

Chapter 04

Loops

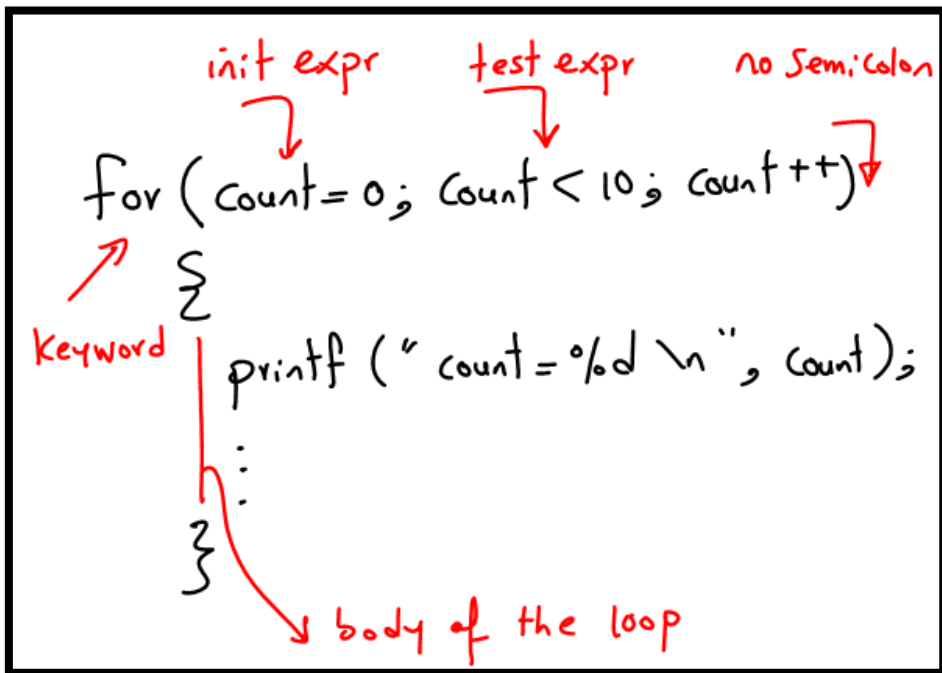
■ Loops

- If something is worth doing, it is worth doing more than once. Programming is the same; we frequently need to perform an action over and over, often with variations in the details each time — the mechanism that meets this need is the **loop**

- There are three major loop structures in C
 - **For** loop
 - **While** loop
 - **Do-while** loop

■ FOR - Loop

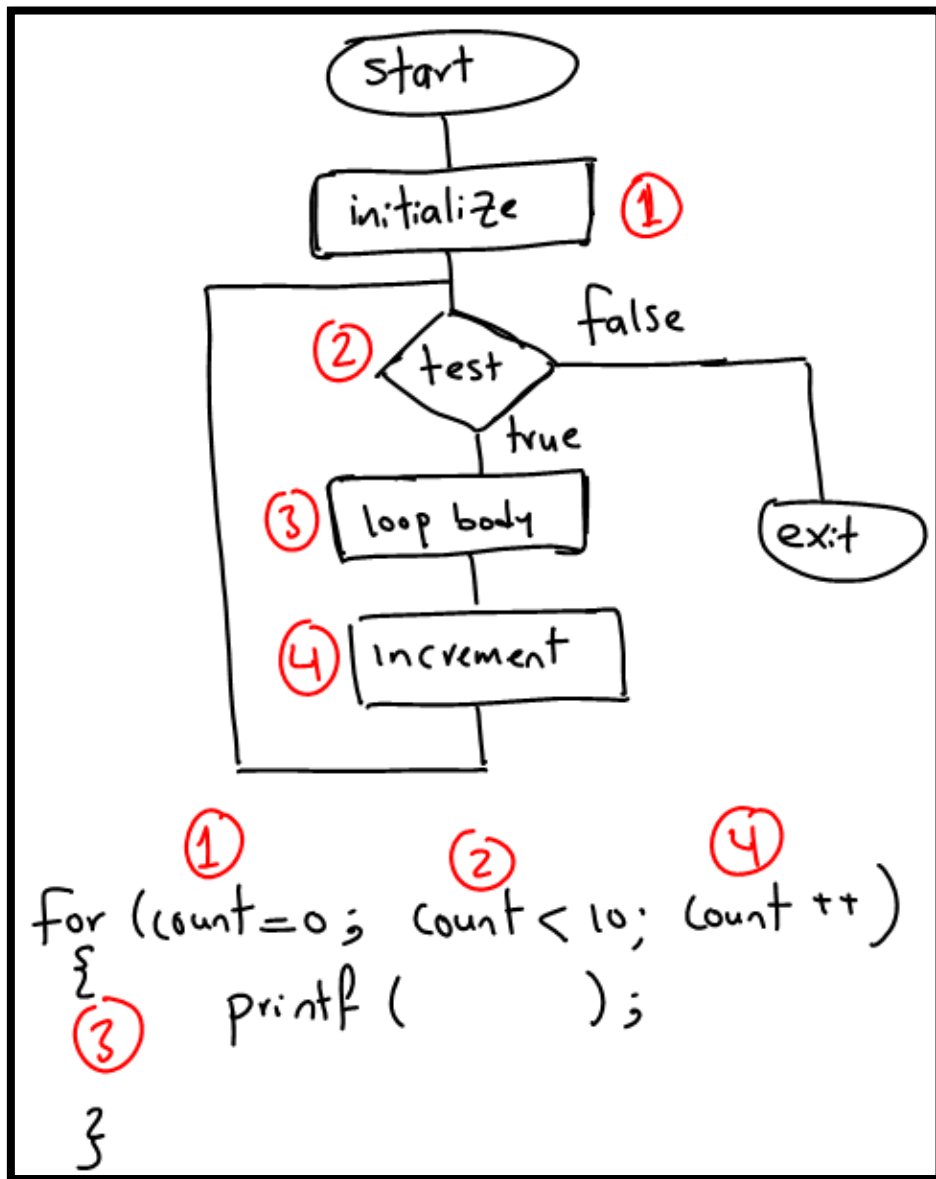
- You want to do something a fixed number of times — perhaps you want to calculate the pay checks for 120 employees or print out the squares of all the numbers from 1 to 50
- Here you should know before executing the program, how many times you need to repeat your task

**FOR-Loop**

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int count;
    for(count=0;count<10;count++)
    {
        printf("count = %d\n",count);
    }
}
```

```
count = 0  
count = 1  
count = 2  
count = 3  
count = 4  
count = 5  
count = 6  
count = 7  
count = 8  
count = 9
```

- The FOR loop may be started at any value, here `count = 0`, however, loops are often started at 1 or some other number
- `Count<10` — tests each time through the loop to see if `count` is less than 10. If the test expression is true (`count` is less than 10), the body of the loop will be executed. If the expression becomes false, the loop will be terminated and control will pass to the statements following the loop
- `Count++`, increments the variable `count` each time the loop is executed
- the loop variable in a FOR loop does not have to be increased by 1 each time through the loop
 - `count = count +3`
- in a FOR loop, **do not place a semicolon between the loop expression and the body of the loop**, since the keyword `for` and the loop expression do not constitute a complete C statement
- It is not important to consider the braces when the `for` loop is one statement



FOR-Loop // multiple statements

```
#include <stdio.h>  
#include <stdlib.h>  
main()
```

```
{  
    int count,total = 0;  
    for(count=0;count<10;count++)  
    {  
        total = total +count;  
        printf("total = %d\n",total);  
    }  
}
```

```
total = 0  
total = 1  
total = 3  
total = 6  
total = 10  
total = 15  
total = 21  
total = 28  
total = 36  
total = 45
```

FOR-Loop // multiple statements

```
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int count,total ;  
    for(count=0,total = 0;count<10;count++)  
    {  
        total += count;  
        printf("total = %d\n",total);  
    }  
}
```

▪ Loop style Note

Many C programmers, including the venerable Kernighan and Ritchie, handles braces in a somewhat different way than we've shown. They put the opening brace on the same line as the loop expression

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    int count, total ;
    for(count=0, total = 0; count<10; count++) {
        total += count;
        printf("total = %d\n", total);
    }
}
```

■ ASCII Table Program

ASCII table

<pre>#include <stdio.h> #include <stdlib.h> main() { int n; for (n = 0; n<256; n++) { printf("%3d=%c\t", n, n); } }</pre>

- The program uses the tab character ('\t') in the printf() statement. This causes the next item printed to start eight spaces from the start of the last item. In other words, it divides the screen into columns eight characters wide.
- On a 80-column screen then, we have room for 10 items
- Format specifiers can interpret the same variable in different ways
- What's actually stored in the computer's memory is an integer, n as specified in the type declaration statement. The %d format specifier prints the decimal representation of n, while the %c format specifier prints the character whose ASCII code is n

■ Drawing a line with a Graphics character

line

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int n;
    for (n = 1;n<40;n++)
        printf("%c", '\xDB');
}
```

- No braces here for the loop body — it is only one single statement

Example:

Write a C program to calculate the area of circle number of times equal to n given by the user.

Area of N circles

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,n;
    float radius, area;
    printf("Type No. of circles: ",n);
    scanf("%d",&n);
    for (i = 1;i<=n;i++)
    {
        printf("Circle %d radius = ",i) ;
        scanf("%f",&radius);
        area = 3.14*radius*radius;
        printf("Area of circle %d =
%f\n",i,area);
    }
}
```

```
Type No. of circles: 2
Circle 1 radius = 10
Area of circle 1 = 314.000000
Circle 2 radius = 2
Area of circle 2 = 12.560000
```

Example:

Write a C program to calculate the sum of starting from 1 to 10.

Series of numbers

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,sum ;
    for(i = 1;i<=10;i++)
    {
        sum = 0;
        sum = sum + i;
        printf("sum = %d",sum);
    }
}
```

- Run and execute the above code
- what is wrong with this code

Example:

Write a C program to calculate the sum of starting from 1 to 10.

Series of numbers

<pre>#include <stdio.h> #include <stdlib.h> main() { int i,sum = 0 ; for(i = 1;i<=10;i++) { sum = sum + i; printf("sum = %d",sum); } }</pre>

- | |
|---|
| <ul style="list-style-type: none">▪ Run and execute the above code▪ what is wrong with this code |
|---|

Example:

Write a C program to calculate the sum of starting from 1 to 10.

Series of numbers

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,sum = 0 ;
    for(i = 1;i<=10;i++)
    {
        sum = sum + i;

    }
    printf("sum = %d",sum);
}
```

- Run and execute the above code
- what is wrong with this code

Example:

Write a C program to calculate the sum of starting from 1 to 10.

Series of numbers

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,sum = 0 ;
    for(i = 1;i<=10;i++)
        sum = sum + i;
    printf("sum = %d",sum);
}
```

- Run and execute the above code
- what is wrong with this code

Example:

Write a C program to calculate the sum of 5 numbers entered by user

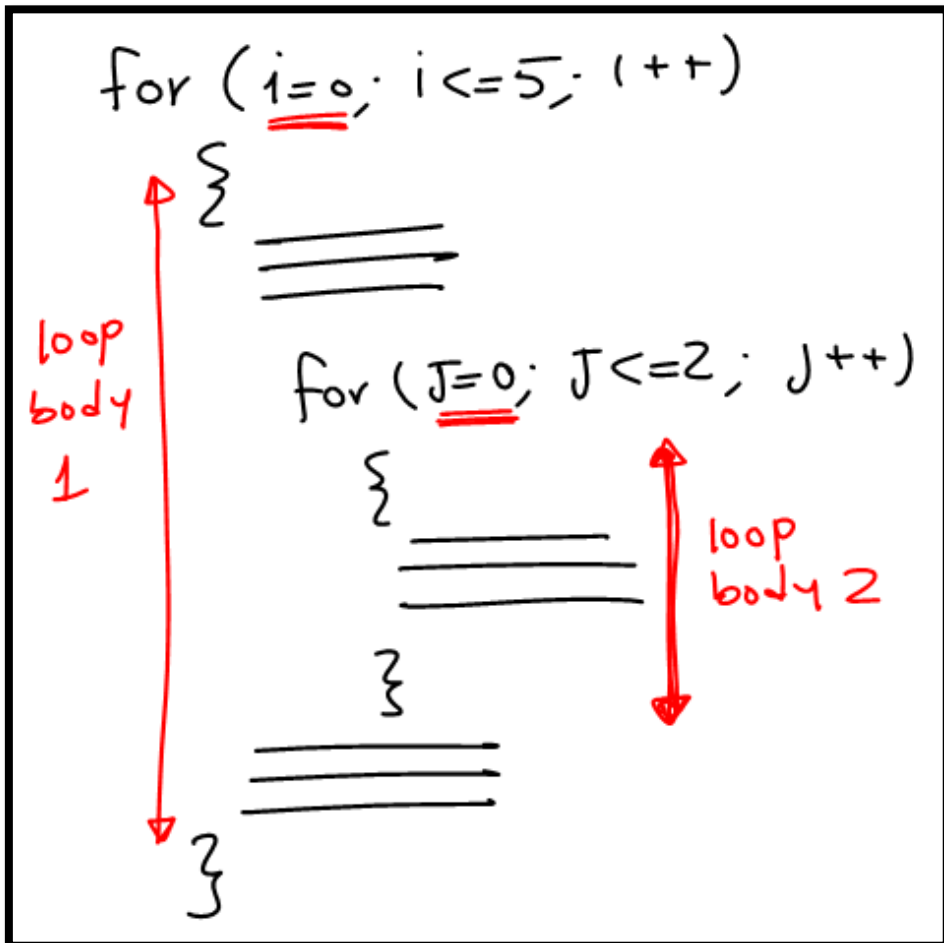
Series of numbers

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,sum = 0 ;
    float num;
    for(i = 1;i<=5;i++)
    {
        scanf("%f",&num);
        sum = sum + num;
    }
    printf("sum = %d",sum);
}
```

```
5
3
6
7
8
sum = 29
```

■ Nested For Loops

It is possible to nest one FOR loop inside another



- First i is initialized to 0 then execute the large loop. Coming to the loop expression 2, j is initialized to 0. Inside loop body 2, $i = 0, j = 0$.

- When coming to the second brace that delimit the second FOR, j is incremented to 1, and now $i = 0, j = 1$. Then it continues till reached $j = 2$.
- After that, it executes the statements between the two loops, till reaches the last brace, here i is incremented to reach 1, executing the statements between the two loops.
- When it come to the second loop, j will takes the value of 0.i.e. $i = 1, j = 0$, and continues as before
- To conclude, i and j will take these values
 - $(i, j) = 0, 0$
 - $(i, j) = 0, 1$
 - $(i, j) = 0, 2$
 - $(i, j) = 1, 0$
 - $(i, j) = 1, 1$
 - $(i, j) = 1, 2$
 - ...
 - $(i, j) = 5, 2$

Example:

Write a C program to print out the multiplication table

Multiplication table

```
#include <stdio.h>
#include <stdlib.h>
```

```
main()
{
    int i,j;
    for (i=1;i<=12;i++)
    {
        for (j=1;j<=12;j++)
        {
            printf("%dX%d=%d",i,j,i*j);
        }
    }
}
```

- execute and run the code written above
- what is wrong with the code above

Write a C program to print out the multiplication table

Multiplication table

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,j;
    for (i=1;i<=12;i++)
    {
        for (j=1;j<=12;j++)
        {
            printf("%dX%d=%d\t",i,j,i*j);
        }
        printf("\n");
    }
}
```

- execute and run the code written above
- what is wrong with the code above

Write a C program to print out the multiplication table

Multiplication table

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,j;
    for (i=1;i<=12;i++)
    {
        for (j=i;j<=12;j++)
        {
            printf("%dX%d=%d\t",i,j,i*j);
        }
        printf("\n");
    }
}
```

- execute and run the code written above
- what is wrong with the code above

- Drawing a Rectangle (square) in terms of line using Nested FOR

Rectangle

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i,n;
    for(i=0;i<=4;i++)
    {
        for (n = 1;n<40;n++)
            printf("%c",'\xDB');
        printf("\n");
    }
}
```

- Execute and run the code above

■ The WHILE –LOOP

- It actually uses the same elements, but they are distributed throughout the program

Handwritten code snippet illustrating a while loop structure with annotations:

```
Count = 0;  
While (Count < 10)  
{  
    _____  
    _____  
    _____  
    :  
    Count ++  
}
```

Annotations:

- Red arrow pointing to `Count`: loop variable
- Red arrow pointing to `(Count < 10)`: condition
- Red text next to the loop body: loop body

While

```
#include <stdio.h>  
#include <stdlib.h>  
main()  
{
```

<pre> int count = 0; while (count <10) { printf("\nCount = %d",count); count ++; } </pre>	
<pre> Count = 0 Count = 1 Count = 2 Count = 3 Count = 4 Count = 5 Count = 6 Count = 7 Count = 8 Count = 9 </pre>	<pre> #include <stdio.h> #include <stdlib.h> main() { int count = 0; while (count <10) printf("\nCount = %d",count++); } </pre>

 Note:

- While loops are more appropriate than FOR loops when the condition that terminates the loop occurs unexpectedly

Character Count

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char ch = 'X';
    int count = 0;
    while(ch != '\r')
    {
        count++;
        ch = getche();
    }
    printf("\nNo. of characters typed is: %d",--count);
}
```

```
Believe me, You are FASHEL KABEER:-)
No. of characters typed is: 36
```

- This program invites you to type in a phrase. As you enter each character it keeps a count of how many characters you've typed, and when you hit [Return], it prints out the total
- `ch = getche();` returns the value of a character the instant the character is typed
- `'\r'` the carriage return character
- The while loop will continue to be executed as long as it does not encounter a `'\r'`. when it does encounter it, the

loop will terminate and the total number of characters typed will be printed out

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    /*char ch = 'X';*/
    int count = 0;
    while(getche() != '\r')
        count++;
    printf("\nNo. of characters typed is:
%d",count);
}
```

```
Believe me, You are FASHEL KABEER:-)
No. of characters typed is: 36
```

■ Example

Write a C program to ask the user to type a character and then prints out the ASCII code for the character.

ASCII for each character

```
#include <stdio.h>
#include <stdlib.h>
```

```
main()
{
    char ch = 'X';
    while(ch != '\r')
    {
        printf("\n Type in a character: ");
        ch = getche();
        printf("\nThe code for %c is %d",ch,ch) ;
    }
}
```

```
Type in a character: c
The code for c is 99
Type in a character: A
The code for A is 65
Type in a character: Z
The code for Z is 90
Type in a character: G
The code for G is 71
Type in a character:
```

■ Nested-While loop

Write a C program to let you guess a lowercase letter from 'a' to 'e' and tells you if you're right or wrong

While loop nested within for loop

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char ch;
    int i;
    for(i=0;i<5;i++)
    {
        printf("\nType a letter from 'a' to
'e':\n");
        while ((ch=getche()) != 'd')
        {
            printf("\nSorry, %c is
incorrect.",ch);
            printf("\nTry again. \n");
        }
        printf("\nThat's it!");
    }
}
```

```
Type in a character: c
The code for c is 99
Type in a character: A
The code for A is 65
Type in a character: Z
The code for Z is 90
Type in a character: G
The code for G is 71
Type in a character:
```

- Assignement Versus relational operator
- **while** ((**ch=getche()**) != 'd')
- If the parenthese weren't here, the compiler would interpret the expression like this:
 - **while** (**ch=getche()** != 'd')
 - relational operator precednce is greater than that of the assignement operator

■ Example

Write a C program to calculate some numbers factorial ended by zero

ASCII for each character

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    long number = 1;
    long answer;
    while (number !=0)
    {
        printf("\nEnter a number: ");
        scanf("%ld",&number);
        answer = 1;
        while(number>1)
            answer = answer *number--;
        printf("Factorial is: %ld\n",answer);
    }
}
```

```
Enter a number: 3
Factorial is: 6

Enter a number: 4
Factorial is: 24

Enter a number: 6
Factorial is: 720

Enter a number: 5
Factorial is: 120

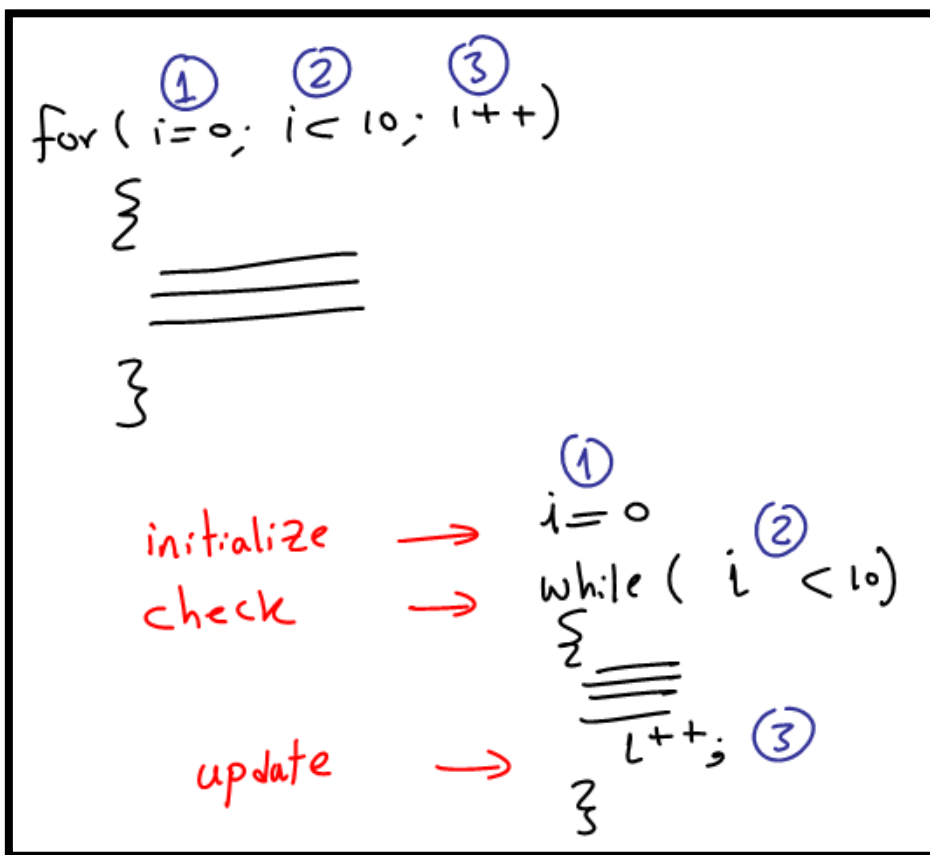
Enter a number: 2
Factorial is: 2

Enter a number: 1
Factorial is: 1

Enter a number: 0
Factorial is: 1
```

✂ FOR-LOOP or While LOOP

- if at the time you enter the loop you already know how many times you want to execute it, you are probably better off with the FOR loop
- if, on the other hand, the conditions for terminating the loop are imposed by the outside world, such as the user typing a certain character, then you are better off with the WHILE loop



■ DO-WHILE loop

- The loop is very similar to the while loop — the difference is that in the do loop the test condition is evaluated after the loop is executed, rather than before
- The **do** keyword marks the beginning of the loop — it has no other function
- The **while** keyword marks the end of the loop and contains the loop expression

A handwritten diagram illustrating the structure of a do-while loop. It starts with the word 'do' at the top left. Below it is an opening curly brace '{'. To the right of the brace are three horizontal lines representing the loop body. A red arrow points from these lines to the text 'loop body'. Below the lines is the code 'i++;' with a red arrow pointing to it from the text 'update'. This is followed by a closing curly brace '}'. Below the brace is the code 'while (i < 10) ;'. A red arrow points from the text 'condition' to the expression 'i < 10' in the while statement.

- An important detail to note is that this loop, unlike the **for** and **while** loops, is terminated with a semicolon — that is

the test condition in the parentheses following **while** ends with a semicolon

- When would you use a **do-while**? Any time you want to be sure the loop body is executed at least once.
- This situation is less common than that where the loop might not be executed at all, so the **do** loop is used less often than the **while** loop
- The **do** loop is useful when the body of a loop will always be executed at least once

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int total = 0;
    int i = 0;
    do
    {
        total +=i;
        i++;
    }
    while(i<=10);
    printf("\nTotal = %d",total);
}
```

■ The **break** and **continue**

- The **break** statement gets you out of loop as soon as it's executed. It is often used when an unexpected condition occurs — one that the loop test condition is not looking for
- The **continue** statement is inserted in the body of the loop, and when executed, takes you back to the beginning of the loop — bypassing any statements not yet executed

Exercise
Write a program that prints the squares of all the numbers from 1 to 20

■ Sheet

1. An interesting problem in number theory is sometimes called the "necklace problem." This problem begins with two single-digit numbers. The next number is obtained by adding the first two numbers together and saving only the ones-digit. This process is repeated until the "necklace" closes by returning to the original two numbers. For example, if the starting numbers are 1 and 8, twelve steps are required to close the "necklace":

18976392134718

Write a program that asks the user for two starting numbers, and then displays the sequence and the number of steps taken.

The program should look similar to:

Enter first number: 1

Enter second number: 8

1 8 9 7 6 3 9 2 1 3 4 7 1 8

Your numbers required 12 steps

2. Write a program that displays the sum of all of the integers between two input numbers, inclusive. The program output should look similar to:

Enter starting number: 25

Enter ending number: 47

The sum of integers from 25 to 47 is 828

3. Write a program that reads in letter grades for a class of 20 students and displays the number of students who passed (D or better) and the number who failed (E or F). the program output should look similar to:

```
Enter a grade: C
```

```
Enter a grade: A
```

```
...
```

```
16 students passed
```

```
4 students failed
```

4. Write a program that asks the user for starting and ending values and then displays a formatted table of squares and cubes. Write the program so that it is easy to change the column widths. The program output should look similar to:

X	X SQUARED	X CUBED
---	-----------	---------

5	25	125
---	----	-----

6	36	216
---	----	-----

12	144	1728
----	-----	------

5. write a C program that calculates the average and total for a known number of test grades. The program should first read in the number of grades and then the grades. The program output should look similar to:

```
How many grades? 12
```

```
Enter a grade: 95
```

...

Enter a grade: 69

The total is 1020

The average is 85.00

6. the factorial of a positive number is the product of all positive integers less than or equal to the numbers. For example, 5 factorials (written 5!) is equal to $5*4*3*2*1$ or 120. Write a C program that reads a positive integer and displays its factorial. The program output should look similar to:

Enter an integer:5

5! = 120

