C Programming

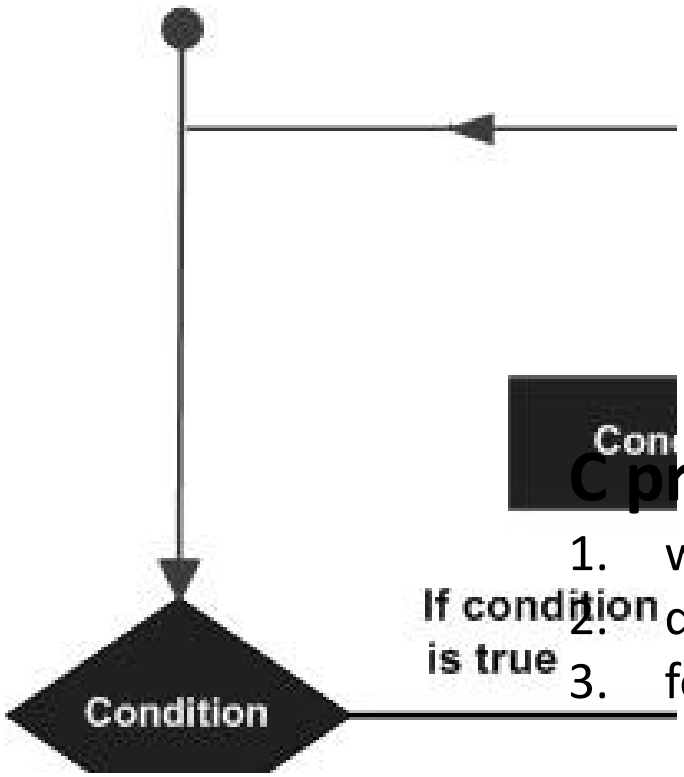
Iterative execution of code using Loops

Lecture-7-8-9-10

Dr. Asif Uddin Khan

Iterative execution of code using Loops

- A loop statement allows execution of a statement or group of statements multiple times.



C programming has three types of loops

1. while loop
2. do...while loop
3. for loop

while loop

- **while** loop repeatedly executes a target statements as long as a given condition is true.

Syntax

```
while(condition) {
    statement(s); // body
}
```

✓**statement(s)** may be a single statement or a block of statements.
✓The **condition** may be any expression, and true is any nonzero value.
✓The loop iterates while the condition is true.
✓When the condition becomes false, the program control passes to the line immediately following the loop.

Example 1: while loop

```
// Print numbers from 1 to 5
#include <stdio.h>
int main() {
int i = 1;
while (i <= 5) {
printf("%d\n", i);
++i;
}
return 0;
}
```

Output

```
1
2
3
4
5
```

Example 2: while loop

```
int main () {
  /* local variable definition */
  int a = 10;
  /* while loop execution */
  while( a < 20 ) {
    printf("value of a: %d\n", a);
    a++;
  }
  return 0;
}
```

Output

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

do...while loop

- The do..while loop is similar to the while loop with one important difference.
- The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

<u>Syntax</u>

```
do
{
    // statements inside the b
}
```

How do...while loop works?

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

Program using do...while loop

```
#include <stdio.h>
int main()
{
    double number, sum = 0;

    // the body of the loop is execu
    do
    {
        printf("Enter a number: ");
        scanf("%lf", &number);
        sum += number;
    }
```

for Loop

Syntax

```
for (initializationStatement; testExpress
{
    // statements inside the body of loop
```

```
// Print numbers fro
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i < 11
  {
```

- **How for loop works?**
- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.

Logic for adding the digits of a number

```
int sum,x,m;
sum=0;
while (x > 0) {
        m = x%10;
        sum = sum+m;
        x = x/10;
    }
```

Logic to find the reverse of a number

```
while(n!=0)
 {
   rem=n%10;
   reverse=reverse*10+rem;
   n=n/10;
 }
```

Nested Loops in C

- Looping of statements inside another loop
- Any number of loops can be defined inside another loop
- You can define any type of loop inside another loop; for example, you can define '**while**' loop inside a '**for**' loop.

```
Outer_loop
{
    Inner_loop
    {
        // inner loop
    }
```

Nested for loop

```
for (initialization; condition; update)
{
    for(initialization; condition; update)
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Program to print table from 1 to n

```
#include <stdio.h>
int main()
{
  int  i,j,n;// variable declaration
  printf("Enter the value of n :");
  scanf("%d",&n)
  // Displaying the n tables.
  for(i=1;i<=n;i++)  // outer loop
  {
     for(j=1;j<=10;j++)  // inner loop
     {
        printf("%d\t",(i*j)); // printing the value.
     }
     printf("\n");
  }
```

Output:

```
Enter the value of n : 3
1       2       3       4       5       6       7
2       4       6       8       10      12      14
3       6       9       12      15      18      21
```

Program for Half Pyramid of *

```
#include <stdio.h>
 int main() {
 int i, j, rows;
 printf("Enter the number of rows: ");
 scanf("%d", &rows);
 for (i = 1; i <= rows; ++i) {
   for (j = 1; j <= i; ++j) {
     printf("* ");
   }
   printf("\n");
 }
 return 0;
}
```

Output

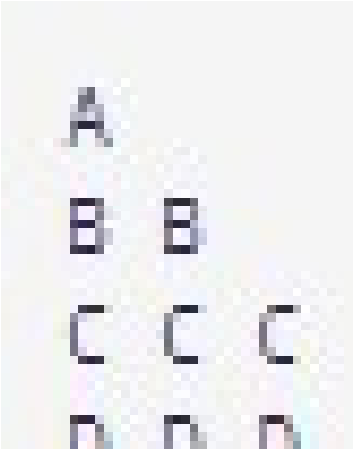
```
*
**
***
****
*****
```

Program for Half Pyramid of Numbers

```
#include <stdio.h>
int main() {
  int i, j, rows;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  for (i = 1; i <= rows; ++i) {
    for (j = 1; j <= i; ++j) {
      printf("%d ", j);
    }
    printf("\n");
  }
  return 0;
}
```

Program for Half Pyramid of Alphabets

```
#include <stdio.h>
int main() {
  int i, j;
  char input, alphabet = 'A';
  printf("Enter an uppercase character you want
    to print in the last row: ");
  scanf("%c", &input);
  for (i = 1; i <= (input - 'A' + 1); ++i) {
    for (j = 1; j <= i; ++j) {
      printf("%c ", alphabet);
    }
    ++alphabet;
    printf("\n");
  }
  return 0;
}
```



Nested while loop

```
while(condition)
{
    while(condition)
    {
        // inner loop statements.
    }
// outer loop statements.
}
```

Multiplication table printing using nested while loop

```
#include <stdio.h>
int main()
{
    printf("Multiplication table\n\n");
    int i=1,j;
    while(i<=10){
    j=1;
    while(j<=10){
            printf("%d\t",j*i);
            j++;
            }
    printf("\n");
    i++;
    }
return 0;
}
```

Output

1	2	3	4	5	6	7	8
2	4	6	8	10	12	14	1
3	6	9	12	15	18	21	2
4	8	12	16	20	24	28	
5	10	15	20	25	30	35	
6	12	18	24	30	36	42	
7	14	21	28	35	42	49	

Example of nested while loop

```
#include <stdio.h>
int main()
{
    int i=1;
    while(i<=3){
    printf("%d Outer Loop\n",i);
    int j=1;
    while(j<=4){
            printf("%d :Inner Loop\n",j);
            j++;
    }
    i++;
    }
    return 0;
}
```

Rectangular number pattern printing using nested while loop in C language

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i=1,j;
    while(i<=10){
    j=1;
    while(j<=10){
    printf("%d",j);
    j++;
}
    printf("\n");
    i++;
}
    return 0;
}
```

Output

```
123456789
123456789
123456789
123456789
123456789
123456789
123456789
```

Nested do while

```
statements
do{  // Outer do
        statements
            do{ // Inner do
                statements
                }while(condition); // Inner while
        statements
   }while(condition); //Outer while
```

Program using nested do while

```
#include <stdio.h>
int main()
{
    int i,j;
    i=1;
    printf("Square number p
    printf("Here your patte
 do{
    j=1;
 do{
    printf("%d",j);
    j++;
```

Output

```
Square number

Here your patte

12345678910
12345678910
12345678910
12345678910
12345678910
```

Program to displays a Floyd triangle number pattern using nested do-while loop

```
#include <stdio.h>
int main()
{
    int i,j;
    i=1;
    printf("Triangle number p
    printf("Here your patter
    do{
        j=1;
        do{
            printf("%d",j);
            j++;
```

Output

```
Triangle number p

Here your pattern

1
12
123
1234
12345
```

The comma operator in Loop

- C has a comma operator, that basically combines two statements so that they can be considered as a single statement.
- It provides multiple initializations or to allow for multiple incrementations.
- For example:

```
#include<stdio.h>
int main(){
        int i,j;
        for(i=1,j=1;i<10,j<10;i++,j++){
        printf("i = %d \t j = %d\n",i,j);
}
return 0;
}
```

Loop depends on last condition

Incrementing/Decrementing different values in loop

```
#include<stdio.h>
int main(){
        int i,j;
        for(i=1,j=1;i<10,j<10;i+=2,j++){
        printf("i = %d \t j = %d\n",i,j);
}
return 0;
}
```

Loop depends on last conditional expression

Infinite Loops

- Infinite loops are loops that repeat forever without stopping.
- Usually they are caused by some sort of error, such as the following example in which the wrong variable is incremented

Example

```
int i, j;
for( i = 0; i < 5; j++ )
    printf( "i = %d\n", i );
printf( "This line will never execute\n" );
```

- Other times infinite loops serve a useful purpose, such as this alternate means of checking user input:

```
while( true ) {
  printf( "Please enter a month from 1 to 12 > " );
  scanf( "%d", &month );
  if( month < 0 && month >13 )
   break;
  printf( "%d is not a valid month.\n Please try again.\n", month );
 }
```

Empty Loops

- A common error is to place an extra semi-colon at the end of the while or for statement, producing an empty loop body between the closing parenthesis and the semi-colon, such as:

```
int i;
for( i = 0; i < 5; i++ );
printf( "i = %d\n", i );  // This line is AFTER the loop, not inside it.
```

Empty and infinite Loop

```
int i = 0;
while( i < 5 );  // Error - empty loop on this line
printf( "i = %d\n", i++ ); // Again, this line is AFTER the loop.
```

GCD

```
for(i=1; i <= n1 && i <= n2; ++i)
{ // Checks if i is factor of both integers
   if(n1%i==0 && n2%i==0)
   gcd = i;
}
```

When to Use Which Loop ?

- If you know (or can calculate) how many iterations you need, then use a counter-controlled (for) loop.
- Otherwise, if it is important that the loop complete at least once before checking for the stopping condition, or if it is not possible or meaningful to check the stopping condition before the loop has executed at least once, then use a do-while loop.
- Otherwise use a while loop.

Break statement in C

- The break statement ends the loop immediately when it is encountered.

Syntax

break;

Example

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // c
    if (
        br
    }
    // c
}
while (
```

```
for (init; testExpression; updat
    // codes
```

Example program using break statement

- // Program to calculate the sum of numbers (10 numbers max).If the user enters a negative number, the loop terminates

```
#include <stdio.h>
int main() {
    int i;
    double number, sum = 0.0;

    for (i = 1; i <= 10; ++i) {
        printf("Enter a n%d: ", i);
        scanf("%lf", &number);

        // if the user enters a negative nu
        if (number < 0.0) {
            break;
        }
```

C continue statement

- The continue statement skips the current iteration of the loop and continues with the next iteration.
- <u>syntax:</u> **continue;**

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    //
    if
    }
    //
}
while
```

```
for (init; testExpression; upda
    // codes
```

Example: continue statement

```
// Program to calculate the sum of numbers (10
// If the user enters a negative number, it's n

#include <stdio.h>
int main() {
    int i;
    double number, sum = 0.0;

    for (i = 1; i <= 10; ++i) {
        printf("Enter a n%d: ", i);
        scanf("%lf", &number);

        if (number < 0.0) {
            continue;
        }
```

goto Statement

- The goto statement transfer control of the program to the specified label.
- The label is an identifier.
- When the goto statement is encountered, the control of the program jumps to label: and starts executing the code.

```
goto labe
... .. ..
... .. ..

label:
```

Example-1: goto Statement

```
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
  return 0;
}
```

Example-2: goto Statement

```
// Program to calculate the sum and average of pos:
// If the user enters a negative number, the sum ar
#include <stdio.h>
int main() {
    const int maxInput = 10;
    int i;
    double number, average, sum = 0.0;
    for (i = 1; i <= maxInput; ++i) {
        printf("%d. Enter a number: ", i);
        scanf("%lf", &number);

        // go to jump if the user enters a negative r
        if (number < 0.0) {
            goto jump;
        }
        sum += number;
```

References

1. C programming by E Balaguruswami
2. Programming C by Y. kanitkar
3. Programming C by Denis Ritchie
4. NPTEL Lecture note of Dr. Partha Pratim Das, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur.
5. https://docs.oracle.com/cd/E18752_01/html/817-6223/chp-typeopexpr-2.html
6. https://data-flair.training/blogs/escape-sequence-in-c/
7. Internet source