An alternative to the if, else if, else control structure is the switch-case structure.

switch statement A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements You can use one switch statement inside another switch statement(s).

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case.**

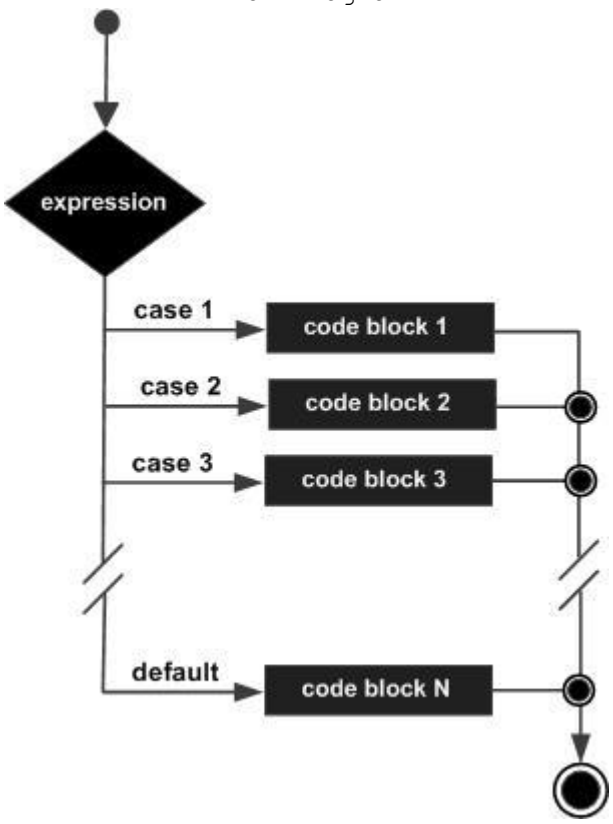
The syntax for a **switch** statement in C programming language is as follows:

```
switch(expression)
{
   case constant-expression:
      statement(s);
      break; /* optional */

   case constant-expression:
      statement(s);
      break; /* optional */

   /* you can have any number of case statements */
   default: /* Optional */
   statement(s);
}
```

The following rules apply to a **switch** statement:

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Flow Diagram



It is possible to have a switch as a part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

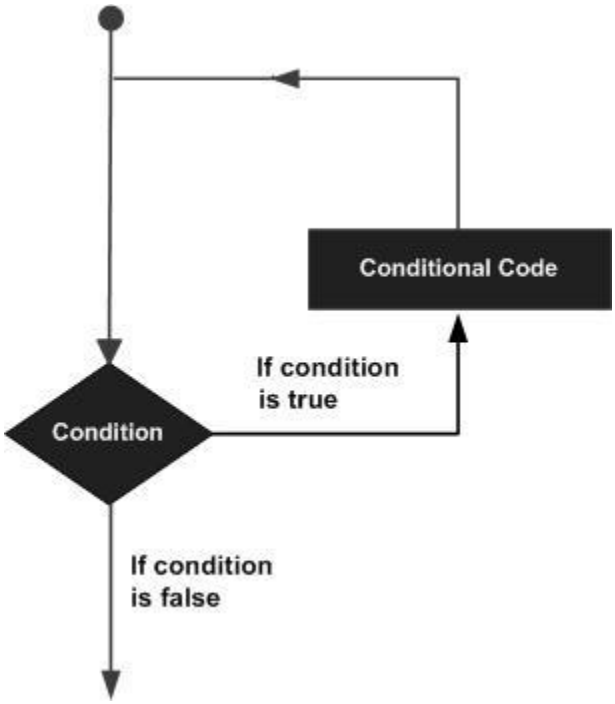
The syntax for a **nested switch** statement is as follows:

```
switch(ch1)
{
   case 'A':
      printf("This A is part of outer switch" );

      switch(ch2) {
         case 'A':
            printf("This A is part of inner switch" );
            break;
         case 'B': /* case code */
      }

      break;
   case 'B': /* case code */
}
```

You may encounter situations, when a block of code needs to be executed several number of 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. Given below is the general form of a loop statement in most of the programming languages:



C programming language provides the following types of loops to handle looping requirements:

while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop It is more like a while statement, except that it tests the condition at the end of the loop body.
nested loops You can use one or more loops inside any other while, for, or do..while loop.

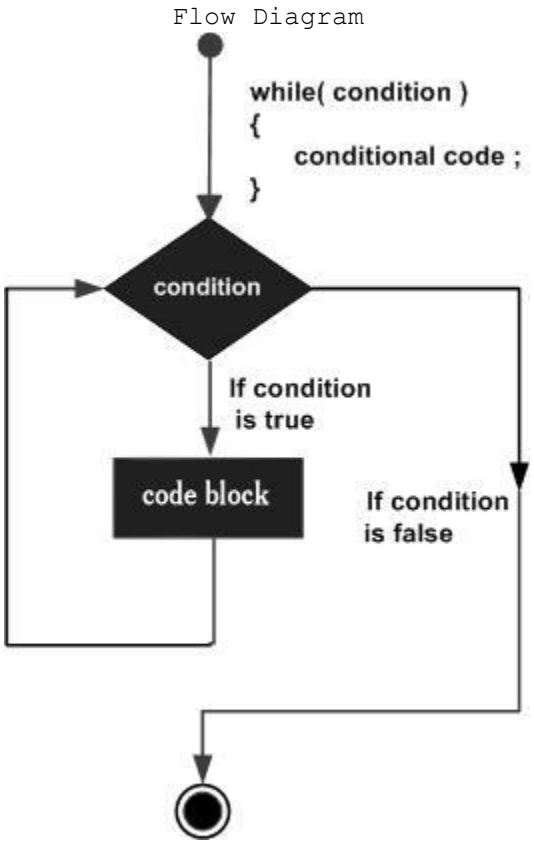
A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

The syntax of a **while** loop in C programming language is:

```
while(condition)
{
   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.



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

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.

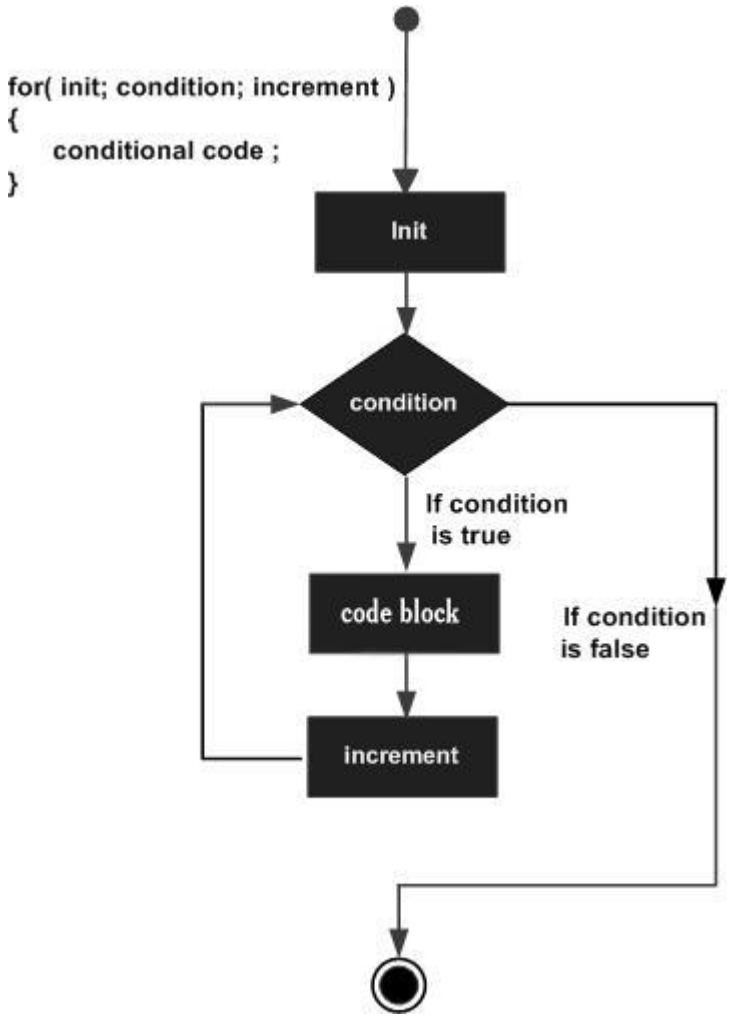
The syntax of a **for** loop in C programming language is:

```
for (init; condition; increment)
{
   statement(s);
}
```

Here is the flow of control in a 'for' loop:

- The **init** 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.
- 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.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** 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.
- 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.

Flow Diagram



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.

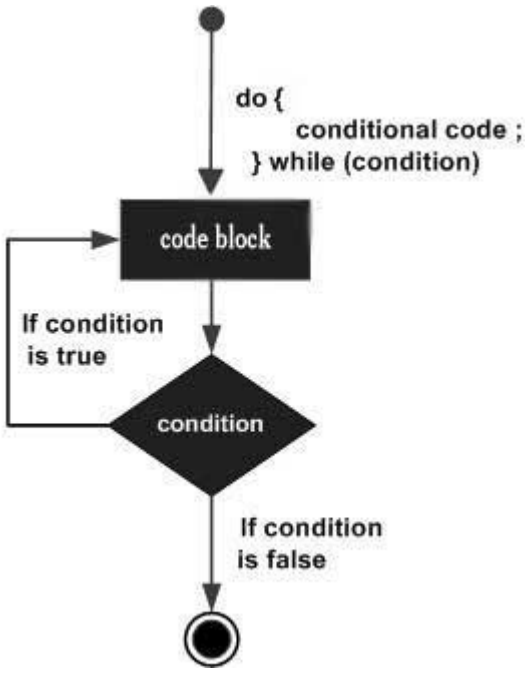
The syntax of a **do...while** loop in C programming language is:

```
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.

Flow Diagram



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

The syntax for a **nested for loop** statement in C is as follows:

```
for (init; condition; increment)
{
   for (init; condition; increment)
   {
      statement(s);
   }
   statement(s);
}
```

The syntax for a **nested while loop** statement in C programming language is as follows:

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

The syntax for a **nested do...while loop** statement in C programming language is as follows:

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

A final note on loop nesting is that you can put any type of loop inside any other type of loop. For example, a 'for' loop can be inside a 'while' loop or vice versa.

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements:

break statement Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

The **break** statement in C programming has the following two usages:

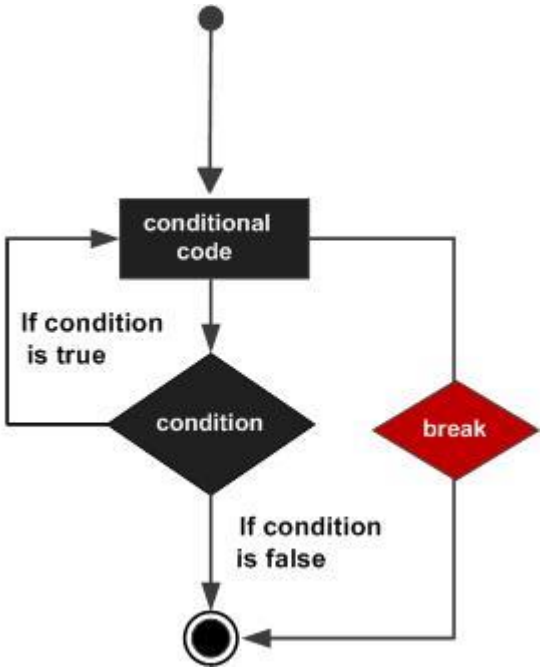
- When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the **switch** statement (covered in the next chapter).

If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

The syntax for a **break** statement in C is as follows:

```
break;
```

Flow Diagram



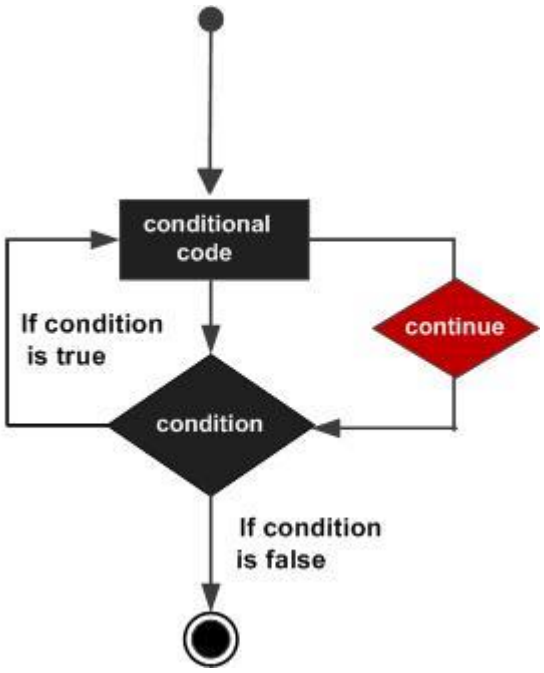
The **continue** statement in C programming works somewhat like the **break** statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control to pass to the conditional tests.

The syntax for a **continue** statement in C is as follows:

```
continue;
```

Flow Diagram



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.

```
#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.