Control Structures

Sometimes a block of code needs to execute several times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times.

1] if statement:
This is simplest form of decision making statements in C. Depending upon the result (True or False), the program execution proceeds in one direction or another.

Syntax:

```c
if (expression)
    statement;
```

The statement is executed only when condition is true. If the if statement body is consists of several statement then better to use pair of curly braces. Here in case condition is false then compiler skip the line within the if block.

```c
void main()
{
    int n;
    printf (" enter a number: ");
    scanf("%d", &n);
    if (n > 10)
        printf(" number is greater");
}
```

Output:
Enter a number: 12
Number is greater

2] if.....else ... Statement
It is bidirectional conditional control statement that contains one condition & two possible action. Condition may be true or false, where non-zero value regarded as true & zero value regarded as false. If condition are satisfy true, then a single or block of statement executed otherwise another single or block of statement is executed.
Its syntax is:-

```c
if (expression)
{
    Statement1;
    Statement2;
}
else
{
    Statement1;
```
Else statement cannot be used without if or no multiple else statement are allowed within one if statement. It means there must be one if statement within an else statement.

Example:

Write a program to check a number is even or odd

```c
void main()
{
    int n;
    printf("enter a number:");
    scanf("%d", &n);
    if (n%2==0)
        printf("even number");
    else
        printf("odd number");
}
```

Output: enter a number:121
odd number

3] Nesting of if ...else

When there are another if else statement in if-block or else-block, then it is called nesting of if-else statement.

Syntax is :-
if (condition)
{
    if (condition)
        statement1;
    else
        statement2;
}
statement3;

4] If....else LADDER

In this type of nesting there is if else statement in every else part except the last part. If condition is false control pass to block where condition is again checked with its if statement.

Syntax is :-
if (condition)
    statement1;
else if (condition)
    statement2;
else if (condition)
    statement3;
else
  statement4;

This process continue until there is no if statement in the last block. If one of the condition is satisfied then condition in other nested “else if” would not executed. But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all conditions are checked.

5] while Loop

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax:

```
while(expression)
{
  statement(s);
}
```

Here, 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.

```
#include <stdio.h>
int main()
{
  int a = 5;
  while( a < 11 )
  {
    printf("value of a: %d\n", a);
    a++;
  }
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
value of a: 10
6] for Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

```c
for ( initialization; condition; increment/decrement )
{
    statement(s);
}
```

Here is the flow of control in a ‘for’ loop:
1] The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

2] Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the ‘for’ loop.

3] After the body of the ‘for’ loop executes, the flow of control jumps back up to the increment/decrement statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

4] The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the ‘for’ loop terminates.

```c
#include <stdio.h>
int main()
{
    int a;
    for(a = 5; a < 11; a = a + 1 )
    {
        printf("value of a: %d\n", a);
    }
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
value of a: 10
7] do...while Loop

Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop. A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax:

do
{
    statement(s);
}while( condition );

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

#include <stdio.h>
int main()
{
    int a = 5;
    do
    {
        printf("value of a: %d\n", a);
        a = a + 1;
    }while( a < 11 );
    return 0;
}

When the above code is compiled and executed, it produces the following result:
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
value of a: 10

8] Nested Loops

C programming allows one loop inside another loop. The following section shows a few examples to illustrate the concept.

Syntax:

for ( initialization; condition; increment /decrement)
{
    for (initialization; condition; increment /decrement )
    {
        statement(s);
    }
    statement(s);
}
Example
Write a program to find the prime numbers from 2 to 20:
```c
#include <stdio.h>
int main()
{
    int i, j;
    for(i=2; i<20; i++)
    {
        for(j=2; j <= (i/j); j++)
        {
            if(!(i%j))
                break; // if factor found, not prime
        }
        if(j > (i/j))
            printf("%d is prime\n", i);
    }
    return 0;
}
```
When the above code is compiled and executed, it produces the following result:
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime

Break statement (break)
-- Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements.
-- It cause immediate exit from that loop in which it appears and it is generally written with condition. It is written with the keyword as break.
-- When break statement is encountered loop is terminated and control is transferred to the statement, immediately after loop or situation where we want to jump out of the loop instantly without waiting to get back to conditional state.
-- When break is encountered inside any loop, control automatically passes to the first statement after the loop. This break statement is usually associated with if statement.

Syntax:
```c
    break;
```

Example:
```c
void main()
{
    int j;
    for(j=0; j<6; j++)
    {
```
if(j==4)
{
    break;
}
}
Output: 0 1 2 3

**Continue Statement**

---
Continue statement is used for continuing next iteration of loop after skipping some statement of loop.
---
When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program.
---
The difference between break and continue is, when the break encountered loop is terminated and it transfer to the next statement and when continue is encountered control come back to the beginning position.
---
In while and do while loop after continue statement control transfer to the test condition and then loop continue whereas in for loop after continue control transferred to the updating expression and condition is tested.

Syntax

```c
continue;
```

Example:-

```c
int main()
{
    int i;
    for(i=2; i<=9; i++)
    {
        if(i==4)
            continue;
        printf("%d", i);
    }
    printf("out of loop");
}
```

Output: 2 3 5 6 7 8 9 out of loop

**The Infinite Loop**

A loop becomes an infinite loop if a condition never becomes false. The `for` loop is traditionally used for this purpose. Since none of the three expressions that form the `for` loop are required, you can make an endless loop by leaving the conditional expression empty.

```c
#include <stdio.h>
int main()
{
    for( ; ; )
    {
```
printf("This loop will run forever.\n");
}
return 0;
}

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

**NOTE:** You can terminate an infinite loop by pressing Ctrl + C keys.