do while loop in C

A ***loop*** is a programming control structure that allows you to execute a ***block of code*** indefinitely if a specific condition is met. Loops are used to execute repeating activities and boost programming performance. There are multiple loops in the C programming language, one of which is the ***"do-while" loop***.

A ***"do-while" loop*** is a form of a ***loop*** in C that executes the code block first, followed by the condition. If the condition is ***true***, the ***loop*** continues to run; else, it stops. However, whether the condition is originally ***true***, it ensures that the code block is performed at least once.

do while loop syntax

The syntax of the C language do-while loop is given below:

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

The components are divided into the following:

- The ***do keyword*** marks the beginning of the Loop.
- The ***code block*** within ***curly braces {}*** is the body of the loop, which contains the code you want to repeat.
- The ***while keyword*** is followed by a condition enclosed in parentheses (). After the code block has been run, this condition is verified. If the condition is ***true***, the loop continues else, the ***loop ends***.

Working of do while Loop in C

Let us look at an example of how a ***do-while loop*** works in C. In this example, we will write a simple program that questions the user for a ***password*** and keeps asking until the right password is input.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
char password[] = "secret";
```

```
5. char input[20];
6. do {
7. printf("Enter the password: ");
8. scanf("%s", input);
9. } while (strcmp(input, password) != 0);
10. printf("Access granted!\n");
11. return 0;
12. }
```

The program runs as follows:

1. The following header files are included: ***<stdio.h>*** for standard ***input*** and ***output*** routines and ***<string.h>*** for string ***manipulation functions***.
2. The correct password is defined as a ***character array (char password[])*** with the value ***"secret"***
3. After that, we define another character array input to store the user's input.
4. The ***do keyword*** indicates that the code block included within the ***loop*** will be performed at least once.
5. Using the ***printf() function***, we display a prompt requesting the user to input their password inside the Loop.
6. Next, we read the ***user's input*** using the ***scanf() function*** and store it in the ***input array***.
7. After reading the ***input***, we use the ***strcmp() function*** to compare the input with the correct password. If the strings are ***equal,*** the ***strcmp function*** returns 0. So, we continue looping as long as the input and the password are not equal.
8. Once the ***correct password*** is entered, the loop terminates, and we print ***"Access granted!"*** using the ***printf() function***.
9. After that, the program returns 0 to indicate successful execution.

Output:

Let us walk through a possible scenario:

```
Enter the password: 123
Enter the password: abc
Enter the password: secret
Access Granted!
```

Explanation:

In this example, the user initially enters the wrong passwords, ***"123"*** and ***"abc"***. The loop prompts the user until the correct password ***"secret"*** is entered. Once the correct password is provided, the loop terminates, and the ***"Access granted!"*** message is displayed.

Example of do while loop in C:

Example 1:

Here is a simple example of a ***"do-while" loop*** in C that prints numbers from 1 to 5:

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

Output:

```
1
2
3
4
5
```

Explanation:

In this example, the ***code block*** within the do loop will be executed at least once, printing numbers from ***1 to 5***. After each iteration, the ***i value*** is incremented, and the condition ***i<= 5*** is checked. If the condition is still true, the loop continues; otherwise, it terminates.

Example 2:

Program to print table for the given number using do while Loop

```
#include<stdio.h>
intmain(){
```

```
3. inti=1,number=0;
4. printf("Enter a number: ");
5. scanf("%d",&number);
6. do{
7. printf("%d \n",(number*i));
8. i++;
9. }while(i<=10);
10. return 0;
11. }
```

Output:

```
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
Enter a number: 10
10
20
30
40
50
60
70
80
90
100
```

Example 3:

Let's take a program that prints the multiplication table of a given number N using a ***do...while Loop*:**

```
1. #include <stdio.h>
2. int main() {
int N;
printf("Enter a number to generate its multiplication table: ");
scanf("%d", &N);
inti = 1;
do {
```

```
printf("%d x %d = %d\n", N, i, N * i);
i++;
} while (i<= 10);
3. return 0;
4. }
```

Output:

Let us say you enter the number 7 as input:

```
Please enter a number to generate its multiplication table: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

The program calculates and prints the multiplication table for ***7*** from 1 to 10.

Infinite do while loop

An ***infinite loop*** is a loop that runs indefinitely as its condition is always ***true*** or it lacks a terminating condition. Here is an example of an ***infinite do...while loop*** in C:

Example:

```
1. #include <stdio.h>
2. int main() {
3. inti = 1;
4. do {
5. printf("Iteration %d\n", i);
6. i++;
7. } while (1); // Condition is always true
8.
9. return 0;
10. }
```

In this ***example***, the ***loop*** will keep running ***indefinitely*** because ***condition 1*** is always ***true***.

Output:

When you run the program, you will see that it continues printing ***"Iteration x",*** where x is the ***iteration number*** without stopping:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
... (and so on)
```

To interrupt an infinite loop like this, you generally use a ***break statement*** within the ***loop*** or some external condition you can control, such as ***hitting*** a specific key combination. In most desktop settings, the keyboard shortcut ***Ctrl+C*** can escape the Loop.

Nested do while loop in C

In C, we take an example of a ***nested do...while loop***. In this example, we will write a program that uses ***nested do...while loops*** to create a numerical pattern.

Example:

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

In this program, we use ***nested do...while loops*** to generate a pattern of numbers. The ***outer loop*** controls the number of rows, and the ***inner loop*** generates the numbers for each row.

Output:

Let us say you input five as the number of rows:

```
Enter the number of rows: 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Explanation:

In this example, the program generates a pattern of numbers in a ***triangular shape***. The ***outer loop*** iterates over the rows, and the ***inner loop*** iterates within each row, printing the numbers from 1 up to the current row number.

Difference between while and do while Loop

Here is a tabular comparison between the while loop and the do-while Loop in C:

Aspect	while loop	do-while loop
Syntax	while (condition) { ... }	do { ... } while (condition);
Loop Body Execution	Condition is checked before execution.	The body is executed before the condition.
First Execution	The condition must be true initially.	The body is executed at least once.
Loop Execution	May execute zero or more times.	Will execute at least once.
Example	while (i< 5) { printf("%d\n", i); i++; }	do { printf("%d\n", i); i++; } while (i< 5);
Common Use Cases	When the loop may not run at all.	When you want the loop to run at least once.

While Loop: The loop body is executed before the condition is checked. If the condition is initially ***false***, the loop may not execute.

Do-while Loop: The ***loop body*** is executed at least once before the condition is ***checked***. This guarantees that the loop completes at least one iteration.

When you want the ***loop*** to run based on a condition that may be ***false*** at first, use the ***while loop***, and when you want the loop to run at least once regardless of the starting state, use the ***do-while loop.***

Features of do while loop

The do-while loop in C has several fundamental characteristics that make it an effective programming technique in certain situations. The following are the significant characteristics of the do-while loop:

- **Guaranteed Execution:** Unlike other ***loop structures***, the ***do-while oop*** ensures that the loop body is executed at least once. Because the condition is assessed after the loop body, the code within the loop is performed before the condition is verified.
- **Loop after testing:** The ***do-while loop*** is a post-tested loop which implies that the loop condition is assessed after the loop body has been executed. If the condition is true, the loop body is run once again. This behavior allows you to verify the condition for repetition before ensuring that a given activity is completed.
- **Conditionally Controlled:** The loop continues to execute as long as the condition specified after the while keyword remains ***true***. When the condition evaluates to ***false***, the loop is terminated, and control shifts to the sentence after the loop.
- **Flexibility:** The ***do-while loop*** may be utilized in several contexts. It is typically used in cases where a piece of code must be executed at least once, such as ***menu-driven programs, input validation,*** or ***repetitive computations***.
- **Nesting Capability:** Similar to other ***loop constructs***, the ***do-while loop*** can be ***nested*** inside other ***loops*** or ***control structures*** to create more complex control flow patterns. It allows for the creation of ***nested loops*** and the implementation of intricate repetitive tasks.
- **Break and Continue:** The break statement can be used within a ***do-while loop*** to terminate the loop execution and exit the loop prematurely. The ***continue statement*** can skip the remaining code in the current iteration and jump to the next iteration of the loop.

- **Local Scope:** Variables declared inside the ***do-while loop*** body have local scope and are accessible only within the ***loop block.*** They cannot be accessed outside the loop or by other loops or control structures.
- **Infinite Loop Control:** It is crucial to ensure that the loop's condition is eventually modified within the ***loop body***. This modification is necessary to prevent infinite loops where the condition continually evaluates to true. Modifying the condition ensures that the loop terminates at some point.