09. switch

Objectives

- Write switch statements
- Write switch statements where cases do not have breaks
- Write switch statements where the default case is missing

The if and if-else statements allow us to decide if we want to execute a statement. It allows us to write programs that can make decisions. There is another way of doing that.

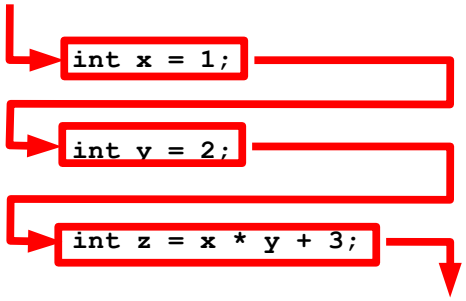
This is the switch statement.

Diagram of flow of execution of `if` statements

Here's the big picture for if.

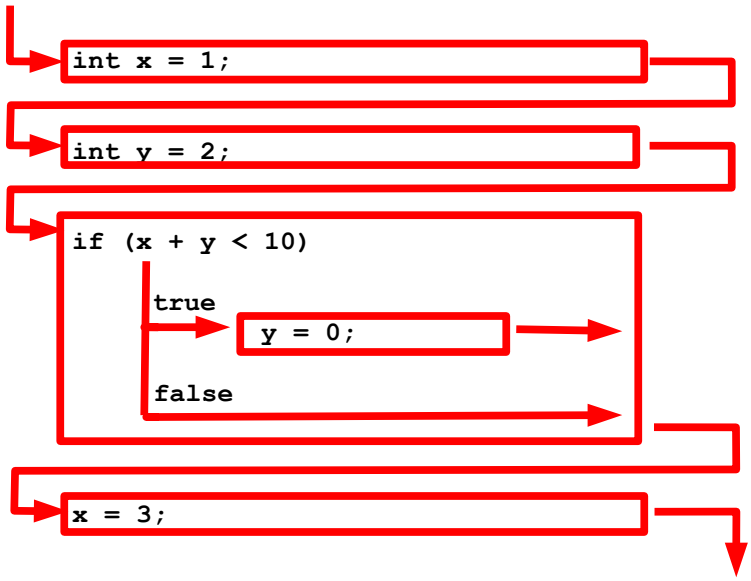
Before the if statement, the code in main executes one statement at a time. For example:

```
int x = 1;
int y = 2;
int z = x * y + 3;
```



Now with the if statement we have:

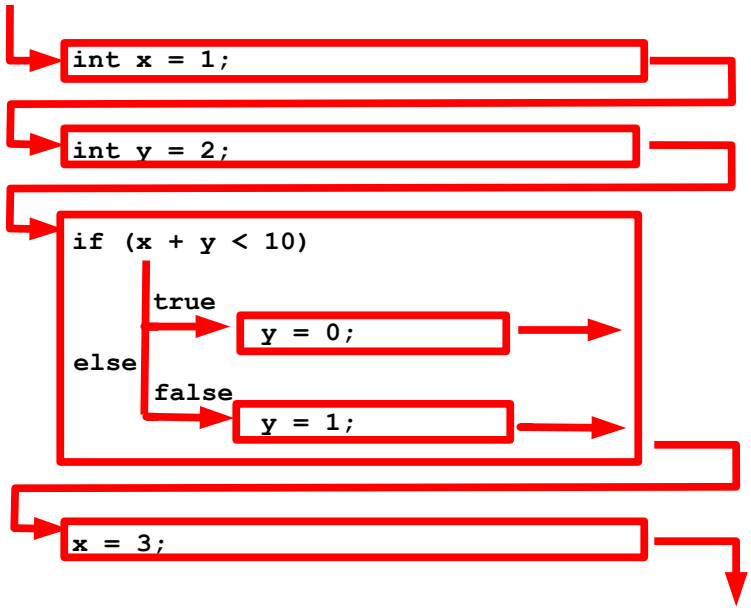
```
int x = 1;
int y = 2;
if (x + y < 10)
   y = 0;
x = 3;
```



Of course it's just as easy to visualize the flow of execution for if-else. If you have:

```
int x = 1;
int y = 2;
int z = 0;
if (x + y < 10)
{
    y = 0;
}
else
{
    y = 1;
}
x = 3;
```

then



Note that the path of execution is determined by the **boolean value** of a boolean expression.

switch

Now I'll show you something that's very similar to the if and if-else statement. Watch out! There are **subtle differences** ....

OK. First run this:

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

if (x == 1)
{
    std::cout << "yo" << std::endl;
}
else if (x == 2)
{
    std::cout << "right ..." << std::endl;
}
else if (x == 3)
{
    std::cout << "oh?" << std::endl;
}
else
{
    std::cout << "huh?" << std::endl;
}
std::cout << "out of if-else" << std::endl;
```

Run this with x = 0, x = 1, x = 2, x = 3, x = 4. No surprises right?

Note that all the conditions uses the == boolean operator and it is used to compare integer values.

Modify the above:

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

switch (x)
{
    case 1:
        std::cout << "yo" << std::endl;
        break;

    case 2:
        std::cout << "right ..." << std::endl;
        break;

    case 3:
        std::cout << "oh?" << std::endl;
        break;

    default:
        std::cout << "huh?" << std::endl;
}
std::cout << "out of switch" << std::endl;
```

Run it again.

The `switch` statement is like a switchboard, directing the flow of execution based on the integer value of a variable. The flow of execution is passed to the case whose value matches the value of the variable.

Now modify it again by getting rid of the `break` statement in case 2:

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

switch (x)
{
    case 1:
        std::cout << "yo" << std::endl;
        break;

    case 2:
        std::cout << "right ..." << std::endl;

    case 3:
        std::cout << "oh?" << std::endl;
        break;

    default:
        std::cout << "huh?" << std::endl;
}
std::cout << "out of switch" << std::endl;
```

What does the `break` statement do? If you do **not** have a `break` statement, execution will actually **go to the statements of the next case without checking the**

value for that case!!!

There are **two** ways to get **out** of a `switch`: either through a **`break`** statement or when execution has reached the **end of the `switch` block**.

By now, I don't have to tell you that you can put any statement in the switch statement. You can put an if or an if-else into the switch. You can put a switch into a switch. You can put a switch into an if-else, etc. Isn't life ever so colorful?

Why? Why? Why?

So why do we need the switch when it's clear that a switch can be written as a nested if-else?

The reason is because the switch statement is extremely fast when there are many cases when compared to the if-else statement.

Note that for a nested if statement:

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

if (x == 1)
{
    std::cout << "yo" << std::endl;
}
else if (x == 2)
{
    std::cout << "right ..." << std::endl;
}
else if (x == 3)
{
    std::cout << "oh?" << std::endl;
}
else
{
    std::cout << "huh?" << std::endl;
}
std::cout << "out of if-else" << std::endl;
```

each time a boolean condition is false you have to go to the nested level in the if-else stack. This means that for instance to go into the final else block, C++ must evaluate three boolean expressions.

(One optimization trick used by C/C++ programmers is therefore to always put the most likely case first in the stack of if-else statements.)

This is however not the case for the switch statement. If the above is converted to the switch statement, the time taken to get to any case is the same. Even if you have 100 cases, the time to get to the first case is usually the same as the time to get to the last.

Combining cases

Now run this:

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

switch (x)
{
    case 1:
        std::cout << "what?" << std::endl;
        break;

    case 2:
    case 3:
        std::cout << "oh?" << std::endl;
        break;

    default:
        std::cout << "huh?" << std::endl;
}
std::cout << "out of switch" << std::endl;
```

The above is a useful trick when two cases execute the same statements. The above has the same effect as the following (although the following is probably slower):

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

if (x == 1)
{
    std::cout << "what?" << std::endl;
}
else if (x == 2 || x == 3)
{
    std::cout << "oh?" << std::endl;
}
else
{
    std::cout << "huh?" << std::endl;
}
std::cout << "out of nested if-else" << std::endl;
```

One more example to make sure you get it:

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

switch (x)
{
    case 1:
        std::cout << "what?" << std::endl;
        break;

    case 2:
    case 3:
    case 4:
    case 5:
        std::cout << "oh?" << std::endl;
        break;

    default:
        std::cout << "huh?" << std::endl;
}
std::cout << "out of switch" << std::endl;
```

Run it with several values for `x`.

No default

You can leave out the default case:

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

switch (x)
{
    case 1:
        std::cout << "yo" << std::endl;
        break;

    case 2:
        std::cout << "right ..." << std::endl;

    case 3:
        std::cout << "oh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Run this with several values for x including for instance x = 0 and x = 100.

Gotchas

Here are some gotchas ... and good source of trick questions for the instructor :)

Each of the following exercises shows you a gotcha for beginning C/C++ programmer. Make sure you find it.

Exercise. Find the bug:

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

switch x
{
    case 1:
        std::cout << "what?" << std::endl;
        break;

    case 2:
        std::cout << "oh?" << std::endl;

    case 3:
        std::cout << "huh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Exercise. Find the bug:

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

switch (x)
{
    case < 1 :
        std::cout << "what?" << std::endl;
        break;

    case 2:
        std::cout << "oh?" << std::endl;

    case 3:
        std::cout << "huh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Exercise. Bug, bug, bug! Find it! ...

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

switch (x)
{
    case 1.1:
        std::cout << "what?" << std::endl;
        break;

    case 1.2:
        std::cout << "oh?" << std::endl;

    case 1.3:
        std::cout << "huh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Exercise. Where's that sneaky bug??

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

int y = 1;

switch (x)
{
    case y:
        std::cout << "what?" << std::endl;
        break;

    case 2:
        std::cout << "oh?" << std::endl;

    case default:
        std::cout << "huh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Exercise. Recall that a character is the “atom” of a string. The name of the character type is `char`. You can declare character variables, get character values from the keyboard, and print the character value of a character variable. Try this:

```
char x = ' ';
std::cin >> x;
std::cout << x << std::endl;
```

Now here's the question: Can you use characters for the cases?

```
int x, y;
char op;
std::cout << "x: ";
std::cin >> x;
```

```
std::cout << "y: ";
std::cin >> y;
std::cout << "operator (+ or -): ";
std::cin >> op;

switch (op)
{
    case '+':
        std::cout << x + y << std::endl;
        break;

    case '-':
        std::cout << x - y << std::endl;
        break;

    default:
        std::cout << "no such op!" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Add all the binary integer operators you know so far to the above program.

Coding style

Good::

```
switch (x)
{
    case 1:
        std::cout << "what?" << std::endl;
        break;

    case 2:
    case 3:
        std::cout << "oh?" << std::endl;
        break;

    default:
        std::cout << "huh?" << std::endl;
}
```

This is also perfectly OK: :

```
switch (x)
{
    case 1:
    {
        std::cout << "what?" << std::endl;
        break;
    }

    case 2:
    case 3:
    {
        std::cout << "oh?" << std::endl;
        break;
    }

    default:
    {
        std::cout << "huh?" << std::endl;
    }
}
```

More information

Actually you can use the value of an **integer expression** and not just a variable to determine the flow of execution:

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

int y = 1;

switch (x + y)
{
    case 1:
        std::cout << "yo" << std::endl;
        break;

    case 2:
        std::cout << "right ..." << std::endl;

    case 3:
        std::cout << "oh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

And although you cannot use variables for the case values, you can use **integer constants** or even **constant integer expressions:**

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

const int y = 1;
const int z = 2;

switch (x)
{
    case y + z + 1:
        std::cout << "yo" << std::endl;
        break;

    case 2:
        std::cout << "right ..." << std::endl;
        break;

    case 3:
        std::cout << "oh?" << std::endl;
        break;
}
std::cout << "out of switch" << std::endl;
```

Summary

The statement looks like this:

```
switch ([int expr])
{
    case [const int expr]:
        [stmt]

    ...
}
```

or

```
switch ([int expr])
{
      case [const int expr]:
          [stmt]

      ...

      default:
          [stmt2]
}
```

where *[stmt]* and *[stmt2]* are either statements or blocks of statements. *[int expr]* is an expression that can be evaluated to an integer and *[const int expr]* is a constant integer expression.

Once the flow of execution enters a case, it will execute all the statements in that case until a break or end of switch block is reached. If there is no break, execution will continue with the statements of the next case without checking the value for that case, if there is one.

On executing the `break` statement in a `switch` statement, flow of control will resume to the statement immediately after the `switch` statement.

Exercise.

Q1. Read this program carefully. Determine which part of the program can be rewritten using the switch statement. Rewrite it and test it.

```
int numHeads = 0;
std::cout << "How many heads do you have? ";
std::cin >> numHeads;

if (numHeads < 1)
{
    std::cout << "Huh?" << std::endl;
}
else
{
    if (numHeads == 1)
    {
        std::cout << "Let me introduce you to our "
                  << "surgeons." << std::endl;
    }
    else
    {
        if (numHeads == 2)
        {
            std::cout << "Are you Zaphod?";
        }
        else
        {
            std::cout << "You have way too many."
                      << std::endl;
        }
    }
}
```

Q2. Rewrite the nested if-else statement using the switch statement:

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

int y = 0;

if (x == 0)
{
    y = 5;
}
else if (x == 1 || x == 2 || x == 3)
{
    y = 7;
}
else if (x == 6 || x == 7 || x = 42)
{
    y = 0;
}
std::cout << y << std::endl;
```

Avoid code duplication!

Q3. Write a program that declares four integer constants N, S, E, W. with values 0, 1, 2, 3 respectively. Generates a random integer from 0 to 3. If the random integer is N, print “go north”. If the random integer is S, print “go south”. If the random integer is E, print “go east”. If the random integer is W, print “go west”. Use a `switch` statement.