switch and case

This lesson introduces switch and case statements, how and when they are used and the implementation of the final switch statement.

WE'LL COVER THE FOLLOWING

- `switch` and `case`
  - The `goto` statement
  - The expression type
  - Value ranges
  - Distinct values
- The `final switch` statement
  - When to use switch-case

`switch` and `case`

`switch` is a statement that allows comparing the value of an expression against multiple possible values. It is similar to but not the same as an "`if`, `else if`, `else`" chain. `case` is used for specifying which values are to be compared with switch's expression. It is only a part of switch statement and not a statement itself.

`switch` takes an expression within parentheses, compares the value of that expression to the `case` values and executes the operations of the `case` that is equal to the value of the expression. Its syntax consists of a `switch` block that contains one or more `case` sections and a `default` section:

```
switch (expression) {
case value_1:
  // operations to execute if the expression is equal to value_1
  // ...
  break;
case value_2:
```

```
  // operations to execute if the expression is equal to value_2
  // ...

  break;

// ... other cases ...

default:
  // operations to execute if the expression is not equal to any case
  // ...
  break;
}
```

The expression that `switch` takes is not used directly as a logical expression. It is not evaluated as “if this condition is true,” because it would be in an `if` statement. The value of the `switch` expression is used in equality comparisons with the `case` values. It is similar to an "`if`, `else if`, `else`" chain that has only equality comparisons:

```
auto value = expression;
if (value == value_1) {
  // operations for value_1
  // ...
} else if (value == value_2) {
  // operations for value_2
  // ...
}

// ... other 'else if's ...

} else {
    // operations for other values
  // ...
}
```

However, the "`if`, `else if`, `else`" above is not an exact equivalent of the `switch` statement. The reasons for this will be explained in the following sections.

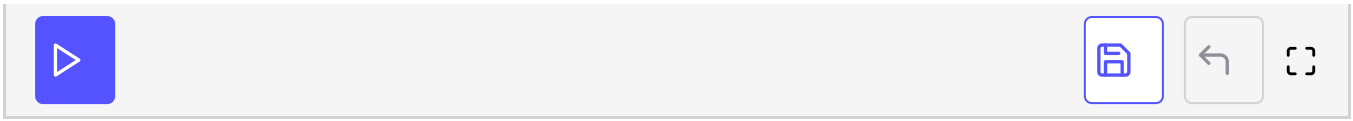
If a `case` value matches the value of the `switch` expression, then the operations that are under the `case` are executed. If no value matches, then the operations that are under the `default` are executed.

The `goto` statement

The use of `goto` is generally advised against in most programming languages. However, `goto` is useful in `switch` statements in some situations.

`case` does not introduce a scope like the `if` statement does. Once the operations within an `if` or `else` scope are finished, the evaluation of the entire `if` statement is also finished. That does not happen with the `case` sections; once a matching `case` is found, the execution of the program jumps to that `case` and executes the operations under it. When needed in rare situations, `goto case` makes the program execution jump to the next `case`:

```
import std.stdio;

void main() {
    int value = 5;

    switch (value) {
    case 5: writeln("five");
        goto case;    // continues to the next case

    case 4: writeln("four");
        break;

    default:
        writeln("unknown");
        break;
    }

}
```



Use of goto case in switch

As the `value` is **5**, the execution continues under the `case 5` line, and the program prints “five.” Then, the `goto case` statement causes the execution to continue to the next `case`, and as a result, “four” is also displayed:

```
five
four
```

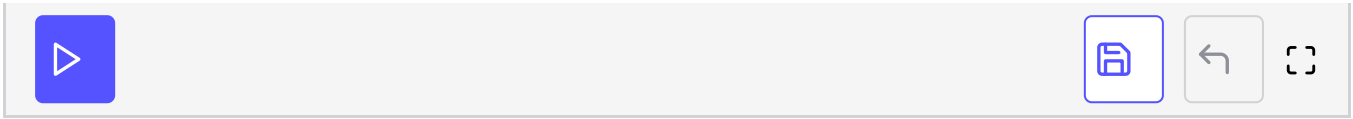
`goto` can appear in three ways under `case` sections:

- `goto case` causes the execution to continue to the next `case`.
- `goto default` causes the execution to continue to the `default` section.
- `goto case` *expression* causes the execution to continue to the `case` that

matches that expression.

The following program demonstrates these three uses by taking advantage of a `foreach` loop:

```
import std.stdio;

void main() {
    foreach (value; [ 1, 2, 3, 10, 20 ]) {
        writefln("--- value: %s ---", value);

        switch (value) {

        case 1:
            writeln("case 1");
            goto case;

        case 2:
            writeln("case 2");
            goto case 10;

        case 3:
            writeln("case 3");
            goto default;

        case 10:
            writeln("case 10");
            break;

        default:
            writeln("default");
            break;
        }
    }
}
```



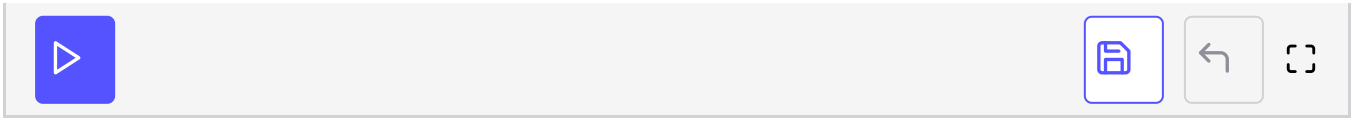
Use of goto expression

The expression type

Any type can be used in equality comparisons in `if` statements. On the other hand, the type of the `switch` expression is limited to integer types, string types and `bool`.

```
import std.stdio;
import std.format;

void main () {

    string op = "add";
```

```
    int result;
    int first = 16;
    int second = 8;

    switch (op) {
    case "add":
        result = first + second; break;

    case "subtract":
        result = first - second; break;

    case "multiply":
        result = first * second; break;

    case "divide":
        result = first / second; break;

    default:
    throw new Exception(format("Unknown operation: %s", op));
    }

    writefln("Result is %d.", result);

}
```



Type of expression in switch

> **Note:** The code above *throws an exception* when the operation is not recognized by the program. We will see exceptions in a later chapter.

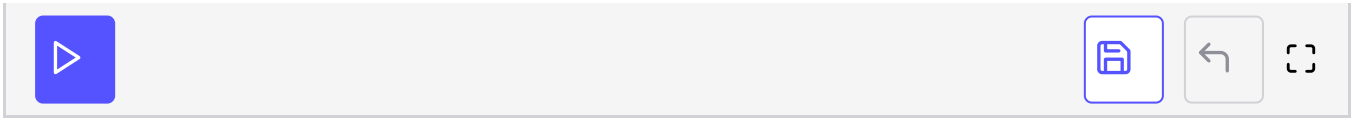
Although it is possible to use `bool` expressions in `switch`, since `bool` has only two values, there will only be two case statements in this scenario. It may be more suitable to use an if statement or the ternary operator (`?:`) with `bool` expressions.

Value ranges

A range of cases values can be specified using `..` between cases:

```
import std.stdio;

void main() {
    int dieValue = 3;

    switch (dieValue) {
    case 1:
        writeln("You won");
        break;
```

```
    case 2: .. case 5:
        writeln("It's a draw");
        break;

    case 6:
        writeln("I won");
        break;
    default:
    /* The program should never get here
    because the cases above cover the entire
    range of valid die values. (See 'final
    switch' below.) */
    break;
    }
}
```



The code above determines that the game ends in a draw when the die value is 2, 3, 4 or 5.

Distinct values

Let's assume that it is a draw for the values 2 and 4, rather than for the values that are in the range [2, 5]. Distinct values of a case are separated by commas:

```
case 2, 4:
  writeln("It's a draw");
  break;
```

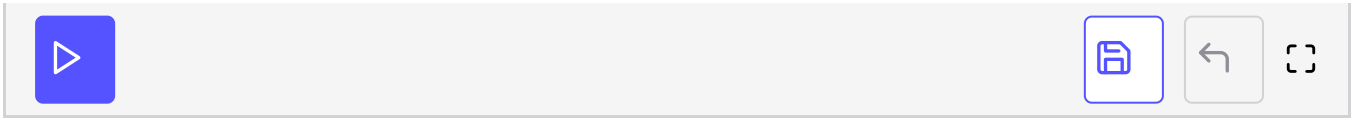
The `final switch` statement

The `final switch` statement works similarly to the regular `switch` statement, with the following differences:

- It cannot have a `default` section. Note that this section is meaningless when the `case` sections cover the entire range of values anyway, similar to the six values of the die above.
- Value ranges cannot be used with `case` sections (distinct values can be).
- If the expression is of an `enum` type, all values of the `enum` must be covered by the `case` statements.

```
import std.stdio;
```

```
void main() {


    int dieValue = 1;

    final switch (dieValue) {
    case 1:
        writeln("You won");
        break;

    case 2, 3, 4, 5:
        writeln("It's a draw");
        break;

    case 6:
        writeln("I won");
        break;
    }
}
```



When to use switch-case

`switch` is suitable for comparing the value of an expression against a set of values that are known at compile time.
When there are only two values to compare, an `if` statement may make more sense. For example, to check whether it is heads or tails:

```
if (headsTailsResult == heads) { // ...
} else {
// ...
}
```

As a general rule, `switch` is more suitable when there are three or more values to compare.
When all values of expression need to be handled, then it is preferable to use the `final switch`. This is especially the case for `enum` types.

In the next lesson, we will learn how to build a simple calculator using switch and case statements.