C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios: With switch case

1. With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Flowchart of break in c

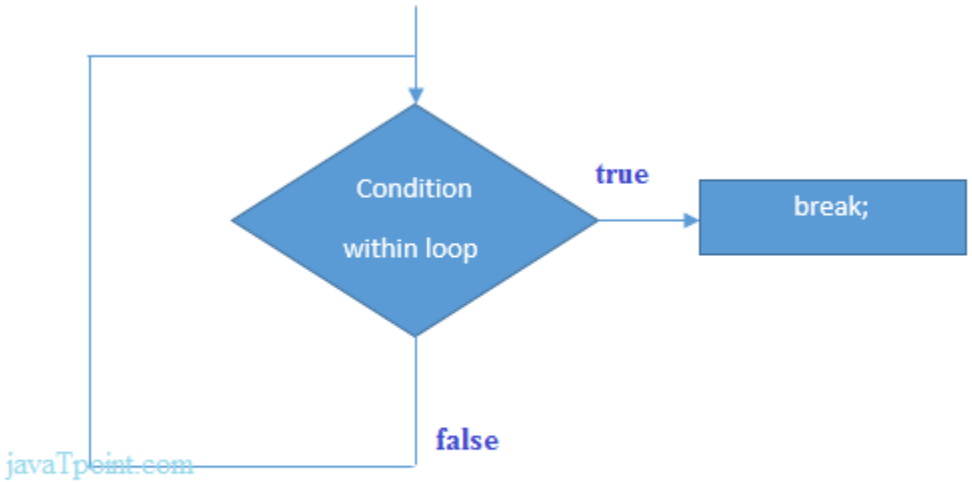

Figure: Flowchart of break statement

Example

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. void main ()
4. {
```

```
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
        break;
    }
    printf("came outside of loop i = %d",i);

}
```

Output

```
0 1 2 3 4 5 came outside of loop i = 5
```

Example of C break statement with switch case

C break statement with the nested loop

In such case, it breaks only the inner loop, but not outer loop.

```
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
printf("%d &d\n",i,j);
if(i==2 && j==2){
break;//will break loop of j only
}
}//end of for loop
return 0;
}
```

Output

```
1 1
1 2
1 3
2 1
2 2
3 1
3 2
3 3
```

As you can see the output on the console, 2 3 is not printed because there is a break statement after printing i==2 and j==2. But 3 1, 3 2 and 3 3 are printed because the break statement is used to break the inner loop only.

break statement with while loop

Consider the following example to use break statement inside while loop.

```
1. #include<stdio.h>
2. void main ()
3. {
4.     int i = 0;
5.     while(1)
6.     {
7.         printf("%d  ",i);
8.         i++;
9.         if(i == 10)
10.        break;
11.    }
12.    printf("came out of while loop");
13.}
```

Output

```
0  1  2  3  4  5  6  7  8  9  came out of while loop
```

break statement with do-while loop

Consider the following example to use the break statement with a do-while loop.


```
1. #include<stdio.h>
```

```
2.  void main ()
3.  {
4.     int n=2,i,choice;
5.     do
6.     {
7.        i=1;
8.        while(i<=10)
9.        {
10.          printf("%d X %d = %d\n",n,i,n*i);
11.          i++;
12.       }
13.       printf("do you want to continue with the table of %d , enter any non-
      zero value to continue.",n+1);
14.       scanf("%d",&choice);
15.    if(choice == 0)
16.       {
17.          break;
18.       }
19.       n++;
20.   }while(1);
21.}
```

Output

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
do you want to continue with the table of 3 , enter any non-zero value to
continue.1
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
do you want to continue with the table of 4 , enter any non-zero value to
continue.0
```

C continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
1. //loop statements
2. continue;
3. //some lines of the code which is to be skipped
```

Continue statement example 1

```
1. #include<stdio.h>
2. void main ()
3. {
4.     int i = 0;
```

```
5.      while(i!=10)
6.      {
7.          printf("%d", i);
8.          continue;
9.          i++;
10.     }
11.}
```

Output

```
infinite loop
```

Continue statement example 2

```
1.  #include<stdio.h>
2.  int main(){
3.  int i=1;//initializing a local variable
4.  //starting a loop from 1 to 10
5.  for(i=1;i<=10;i++){
6.  if(i==5){//if value of i is equal to 5, it will continue the loop
7.  continue;
8.  }
9.  printf("%d \n",i);
10.}//end of for loop
11.return 0;
12.}
```

Output

```
1
2
3
4
6
7
8
9
10
```

As you can see, 5 is not printed on the console because loop is continued at i==5.

C continue statement with inner loop

In such case, C continue statement continues only inner loop, but not outer loop.

```
1. #include<stdio.h>
2. int main(){
3. int i=1,j=1;//initializing a local variable
4. for(i=1;i<=3;i++){
5. for(j=1;j<=3;j++){
6. if(i==2 && j==2){
7. continue;//will continue loop of j only
8. }
9. printf("%d %d\n",i,j);
10.}
11.}//end of for loop
12.return 0;
13.}
```

Output

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

As you can see, 2 2 is not printed on the console because inner loop is continued at i==2 and j==2

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complicated.

Syntax:

```
label:
//some part of the code;
goto label;
```

goto example

Let's see a simple example to use goto statement in C language.

```
#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;


    }
```

Output:

```
Enter the number whose table you want to print?10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

When should we use goto?

The only condition in which using goto is preferable is when we need to break the multiple loops using a single statement at the same time. Consider the following example.

```
#include <stdio.h>
int main()
{
  int i, j, k;
  for(i=0;i<10;i++)
  {
    for(j=0;j<5;j++)
    {
      for(k=0;k<3;k++)
      {
        printf("%d %d %d\n",i,j,k);
        if(j == 3)
        {
          goto out;
        }
      }
    }
  }
  out:
  printf("came out of the loop");
}
```

Output

```
0 0 0
0 0 1
0 0 2
0 1 0
0 1 1
0 1 2
0 2 0
0 2 1
0 2 2
0 3 0
came out of the loop
```