07. `if` and `if-else`: Part 1

Objectives

- Write if statements (including blocks)
- Write if-else statements (including blocks)
- Write nested if-else statements

We will now talk about writing statements that make decisions. In other words, statements that decide whether to run other statements or not.

This is significant … we are now embarking on writing program with intelligence …

ONWARD!!!

`if` statements

So far you know that C++ can do arithmetic, print integer values, print string values, accept integer values from the keyboards, create and update variables.

You should expect more! (Otherwise C++ is pretty dumb.)

For instance you would expect programs to make decisions. Now wouldn't it be nice if C++ can do this:

```
      if alien is killed:
          score = score + 10
```

or

```
      if my ship's energy is 0:
          end the game
```

or

```
      if plane's left engine has failed and
         plane's right engine has failed:
          activate seat ejection!!!
```

Note that for the first case, if the alien is not killed, you do *not* want to execute the statement `score = score + 10`. Otherwise everyone would keeping getting the same score! What kind of dumb game is that?

In fact C++ *does* understand the `if` statement.

Run this program

```
int x = 0;
std::cin >> x;

if (x > 0) std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

Run the program several time, entering different values for x. Note that the statement printing `"spam!"` is controlled by the `if`. However the statement that prints `"eggs"` is not.

For readability, the if statement is written like this:

```
int x = 0;
std::cin >> x;

if (x > 0)
    std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

Big picture: The format of the `if` statement looks like this

```
if ([bool expr])
    [stmt]
```

where [`bool expr`] is a **boolean expression** – this means that it's an expression that can be evaluated to a boolean value – and [`stmt`] is a C++ statement (of course there's a semi-colon at the end). Since we need boolean expressions it's a good time now to recall the following:

`!`	"not"
`&&`	"and"
`||`	"or"
`==`	"is equal to"
`!=`	"is not equal to"
`<`	"is less than"
`<=`	"is less than or equal to"
`>`	"is greater than"
`>=`	"is greater than or equal to"

An example of a boolean expression is

```
[variable] [bool op] [value]
```

where `[variable]` is a variable of int or double or float type, `[bool op]` is one of the above boolean operators and `[value]` is an int or double or float. Don't forget that if there is a type mismatch, C++ will try to type promote one value to match the type of the other. For instance if `x` is an `int` variable and the boolean expression is

```
x < 1.23
```

then C++ will use the `double` of `x` (i.e., `x` is automatically promoted to a `double`):

```
double(x) < 1.23
```

Exercise. Modify the above program so that `"spam!"` is printed when the integer entered is 42. Test your program.

Exercise. Modify the above program so that the program prompts the user for his/her favorite number and print "you don't like 42?!?" if the integer entered is not 42. Here's an execution:

```
what's your favorite number? 42
```

Here's another:

```
what's your favorite number? 1
```

```
you don't like 42?!?
```

Exercise. Write a program that prompts the user for an integer and prints “That's even. Am I smart?” if the integer entered is even. Here's an execution

```
100
That's even. Am I smart?
```

Here's another

```
101
```

Exercise. Write a program that prompts the user for his/her age and prints “You're lying! Think you can fool me???” if the value entered is less than 18. Here's an execution:

```
Enter your age: 16
You're lying! Think you can fool me???
```

Here's another:

```
Enter your age: 100
```

Of course this is another boolean expression:

```
[var1] [bool op] [var2]
```

In this case you're comparing two (integer or double) variables.

Exercise. Write a program that prompts the user for two integer values and prints “First is bigger” if the first integer entered is larger than the second. Here's an execution of the program:

```
1 2
```

Here's another

```
2 -1
First is bigger
```

Here's yet another boolean expression:

```
[expr] [bool op] [value]
```

where `[expr]` is an (integer or double) expression.

Exercise. Write a program that prompts the user for his/her height (in ft) and weight (in lbs) and prints “you will live more than 100 years!!!” if the product of the height and weight is at least 200.45. (This is not supported by any form of research.) Here's an execution of the program:

```
6.1 180.180
```

```
you will live more than 100 years!!!
```

Here's another

```
3.4 23.4
```

Exercise. Write a program that prompts the user for the year of his/her date of birth and the current year and prints his/her approximate age and then prints “you must have watched lots of movies by now” if the difference of the two values is at least 20. Here's an execution:

```
1900 2000
you are about 100 years old
you must have watched lots of movies by now
```

Here's another:

```
2000 2006
you are about 6 years old
```

One last one ... here's one more boolean expression:

```
[expr1] [bool op] [expr2]
```

Now, we're comparing two expressions.

Exercise. Write a program that prompts the user for the month, day of his/her date of birth and also his/her height (in ft) and weight (in lbs). Print the approximate number of days from January 1 of the year he/she was born and the sum of his/her height and weight. If the first printed number is greater than the second, print “According to the theory of relativity, you are going to win the next powerball.”.

```
Enter the month, day of your DOB: 1 1
Enter your height and weight: 6 100
1 106
```

Here's another:

```
Enter the month, day of your DOB: 12 25
Enter your height and weight: 6 200
355 206
According to the theory of relativity, you are going
to win the next powerball.
```

(You may assume the number of days in a month is 30.)

Exercise. Try this:

```
int numHeads = 0;
std::cout << "How many heads do you have? ";
std::cin >> numHeads;
if (numHeads == 2)
    std::cout << "Are you Zaphod?" << std::endl;
```

And then this:

```
int numHeads = 0;
std::cout << "How many heads do you have? ";
std::cin >> numHeads;
if (numHeads != 2)
    std::cout << "You are not Zaphod" << std::endl;
```

Combine both into one program so that I get the following execution:

```
How many heads do you have? 1
You are not Zaphod
```

And here's an execution of the same program:

```
How many heads do you have? 2
Are you Zaphod?
```

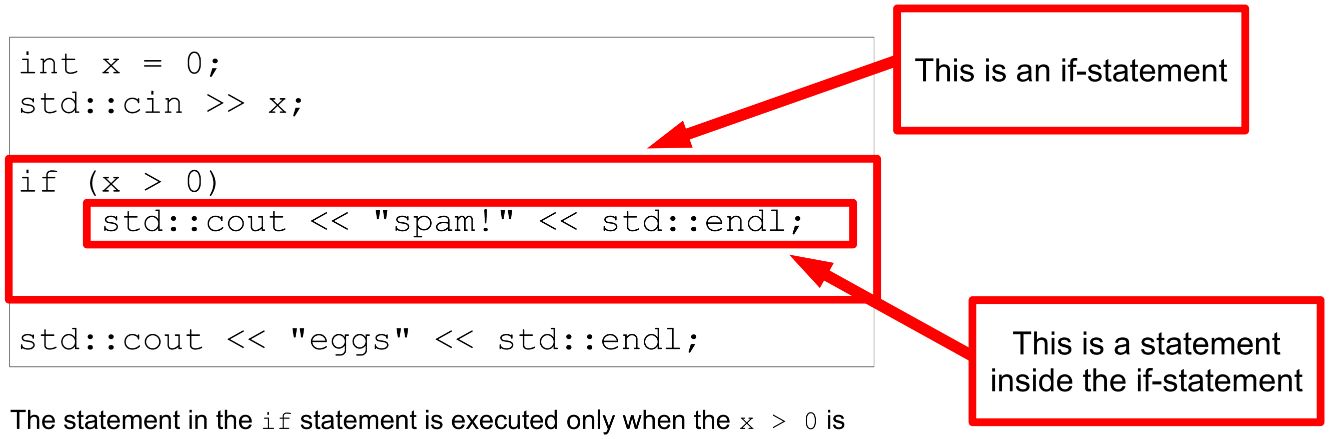
Next modify the program so that, in addition to the above requirements, if the number of heads entered is greater than 2, the program prints "No kidding!". Finally, modify the program so that, in addition to the above, if the number of heads is 1, the program prints "Let me introduce you to our surgeons." Test your program by entering 0, 1, 2, 3.
(There's a better way to do this exercise ...)

Mental picture: flow of execution for `if`

The `if` statement is difficult for some beginning programmers because it alters the flow of execution of a program.

Before this set of notes, C++ executes one statement at a time from top to bottom; it executes every statement.

The `if` statement is different because it has a statement **inside** the `if` statement.



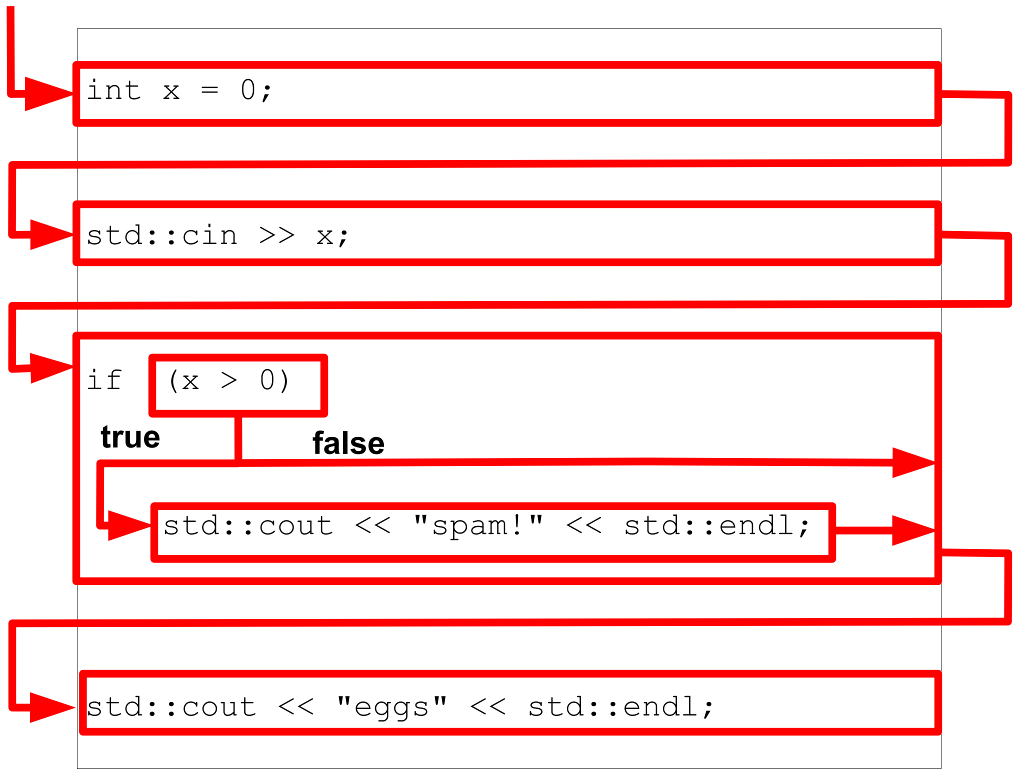
```
int x = 0;
std::cin >> x;

if (x > 0)
    std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

The statement in the `if` statement is executed only when the `x > 0` is true. If the condition is not true, the statement in the `if` statement is not executed.

In the following diagram, you can follow the arrows to get a sense of the flow of execution:



```
int x = 0;

std::cin >> x;

if (x > 0)
    std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

Of course the program is still executing one statement at a time from the top to the bottom if you view the if statement as a statement (i.e. forget about the internals of the if statement).

Gotchas

Exercise. Try this:

```
int x = 0;
std::cin >> x;

if x > 0
    std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

Fix it.

Exercise. Try this:

```
int x = 0;
std::cin >> x;

if (x > 0);
    std::cout << "spam!" << std::endl;

std::cout << "eggs" << std::endl;
```

What do you learn from this example?

Exercise. Here's another wacky example ...

```
int x = 0;
std::cin >> x;

if (x > 0);

std::cout << "eggs" << std::endl;
```

Multiplication game: Part 1

Exercise. Write a program that tests if the user can do two digit multiplication. Right now we will just ask the one question: the product of 97 and 94. The program congratulates the user if the right answer is entered. Otherwise the program stops immediately. Here's an execution of the program:

```
What is the product of 97 and 94? 1
```

Here's another execution of the program:

```
What is the product of 97 and 94? 9118
You smart dawg!
```

This is kind of a dumb program because it asks the same question again and again.

Here's the **pseudocode** (pseudocode = informal description of steps in a program):

```
Declare integer variable guess
Print a prompt string to user
Prompt user for integer value for guess
If guess is 9118
      print congratulatory message
```

Note that this is not a C++ program yet!!! You can't run it!!!

Complete the program using the above pseudocode. Test your program.

Blocks

What if you want to execute ***several*** statements when a boolean expression is true? Here's one way:

```
int x = 0;
std::cin >> x;

if (x > 0)
    std::cout << "spam!" << std::endl;
if (x > 0)
    std::cout << "ham!" << std::endl;

std::cout << "eggs" << std::endl;
```

It does work … but ... that's the BAD way of executing two statements when a condition holds.

This is the right way:

```
int x = 0;
std::cin >> x;

if (x > 0)
{
    std::cout << "spam!" << std::endl;
    std::cout << "ham!" << std::endl;
}

std::cout << "eggs" << std::endl;
```

The `{...}` is called a **block** or **block of statements.** (You've actually seen this already. Look at your `main()` …) You can have as many statements as you like in a block, not just two.

Exercise. Here's part of a game that you're writing … complete it. If the missile hits the alien, increment the `score` by 100, set `play_explosion` to `true`, increment your `energy` by the alien's energy (i.e. `alien_energy`), and set the alien's energy to 0. Test your code with different inputs.

```
int score = 1234;
int energy = 6789;
int alien_energy = 42;
bool play_explosion = false;
bool missile_hit_alien;
std::cin >> missile_hit_alien;
```

```
std::cout << "score: " << score << '\n'
          << "energy: " << energy << '\n'
          << "alien_energy: " << alien_energy << '\n'
          << "play_explosion: " << score << '\n';
```

Random integers

Generating a random integer occurs commonly in computer programs. For instance if you want to write a strategy role-playing game, you might have different rooms in a mansion and you want to put different things in different rooms. Suppose you have a magic potion in a bottle. Now suppose the rooms are numbered from 1 to 1000. You don't want your magic potion bottle to be in room 5 whenever you start the game. You want each game to be different.

At a more complex level you might want to randomly make doors between two rooms. This creates a random maze.

So you'd better learn how to generate random things.

Try this:

```
#include <iostream>

int main()
{
  std::cout << "the potion is in room " << rand()
            << std::endl;
  std::cout << "the potion is in room " << rand()
            << std::endl;
  std::cout << "the potion is in room " << rand()
            <<  std::endl;
  std::cout << "total number of rooms: "
            << RAND_MAX << std::endl;

  return 0;
}
```

(Note: You might need `#include <cstdlib>` just after `#include <iostream>`. This depends on your compiler.)

Looks like you're getting random integers. But wait ... run the program a second time ... a third time ... a fourth time. Notice something?

OK. Let's modify the program:

```
#include <iostream>
#include <ctime>

int main()
{
  srand((unsigned int) time(NULL));

  std::cout << "the potion is in room " << rand()
            << std::endl;
  std::cout << "the potion is in room " << rand()
            <<  std::endl;
  std::cout << "the potion is in room " << rand()
```

```
            << std::endl;
  std::cout << "total number of rooms: "
            << RAND_MAX << std::endl;
  return 0;
}
```

Run this program several times. Notice that the room number is different each time you run the program.

I will not go into details on the statement

```
  srand((unsigned int) time(NULL));
```

The only thing I will say is that it "seeds" the random generator so as to make it "more random". The important thing to remember is that you must execute this statement and furthermore you execute it ***<u>once</u>*** (usually) and ***<u>before</u>*** you call the `rand()` function. The standard practice is to execute the seeding in `main()` and at the beginning:

```
#include <iostream>
#include <ctime>

int main()
{
  srand((unsigned int) time(NULL));
  ...

  return 0;
}
```

Since `rand()` gives you a random number from `0` to `RAND_MAX`,

```
      rand() / RAND_MAX
```

will give you a random `double` between `0.0` and `1.0`. But this is not right because the / is an ***<u>integer</u>*** division. Therefore we do this:

```
      double(rand()) / RAND_MAX
```

to get a random double from 0.0 to 1.0.

Exercise. Check that I was not lying by printing 10 random doubles from 1.0 to 0.0. By the way, if the numbers are truly random, you would expect their average to be close to 0.5. Check that too.

There are times when you need to generate a random ***<u>integer</u>*** between two bounds. For instance if you have a maze game and the rooms are numbered 1 to 100, you want a random room number for a magic potion. How many values are there from 1 to 100? You have 100. Since `rand()` gives you a random integer from 0 to `RAND_MAX`, if you do

```
rand() % 100
```

you get 100 numbers, from 0 to 99. You add 1 to it and you get a random integer from 1 to 100. In other words

```
1 + rand() % 100
```

does the trick.

Exercise. How do you generate random integers from 1,..., 6? (For instance to simulate a dice roll.) Verify with a C++ program.

Exercise. How do you generate random integers from 1,..., 10? Verify with a C++ program.

What about generating random integers from 5,..., 20? In this case the the lower bound is 5 and not 1. You use the same idea. Note that if you do something like

```
rand() % 10
```

or

```
rand() % 18
```

the range of numbers starts from 0. So if you want to generate random numbers in the range of 5,..., 20, the first thing you do is to think about the range

0, …, 15

first. In other words move the given range 5, …, 20 so that the lower bound of the range becomes 0. That means subtracting 5 from the range. To generate a number in the range of 0, …, 15, you do

```
rand() % 16
```

This gives a number in the range 0, …, 15. You then add 5 to it:

```
5 + rand() % 16
```

Exercise. Verify that the expression

```
5 + rand() % 16
```

does give you numbers in the range of 5, …, 20. .

Exercise. How do you generate random integers from 9,...142? Verify with a C++ program.

Exercise. How do you generate random integers from -5,...,5? Verify with a C++ program.

Exercise. How do you generate random `double` from 0,...,2.0? Verify with a C++ program.

Exercise. How do you generate random double from -1.0,...,1.0? Verify with a C++ program.

Exercise. How do you generate random double from 0,...,3.5? Verify with a C++ program.

Exercise. How do you generate random double from -2.0,...,2.0? Verify with a C++ program.

The important swap "trick"

Recall the following extremely important "trick" that you must know. It allows you to swap the values in two variables:

```
int a = 42, b = 24;
std::cout << a << " " << b << std::endl;

int t = a;
a = b;
b = t;

std::cout << a << " " << b << std::endl;
```

Exercise. Write a program that declares two doubles, initializing them with 1.234 and 4.567. Print the values of the variables. Swap their values. Print the values of the variables.

First sorting example

Here's your first sorting problem:

- Prompt the user for two integer values and assign them to x and y.
- Sort the values so that the value of x is less than or equal to the value of y

Exercise. Write the above program. Here are some test cases:

Test 1
```
1 2
1 2
```

Test 2
```
2 1
1 2
```

Test 3
```
-1 5
-1 5
```

Test 4
```
5 -1
-1 5
```

Test 5
```
2 2
2 2
```

More sorting examples: bubblesort

Now suppose you have three integer variables and you want to sort the values in the variables. What I mean is this. Suppose your variables are x, y, and z. After some operations you want

```
x <= y && y <= z
```

to be true (NOT `x <= y <= z` !!! Remember???)

Here's the idea. First do this:

- Swap the values of x and y so that x <= y.
- Swap the values of y and z so that y <= z.

This will guarantee that z is the largest.

Exercise. Take a piece of paper. And write down three integer values for x, y, and z and follow the above steps. Do you see that the largest will always be in variable z's box?

Exercise. Does this means that the smallest will be in x?

The above two steps is said to be one ***pass*** of this sorting process. Now do this:

- Swap (if necessary) the values of x and y so that x <= y.

This will guarantee that the larger value between x and y will be in y.

Now everything is in ascending order, i.e.

```
x <= y && y <= z
```

Altogether these are the steps:

Pass 1: To get largest among x, y, z into z do this:
- Swap (if necessary) the values of x and y so that x <= y.
- Swap (if necessary) the values of y and z so that y <= z.

Pass 2: To get largest among x, y into y
- Swap (if necessary) the values of x and y so that x <= y.

Such a “recipe” (a finite number of steps to achieve a goal) is called an **algorithm**. This sorting algorithm is called **bubblesort**. There are actually many different sorting algorithms.

Exercise. Write a program that prompts the user for three integers and prints the three integers in ascending order. Here are some executions:

```
1 2 3
1 2 3
```

```
1 3 2
1 2 3
```

```
2 1 3
1 2 3
```

```
2 3 1
1 2 3
```

```
3 1 2
1 2 3
```

```
3 2 1
1 2 3
```

What if you have 4 values to sort (in ascending order)? Say the values are stored in a, b, c, d. If you understood the above bubblesort for 3 values, the bubblesort for 4 values is similar:

Pass 1: To get largest among a, b, c, d into d do this:
- Swap (if necessary) the values of a and b so that a <= b.
- Swap (if necessary) the values of b and c so that b <= c.
- Swap (if necessary) the values of c and d so that c <= d.

Pass 2: To get largest among a, b, c into c
- Swap (if necessary) the values of a and b so that a <= b.
- Swap (if necessary) the values of b and c so that b <= c.

Pass 3: To get largest among a, b into b
- Swap (if necessary) the values of a and b so that a <= b.

Exercise. Test the above bubblesort for 4 values: Write a program that prompts the user for 4 integers and prints the values in ascending order.

Exercise. What if you have five variables a, b, c, d, e? How would you take the values from a, b, c, d, e sort them into ascending order, and put them into a, b, c, d, e?

Exercise. Write a program that prompts the user for 4 **`doubles`** and prints the `double`s in ascending order.

Exercise. Write a program that prompts the user for 4 integers and prints the integer values in **descending** order. (You need to modify the above bubblesort algorithm.)

Exercise. Now, you have four variables a, b, c, d. You know that one pass of the bubblesort will put the largest value of a, b, c, d into d. What would you do if I ask you to write a program to prompt the user for four values and then print the second largest. Now write the program and test it. [Hint: You need **two** passes.]

Exercise. Again suppose you have four variables a, b, c, d. Suppose during the second pass, you notice that no swap was performed. Do you see that you do not need to perform any more passes?

You must memorize the bubblesort algorithm for any number of variables.