Introduction to Programming

Week – *4*, Lecture – *1*

Loops in C – `while` and `do-while`

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

Using loops to solve problems

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    Condition over one or more variables (to go out of the loop)
}
Statements after the loop
```

Where shall we put the loop's condition?

We can put it at the end of the of the loop

Where shall we put the loop's condition?

We can put it at the end of the of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    Condition over one or more variables (to go out of the loop)
}
Statements after the loop
```

Where shall we put the loop's condition?

We can put it at the end of the of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    Condition over one or more variables (to go out of the loop)
}
Statements after the loop
```

Where shall we put the loop's condition?

We can put it at the end of the of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    Condition over one or more variables (to go out of the loop)
}
Statements after the loop
```

In this case, the loop's body is executed at least once – because the condition, even if it is false, is evaluated at the end of the loop

The `do-while` loop in C

```
Statements before the loop
do
{
    Statements to execute in the loop
}
while(condition to "stay" in the loop);
Statements after the loop
```

The `do-while` loop in C

```
Statements before the loop
do
{
    Statements to execute in the loop
}
while(condition to "stay" in the loop);
Statements after the loop
```

Remember that if you have worked out the condition to go out of loop, if you apply a `!` operator in front of it, it becomes the condition to stay in the loop

Factorial with `do-while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        if(num > 0)
        {
                do
                {
                        result *= num;
                        // This is a short-cut to write
                        // result = result * num;
                        num -= 1;
                        // This is a short-cut to write
                        // num = num - 1;
                }
                while(num > 1);
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
~
```

Factorial with `do-while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        if(num > 0)
        {
                do
                {
                        result *= num;
                        // This is a short-cut to write
                        // result = result * num;
                        num -= 1;
                        // This is a short-cut to write
                        // num = num - 1;
                }
                while(num > 1);
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

The loop continues, till the value of `num` is greater than `1`

Factorial with `do-while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        if(num > 0)
        {
                do
                {
                        result *= num;
                        // This is a short-cut to write
                        // result = result * num;
                        num -= 1;
                        // This is a short-cut to write
                        // num = num - 1;
                }
                while(num > 1);
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

We need this condition to avoid getting inside the loop's body, when `num` is `0`

Where shall we put the loop's condition?

We can also put the condition at the beginning of the loop

Where shall we put the loop's condition?

We can also put the condition at the beginning of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Condition over one or more variables (to go out of the loop)
    Statements, using different values of one or more variables
    Update to the values of one or more variables
}
Statements after the loop
```

Where shall we put the loop's condition?

We can also put the condition at the beginning of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Condition over one or more variables (to go out of the loop)
    Statements, using different values of one or more variables
    Update to the values of one or more variables
}
Statements after the loop
```

Where shall we put the loop's condition?

We can also put the condition at the beginning of the loop

```
Statements before the loop
Initialisation – setting values for important variables
{
    Condition over one or more variables (to go out of the loop)
    Statements, using different values of one or more variables
    Update to the values of one or more variables
}
Statements after the loop
```

In this case, it is possible that the loop is never executed at all, if the condition is false, when evaluated for the first time itself !!

The `while` loop in C

```
Statements before the loop
while(condition to "stay" in the loop)
{
    Statements to execute in the loop
}
Statements after the loop
```

The `while` loop in C

```
Statements before the loop
while(condition to "stay" in the loop)
{
    Statements to execute in the loop
}
Statements after the loop
```

The `while` loop in C

```
Statements before the loop
while(condition to "stay" in the loop)
{
    Statements to execute in the loop
}
Statements after the loop
```

If the condition is false when the control reaches the while statement, it essentially behaves like an `if` statement – i.e. the control passes to the statements after the loop

Factorial with `while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

Factorial with `while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

The control enters the loop if `num` is greater than `1`, and continues to execute inside the loop, till `num` is greater than `1`

Factorial with `while` loop

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        printf("Give me a small positive integer: ");
        scanf("%d", &num);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

The control enters the loop if `num` is greater than `1`, and continues to execute inside the loop, till `num` is greater than `1`

We also don't need an additional `if` condition here to avoid execution of the loop when `num` is `0`

When to use which loop?

In theory, both loops are equivalent

- This means that you can choose either for a particular case

When to use which loop?

In theory, both loops are equivalent

- This means that you can choose either for a particular case

However, usually we use do-while loop when we want the loop to execute at least once

- There are cases like these, by the way…

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

We use the `do-while` loop to seek a positive integer from the user

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

We use the `do-while` loop to seek a positive integer from the user

The idea is to keep repeating the demand, till we get a valid input

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

We use the `do-while` loop to seek a positive integer from the user

The idea is to keep repeating the demand, till we get a valid input

The moment we get a valid input, i.e. the condition `num < 0` becomes false (or, `num >= 0` becomes true), the control comes out of the loop

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

We use the `do-while` loop to seek a positive integer from the user

The idea is to keep repeating the demand, till we get a valid input

The moment we get a valid input, i.e. the condition `num < 0` becomes false (or, `num >= 0` becomes true), the control comes out of the loop

Since we want the body of the loop to be executed at least once, a `do-while` loop is a good choice

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

For the second loop, we do not wish the body of the loop to be executed even once, if the user entered `0` as the value for `num`

Examples of both loops !!

```c
#include<stdio.h>

int main()
{
        int num;
        long result = 1;

        do
        {
                printf("Give me a small positive integer: ");
                scanf("%d", &num);
        }
        while(num < 0);

        while(num > 1)
        {
                result *= num;
                num -= 1;
        }

        printf("Calculated Factorial: %ld\n", result);

        return 0;
}
```

For the second loop, we do not wish the body of the loop to be executed even once, if the user entered `0` as the value for `num`

So, `while` seems to be a better choice here

An important observation

It may seem that a `do-while` loop, runs one more time, when compared to a `while` loop

- … when everything else is the same

An important observation

It may seem that a `do-while` loop, runs one more time, when compared to a `while` loop

- … when everything else is the same

But that is not true !!

An important observation

It may seem that a `do-while` loop, runs one more time, when compared to a `while` loop

- … when everything else is the same

But that is not true !!

If you are writing a loop that is supposed to run one or more times, the two loops are equivalent

An important observation

It may seem that a `do-while` loop, runs one more time, when compared to a `while` loop

- … when everything else is the same

But that is not true !!

If you are writing a loop that is supposed to run one or more times, the two loops are equivalent

The difference is that a `do-while` loop is guaranteed to execute at least once

Homework !!

Read more about `while` and `do-while` loops, and the differences between them

- These two links are interesting reads:
  https://www.geeksforgeeks.org/difference-between-while-and-do-while-loop-in-c-c-java/
  https://stackoverflow.com/questions/224059/test-loops-at-the-top-or-bottom-while-vs-do-while

Come up with at least one instance each (other than what we discussed), where

- … it is natural to use a `while` loop, as against a `do-while` loop
- … it is natural to use a `do-while` loop, as against a `while` loop