codecademy

Control Flow in C++

Conditional Statements

Conditional statements are used to control the flow of code execution by testing for a condition for truth.

- `if` statements execute code only if the provided condition is `true`.
- `else` statements execute code only if the provided condition in the `if` statement is `false`.
- one or more `else if` statements can be added in between the `if` and `else` to provide additional condition(s) to check.

Some useful tricks with conditional statements:

- It is possible to condense an `if - else` expression into a single statement using the following syntax:

```
variable = (condition) ? con
```

- Curly brackets `{ }` may be omitted if there is only a single statement inside a conditional statement.

```
int temperature = 60;

if (temperature < 65) {
  std::cout << "Too cold!";
}
else if (temperature > 75) {
  std::cout << "Too hot!";
}
else    // brackets may be omitted here
  std::cout << "Just right...";
```

Switch Statements

A `switch` statement provides a means of checking an expression against various `case` s. If there is a match, the code within starts to execute.

The `break` keyword can be used to terminate a `case` . If the `break` keyword is missing from a `case` , it will cause code execution to overflow to subsequent `case` s.

The code within the `default` block is executed when no other `case` matches.

```
switch (grade) {
  case 9:
    std::cout << "Freshman\n";
    break;
  case 10:
    std::cout << "Sophomore\n";
    break;
  case 11:
    std::cout << "Junior\n";
    break;
  case 12:
    std::cout << "Senior\n";
    break;
  default:
    std::cout << "Invalid\n";
    break;
}
```

Loops

In C++, loops repeatedly execute code as long as the provided condition is `true`.
There are four main types of loops in C++:

1. `while` loops: repeats a block of code as long as the given boolean condition is `true`.
2. `do-while` loops: similar to `while` loops, but run at least once.
3. `for` loops: repeats a block of code a specific number of times.
4. `for-each` loops: used to iterate through every item in an array or list-like structure.

```
// while loop
int count = 0;
while (count <= 10) {
  std::cout << count;
  count++;
}

// do-while loop
int price = 300;
do {
  std::cout << "Too expensive!";
} while (price > 500);

// for loop
for (int i = 0; i <= 10; i++) {
  std::cout << i;
}

// for-each loop
int fibonacci[5] = { 0, 1, 1, 2, 3 };
for (auto number:fibonacci){
  std::cout << number;
}
```

Break and Continue

In C++, the `break` keyword is used to exit a switch or loop.
The `continue` keyword is used to skip an iteration of a loop.

```
// Prints: 0123
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  std::cout << i;
}

// Prints: 012356789
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  std::cout << i;
}
```

`if` Statement

An `if` statement is used to test an expression for truth.

- If the condition evaluates to `true`, then the code within the block is executed; otherwise, it will be skipped.

```
if (a == 10) {
  // Code goes here
}
```

`else` Clause

An `else` clause can be added to an `if` statement.

- If the condition evaluates to `true`, code in the `if` part is executed.
- If the condition evaluates to `false`, code in the `else` part is executed.

```
if (year == 1991) {
  // This runs if it is true
}
else {
  // This runs if it is false
}
```

Relational Operators

Relational operators are used to compare two values and return `true` or `false` depending on the comparison:

- `==` equal to
- `!=` not equal to
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to

```
if (a > 10) {
   // ☝ means greater than
}
```

`else if` Statement

One or more `else if` statements can be added in between the `if` and `else` to provide additional condition(s) to check.

```
if (apple > 8) {
  // Some code here
}
else if (apple > 6) {
  // Some code here
}
else {
  // Some code here
}
```