BIRZEIT UNIVERSITY

Repetition and Loop Statements

PROBLEM SOLVING AND PROGRAM DESIGN In C
7th EDITION
Jeri R. Hanly, Elliot B. Koffman

By: Mamoun Nawahdah (PhD)
2013/2014

What is a Loop?

- ❖ A loop is a repetition control structure.
- ❖ It causes a single statement or block to be executed repeatedly.

Loop Constructs in C

- **while**
  - Execute block of statements repeatedly as long as some condition is true.
  - Condition is tested before block of statements are executed.
- **do – while**
  - Similar to while, however, condition is tested after block of statements are executed.
- **for**
  - An abbreviation for a collection of statements that use a while loop for creating counting loops.

جامعة بيرزيت
BIRZEIT UNIVERSITY

while

Loop Statement

The **while** Statement

```
while (it's raining) {
    <keep the umbrella up>
}
```

While Loop : Syntax

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

- “**condition**” is a logical expression that evaluates to **true** or **false**. It could be a relational or Boolean expression.
- “**statement**” could be a single statement or more than one statement bounded by **{ }**. It is often referred to as the **body of the loop**.

While Loop : Semantics

1. The condition is evaluated.
2. If it is **true**, then the body of the loop is executed. Then the control is transferred back to the condition for re-evaluation.
3. If the logical expression is **false**, the while loop is exited and control is transferred to the statement after the while statement.

More about while

- Initialize the conditions which are evaluated in the "**while**".
- Conditions are evaluated at the "top" of while statement or before the execution of the body.
- "**while**" body is executed **0** or more times.
- Updating of the conditions are done inside the body of the "**while**".
- Use "**while**" when the number of times a loop is executed is dependent on some condition set during execution of the body.
- Example: Input a list of positive number. The list is terminated by a negative number.

Example

❖ Display the values **1** to **100**:

```
int count=1;
while (count <= 100) {
    printf(“%d\n”, count);
    count + = 1;
}
```

Example

```
#include <stdio.h>
int main( ) {
    int total = 0, num=0;
    /* Read until the user enters -1 */
    while (num != -1)  {
            total+=num;
            printf("Enter a number (-1 to stop )");
            scanf("%d",&num);
    }
    printf( “Total = %d”, total);
    return 0;
}
```

Problem

- Design a **C** program that takes **n** integer numbers. When the user enters zero, the program will count the number positive and negative numbers entered by the user.

Give The Exact Output

```
int grade, counter = 1;
int total = 0;
while ( counter <= 10 ) {
        printf( "Enter grade: " );
        scanf( "%d", &grade );
        total += grade;
        counter++;
}
double average = total / 10;
printf( "Class average is %f\n", average );
```

Give The Exact Output

```
int x = 1, total =0, y;
while (x <= 6) {
      y = x * x;
      printf( "%d\n", y);
      total += y;
      x++;
}
printf("Total is %d\n", total);
```

What is the Exact Output?

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

Give the Exact Output

```
int  x = 0;
while(x < 10) {
   printf("%d %2d %3d\n", x, x*x, x*x*x);
   x++;
}
```

BIRZEIT UNIVERSITY

do-while

Loop Statement

When to Use do-while Statement

- Use this control structure:
  - When a loop needs to be executed at **least once**.
  - When the testing of the conditions needs to be done at the bottom.

The do-while Statement Syntax

```
do {
    actions;
} while (condition);
```

- How it works:
  - Execute action
  - If condition is **true** then execute action again
  - Repeat this process until condition evaluates to **false**.
  - Action is either a single statement or a group of statements within braces.

Example

```
int num=45,guess;
printf("Guessing a number \n");
do{
        printf("Enter your guess:");
        scanf("%d",&guess);
        if (guess > num)
                        printf("Too high\n");
        else if (guess < num)
                        printf("Too low\n");
} while (guess!=num);
printf("You win. The answer is %d", num);
```

Comparing **while** and **do-while**

- The difference is very subtle.
  - In the **while** loop the condition is checked **BEFORE** each iteration.
  - In the **do-while** loop the condition is checked **AFTER** each iteration.
- BUT checking **AFTER** each iteration is the same as checking **BEFORE** the next iteration
  - Except for the very first iteration !!!

What is the Output?

```
int x=3;
do {
    printf("x=%d \n", x);
    x--;
} while (x>0);
```

What is the Output?

```
int grade, sum=0, count=1;
do {
        printf("Enter grade:");
        scanf("%d", &grade);
        sum+=grade;
        count++;
} while(count<=5);
printf("The average is %.2f", sum/5);
```

Try each of these Inputs : 345, 82, and 6

```
int num;
printf("Enter a positive integer: ");
scanf("%d", &num);
do{
      printf("%d ", num%10);
      num /= 10;
}  while(num>0);
```

BIRZEIT UNIVERSITY

for

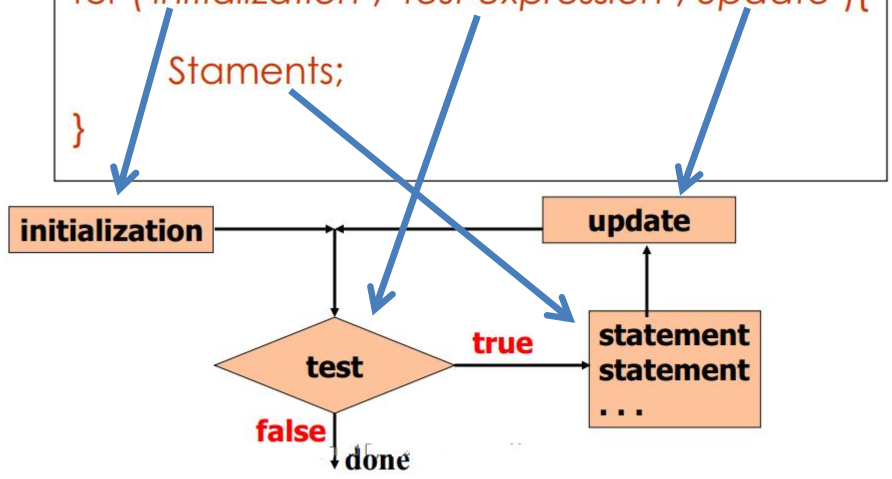
Loop Statement

Objectives

- ❖ Give the syntax of **for** statement.
- ❖ Discuss the semantics of **for** statement.
- ❖ Give an illustrative example.
- ❖ Trace the output of a code fragment.
- ❖ Compare **for** to **while** statement.
- ❖ Give more examples of **for** loop.

Syntax and Flowchart

```
for ( initialization ;  test expression ; update ){
        Staments;
}
```



for Loop - Semantics

1. **Initialize** expression is executed.

2. **Test** expression is evaluated.

❖ **If** it is **TRUE** , body of **for** is executed.

3. After the execution of the body, **update** expression is executed; go to **Step 2** above.

❖ **else** exit **for** loop.

Note

❖ "**for**" is **lower case.**

❖ The 3 parameters are surrounded by (**).**

❖ The 3 parameters are separated by **semicolons.**

❖ All 3 parameters must be identified (**but could be empty**).

❖ Can have **multiple initialization** and **update** statements (separated by **commas**).

❖ Update expression may involve **++** or **--** operators.

❖ No **semicolon** follows the closing parenthesis.

Simple for Loop Example

```
int counter;

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

     printf (“Counter value is: %2d\n", counter);

}
```

Example of for Repetition

When the loop control condition is evaluated and has value **false**, control passes to the statement following the **for** statement.

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

```
Output:
     1 Computer(s)
     2 Computer(s)
     3 Computer(s)
```

Example

```
int count;
for  (count = 0 ; count < 4 ; count++ ){
        printf(“%d \n”, count);
}
printf(  “Done\n”);
```

```
OUTPUT:
            0
            1
            2
            3
            Done
```

Equivalence with while Loop

❖ The following **while** loop is equivalent to the previous **for** loop:

```
int  count = 0;             // initialization
while  (count < 4) {        // test expression
     printf(“%d \n”, count);
     count++;               // update loop variable
}
printf(  “Done\n”);
```

Summing Values in Loops

```
int  num, count, total = 0;
for ( count = 0 ;  count < 5 ;  count++ ){
        printf("Enter a number: ");
        scanf ("%d", &num);
        total += num;
}
```

A Loop that Counts until a User Response Terminates the Loop

```
int count, response=1;
for (count = 1; response != -1; count++) {
        printf("%d\n", count);
        printf("Continue (-1 to stop): \n");
        scanf("%d", &response);
}
```

❖ Suppose the user of the last example never enters **"-1"**, but the loop should terminate when **100** is reached, regardless:

```
for ( count = 1 ;
    (response != -1) && (count <= 100) ;
    count++ ) {
    printf("%d\n", count);
    printf("Continue (-1 to stop): \n");
    scanf("%d", &response);
}
```

Multiple Initializations and Actions

❖ This loop starts one counter at **0** and another at **100**, and finds a midpoint between them:

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

Initializations are Optional

- ❖ For instance, suppose we need to count from a user specified number to **100**.
- ❖ The 1st 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);
}
```

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");
         response = getchar();
}
```

Example of for Loop

```
#include <stdio.h>
int main ( ) {
   int k, n ;
   for (k = 1, n = 12 ; k < 9 && n > 6 ; k++, n--)
        printf ("k=%d,  n=%d \n" , k , n ) ;
}
```

Another Example of for Loop

```
#include <stdio.h>
int main ( ) {
   int k = 1, n = 12 ;
   for (    ; k < 9 && n > 6 ;    ){
        printf ("k=%d,  n=%d \n" , k++ , n-- ) ;
}
```

for Loop Exercises 1

```
for ( i = 0;  i < 10;  i +=2 ){
  printf("%d \n",  i );
}
```

```
0
2
4
6
8
```

for Loop Exercises 2

```
for ( j = 20; j >= 0; j -= 3){
  printf("%d \n",  j );
}
```

```
20
17
14
11
8
5
2
```

for Loop Exercises 3

```
int x, total=0, y;
for(x=1;  x<=7;  x++)  {
        y = x * x;
        printf("%d \n", y);
        total += y;
}
printf("Total is %d", total);
```

for Loop Exercises 4

```
for (count=1; count<=99; count+=2)  {
        printf("%d \n", count);
        sum += count;
}
printf("The sum of odd numbers = %d ", sum);
```

Nested Loops

❖ Loops may be nested like other control structures.

❖ Nested loops consists an **outer** loop with one or more **inner** loops.

❖ Each time the **outer** loop is repeated, the **inner** loops are re-entered, their loop control expressions (initializations, conditions, and updates) are evaluated, and all required iterations are performed.

Example

```
int num, i, j;
printf("Enter  the  number  of  stars:  ");
scanf("%d", &num);
for ( i = 1;   i < num;   i++) {
      for ( j=1;   j<=i ;   j++){
               printf("* ");
      }
      printf("\n");
}
```

```
Enter number of stars: 5
*
* *
* * *
* * * *
```

What is printed to the screen?

```
for (j = 0; j <= 3; j ++) {
   for (k = j; k < 5; k ++) {
        printf("%d  %d \n", j , k);
   }
   printf ("\n");
}
```

```
0 0
0 1
0 2
0 3
0 4

1 1
1 2
1 3
1 4

2 2
2 3
2 4

3 3
3 4
```

Example 1

```
int i,k,j;
for (i=0;i<2;i++){
       printf("Outer %4d\n",i);
       for (j=0; j<3; j++)
              printf("  Inner%3d%3d\n",i , j);
       for (k=2; k>0; k--)
              printf("  Inner%3d%3d\n",i , k);
}
```

Example 2

```
int a,b;
for (a=1; a<7; a++){
   for (b=0; b<a; b++){
        printf("%4d", a*b);
   }
   printf("\n");
}
```

```
0
0 2
0 3 6
0 4 8 12
0 5 10 15 20
0 6 12 18 24 30
```

Example 2: Using while

```
int a,b;
a=1;
while (a<7){
   b=0;
   while (b<a) {
        printf("%4d", a*b);
        b++;
   }
   printf("\n");
   a++;
}
```

```
0
0 2
0 3 6
0 4 8 12
0 5 10 15 20
0 6 12 18 24 30
```

Exercise

- Write a **C** program fragment that will display the following:

```
w w w w w w
w w w w w
w w w w
w w w
w w
w
```

Comparing for, while and do-while

- while loop:

```
int i = 1;
while (i <= 20) {
        printf("%d  ", i);
        i++;
}
i = 20;
while (i > 0) {
        printf("%d  ", i);
        i--;
}
```

❖ **for** loop:

```
int i;
for (i = 1; i <= 20; i++) {
      printf("%d  ", i);
}
for (i = 20; i > 0; i--) {
      printf("%d  ", i);
}
```

❖ **do-while** loop:

```
int i = 1;
do {
      printf("%d  ",i);
      i++;
} while (i <= 20)
i = 20;
do {
      printf("%d  ",i);
      i--;
} while (i > 0);
```

Jump Statements

- Jump statements transfer control unconditionally.
- There are 4 types of jump statements in **C**:
  - ~~**goto**~~
  - **continue**
  - **break**
  - **return**

break Statement

- The **break** statement is used to get out of a **for** loop, **while** loop, **do** loop, or **switch** statement.
- Control passes to the statement following the terminated statement.
- You can exit out of a loop at any time using the break statement. This is useful when you want a loop to stop running because a condition has been met other than the loop end condition.

Example

❖ The example below will exit the **while-loop** when the variable **i** becomes 5:

```
int i = 0;
while (i < 10){
        printf("%d\n", i++ );
        if (i == 5)
                break;
}
```

Give the Exact Output?

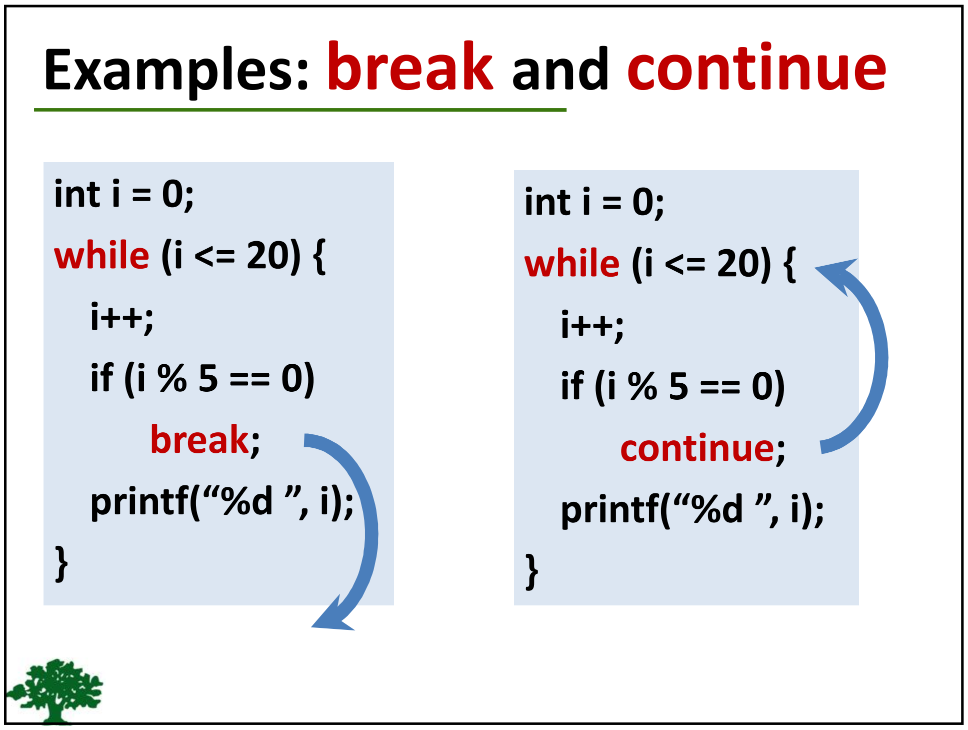
```
int i = 10;
while ( i > 0 ){
     printf("Hello %d\n", i );
     i--;
     if( i == 6 )
          break;
}
```

continue Statement

- A **continue** statement may appear only within an iteration statement.
- You can use **continue** to skip the rest of the current loop and start from the top again while updating the loop variable again.
- The following example will never print "***Hello***" because of the **continue**:

Example

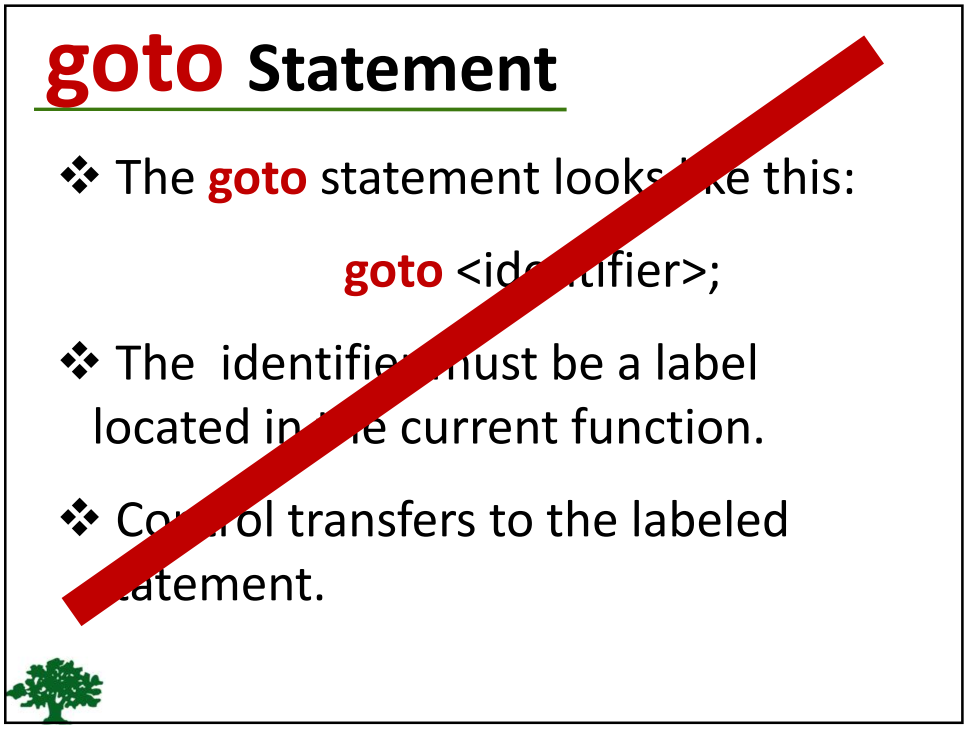
```
int i = 0;
while (i < 10){
    i++;
    continue;
    printf("Hello\n");
}
```



Examples: break and continue

```
int i = 0;
while (i <= 20) {
   i++;
   if (i % 5 == 0)
        break;
   printf("%d ", i);
}
```

```
int i = 0;
while (i <= 20) {
   i++;
   if (i % 5 == 0)
        continue;
   printf("%d ", i);
}
```



~~goto Statement~~

- ~~The goto statement looks like this:~~

  ~~goto <identifier>;~~

- ~~The identifier must be a label located in the current function.~~
- ~~Control transfers to the labeled statement.~~

Example

❖ The following program increments the ***ind*** [illegible] variable until 1000 and then transfers the contr[illegible]o the statement after the **for** loop labele[illegible]inish_up** using the **goto** statement:

```
int main(void){
   int index;
   for(in[illegible]0;;){
         index++;

         if(index == 1000)    goto    finish_up;
   }
   finish_up:
   printf("Goodbye.\n");
}
```