CMSC 104 - Lecture 10
John Y. Park, adapted by C Grasso

More Loops

More Loops

Topics

- Counter-Controlled (Definite) Repetition
- Event-Controlled (Indefinite) Repetition
- for Loops
- do-while Loops
- Choosing an Appropriate Loop
- Break and Continue Statements

Counter-Controlled Repetition (Definite Repetition)

- If it is known in advance exactly how many times a loop will execute, it is known as a **counter-controlled loop**.

```
int i = 1 ;
while ( i <= 10 ) {
   printf("i = %d  \n", i) ;
   i = i + 1 ;
}
```

Event-Controlled Repetition (Indefinite Repetition)

- If it is NOT known in advance exactly how many times a loop will execute, it is known as an **event-controlled loop**.

```
sum = 0 ;
printf("Enter an integer value: ") ;
scanf("%d", &value) ;

while ( value != -1) {
   sum = sum + value ;
   printf("Enter another value: ") ;
   scanf("%d", &value) ;
}
```

Event-Controlled Repetition

- An event-controlled loop will terminate when some **event** occurs.
- The event may be the occurrence of a sentinel value, as in the previous example.
- There are other types of events that may occur, such as reaching the end of a data file.

The 3 Parts of a Loop

```
#include <stdio.h>

int main () {

   int i = 1 ;          <-- init of loop control variable

   /* count from 1 to 100 */
   while ( i < 101 ) {  <-- test of loop termination

      printf ("%d ", i) ;
      i = i + 1 ;       <-- modify loop control
   }
      return 0 ;
}
```

The for Loop Repetition Structure

- The **for** loop handles details of the counter-controlled loop "automatically".
- The initialization of the the loop control variable, the termination condition test, and control variable modification are handled in the **for** loop structure.

```
for (i = 1; i < 101; i = i + 1) {
}
```

When Does a for Loop Initialize, Test and Modify?

- Just as with a while loop, a **for** loop
  - initializes the loop control variable before beginning the first loop iteration
  - performs the loop termination test before each iteration of the loop
  - modifies the loop control variable at the very end of each iteration of the loop
- The for loop is easier to write and read for counter-controlled loops.

A for Loop That Counts From 0 to 9

```
for ( i = 0;  i < 10;  i = i + 1 )
{
  printf (“%d \n”, i) ;
}
```

We Can Count Backwards, Too

```
for ( i = 9; i >= 0; i = i - 1 )
{
  printf (“%d \n”, i) ;
}
```

We Can Count By 2's … or 7's … or Whatever

```
for ( i = 0; i < 10; i = i + 2 )
{
  printf (“%d\n”, i) ;
}
```

The do-while Repetition Structure

```
do {
   statement(s)
} while ( condition ) ;
```

- The body of a **do-while** is ALWAYS executed at least once.
  - Is this true of a **while** loop?
  - What about a **for** loop?

Example

```
do {
  printf (“Enter a positive number: “);
  scanf (“%d”, &num) ;

  if ( num <= 0 ) {
     printf (“\n Not positive. Try again\n”);
  }

} while ( num <= 0 ) ;
```

An Equivalent while Loop

```
printf (“Enter a positive number: “) ;
scanf (“%d”, &num) ;

while ( num <= 0 ) {
  printf (“\nNot positive. Try again\n”) ;
  printf (“Enter a positive number: “) ;
  scanf (“%d”, &num) ;
}
```

- Notice that using a while loop in this case requires a priming read.

An Equivalent for Loop

```
printf (“Enter a positive number: “) ;
scanf (“%d”, &num) ;

for ( ; num <= 0; ) {
     printf (“\nNot positive. Try again\n”) ;
     printf (“Enter a positive number: “) ;
     scanf (“%d”, &num) ;
}
```

• A for loop is a very awkward choice here because the loop is event-controlled.

So, Which Type of Loop Should I Use?

- **for** loop
  - for counter-controlled repetition.
- **while** or **do-while** loop
  - for event-controlled repetition.
    - Use a **do-while** loop when the loop must execute at least once.
    - Use a **while** loop when it is possible that body of the loop may never execute.

Nested Loops

- Loops may be **nested** (**embedded**) inside of each other.
- Actually, any control structure (sequence, selection, or repetition) may be nested inside of any other control structure.
- It is common to see nested for loops.

Nested for Loops

- How many times is the "if" statement executed?
- What is the output ?

```
for ( i = 1; i < 5; i = i + 1 )
{
    for ( j = 1; j < 3; j = j + 1 )
   {
        if ( j % 2 == 0 )
        {
            printf ("O");
        } else {
            printf ("X") ;
        }
    }
    printf ("\n") ;
}
```

The break Statement

- The **break** statement can be used in **while**, **do-while**, and **for** loops to cause premature exit of the loop.

- THIS IS ***NOT*** A RECOMMENDED CODING TECHNIQUE.

Example break in a for Loop

- What is the output ?

```
#include <stdio.h>

int main ( )
{
  int i ;
  for ( i = 1; i < 10; i = i + 1 )
  {
     if (i == 5) {
         break ;
     }
     printf (“%d “, i) ;
  }
  printf (“\nBroke out of loop at i = %d.\n”, i) ;
  return 0 ;
}
```

The continue Statement

- The **continue** statement can be used in **while**, **do-while**, and **for** loops.
- It causes the remaining statements in the body of the loop to be skipped for the current iteration of the loop.
- THIS IS ***NOT*** A RECOMMENDED CODING TECHNIQUE.

Example continue in a for Loop

- What is the output ?

```
#include <stdio.h>

int main ( )
{
   int i ;
   for ( i = 1; i < 10; i = i + 1 ) {
      if (i == 5) {
          continue ;
      }
      printf (“%d ”, i) ;
   }
   printf (“\nDone.\n”) ;
   return 0 ;
}
```