LOOPING CONSTRUCTS FOR EMBEDDED C

By
Dr. H. Parveen Sultana
Professor/SCOPE,
VIT,Vellore

19/12/2023

TYPES OF LOOPING CONSTRUCTS

1. For Loop
2. While
3. Do-While

FOR LOOP SYNTAX

for (*statement 1*; *statement 2*; *statement 3*) {
// code block to be executed }

Statement 1 - executed one time before the execution of the code block.

Statement 2 - the condition for executing the code block.

Statement 3 - executed every time after the code block has been executed.

How for loop works?

- The initialization statement is executed only once.
- Then, the condition is evaluated. If the test expression is false, the for loop is terminated.
- If the condition is evaluated to true, statements inside the body of the for loop are executed, And then the expression is updated.
- Again the condition is evaluated.
- This process goes on until the condition is false. When the condition is false, the loop terminates.

EXAMPLE

```
int i;

for (i = 0; i < 5; i++) {
  printf("%3d", i);

}
```

Output : 0 1 2 3 4

FOR LOOP – TYPE1

Note: return 0 is implicit at the end of the programs, please ignore that.

```
int main()
{
  int lower_limit = 0;
  int upper_limit = 100;
   int count;
```

Initialization Termination Update

```
for(count = lower_limit; count < upper_limit; count ++) {
                printf("%d, ",count);     }
printf("\n");
}
```

Output

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35
, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46
, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57
, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68
, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79
, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90
, 91, 92, 93, 94, 95, 96, 97, 98, 99,
```

FOR LOOP- TYPE2

```
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;

        int count = lower_limit;

        for (; count < upper_limit; count++){
                printf ("%d, ", count);
        }
        printf ("\n");
}
```

Initialization

Termination

Update

Initialization replaced with a semicolon

FOR LOOP - TYPE3

```
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;

        int count = lower_limit;

        for (; count < upper_limit; ){

                printf ("%d, ", count);

                count++;
        }
        printf ("\n");
}
```

Note: Termination statement cannot be omitted from the for loop.

Initialization

Termination

Initialization replaced with a semicolon

Omitted the update statement

Update

FOR LOOP AND THE COMMA OPERATOR

```c
#include <stdio.h>

int main (){

        int i, j;
        int x_coord[10] = {1,2,3,4,5,6,7,8,9,10};
        int y_coord[10] = {1,4,9,16,25,36,49,64,81,100};
        for (i = 0, j = 0; i < 10 && j < 10; i++, j++){
                printf ("Plot: (%d, %d) \n", x_coord[i], y_coord[i]);
        }
        return 0;
}
```

Output

```
Plot: (1, 1)
Plot: (2, 4)
Plot: (3, 9)
Plot: (4, 16)
Plot: (5, 25)
Plot: (6, 36)
Plot: (7, 49)
Plot: (8, 64)
Plot: (9, 81)
Plot: (10, 100)
```

Notice ‘i’ and ‘j’ looping together in a single for loop separated by the comma operator

CONTD…

```
#include <stdio.h>

int main(){
    int i, j;

    for (i = 0, j = 10; i < 3 && j > 8; i++, j--){
        printf (" the value of i and j : %3d %3d\n",i, j);
    }
}
```

Output:

```
the value of i and j :  0   10
the value of i and j :  1    9
```

WHILE LOOP SYNTAX

```
while (testExpression) {
// the body of the loop }
```

How while loop works?

- The while loop evaluates the testExpression inside the parentheses ().
- If testExpression is true, statements inside the body of while loop are executed. Then, testExpression is evaluated again.
- The process goes on until testExpression is evaluated to false.
- If testExpression is false, the loop terminates.

EXAMPLE

```
int i = 0;

while (i < 5) {
  printf("%3d", i);
  i++;
}
```

output : 0 1 2 3 4

CONTD…

Notice the similarity

```c
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;

        int count = lower_limit;

        for (; count < upper_limit; ){

                printf ("%d, ", count);

                count++;
        }
        printf ("\n");
}
```

```c
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;

        int count = lower_limit;

        while (count < upper_limit){

                printf ("%d, ", count);

                count++;
        }
        printf ("\n");
}
```

Initialization

Termination

Update

PREDICT THE OUTPUT?

```
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;
        int count = lower_limit;
        while (count < upper_limit){
                printf ("%d, ", count);
        }
        printf ("\n");
}
```

WHILE (1) OR WHILE (TRUE) & BREAK STATEMENT

```
#include <stdio.h>

int main (){
        int lower_limit = 0;
        int upper_limit = 100;
        int count = lower_limit;
        while (1){
                if (count == upper_limit){
                        break;
                }
                else{
                        printf ("%d, ", count);
                        count++;
                }
        }
        printf ("\n");
}
```

Run forever

If the condition is satisfied break from the while loop

Otherwise, print count and update

DO- WHILE LOOP SYNTAX

```
do {
// code block to be executed
}while (condition);
```

How do...while loop works?

- The body of do...while loop is executed once. Only then, the condition is evaluated.
- If condition is true, the body of the loop is executed again and condition is evaluated once more.
- This process goes on until condition becomes false.
- If condition is false, the loop ends.

EXAMPLE

```
int i = 0;

do {
  printf("%3d", i);
  i++;
}
while (i < 5);
```

Output : 0 1 2 3 4

CONTD…

```c
#include <stdio.h>

int main (){

        int number = 3764;
        int copy = number; // Keep a copy of the number for arithmetic manipulation
        int count = 0;

        do{
                count++;
                copy = copy/10;
        } while (copy != 0);

        printf ("The number of digits in %d are %d\n", number, count);
}
```

Execute at least once

Start checking condition after executing at least once

Output

```
The number of digits in 3456 are 4
```

CONTD…

```c
#include <stdio.h>

int main (){

        int number = 3456;
        int copy = number; // Keep a copy of the number for arithmetic manipulation
        int count = 0;

        while (copy != 0)
        {
                count++;
                copy = copy/10;
        }

        printf ("The number of digits in %d are %d\n", number, count);
}
```

Output

```
The number of digits in 3456 are 4
```

CONTD…

```c
#include <stdio.h>

int main (){

        int number = 0;
        int copy = number; // Keep a copy of the number for arithmetic manipulation
        int count = 0;

        while (copy != 0)
        {
                count++;
                copy = copy/10;
        }

        printf ("The number of digits in %d are %d\n", number, count);
}
```

On which number will this program give an incorrect result ?

Output

```
The number of digits in 0 are 0
```

CONTD…

```c
#include <stdio.h>

int main (){

        int number = 0;
        int copy = number; // Keep a copy of the number for arithmetic manipulation
        int count = 0;

        do{
                count++;
                copy = copy/10;
        } while (copy != 0);

        printf ("The number of digits in %d are %d\n", number, count);
}
```

Output

```
The number of digits in 0 are 1
```

NESTED FOR LOOP

```c
int main (){

        int i, j = 0;
        const int limit = 5;
        for (i = 0; i<limit; i++){
                for (j = 0; j<limit; j++){
                        printf ("(%d, %d) ", i, j);
                }
                printf("\n");
        }
        return 0;
}
```

Matrix

(0,0)	(0,1)	…	…	(0,4)
(1,0)	:	:	:	(1,4)
:	:	:	:	:
:	:	:	:	:
(4,0)	…	…	…	(4,4)

Output

```
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4)
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4)
```

CONTINUE STATEMENT

```c
#include <stdio.h>

int main (){

        int i;
        for (i = 1; i < 100; i++){
                if (i%2 == 1){
                        printf ("%d ", i);
                }
        }
        printf ("\n");
        return 0;
}
```

(a)

```c
#include <stdio.h>

int main (){

        int i;
        for (i = 1; i < 100; i++){
                if (i%2 == 0){
                        continue;
                }
                else{
                        printf ("%d ", i);
                }
        }
        printf ("\n");
        return 0;
}
```

(b)

Both (a) and (b) print odd numbers between 1 and 100. Note that (a) has an implicit 'continue' when the condition is not satisfied. (b) has an explicit 'continue' on the even numbers.

Output

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27
 29 31 33 35 37 39 41 43 45 47 49 51
 53 55 57 59 61 63 65 67 69 71 73 75
 77 79 81 83 85 87 89 91 93 95 97 99
```

PRACTICE PROBLEMS ON LOOPING

1. Find the total number of digits in a given number.
2. Find the sum of digits of a number.
3. Find if a given number is an Armstrong number.
4. Find if a given number is prime.
5. Reverse a number.

THANK YOU