C Programs: Loops
Part II

Loops with conditions

`for`-loop is most often used to repeat the same set of instructions a **specific** number of times.

In other situations, we may not know the exact number of times the loop has to repeat. Instead, we may know **loop repetition conditions** or **loop termination conditions**.

A **loop repetition condition** is an expression which when evaluated to true specifies that the loop has to make one more iteration. Conversely, if the loop repetition condition evaluates to false, the loop terminates.

A **loop termination condition** is the inverse of the loop repetition condition.

C provides two types of statements for loops with conditions: `while`-loop and `do-while` loop.

Loops with Preconditions

Syntax

```
while (<expression>)
   <statement-block>
```

Here, `<expression>` is the loop repetition condition, and `<statement-block>` is a single statement, or a sequence of statements that are executed at each loop iteration.

Semantics

The execution of the `while`-loop proceeds as follows.

1. `<expression>` is evaluated. If it evaluates to false, the loop is terminated (or not started).
2. If `<expression>` evaluates to true, `<statement-block>` is executed. Then, step 1 of evaluation is performed again.

Note: `<expression>` is evaluated **before** any iterations of the `while`-loop occur.

Uses

As a counting loop. `While`-loop can be used to perform a set of instructions a pre-defined number of times, by using a loop counter variable and updating it manually in the body of the loop.

```
/* computng table of squares */
int x,s;
x=0;
while (x <=20) {
   s = x*x;
   printf("%d*%d = %d\n", x,x,s);
   x = x+1;                 /* don't forget to increment the counter */
}
```

is equivalent to:

```
/* computng table of squares */
int x, s;
for(x=0;x<=20;x=x+1) {
   s=x*x;
   printf("%d*%d = %d\n", x,x,s);
}
```

Input consumption This works well, if you don't know the size of the input, but the end of the input is signified by a specific values.

```
/* output the square of input numbers.              */
 int success;
 int x;

 success = 0;

 while (!(success)) {
    scanf("%d", &x);
    printf("%d*%d = %d\n", x,x,x*x);
    if (x < 0) {
       success = true;
    }
 }
```

Note: The input to the program is a sequence of numbers, terminated with a negative number.

Note: we use an `int` (boolean, really) variable, `success` to keep track of whether next iteration should be performed.

Loops with Postconditions

Syntax

```
do
  <statement-block>
while (<statement>);
```

Here, `<expression>` is the loop repetition condition, and `<statement-block>` is a single statement, or a sequence of statements that are executed at each loop iteration.

Semantics

The execution of the `do-while`-loop proceeds as follows.

1. `<statement-block>` is executed. Then, step 1 of evaluation is performed again.
2. `<expression>` is evaluated. If it evaluates to false, the loop is terminated. If it evaluated to true, control is passed to Step 1.

Note: `<expression>` is evaluated, for the first time, **after** the first iteration of the `do-while`-loop occurs.

Uses

Sentinels. (Variable guards)

```
int x;

do {
   scanf("%f",&x);
} while((x <0) || (x>100))
```

Note: `scanf` statement asks for an `int` with value between 0 and 100 (exclusively). If the enetered values are correct, loop postcondition is false and the control is passed to the first statement following the loop. Otherwise, the entered value was either negative or more than 100. The latter triggers another interation of the loop.