Structured Programming

Dr. Mohamed Khedr

Lecture 9

http://webmail.aast.edu/~khedr

Two Types of Loops

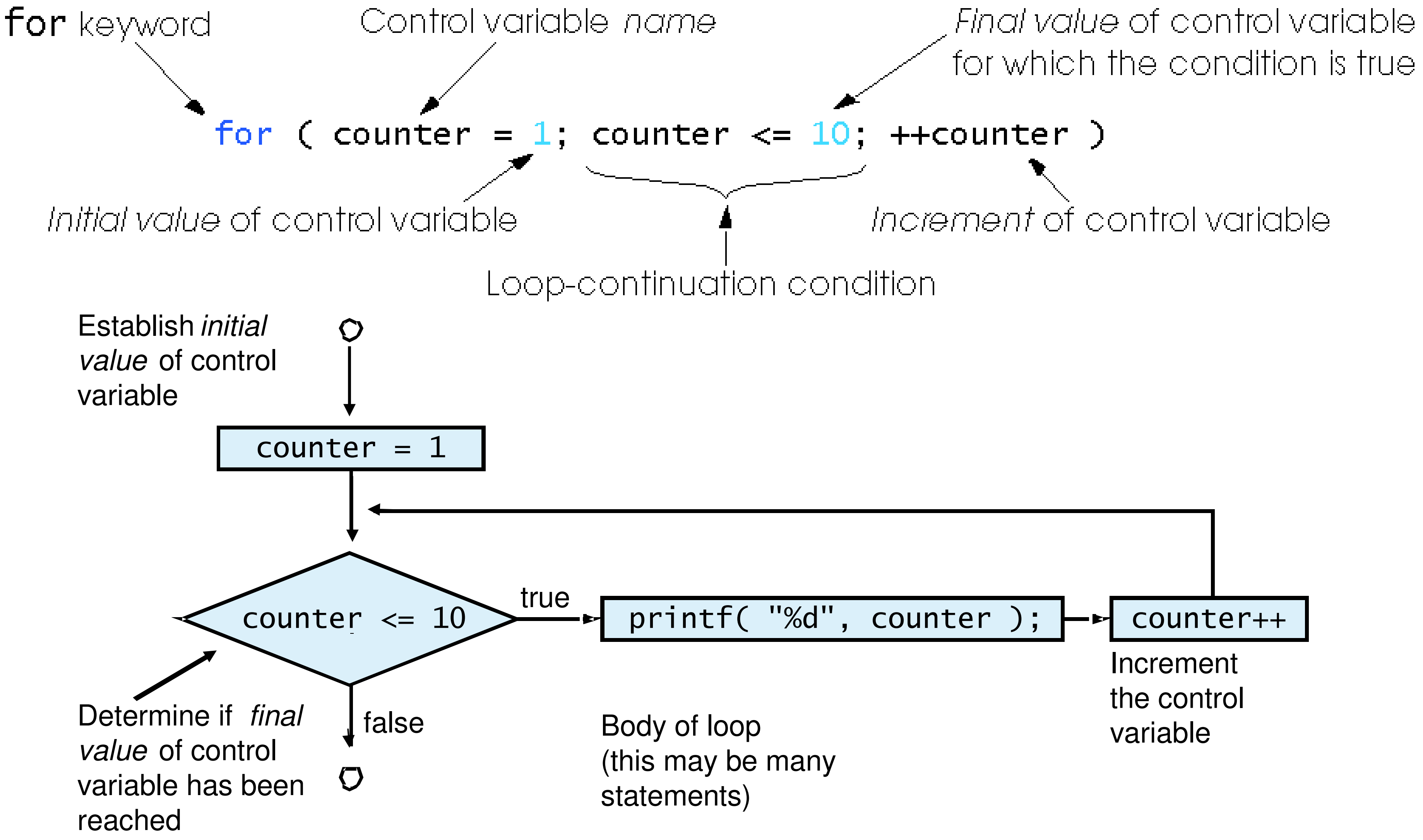
count controlled loops

repeat a specified number of times

event-controlled loops

some condition within the loop body changes and this causes the repeating to stop

The `for` Repetition Statement



```
for ( counter = 1; counter <= 10; ++counter )
```

`for` keyword

Control variable *name*

Final value of control variable for which the condition is true

Initial value of control variable

Increment of control variable

Loop-continuation condition

Establish *initial value* of control variable

`counter = 1`

`counter <= 10`

true

`printf( "%d", counter );`

`counter++`

false

Determine if *final value* of control variable has been reached

Body of loop (this may be many statements)

Increment the control variable

Repetition Structure: `for`

- `for` loops syntax

  for (initialization ; loopContinuationTest ; increment)
  statement

  Example: Prints the integers from one to ten

```
for ( counter = 1; counter <= 10; counter++ )
  printf( "%d\n", counter );
```

No semicolon (`;`) after last expression

- For loops can usually be rewritten as `while` loops:

```
initialization;
while ( loopContinuationTest ) {
  statement;
  increment;
}
```

- Initialization and increment
  - Can be comma-separated list of statements

  Example:

```
for ( i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```

The `for` Structure (cont.)

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If **`x`** equals **`2`** and **`y`** equals **`10`**

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

  is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

- Notes about the **`for`** structure:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially **`false`**
    - The body of the **`for`** structure is not performed (i.e. pre-test)
    - Control proceeds with the next statement after the **`for`** structure
  - Control variable
    - Often printed or used inside for body, but not necessarily

The `for` Structure (cont.)

```c
/* Fig. 4.5: fig04_05.c
   Summation with for */
#include <stdio.h>

int main()
{
   int sum = 0, number;

   for ( number = 2; number <= 100; number += 2 )
      sum += number;

   printf( "Sum is %d\n", sum );

   return 0;
}
```

1. Initialize variables

2. `for` repetition structure

Program Output:

```
Sum is 2550
```

2 + 4 + 8 + … +100 = 2550

for = = while

-
  ```
  for(expr1;expr2;expr3)
     statement;
  ```
  - Is equivalent to:

-
  ```
  expr1;
  while(expr2){
     statement;
     expr3;
  }
  ```
  - This will create an infinite loop:

-
  ```
  for(;;){ . . .}
  ```

Comma in For Loops

- Can put commas in for loops
- Evaluated left to right

```
#include <stdio.h>
int main()
{
    int a=0,b=0;
    for(a=0, b=10; a<b; a++, b--)
      printf("a=%d b=%d\n",a,b);
    return 0;
```

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )
{
     printf(   “ % d Potato \n ” , num );
}
```

num ?

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

    printf(   “ % d Potato \n ” , num );
```

OUTPUT

num 1

Example of Repetition

```
int   num;

for  ( num = 1 ;  num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

num 1

Example of Repetition

true

```
int   num;

for  ( num = 1 ;  num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

num 1

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato
```

num 2

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato
```

num 2

Example of Repetition

```
int   num;
                              true
for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato
```

num 2

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato

2Potato
```

num 3

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato

2Potato
```

num 3

Example of Repetition

```
int   num;
                               true
for  ( num = 1 ;  num <= 3 ;  num++ )

     printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato

2Potato
```

num 3

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ;  num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato

2Potato

3Potato
```

num 4

Example of Repetition

```
int   num;

for  ( num = 1 ;   num <= 3  ; num++ )

      printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato
2Potato
3Potato
```

num 4

Example of Repetition

```
int   num;
                              false
for  ( num = 1 ;   num <= 3  ;  num++ )

     printf(   “ % d Potato \n ” , num );
```

OUTPUT

```
1Potato
2Potato
3Potato
```

num 4

Example of Repetition

false

```
int   num;

for  ( num = 1 ;  num <= 3  ;  num++ )

      printf(   " % d Potato \n " , num );
```

When the loop control condition is evaluated and has value false, the loop is said to be "satisfied" and control passes to the statement following the For statement.

The output was:

```
1Potato
2Potato
3Potato
```

Count-controlled Loop

```
int  count ;

for  ( count = 4 ; count > 0 ; count-- )
{
        printf( “ %d \n “ ,  count  ) ;
}

printf (  “Done”  ) ;
```

OUTPUT:

```
4
3
2
1
Done
```

What is the output?

```
int  count;

for ( count = 0 ;  count < 10 ;  count++ )
{
            printf(  “*”  );
}
```

OUTPUT

```
**********
```

NOTE: the 10 asterisks are all on one line. Why?

For Loop Variations

- any expression may be more complex:

```
for (i=100*y; i>=1; i--)
for (i=100; i>=y/2; i--)
for (i=100; i>=1; i-=4*y)
```

For Loop Variations

- Increment may be negative:

```
for (i=100; i>=1; i--)
```

  - This counts from 100 to 1.

- Increment may be greater than 1:

```
for (i=100; i>=5; i-=5)
```

  - This counts from 100 to 5 in steps of 5

What output from this loop?

```
int  count;

for  (count = 0;  count < 10;  count++) ;
{
          printf (  “*”  );
}
```

OUTPUT

- One * ! Why?
- the ; right after the ( ) means that the body statement is a null statement
- in general, the Body of the for loop is whatever statement *immediately* follows the ( )
- that statement can be a single statement, a block, or a null statement
- actually, the code outputs one * after the loop completes its counting to 10

Infinite Loop

- You can still end up with an infinite loop when using for loops

```
for (counter=0; counter<=10; counter--)
```

```c
/* Fig. 4.6: fig04_06.c
   Calculating compound interest */
#include <stdio.h>
#include <math.h>

/* function main begins program execution */
int main( void )
{
   double amount;              /* amount on deposit */
   double principal = 1000.0; /* starting principal */
   double rate = .05;          /* annual interest rate */
   int year;                   /* year counter */

   /* output table column head */
   printf( "%4s%21s\n", "Year", "Amount on deposit" );

   /* calculate amount on deposit for each of ten years */
   for ( year = 1; year <= 10; year++ ) {

      /* calculate new amount for specified year */
      amount = principal * pow( 1.0 + rate, year );

      /* output one table row */
      printf( "%4d%21.2f\n", year, amount );
   } /* end for */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

additional header

pow function calculates the value of the first argument raised to the power of the second argument

Outline

```
Year     Amount on deposit
   1               1050.00
   2               1102.50
   3               1157.63
   4               1215.51
   5               1276.28
   6               1340.10
   7               1407.10
   8               1477.46
   9               1551.33
  10               1628.89
```

Nested For Loops

- It is also possible to place a for loop inside another for loop.

```
int rows, columns;

for (rows=1; rows<=5; rows++)
{
   for (columns=1; columns<=10; columns++)
   {
         printf ("*");
   }
   printf ("\n");
}
```

Outer Loop

Inner Loop

Output:

```
**********
**********
**********
**********
**********
```

Nested For Loops, Example #2

```
#include <stdio.h>

main ()
{
   int rows, columns;

   for (rows=1; rows<=5; rows++)
   {
        for (columns=1; columns<=rows; columns++)
        {
                printf ("*");
        }
        printf ("\n");
   }
}
```

Outer Loop

Inner Loop

Output:

```
*
**
***
****
*****
```

Nested Loops

```
initialize outer loop
while  ( outer loop condition )
{         . . .

          initialize inner loop
          while ( inner loop condition )
          {
                    inner loop processing and update
          }
          . . .
}
```

Example

Write a program that **displays the multiplication tables (1 - 12).**

```
 1  x  1  =  1
 1  x  2  =  2
 ....
1   x  12  =  12
2  x  1  =   2
 ....
 12 x 12  = 144
```