CS50 Boolean Expressions

Overview

Conditions are how programmers can make decisions in programs, by allowing some parts of the code to only run under certain circumstances. Conditions will generally work by evaluating a **boolean expression**, which is an expression that will have a value of either `true` or `false`. Programmers can set conditions such that different code will run depending on what the value of the boolean expression is.

Key Terms

- condition
- boolean expression
- if statement
- switch statement
- ternary operator

```
1 | bool a = 3 < 5;
2 | bool b = 2 >= 8;
3 | bool c = a && b;
4 | bool d = a || b;
5 | bool e = !d;
```

a	b	c	d	e
true	false	false	true	false

Boolean Operators

Boolean operators are used to create boolean expressions that evaluate to `true` or `false`. Common boolean operators include the comparison operators: `<` (less than), `>` (greater than), `==` (equal to), `<=` (less than or equal to), `>=` (greater than or equal to), and `!=` (not equal to). For instance, in line 1 to the left, `a` is set to `true` because the expression `3 < 5` is true (because 3 is in fact less than 5). In line 2, `b` is set to `false` because the expression `2 >= 8` is not true.

Logical operators can also be used to combine boolean expressions. `&&` is the logical AND operator: it will evaluate to `true` if both expressions on either side of it are true. `||` is the logical OR operator: it evaluates to `true` if at least one of the two expressions on either side is true. And `!`, the logical NOT operator, evaluates to the opposite of whatever the expression immediately after it is.

Conditions

Conditional branching refers to the idea that different parts of code should execute under different circumstances. The most common type of conditional is the **if statement**: where a certain block of code (enclosed in brackets) will only run if the condition (whatever is in the parentheses after the word `if`) evaluates to `true`.

Optionally, C also allows you to include an `else` block after an `if` statement, which defines which code should run if the `if` condition evaluates to `false`. C will also allow you to include one or multiple `else if` statement after an `if` statement, to add additional conditions that could run different blocks of code. The if statement to the right (lines 1-12) will print `"positive\n"` if the value of `x` is greater than `0`, `"negative\n"` if the value of `x` is less than `0`, and `"zero\n"` if the value of `x` is equal to `0`.

C also has other ways of expressing conditionals. The **switch statement**, shown to the right (lines 15-25), takes one variable, and defines what code should run based on which `case` the variable matches. In the example at right, if `x` is equal to `1`, `"A\n"` is printed; if x is equal to `2`, `"B\n"` is printed, and in all other cases (the `default` case), `"C\n"` is printed. Code within cases should end with `break` so that the program knows to stop executing code and go to the end of the `switch` statement.

The ternary operator is a third type of condition. The **ternary operator** takes an expression, and evaluates to one value if the expression is true, and another value if it is false. In the example on line 28, if `x > 3`, `y` is set to 2, and `1` otherwise.

```
1  if (x > 0)
2  {
3      printf("positive\n");
4  }
5  else if (x < 0)
6  {
7      printf("negative\n");
8  }
9  else
10 {
11     printf("zero\n");
12 }
13
14
15 switch (x)
16 {
17     case 1:
18         printf("A\n");
19         break;
20     case 2:
21         printf("B\n");
22         break;
23     default:
24         printf("C\n");
25 }
26
27
28 int y = (x > 3) ? 2 : 1;
```