Reference: Big C++.

Exercises 3:

Exercise R3.1.

1. if **(**quarters > 0**)**

cout << quarters << " quarters"; // delete the word “**then**”

1. if (1 + x > pow(x, sqrt(2))**)**

y = y + x;

1. if (x **==** 1)

y++;

else if (x **==** 2)

y = y + 2;

1. if (x **== 0** && y == 0)

cwin << Point(0, 0);

1. if (1 <= x **&& x** <= 10)

cout << "Enter y: ";

cin >> y;

1. if (s != "nick" **&&** s != "penn"

**&&** s != "dime" **&&** s != "quar")

cout << "Input error!";

1. if (input == "N" **|| input ==** "NO")

return 0;

1. cin >> x;

if (**!**cin.fail())

y = y + x;

OR

cin >> x;

if (**cin**)

y = y + x;

1. language = “English”;

if (country == “USA”)

if (state == “PR”)

language = “Spanish”;

else if (Country **==** “China”)

language = “Chinise”;

Exercise R3.2.

Because of the round-off errors in floating-point numbers in c++, for instance, floating-point number 2 is not exactly the integer 2.

How is that?

The floating-point number 2 may be equals to 2.00000000000004.

int n;

if (n == 10)

…..

\*\*\*\*\*\*

double x;

double epsilon = pow(10, -14);

if (x – 10 <= epsilon)

…..

Exercise R3.3.

1. "Tom" comes after "Dick"
2. "Tom" comes before "Tomato"
3. "Churchill" comes before "church"
4. "car manufacturer" comes before "carburetor"
5. "Harry" comes before "hairy"
6. "C++" comes before "Car"
7. "Tom" comes before "Tom"!
8. "Car" comes before "Carl"
9. "bar" comes before "car"

Code:

int main()

{

string firstName;

string lastName;

while (1)

{

getline(cin, firstName);

getline(cin, lastName);

if (firstName < lastName)

cout << "\"" << firstName << "\" comes before \"" << lastName << "\"" << endl;

else

cout << "\"" << lastName << "\" comes before \"" << firstName << "\"" << endl;

}

return 0;

}

Exercise R3.4.

When reading a number in, the stream fails if it was entered a character that is not a number, for instance:

int num;

cin >> num;

if entered the char ‘a’

the stream fails.

The second way for the stream to fail is entering floating-point number to integer variable

int num1;

int num2;

cin >> num1 >> num2;

if I entered

4.25

5

num1 will equal 4

but num2 will not equal 5 because the buffer still has 0.25 in it.

This situation does not appear with strings, because whatever your input was, It will always be considered as an array of characters.

Exercise R3.5.

cout << "Enter the number of quarters: ";

cin >> quarters;

**if (cin.fail()) cout << “Input error.”;**

**else**

**total = total + quarters \* 0.25;**

Exercise R3.6.

1. 15.9

quarters = 15

Stream will not fail in this reading but it will fail in the next “integer” reading.

Complete code:

cout << "Enter the number of quarters: ";

int quarters;

int next;

cin >> quarters;

cout << quarters << endl;

cin >> next;

cout << next; // next = random number.

However, if the next reading was for a double variable, the stream would not fail.

Complete code:

cout << "Enter the number of quarters: ";

int quarters;

double next;

cin >> quarters;

cout << quarters << endl;

cin >> next;

cout << next; //next = the floating-point part of the number. i.e. = 0.9

1. 15 9

quarters = 15

Stream will not fail in this reading or the next.

1. +159

quarters = 159

Stream will not fail.

1. -159

quarters = -159

Stream will not fail

1. Fifteen

Stream fails

1. –Fifteen

Stream fails

1. + 15

quarters = random number. Because of the sign is followed by a space.

Stream fails

1. 1.5E3 or 1.5e3

quarters = 1

Stream will not fail in this or in the next reading.

1. +1+5

quarters = 1

Stream will not fail in this or in the next reading.

Exercise R3.7.

if / else if / else

is a statement in which the first condition is verified and checked, the code under it will be executed.

And just one statement only will be executed, although if there was two conditions true.

Example:

If (condition 1)

do something;

else if (condition 2)

do another something;

else // i.e. if condition 1 **and** condition 2 are not true.

do something else.

If the two first conditions were true, then the first statement is only executed and the compiler ignores the rest of the statements.

Nested if statements:

is a statement in which if the condition is true then the compiler moves to the next layer.

For instance:

if (condition 1)

{

if ( condition 2)

do something;

else

do something else;

}

If the first condition is false, then it ignores the if statement and its else statement inside the first if statement.

If the condition is true, then it checks if the second condition is true or not, if true it will execute that “something”. If not it will execute that “something else”.

Exercise R3.8.

Example for an if/else/else statement where the order of the tests does not matter:

if (x > 0)

std::cout << “Positive number.\n”;

else if (x < 0)

std::cout << “Negative number.\n”;

else

std::cout << “Zero\n”;

In this example the order doesn’t matter, because of that the three conditions are different, and this means that one and only one condition is true and one statement is executed.

Example for an if/else/else statement where the order of the tests does matter:

if (x > 50)

std::cout << “Greater than 50.\n”;

else if (x > 20)

std::cout << “Greater than 20.\n”;

else

std::cout << “Less than or equal 20\n”;

Here, the order does matter. Why? Because if x equals to 60 then this number is greater than 50 but in the same time it is greater than 20.

Which one will be executed? The one that came first. In this case it is the first if statement.

Exercise R3.10.

True, A && B is the same as B && A for any Boolean conditions A and B.

Exercise R3.11.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p | q | r | !r | (q || !r) | (p && q) || !r | !(p && (q || !r)) |
| false | false | false | true | true | true | true |
| false | false | true | false | false | false | true |
| false | true | false | true | true | true | true |
| false | true | true | false | true | false | true |
| true | false | false | true | true | true | false |
| true | false | true | false | false | false | true |
| true | true | false | true | true | true | false |
| true | true | true | false | true | true | false |

Exercise R3.12.

s = 0;

if (x > 0) s++;

if (y > 0) s++;

this means that if x > 0, s will increase by one. And if y > 0 ,s also will increase again.

But,

s = 0;

if (x > 0) s++;

else if (y > 0) s++;

Here only one statement will be executed and it depends of course on the order of the statements.

For example if x > 0, then s will increase and the second condition (y > 0) will not even be executed.

Exercise R3.13.

1. !(x > 0 && y > 0)

Solution: (x <= 0 || y <= 0)

1. !(x != 0 || y != 0)

Solution: (x == 0 && y == 0)

1. !(country == "USA" && state != "HI" && state != "AK")

Solution: (country != “USA” || state == “HI” || state == “AK”)

1. !(x % 4 != 0 || !(x % 100 == 0 && x % 400 != 0))

Solution: (x % 4 == 0 && x % 100 == 0 && x % 400 != 0)

Exercise R3.14.

double GPA;

std::cout << “Enter your GPA: “;

std::cin >> GPA;

if (GPA < 2)

if (GPA > 1.5)

std::cout << “Passed\n”;

else

std::cout << ”Failed\n”;

the else statement here belongs to the second if.

Exercise R3.15.

An infinite loop is the loop in which the condition is always true, so it keeps iterating.

By ctrl + c.

Exercise R3.16.

Off-by-one error is the error which happen when the programmer does not know the exact start (initialization) of some variables.

For example, when calculating the sum of some variables, Should sum start with zero or one? The product of those variables, Should it start with zero or one!

Exercise R3.17.

Sentinel value is a value that indicates the end of some inputs, without being part of that input.

Rules to use sentinel instead of the end of a file eof:

* If the sentinel is not of the same type of the input, for example use a number sentinel for some string inputs, but do not use a number as a sentinel for some of number inputs.
* Use eof for files not a sentinel.

Exercise R3.18.

A loop and a half is a loop in which the terminating condition is in the half of the loop, so one should go halfway to find out if he should terminate or not.

The first loop:

bool more = true;

loop (while more == true)

read employee name

if not OK, more = false

read employee salary

if not OK, more = false

give employee 5 percent raise

print employee data

The second loop:

Loop (while true)

read employee name

if not OK, break;

read employee salary

if not OK, break;

give employee 5 percent raise

print employee data

Exercise R3.21.

1. for (int i = 0; i <= 10; i++) …

Iteration count: (10 – 0) / 1 + 1 = 10 + 1 = 11.

1. for (int i = 0; i < 10; i++) …

Iteration count: (10 – 0) / 1= 10.

1. for (int i = 10; i > 0; i--) ...

Iteration count: (0 – 10) / -1 = 10.

1. for (int i = -10; i <= 10; i++) ...

Iteration count: (10 - -10) / 1 + 1 = 21

1. for (int i = 10; i >= 0; i++) ...

Iteration count: (0 – 10) / 1 + 1 = -9

This means that i will always be larger than zero so it is an infinite loop.

1. for (int i = -10; i <= 10; i = i + 2) ...

Iteration count: (10 - -10) / 2 + 1 = 11

1. for (int i = -10; i <= 10; i = i + 3) ...

Iteration count: (10 - -10) / 3 + 1 = 7 times only

Exercise R3.22.

int i = 1, s = 0;

while ( i <= 10)

{

s = s + i;

i++;

}

Exercise R3.23.

int n;

std::cin >> n;

double x = 0;

double s = 1.0 / (1 + n \* n);

while (s > 0.01)

{

s = 1.0 / (1 + n \* n);

n++;

x = x + s;

}

Exercise R3.24.

1. s = 11
2. s = 11
3. s = 211

Exercise R3.25.

1. 2471116
2. 4916
3. 10 7

Exercise R3.26.

1. 106
2. 3
3. 0