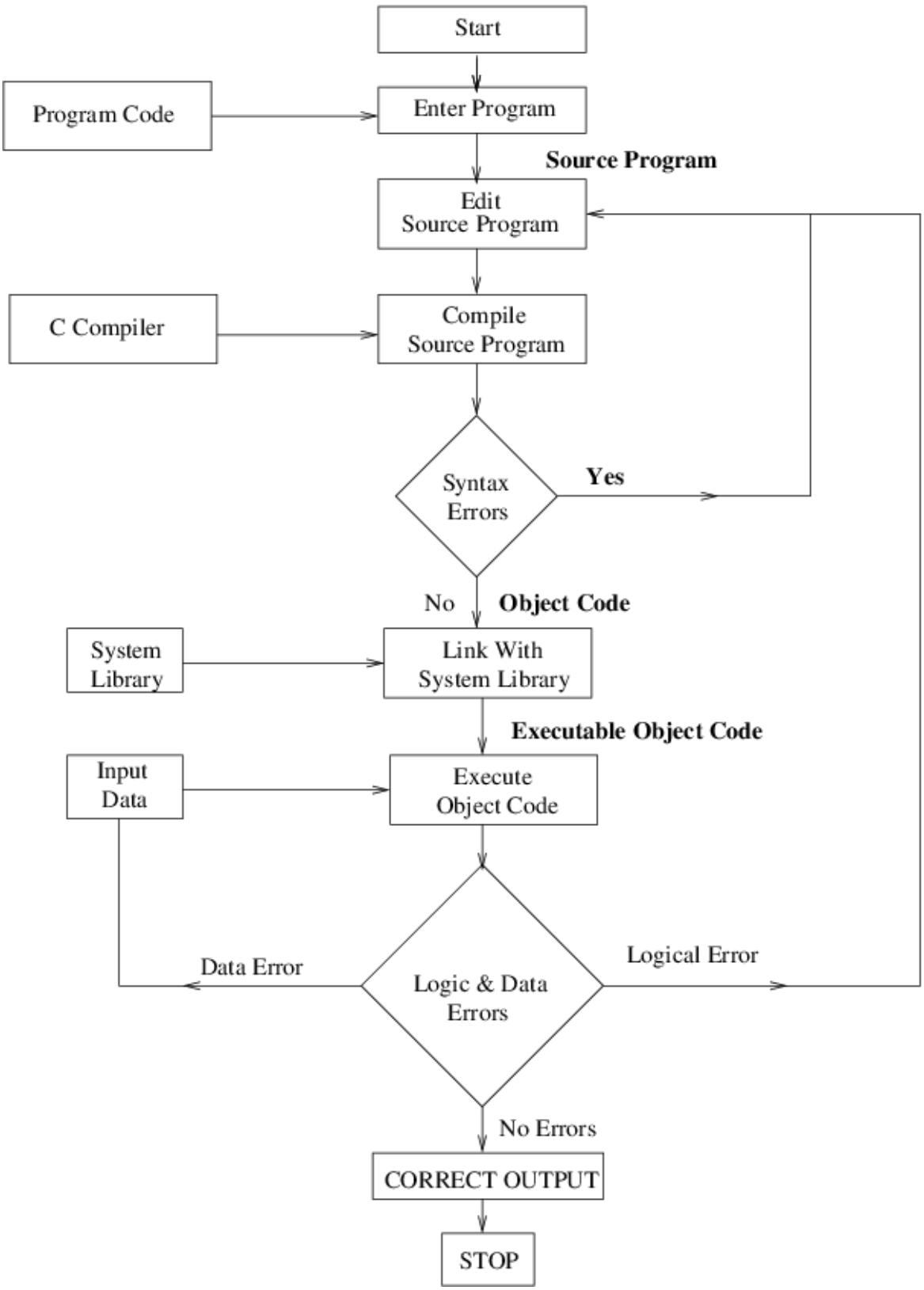
Tutorial No. 2 - Solution
(Overview of C)

1. Explain the C program development life cycle using flowchart in detail. OR

Explain the process of compiling and running a C program in detail.



The above figure illustrates the process of creating, compiling and executing a C program.
Executing a program written in C involves following series of steps:

1. Creating the program
2. Compiling the program
3. Linking the program with functions that are needed from the C library
4. Executing the program

Creating the Program

Once the computer (or operating system) is ready, the program must be entered into a file. The file is created with the help of a text editor or a software application like Turbo C.
Program or source code is a series of statements or commands that are used to instruct the computer to perform our desired tasks. Most compilers come with a built-in editor that can be used to enter source code. When we save a source file, we must give it a valid name. The name should describe what the program does. In addition, when we save C program source files, we must give the file a .C extension.

Compiling the Program

Although we might be able to understand C source code, the computer can't. A computer requires digital, or binary, instructions called machine language. Before the C program can be run on a computer, it must be translated from source code to machine language. This translation is performed by a program called a compiler. The translation is done after examining each instruction for its correctness. The compiler takes the source code file as input and produces another file containing the machine language instructions that correspond to the source code statements. The machine language instructions created by the compiler are called object code, and the file containing them is called an object file which is saved with an .OBJ extension.
The compiler mainly performs following two important functions:

1) Error checking by detecting the **syntax errors** in the program.
2) Translation of the program from higher level language to machine language.

In case any syntax errors are found in the created program, it needs to be modified again. And after the modification of the program, it is compiled once again. This process is continued until there is no syntax errors in the whole program. Once all the errors are solved, the compilation process completes successfully and the object file is generated.

Linking the Program

Linking is the process of putting together other program files and functions that are required by the program. For example if the program is using sqrt() function (square root function), then the object code of this function should be brought from the math library of the system and linked to the main program. Most of the times the process of linking is done along with the process of compiling of the program.

Executing the Program

When the program is run, the object code is loaded into the computer memory and the instructions are executed. During execution, the program may request for some data to be entered through the keyboard. Sometimes the program does not produce the desired results. Perhaps something is wrong with the

program logic or data. Then it would be necessary to correct the source program or the data. In case of **logical error**, the source code is modified and the entire process of compiling, linking and executing the program should be repeated. If the error is with the input data (**data error**), there is no need to modify the program and the program is executed once again and asked for the input again. This process is continued until the desired input data is entered. Once the desired input is supplied to the program it computes the output and gives the desired result to the user.

2. Explain in brief the features of C language. OR

What is the importance of C language?

1. **C is a General Purpose Programming Language**

   C compiler combines capabilities of an assembly language with the features of a high level language. Therefore it is well suited for writing both system softwares and application or business packages.
2. **C is a Robust (Powerful) Programming Language**
   Due to its rich set of built-in functions and operators, it can be used to write any complex program.
3. **Programs written in C are Efficient and Fast**

   This is due to its variety of data types and powerful operators
4. **C is Highly Portable**

   This means that C programs written for one computer can be run on another with little or no modification. This refers to the ability of a program to run in different environments. With the availability of compilers for almost all operating systems and hardware platforms, it is easy to write code on one system which can be easily ported to another.
5. **C Language is well Suited for Structured Programming**

   Ability to breakdown a large module into manageable sub modules called as modularity which is an important feature of structured programming languages. Thus C enables a programmer to think of a program in terms of function modules or blocks.
6. **C is an Extensive Language**

   It has ability to extend itself. A program is a collection of functions that are supported by the C library. We can continuously add our own functions to C library.
7. **C is a Case-Sensitive Language**

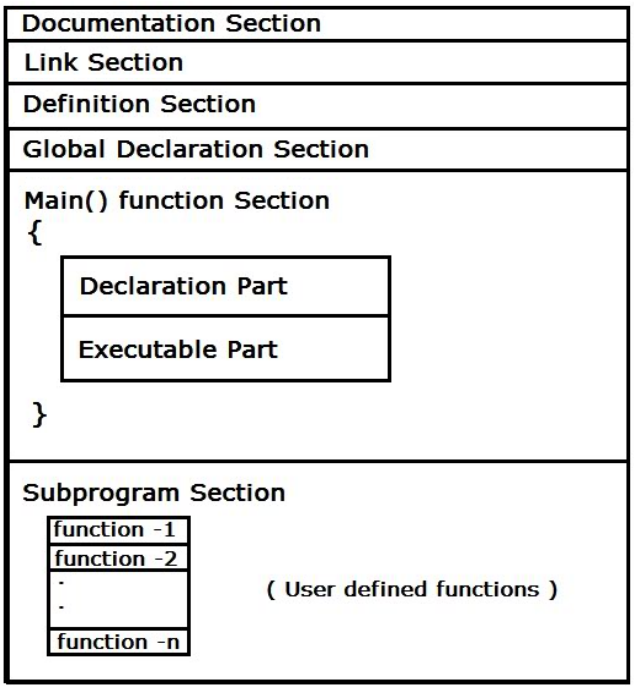
   It means that a character written in lowercase is different from the same character written in uppercase.
8. **C is a Free-Form Language**

   The C compiler does not care, where on the line we begin typing. We can group all statements of a program together in a single line.
9. **Wide Acceptability**

   C is widely popular in the software industry. Its importance is not entirely derived from its use as primary development language but also because of its use as an interface language to some of the visual languages.
10. **High Availability**

    The software of C Language is readily available in market and can be easily installed on any computer.

3. Explain the basic structure of a C program?



1. **Documentation Section**

   It consists of a set of comment lines giving the name of the program, the author and other details which the programmer would like to use later.

2. **Link Section**

   It provides instructions to the compiler to link functions from the system library. It consists of pre-processor statements which begin with # symbol and are also called the pre-processor directive. These statements instruct the compiler to include C pre-processors such as header files and symbolic constants before compiling the C program.

3. **Definition Section**

   This section defines all the symbolic constants.

4. **Global Declaration Section**

   This section is used to declare the global variables used in the program. Global variables are the variables used in more than one function. This section also declares all the user defined functions

5. **main() Function Section**

   Every C program must have one main function section. This section contains two parts:

   1) **Declaration Part** declares all the variables used in the executable part.

   2) **Execution Part** must contain at least one statement which contains instructions to perform certain tasks.

   The declaration and executable part must appear between the opening and closing braces. All statements in the declaration part must end with the semicolon.

6. **Subprogram Section**

   This section contains all the user defined functions that are called in the main function. These functions are performed by user specific tasks and this also contains set of program statements. They may be written before or after a main () function and called within main () function

4. Why and when do we use #include directive?

C programs are divided into modules or functions. Some functions are written by users, and many others are stored in the C library. Library functions are grouped category-wise and stored in different files known as header files. If we want to access the functions stored in the library, it is necessary to tell the compiler about the files to be accessed.

An external file containing functions or macro definitions can be included as a part of a program so that we need not rewrite those functions or macro definitions.

This is achieved by using the preprocessor directive #include as follows:

#include<filename> or #include “filename”

where, filename is the name of the library file that contains the required function definition. At this point the preprocessor inserts the entire contents of filename into the source code of the program. When the filename is included within the double quotation marks, the search for the file is made first in the current directory and then in the standard directories. And when the filename is written without double quotation marks, the file is searched only in the standard directories.

Typically, such statements are placed at the top of a program--hence the name "header file" for files thus included.

The #include directive tells the compiler to include all the functions, and stuff from a header file.

For eg-:

#include <stdio.h>

It means we are telling the compiler to include the code from the header file stdio.h in our program.

After including a header file in the program, we can use the functions defined in it, in our program.

5. Why and when do we use #define directive?

#define defines a macro substitution. Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens. This statement, usually known as macro definition takes the following general form:

#define identifier string

If this statement is included in the program at the beginning, then the preprocessor replaces every occurrence of the *identifier* in the source code by the *string*. The keyword #define is written just as shown above, followed by the identifier and a string with at least one blank space between them.

The #define is a preprocessor compiler directive and not a statement. Therefore #define lines should not end with a semicolon. Symbolic constants are generally written in uppercase so that they are easily distinguished from lowercase variable names. #define instructions are usually placed at the beginning before the main() function.

The #define instruction defines value to a symbolic constant for use in the program.

For eg-:

#define PI 3.14

It means we are telling the compiler to replace all the occurrences of the identifier “PI” with the string/value 3.14, in our program.

6. What is the need of using comments in a C program? What are the different ways of using comment lines in a program?

Commenting involves placing human readable descriptions inside of computer programs detailing what the code is doing. Proper use of commenting can make code maintenance much easier, as well as helping make finding bugs faster.

All programs should be commented in such a manner as to easily describe (in English) the purpose of the code and any algorithms used to accomplish the purpose.

Commenting is the art of describing what our program is going to do in "high level" English statements. Commenting is best done before actually writing the code for your program. Comments are specially marked lines of text in the program that are not evaluated.

There are usually two ways of writing comments in a program:

1) Single line comment

Single line comment, as implied, only applies to a single line in the source code.

In C a single line can be made a comment line by placing the line comment delimiter "//" at the beginning of the line.

For eg.

```
// This is a Comment line in C
```

2) Multiple line comment or Block comment

Block comment usually refers to a paragraph of text. A block comment has a start symbol and an end symbol and everything between is ignored by the computer. The lines beginning with /* and ending with */ becomes comment lines.

For eg.

```
/* This is first comment line
This is second comment line
This is third comment line */
```

Need of using Comments in a program

1) It provides documentation of program definition and other necessary details about the program.
2) Comments make the program more user friendly by enhancing the program's readability and understanding.
3) They inform the person viewing the code, what the code actually does. This is helpful when a programmer revisits the code at a later stage or when a third person views the code written by some other programmer.
4) Comments are useful in understanding large programs and program flow.
5) The compiler ignores all the comments. Hence commenting does not affect the execution speed (or efficiency) of the program.
6) One can use comments to comment out some sections of code when finding errors, instead of deletion. Thus comments are often helpful in finding out errors in the program.

7. Write a program to display the equation of a line in the form

aX + bY = c
For a=5, b=8 and c=18.

```
// Program for displaying the Equation of a line
#include<stdio.h>
#include<conio.h>

void main()
{
    int a=5,b=8,c=18;
    int x,y,z;
    clrscr();
    printf("The Equation of line is\n");
    printf("%dX + %dY = %d",a,b,c);
    getch();
}
```

8. Given the values of three variables a, b and c, write a program to compute and display the value of X, where X = a / (b – c).

```
// Program for finding the value of X
#include<stdio.h>
#include<conio.h>

void main()
{
    int a,b,c,x;
    clrscr();
    printf("Enter the values of a, b and c");
    scanf("%d %d %d", &a,&b,&c);
    x=a/(b-c);
    printf("The value of X is %d",x);
    getch();
}
```

9. A point on the circumference of a circle whose center is (0, 0) is (4, 5). Write a program to compute perimeter and area of the circle.

```
// Program for finding perimeter and area of a circle
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

```
void main()
{
    int x1=0,y1=0,x2=4,y2=5;
    float radius,perimeter,area;
    clrscr();
    radius=sqrt( ((x2-x1)*(x2-x1)) + ((y2-y1)*(y2-y1))  );
    perimeter=2*3.14*radius;
    area=3.14*radius*radius;
    printf("Perimeter : %f\n",perimeter);
    printf("Area : %f",area);
    getch();
}
```

10. State whether the following statements are true or false with suitable reason

a. Every line in a C program should end with a semicolon.

False.

Not every line of a C program needs to end with a semicolon. Only the statements (or lines) within the main function of the C program must end with a semicolon.

b. A printf statement can generate only one line of output.

False

A printf statement can generate multiple lines of output by using backslash character constant (or escape sequence) '\n'.

c. The closing brace of the main() function in a program is the logical end of the program.

True

d. The purpose of the header file is to store the source code of a program.

False

The purpose of header file is to store various functions and macros definitions and not the source code of any program.

e. Syntax errors will be detected by the compiler.

True

f. Use of comments reduces the speed of execution of a program.

False

Since the compiler ignores all the comment lines, the speed of execution of any program remain unaffected by the use of comment lines in the program.