Chapter 5
Program Looping

Objectives

- Basic Loop Structures
- The `while` Statement
- Computing Sums and Averages Using a `while` Loop
- The `for` Statement

Objectives (continued)

- Case Studies: Loop Programming Techniques
- Nested Loops
- The `do-while` Statement
- Common Programming and Compiler Errors

Introduction

- A section of code that is repeated is called a **loop**, because after the last statement in the code is executed, the program branches, or loops, back to the first statement and starts another repetition through the code
- Each repetition is also called an **iteration** or **pass through the loop**

Basic Loop Structures

- Constructing a repeating section of code requires that four elements be present:
  - Repetition statement
    - `while` statement
    - `for` statement
    - `do-while` statement
  - Condition
  - A statement that initially sets the condition being tested
  - A statement within the repeating section of code that alters the condition so that it eventually becomes false

Counter-Controlled and Condition-Controlled Loops

- **Counter-controlled loop:** the condition is used to keep track of the number of repetitions
  - Also known as a **fixed-count loop**
- **Condition-controlled loop:** the tested condition does not depend on a count being achieved, but rather on a specific value being encountered

Basic Loop Structures

Table 5.1 Comparison of loop types

Type of Loop	Description
Counter-controlled (Fixed-count)	The number of repetitions is known before the loop executes.
Condition-controlled	The number of repetitions is not known before the loop executes. The loop is terminated when one or more specific values are encountered.
Sentinel-controlled	This is a condition-controlled loop where one specific value is required to terminate the loop.
Input-validation	This is a condition-controlled loop that terminates when a value within a valid range is entered.

The `while` Statement

- The general form of the `while` statement is

```
while (expression)
  statement;
```

- The transfer of control back to the start of a `while` statement to reevaluate the expression is known as a **program loop**
- The following is a valid but infinite loop:

```
while (count <= 10)
  printf("%d ",count);
```

The `while` Statement (continued)

Program 5.1

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int count;
 5
 6    count = 1; /* initialize count */
 7    while (count <= 10)
 8    {
 9      printf("%d ",count);
10      count++; /* add 1 to count */
11    }
12
13    printf("\n");  /* print a blank line */
14
15    return 0;
16  }
```

Output is:

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

The `while` Statement (continued)

Program 5.2

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int i;
 5
 6    i = 10;
 7    while (i >= 1)
 8    {
 9      printf("%d ",i);
10      i--; /* subtract 1 from i */
11    }
12
13    printf("\n");   /* print a blank line */
14
15    return 0;
16  }
```

Output is:

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

The `while` Statement (continued)

Program 5.3

```
 1  #include <stdio.h>
 2  int main ()
 3  {
 4    #define TABLESIZE 10
 5    int num;
 6
 7    printf("NUMBER SQUARE CUBE\n");
 8    printf("------ ------ ----\n");
 9    num = 1;
10    while (num <= TABLESIZE)
11    {
12      printf("%3d %7d %6d\n", num, num*num, num*num*num);
13
14      num++; /* add 1 to num */
15    }
16
17    return 0;
18  }
```

Output is:

```
NUMBER  SQUARE  CUBE
------ ------ ----
   1       1     1
   2       4     8
   3       9    27
   4      16    64
   5      25   125
   6      36   216
   7      49   343
   8      64   512
   9      81   729
  10     100  1000
```

The `while` Statement (continued)

Program 5.4

```
1  #include <stdio.h>
2   #define ENDVALUE 50
3  int main() /* program to convert Celsius to Fahrenheit */
4  {
5    int celsius;
6    float fahren;
7
8    /* display the heading lines */
9    printf("DEGREES DEGREES\n");
10   printf("CELSIUS FAHRENHEIT\n");
11   printf("------- ----------\n");
12
13   // now fill in the table using a while loop
14   celsius = 5; /* starting Celsius value */
15
16   while (celsius <= ENDVALUE)
17   {
18     fahren = (9.0/5.0) * celsius + 32.0;
19     printf("%5d%11.2f\n",celsius, fahren);
20     celsius = celsius + 5;
21   }
22
23   return 0;
24 }
```

Condition-controlled loop

Output is:

```
DEGREES  DEGREES
CELSIUS  FAHRENHEIT
-------  ----------
      5       41.00
     10       50.00
     15       59.00
     20       68.00
     25       77.00
     30       86.00
     35       95.00
     40      104.00
     45      113.00
     50      122.00
```

Computing Sums and Averages Using a `while` Loop

Program 5.5

```
 1  #include <stdio.h>
 2  #define MAXCOUNT 4
 3  int main()
 4  {
 5    int count;
 6    float num;
 7
 8    printf("\nThis program will ask you to enter %d numbers.\n\n", MAXCOUNT);
 9
10    count = 1;
11    while (count <= MAXCOUNT)
12    {
13      printf("Enter a number: ");
14      scanf("%f", &num);
15      printf("The number entered is %f\n", num);
16      count++;
17    }
18
19    return 0;
20  }
```

Computing Sums and Averages Using a `while` Loop (continued)

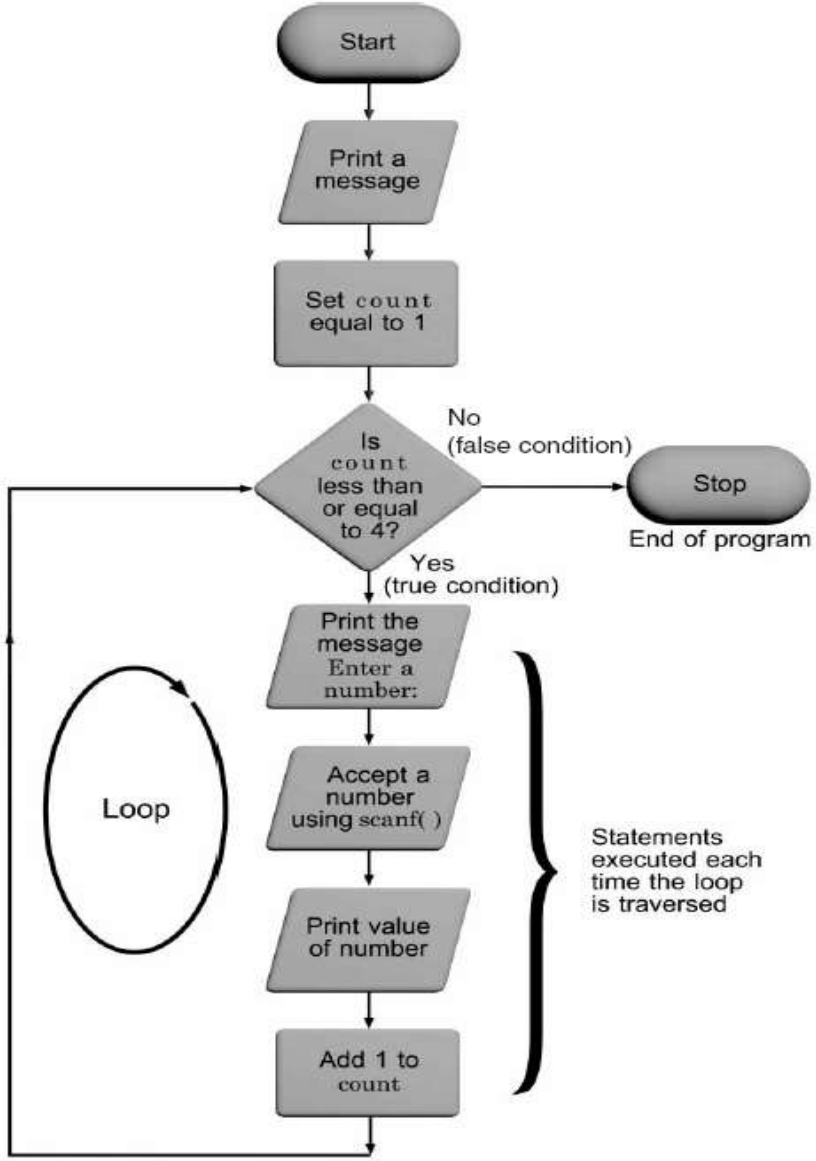


Figure 5.4 Flow-of-control diagram for Program 5.5

Computing Sums and Averages Using a `while` Loop (continued)

Program 5.6

```
1  #include <stdio.h>
2   #define MAXCOUNT 4
3  int main()
4  {
5    int count;
6    float num, total;
7
8    printf("\nThis program will ask you to enter %d numbers.\n\n", MAXCOUNT);
9
10   count = 1;
11   total = 0.0;
12
13   while (count <= MAXCOUNT)
14   {
15     printf("Enter a number: ");
16     scanf("%f", &num);
17     total += num;
18     printf("The total is now %f\n", total);
19     count++;
20   }
21
22   printf("\n\nThe final total of the %d numbers is %f\n", MAXCOUNT, total);
23
24   return 0;
25 }
```

Ensures that any previous value present in the storage locations assigned to the variable `total` is overwritten and the `total` starts at a correct value

Accumulating statement

Computing Sums and Averages Using a `while` Loop (continued)

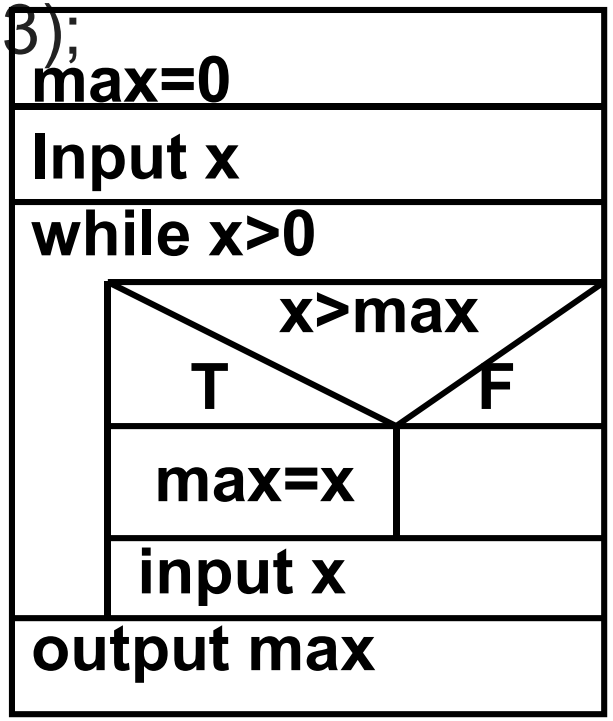
Program 5.7

```
 1  #include <stdio.h>
 2   #define MAXCOUNT 4
 3  int main()
 4  {
 5    int count;
 6    float num, total, average;
 7
 8    printf("\nThis program will ask you to enter %d numbers.\n\n", MAXCOUNT);
 9
10    count = 1;
11    total = 0.0;
12
13    while (count <= MAXCOUNT)
14    {
15      printf("Enter a number: ");
16      scanf("%f", &num);
17      total += num;
18      count++;
19    }
20
21    average = total / MAXCOUNT;
22    printf("\nThe average of the %d numbers is %8.4f\n", MAXCOUNT, average);
23
24    return 0;
25  }
```

Calculating an average

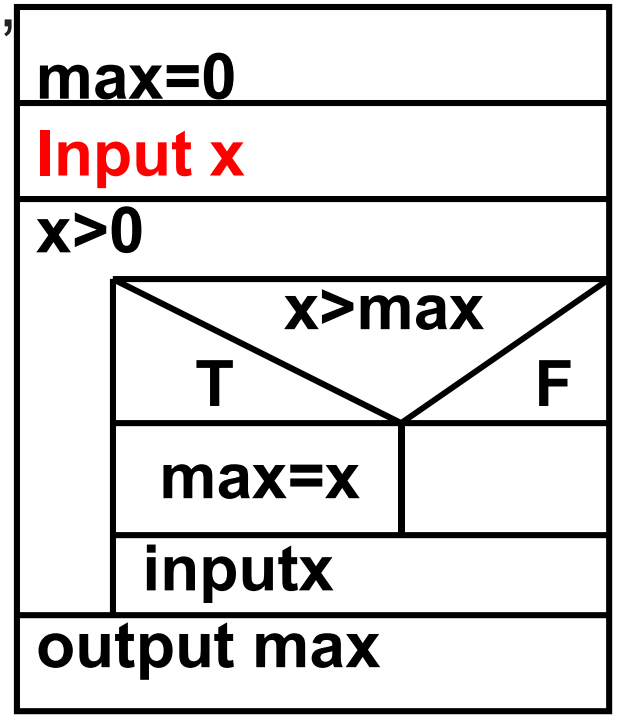
【ex. 5.2】 Find the max one among some positive number that input from keybord

- review： Find the max one among three numbers a,b,c.
- main()
- { float n1,n2,n3,max;
- scanf(″ %f%f%f ″ ,&n1,&n2,&n3);
- max=n1;
- if(n2>max) max=n2;
- if(n3>max) max=n3;
- printf(″The largest number is
- %.2f\n″,max);
- }



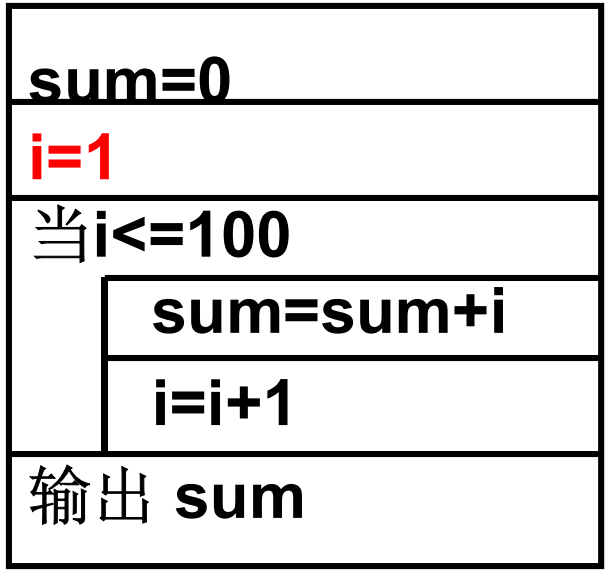
```
•main( )
•{  float x,max;
•    max=0;
•    printf("x="); scanf("%f",&x);
•    while (x>0)
•          {  if (x>max)  max=x;
•              printf("x="); scanf("%f",&x);
•           }
•     printf("max=%-8.2f",max);
• }
```

• question:Give the serial number of input numbers,and the count of all the numbers.



【ex. 5.2】Compute $\sum_{i=1}^{100} i$ = 1+2+3+…+100，use while statement 。

- int i,sum;
- sum=0;
- i=1;
- while (i<=100)
- { sum=sum+i;
- i=i+1;
- }
- printf(“1+2+3+...+100=%d”,sum);




```
for (i=1;i<=100;i++)
  sum=sum+i;
```

The `break` and `continue` Statements

- A `break` forces an immediate exit from `while`, `switch`, `for`, and `do-while` statements only

```
while(count <= 10)
{
  printf("Enter a number: ");
  scanf("%f", &num);
  if (num > 76)
  {
    printf("You lose!");
    break; /* break out of the loop */
  }
  else
    printf("Keep on truckin!");
}
/* break jumps to here */
```

The `break` and `continue` Statements (continued)

- The `continue` applies to loops only; when a `continue` statement is encountered in a loop, the next iteration of the loop begins immediately

```
while (count < 30)
{
  printf("Enter a grade: ");
  scanf("%f", &grade);
  if(grade < 0 || grade > 100)
    continue;
  total = total + grade;
  count = count + 1;
}
```

The Null Statement

- A semicolon with nothing preceding it is also a valid statement, called the **null statement**

```
;
```

- Use the null statement where a statement is syntactically required, but no action is needed
- Null statements typically are used either with `while` or `for` statements

The `do-while` Statement (continued)

- The general form of the `do` statement is

```
do
   statement;
while (expression);
```

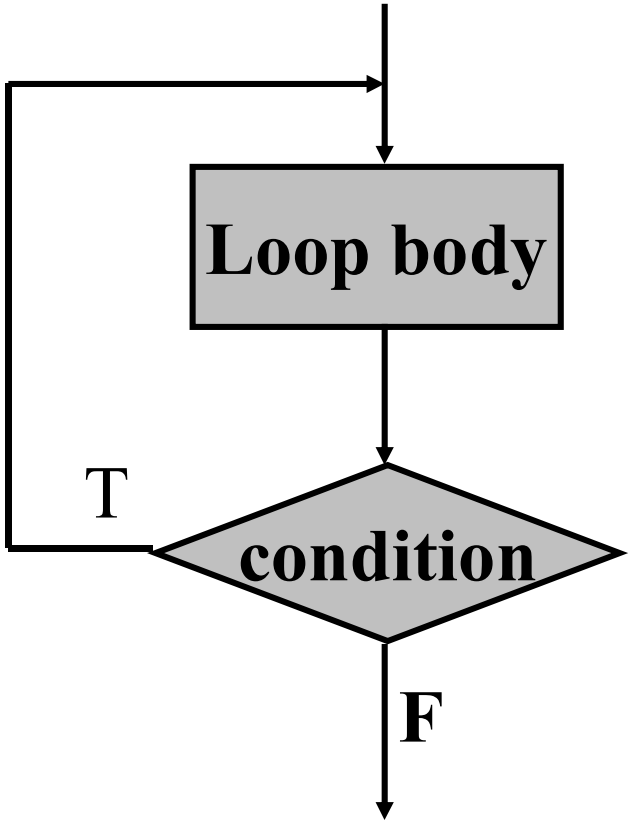
- `do-while` is a posttest loop
- One type of application is ideally suited for a posttest loop:
  - **Input data validation application**

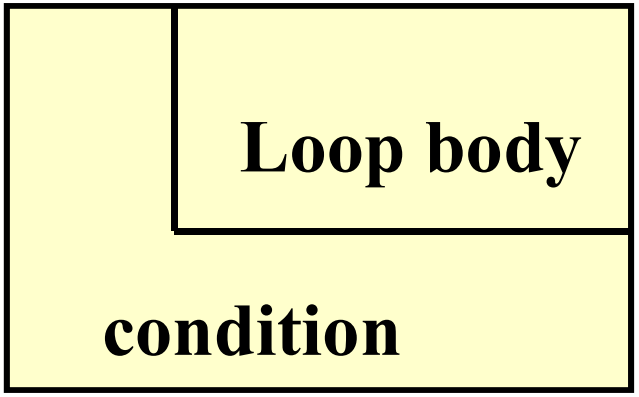
5.4 Do While Loop

- 1 .Form do
- <statements>
- while <expression>
- 2 .Function

Loop body

condition





The `do-while` Statement (continued)

```
do
{
  printf("\nEnter an ID number: ");
  scanf("%f", &idNum);
} while (idNum < 1000 || idNum > 1999);
```

The `for` Statement

- The `for` statement combines all four elements required to easily produce a loop on the same line

```
for(initializing list; tested expression; altering list)
    statement;
```

- This statement does not require that any of the items in parentheses be present or that they actually be used for initializing or altering the values in the expression statements
  - However, the two semicolons must be present
    - `for ( ; count <= 20; )` is valid
    - Omitting tested expression results in infinite loop

The `for` Statement (continued)

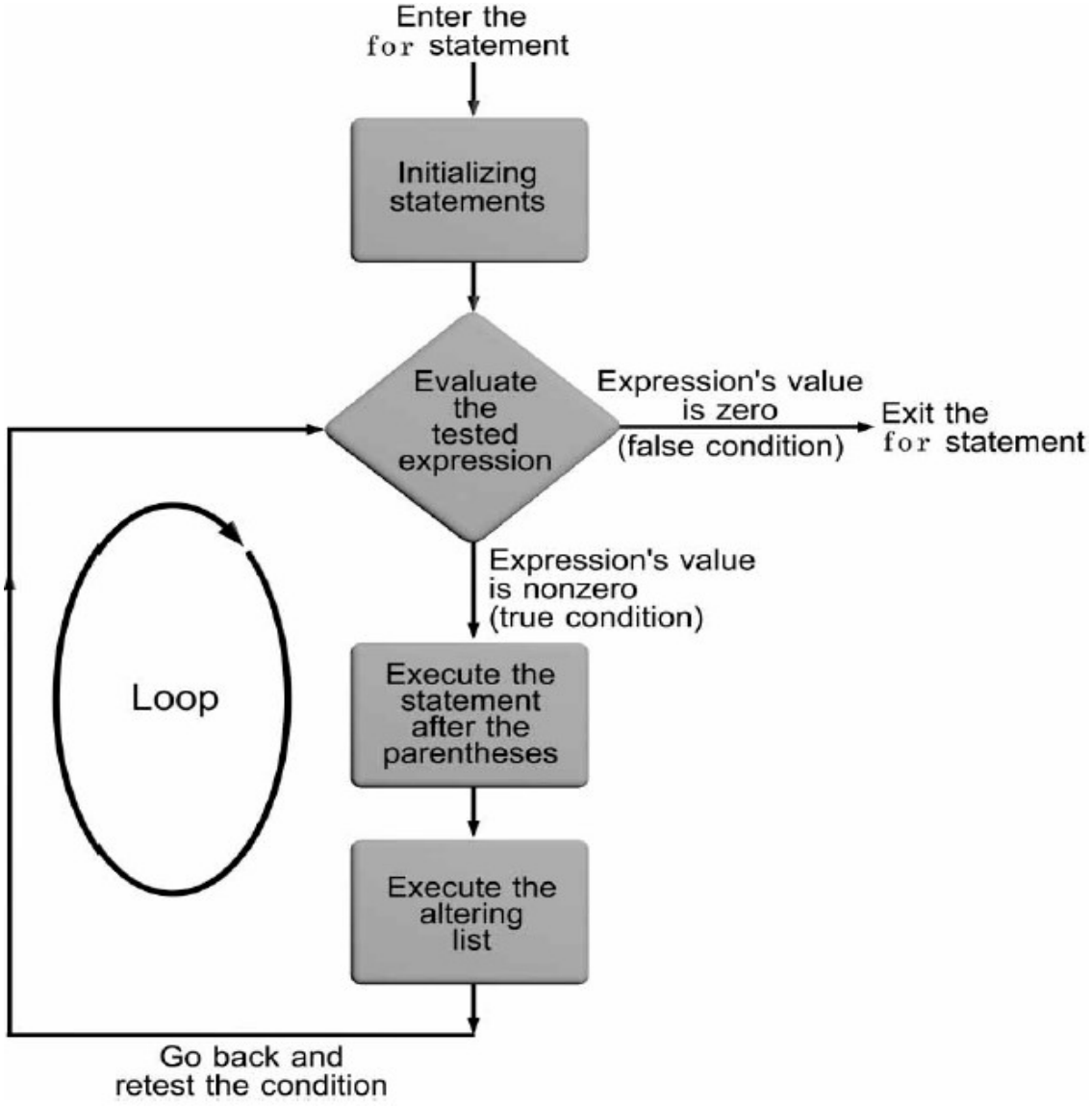


Figure 5.8 `for` statement flow of control

The `for` Statement (continued)

Program 5.10

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define MAXCOUNT 20
 5    int count;
 6
 7    for (count = 2; count <= MAXCOUNT; count += 2)
 8      printf("%d ",count);
 9
10    return 0;
11  }
```

- Output is:

```
2 4 6 8 10 12 14 16 18 20
```

The `for` Statement (continued)

Program 5.10a

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int count;
 5
 6    count = 2; /* initializer outside for statement */
 7    for ( ; count <= 20; count += 2)
 8      printf("%d ",count);
 9
10    return 0;
11  }
```

The `for` Statement (continued)

Program 5.10b

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int count;
 5
 6    count = 2; /* initializer outside for statement */
 7    for( ; count <= 20; )
 8    {
 9      printf("%d ",count);
10      count += 2; /* alteration statement */
11    }
12
13    return 0;
14  }
```

The `for` Statement (continued)

Program 5.10c

```
1  #include <stdio.h>
2  int main() /* all expressions within the for's parentheses */
3  {
4    int count;
5
6    for (count = 2; count <= 20; printf("%d ",count), count += 2);
7
8    return 0;
9  }
```

Comma-separated list

The `for` Statement (continued)

Program 5.11 (compare with Program 5.3)

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define TABLESIZE 10
 5    int num;
 6
 7    printf("NUMBER SQUARE CUBE\n");
 8    printf("------ ------ ----\n");
 9
10    for (num = 1; num <= TABLESIZE; num++)
11      printf("%3d %7d %6d\n", num, num*num, num*num*num);
12
13    return 0;
14  }
```

Computing Sums and Averages Using a `for` Loop

Program 5.12

```
 1  #include <stdio.h>
 2  #define MAXCOUNT 5
 3  int main()
 4  /* This program calculates the average */
 5  /* of five user-entered numbers.       */
 6  {
 7    int count;
 8    float num, total, average;
 9
10    total = 0.0;
11
12    for (count = 0; count < MAXCOUNT; count++)
13    {
14      printf("\nEnter a number: ");
15      scanf("%f", &num);
16      total += num;
17    }
18
19    average = total / MAXCOUNT;
20    printf("\n\nThe average of the %d numbers entered is %f\n",
21                                          MAXCOUNT, average);
22
23    return 0;
24  }
```

Case Studies: Loop Programming Techniques

- Technique 1: Selection within a loop
- Technique 2: Input data validation
- Technique 3: Interactive loop control
- Technique 4: Evaluating equations

Technique 1: Selection within a Loop

Program 5.13

```
 1  #include <stdio.h>
 2  #define MAXNUMS 5
 3  int main()
 4  /* this program computes the positive and negative sums of a set */
 5  /* of MAXNUMS user entered numbers                               */
 6  {
 7    int i;
 8    float number;
 9    float postotal = 0.0f;
10    float negtotal = 0.0f;
11
12    for (i = 1; i <= MAXNUMS; i++)
13    {
14      printf("Enter a number (positive or negative) : ");
15      scanf("%f", &number);
16      if (number > 0)
17        postotal += number;
18      else
19        negtotal += number;
20    }
21
22    printf("\nThe positive total is %f", postotal);
23    printf("\nThe negative total is %f\n", negtotal);
24
25    return 0;
26  }
```

Technique 2: Input Data Validation

Program 5.14

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int month;
 5
 6    printf("\nEnter a month between 1 and 12: ");
 7    scanf("%d", &month);
 8
 9    while (month < 1 || month > 12)
10    {
11      printf("Error - the month you entered is not valid.\n");
12      printf("\nEnter a month between 1 and 12: ");
13      scanf("%d", &month);
14    }
15
16    printf("The month accepted is %d\n", month);
17
18    return 0;
19  }
```

Same code used in lines 6-7!

Technique 2: Input Data Validation (continued)

Program 5.15

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define TRUE 1
 5    int month;
 6
 7    while (TRUE)  /* this is always true */
 8    {
 9      printf("\nEnter a month between 1 and 12: ");
10      scanf("%d", &month);
11
12      if (month > 1 && month < 12)  /* the test is made here */
13        break;
14
15      printf("Error - the month you entered is not valid.\n");
16    }
17
18    printf("The month accepted is %d\n", month);
19
20    return 0;
21  }
```

Technique 3: Interactive Loop Control

Program 5.16

```
 1  #include <stdio.h>
 2  int main()
 3  /* this program displays a table of numbers, their squares and cubes */
 4  /* starting from the number 1. The final number in the table is */
 5  /* input by the user */
 6  {
 7    int num, final;
 8
 9    printf("Enter the final number for the table: ");
10    scanf("%d", &final);
11
12    printf("Number Square Cube\n");
13    printf("------ ------ ----\n");
14
15    for (num = 1; num <= final; num++)
16      printf("%3d %7d %6d\n", num, num*num, num*num*num);
17
18    return 0;
19  }
```

Technique 4: Evaluating Equations

Program 5.17

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main()
 4  {
 5    int x, y;
 6
 7    printf("x value   y value\n");
 8    printf("-------   --------\n");
 9
10    for (x = 2; x <= 6; x++)
11    {
12      y = 10 * pow(x,2) + 3 * x - 2;
13      printf("%4d %10d\n", x, y);
14    }
15
16    return 0;
17  }
```

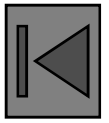
Technique 4: Evaluating Equations (continued)

Program 5.18

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main()
 4  {
 5    float x, y;
 6
 7    printf("x value      y value\n");
 8    printf("--------    ----------\n");
 9
10    for (x = 2.0; x <= 6.0; x += 0.5)
11    {
12      y = 10.0 * pow(x,2) + 3.0 * x - 2.0;
13      printf("%8.6f %13.6f\n", x, y);
14    }
15
16    return 0;
17  }
```

5.5 Nested Loops and Application

- 一、Nested Loops
- • Nested Loops: a loop structure includes another loop
- •explanation：
- (1)loop structure may be one of three kinds
- (2)no limit to counter of nest
- (3)inner loop must be included in outer loop



Nested Loops

Program 5.19

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    int i,j;
 5
 6    for(i = 1; i <= 5; i++)          /* start of outer loop <---+ */
 7    {                                /*                        | */
 8      printf("\ni is now %d\n",i); /*                        | */
 9      for(j = 1; j <=4; j++)         /* start of inner loop    | */
10        printf(" j = %d", j);        /* end of inner loop      | */
11    }                                /* end of outer loop   <---+ */
12
13    return 0;
14  }
```

Nested Loops (continued)

- Sample run:

```
i is now 1
j = 1 j = 2 j = 3 j = 4
i is now 2
j = 1 j = 2 j = 3 j = 4
i is now 3
j = 1 j = 2 j = 3 j = 4
i is now 4
j = 1 j = 2 j = 3 j = 4
i is now 5
j = 1 j = 2 j = 3 j = 4
```

Nested Loops (continued)

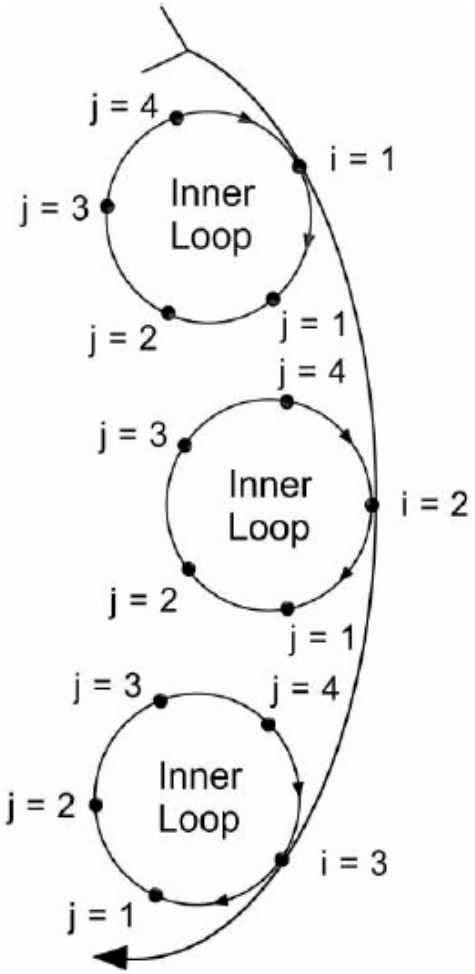


Figure 5.9 `j` loops once for each `i`

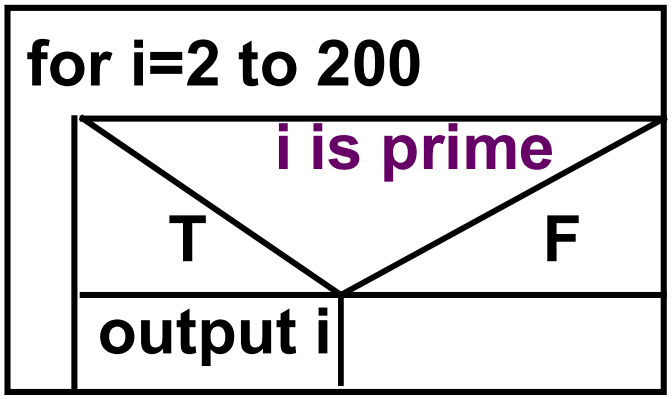
Nested Loops (continued)

Program 5.20

```
 1  #include <stdio.h>
 2    #define NUMSTUDENTS 20
 3    #define NUMGRADES 4
 4  int main()
 5  {
 6    int i,j;
 7    float grade, total, average;
 8
 9    for (i = 1; i <= NUMSTUDENTS; i++) /* start of outer loop */
10    {
11      total = 0; /* clear the total for this student */
12
13      for (j = 1; j <= NUMGRADES; j++) /* start of inner loop */
14      {
15        printf("Enter an examination grade for this student: ");
16        scanf("%f", &grade);
17        total = total + grade; /* add the grade into the total */
18      } /* end of the inner for loop */
19
20      average = total / NUMGRADES; /* calculate the average */
21      printf("\nThe average for student %d is %f\n\n",i,average);
22    } /* end of the outer for loop */
23
24    return 0;
25  }
```

二、Examples

[ex. 5.11] Output all primes between 2～200




```
•main()
{  int  i, j, counter=0;
    for(i=2 ; i<=200; i++)
    {   for(j=2; j<=i-1; j++)
            if(i%j==0)
•               break;
•           if( j == i )
              { printf("%8d",i);
                counter++;
              }
      }

printf("\ncounter=%d\n",counter);
•}
```

【ex. 5.12】Find all daffodil numbers

```
main()
{ int a,b,c,n;
 for(a=1;a<=9;a++)
   for(b=0;b<=9;b++)
     for(c=0;c<=9;c++)
      { n=a*100+b*10+c;
        if (a*a*a+b*b*b+
            c*c*c==n)
         printf("%8d",n);
       }
 printf("\n");
}
```

```
main()
{ int a,b,c,n;
  for(n=100;n<=999;n++)
   {  a=n/100;
      b=n/10%10;
      c=n%10;
      if (a*a*a+b*b*b+
            c*c*c==n)
        printf("%8d",n);
   }
  printf("\n");
}
```

- result：153 370 371 407
-

Common Programming Errors

- “Off by one” error, in which the loop executes either one too many or one too few times than intended
- Using the assignment operator, =, instead of the equality operator, ==, in the tested expression
- As with the `if` statement, repetition statements should not use the equality operator, ==, when testing single-precision or double-precision operands

Common Programming Errors (continued)

- Placing a semicolon at the end of the `for`'s parentheses (creates a do-nothing loop)
- Using commas to separate the items in a `for` statement instead of the required semicolons
- Omitting the final semicolon from the `do` statement

Common Compiler Errors

Error	Typical Unix-based Compiler Error Message	Typical Windows-based Compiler Error Message
Separating the statements in a `for` loop with commas rather than semicolons. For example, `for(init,cond,alt)`	(S) Syntax error: possible missing ';' or ','?	error: syntax error : missing ';' before ')'

Omitting the parenthesis in a `while` statement. For example,
`while condition`
`{`
`  statement;`
`}`
	(S) Syntax error: possible missing '('?	error: syntax error : missing ';' before '{'

Omitting the `;` at the end of the `do-while` statement. For example,
`do`
`{`
`  statement;`
`}while(condition)`
	(S) Syntax error.
	(This error tends to lead programmers astray. You would expect to get the error generated by a missing semicolon or comma, but instead you get a syntax error.)
		error: syntax error : missing ';'

Omitting the second `+` or `-` in a post increment or decrement statement. For example, `val+;` or `val-;`	(S) Syntax error. (Note that `+val;` and `-val;` do not generates a compiler error because these are valid expressions)	error: syntax error : ';'

Summary

- A section of repeating code is called a loop
- The three C repetition statements are `while`, `for` and `do-while`
- Loops are also classified as to the type of tested condition
- The most commonly used syntax for a `while` loop is

```
while (expression)
{
  statements;
}
```

Summary (continued)

- A `for` statement performs the same functions as the `while` statement, but uses a different form
- The `for` statement is extremely useful in creating counter-controlled loops
- The `do-while` statement is used to create posttest loops because it checks its expression at the end of the loop

Output table

- 【ex】Display multiplication table
- format 1:
- 1*1= 1
- 2*1= 2 2*2= 4
- 3*1= 3 3*2= 6 3*3= 9
- …… …… …… ……
- 9*1= 9 9*2=18 9*3=27 ……
  9*9=81