# 1 Conditionals

*Conditionals* let you execute code based on whether or not something is true. As shown in the following sections, there are three main types of conditionals in C++.

## 1.1 if/else statements

The most common is the if/else statement

```
if (i > 4) {
// Do something
} else if (i>2) {
// Do something else
} else {
// Do soemthing else
}
```

The expression between the parathesis must be a boolean value.

# 2 switch statements

The switch statement is an alternate syntax for performing actions based on the value of an expression. In C++ switch statements, the expression must be of an integral type or of a type that is convertible to integral type, and must be compared to constants. Each value represents a ”case”. Then the code following the case is executed until a break statement is reached. You can also provide a default case, which is matched if none of the other cases match.

```
switch(menuItem) {
    case OpenMenuItem:
        // Code to open a file
        break;
    case SaveMenuItem:
        // Code to save a file
        break;
    default:
        // Code to give an error message
        break;
}
```

A switch statement can always be converted into if/else statements. The previous statement can be converted as follows:

```
if (menuItem == OpenMenuItem) {
    // Code to open a file
} else if (menuItem == SaveMenuItem) {
    // Code to save a file
} else {
    // Code to give an error message
}
```

If a case section does not have a break statement, the code for that case section is executed first, followed by a *fallthrough*, executing the code for the next case section whether or not that case matches. This can be a source of bugs, but is sometimes useful. One example is to have a single case section that is executed for several different cases. For example,

```
case ColorDarkBlue;
case ColorBlack:
   // Code to execute for both a dark blue or black background colour.
   break;
case ColorRed:
   // Code to execute for a red background colour
   break;
```

## 2.1 The conditional operator

C++ has one operator that takes three arguments, known as the *ternary operator*. It is used as a shorthand if/else statement. The following code will output ”yes” if the variable is greater than 2, and ”no” otherwise.

```
std::cout << ((i > 2) ? "yes" : "no");
```

Unlike an if statement or a switch statemnt, the conditional operator doesn't execute code blocks based on the result. Instead it is used *within* code, as shown as the preceding example. In a sense, it is an operator (like + and −) as opposed to a true conditional.

## 2.2 Logical Evaluation Operators

| OP | DESCRIPTION | USAGE |
|---|---|---|
| <, <=, >, >= | Determines if the left-hand side is less than, less or equal to, greater than, or greater than or equal to the right hand side | if (i < 0) {std::cout << ”i is negative”);} |
| == | Determines if left hand side equals the right hand side. | if (i == 3) {std::cout ¡¡ ”i is 3”;} |
| ! = | Not equals. True if left hand side not equal right hand side | if (i ! = 3) {std::cout << ”i is not 3;”} |
| ! | Logical NOT. Complements true/false of status of a boolean expression. Unary operator | if (!someBoolean) {std::cout << ”someBoolean is false”;} |
| && | Logical AND. Result is true if both parts is true. | if (someBoolean && someOtherBoolean) {std::cout << ”both are true”;} |
| \|\| | Logical OR. Result if true if either part is true. | if (someBoolean \|\| someOtherBoolean) {std::cout << ”at least one is true”;} |