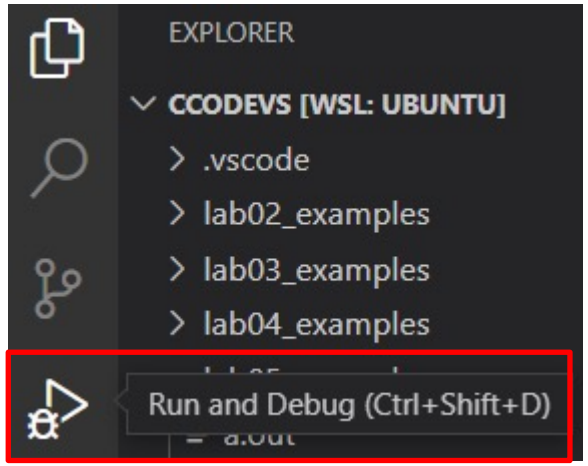


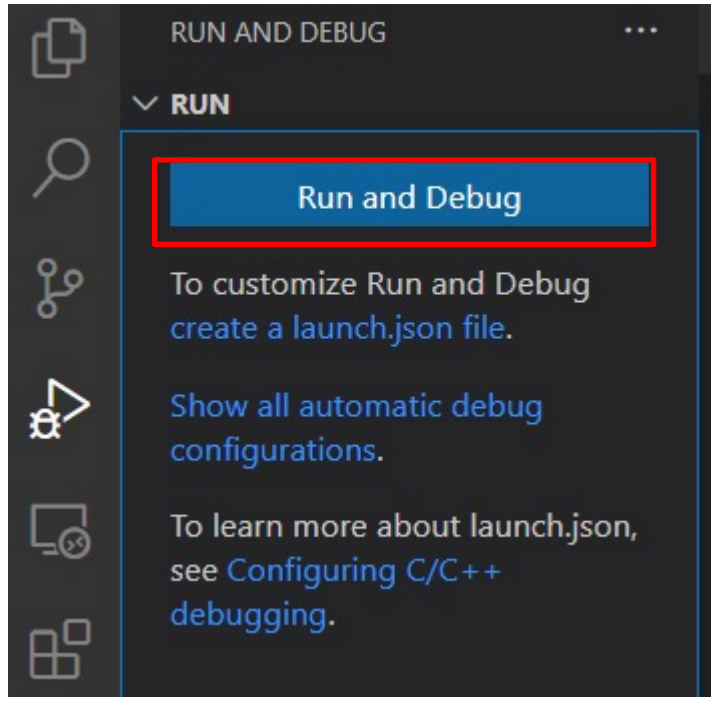

```cpp
#include <iostream>
using namespace std;

int main()
{
    float richter;

    cout << "Please input the richter:";
    cin >> richter;

    if(richter >= 8.0)
        cout << "Most structures fall." << endl;
    if(richter >= 7.0)
        cout << "Many building destroyed." << endl;
    if(richter >= 6.0)
        cout << "Many building considerably damaged, some collapse." << endl;
    if(richter >= 4.5)
        cout << "Damage to poorly constructed buildings" << endl;

    return 0;
}
```

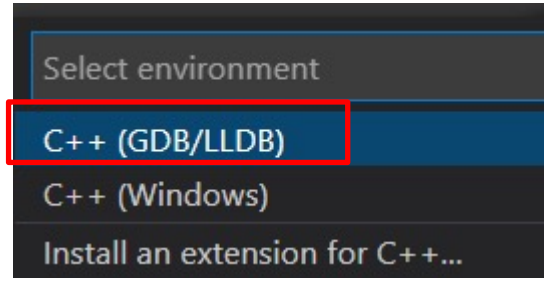
Note: Clicking the breakpoint
Again can remove the breakpoint.

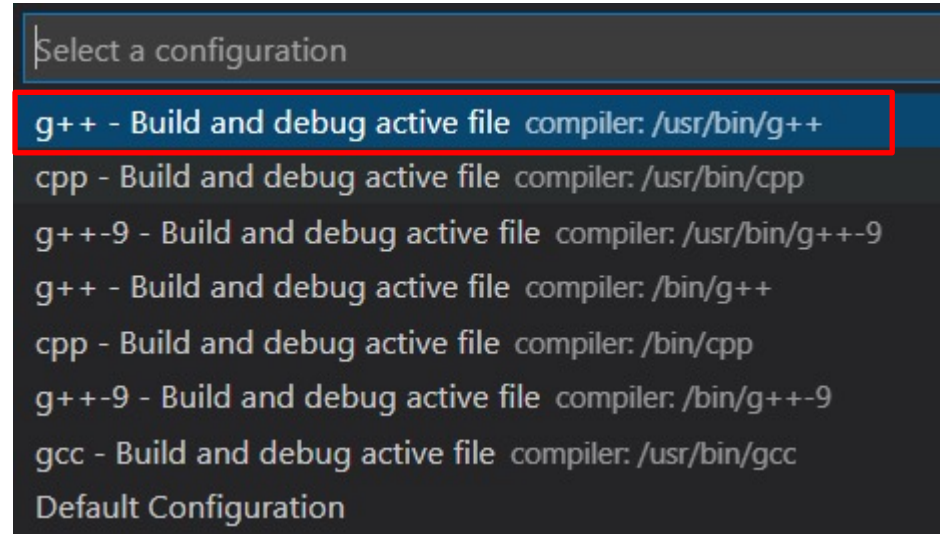
Debugging with VSCode

- Click the “Run and Debug” button on the left and choose the proper items.

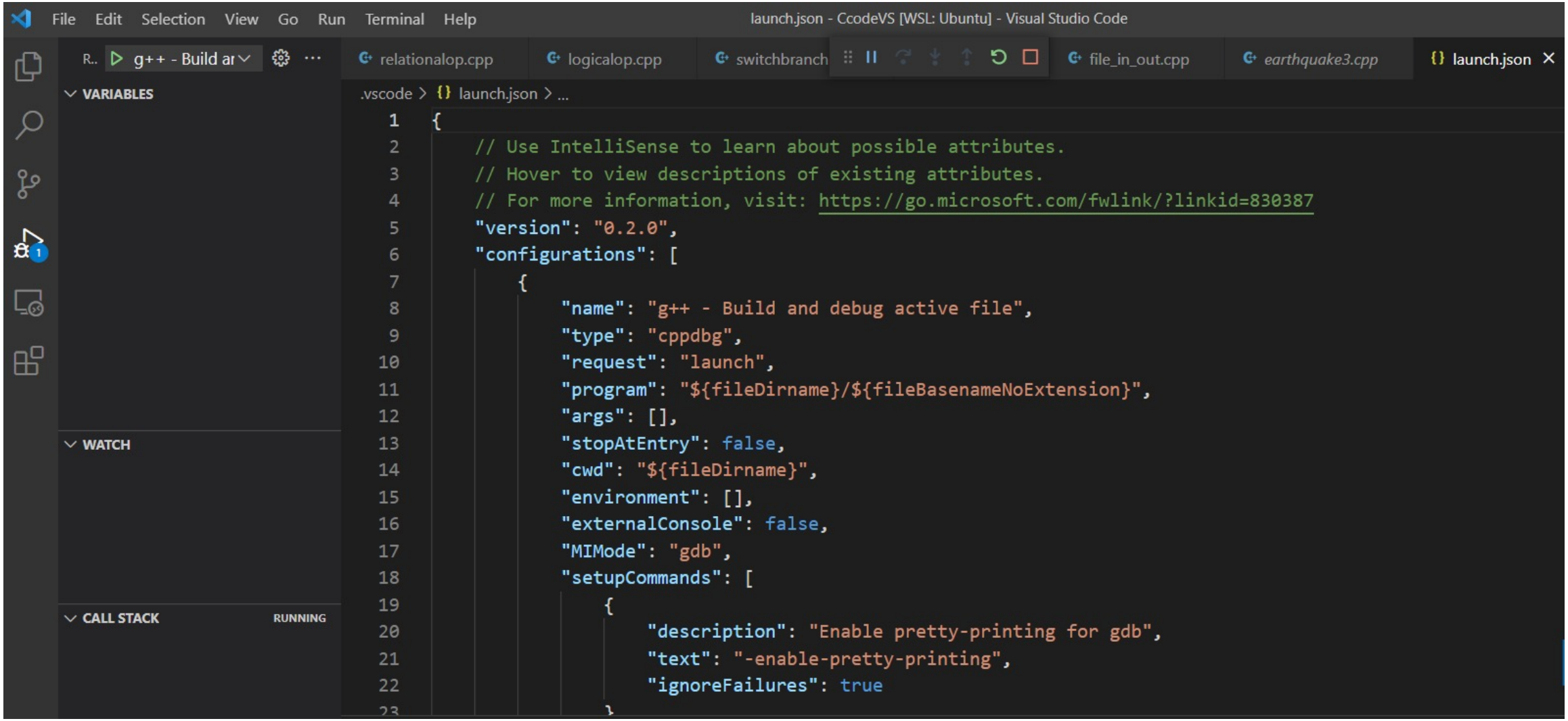


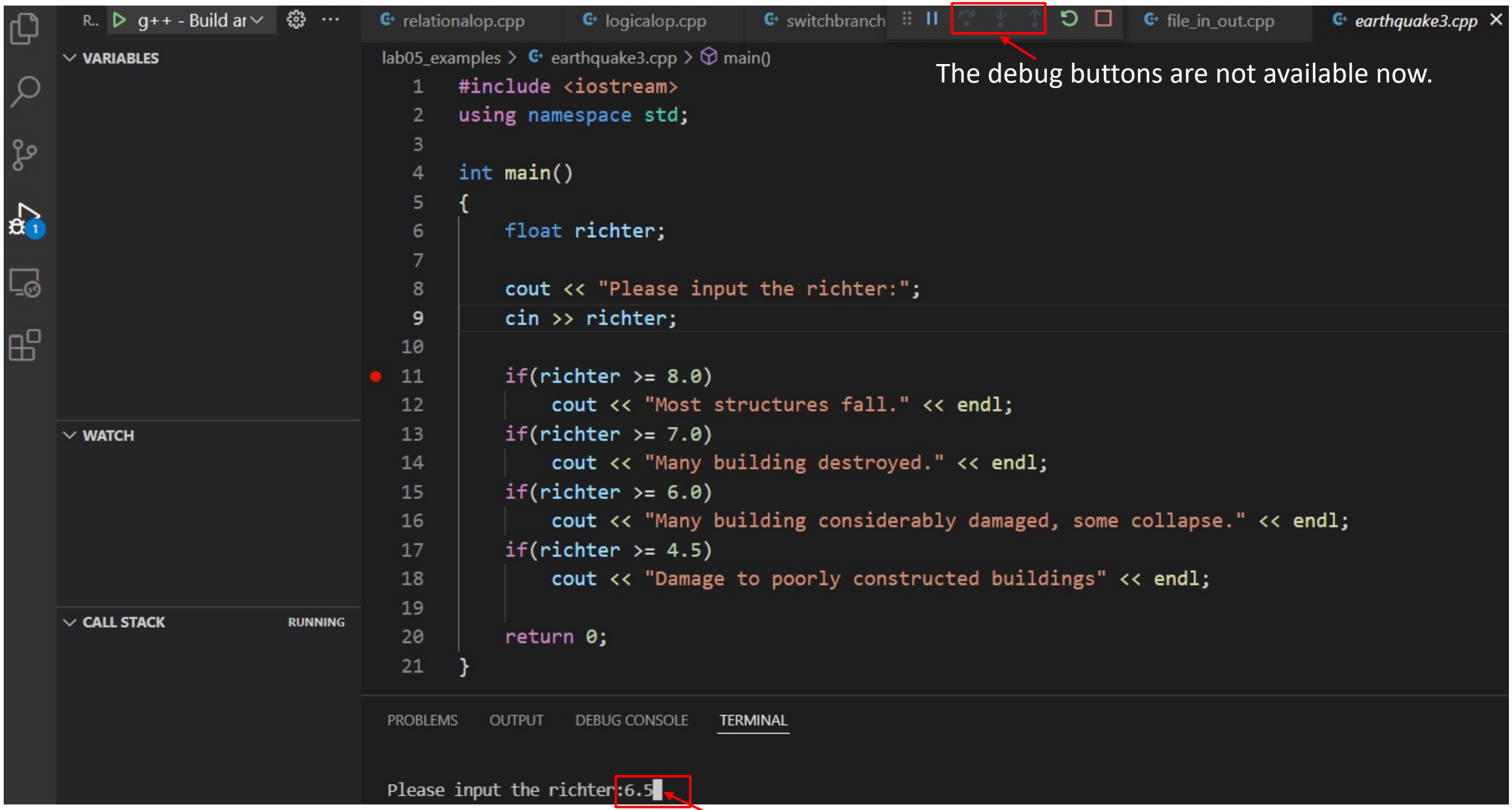


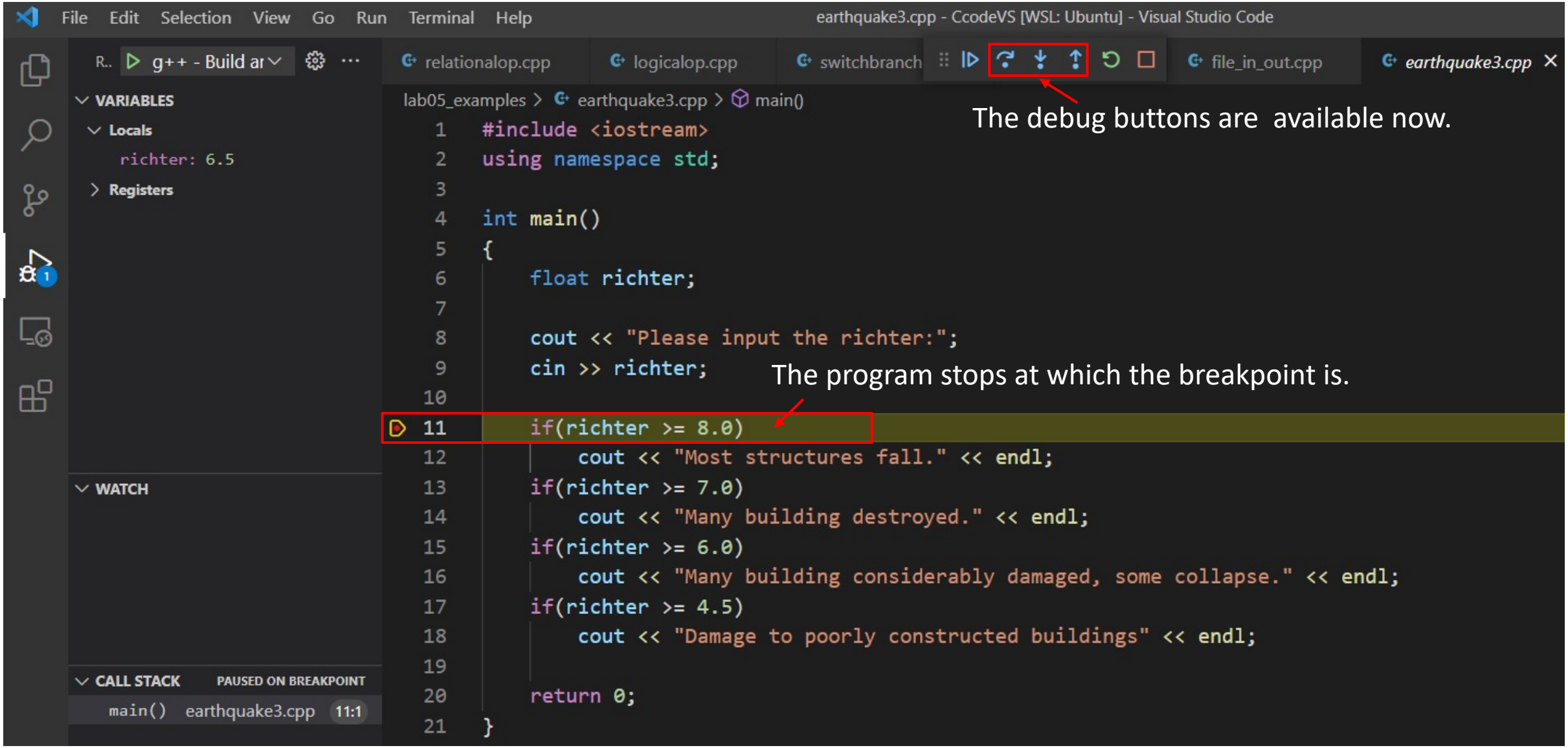


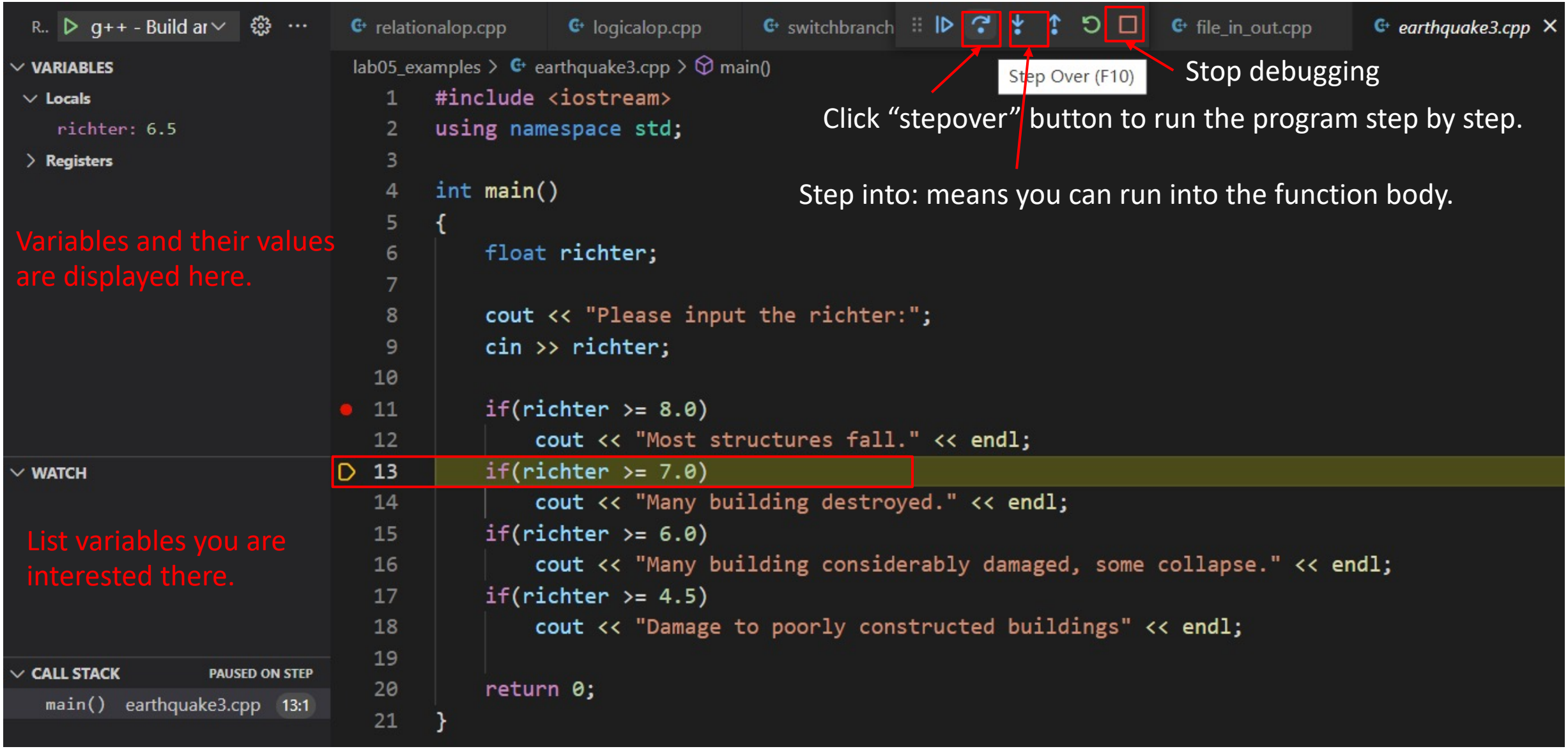


Maybe VSCode could open the launch.json file, if you don't want modify it, close it and switch to the program you want to debug.




R.. g++ - Build ar
relationalop.cpp
logicalop.cpp
switchbranch
file_in_out.cpp
earthquake3.cpp
VARIABLES
WATCH
CALL STACK
RUNNING
lab05_examples > earthquake3.cpp > main()
The debug buttons are not available now.
#include <iostream>
using namespace std;
int main()
{
float richter;
cout << "Please input the richter:";
cin >> richter;
if(richter >= 8.0)
cout << "Most structures fall." << endl;
if(richter >= 7.0)
cout << "Many building destroyed." << endl;
if(richter >= 6.0)
cout << "Many building considerably damaged, some collapse." << endl;
if(richter >= 4.5)
cout << "Damage to poorly constructed buildings" << endl;
return 0;
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Please input the richter:6.5
Input a numeric value and press Enter key


File Edit Selection View Go Run Terminal Help
earthquake3.cpp - CcodeVS [WSL: Ubuntu] - Visual Studio Code
g++ - Build ar
relationalop.cpp
logicalop.cpp
switchbranch
file_in_out.cpp
earthquake3.cpp
VARIABLES
Locals
richter: 6.5
Registers
WATCH
CALL STACK
PAUSED ON BREAKPOINT
main() earthquake3.cpp 11:1
lab05_examples > earthquake3.cpp > main()
The debug buttons are available now.
The program stops at which the breakpoint is.
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6 float richter;
7
8 cout << "Please input the richter:";
9 cin >> richter;
10
11 if(richter >= 8.0)
12 cout << "Most structures fall." << endl;
13 if(richter >= 7.0)
14 cout << "Many building destroyed." << endl;
15 if(richter >= 6.0)
16 cout << "Many building considerably damaged, some collapse." << endl;
17 if(richter >= 4.5)
18 cout << "Damage to poorly constructed buildings" << endl;
19
20 return 0;
21 }



https://code.visualstudio.com/Docs/editor/debugging

3 Exercises

1. Write a program that asks the user to type in numbers. After each entry, the program should report the cumulative sum of the entries to date. The program should terminate when the user enters 0.

Sample output:

```
Enter an integer number:3
The cumulative sum of the entries to date is :3
Enter an integer number:8
The cumulative sum of the entries to date is :11
Enter an integer number:21
The cumulative sum of the entries to date is :32
Enter an integer number:46
The cumulative sum of the entries to date is :78
Enter an integer number:0
The cumulative sum of the entries to date is :78
```

2. 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):
one two three four fine
always happy with done for sure
You entered a total of 8 words.
```

3. 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. One approach is to use **isalpha()** to discriminate between words beginning with letters and those that don't and then use an **if** or **switch** statement to further identify those passing the **isalpha()** test that begin with vowels. A sample run might look like this:

```
Enter word (q to quit):
The 12 unverisities ocean ambled
quietly Example 15 meters of lawn. q
5 words beginning with vowels
4 words beginning with consonants
2 others
```

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

const int vn = 5;
const char vowel[vn] = {'a','e','i','o','u'};

int main() {
    cout << "Enter word (q to quit):" << endl;
    int nVowel = 0;
    int nConsonant = 0;
    int nOther = 0;
    char word[30];


    while(cin >> word)
    {
        if(isalpha(word[0]))
        {
            if(strlen(word) == 1 && word[0] == 'q')
                break;

            char x = tolower(word[0]);

            // complete code here

        }

        // complete code herer
    }

    cout << nVowel << " words beginning with vowels" << endl
    << nConsonant << " words beginning with consonats" << endl
    << nOther << " others" << endl;

    return 0;
}
```

4. 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
```