Unit # 4

DATA AND REPETITION

Students Learning Outcomes

After completing this unit students will be able to

- Understand the structure of array
- Declare and use one dimensional arrays
- Use variable as an index in array
- Read and write values in array
- Explain the concept of loop structure
- Know that for loop structure is composed of:
  - For
  - Initialization expression
  - Test expression
  - Body of the loop
  - Increment/decrement expression
- Explain the concept of a nested loop
- Use loops to read and write data in array

Unit Introduction

While writing computer programs, we may find situations where we need to process large quantities of data. The techniques that we have learnt so far may not seem suitable in these situations. So we need to have better mechanisms for storage and processing of large amounts of data. Another common problem that we face is how to repeat a set of instructions for multiple times without writing them again and again. This chapter discusses the ways that C programming language provides in order to deal with data and repetitions.

4.1 Data Structures

In the previous chapters, we learnt how to store pieces of data in the variables. What if we need to store and process large amount of data e.g. the marks of 100 students? Probably, we need to declare 100 variables, which does not seem an appropriate solution. So, high level programming languages provide ***data structures*** in order to store and organize data. A data structure can be defined as follows:

> Data structure is a container to store collection of data items in a specific layout.

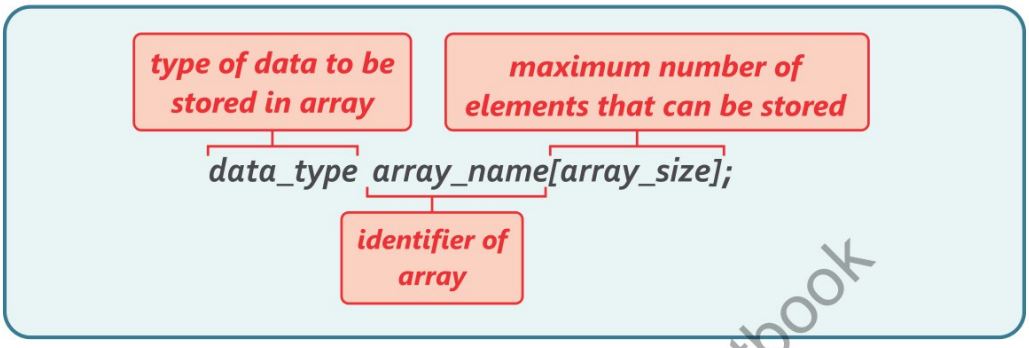
Different data structures are available in C programming language, however this chapter discusses only one of them, which is called ***Array***. It is one of the most commonly used data structures.

4.1.1 Array

An array is a data structure that can hold multiple values of same data type e.g. an *int* array can hold multiple integer values, a *float* array can hold multiple real values and so on. An important property of array is that it stores all the values at consecutive locations inside the computer memory.

4.1.2 Array Declaration

In C language, an array can be declared as follows:



If we want to declare an array of type *int* that holds the daily wages of a laborer for seven days, then we can declare it as follows:

```
int daily_wage[7];
```

Following is the example of the declaration of a *float* type array that holds marks of 20 students.

```
float marks[20];
```

4.1.3 Array Initialization

Assigning values to an array for the first time, is called array initialization. An array can be initialized at the time of its declaration, or later. Array initialization at the time of declaration can be done in the following manner.

data_type array_name[N] = {value1, value2, value3,....., valueN};

Following example demonstrates the declaration and initialization of a float array to store the heights of seven persons.

```
float height[7] = {5.7, 6.2, 5.9, 6.1, 5.0, 5.5, 6.2};
```

Here is another example that initializes an array of characters to store five vowels of English language.

```
char vowels[5] = {'a', 'e', 'i', 'o', 'u'};
```

Important Note:

If we do not initialize an array at the time of declaration, then we need to initialize the array elements one by one. It means that we cannot initialize all the elements of array in a single statement. This is demonstrated by the following example.

</> EXAMPLE CODE 4.1

```
void main()
{
      int array[5];
      array[5] = {10, 20, 30, 40, 50};
}
```

initializing whole array after declaration not allowed — ERROR

The compiler generates an error on the above example code, as we try to initialize the whole array in one separate statement after declaring it.

4.1.4 Accessing array elements

Each element of an array has an *index* that can be used with the array name as *array_name[index]* to access the data stored at that particular *index*.

First element has the index 0, second element has the index 1 and so on. Thus *height[0]* refers to the first element of array *height*, *height[1]* refers to the second element and so on. Figure 4.1 shows graphical representation of array *height* initialized in the last section.

Figure 4.1: Graphical representation of array *height*

PROGRAMMING TIME 4.1

Write a program that stores the ages of five persons in an array, and then displays on screen.

Solution:

```
#include<stdio.h>
void main()
{
	int age[5];
	/* Following statements assign values at different
	indices of array age. We can see that the first value is
	stored at index 0 and the last value is stored at index 4
	*/
	age[0] = 25;
	age[1] = 34;
	age[2] = 29;
	age[3] = 43;
	age[4] = 19;
	/* Following statement displays the ages of five persons
	stored in the array */
	printf("The ages of five persons are: %d, %d, %d, %d,
	%d", age[0], age[1], age[2], age[3], age[4]);
}
```

PROGRAMMING TIME 4.2

Write a program that takes the marks obtained in 4 subjects as input from the user, calculates the total marks and displays on screen.

Solution:

```
#include<stdio.h>
void main()
{
	float marks[4], total_marks;
	printf("Please enter the marks obtained in 4 subjects:
	");
	scanf("%f%f%f%f",&marks[0], &marks[1], &marks[2],
	&marks[3]);
	total_marks = marks[0] + marks[1] + marks[2] + marks[3];
	printf("Total marks obtained by student are %f",
	total_marks);
}
```

4.1.5 Using variables as array indexes

A very important feature of arrays is that we can use variables as array indices e.g. look at the following program:

</> EXAMPLE CODE 4.2

```
#include<stdio.h>
void main()
{
      int array[5] = {10, 20, 30, 40, 50};
      int i;
      /* Following statements ask the user to input an index
      into variable i. */
      printf("Please enter the index whose value you want to
      display");
      scanf("%d", &i);
      /* Following statement displays the value of the array
      at the index entered by user. */
      printf("The value is %d", array[i]);
}
```


Following program demonstrates that when we change the value of a variable, its later usage uses the updated value.

</> EXAMPLE CODE 4.3

```
#include<stdio.h>
void main()
{
      int array[5] = {10, 20, 30, 40, 50};
      int i = 2;
      /* Following statement displays value 30, as i contains
      2 and the value at array[2] is 30 */
      printf("%d", array[i]);
      i++;
      /* Following statement displays value 40, as i has been
      incremented to 3 and the value at array[3] is 40. */
      printf("\n%d", array[i]);
}
```

4.2 Loop Structure

If we need to repeat one or more statements, then we use loops. For example, if we need to write Pakistan thousand times on the screen, then instead of writing `printf("Pakistan");` a thousand times, we use loops. C language provides three kind of loop structures:

1- For loop
2- While loop
3- Do While loop

In this chapter, our focus is on *for loops*.

4.2.1 General structure of loops

If we closely observe the process that humans follow for repeating a task for specific number of times then it becomes easier for us to understand the loop structures that C language provides us for controlling the repetitions.

Let's assume that our sports instructor asks us to take 10 rounds of the running track. How do we perform this task? First we set a counter to zero, because we have not yet taken a single round of the track. Then we start taking the rounds. After each round we increase our counter by 1 and check whether we have completed 10 rounds or not yet. If we have not yet completed the 10 rounds then we again take a round, increase our counter by 1, and again check whether we have taken 10 rounds or not. We repeat this process till our counter reaches 10.

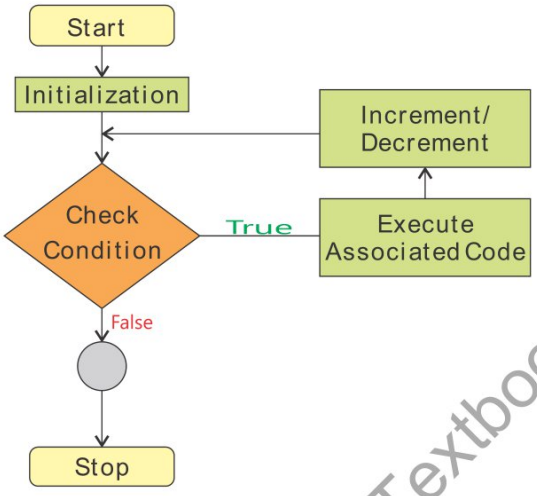
Different programming languages follow similar philosophy in the loop structures for repeating a set of instructions.

4.2.2 General syntax of for loop

In C programming language, for loop has the following general syntax.

```
for(initialization; condition; increment/decrement)
{
        Code to repeat
}
```

In order to understand the *for* loop structure let's look at the following flow chart.



From the flow chart, we can observe the following sequence:

1- *Initialization* is the first part to be executed in a *for* loop. Here we initialize our counter variable and then move to the *condition* part.

2- *Condition* is checked, and if it turns out to be *false*, then we come out of loop.

3- If the *condition* is *true*, then *body of the loop* is executed.

4- After executing the body of loop, the counter variable is increased or decreased depending on the used logic, and then we again move to the step 2.

After executing the *body of loop*, the counter variable is increased or decreased depending on the used logic, and then we again move to the step 2.

</> EXAMPLE CODE 4.4

```
for(int i = 0; i < 3; i++)
{
      printf("Pakistan\n");
}
```

Output:

```
Pakistan
Pakistan
Pakistan
```

Continued

Description:

If we observe the written code and compare it to the flowchart description, we can see the following sequence of execution.

1- *Initialization* expression is executed, i.e. *int i = 0*. Here counter variable *i* is declared and initialized with value *0*.

2- *Condition* is tested, i.e. *i<3*. As variable *i* has value *0* which is less than *3* so *condition* turns out to be *true*, and we move to the *body of the loop*.

3- *Loop body* is executed, i.e. *printf("Pakistan\n");* thus *Pakistan* is displayed on screen.

4- *Increment/decrement* expression is executed, i.e. *i++*. Thus the value of *i* is incremented by 1. As variable *i* had value *0*, so after this statement *i* contains value *1*.

5- Now *condition* is again tested. Because, value of *i* is *1* which is less than *3* so *condition* again turns out to be *true* and *loop body* is executed again i.e. *Pakistan* is again displayed on screen. The value of *i* gets incremented to *2*.

6- Now *condition* is again tested. Because, value of *i* is *2* which is less than *3* so *Pakistan* is again displayed on screen. The value of *i* gets incremented to 3.

7- *Now condition is again tested. Because, value of i is 3 which is not less than 3, so the condition turns out to be false and control comes out of the loop.*

PROGRAMMING TIME 4.3

Write a program that displays the values from 1 – 10 on the computer screen.

Program:

```
for(int i = 1; i <= 10; i++)
{
        printf("%d\n", i);
}
```

Continued

Description:

Consider the example program given above.

- First of all, the value of *i* is set to *1* and then *condition* is checked.
- As the *condition* is *true (1<=10)* so *loop body* executes. As in the *loop body*, we are displaying the value of the counter variable, so *1* is displayed on console.
- After increment, the value of *i* becomes *2*. The *condition* is again checked. It is *true* as *(2<=10)* so this time *2* is printed.
- The procedure continues till *10* is displayed and after increment the value of *i* becomes *11*. *Condition* is checked and it turns out to be *false (11>10)* so the loop finally terminates after printing the numbers from *1* to *10*.

ACTIVITY 4.1

Write a program that displays the table of 2.

IMPORTANT TIP

Always make sure that the condition becomes false at some point, otherwise the loop repeats infinitely and never terminates.

DID YOU KNOW?

Each run of a loop is called an iteration.

PROGRAMMING TIME 4.4

Write a program that calculates the factorial of a number input by user.

Program Logic:

When we want to solve a problem programmatically, first we need to know exactly what we want to achieve. In this example, we are required to find the factorial of a given number, so first we need to know the formula to find factorial of a number.

N! = 1 * 2 * 3 * 4 * ..... * (N – 1) * N

We can see the pattern that is being repeated, so we can solve the problem using for loop.

Continued

Program:

```
#include<stdio.h>
void main()
{
      int n, fact = 1;
      printf("Please enter a positive number whose factorial
      you want to find");
      scanf("%d", &n);
      for(int i = 1; i <= n; i++)
      {
            fact = fact * i;
      }
      printf("The factorial of input number %d is %d", n,
      fact);
}
```

Description:

Following table shows the working of program, if the input number is 5. It demonstrates the changes in the values of variables at each iteration.

Iteration	Value of counter	Condition	Loop body	Result
				fact=1
1	i = 1	TRUE (1<=5)	fact = fact * i	fact=1*1=1
2	i = 2	TRUE (2<=5)	fact = fact * i	fact=1*2=2
3	i = 3	TRUE (3<=5)	fact = fact * i	fact=2*3=6
4	i = 4	TRUE (4<=5)	fact = fact * i	fact=6*4=24
5	i = 5	TRUE (5<=5)	fact = fact * i	fact=24*5=120
6	i = 6	FALSE (6>5)		

4.2.3 Nested Loops

Let's carefully observe the general structure of a loop.

```
for(initialization; condition; increment/decrement)
{
      Code to repeat
}
```

We can observe that *Code to repeat* could be any valid C language code. It can also be another *for* loop e.g. the following structure is a valid loop structure.

```
for(initialization; condition; increment/decrement)
{
      for(initialization; condition; increment/decrement)
      {
            Code to repeat
      }
}
```

When we use a loop inside another loop, it is called nested loop structure.

When do we use nested loops?

When we want to repeat a pattern for multiple times, then we use nested loops, e.g. if 10 times we want to display the numbers from 1 – 10. We can do this by writing the code of displaying the numbers from 1 – 10 in another loop that runs 10 times.

PROGRAMMING TIME 4.5

Problem:

Write a program that 5 times displays the numbers from 1 – 10 on computer screen.

Program:

```
#include<stdio.h>
void main()
{
      for(int i = 1; i <= 5; i++)
      {
            for(int j = 1; j <= 10; j++)
            {
                  printf("%d ", j);
            }
            printf("\n");
      }
}
```

Continued

Output:

Here is the output of above program.

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Description:

As we understand the working of inner loop, so here let's focus on outer loop.

1- For the value *i = 1*, *condition* in outer loop is checked which is *true* (1 <= 5), so whole inner loop is executed and numbers from 1 – 10 are displayed.

2- When control gets out of inner loop, `printf("\n");` is executed which inserts a new line on console.

3- Then *i* is incremented and it becomes *2*. As it is less than 5, so condition is *true*. The whole inner loop is executed, and thus numbers from 1 – 10 are again displayed on screen. Coming out of the inner loop *new line* is inserted again.

4- After five times displaying the numbers from 1 – 10 on screen, the value of *i* gets incremented to *6* and *condition* of outer loop turns *false*. So outer loop also terminates.

PROGRAMMING TIME 4.6

Problem:

Write a program to display the following pattern of stars on screen.

```
*
**
***
****
*****
******
```

Program: Continued

```
#include<stdio.h>
void main()
{
      for(int i = 1; i <= 6; i++)
      {
            for(int j = 1; j <= i; j++)
                  printf("*");
            printf("\n");
      }
}
```

Description:

Here is the description of above code.

1- As we have to display 6 lines containing stars, so we run the outer loop from 1 to 6.

2- We can observe that in the given pattern we have 1 star on 1st line, 2 stars on 2nd line, 3 stars on 3rd line and so on. So, the inner loop is dependent on the outer loop, i.e. if counter of outer loop is 1 then inner loop should run 1 time, if the counter of outer loop is 2 then inner loop should run 2 times and so on. So, we use the counter of outer loop in the termination condition of inner loop i.e. j <= i.

3- When outer loop counter *i* has value 1, inner loop only runs 1 time, so only 1 star is displayed. When outer loop counter is 2, the inner loop runs 2 times, so 2 stars are displayed and the process is repeated until six lines are complete.

ACTIVITY 4.2

Write a program that displays the tables of 2, 3, 4, 5 and 6.

IMPORTANT TIP

We can use *if* structures inside loop structures or loop structures inside *if* structures in any imaginable manners.

4.2.4 Solved Example Problems

PROGRAMMING TIME 4.7

Problem:

Write a program that counts multiples of a given number lying between two numbers.

Program:

```
#include <stdio.h>
void main ()
{
      int n, lower, upper, count = 0;
      printf ("Enter the number: ");
      scanf ("%d", &n);
      printf ("Enter the lower and upper limit of
      multiples:\n");
      scanf ("%d%d", &lower, &upper);
      for(int i = lower; i <= upper; i++)
            if(i % n == 0)
                  count++;
      printf ("Number of multiples of %d between %d and %d are
      %d", n, lower, upper, count);
}
```

PROGRAMMING TIME 4.8

Problem:

Write a program to find even numbers in integers ranging from *n1* to *n2* (where *n1* is greater than *n2*).

Program:

```
#include <stdio.h>
void main ()
{
      int n1, n2;
      printf ("Enter the lower and upper limit of even
      numbers:\n");
      scanf ("%d%d", &n2, &n1);
```

Continued

```
    if(n1 > n2)
    {
        for (int i = n1; i >= n2; I--)
        {
            if(i % 2 == 0)
                printf ("%d ", i);
        }
    }
}
```

PROGRAMMING TIME 4.9

Problem:

Write a program to determine whether a given number is prime number or not.

Program:

```
#include <stdio.h>
void main ()
{
    int n;
    int flag = 1;
    printf ("Enter a number: ");
    scanf ("%d", &n);
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
            flag = 0;
    }
    if (flag == 1)
        printf ("This is a prime number");
    else
        printf ("This is not a prime number");
}
```

PROGRAMMING TIME 4.10

Problem:

Write a program to display prime numbers ranging from 2 to 100.

Program:

```
# include<stdio.h>
int main ()
{
      int flag;
      for (int j = 2; j <= 100; j++)
      {
            flag = 1;
            for (int i = 2; i < j; i++)
            {
                  if(j % i == 0)
                  {
                        flag = 0;
                  }
            }
            if (flag == 1)
            {
                  printf ("%d ", j);
            }
      }
}
```


4.2.5 Loops and Arrays

As variables can be used as array indexes, so we can use loops to perform different operations on arrays. If we want to display the whole array, then instead of writing all the elements one by one, we can loop over the array elements by using the loop counter as array index.

In the following, we discuss how loops can be used to read and write values in arrays.

1) Writing values in Arrays using Loops: Using loops, we can easily take input in arrays. If we want to take input from user in an array of size 10, we can simply use a loop as follows:

EXAMPLE CODE 4.5

```
int a[10];
for (int i = 0; i < 10; i++)
      scanf ("%d", &a[i]);
```

PROGRAMMING TIME 4.11

Problem:
Write a program that assigns first 5 multiples of 23 to an array of size 5.
Program:

```
#include<stdio.h>
void main()
{
      int multiples[5];
      for (int i = 0; i < 5; i++)
            multiples[i]= (i + 1) * 23 ;
}
```


2) Reading values from Arrays using Loops: Let's see how loops help us in reading the values from array. The following code can be used to display the elements of an array having 100 elements:

EXAMPLE CODE 4.6

```
      for (int i = 0; i < 100; i++)
            printf("%d ", a[i]);
```

The following code can be used to add all the elements of an array having 100 elements.

EXAMPLE CODE 4.7

```
int sum = 0;
for(int i = 0; i < 100; i++)
      sum = sum + a[i];
printf("The sum of all the elements of array is %d", sum);
```

ACTIVITY 4.3

Write a program that takes as input the marks obtained in matriculation by 30 students of a class. The program should display the average marks of the class.

4.2.6 Solved Example Problems

PROGRAMMING TIME 4.12

Problem:

Write a program that adds corresponding elements of two arrays.

Program:

```
#include <stdio.h>
void main ()
{
      int a[] = {2, 3, 54, 22, 67, 34, 29, 19};
      int b[] = {65,73,26, 10, 4, 2, 84, 26};
      for (int i=0; i<8; i++)
            printf ("%d   ", a[i] + b[i]);
}
```

SUMMARY

- ***Data structure*** is a container to store collection of data items in a specific layout.
- An ***Array*** is a data structure that can hold multiple values of same data type. It stores all the values at contiguous locations inside the computer's memory.
- In C language, an array can be declared as follows:

  data_type array_name[array_size];

  - ***Data Type*** is the type of data that we want to store in the array.
  - ***Array Name*** is the unique identifier that we use to refer to the array.
  - ***Array Size*** is the maximum number of elements that the array can hold.
- Assigning values to an array for the first time, is called ***Array Initialization***. An array can be initialized at the time of its declaration, or later. Array initialization at the time of declaration can be done in the following manner.

  data_type array_name[N] = {value1, value2, value3,..., valueN};
- Each element of an array has an ***index*** that can be used with the array name as *array_name[index]* to access the data stored at that particular *index*. Variables can also be used as array indices.
- ***Loop Structure*** is used to repeat a set of statements. Three types of loops are for loop, while loop, do-while loop.
- In C programming language, *for loop* has the following general structure.

```
for(initialization; condition; increment/decrement)
{
        Code to repeat;
}
```

- When we use a loop inside another loop, it is called nested loop structure. We use nested loops to repeat a pattern multiple times.
- Loops make it easier to read and write values in arrays.

Exercise

Q1 Multiple Choice Questions

1) An array is a _____ structure.

a) Loop **b)** Control **c)** Data **d)** Conditional

2) Array elements are stored at _____ memory locations.

a) Contiguous **b)** Scattered **c)** Divided **d)** None

3) If the size of an array is 100, the range of indexes will be _____.

a) 0-99 **b)** 0-100 **c)** 1-100 **d)** 2-102

4) _____ structure allows repetition of a set of instructions.

a) Loop **b)** Conditional **c)** Control **d)** Data

5) _____ is the unique identifier, used to refer to the array.

a) Data Type **b)** Array name **c)** Array size **d)** None

6) Array can be initialized _____ declaration.

a) At the time of **b)** After **c)** Before **d)** Both a & b

7) Using loops inside loops is called _____ loops.

a) For **b)** While **c)** Do-while **d)** Nested

8) _____ part of *for* loop is executed first.

a) Condition **b)** Body

c) Initialization **d)** Increment/Decrement

9) _____ make it easier to read and write values in array.

a) Loops **b)** Conditions **c)** Expressions **d)** Functions

10) To initialize the array in a single statement, initialize it _____ declaration.

a) At the time of **b)** After **c)** Before **d)** Both a & b

Q2 Define the following terms.

1) Data Structure **2)** Array **3)** Array Initialization

4) Loop Structure **5)** Nested Loops

Q3 Briefly answer the following Questions.

1) Is loop a data structure? Justify your answer.

2) What is the use of nested loops?

3) What is the advantage of initializing an array at the time of declaration?

4) Describe the structure of a *for* loop.

5) How can you declare an array? Briefly describe the three parts of array declaration.

Q4 Identify the errors in the following code segments.

a)
```
int a[] = ({2},{3},{4});
```

b)
```
for (int i = 0, i < 10, i++)
      printf (“%d\n”, i);
```

c)
```
int a[] = {1,2,3,4,5};
for (int j = 0; j < 5; j++)
      printf (“%d “, a(j));
```

d)
```
float f[] = {1.4, 3.5, 7.3, 5.9};
int size = 4;
for (int n = -1; n < size; n--)
      printf (“%f\n”, f[n]);
```

e)
```
int count = 0;
for (int i = 4; i < 6; i--)
      for (int j = i, j < 45; j++)
      {
            count++;
            printf (“%count”, count)
      }
```

Q5 Write down output of the following code segments.

a)
```
int sum = 0, p;
for (p = 5; p <= 25; p = p + 5)
      sum = sum + 5;
printf ("Sum is %d", sum);
```

b)
```
int i;
for (i = 34; i <= 60; i = i * 2)
      printf ("* ");
```

c)
```
for (int i = 50; i <= 50; i++)
{
      for (j = i; j >= 48; j--)
            printf ("j = %d \ n", j);
      printf ("i = %d\n", i);
}
```

d)
```
int i, arr[] = {2, 3, 4, 5, 6, 7, 8};
for (i = 0; i < 7; i++)
{
      printf ("%d\n", arr[i] * arr[i]);
      i++;
}
```

e)
```
int i,j;
float ar1[] = {1.1, 1.2, 1.3};
float ar2[] = {2.1, 2.2, 2.3};
for (i = 0; i < 3; i++)
      for (j = i; j < 3; j++)
            printf ("%f\n", ar1[i] * ar2[j] * i * j);
```

Programming Exercises

Exercise 1

Use loops to print the following patterns on console.

a)
```
*****
*****
*****
```

b)
```
A
BC
DEF
GHIJ
KLMN
```

Exercise 2

Write a program that takes two positive integers *a* and *b* as input and displays the value of a^b.

Exercise 3

Write a program that takes two numbers as input and displays their Greatest Common Divisor (GCD) using Euclidean method.

Exercise 4

Write a program to display factorials numbers from 1 to 7. (Hint: Use Nested Loops)

Exercise 5

Write a program that takes 10 numbers as input in an array and displays the product of first and last element on console.

Exercise 6

Write a program that declares and initializes an array of 7 elements and tells how many elements in the array are greater than 10.