C Programming

Loops (Iteration)

There are going to be times when you want code in a program to execute more than once. All programming languages support the use of loops (iteration) to facilitate code executing more than once.

In C, there are 3 different loops. These are as follows:

1. The while loop

The while loop is as follows:

```
while (condition)
{
        statement 1;
        …
        …
        statement n;
}
```

Let's look at the following code example using a while loop:

```c
/*
Program to demonstrate the use of a while loop. It will keep a running
total of numbers entered by the user
*/

#include <stdio.h>

int main()
{
      float num = 1;
      float total = 0;

      // while the number entered is not equal to zero
      while (num != 0)
      {
```

```
            printf("\nEnter a number (enter 0 to end program)\n");
            scanf("%f", &num);

            /* Add the number entered to the current value in total.
This is the new total
            */
            total = total + num;

      } // end while

      // Display the final total
      printf("\nThe final total is %.1f", total);

      return 0;
}
```

Repl 5.1: https://replit.com/@michaelTUDublin/51-while-loop#main.c

You **MUST** ensure that the variable used in the condition, i.e., while (condition), changes in the loop code block statements. Otherwise, the condition will always be True, and the loop will never end. This is called an Infinite Loop.

2. The do .. while loop

The do .. while loop is as follows:

```
do
{
      statement 1;
      …
      …
      statement n;
}
while (condition);  // You MUST place a semi-colon at the end of the do .. while
```

Let's look at the following code example using a while loop:

```
/*
Program to demonstrate the use of a do .. while loop. It will keep a
running total of numbers entered by the user
```

```
*/

#include <stdio.h>

int main()
{
	float num = 1;
	float total = 0;

	// this do loop will execute at least once
	do
	{
		printf("\nEnter a number (enter 0 to end program)\n");
		scanf("%f", &num);

		// if the user enters zero
		if (num == 0)
		{
			printf("\nBye bye");
			break;
		} // end if

		/* Add the number entered to the current value in total.
This is the new total
		*/
		total = total + num;

	} // end do
	// while the number entered is not equal to zero
	while (num != 0);

	// Display the final total
	printf("\nThe final total is %.1f", total);

	return 0;
} // end main
```

Note: note that there is a semi-colon after the condition.
Repl 5.2: https://replit.com/@michaelTUDublin/52-do-while-loop#main.c

Q: What is the primary difference between a regular while loop and a do .. while loop?

A: In a regular while loop, there is NO guarantee the statements will execute once. However, a do .. while loop DOES guarantee to execute the statements at least once

3. The for loop

The for loop is as follows:

```
for (initial_value;  condition;  counter)
{
        statement 1;
        …
        …
        statement n;
}
```

Let's take a look at an example:

```
/*
Program to demonstrate the use of a for loop. It will display a set of
numbers 1 - 10
*/

#include <stdio.h>

int main()
{
      int i;

      // Display the numbers 1 - 10
      for(i = 1; i < 11; i++)
      {
            // Display the contents of the variable i
            printf("%d\n", i);
      }
```

```
	return 0;

} // end main()
```

Repl 5.3: https://replit.com/@michaelTUDublin/53-for-loop#main.c

Nested loops

Let's have a look at using a nested for loop. This is a for loop inside another for loop.

Have a look at this example:

```
/*
Program to demonstrate a nested for loop
*/
#include <stdio.h>

int main()
{
	int i;
	int j;

	// This is the outer for loop
	for(i = 0; i < 3; i++)
	{
		// This is the inner for loop
		for(j = 0; j < 3; j++)
		{
			// Display the value of i and j index variables
			printf("i is %d, j is %d\n", i, j);
		} // end inner for

	} // end outer for

	return 0;

} // end main()
```

Repl 5.4: https://replit.com/@michaelTUDublin/54-Nested-for-loop#main.c

Programming pitfall

1. There is no semi-colon at the end of a while loop or for loop

e.g.,

```
int num = 1;

while (num < 10); // You must remove the semi-colon here
{
        printf("num is %d\n", num);

        num++;
}
```

2. You must be careful how you specify the termination condition in a loop.

e.g.,

```
/*
Program to demonstrate a nested for loop
*/
#include <stdio.h>

int main()
{
      int i;

      // redundant loop because the condition is immediately false
      for (i = 0; i == 10; i++)
      {
            printf("i is %d \n", i);

      } // end for

      return 0;
}
```

Make sure that your condition is not false at the very start (as above) in which case the code inside the loop will never execute.

3. Always use a whole number data type as your index variable, i.e., short, int or long. The default data type is an int