

Programming Fundamentals for Engineers  
0702113

# 4. Loops

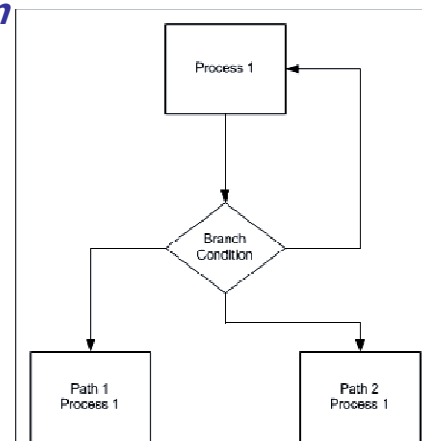
*do, while* and *for* Constructs

**MUNTASER ABULAFI**  
**YACOUB SABATIN**  
**DR. LABIB ARAFEH**

1

## Loops

- A *loop* is a statement whose job is to repeatedly execute some other statements (*loop body*).
- Loops have:
  - *Controlling Expression (condition)*
  - & *Loop Body*



2

## Loops

- C Supports: three types of loops:
  - *while*, *do while* and *for*;
  - All have:
    - *Controlling Expression* &
    - *Loop Body*
- *while loop*: Keeps repeating an action until an associated test returns **false**;
  - Useful where the **programmer does not know** in advance how many times the loop will be traversed.
- *do while loop*: Similar to *while*, but the test occurs after the loop body is executed.
  - This ensures: **loop body is run at least once**.
- *for loop*: Frequently used, where the loop will be traversed a **fixed number of times**.

3

C\_Prog. Spring 2012

## While loop

- *while loop* is used to evaluate a block of code (statements) as long as some condition is **true**. (while loop repeated until condition becomes false)
  - If the condition is **false**, from the start the block of code **is not executed at all**.
  - Syntax is as follows

```
while ( tested condition is satisfied ){
    statement
}
```

Controlling expression

- Pseudocode:

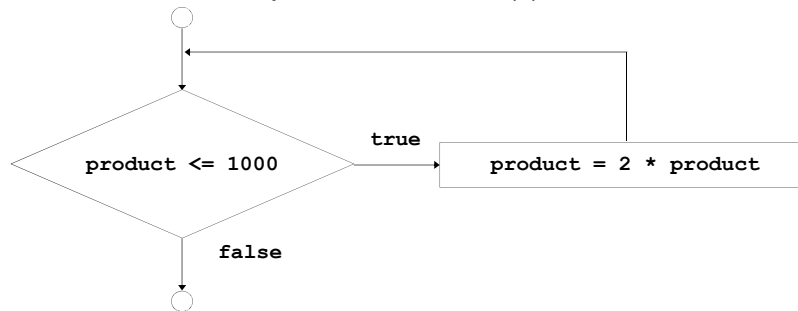
*While there are more items on my shopping list  
Purchase next item and cross it off my list*

4

C\_Prog. Spring 2012

- **Consider:**

```
int counter = 1;    // initialization
while ( counter <= 10 ){// Repetition condition
    printf( "%d\n", counter );
    ++counter; }    // increment
```



```
int product = 2;
while ( product <= 1000 )
    product = 2 * product;
```

5

C\_Prog. Spring 2012

- **Example 1:**

```
int number = 99;
while (number !=999)
{
    printf(\n Input Value=%d\n ", number);
    scanf( "%d" , &number); //number is updated
}
```

- **Example 2:**

```
x = 0;
while (x < 100);
x++; // Infinite loop!
```

6

C\_Prog. Spring 2012

- Condensed code: Make the program more concise!
  - Initialize counter to 0
    - while ( ++counter <= 10 )
      - printf( "%d\n", counter );

**Example 3:** This program counts from 1 to 100.

```
#include <stdio.h>
int main()
{
    int count = 1;
    while (count <= 100)
    {
        printf("%d\n", count);
        count += 1; //count = count + 1
    }return 0; }
```

7

C\_Prog. Spring 2012

**Example 4:** Nested while loops used to print out multiplication tables:

```
int i=1, j=1;
while(i <= 12) /*goes "down" page */
{
    j = 1;
    while(j <= 12) // goes "across" page
    {
        printf(" %3d", i*j);
        j++;
    }
    printf("\n"); // end of line
    i++;
}
```

8

C\_Prog. Spring 2012

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

9

C\_Prog. Spring 2012

- **Example5:** Develop a C code to sum a series of integers?
- **Reasoning/Think:** Queries to ask & answer!:
  - Number of Integers we need to sum up?
  - How can we keep entering the integers → Loop!
  - How many times: Unknown!
  - How can we stop the loop (To avoid infinite loop)!
- **Analysis:**
  - Output: Sum\_Of\_Integers;
  - Input: Number of Integers (Integer);
  - Process:
    - Sum\_Of\_Integers=0;
    - Read Integer;
    - while (Integer !=0){
      - Sum\_Of\_Integers = Sum\_Of\_Integers + Integer;
      - Read Integer;}

10

C\_Prog. Spring 2012

- **Algorithm:**

1. Start;
2. Sum\_Of\_Integers = 0;
3. Read Integer;
4. while (Integer !=0){  
    Sum\_Of\_Integers = Sum\_Of\_Integers + Integer;  
    Read Integer;}
5. Output Sum\_Of\_Integers;
6. End.

```
#include <stdio.h>
main()
{
    int Integer, Sum_Of_Integers;
```

11

C\_Prog. Spring 2012

```
Sum_Of_Integers = 0;
printf( "\n Please enter an integer
you need" );
printf( "\n Please enter 0 to
stop" );
scanf( "%d" , &Integer);
while (Integer !=0){
    Sum_Of_Integers += Integer;
    scanf( "%d" , &Integer);}
printf( "\n The sum of the entered
Integers=%5d" , Sum_Of_Integers);
return 0;
{
```

- **Can you modify to calculate the Average?**

12

C\_Prog. Spring 2012

- **Example 6:** A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are to be read. Determine the class average on the quiz?

– Pseudo code:

*Set total to zero*

*Set grade counter to one*

*While grade counter is less than or equal to ten*

*Input the next grade*

*Add the grade into the total*

*Add one to the grade counter*

*Set the class average to the total divided by ten*

*Print the class average.*

– **Convert the above Pseudo code to a C code!**

13

C\_Prog. Spring 2012

- **Example 7:** A college has a list of test results (1 = pass, 2 = fail) for 10 students. Develop a C program that analyzes the results. If more than 8 students pass, print "Raise Tuition". After understanding the **Pseudo code**, code in C!

- **Reasoning:**

– The program must process 10 test results

- Counter-controlled loop will be used

– Two counters can be used

- One for number of passes, one for number of fails

– Each test result is a number—either a 1 or a 2

- If the number is not a 1, we assume that it is a 2

14

C\_Prog. Spring 2012

- **Pseudo code:**

```

Initialize passes to zero
Initialize failures to zero
Initialize student to one
While student counter is less than or equal to ten
    Input the next exam result
    If the student passed
        Add one to passes
    else
        Add one to failures
    Add one to student counter
Print the number of passes
Print the number of failures
If more than eight students passed
    Print "Raise tuition"

```

15

C\_Prog. Spring 2012

## Do Loop

- **do loop** executes a block of code as long as a condition is satisfied (**true**).
- The difference between a **do** & **while loop** is:
  - **while loop** tests its condition at **top** of its loop;
    - i.e. if the test condition is **false** as the **while loop** is entered the block of code is **skipped**.
  - **do loop** tests its condition at **bottom** of its loop
    - i.e. Since the condition is tested at the bottom of a **do loop**, its block of code is **always** executed **at least once**;
    - Syntax:
 

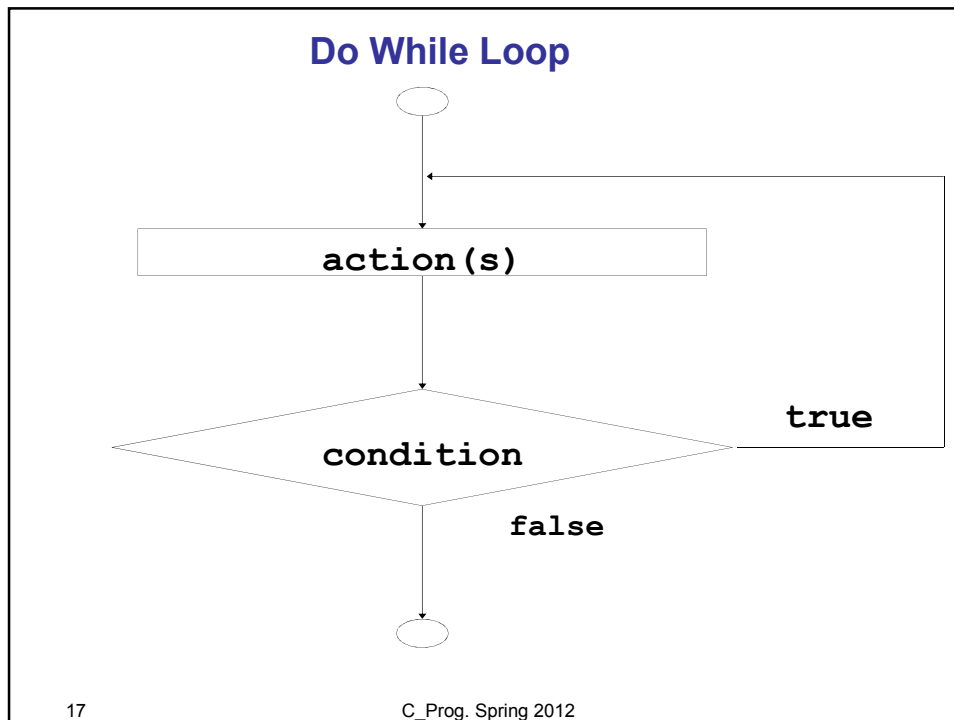
```

do {
    block of code (statement);
} while ( condition / Expression is satisfied);

```

16

C\_Prog. Spring 2012



- **Example 1:** What is the output of the following segment of code:

```

int x = 21;
do
printf("x = %d", x++);
while (x < 100);
  
```

→ A printout of a list of numbers 21 to 99!

- **Example 2:**

```

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

→ Prints the integers from 1 to 10

**Example 3:** A game that allows a user to guess a number between 1 and 100.

We will use: A **do loop** since we know that winning the game always requires at least one guess!

```
number = 44;

do {
    Read guess;

    if (guess > number) {printf("Too High\n"); }
    if (guess < number) {printf("Too Low\n"); }
    } while (guess != number)

Output number;
```

19

C\_Prog. Spring 2012

```
#include <stdio.h>
main()
{ int guess, number = 44;
printf("Guess a number between 1 & 100\n");
do {
    printf("Enter your guess: ");
    scanf("%d",&guess);
    if (guess > number) {
        printf("Too High\n"); }
    if (guess < number) {
        printf("Too Low\n"); }
    } while (guess != number);
    printf("You win. The answer %d", number);
return 0; }
```

20

C\_Prog. Spring 2012

**Example 4:** Develop a C code to compute the number of digits in a nonnegative integer?

**Reasoning:** To calculate a nonnegative number of digits:

Repeat → number /=10;  
 digits++  
 while number > 0

**Algorithm:**

1. Start
2. Digits = 0;
3. Read Number;
4. Do { number /=10;  
 digits++  
 } while (number > 0)
5. Output Digits;
6. End.

21

C\_Prog. Spring 2012

## For loop

- *for loop* can execute a block of code for a **fixed number of repetitions**.

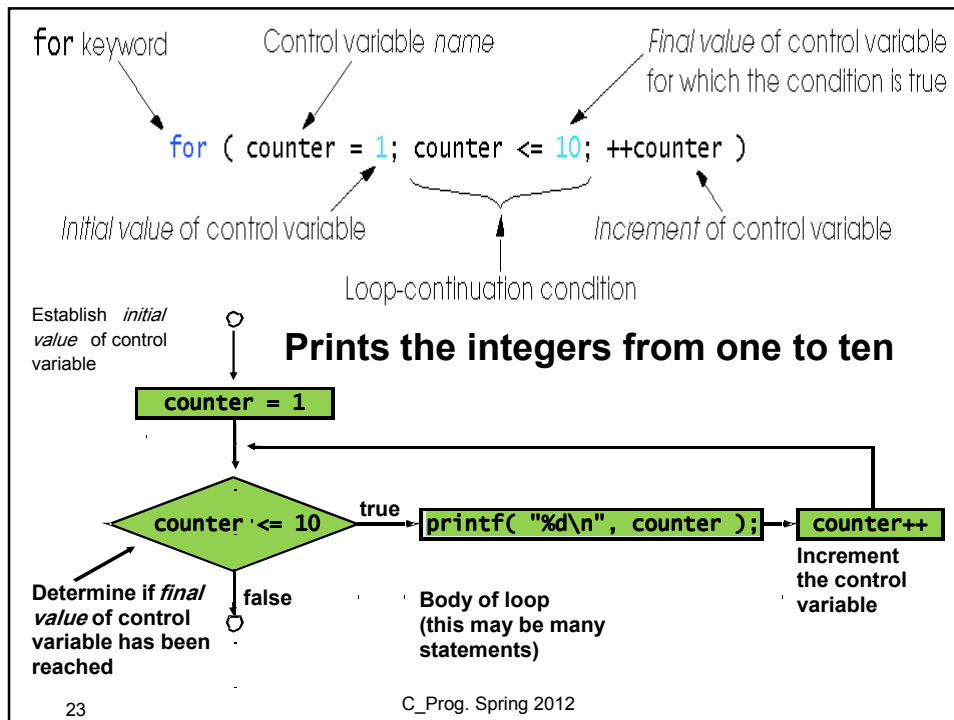
- Best choice for loops that:
  - **count up (increment)**; OR,
  - **count down (decrement)**

- *for loop* syntax is as follows:

```
for (initializations; test conditions; operation)
{
  block of code
}
for (expression1; expression2; expression3)
{
  Statement
}
```

22

C\_Prog. Spring 2012



```
initialization;
while ( loopContinuationTest / expression2 ) {
    statement;
    increment;
}
```

```
int counter = 1;
while ( counter <= 10 ) {
    printf( "%d\\n", counter );
    counter++;
} // prints 1 to 10!
```

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

**Examples 1:**

- *Counting up from 0 to n-1:*  
`for (count = 0; count < n; count++){  
printf("%d\n",count); }`
- *Counting up from 1 to n:*  
`for (count = 1; count <= n; count++){  
printf("%d\n",count); }`
- *Counting down from n-1 to 0:*  
`for (count = n-1; count >= 0; count--){  
printf("%d\n",count); }`
- *Counting down from n to 1:*  
`for (count = n; count > 0; count--){  
printf("%d\n",count); }`

25

C\_Prog. Spring 2012

**Examples 2:**

- A for loop that counts from 10 to 20:  
`for (count = 10; count <= 20; count++)  
{  
printf("%d\n", count);  
}`
- A loop that counts until a user response terminates the loop:  
`for (count = 1; response != 'N'; count++)  
{  
printf("%d\n",count);  
printf("Continue (Y/N): \n");  
scanf("%c",&response);  
}`

26

C\_Prog. Spring 2012

- More complicated test conditions are also allowed!
- Suppose the user of the last example never enters "N", but the loop should terminate when 100 is reached, regardless:

```
for (count = 1; (response != 'N') &&
                (count <=
100); count++)
{
    printf("%d\n",count);
    printf("Continue (Y/N): \n");
    scanf("%c",&response);
}
```

27

C\_Prog. Spring 2012

- It is also possible to have multiple initializations & multiple actions (comma operator)
- Example: This loop starts one counter at 0 and another at 100, & finds a midpoint between them.

```
for (i = 0, j = 100; j != i; i++, j--)
{
    printf("i = %d, j = %d\n", i, j);
}
printf("i = %d, j = %d\n", i, j);
```

28

C\_Prog. Spring 2012

- **Initializations are optional.**

- Suppose we need to count from a user specified number to 100. The first semicolon is still required as a place keeper:

```
printf("Enter a number to start the
count: ");
scanf("%d",&count);
for ( ; count < 100 ; count++)
{      printf("%d\n",count);      }
```

29

C\_Prog. Spring 2012

- The actions are also optional.  
Here is a silly example that will repeatedly echo a single number until a user terminates the loop;

```
for (number = 5; response != 'Y';)
{
    printf("%d\n",number);
    printf("Had Enough (Y/N) \n");
    scanf("%c",&response);
}
```

30

C\_Prog. Spring 2012

Summarizing last 2 slides:

- All three expressions (*initializations; test conditions; operations*) in *for loops* are optional BUT not the semicolons;
- If *initializations* is absent, then **NO initialization** takes place before loop is executed:  

```
i = 10;
for (; i > 0; --i)
    printf("T minus %d and counting\n", i)
```
- If *actions* is absent, then *either* the **Loop Body** or *test conditions* will have to **increment** or **decrement** the counter  

```
for (i = 10; i > 0;)
    printf("T minus %d and counting\n", i--)
```

31

C\_Prog. Spring 2012

- If *test conditions* is absent, we assume a default value of **1** (**true**), which causes an infinite loop  

```
for (;;) /* establishes an infinite loop */
```
- To **break from infinite loop** we use Ctrl Y/Break, etc.
- If *initializations* & *actions* are absent, the resulting loop is a *while loop*!

```
for (; i > 10;)
    printf("T minus %d and counting\n", i--)
```

Is same as

```
while ( i > 10)
    printf("T minus %d and counting\n", i--)
```

32

C\_Prog. Spring 2012

Implement in 3 separate codes a count down from *10 to 1* using **while**, **do**, and **for** loops:

```
• int i = 10;
  while (i > 0)
  { printf("%d\n",i);
    i = i - 1; }
```

```
• int i = 10;
  do
  { printf("%d\n",i);
    i = i - 1;
  } while (i > 0);
```

```
• for (int i = 10; i > 0; i--)
  {
    printf("%d\n",i);
  }
```

33

**Example:**

```
/* Summation of even numbers with the for loop */
int sum = 0; /* initialize sum */
int number; /* number to be added to sum */
for (number = 2; number <= 100; number += 2)
{
    sum += number; /* add number to sum */
} /* end for */
printf( "Sum = %d.\n", sum ); /* output sum */
```

34

## The Comma Operator

- Comma operator, written as a comma (,) **combines several expressions into a single statement**
- It evaluates its **leftmost** expression and **discards** the value, and **returns** the value of the **rightmost** expression:

→ `7 + 3, 6 * 2;`      */\* Returns 12 \*/*

→ `3, 14;`      */\* Gives 14 (the rightmost expn and **not** pi) \*/*

→ `a = 0, b = 3, a += b, b = 0;`  
*/\* sets a to 3, b to 0 and returns 0 \*/*

35

- It is commonly used in **for** loops

### Example:

Count from 1 to n using one variable, and simultaneously from n to 1 using another variable

```
for (i = 1, j = n; i <= n; i++, j--) {.....}
```

- The comma operator has the lowest precedence of any operator, binary, infix & left associative operator (LtR).
- The comma operator is seldom necessary, except in function-like preprocessor macros

36

**Examples:**

- Suppose **i=1, j=5**; when the expression **++i,i+j** is evaluated, **i** is first **incremented**, then **i+j** is **evaluated**, the value of expression is **7** (and **i** is **2** now).

- We can write:

```
sum=0;
for (i=1 ; i<=N ; i++)
    sum += i;
```

As:

```
for (sum=0, i=1 ; i<=N ; i++)
    sum += i;
```

37

## Exiting from a loop

There are number of ways to exit from a loop:

- The **break** Statement:
  - **break** statement can be used with **switch** or any of the **3 looping structures**;
  - It causes an **immediate exit** from the **switch, while, do-while, or for** statement in which it appears;
  - If the **break** is inside *nested structures*, control exits only the **innermost** structure containing it.
  - Program execution continues with the first statement **after** the structure.

38

- Common uses of the **break** statement
  - Escape *early* from a loop;
  - Skip the remainder of a **switch** statement;
  - Avoid unnecessary execution of statements.

39

**Example:** Check whether **Number** is a **prime**: Use **for** loop that **divides** **Number** by numbers (**nums**) between **2** and **Number-1**!

```
for (nums = 2; nums < Number; nums++)
  if (Number % nums == 0) break;

/* if we exited the loop somewhere during iterations, this
   means that nums is still < Number */
if (nums < Number)
  printf("%d is divisible by %d\n", Number, nums);

else /* nums=Number i.e. we reached the end of the loop */
  printf("%d is prime\n", Number);
```

40

- **Continue Statement:**

- Is valid **only** within loops;
- Terminates the **current loop iteration**, but **not** the **entire loop**;
- In a *for* or *while*, *continue* causes **the rest** of the body statement to be **skipped**.
- In a *for* statement, the update is done;
- In a *do-while*, the exit condition is tested, and if true, the next loop iteration is begun.

41

**Example:**

```
for (x = 10; x > 0; x = x - 1)
{
    if(x > 5) continue;
    printf("x = %d\n",x);
    // will print only numbers NOT > 5
} // Check!!
```

42

- The `goto` Statement:
  - Capable of jumping to any statement in a function;
  - Used as: `goto identifier;`
  - Identifier is a label at the beginning of a statement or number of statements.
  - NOT RECOMMENDED “spaghetti code”.
  - Example:

```
for (nums = 2; nums < Number; nums++)
    if (Number % nums == 0) goto done;
done:
if (nums < Number)
    printf("%d is divisible by %d\n", Number,
        nums);
else
    printf("%d is prime\n", Number);
```

43