Section 1 / For Loops, Continue and Break

Overview

We have already covered the `if` and `while` statements. We demonstrated that a `while` loop is nothing more than an `if` statement with one additional label preceding and one unconditional branch following the code for an `if` statement.

A `for` loop is only slightly more complex.

In C++ and C

In C++ and C, a for loop looks like this:

```
for (set up; decision; post step)                    // 1
{                                                    // 2
    // CODE BLOCK                                    // 3
}                                                    // 4
```

How You Picture It Versus How It is Implemented

The image on the left, below, represents from top to bottom what the parts of the `for` are from left to right (the post step being found at the bottom).

The image on the right, above, depicts how `for` loops are *typically* implemented. The reason for this becomes clear when we see the assembly language.

In Assembly Language

This code:

```
for (long i = 0; i < 10; i++)                        // 1
{                                                    // 2
    // CODE BLOCK                                    // 3
}                                                    // 4
```

could be implemented like this in assembly language:

```
    // Assume i is implemented using x0                  // 1
                                                         // 2
    mov     x0, xzr                                      // 3
                                                         // 4
1:  cmp     x0, 10                                       // 5
    bge     2f                                           // 6
                                                         // 7
    // CODE BLOCK                                        // 8
                                                         // 9
    add     x0, x0, 1                                    // 10
    b       1b                                           // 11
```

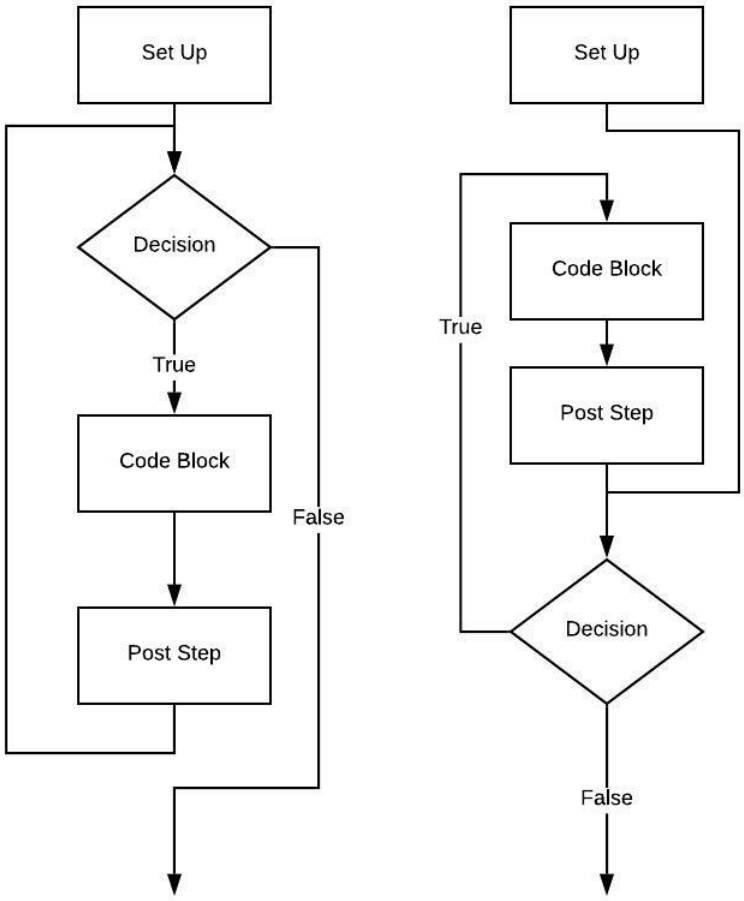
Set Up
Decision
True
Code Block
Post Step
False
Set Up
Code Block
True
Post Step
Decision
False

Figure 1: for

```
                                                                    // 12
2:                                                                  // 13
```

This corresponds to the flow chart on the **left**, above. There are 4 instructions in the loop (ignoring the code block).

The next set of assembly language corresponds to the flow chart on the right, above, where the post step and decision comes *after* the code block.

```
    // Assume i is implemented using x0                             // 1
                                                                    // 2
    mov     x0, xzr                                                 // 3
    b       2f                                                      // 4
                                                                    // 5
1:                                                                  // 6
                                                                    // 7
    // CODE BLOCK                                                   // 8
                                                                    // 9
    add     x0, x0, 1                                               // 10
2:  cmp     x0, 10                                                  // 11
    blt     1b                                                      // 12
```

Notice this contains one fewer lines of assembly language within the loop itself (3 lines versus 4). Again, the contents of the code block are not counted.

Implementing a `continue`

Now let's add a `continue` to the code block, dividing it in two.

```
for (long i = 0; i < 10; i++) {
    // CODE BLOCK "A"
    if (i == 5)
        continue;
    // CODE BLOCK "B"
}
```

Upon encountering a `continue`, control branches to the post step of a `for`, or with a `while` to the decision test to keep going.

Here is what we would need to write to support a `continue` if the "conventional" ordering were used with the decision evaluation at the top:

```
    // Assume i is implemented using x0                             // 1
                                                                    // 2
    mov x0, xzr                                                     // 3
                                                                    // 4
1:  cmp x0, 10                                                      // 5
    bge 3f                                                          // 6
                                                                    // 7
    // CODE BLOCK "A".                                              // 8
```

```
                                                                    // 9
    // if (i == 5)                                                  // 10
    //      continue                                                // 11
                                                                    // 12
    cmp x0, 5                                                       // 13
    beq 2f                                                          // 14
                                                                    // 15
    // CODE BLOCK "B"                                               // 16
                                                                    // 17
2:  add x0, x0, 1                                                   // 18
    b   1b                                                          // 19
                                                                    // 20
3:                                                                  // 21
```

Here is the original code.

Below, is how a `for` loop is **typically** implemented.

```
    // Assume i is implemented using x0                             // 1
                                                                    // 2
    mov x0, xzr                                                     // 3
    b   3f                                                          // 4
                                                                    // 5
1:                                                                  // 6
                                                                    // 7
    // CODE BLOCK "A"                                               // 8
                                                                    // 9
    // if (i == 5)                                                  // 10
    //      continue                                                // 11
                                                                    // 12
    cmp x0, 5                                                       // 13
    beq 2f                                                          // 14
                                                                    // 15
    // CODE BLOCK "B"                                               // 16
                                                                    // 17
2:  add x0, x0, 1                                                   // 18
3:  cmp x0, 10                                                      // 19
    blt 1b                                                          // 20
```

Here is the original code.

Once again, the code moving the post step and decision evaluation to the bottom is one fewer instruction inside the loop.

Implementing a `break`

The implementation of `break` is very similar to that of `continue`. Upon encountering a `break`, control branches to the instruction beyond the bottom of the

loop. There is no difference in implementation of `break` for `while` loops and `for` loops.

```
for (long i = 0; i < 10; i++) {
    // CODE BLOCK "A"
    if (i == 5)
        break;
    // CODE BLOCK "B"
}
```

In assembly language, a `break` is implemented by a branch to beyond the end of the loop (either `for` or `while`).

```
    // Assume i is implemented using x0                    // 1
                                                           // 2
    mov x0, xzr                                            // 3
    b   3f                                                 // 4
                                                           // 5
1:                                                         // 6
                                                           // 7
    // CODE BLOCK "A"                                      // 8
                                                           // 9
    // if (i == 5)                                         // 10
    //      continue                                       // 11
                                                           // 12
    cmp x0, 5                                              // 13
    beq 4f                                                 // 14
                                                           // 15
    // CODE BLOCK "B"                                      // 16
                                                           // 17
2:  add x0, x0, 1                                          // 18
3:  cmp x0, 10                                             // 19
    blt 1b                                                 // 20
                                                           // 21
4:                                                         // 22
```

Compare `line 14` of the `break` example to the same line in the `continue` example.

Summary

`for` loops typically contain code ordering different from what one might expect. This is done to save an instruction within the loop. While this doesn't sound like much, consider the case where the loop is executed billions of times. In this case, saving one instruction per loop prevents the execution of a billion instructions.

The shorter the code block is, the more important it is to save one instruction from within the loop.

The discussion of `break` and `continue` is found here, as `for` loops are slightly more complicated than `while` loops due to the post step. However, the ideas presented here translate to the `while` in a straight forward manner,

Questions

1

(T | F) Given that both the `while` and `for` are based on `if` statements and branches, any `for` loop can be converted to a `while` and vice versa?

Answer: True - `for` and `while` are interchangeable.

2

Is there a rule of thumb that will tell you when to use a `while` and when to use a `for`? If so, what is it?

Answer: Yes! When you know in advance how many times you will loop, use a `for`. If you don't know in advance, use a `while`.