

## Documentation of Day\_3

### Exercise 1-5:

This program uses the formula  $C = (5/9) * (F - 32)$  to print a table of Fahrenheit temperatures and their centigrade or Celsius equivalents:

The source code of the following program is as follows:

```
1  #include <stdio.h>
2  |
3  |
4  | /* ***** */
5  | /* FUNCTION NAME : main */
6  | /* INPUTS      : 1. argc -- the number of parameters provided to the main function */
7  | /*              2. argv -- the pointer to the input string array of parameters */
8  | /* */
9  | /* RETURN      : = 0    -- Success */
10 | /*              : < 0    -- Failed */
11 | /* */
12 | /* NOTES       : print Fahrenheit-Celsius table for fahr = 300, 280, ..., 0 using while loop */
13 | /* ***** */
14 | int main(int argc, char *argv[]) {
15 |     float fahr, celsius;
16 |     float lower, upper, step;
17 |
18 |     lower = 0; /* Lower limit of temperature scale */
19 |     upper = 300; /* upper limit */
20 |     step = 20; /* step size */
21 |
22 |     fahr = upper;
23 |     printf("Fahrenheit\tCelsius\n");
24 |     while (fahr >= lower) {
25 |         celsius = (5.0/9.0) * (fahr-32.0);
26 |         printf("%3.0f\t\t%6.1f\n", fahr, celsius);
27 |         fahr = fahr - step;
28 |     }
29 |     return 0;
30 | }
```

### Steps:

- i) At first take two variables fahr and Celsius to store the value of fahrenheit and Celsius.
- ii) After that declare three variables to set the lower, upper limit of the temperature scale and set the increment step.
- iii) Then assign the upper value in fahr variable as we want to print the Fahrenheit-Celsius table in reverse order (from upper scale to lower scale).
- iv) Then print the header of the table.
- v) Then run a while loop and the loop will run until the fahr is greater than or equal to the lower scale of the temperature.
- vi) Then inside the loop the conversion formula from Fahrenheit to Celsius is written and the table is printed.
- vii) At last just decrement the fahr with the step value.
- viii) The while loop will run and everything inside the loop body will be executed until the value of fahr become lesser than the temperature scale lower limit (set in the while loop condition) and thus all the value of Celsius corresponding to the Fahrenheit value will be printed as form of a table. The output is shown in next snapshot.

## Output:

```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-5_f-c_while.exe
Fahrenheit      Celsius
300             148.9
280             137.8
260             126.7
240             115.6
220             104.4
200             93.3
180             82.2
160             71.1
140             60.0
120             48.9
100             37.8
80              26.7
60              15.6
40              4.4
20              -6.7
0               -17.8

-----
Process exited after 0.03478 seconds with return value 0
Press any key to continue . . .
```

This program uses the formula  $F = (9/5) * (C + 32)$  to print the corresponding Celsius to Fahrenheit table.:

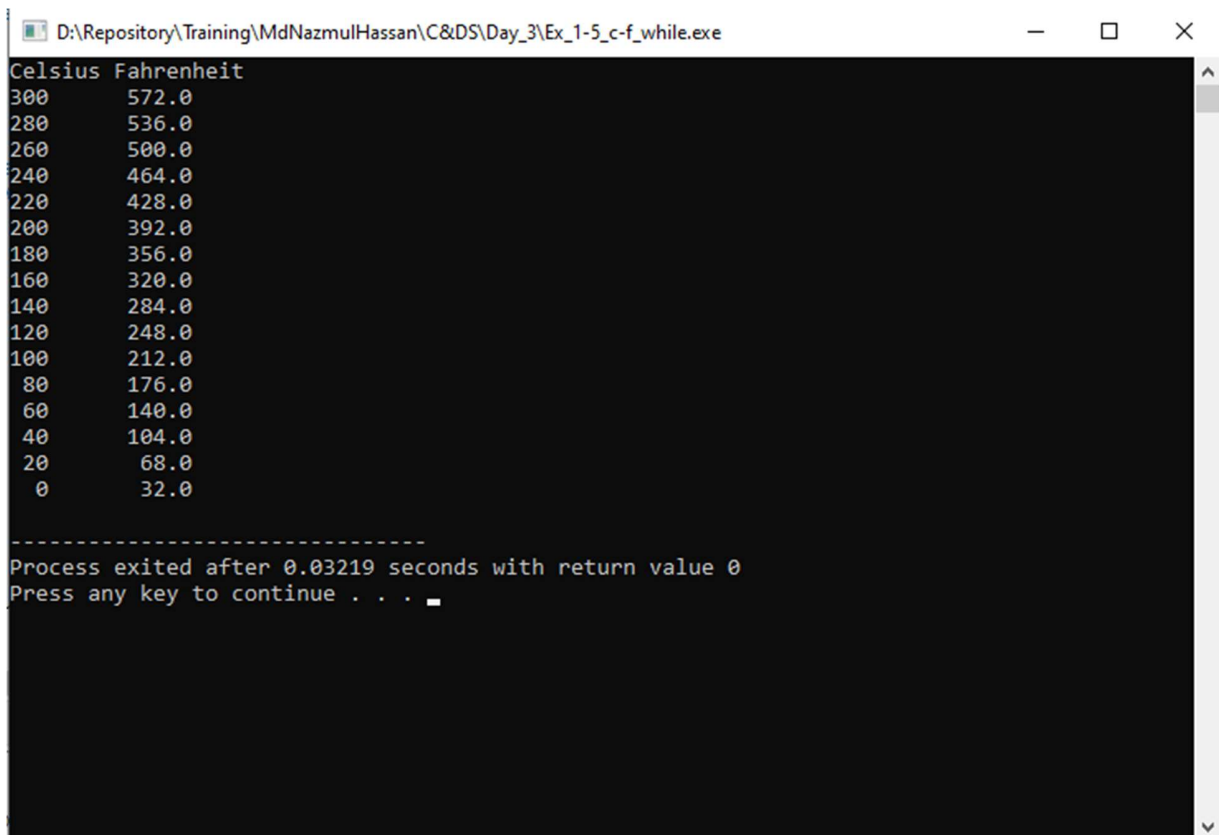
The Source Code is as follows:

```
1  #include <stdio.h>
2
3  /*****
4  /* FUNCTION NAME : main
5  /*
6  /* INPUTS      : 1. argc -- the number of parameters provided to the main function
7  /*              2. argv -- the pointer to the input string array of parameters
8  /*
9  /* RETURN      : = 0    -- Success
10 /*              : < 0    -- Failed
11 /*
12 /* NOTES       : print Celsius-Fahrenheit table for celsius = 300, 280, ... 0 using while loop
13 *****/
14 int main(int argc, char *argv[]) {
15     float celsius, fahr;
16     float lower, upper, step;
17
18     lower = 0;
19     upper = 300;
20     step = 20;
21
22     celsius = upper;
23     printf("Celsius\tFahrenheit\n");
24     while (celsius >= lower) {
25         fahr = (9.0 / 5.0) * celsius + 32.0f;
26         printf("%3.0f\t%6.1f\n", celsius, fahr);
27         celsius = celsius - step;
28     }
29     return 0;
30 }
```

### Steps:

- i) At first take two variables fahr and Celsius to store the value of fahrenheit and Celsius.
- ii) After that declare three variables to set the lower, upper limit of the temperature scale and set the increment step.
- iii) Then assign the upper value in celsius variable as we want to print the Fahrenheit-Celsius table in reverse order (from upper scale to lower scale).
- iv) Then print the header of the table.
- v) Then run a while loop and the loop will run until the celsius is greater than or equal to the lower scale of the temperature.
- vi) Then inside the loop the conversion formula from Celsius to Fahrenheit is written and the table is printed.
- vii) At last just decrement the celsius with the step value.
- viii) The while loop will run and everything inside the loop body will be executed until the value of celsius become lesser than the temperature scale lower limit (set in the while loop condition) and thus all the value of Celsius corresponding to the Fahrenheit value will be printed as form of a table. The output is shown in next snapshot.

### Output:



```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-5_c-f_while.exe
Celsius Fahrenheit
300      572.0
280      536.0
260      500.0
240      464.0
220      428.0
200      392.0
180      356.0
160      320.0
140      284.0
120      248.0
100      212.0
 80      176.0
 60      140.0
 40      104.0
 20       68.0
  0       32.0

-----
Process exited after 0.03219 seconds with return value 0
Press any key to continue . . .
```

This program uses the same formula  $C = (5/9) * (F - 32)$  to print a table of Fahrenheit temperatures and their centigrade or Celsius equivalents but in a different way. There are plenty of different ways to write a program for a particular task. I have tried a variation on the temperature converter. Let us have a look at the source code of this program first:

```

1  #include <stdio.h>
2
3  /*****
4  /* FUNCTION NAME : main
5  /*
6  /* INPUTS      : 1. argc -- the number of parameters provided to the main function
7  /*              2. argv -- the pointer to the input string array of parameters
8  /*
9  /* RETURN      : = 0    -- Success
10 /*              : < 0    -- Failed
11 /*
12 /* NOTES       : print Fahrenheit-Celsius table for Fahrenheit = 300, 280, ... 0 using for loop
13 *****/
14 int main(int argc, char *argv[]) {
15     int fahr;
16
17     printf("Fahrenheit\tCelsius\n");
18     for(fahr=300; fahr>=0; fahr-=20) {
19         printf("%3d \t\t%6.1f\n", fahr, (5.0/9.0)*(fahr-32));
20     }
21     return 0;
22 }
23

```

### Explanation and steps:

This driver code produces the same output as a table like previous but there are some major changes. One of the changes is the elimination of most of the variables. Here only fahr variable remains and I have made it an integer instead of float. Here I have introduced a for loop instead of a while loop. The lower and upper limits and the step size appear as constants in the for statement. The expression that computes the Celsius temperature now appears as the third argument of printf instead of a separate assignment statement.

Since the third argument of printf must be a floating-point value to match the %6.1f, any floating-point expression can occur here.

The for statement is a loop, a generalization of the while. If we compare it to the earlier while, we can understand that both does the same task. Within the parentheses, there are three parts, separated by semicolons. The first part, the initialization fahr = 300 is done once, before the loop proper is entered. The second part is the test or condition that controls the loop:

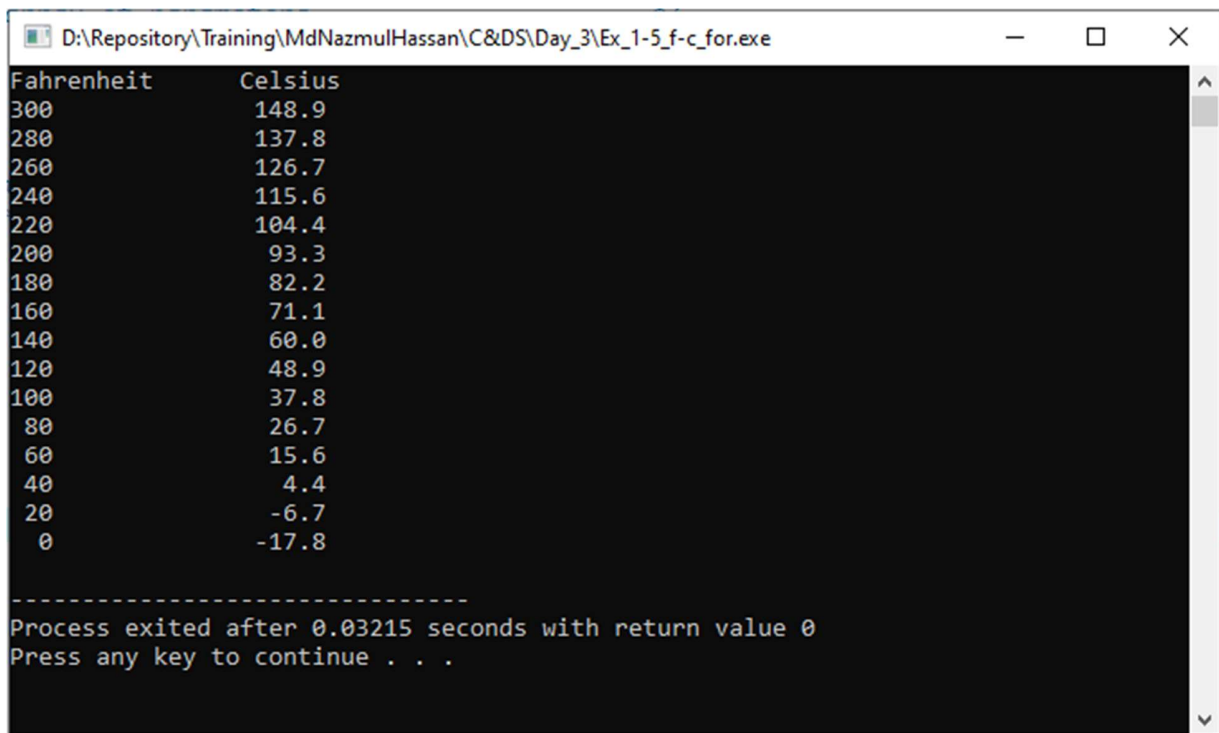
fahr >= 0

This condition is evaluated; if it is true, the body of the loop (here a single printf) is executed. Then the increment step

fahr -= 20 or fahr = fahr - 20 ,both are same (here we have to decrement the fahr value as we want to print the table from upper scale to lower)

is executed, and the condition re-evaluated. The loop will run until the condition is false which is the value of fahr has to be lower than the lower scale value of temperature. Every time the loop runs all the statements of the loop body will be executed. Here there is only a printf statement inside the for loop. Through this printf statement the table of Fahrenheit and its equivalent Celsius value will be printed until the for loop is terminated with its termination condition. The output of this program is given in the next snapshot.

### **Output:**



```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-5_f-c_for.exe
Fahrenheit    Celsius
300           148.9
280           137.8
260           126.7
240           115.6
220           104.4
200           93.3
180           82.2
160           71.1
140           60.0
120           48.9
100           37.8
80            26.7
60            15.6
40            4.4
20            -6.7
0             -17.8

-----
Process exited after 0.03215 seconds with return value 0
Press any key to continue . . .
```

This program uses the same formula  $F = (9/5) * (C + 32)$  to print a table of Celsius temperatures and their Fahrenheit equivalents but in a different way. There are plenty of different ways to write a program for a particular task. I have tried a variation on the temperature converter. Let us have a look at the source code of this program first:

```

1  #include <stdio.h>
2
3  /*****
4  /* FUNCTION NAME : main
5  /*
6  /* INPUTS      : 1. argc -- the number of parameters provided to the main function
7  /*              2. argv -- the pointer to the input string array of parameters
8  /*
9  /* RETURN      : = 0    -- Success
10 /*              : < 0    -- Failed
11 /*
12 /* NOTES       : print Celsius-Fahrenheit table for celsius = 300, 280, ... 0 using for loop
13 *****/
14 int main(int argc, char *argv[]) {
15     int celsius;
16
17     printf("Celsius\tFahrenheit\n");
18     for(celsius=300; celsius>=0; celsius-=20) {
19         printf("%3d \t%6.1f\n", celsius, (9.0/5.0)*celsius+32);
20     }
21     return 0;
22 }
23

```

### Explanation and steps:

This driver code produces the same output as a table like previous but there are some major changes. One of the changes is the elimination of most of the variables. Here only celsius variable remains and I have made it an integer instead of float. Here I have introduced a for loop instead of a while loop. The lower and upper limits and the step size appear as constants in the for statement. The expression that computes the fahrenheit temperature now appears as the third argument of printf instead of a separate assignment statement.

Since the third argument of printf must be a floating-point value to match the %6.1f, any floating-point expression can occur here.

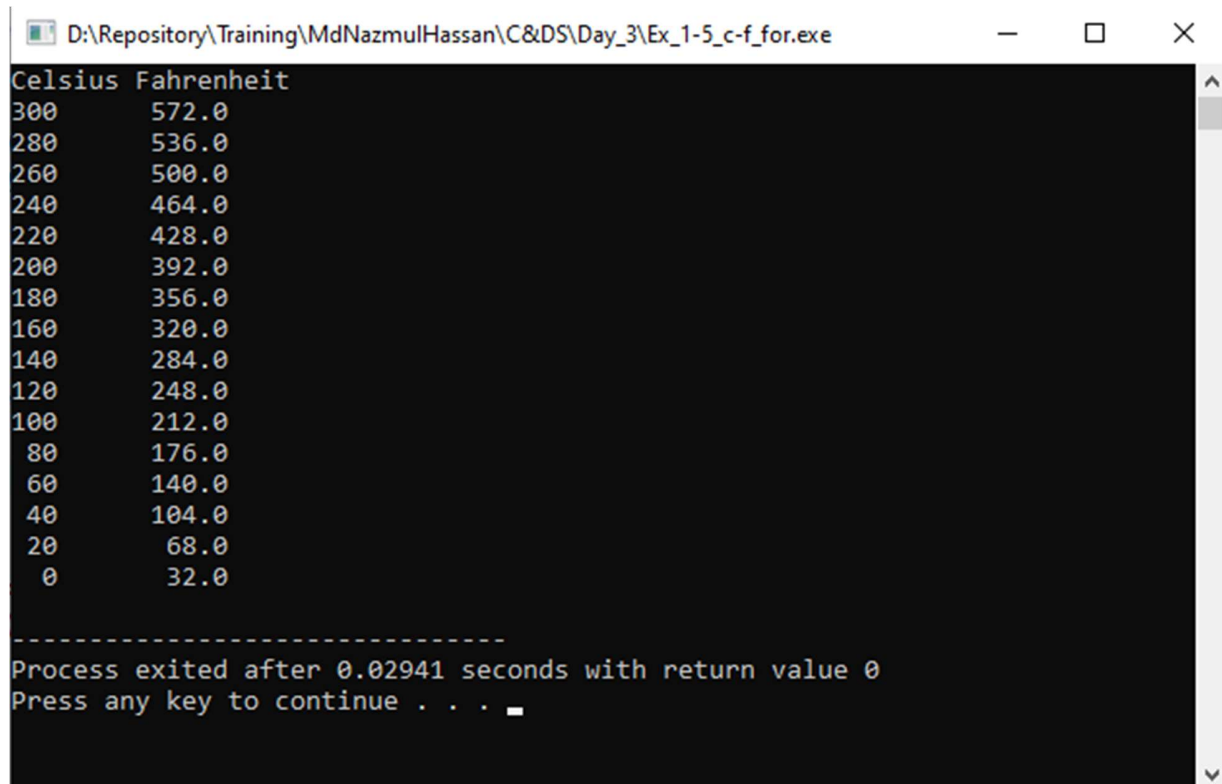
The for statement is a loop, a generalization of the while. If we compare it to the earlier while loop, we can understand that both does the same task. Within the parentheses, there are three parts, separated by semicolons. The first part, the initialization celsius = 300 is done once, before the loop proper is entered. The second part is the test or condition that controls the loop:

celsius >= 0

This condition is evaluated; if it is true, the body of the loop (here a single printf) is executed. Then the increment step

celsius -= 20 or celsius = celsius - 20 ,both are same (here we have to decrement the celsius value as we want to print the table from upper scale to lower)

is executed, and the condition re-evaluated. The loop will run until the condition is false which is the value of celsius has to be lower than the lower scale value of temperature. Every time the loop runs all the statements of the loop body will be executed. Here there is only a printf statement inside the for loop. Through this printf statement the table of Fahrenheit and its equivalent Celsius value will be printed until the for loop is terminated with its termination condition. The output of this program is given in the next snapshot.



```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-5_c-f_for.exe
Celsius Fahrenheit
300      572.0
280      536.0
260      500.0
240      464.0
220      428.0
200      392.0
180      356.0
160      320.0
140      284.0
120      248.0
100      212.0
 80      176.0
 60      140.0
 40      104.0
 20       68.0
 0        32.0

-----
Process exited after 0.02941 seconds with return value 0
Press any key to continue . . . _
```



### Exercise 1-11:

A word count program usually takes the input from user and counts the number of words, characters and newline and produces the outputs in the output console. Let us have a look of the word counting program as follows:

```
1  #include <stdio.h>
2  #define IN 1 /* inside a word */
3  #define OUT 0 /* outside a word */
4
5  /******
6  /* FUNCTION NAME : main
7  /*
8  /* INPUTS      : 1. argc -- the number of parameters provided to the main function
9  /*              : 2. argv -- the pointer to the input string array of parameters
10 /*
11 /* RETURN      : = 0    -- Success
12 /*              : < 0   -- Failed
13 /*
14 /* NOTES       : testing a word count program that counts lines, words, and characters in input
15 /******
16 int main(int argc, char *argv[]) {
17     int c, nl, nw, nc, state;
18     state = OUT;
19     nl = nw = nc = 0;
20
21     while ((c = getchar()) != EOF) {
22         ++nc;
23         if (c == '\n')
24             ++nl;
25         if (c == ' ' || c == '\n' || c == '\t')
26             state = OUT;
27         else if (state == OUT) {
28             state = IN;
29             ++nw;
30         }
31     }
32     printf("New-Line:%d New-Word:%d New-Charcter:%d\n", nl, nw, nc);
33     return 0;
34 }
```

Basically testing of a program is to check whether our program gives the expected output correctly or not. Again testing is needed to find out any bugs or any errors in our program. Actually in C program when we write any code we have to check that if our program is giving the correct output as expected and it can be done through checking the output from the output console.

The above program will take user input as character and store the inputs in a variable 'c' until the End of File (EOF) input.

To test the above word count program and uncover potential bugs we should consider the following types of input:

1. **Basic input:** We can input a few words, lines and characters to check if the program correctly counts them. Examples are:  
Input: Shanghai BDCOM Infotech. Co. Ltd  
The expected output should be:  
New-Line:1 New-Word:5 New-Charcter:33



Here is the actual output after running the program with the above test case:

```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-11.exe
Shanghai BDCOM Infotech. Co. Ltd
^Z
New-Line:1 New-Word:5 New-Charcter:33
-----
Process exited after 60.99 seconds with return value 0
Press any key to continue . . .
```

Here we can see that the output of the following program is correct as expected.

- 2. Empty input:** We can test the program with no input. If the program handles the empty input correctly the expected output should be:

New-Line:0 New-Word:0 New-Charcter:0

Here is the actual output after running the program with the above test case:

```
^Z
New-Line:0 New-Word:0 New-Charcter:0
-----
Process exited after 5.94 seconds with return value 0
Press any key to continue . . .
```

The output of the following program is correct as expected.

- 3. Input with multiple spaces and tabs:** We can include multiple spaces and tabs between words to verify that the program correctly handles such cases. For example:

Hello      world

Output should be:

New-Line:1 New-Word:2 New-Charcter:14

The actual output of the program is:

```
hello  world
^Z
New-Line:1 New-Word:2 New-Charcter:14

-----
Process exited after 13.92 seconds with return value 0
Press any key to continue . . .
```

So we can see that the program takes the spaces and tabs as character input that means it is considering the tabs and spaces.

- 4. Input with new lines:** We can input with multiple new lines to ensure that the program correctly counts them.

For example:

Hello

World

!

Output of the input should be:

New-Line:4 New-Word:3 New-Charcter:15

The actual output is:

```
Hello
World

!
^Z
New-Line:4 New-Word:3 New-Charcter:15

-----
Process exited after 14.66 seconds with return value 0
Press any key to continue . . .
```

So the above program is giving correct output in terms of multiple new lines.

- 5. Input with leading and trailing empty spaces:** We can test with such inputs that contains multiple leading and trailing spaces to check whether our program ignores them or not. Because they are unnecessary spaces or white spaces and they should be ignored.

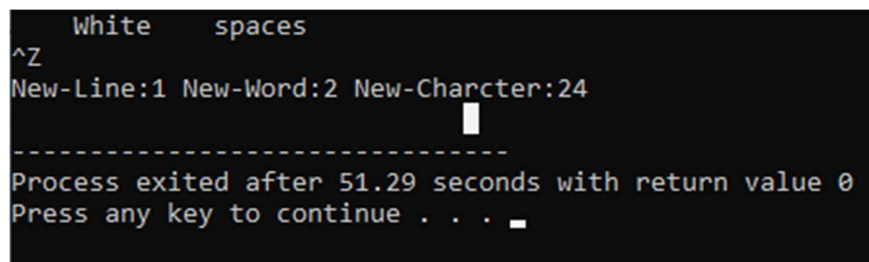
Input:

White spaces

The expected output should be

New-Line:1 New-Word:2 New-Charcter:15

The actual output of the above program is:



```
White spaces
^Z
New-Line:1 New-Word:2 New-Charcter:24
-----
Process exited after 51.29 seconds with return value 0
Press any key to continue . . .
```

Here we can see that the program is treating the leading and trailing spaces as characters. So we can say that it is a bug of this program. It can be resolved by ignoring the leading and trailing spaces.

- 6. Large Input:** We can test the program with a large input file, including a mix of words, lines and characters. This will help verify the program's performance and ensure it handles larger inputs correctly.
- 7. Special Characters with words:** We can include special characters, punctuation marks and numbers in the input to ensure they are correctly counted as characters but not as separate words.

Examples:

Hi! How are #? 123\*

Output should be:

New-Line:1 New-Word:4 New-Charcter:19

- 8. Only special characters:** We can test the word count program with only special characters to figure out whether the program counts them correctly or not.

Examples:

Input:

# %% \* @

Output:

New-Line:0 New-Word:4 New-Charcter:8

9. The program can be tested with the input containing consecutive spaces between words. This test case will ensure that multiple spaces are treated as a single delimiter. For example the input and output to check the test case can be:

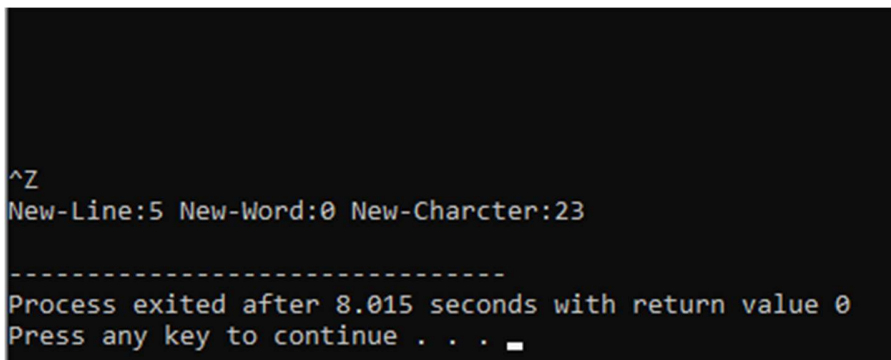
Input: Research and development

The expected output should be:

New-Line:0 New-Word:3 New-Charcter:24

10. **Input with no spaces between words:** The program can be tested with words where words are not separated with any spaces between them. This will help ensure that the program correctly detects the word boundaries.
11. **Input with only whitespace characters:** Input with only whitespace and characters (spaces, tabs, new lines) without any actual words. This will help verify the program correctly handles such cases. For example:

The input and the output can be like as the following snapshot:



```
^Z
New-Line:5 New-Word:0 New-Charcter:23
-----
Process exited after 8.015 seconds with return value 0
Press any key to continue . . .
```

By testing the program with various types of input, including the corner cases and different combinations of characters, we can increase the possibility of uncovering any potential bugs or issues in the word count program.

### Exercise 1-12:

In this exercise I have written a program that print its input one word per-line.

The driver code for this program is as follows:

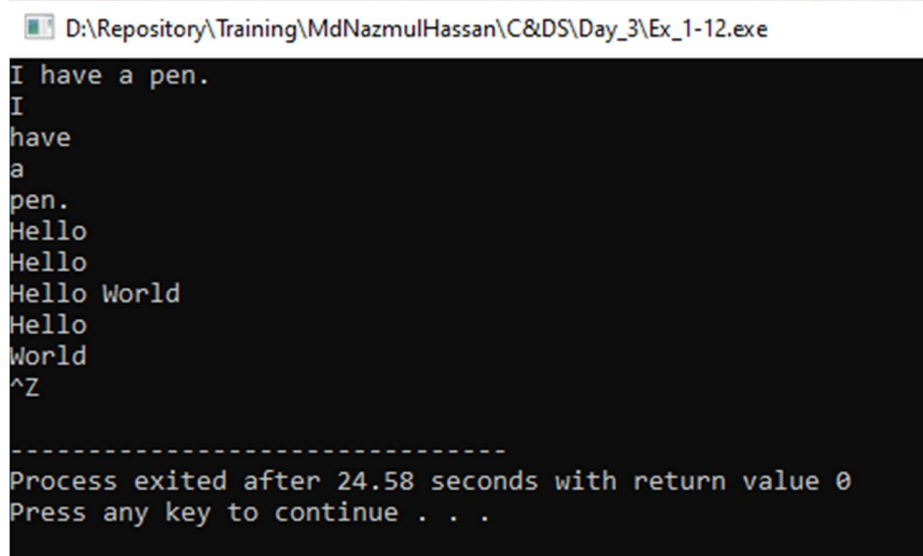
```
1  #include <stdio.h>
2  #define OUT 0
3  #define IN 1
4
5  /******
6  /* FUNCTION NAME : main
7  /*
8  /* INPUTS      : 1. argc -- the number of parameters provided to the main function
9  /*              : 2. argv -- the pointer to the input string array of parameters
10 /*
11 /* RETURN      : = 0    -- Success
12 /*              : < 0   -- Failed
13 /*
14 /* NOTES       : program to print its input one word per line
15 /******
16 int main(int argc, char *argv[]) {
17     int c;
18     int state;
19
20     state = OUT;
21     while ((c = getchar()) != EOF) {
22         if (c != ' ' && c != '\t' && c != '\n') {
23             putchar(c);
24             state = IN;
25         }
26         else if (state) {
27             putchar('\n');
28             state = OUT;
29         }
30     }
31     return 0;
32 }
```

### Steps:

1. First of all I have defined two macros which is OUT and IN and set the values of them as 0 and 1 respectively. This macros are used to determine the state of the input stream whether the input pointer is in the word or outside the word.
2. After that I have declared two variables. One is c and the other is state. Both of the variables are of integer type. In the variable c the inputted character will be stored. And state variable indicates whether the program is inside or outside the word.
3. Then we initialized the state to OUT because OUT determines that the program is outside the word and at the beginning of the program there is no character or word.
4. After that I have introduced a while loop and inside the loop I have initialized c with getchar(), which takes the user input as character and store it in c and checked a condition to check if the program find any End of File input from user. This is a termination condition for the loop. When the condition becomes false the while loop terminates and the loop run till the condition is true.
5. Inside the loop body I have checked the conditions to ignore the spaces, tabs and newline to separate each new word input.

6. In the if block the user input will be printed through the putchar(c) statement. As the program is inside the word the state is set to IN after the putchar statement.
7. In the else if block I have just printed a new line after printing every word.
8. At the end of the input when the program finds any End of File user input the while loop is terminated as the terminating condition of the while loop becomes false. The output of the program is given in the next snapshot.

Output:



```
D:\Repository\Training\MdNazmulHassan\C&DS\Day_3\Ex_1-12.exe
I have a pen.
I
have
a
pen.
Hello
Hello
Hello World
Hello
World
^Z
-----
Process exited after 24.58 seconds with return value 0
Press any key to continue . . .
```