ECE 131 – Programming Fundamentals

Module 3, Lecture 3: Program Flow Control – Loop Statements

Dr. Daryl O. Lee

University of New Mexico

Loop Statements

The three types of loop statements in C are:

- for loops
- while loops
- do-while loops

The for Loop

The C syntax of the for loop statement is:

> for (<*initialization expr*>; <*continuation expr*>; <*loop expr*>)
> *program block B*

Where:

- The *initialization expr* is executed once, when the for loop is first encountered.
- The *continuation expr* is evaluated prior to executing each iteration of the loop. If this expression is true, another iteration through *program block B* is performed; and if it is false, control jumps to the statement immediately following this program block.
- The *loop expr* is executed at the completion of each iteration of the loop.

Note: Semicolons are required to separate the three expressions in the for loop statement.

The for Loop

Ex. Compute the sum of the first n integers, i.e.,

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n$$

```
#include <stdio.h>
main() {
   int i; // loop index
   int n, result=0;
   printf("Enter an integer -> ");
   scanf("%d", &n);
   for(i=1; i<=n; i++)
      result += i;
   printf("Sum of first %d integers = %d \n", n,result);
}
```

The scanf() Function

The scanf function read (or scans) input from the keyboard according to a specified format.

- Syntax: int scanf(*format*, *arg list*)
  where *format* specifies the type of data you will input, and the *arg list* contains the address where you want the input stored.
- In the previous program, the argument list used the address-of operator (&). This operator returns the address in memory where the variable or constant provided to the right resides.
- The return value of scanf() is the number of input items that were successfully stored in the provided addresses.
- So if the scanf() function fails to store any input, a 0 will be returned. We'll use this fact shortly.
- Later, we'll use a related function, fscanf(), to read data from a file.

The for Loop

The loop expression can be more complicated, i.e., it can be more than a simple increment statement.

Ex. Compute, $\sum_{\substack{i=1 \\ i\ even}}^{n} i$, the sum of the first n even integers:

```
#include <stdio.h>
main() {
  int i; // loop index
  int n, result=0;
  printf("Enter an integer -> ");
  scanf("%d", &n);
  for(i=2; i<=n; i+=2)
     result += i;
  printf("Sum of first %d even integers = %d \n", n,result);
}
```

The for Loop

Ex. Compute the sum of the first n even and odd integers:

```
#include <stdio.h>
main() {
  int i,j; // loop indicies
  int n, evenSum=0, oddSum=0;
  printf("Enter an integer -> ");
  scanf("%d", &n);
  for(i=2, j=1; i<=n; i+=2, j+=2) {
     evenSum += i;
     oddSum += j;
  }
  printf("Sum of first %d even integers = %d \n", n,evenSum);
  printf("Sum of first %d odd integers = %d \n", n,oddSum);
}
```

The for Loop

To count down, simply decrement the loop index:

```
#include <stdio.h>
main() {
   int i; // loop index
   for(i=10; i>0; i--)
      printf("%d\n", i);
   printf("blast off!\n");
}
```

This produces the output:

```
10
9
8
7
6
5
4
3
2
1
blast off!
```

The while Loop

The C syntax of the while loop statement is:

```
while (continuation expr)
    program block B
```

- The *continuation expr* is evaluated prior to executing each iteration of the loop. If this expression is true, another iteration through *program block B* is performed; and if it is false, control jumps to the statement immediately following this program block.
- Notice that if you have the while loop, you strictly don't really need the for loop construct.
- Nevertheless, you should use the for loop when the situation warrants its use.

The while Loop

Any for loop:

```
for (initialization expr; continuation expr; loop expr)
    program block B
```

can be implemented using a while loop as follows:

```
initialization expr
while (continuation expr) {
    program block B
    loop expr
}
```

- The for loop is a cleaner syntax to use if you have a situation that requires initialization of a loop variable, and incrementing (or decrementing) of the loop variable on each iteration.

The while Loop – Example

Ex. We can implement Euclid's algorithm from Module #1 using a while loop:

```
#include <stdio.h>
int euclid(int a, int b) {
   int temp;
   while (b != 0) {
      temp = b;
      b = a % b;
      a = temp;
   }
   return a;
}
```

The for Loop – Example

Ex. Here's the same algorithm implemented using a for loop.

```
#include <stdio.h>
int euclid(int a, int b) {
   int i, j, temp;
   for (i=a, j=b; j!=0; temp=j, j=i%j, i=temp);
   return i;
}
```

- All of the logic associated with Euclid's algorithm is implemented in one line — there's no program block!
- This implementation is more difficult to understand than the previous one, but much more compact.

The do-while Loop

- The previous loop statements are said to be entry-condition loops. I.e., the test for determining whether or not another iteration should be performed is conducted prior to entering the loop.
- The do-while loop is an exit-condition loop. I.e., one iteration of the loop is performed prior to checking the continuation expression.
- In programming, the need for exit-condition loops is not nearly as common as the need for entry-condition loops.
- Choose the loop that makes sense for your application!

The do-while Loop

The C syntax of the while loop statement is:

```
do
    program block B
while (continuation expr);
```

Ex. This loop will continue to execute as long as a “Y” is entered:

```
char ans;
int num;
do {
   printf("Enter an integer number -> ");
   scanf("%d", &num);
   getchar(); // flush the buffer
   // program statements
   printf("Enter another number (Y or N) -> ");
   ans = getchar();
   getchar(); // flush the buffer
} while (ans == ’Y’);
```

Buffered Input

- Why did we need to "flush the buffer"? When you supply input, it's buffered, i.e., it's stored in a buffer, and that's where scanf() actually takes its values from.
- When you entered a integer number in the previous program, you first entered a number, and then hit the "enter" or "return" key. Both the integer number and the enter character are stored in the buffer. The scanf() function will "eat" the first integer, but it leaves the enter character sitting in the buffer.
- Thus, we remove the enter character by using the getchar() function, defined in stdio.h. This function reads a single character from the buffer.

  Note: We also could have used the scanf() function with the %c format in order to "eat" the enter character.