Vidyavardhini's College of Engineering and Technology

C Programming

Class – FE – A **Notes – PART 1** **EVEN SEM 2020 06/02/2020**

1. **What do you mean by algorithm? Which points should be consider while developing an algorithm?**

Ans:

Algorithm is the collection of simple instructions for carrying out a task.
Algorithm can also be considered as sequence of unique instructions for solving a specific problem.
That is whatever the size of input data is; the algorithm can solve the given problem in finite amount of time.
An algorithm is translated into flow charts and then into source codes for implementation of the system.

Steps:

Steps of defining an algorithm is shown below in figure 2.

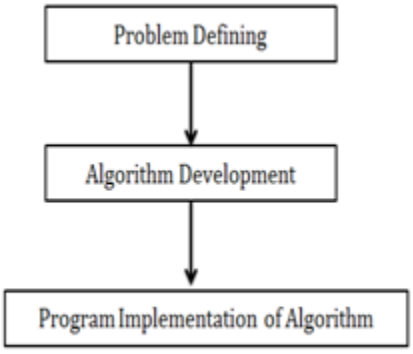


Figure 2

It includes:

- Defining your algorithm input.
- Defining variables.
- Outlining the algorithm operations.
- Output the result of your algorithm operations.

- Include special cases if any present.

Developing an algorithm in pseudo code form:

- The algorithm should always begin with the statement “START” and end with the statement “STOP”.
- To accept the input from user we will use “INPUT” or “READ” or “GET” statement.
- To display the output on monitor, we will use “PRINT” or “DISPLAY” statement.
- We will use the basic arithmetic operators to indicate the operations.
- We will use “AND”, “OR” and “NOT” to indicate conjunction, disjunction and negation respectively.
- To check conditions we can use the “IF”, “THEN” and “ELSE” constructs.
- To branch form one step to another step the construct used is “GOTO”.

2. Write an algorithm and flowchart to calculate roots of quadratic equation.

Ans:

Step Form Algorithm:

- Start.
- Declare the required variables.
- Indicate the user to enter the coefficients of the quadratic equation by displaying suitable sentences using printf() function.
- Wait using the scanf() function for the user to enter the input.
- Calculate the roots of quadratic equation using the proper formulae.
- Display the result.
- Wait for user to press a key using getch() function.
- Stop.

Pseudo Code Algorithm:

- Start.
- Input a, b, c.
- D ← sqrt (b × b – 4 × a × c).
- X1 ← (-b + d) / (2 × a).
- X2 ← (-b - d) / (2 × a).
- Print x1, x2.
- Stop.
- **Flowchart:**
- Flowchart to calculate the roots of quadratic equation is shown below in figure 3.

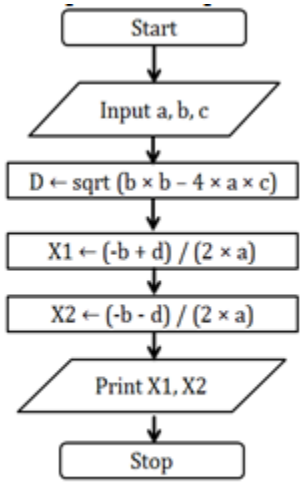


Figure 3

3. **Write an algorithm and flowchart to check whether given number is Armstrong or not.**

Ans:

What is an Armstrong number?

An Integer number in which the sum of the cubes of its digits is equal to the number itself is called Armstrong Number. For example, 153 is an Armstrong number since 1**3 + 5**3 + 3**3 = 153.

Algorithm to find whether number is Armstrong Number or Not

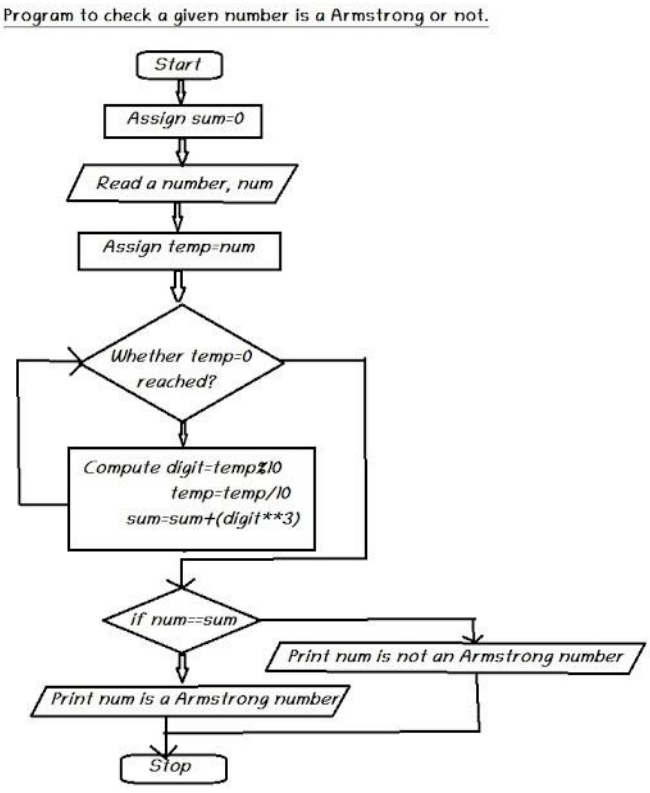
Step 1: Start
Step 2: Declare Variable sum, temp, num
Step 3: Read num from User
Step 4: Initialize Variable sum=0 and temp=num
Step 5: Repeat Until num>=0 5.1 sum=sum + cube of last digit i.e [(num%10)*(num%10)*(num%10)]
5.2 num=num/10
Step 6: IF sum==temp Print "Armstrong Number" ELSE Print "Not Armstrong Number"
Step 7: Stop

Pseudocode to find whether number is Armstrong Number or Not

```
READ n
temp=n
sum=0
WHILE n>=0
    sum=sum+(n%10)*(n%10)*(n%10)
    n=n/10
ENDWHILE
IF sum==temp
    WRITE "NUMBER IS AN ARMSTRONG NUMBER"
ELSE
    WRITE "NUMBER IS NOT AN ARMSTRONG NUMBER"
```

We first take input from user and store it in variable n. Then we initialize 2 variables temp to n and sum to 0. We calculate the cube of last digit by this expression [(n%10)*(n%10)*(n%10)] and add it to value of sum and also divide n by 10. We repeat the above step until n is greater than or equal to 0. At last, we check whether sum is equal to temp, if yes Print "Number is Armstrong Number" else print "Number is Not Armstrong Number".

Flowchart For Armstrong Number



4. **Explain in detail concept of structured programming. Define flowchart. Draw flowchart to check given number is prime number or not.**

Ans:

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Structured programming is most frequently used with deviations that allow for clearer programs in some particular cases, such as when exception handling has to be performed.

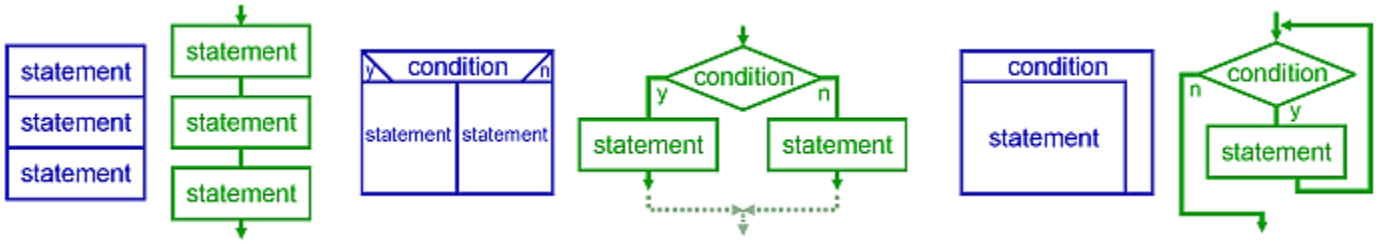
Elements

- 1.1 Control structures
- 1.2 Subroutines
- 1.3 Blocks

Control structures

Following the structured program theorem, all programs are seen as composed of control structures:

- "Sequence"; ordered statements or subroutines executed in sequence.
- "Selection"; one or a number of statements is executed depending on the state of the program. This is usually expressed with keywords such as `if..then..else..endif`.
- "Iteration"; a statement or block is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as `while`, `repeat`, `for` or `do..until`. Often it is recommended that each loop should only have one entry point (and in the original structural programming, also only one exit point, and a few languages enforce this).
- "Recursion"; a statement is executed by repeatedly calling itself until termination conditions are met. While similar in practice to iterative loops, recursive loops may be more computationally efficient, and are implemented differently as a cascading stack.



Subroutines

Subroutines; callable units such as procedures, functions, methods, or subprograms are used to allow a sequence to be referred to by a single statement.

Blocks

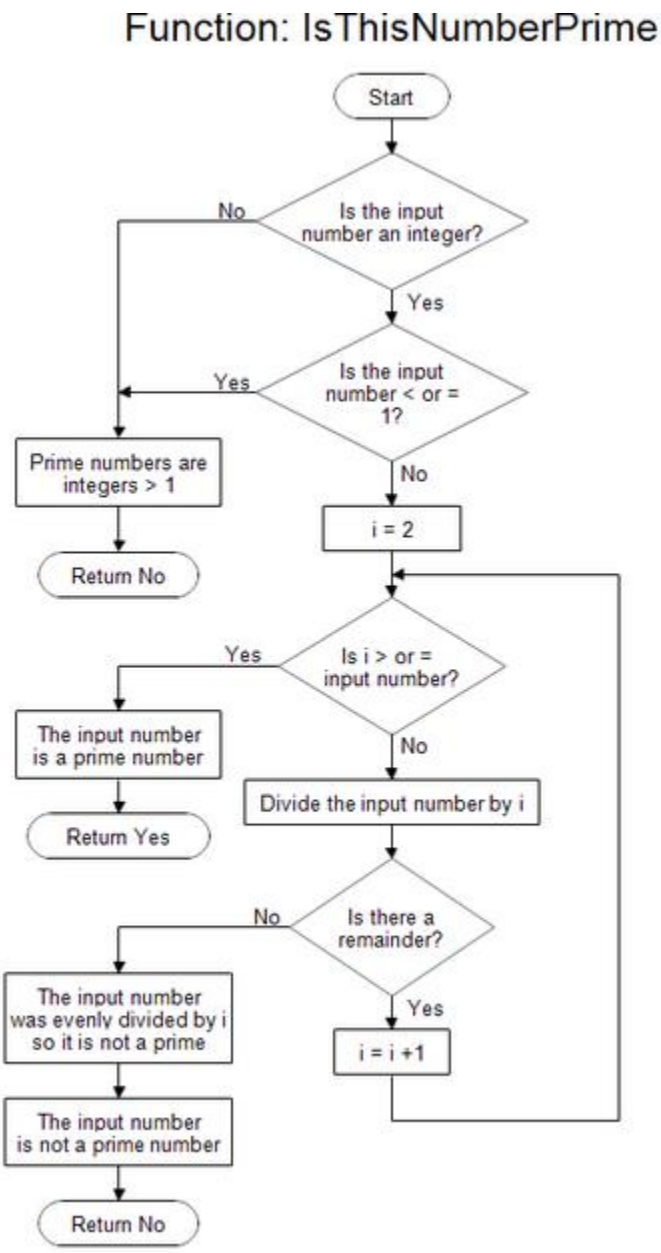
Blocks are used to enable groups of statements to be treated as if they were one statement. *Block-structured* languages have a syntax for enclosing structures in some formal way, such as an if-statement bracketed by `if..fi` as in ALGOL 68, or a code section bracketed by `BEGIN..END`, as in PL/I and Pascal, whitespace indentation as in Python - or the curly braces `{...}` of C and many later languages.

Flowchart:

It is a diagram of the sequence of movements or actions of people or things involved in a complex system or activity.

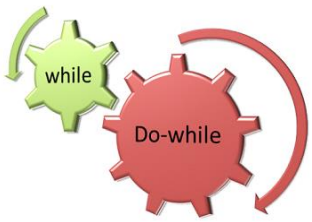
It is a graphical representation of a computer program in relation to its sequence of functions (as distinct from the data it processes).

A *flowchart* is a type of diagram that represents an algorithm, workflow or process. The *flowchart* shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem.



5. Difference between while and do while with programming example.

Ans:



Iteration statements allow the set of instructions to execute repeatedly till the condition doesn't turn out false. The Iteration statements in C are, for loop, while loop and do while loop. These statements are commonly called loops. Here, the main difference between a while loop and do while loop is that while loop check condition before iteration of the loop, whereas do-while loop, checks the condition after the execution of the statements inside the loop.

Comparison Chart

BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	while (condition) {
	statements; //body of loop
	}
		do{
		.
		statements; // body of loop.
		.
		} while(Condition);

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

Definition of while Loop

The while loop is the most fundamental loop available in C++ and Java. The working of a while loop is similar in both C++ and Java.The general form of while loop is:

```
1. while ( condition) {
2. statements; //body of loop
3. }
```

The while loop first verifies the condition, and if the condition is true then, it iterates the loop till the condition turns out false. The condition in while loop can be any boolean expression. When

expression returns any non-zero value, then the condition is “true”, and if an expression returns a zero value, the condition becomes “false”. If the condition becomes true, then loop iterates itself, and if the condition becomes false, then the control passes to the next line of the code immediately followed by the loop.

The statements or the body of the loop can either be an empty statement or a single statement or a block of statements.

Definition of do-while Loop

As in while loop, if the controlling condition becomes false in the first iteration only, then the body of the while loop is not executed at all. But the do-while loop is somewhat different from while loop. The do-while loop executes the body of the loop at least once even if the condition is false at the first attempt.

The general form of do-while is as follows.

```
1. do{
2. .
3. statements // body of loop.
4. .
5. } while( Condition );
```

In a do-while loop, the body of loop occurs before the controlling condition, and the conditional statement is at the bottom of the loop. As in while loop, here also, the body of the loop can be empty as C allow null statements or, there can be only a single statement or, a block of statements. The condition here is also a boolean expression, which is true for all non-zero value.

In a do-while loop, the control first reaches to the statement in the body of a do-while loop. The statements in the body get executed first and then the control reaches to the condition part of the loop. The condition is verified and, if it is true, the loop is iterated again, and if the condition is false, then the control resumes to the next line immediate after the loop.

Key Differences Between while and do-while Loop

1. The while loop checks the condition at the starting of the loop and if the condition is satisfied statement inside the loop, is executed. In do-while loop, the condition is checked after the execution of all statements in the body of the loop.
2. If the condition in a while loop is false not a single statement inside the loop is executed, and if the condition in ‘do-while’ loop is false then also the body of the loop is executed at least once then the condition is tested.

6. Write a program to display following pattern.

a.
```
1
1 2
1 2 3
1 2 3 4
```

b.
```
*
* *
* * *
* * * *
```

Ans:

a.

```
#include <stdio.h>
int main()
{
    int i, j, rows;

    printf("Enter number of rows: ");
    scanf("%d",&rows);

    for(i=1; i<=rows; ++i)
    {
        for(j=1; j<=i; ++j)
        {
            printf("%d ",j);
        }
        printf("\n");
    }
    return 0;
}
```

b.

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.   int n, c, k;
6.
7.   printf("Enter number of rows\n");
8.   scanf("%d", &n);
9.
10.       for (c = 1; c <= n; c++)
11.       {
12.         for(k = 1; k <= c; k++)
13.           printf("*");
14.
15.         printf("\n");
```

```
16.        }
17.
18.        return 0;
19.     }
```

7. Write a menu driven program to find area of circle, rectangle and triangle.

Ans:

```
#include <stdio.h>
void main ()
{
      int choice,r,l,w,b,h;
      float area;
      printf("Input 1 for area of circle\n");
      printf("Input 2 for area of rectangle\n");
      printf("Input 3 for area of triangle\n");
      printf("Input your choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
           case 1:
                 printf("Input radious of the circle : ");
                 scanf("%d",&r);
                 area=3.14*r*r;
                 break;
            case 2:
                  printf("Input length and width of the rectangle : ");
                  scanf("%d%d",&l,&w);
                  area=l*w;
                  break;
            case 3:
                  printf("Input the base and hight of the triangle :");
                  scanf("%d%d",&b,&h);
                  area=.5*b*h;
                  break;
          }
          printf("The area is : %f\n",area);
}
```

Sample Output:

```
Input 1 for area of circle
Input 2 for area of rectangle
Input 3 for area of triangle
Input your choice : 1
Input radious of the circle : 5
The area is : 78.500000
```

8. **Explain bitwise operators with example.**

Ans:

In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level. To perform bit-level operations in C programming, bitwise operators are used.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001

  ________
  00001000  = 8 (In decimal)
```

Example #1: Bitwise AND

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Output

```
Output = 8
```

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Example #2: Bitwise OR

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Output

```
Output = 29
```

Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

Example #3: Bitwise XOR

```
#include <stdio.h>
```

```
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

Output

```
Output = 21
```

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
  ________
  11011100  = 220 (In decimal)
```

Example #4: Bitwise complement

```
#include <stdio.h>
int main()
{
    printf("Output = %d\n",~35);
    printf("Output = %d\n",~-12);
    return 0;
}
```

Output

```
Output = -36
Output = 11
```

Shift Operators in C programming

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example #5: Shift Operators

```
#include <stdio.h>
int main()
{
    int num=212, i;
    for (i=0; i<=2; ++i)
        printf("Right shift by %d: %d\n", i, num>>i);

     printf("\n");

     for (i=0; i<=2; ++i)
        printf("Left shift by %d: %d\n", i, num<<i);

     return 0;
}
```

```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

9. **Convert decimal number into binary and also binary number into decimal.**

Ans:

10. Write a program to find factorial of a number entered by the user.

Ans:

Factorial of a Number

```
#include <stdio.h>
int main()
{
    int n, i;
    unsigned long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d",&n);

    // show error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");

    else
    {
        for(i=1; i<=n; ++i)
        {
            factorial *= i;              // factorial = factorial*i;
        }
        printf("Factorial of %d = %llu", n, factorial);
    }

    return 0;
}
```

Output

```
Enter an integer: 10
Factorial of 10 = 3628800
```

11. Write a program to sort an array elements in ascending order.

Ans:

Problem Description

This program will implement a one-dimentional array of some fixed size, filled with some random numbers, then will sort all the filled elements of the array.

Problem Solution

1. Create an array of fixed size (maximum capacity), lets say 10.
2. Take n, a variable which stores the number of elements of the array, less than maximum capacity of array.

3. Iterate via for loop to take array elements as input, and print them.
4. The array elements are in unsorted fashion, to sort them, make a nested loop.
5. In the nested loop, the each element will be compared to all the elements below it.
6. In case the element is greater than the element present below it, then they are interchanged
7. After executing the nested loop, we will obtain an array in ascending order arranged elements.

Program/Source Code

Here is source code of the C program to sort the array in an ascending order. The program is successfully compiled and tested using Turbo C compiler in windows environment. The program output is also shown below.

```
/*
 * C program to accept N numbers and arrange them in an ascending order
 */

#include <stdio.h>
void main()
{
    int i, j, a, n, number[30];
    printf("Enter the value of N \n");
    scanf("%d", &n);

    printf("Enter the numbers \n");
    for (i = 0; i < n; ++i)
        scanf("%d", &number[i]);

    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            if (number[i] > number[j])
            {
                a =  number[i];
                number[i] = number[j];
                number[j] = a;
            }
        }
     }
     printf("The numbers arranged in ascending order are given below
\n");
    for (i = 0; i < n; ++i)
        printf("%d\n", number[i]);

}
```

Program Explanation

1. Declare an array of some fixed capacity, lets say 30.
2. From users, take a number N as input, which will indicate the number of elements in the array (N <= maximum capacity)
3. Iterating through for loops (from [0 to N)), take integers as input from user and print them. These input are the elements of the array.
4. Now, create a nested for loop with i and j as iterators.
5. Start the sorting in ascending order by extracting each element at position i of outer loop.
6. This element is being compared to every element from position i+1 to size-1 (means all elements present below this extracted element)
7. In case any of the extracted element is greater than the element below it, then these two interchange their position, else the loop continues.
8. After this nested loop gets executed, we get all the elements of the array sorted in ascending order.

OUTPUT:

```
Enter the value of N
6
Enter the numbers
3
78
90
456
780
200
The numbers arranged in ascending order are given below
3
78
90
200
456
780
```

12. Write a program to delete all the occurrence of given number from an array. A = {2,5,3,2,2,9,3} and given number is 2 to delete from array. After deletion array should be A={5,3,9,3}

Ans: