Expressions

Learn about the Arithmetic, Relational and Equality operators. You'll also be able to write better code when you will know about operator precedence and a common bug made with the equality operator.

We'll cover the following

Like in any other programming language, in C, there are many arithmetic, relational, and logical **operators** we can use to write **expressions** that are made up of simpler basic types.

Arithmetic Operators

The following binary arithmetic operators can be used in C: `+`, `-`, `*`, `/` and the modulus operator `%`. When writing arithmetic expressions, we must always be aware of **operator precedence**, which is the order in which operators are applied when evaluating an expression.

For example `4+5*6` evaluates to `34`, because the `*` operator has precedence over the `+` operator, and so the expression is evaluated as `4 + (5*6)`, **not** `(4+5)*6`. My strategy to deal with this is always to use brackets to denote the desired precedence in arithmetic expressions explicitly. So instead of writing:

```
double q = a*x*x+b*x+c;
```

which is a perfectly accurate expression of the quadratic equation:

```
\begin{equation} ax^{2} + bx + c \end{equation}
```

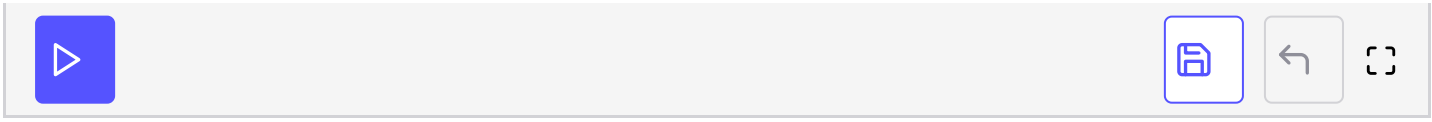
I would rather code it like this:

```
double q = (a*x*x) + (b*x) + c;
```

Another illustration of operator precedence:

What are the values of the `result1`, `result2` and `result3` variables in the following code?

```
#include <stdio.h>
int main() {
  int a=100, b=2, c=25, d=4;
  int result1, result2, result3;
  result1 = a * b + c * d;
  result2 = (a * b) + (c * d);
  result3 = a * (b + c) * d;
  printf("result1=%d, result2=%d, result3=%d\n",
         result1, result2, result3);
  return 0;
}
```



Always using brackets will avoid cases where operator precedence messes up your calculations. These errors are difficult to debug.

Wikipedia provides a chart showing **operator precedence**.

Relational and Logical Operators

The relational operators are `>`, `>=`, `<` and `<=`, which all have equal precedence. There are also two equality operators: `==` and `!=`.

A very common **gotcha** in C programming is to erroneously use the assignment operator `=` when you mean to use the equality operator `==`, for example:

```
if (grade = 49) grade = grade + 1; // INCORRECT !!!
if (grade == 49) grade = grade + 1; // CORRECT
```

In line 1 above, the expression `grade=49` doesn't **test** for the equality of the variable `grade` and the constant `49`, it **assigns** the value `49` to the variable grade. What we want is in line 2 where we use the equality operator `==` to test if `grade==49`. This bug is a tough one to spot when it happens.

There are two **logical operators** `&&` (logical AND) and `||` (logical OR).

By default in C, the results of relational and logical operators are evaluated to integer values: `0` for FALSE and `1` for TRUE.

By now, you probably have a good idea about how to write expressions. In the next lesson, we'll look at different ways to convert one data type into another.