

**CC-112L**

**Programming Fundamentals**

**Laboratory 03**

**Introduction to C & Structured Program Development – II**

**Version: 1.0.0**

**Release Date: 01-08-2022**

**Department of Information Technology**

**University of the Punjab**

**Lahore, Pakistan**

**Contents:**

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
  - Iteration Statement
  - **while** Statement
  - Counter Controlled Iteration
  - Sentinel Controlled Iteration
  - Nested Controlled Statements
  - **for** Statement
  - **switch** Statement
  - **do...while** Iteration Statement
- Activities
  - Pre-Lab Activity
    - **while** Iteration Statement
    - The pseudocode statements
    - Simple Example Program **while** loop
    - Exercise 1
    - Iteration Essentials
    - Exercise 2
    - Counter-controlled iteration
    - Exercise 3
    - Example Program of Counter-controlled iteration
    - Sentinel-controlled iteration
    - Always Use Braces in a **while** Statement
    - Task 01: Sum of Divisors in a range
    - Task 02: Sum of Digits of number
    - Task 03: Dry Run Program
  - In-Lab Activity
    - Nested Controlled Statement
    - Explanation of Nested Controlled Program
    - **for** Iteration Statement
    - **for** Statement Header Components
    - Control variables defined in a **for** header
    - General format of a **for** statement
    - Expressions in the **for-statement's** header are optional
    - **for** Statement Flowchart
    - Exercise 4
    - Exercise 5
    - **do...while** Iteration Statement
    - **do...while** Statement Flowchart
    - Task 01: Printing Stars I
    - Task 02: Printing Stars II
    - Task 03: Printing Stars III
    - Task 04: Floyd's Triangle
    - Task 05: Reverse a number
  - Post-Lab Activity
    - **switch** Statement
    - Rules for **switch** statement in C language

- Functioning of **switch** case statement
- **switch** Multiple-Selection Statement
- Task 01: Multiple Choice Calculator
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

**Learning Objectives:**

- Using **while** Iteration Statement
- Understanding Iteration Essentials
- Understanding Countered Controlled Iteration
- Understanding Sentinel-Controlled Iteration
- Understanding Nested Control Statements
- Using **for** Iteration Statement
- Using **do...while** Iteration Statement
- Using **switch** Statement

**Resources Required:**

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

**General Instructions:**

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

| <b>Teachers:</b>   |                                 |  |
|--------------------|---------------------------------|--|
| Course Instructor  | Prof. Dr. Syed Waqar ul Qounain | <a href="mailto:swjaffry@pucit.edu.pk">swjaffry@pucit.edu.pk</a>     |
| Teacher Assistants | Usman Ali                       | <a href="mailto:bitf19m007@pucit.edu.pk">bitf19m007@pucit.edu.pk</a> |
|                    | Saad Rahman                     | <a href="mailto:bsef19m021@pucit.edu.pk">bsef19m021@pucit.edu.pk</a> |

## Background and Overview:

### Iteration Statement:

A statement which repeats a set of instructions until a condition is satisfied. A single execution of set of instructions is called an iteration.

### while Statement:

A while Statement repeats a set of instructions until a specified expression becomes false.

### Counter Controlled Iteration:

In Counter-Controlled Iteration execution of a code section is repeated a fixed number of predefined times.

### Sentinel Controlled Iteration:

In Sentinel-Controlled Iteration execution of a code section is repeated indefinite times because it is not known in advanced that how many time loop will be executed.

### Nested Controlled Statements:

To use **if** or **if...else** statement inside another **if** or **if...else** statement is referred as Nested Control Statement.

### for Statement:

**for** Statement allows to repeat a set of instructions up to a specified number of times.

### switch Statement:

The **switch** Statement allows you to execute a single block of code using alternative conditions. This can be done using the **if...else** Statement as well but **switch** Statement is easy to read and has a simple syntax.

### do...while Iteration Statement:

**do...while** Statement executes at least once because first iteration runs without checking the condition. The condition is checked after the first iteration.

## Activities:

### Pre-Lab Activities:

#### While Iteration Statement:

An iteration statement (repetition statement / loop) repeats an action while some condition remains true.

#### Syntax:

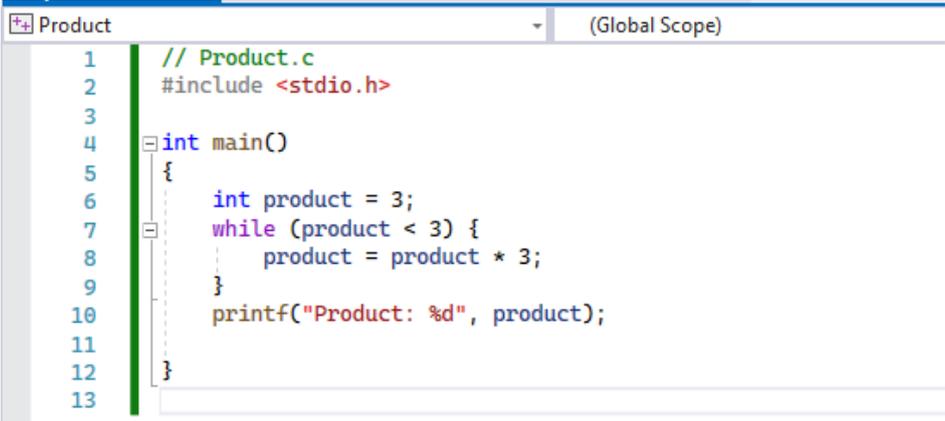
The syntax of a while loop in C language:

```
while (condition) {  
    statement(s);  
}
```

#### Simple Example Program:

As a while statement example, consider a program segment that finds the first power of 3 which is larger than 100.

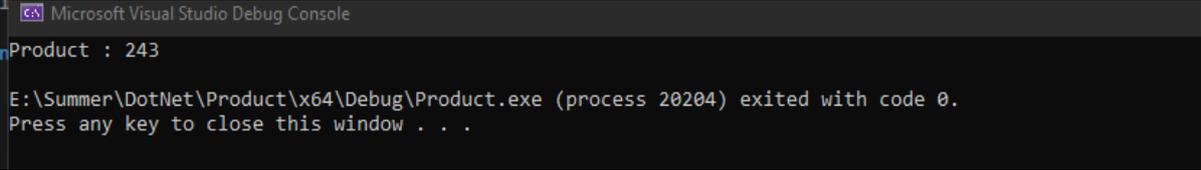
The integer variable product is initialized to 3. When the following code segment finishes executing, product will contain the desired answer:



```
Product (Global Scope)  
1 // Product.c  
2 #include <stdio.h>  
3  
4 int main()  
5 {  
6     int product = 3;  
7     while (product < 3) {  
8         product = product * 3;  
9     }  
10    printf("Product: %d", product);  
11  
12 }  
13
```

Fig 1. (While Iteration)

#### Output:



```
Microsoft Visual Studio Debug Console  
Product : 243  
E:\Summer\DotNet\Product\x64\Debug\Product.exe (process 20204) exited with code 0.  
Press any key to close this window . . .
```

Fig 2. (While Iteration Output)

#### Explanation:

This loop repeatedly multiplies product by 3, so it takes on the values 9, 27 and 81 successively. When product becomes 243, the condition  $product \leq 100$  becomes false, it will terminate the iterations with product's final value is 243.

**Exercise 1:**

```
1  #include <stdio.h>
2
3  int main() {
4
5      // The following code should print the values 1 to 10.
6      int n = 1;
7      while (n < 10) {
8          printf("%d ", n++);
9      }
10
11 }
```

Fig 3. (Exercise No. 1)

Rewrite above code, it should print values 10 to 1.

// Write code below this line:

**Iteration Essentials:**

A loop is a group of instructions the computer repeatedly executes while some loop-continuation condition remains true.

- Counter-controlled iteration
- Sentinel-controlled iteration.

**Exercise 2:**

Dry Run the following program and fill up the trace table:

```
1  // Counter.c
2  #include <stdio.h>
3
4  int main(void) {
5      int counter = 0;
6
7      while (counter <= 5) {
8          printf("%d \n", counter);
9          counter++;
10     }
11 }
```

Fig 4. (Exercise No 2)

| Line # | How many times line will execute |
|--------|----------------------------------|
| 7      |                                  |
| 8      |                                  |
| 9      |                                  |

### Counter-controlled iteration:

Counter-controlled iteration requires:

- The name of a control variable
- The initial value of the control variable
- The increment (or decrement) by which the control variable is modified in each iteration
- The loop-continuation condition that tests for the final value of the control variable to determine whether looping should continue.

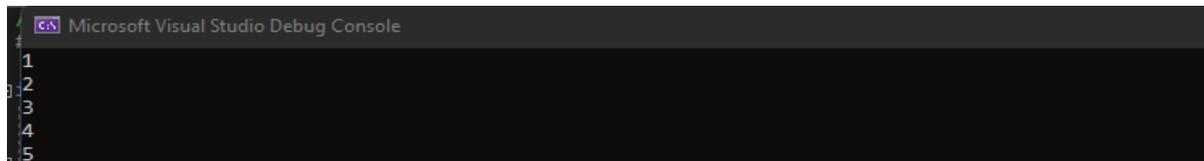
**Example Program:** A program which displays the numbers from 1 through 5.

```

1 // Counter.c
2 #include <stdio.h>
3
4 int main(void) {
5     int counter = 0;
6
7     while (counter <= 5) {
8         printf("%d \n", counter);
9         counter++;
10    }
11 }
```

Fig 5. (Counter-Controlled Iteration)

### Output:



```

1
2
3
4
5
```

Fig 6. (Counter controlled Iteration output)

### Explanation:

The statement `++counter;` increments counter by 1 at the end of each loop iteration. The while's condition `counter <= 5` tests whether the value of the control variable is less than or equal to 5 (the last value for which the condition is true). This while terminates when the control variable exceeds 5 (i.e., counter becomes 6).

### Exercise 3:

```

1 #include <stdio.h>
2
3 int main() {
4     // Find Line having error
5
6     int x = 1;
7     while (x <= 10);
8     ++x;
9 }
```

Fig 7. (Exercise No 3)

| Error Line # | Solution |
|--------------|----------|
|              |          |
|              |          |

### Sentinel-controlled iteration:

In sentinel-controlled iteration use a sentinel value to indicate “end of data entry”. A sentinel value also is called a signal value, a dummy value, or a flag value.

In the Example there will be phases:

- An initialization phase that initializes the program variables.
- A processing phase that inputs data values and process program variable.
- A termination phase that calculates and prints the final results.

```

1 // Grades.c
2 #include <stdio.h>
3
4 int main()
5 {
6     int total = 0;
7     int counter = 0;
8
9     printf("Enter grades -1 to end: "); // prompt for input
10    int grade = 0;
11    scanf("%d", &grade); // read grades from user
12
13    while (grade != -1) {
14        total = total + grade;
15        counter = counter + 1;
16        printf("Enter grades -1 to end: "); // prompt for input
17        scanf("%d", &grade); // read grades from user
18    } // end while
19
20    if (counter != 0) {
21        double average = (double)total / counter;
22        printf("Class average is %.2f\n", average);
23    }
24    else {
25        put("No grades were entered");
26    } // end if-else
27 }

```

Fig 8. (Sentinel-controlled iteration)

### Output:

```

Microsoft Visual Studio Debug Console
Enter grade, -1 to end: 75
Enter grade, -1 to end: 65
Enter grade, -1 to end: 50
Enter grade, -1 to end: 90
Enter grade, -1 to end: -1
Class average is 70.00

```

Fig 9. (Sentinel-controlled iteration output)

In this C program Although only integer grades are entered, the averaging calculation is likely to produce a number with a decimal point. The type **int** cannot represent such a number.

So, this program introduces the data type double to handle numbers with decimal point that is, floating-point numbers.

Here, a cast operator is used to force the averaging calculation to use floating-point numbers. These features are explained after the program listing. Note that lines 13 and 21 both include the sentinel value in the prompts requesting data entry. This is a good practice in a sentinel-controlled loop.

**Always Use Braces in a while Statement:**

Without this while loop's braces (lines 16 and 26), only the statement on line 17 would be in the loop's body. The code would be incorrectly interpreted as

```
while (grade! = -1)
    total = total + grade; // add grade to total
    counter = counter + 1; // increment counter
    // get next grade from user
    printf ("%s", "Enter grade, -1 to end: "); // prompt for input
    scanf ("%d", &grade); // read next grade
```

This would cause an infinite loop if the user did not input -1 as the first grade.

**Task 01: Sum of Dividends in a range****[Estimated 15 minutes / 10 marks]**

Create a C program that:

- Takes input of two integers.
- Finds the sum of all numbers which are dividends of 4 and 16, between these two integers.
- Print an appropriate message if no such dividend (number) exists in the range.
- Display the Sum of all divisible on the Console as shown in figure.



```

Microsoft Visual Studio Debug Console
Enter first Integer: 15
Enter second Integer: 85
16 + 32 + 48 + 64 + 80 = 240

```

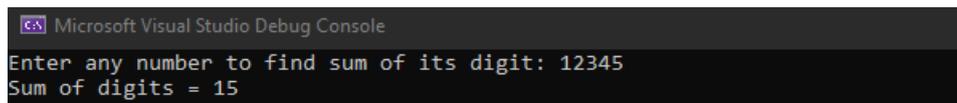
Fig. 10 (Sum of dividends)

- Submit “.c” file named your “Roll No” on Google Classroom

**Task 02: Sum of Digits of number****[Estimated 15 minutes / 10 marks]**

Create a C program which:

- Takes input of a number.
- Calculate sum of its digits using for loop.



```

Microsoft Visual Studio Debug Console
Enter any number to find sum of its digit: 12345
Sum of digits = 15

```

Fig. 11 (Sum of digits of a number)

- Submit “.c” file named your “Roll No” on Google Classroom

**Task 03: Dry Run Program.****[15 minutes / 10 marks]**

```

1 | #include <stdio.h>
2 |
3 | int main(void) {
4 |
5 |     int x = 0;
6 |
7 |     while (x <= 20) {
8 |         if (x % 5 == 0) {
9 |             printf("%d\t", x);
10 |            x = x + 2;
11 |         }
12 |         else {
13 |             printf("%d\t", x);
14 |             x = x + 5;
15 |         }
16 |         ++x;
17 |     }
18 | }

```

Fig 12. (Dry Run Program)

| X = | Output |
|-----|--------|
| 1   |        |
| 3   |        |
| 10  |        |

## In-Lab Activities:

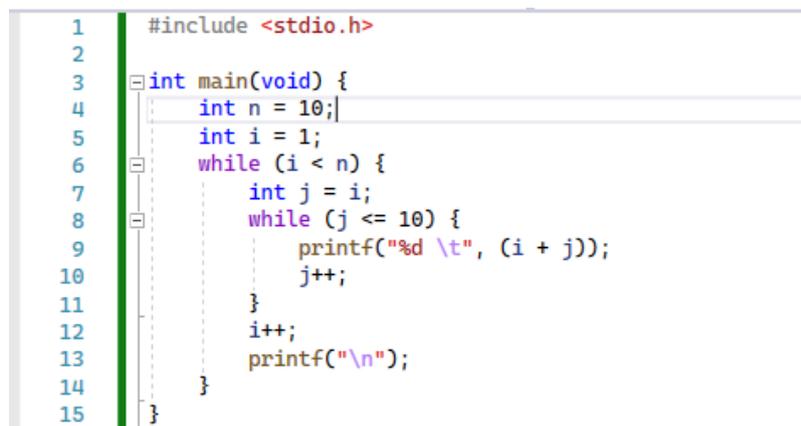
### Nested Controlled Statement:

C supports nesting of loops. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C. Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at **n** times.

### Concept of nested loop:

```
Outer_loop
{
    Inner_loop
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

### Example of nested loop Program:



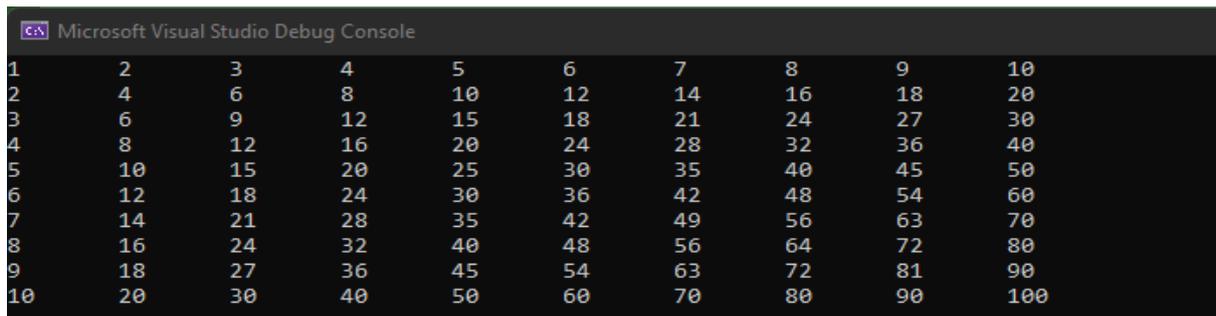
```
1  #include <stdio.h>
2
3  int main(void) {
4      int n = 10;
5      int i = 1;
6      while (i < n) {
7          int j = i;
8          while (j <= 10) {
9              printf("%d \t", (i + j));
10             j++;
11         }
12         i++;
13         printf("\n");
14     }
15 }
```

Fig. 13 (Nested loop Program)

### Explanation of the above code:

- In line 5, first, the 'i' variable is initialized to 1.
- In line 6, the while statement checks whether the condition 'i<=n' is true or not. If the condition is true, then the program control passes to the inner loop.
- Line 7 to line 14, will get executed until the condition inside while at line 6 is true.
- After the execution of the inner statements from line 7 to line 14, the control moves back to the update of the outer loop, i.e., at line 6, where i++ statement is executed
- After incrementing the value of the loop counter, the condition is checked again, i.e., i<=n.
- If the condition is true, then the inner loop will be executed again.
- This process will continue until the condition of the outer loop is true.

### Output:



|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10  |
| 2  | 4  | 6  | 8  | 10 | 12 | 14 | 16 | 18 | 20  |
| 3  | 6  | 9  | 12 | 15 | 18 | 21 | 24 | 27 | 30  |
| 4  | 8  | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40  |
| 5  | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50  |
| 6  | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60  |
| 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70  |
| 8  | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80  |
| 9  | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90  |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

Fig. 14 (Nested Loop Program Output)

**for Iteration Statement:**

Counter.c

```

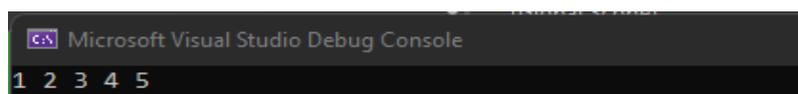
1 // Counter.c
2 // Counter-controlled iteration.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // initialization
7
8     while (counter <= 5) { // iteration condition
9         printf("%d ", counter);
10        ++counter; // increment
11    }
12    put(""); // output a newline
13 }

```

Fig. 15 (For loop Counter)

The a single for-iteration statement handles all the details of counter-controlled iteration of the while loop in Counter.c program from line 6-10. For readability, try to fit the for-statement's header (line 8) on one line. The for statement executes as follows:

- When program begins executing, control variable counter will be initializing it to 1 by for statement.
- Next, it will check loop condition counter <= 5. At start value of counter is 1, so the condition is true, and the for statement executes its printf statement (line 9) to display counter's value, namely 1.
- Next, the for statement increments the variable counter using the expression ++counter and loop will execute so on.
- This loop continues until the control variable counter becomes 6. At this point, the loop condition is false and iteration terminates. The program continues executing with the first statement after the for (line 12).

**Output:**


```

1 2 3 4 5

```

Fig. 16 (For loop Counter Output)

**for Statement Header Components:**

The following diagram takes a closer look at above Program's for statement, which specifies each of the items needed for "for" iteration:

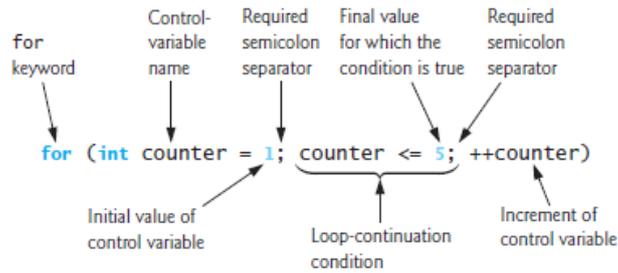


Fig. 17 (for statement Header)

**Control Variables Defined in a for Header Exist Only Until the Loop Terminates:**

When you define the control variable in the **for** header before the first semicolon (;), as in line 8 of Counter.c the control variable exists only until the loop terminates. So, attempting to access the control variable after the for-statement’s closing right brace (}) is a compilation error.

```
for (int counter = 1; counter <= 5; ++counter) {
```

**General Format of a for Statement:**

The general format of the for statement is

```
for (initialization; loopContinuationCondition; increment) {
    statement
}
```

In the for loop:

- Initialization names the loop’s control variable and provides its initial value,
- Loop Condition determines whether the loop should continue executing,
- Increment updates the control variable value after executing the statement so that the loop condition eventually becomes false.

**Expressions in the for-Statement’s Header Are Optional:**

All three expressions in a for header are optional. If you remove the loop condition, the condition is always true, thus creating an infinite loop. You can remove the initialization expression if the program initializes the control variable before the loop. You can also remove the increment expression if the program calculates the increment in the loop’s body or if no increment is needed.

**for Statement Flowchart:**

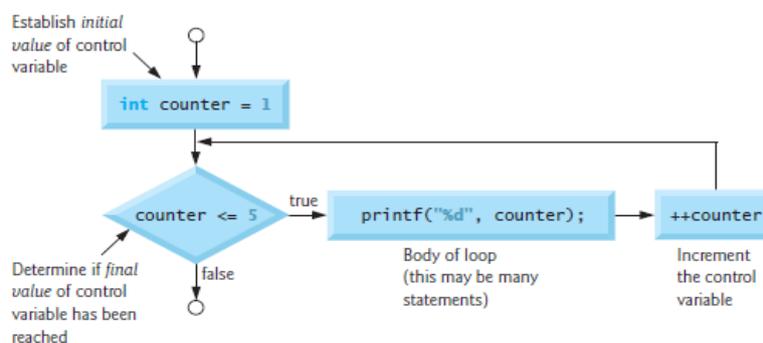


Fig. 18 (for statement flow chart)

**Exercise 4:**

Dry Run the following program and fill up the trace table:

```

// for# 1
for (int x = 2; x <= 13; x += 2) {
    printf("%d, ", x);
}
// for# 2
for (int x = 5; x <= 22; x += 7) {
    printf("%d, ", x);
}
// for# 3
for (int x = 3; x <= 15; x += 3) {
    printf("%d, ", x);
}
// for# 4
for (int x = 1; x <= 5; x += 7) {
    printf("%d, ", x);
}
// for# 5
for (int x = 12; x >= 2; x -= 3) {
    printf("%d, ", x);
}
    
```

Fig 19. (Exercise No 4)

| For loop # | How many times loop run | Output |
|------------|-------------------------|--------|
| 1          |                         |        |
| 2          |                         |        |
| 3          |                         |        |
| 4          |                         |        |
| 5          |                         |        |

**Exercise 5:**

```

1  #include <stdio.h>
2
3  int main() {
4      // Change while loop to for loop
5
6      int x = 0;
7      while (++x <= 20) {
8          if (x % 5 == 0) {
9              printf("%d\n", x);
10         }
11         else {
12             printf("%d\t", x);
13         }
14     }
15 }
    
```

Fig 20. (Exercise No 5)

**Rewrite code:**

```

// Write code below this line:
    
```

**do...while Iteration Statement:**

The **do...while** iteration statement is similar to the while statement. The while statements test its loop condition before executing the loop body. The **do...while** statement tests its loop condition after executing the loop body, so the loop body always executes at least once.

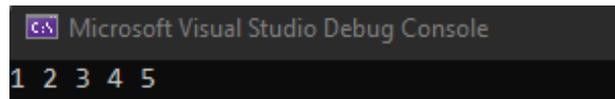
**Dowhile.c**

```

1 // Counter.c
2 // do-while iteration.
3 #include <stdio.h>
4
5 int main(void) {
6     int counter = 1; // initialization
7
8     do {
9         printf("%d ", counter);
10        ++counter; // increment
11    } while (counter <= 5);
12
13    put(""); // output a newline
14 }

```

Fig. 21 (do while Iteration)

**Output:**


```

C:\> Microsoft Visual Studio Debug Console
1 2 3 4 5

```

Fig. 22 (do while Iteration Output)

**do...while Statement Flowchart:**

The following **do...while** statement flowchart makes it clear that the loop-continuation condition does not execute until after the loop's action is performed the first time:

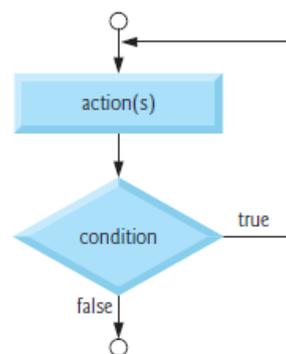


Fig. 23 (do while Iteration Flow Chart)

**Task 01: Printing Stars**

**[20 minutes / 20 marks]**

Write a C program which: [Hint: Use Nested for Statements]

- Take input number of rows
- Output Stars Patterns as shown at the following output screen

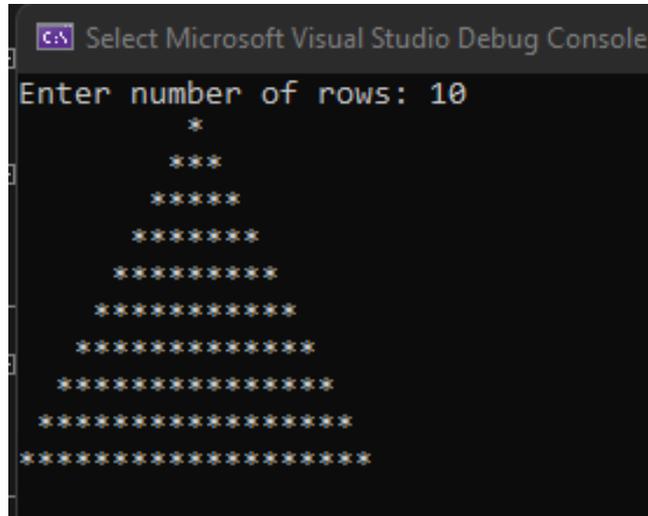


Fig. 24 (In-Lab Task)

**Task 02: Printing Stars**

**[20 minutes / 10 marks]**

Write a C program which: [Hint: Use Nested for Statements]

- Take input number of rows
- Output Stars Patterns as shown at the following output screen

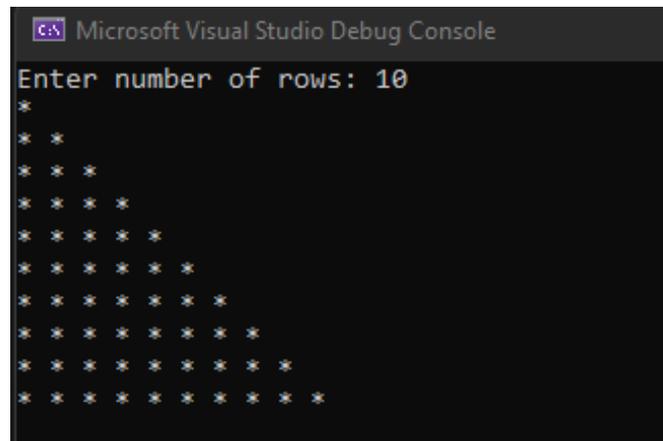


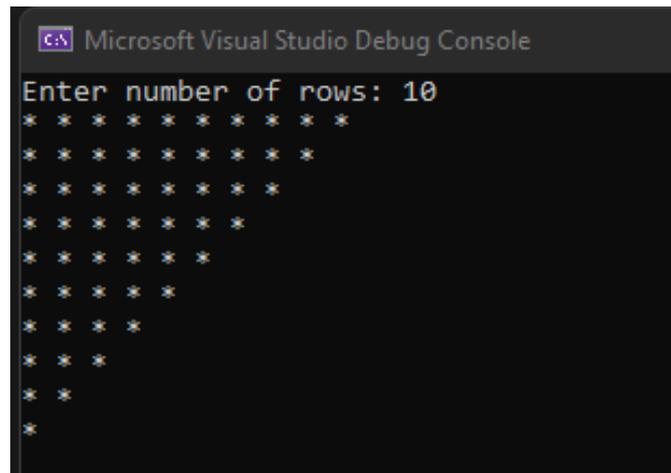
Fig. 25 (In-Lab Task)

**Task 03: Printing Stars**

**[20 minutes / 10 marks]**

Write a C program which: [Hint: Use Nested for Statements]

- Take input a number of rows
- Output Stars Patterns as shown at the following output screen

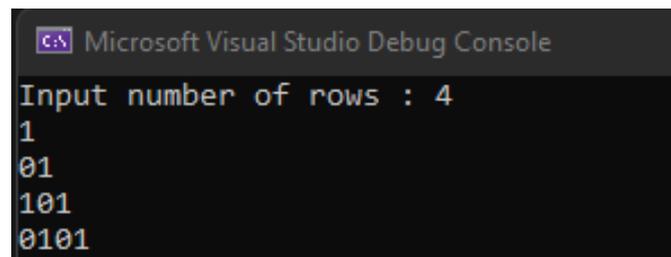


```
C:\> Microsoft Visual Studio Debug Console
Enter number of rows: 10
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

Fig. 26 (In-Lab Task)

**Task 04: Floyd's Triangle****[30 minutes / 20 marks]**

Create a program in C to print the Floyd's Triangle

**Sample Output:**

```
C:\> Microsoft Visual Studio Debug Console
Input number of rows : 4
1
01
101
0101
```

Fig. 27 (In-Lab Task)

**Task 05: Reverse a number****[30 minutes / 10 marks]**

Create a C program that:

- Prompts the user to input an integer
- Outputs the number with the digits reversed.

For example, if the input is 12345, the output should be 54321.

**Post-Lab Activities:****Switch Statement:**

The switch statement in C is an alternate to **if-else-if** statement which allows us to execute multiple operations for the different possible values of a single variable called switch variable. Here, we can define various statements in the multiple cases for the different values of a single variable.

**Switch Statement syntax:**

```
switch (expression) {
  case value1:
    //code to be executed;
    break; //optional
  case value2:
    //code to be executed;
    break; //optional
  .....
  default:
    code to be executed if all cases are not matched;
}
```

Fig. 28 (Switch statement)

**Rules for switch statement in C language:**

- The switch expression must be of an integer or character type.
- The case value must be an integer or character constant.
- The case value can be used only inside the switch statement.
- The break statement in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as fall through the state of C switch statement.

**Functioning of switch case statement:**

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

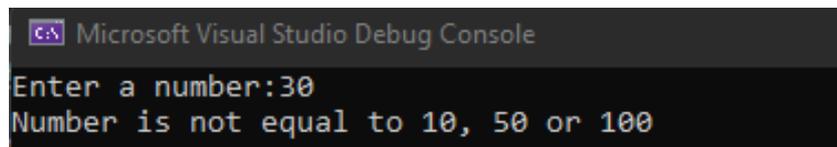
Let's see a simple example of C language switch statement.

**Program:**

```
1  #include <stdio.h>
2
3  int main() {
4      int number = 0;
5      printf("Enter a number: ");
6      scanf("%d", &number);
7
8      switch (number)
9      {
10     case 10:
11         printf("Number is equal to 10\n");
12         break;
13     case 50:
14         printf("Number is equal to 50\n");
15         break;
16     case 100:
17         printf("Number is equal to 100\n");
18         break;
19     default:
20         printf("Number is not equal to 10, 50 and 100\n");
21         break;
22     }
23     return 0;
24 }
25
```

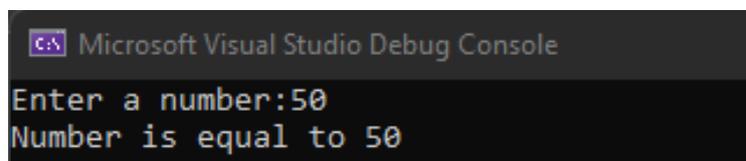
Fig. 29 (Switch Statement Program)

**Output:**



```
C:\> Microsoft Visual Studio Debug Console
Enter a number:30
Number is not equal to 10, 50 or 100
```

Fig. 30 (Switch Statement Program Output)



```
C:\> Microsoft Visual Studio Debug Console
Enter a number:50
Number is equal to 50
```

Fig. 31 (Switch Statement Program Output)

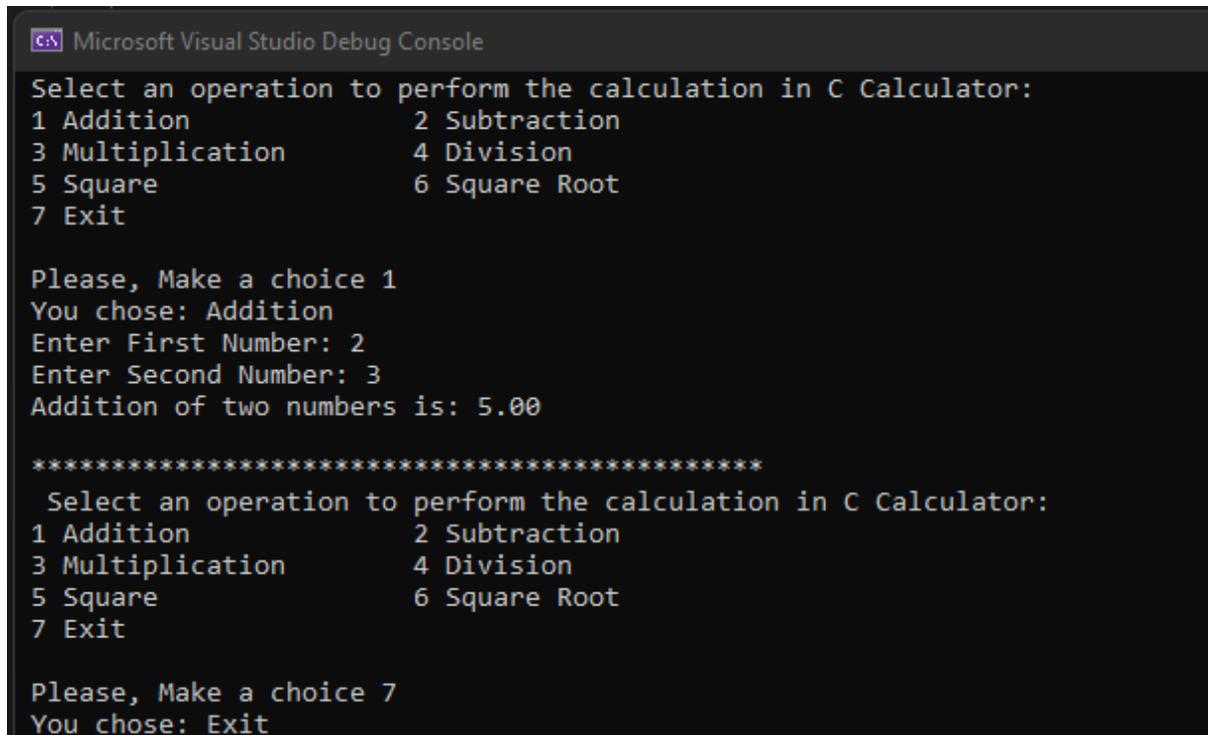
**switch Multiple-Selection Statement:**

if single-selection and the if...else double-selection statements. Occasionally, an algorithm will contain a series of decisions that test a variable or expression separately for each of the integer values it may assume, then perform different actions. This is called multiple selection. C language provides the switch multiple-selection statement to handle such decision making. The switch statement consists of a series of case labels, an optional default case and statements to execute for each case.

**Task 01: Multiple Choice Calculator****[Estimated 45 minutes / 30 marks]**

Create a Console based calculator which:

- Displays the menu on Console and takes an input from user as follows:



```
Microsoft Visual Studio Debug Console

Select an operation to perform the calculation in C Calculator:
1 Addition                2 Subtraction
3 Multiplication          4 Division
5 Square                  6 Square Root
7 Exit

Please, Make a choice 1
You chose: Addition
Enter First Number: 2
Enter Second Number: 3
Addition of two numbers is: 5.00

*****

Select an operation to perform the calculation in C Calculator:
1 Addition                2 Subtraction
3 Multiplication          4 Division
5 Square                  6 Square Root
7 Exit

Please, Make a choice 7
You chose: Exit
```

Fig. 24 (Post-Lab Task)

Submit “.c” file named your “**Calculator**” on Google Classroom.

## Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .c file on Google Classroom and name it to your roll no.

## Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** [35 marks]
  - Sum of Divisors in a range [10 marks]
  - Sum of digit of number [10 marks]
  - Dry Run Program [15 marks]
- **Division of In-Lab marks:** [50 marks]
  - Task 01: Printing Stars I [10 marks]
  - Task 02: Printing Stars II [10 marks]
  - Task 03: Printing Stars III [10 marks]
  - Task 04: Floyd's Triangle [10 marks]
  - Task 05: Reverse Number [10 marks]
- **Division of Post-Lab marks:** [30 marks]
  - Task01: Simple Multiple-Choice Calculator [30 marks]

## References and Additional Material:

- if...else Statement in C  
<https://www.programiz.com/c-programming/c-if-else-statement>
- for-loop Statement in C  
<https://www.programiz.com/c-programming/c-for-loop>
- do-while Statement in C  
<https://www.programiz.com/c-programming/c-do-while-loops>
- switch-case Statement in C  
<https://www.programiz.com/c-programming/c-switch-case-statement>
- break-continue Statement in C  
<https://www.programiz.com/c-programming/c-break-continue-statement>

**Lab Time Activity Simulation Log:**

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:30: Execute C on Visual Studio
- Slot – 03 – 00:30 – 00:45: Execute C on Visual Studio
- Slot – 04 – 00:45 – 01:00: In-Lab Task
- Slot – 05 – 01:00 – 01:15: In-Lab Task
- Slot – 06 – 01:15 – 01:30: In-Lab Task
- Slot – 07 – 01:30 – 01:45: In-Lab Task
- Slot – 08 – 01:45 – 02:00: In-Lab Task
- Slot – 09 – 02:00 – 02:15: In-Lab Task
- Slot – 10 – 02:15 – 02:30: In-Lab Task
- Slot – 11 – 02:30 – 02:45: In-Lab Task
- Slot – 12 – 02:45 – 03:00: Discussion on Post-Lab Task