C Programming for Engineers

UNIVERSITY AT ALBANY
State University of New York

Iteration

ICEN 360– Spring 2017

Prof. Dola Saha

Data type conversions

- Grade average example
  - $class\ average = \frac{\sum grade}{number\ of\ students}$
  - Grade and number of students can be integers
  - Averages do not always evaluate to integer values, needs to be floating point for accuracy.
  - The result of the calculation `total / counter` is an integer because `total` and `counter` are both integer variables.

Explicit conversions

- Dividing two integers results in integer division in which any fractional part of the calculation is truncated (i.e., lost).
- To produce a floating-point calculation with integer values, we create temporary values that are floating-point numbers.
- C provides the unary cast operator to accomplish this task.
  - `average = ( float ) total / counter;`
- includes the cast operator `(float)`, which creates a temporary floating-point copy of its operand, `total`.
- Using a cast operator in this manner is called explicit conversion.
- The calculation now consists of a floating-point value (the temporary `float` version of `total`) divided by the `unsigned int` value stored in `counter`.

Implicit conversion

- C evaluates arithmetic expressions only in which the data types of the operands are *identical*.
- To ensure that the operands are of the *same* type, the compiler performs an operation called implicit conversion on selected operands.
- For example, in an expression containing the data types `unsigned int` and `float`, copies of `unsigned int` operands are made and converted to `float`.
- In our example, after a copy of `counter` is made and converted to `float`, the calculation is performed and the result of the floating-point division is assigned to `average`.

Assignment operators

- C provides several assignment operators for abbreviating assignment expressions.
- For example, the statement
  - `c = c + 3;`
- can be abbreviated with the addition assignment operator `+=` as
  - `c += 3;`
- The += operator
  - adds the value of the expression on the right of the operator to the value of the variable on the left of the operator
  - and stores the result in the variable on the left of the operator.

Comparison of Prefix & Postfix Increments

Before...

i	j
2	?

Increments...

```
j = ++i;
```

prefix:
Increment `i` and then use it.

```
j = i++;
```

postfix:
Use `i` and then increment it.

After...

i	j
3	3

i	j
3	2

Assignment operators

- Any statement of the form
  - *variable = variable operator expression;*
- where *operator is one of the binary operators +, -, *, / or % (or others we'll discuss in Chapter 10), can be written in the form*
  - *variable operator = expression;*
- Thus the assignment c += 3 adds 3 to c.

Assignment operator - examples

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Unary Increment & Decrement Operators

Operator	Sample expression	Explanation
`++`	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Increment Example

```c
// Fig. 3.13: fig03_13.c
// Preincrementing and postincrementing.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int c; // define variable

   // demonstrate postincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", c++ ); // print 5 then postincrement
   printf( "%d\n\n", c ); // print 6

   // demonstrate preincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", ++c ); // preincrement then print 6
   printf( "%d\n", c ); // print 6
} // end function main
```

Increment Example

```c
// Fig. 3.13: fig03_13.c
// Preincrementing and postincrementing.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int c; // define variable

   // demonstrate postincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", c++ ); // print 5 then postincrement
   printf( "%d\n\n", c ); // print 6

   // demonstrate preincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", ++c ); // preincrement then print 6
   printf( "%d\n", c ); // print 6
} // end function main
```

Output

```
5
5
6

5
6
6
```

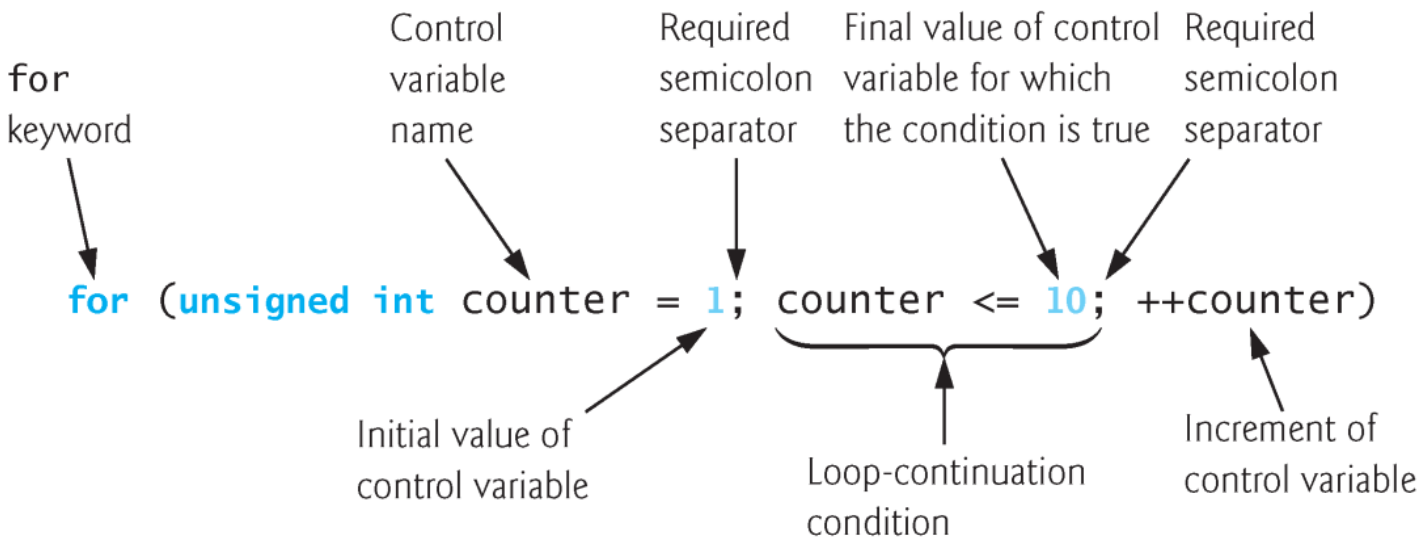
Precedence

Operators	Associativity	Type
++ *(postfix)* -- *(postfix)*	right to left	postfix
+ - (*type*) ++ *(prefix)* -- *(prefix)*	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

for Iteration Statement - Syntax

```
for (initialization expression;
        loop repetition condition;
        update expression)
      statement;
```

```
for  (count_star = 0;
        count_star < N;
        count_star ++)
     printf("*");
```



`for` Iteration Statement - Syntax

- The general format of the `for` statement is

```
for (initialization; condition; update expression) {
  statement
}
```

where

- the initialization expression initializes the loop-control variable (and might define it),
- the condition expression is the loop-continuation condition and
- the update expression increments the control variable.

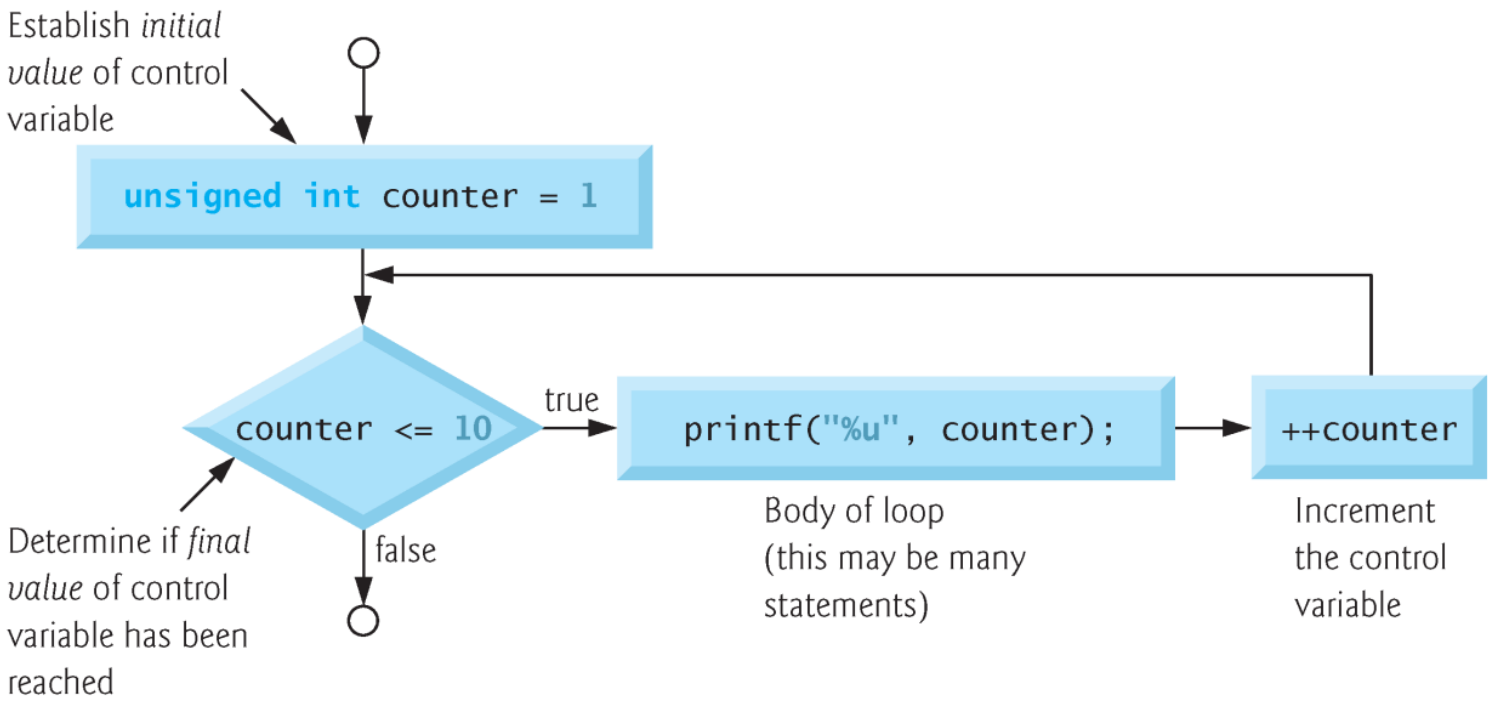
for Iteration Statement

- Counter-controlled iteration

```c
// Fig. 4.2: fig04_02.c
// Counter-controlled iteration with the for statement.
#include <stdio.h>

int main(void)
{
   // initialization, iteration condition, and increment
   //  are all included in the for statement header.
   for (unsigned int counter = 1; counter <= 10; ++counter) {
      printf("%u\n", counter);
   }
}
```

Flow chart


Establish *initial value* of control variable
unsigned int counter = 1
counter <= 10
true
printf("%u", counter);
++counter
Determine if *final value* of control variable has been reached
false
Body of loop (this may be many statements)
Increment the control variable

`for` Iteration Statement – Common Error

Off-By-One Errors

- Notice that program uses the loop-continuation condition `counter <= 10`.
- If you incorrectly wrote `counter < 10`, then the loop would be executed only 9 times.
- This is a common logic error called an off-by-one error.

for Iteration Statement – Common Practice

- Start the loop from 0

```
for (i=0; i<10; i++)
     printf("It will be printed 10 times.\n");
for (i=1; i<=10; i++)
     printf("It will be printed 10 times.\n");
```

Optional Header in `for` Statement

- The three expressions in the `for` statement are optional.
- If the *condition* expression is omitted, C assumes that the condition is true, thus creating an infinite loop.
- You may omit the *initialization* expression if the control variable is initialized elsewhere in the program.
- The *increment* may be omitted if it's calculated by statements in the body of the `for` statement or if no increment is needed.

Valid code snippets

```
for (;;)
      printf("The code is in infinite loop\n");
```

```
int i=0;
for (; i<10; i++)
      printf("It will be printed 10 times.\n");
```

```
int i=0;
for (; i<10; ){
      printf("It will be printed 10 times.\n");
      i++;
}
```

Examples of varying control variable

Task	
Vary the control variable from 1 to 100 in increments of 1.	
Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).	
Vary the control variable from 7 to 77 in steps of 7.	
Vary the control variable from 20 to 2 in steps of -2.	
Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.	
Vary the control variable over the following sequence of values: 44, 33, 22, 11, 0.	

Examples of varying control variable

Task	for Loop
Vary the control variable from 1 to 100 in increments of 1.	`for (i = 1; i <= 100; ++ i)`
Vary the control variable from 100 to 1 in increments of -1 (decrements of 1).	`for (i = 100; i >= 1; --i)`
Vary the control variable from 7 to 77 in steps of 7.	`for (i = 7; i <= 77; i += 7)`
Vary the control variable from 20 to 2 in steps of -2.	`for (i = 20; i >= 2; i -= 2)`
Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.	`for (j = 2; j <= 17; j += 3)`
Vary the control variable over the following sequence of values: 44, 33, 22, 11, 0.	`for (j = 44; j >= 0; j -= 11)`

For Statement Notes

- The initialization, loop-continuation condition and update expression can contain **arithmetic expressions**.

- For example, if `x` = 2 and `y` = `10`, the statement

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

  is equivalent to the statement

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

- If the loop-continuation condition is initially false, the loop body does not execute.

Nested for Loop

```
int row, col;
for (row=0; row<2; row++)
     for (col=0; col<3; col++)
          printf("%d, %d\n", row, col);
```

Nested for Loop

```
int row, col;
for (row=0; row<2; row++)
     for (col=0; col<3; col++)
          printf("%d, %d\n", row, col);
```

Sample Output

```
0, 0
0, 1
0, 2
1, 0
1, 1
1, 2
```

Application: Summing even numbers

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

int main(void)
{
   unsigned int sum = 0; // initialize sum

   for (unsigned int number = 2; number <= 100; number += 2) {
      sum += number; // add number to sum
   }

   printf("Sum is %u\n", sum);
}
```

```
Sum is 2550
```

Application: Compound Interest Calculation

- Consider the following problem statement:
  - A person invests $1000.00 in a savings account yielding 5% interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts:

    $a = p(1 + r)^n$

    where

    p is the original amount invested (i.e., the principal)
    r is the annual interest rate
    n is the number of years
    a is the amount on deposit at the end of the nth year.

C Code for Compound Interest Calculation

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

int main(void)
{
   double principal = 1000.0; // starting principal
   double rate = .05; // annual interest rate

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

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

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

      // output one table row
      printf("%4u%21.2f\n", year, amount);
   }
}
```

Output

```
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
```

Classwork Assignment

- Write a program that finds the smallest of several integers. Assume that the first value read specifies the number of values remaining. Your program should read only one value each time scanf is executed.

- A typical input sequence might be
  - 5 400 500 300 200 100
  - where 5 indicates that the subsequent five values are to be used for finding minimum.

Classwork Assignment

- Write a program that prints the following patterns separately, one below the other. Use for loops to generate the patterns. [*Hint:* The last two patterns require that each line begin with an appropriate number of blanks.]

```
(A)          (B)          (C)          (D)
*            **********    **********            *
**           *********      *********           **
***          ********        ********          ***
****         *******          *******         ****
*****        ******            ******        *****
******       *****              *****       ******
*******      ****                ****      *******
********     ***                  ***     ********
*********    **                    **    *********
**********   *                      *   **********
```

do ... while Iteration Statement

- Similar to the `while` statement.
- `while` (*condition*)
- The loop-continuation condition is tested at the beginning of the loop
- `do`
  statement
  `while` (*condition*);
- The loop-continuation condition *after* the loop body is performed.
- The loop body will be executed at least once.

Example do ... while Iteration Statement

```c
// Fig. 4.9: fig04_09.c
// Using the do...while iteration statement.
#include <stdio.h>

int main(void)
{
   unsigned int counter = 1; // initialize counter

   do {
      printf("%u  ", counter);
   } while (++counter <= 10);
}
```

```
1  2  3  4  5  6  7  8  9  10
```

Flowchart do ... while Iteration Statement