CS50 Loops

Overview

Loops are a way for a program to execute the same code multiple times. Instead of copying and pasting the same lines back-to-back, loops allow for code to be repeated. The resulting code is better designed: if you need to change the code that gets repeated, you only need to change it once. C has multiple different types of loops: all of which can accomplish the same things, though some may be preferable to others depending on the circumstances.

Key Terms

- loops
- for loop
- while loop
- infinite loop
- do while loop

For Loops

```
1  for (int i = 0; i < 10; i++)
2  {
3      printf("hello!\n");
4  }
```

```
5  for (int j = 0; j < 10; j++)
6  {
7      printf("%d\n", j);
8  }
```

The first type of loop in C is the **for loop**. Defining a `for` loop requires three parts (included in parentheses after the word `for`, and separated by semicolons), demonstrated at left (lines 1-4).

The first part is the initialization: we create a variable `i` initially set to `0`. Second is the condition: as long as the condition `i < 10` is `true`, everything within the curly braces will keep running. As soon as the condition is `false`, then the loop ends. The third part is the loop modification: this code is executed at the end of every loop. In this case, we modify our loop by increasing the value of `i` by `1`.

Thus, each time the loop finishes, `i` will increase in value by `1`. As soon as `i` is no longer less than `10`, the condition fails and the loop will end. The end result is that `"hello\n"` is displayed 10 times.

By taking advantage of loop modification, you can also get a loop to do something slightly different each time the loop iterates. In the second `for` loop example (lines 5-8 above), `j` is initially `0`, and so `0` is printed. Then `j` increments to `1`, and `1` is printed in the next loop iteration. This continues until `j` is no longer less than `10`. The result is that each number from `0` to `9` is printed on its own line.

While Loops

C also includes a type of loop called a **while loop**. A `while` loop checks the condition it is given: if it is true, it executes the code within the braces, and then checks the condition again. This process repeats until the condition is false. The example at right (lines 9-14) does exactly the same thing as our second `for` loop (lines 5-8): printing out the numbers from `0` to `9`.

```
 9  int k = 0;
10  while (k < 10)
11  {
12      printf("%d\n", k);
13      k++;
14  }
```

If the `while` loop is given a condition that is always `true` (like the boolean value `true` itself), then the loop will never stop running. The example at right (lines 15-18) is an example of an **infinite loop**: since the condition will never be false, the loop will continue running indefinitely. `While` loops are particular useful when you don't know in advance how many times a loop should run.

```
15  while (true)
16  {
17      printf("hello!\n");
18  }
```

Do-While Loops

```
19  int j;
20  printf("Positive Number: ");
21  do
22  {
23      j = GetInt();
24  }
25  while (j <= 0);
```

The **do-while loop** is similar to a while loop in the sense that it repeats a loop until a condition is false. However, a `do-while` loop, unlike a `while` loop, will always execute at least once, regardless of the condition. This is often valuable in cases where user input is required: the program should definitely ask for input once, and may or may not need to ask for input more times if the input is invalid.

In the example at left, the user will be prompted to enter an integer, and will be re-prompted continuously until a positive one is provided.