owdy!

Welcome to the first in what I hope will be a long series of columns on Mac programming. Creator Code aims to help all sorts of programmers, from C to C++ to Pascal and beyond, with both simple and advanced concepts. I love feedback, so feel free to email me at chilton@applewizards.net . Before you do, though, stop by my website at http://www.devhq.com/ .

This month we'll be doing some relatively simple stuff: loops. We'll be doing this in C++, and on CodeWarrior Pro 4 (IDE v3.3), though the examples we'll list will work on several versions prior to Pro 4. I'm not one for beating around the bush, so let's get to it.

asics of Loops

Every loop has a few things in common. Some are:

Exit Conditions — Specifically, when does the loop end? The conditions are typically updated each time through the loop.

Statements — There'd be no point in having a loop that didn't do anything. Statements are typically contained within curly braces (a.k.a. squigglies) and are executed over and over as the loop dictates. With the exception of for loops, the statement typically contains an update to the exit conditions.

Nestability — Every loop we'll discuss in this article can be nested. I'll give you some code for a little nested loop at the end of this month's column.

o It FOR So Many Times

The For loop is used by programmers when an action or a set of actions is to occur a set number of times — its test is a pre-loop test. For example, a programmer who wanted to print out a list of integers from 0 to 9 could do it using the following loop:

```
for (int counter = 0; counter < 10; counter++)

      cout << counter << endl;
;
```

The output? You guessed it:

```
0
1
2
:
:
8
9
```

To construct a for loop, it's necessary to understand just what the statements between the parentheses are doing.

1. `int counter = 10` — This sets our "counter" to 0 initially. It is important to note that the counter's scope is limited to this for loop. Had we previously declared counter as type int, we could simply specify `(counter = 0; ...)`.

2. `counter < 10` — This is the actual test. It is performed once for each iteration of the loop, and so long as it resolves to true, the statements of the loop are executed. Thus, so long as "counter" is less than 10, the statement is executed.

3. `counter++` — After the statement executes, this operation is performed: it increments our counter a single unit (from 0 to 1 and so on). Control then reverts back to the test portion of the loop, `counter < 10`.

ould you DO Something For Me?

The Do loop executes once before a single test is made. This makes it an effective loop for menus: the programmer can specify a menu of options which will loop and display itself again if incorrect input is entered. Let's take a look at an example:

```
int choice;

do

      cout << "Do you want to: " << endl;
      cout << "1 - Go to the store" << endl;
      cout << "2 - Go home" << endl;
      cout << "3 - Quit" << endl;
      cout << "Your choice: ";
      cin >> choice;
 while (choice < 1 || choice > 3);
```

Upon entering the loop at the do statement, the menu is printed and the user is prompted for a choice. If they enter 1, 2, or 3, both post-loop tests (or a compound test, here using || "or") are false and the loop ends. If the user enters 46, however, the compound test is true, and the loop executes again. More complex loops can be created to print out an additional message, like "You entered incorrect input. Try again." Go ahead, think of how? It's two lines of code:

```
if (choice < 1 || choice > 3)

      cout << "Try again you dummy." << endl;
```

HILE You're At It...

Do loops are great if you want the statement to be executed at least one time. For loops are great for executing a statement a predetermined number of times. While loops fill the remaining void — performing a loop a varying number of times, including zero. Here's some code:

```
char user_enters;
cout << "Enter your own numbers (y or n): ";
cin >> user_enters;

while (user_enters == 'y')

      cout << "Enter the next number: ";
      cin >> next_num;
      total += next_num;

      cout << "Enter more (y or n): ";
      cin >> user_enters;
```

This loop is a pre-test loop: it first tests the condition (in this case, whether

the variable user_enters currently holds the value 'y'). If the test fails, the entire block below is skipped. If it's true, the statements execute. It's important to note that before the loop returns to the test, the condition is updated. In this case, the user must enter 'y' again to perpetuate the loop.

esting a Few

Loops can be nested, and sometimes to great effect. Need an example? Of course:

```
cout << "Now we'll enter 10 positive numbers and average them.\n";

for (int counter = 0; counter < 10; counter++)

      do

            cout << "Enter number " << counter << ": ";
            cin >> number;
       (while number < 1);

      sum += number;
```

ext month? I'm not sure. I'm looking forward to your feedback — send some off to chilton@applewizards.net . Also be sure to drop by my website at http://www.devhq.com/. Until then, happy coding.

Note: Thanks go out to Erik for jumping in at the last minute to write this article for me. I'll be around next month for sure!

Chilton Webb
chilton@applewizards.net