Overview of Computer Software and Programming Language

System Software

System software is computer software designed to operate the computer hardware and to provide a platform for running application software.

The most basic types of system software are:

- The computer BIOS and device firmware, provides basic functionality to operate and control the hardware connected to or built into the computer.
- The operating system (prominent examples being Microsoft Windows, Mac OS X and Linux), which allows the parts of a computer to work together by performing tasks like transferring data between memory and disks or rendering output onto a display device. It also provides a platform to run high-level system software and application software.

Types of System Software Programs:

There are 4 types of system software—operating systems, compilers, interpreters and assemblers. Examples of system software:

- BIOS
- Microsoft Windows
- MAC OS X
- Linux

Application Software

In contrast to system software, software that allows users to do things like create text documents, play games, listen to music, or surf the web is called application software. So, it is computer software designed to help the user to perform specific tasks. Examples include enterprise software, accounting software, office suites, graphics software and media players. Application software applies the power of a particular computing platform or system software to a particular purpose. Some apps such as Microsoft Office are available in versions for several different platforms; others have narrower requirements and are thus called, for example, a Geography application for Windows or an Android application for education or Linux gaming.

Classification of Application software

a. Customized or Tailored Software:

Customized or Tailored Software is the application software which is designed to fulfill the specific requirements of an organization, office or individual. SLC Result Processing Software, Hospital Management Software, School Management Software, Bill Processing Software, Air Ticket Reservation Software, Banking software etc. customized software.

b. Packaged Software

Packaged Software is the readymade software developed for all general users to perform their generalized tasks. Software companies use to develop packaged software. Some commonly used packages are given below.
**Word Processing Software:** It is used for creating documents. Examples: MS-Word, Aldus PageMaker, Word Perfect, etc.

**Electronic Spreadsheet Software:** It is used for keeping accounts and doing calculations. Examples: - MS-Excel, Lotus 1-2-3, etc.

**Database Management System Software:** It is used for database management. Examples: - MS-Access, Dbase, FoxPro, Sybase, SQL Server, Oracle, etc.

**Graphics Software:** It is used for creating and manipulating images. Examples: - CorelDraw, Paintbrush, Photoshop etc.

**Multimedia Software:** It is used for designing multimedia. Examples: - 3D Max, Maya, Flash, PowerPoint, Windows Media Player, etc.

c. **Utility Software**

Utility Software is the helpful software that performs specific tasks related to the maintenance of computer hardware and data. Utility Software helps to keep a computer in the smooth functioning condition. Utility Software provides facilities for performing tasks like transferring data and file, recovering lost data and file, searching and removing computer viruses, disk management etc. PC Tools, Download Accelerator, Anti Viruses, WinZip, Winrar etc. are some examples of Utility Software.

**General Software Features and Recent Trends**

**What major trends are occurring in software?**

Two major software trends are taking place:

**Trend away from:**
(1) Custom-designed programs developed by the professional programmers of an organization.
(2) Technical, machine-specific programming languages using binary-based or symbolic codes
(3) Procedural languages, which use brief statements and mathematical expressions to specify the sequence of instructions a computer must perform.

**Trend towards:**
(1) Use of off-the-shelf software packages acquired by end users from software vendors.
(2) Use of a visual graphic-interface for object-oriented programming, or toward nonprocedural natural languages for programming that are closer to human conversation.

**Reasons for these trends are:**

- Development of relatively inexpensive and easy-to-use application software packages and multipurpose software suites for microcomputers.
- Software packages are designed with web-enabled networking capabilities and collaboration features that optimize their usefulness for end users and workgroups on the Internet and corporate intranets and extranets.
- Many software packages can now be downloaded, updated, managed, and rented or leased from software companies or application service providers (ASP’s) over the Internet and corporate intranets.
- Creation of easy-to-use, nonprocedural fourth-generation languages (4GLs).
Developments in object technology, graphics, and artificial intelligence produce natural language and graphical user interfaces (GUI) that make both programming tools and software packages easier to use. Developments in a new generation of expert-assisted software packages that combine expert system modules and artificial intelligence features (wizards and intelligent agents)

**Results of these trends:**

Current trends are converging to produce a fifth generation of powerful, multipurpose, expert-assisted and network-enabled software packages with natural language and graphical interfaces to support the productivity and collaboration of both end users and are professionals.

**Generation of Programming Language**

Programming languages are the languages in which computer programs are written. A programming language allows a programmer or end user to develop the sets of instructions that constitute a computer program.

- Machine Languages
- Assembler Languages
- High-Level Languages
- Fourth Generation Languages

**Machine Languages:**

Machine languages are the most basic level of programming languages. They were the first generation of programming languages.

**Advantages**

- It makes fast and efficient use of the computer.
- It requires no translator to translate the code

**Disadvantages**

- Programs had to be written using binary codes unique to each computer.
- Programming involved the difficult task of writing instructions in the form of strings of binary digits (ones and zeros) or other number systems.
- Programmers had to have a detailed knowledge of the internal operations of the specific type of CPU they were using.
- Programmers had to write long series of detailed instructions to accomplish even simple processing tasks.
- Programming is difficult and error-prone.

**Assembler Languages:**

Assembler languages are the second generation of programming languages. They were developed to reduce the difficulties in writing machine language programs. Assembler languages are frequently called symbolic languages because symbols are used to represent operation codes and storage locations.

**Advantages:**

- Uses symbolic coded instructions, which are easier to remember.
- Programming is simplified, as a programmer does not need to know the exact storage location of data and instructions.
• Provides programmers greater control and flexibility in designing a program for a particular computer. Programmers can produce more efficient software that requires a minimum of instructions, storage, and CPU time to perform a specific processing assignment.

Disadvantages:

• Assembler languages are unique to particular lines of computers.
• Programs written in assembly languages are usually long.

High-Level Languages:

They languages are called high level language if their syntaxes are closer to human language. They are developed to make programming easier. High-level languages are the third generation of programming languages. They include COBOL (business application programs), BASIC (microcomputer end users), and FORTRAN (scientific and engineering applications).

Advantages:

• Easier to learn and understand than an assembler language as instructions (statements) resemble human language or the standard notation of mathematics.
• Have less-rigid rules, forms, and syntax’s, so the potential for error is reduced.
• Are machine-independent programs therefore programs written in a high-level language do not have to be reprogrammed when a new computer is installed.
• Programmers do not have to learn a new language for each computer they program.

Disadvantages:

Less efficient than assembler language programs and require a greater amount of computer time for translation into machine instructions.

Fourth-Generation Languages (4GLs):

Fourth-generation languages (4GLs) include a variety of programming languages that are more non-procedural and conversational than prior languages.

Advantages:

• Simplify the programming process.
• Use nonprocedural languages that encourage users and programmers to specify the results they want, while the computer determines the sequence of instructions that will accomplish those results.
• Use natural languages that impose no rigid grammatical rules.

Disadvantages:

• Less flexible that other languages.
• Less efficient (in terms of processing speeds and amount of storage capacity needed).

Categorization of high level language:

The high level language is basically classified into 3 types namely
Procedural Language:

The most common high-level languages today are procedure-oriented languages. In these languages, one or more related blocks of statements that perform some complete function are grouped together into a program module, or procedure, and given a name such as “Procedure A.” Procedural languages allow programs to be shorter and easier for the computer to read, but they require the programmer to design each procedure to be general enough to be used in different situations.

Functional Language:

Functional languages treat procedures like mathematical functions and allow them to be processed like any other data in a program. This allows a much higher and more rigorous level of program construction. Functional languages also allow variables—symbols for data that can be specified and changed by the user as the program is running—to be given values only once. This simplifies programming by reducing the need to be concerned with the exact order of statement execution, since a variable does not have to be redeclared, or restated, each time it is used in a program statement. Many of the ideas from functional languages have become key parts of many modern procedural languages.

Object Oriented Language:

Object-oriented languages are outgrowths of functional languages. Object-oriented programming (OOP) languages tie data elements and the procedures or actions that will be performed on them together into objects. Examples include Visual Basic, Turbo C++, C++, Object C++, and Java.

Advantages:

- OOP languages are easier to use and more efficient for programming the graphics-oriented user interface required by many applications.
- Programmed objects are reusable.

What is the difference between a "Compiler" and an "Interpreter"?

Compilers and interpreters have similar functions. They take a program written in some programming language and translate it into machine language. A compiler does the translation all at once. It produces a complete machine language program that can then be executed. An interpreter, on the other hand, just translates one instruction at a time, and then executes that instruction immediately. The initial start up time to run program is more as compared to Interpreter.

Why C is called Middle level language?

C language supports high level language which is user friendly and low level language which is machine friendly. C contains many features of high level language through function, it gives a
modular programming and breakup, increased the efficiency for resolvability. C language is also related to machine level language (low). It provides features like access to memory using pointers and assembly aspects. It reduces the gap between low level language and high level language. So, C language is called as a Middle Level Language.
Problem Solving using Computer

What is a problem?

Problem is defined as the difference between an existing situation and a desired situation, that is, in accordance with calculation; a problem is numerical situation and has complex form. Solution is desired situation and has simplest form. If a problem is solved by computing using machine called computer, then such process is called Problem Solving using Computer.

Problem Analysis

If you have studied a problem statement, then you must analyse the problem and determine how to solve it. First, you should know the type of problem that is, nature of problem. In programming point of view, the problem must be computing. At first you try to solve manually. If it is solvable manually by using your idea and knowledge, then you can use such idea and principle in programming and solve the problem by using computer. So, you must have well knowledge about a problem. In order to get exact solution, you must analyse the problem. To analyse means you should try to know the steps that lead you to have an exact solution.

Suppose you are asked by your father to solve an arithmetic problem and you are not familiar with the steps involved in solving that problem. In such a situation, you will not be able to solve the problem. The same principle applies to writing computer program also. A programmer cannot write the instruction to be followed by a computer unless the programmer knows how to solve the problem manually.

Suppose you know the steps to be followed for solving the given problem but while solving the problem you forget to apply some steps or you apply the calculation steps in the wrong sequences. Obviously, you will get a wrong answer. Similarly, while writing a computer program, if the programmer leaves out some of the instructions for the computer or writes the instructions in the wrong sequences, then the computer will calculate a wrong answer. Thus to produce an effective computer program, it is necessary that the programmers write each and every instruction in the proper sequence. However, the instruction sequence (logic) of a computer program can be very complex. Hence, in order to ensure that the program instructions are appropriate for the problem and are in correct sequence, program must be planned before they are written.
Algorithm Development & Flowcharting:
The term algorithm may be formally defined as a sequence of instructions designed in such a way that if the instructions are executed in the specified sequence, the desired result will be obtained. An algorithm must possess the following characteristics:

1. Each and every instruction should be precise and unambiguous.
2. Each instruction should be such that it can be performed in a finite time.
3. One or more instruction should not be repeated infinitely. This ensures that the algorithm will ultimately terminate.
4. After performing the instructions, that is after the algorithm terminates, the desired results must be obtained.

Problem:
There are 50 students in a class who appeared in their final examination. Their mark sheets have been given to you. Write an algorithm to calculate and print the total number of students who passed in first division.

Algorithm:
Step 1: Initialize Total First Division and Total Mark sheet checked to zero i.e.
   \[ \text{total\_first\_div} = 0; \]
   \[ \text{total\_marksheet\_chkd} = 0; \]
Step 2: Take the mark sheet of the next student.
Step 3: Check the division column of the mark sheet to see if it is I: if no, go to step 5.
Step 4: Add 1 to Total First Division i.e.
   \[ \text{total\_first\_div} +1; \]
Step 5: Add 1 to Total Mark sheets checked i.e.
   \[ \text{total\_marksheet\_chkd} +1; \]
Step 6: Is Total Mark sheets checked = 50 : if no go to step 2
Step 7: Print Total First Division.
Step 8: Stop (End)

The above mentioned example is simpler one but development of an algorithm of a complex problem is very difficult. It may also be noted that in order to solve a given problem, each and every instruction must be strictly carried out in a particular sequence.

Flowchart:
A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed.

Normally, an algorithm is first represented in the form of a flowchart and the flowchart is then expressed in some programming language to prepare a computer program. The main advantage of this two steps approach in program writing is that while drawing a flowchart one is not concerned with the details of the elements of programming language. Since a flowchart shows the flow of operations in pictorial form, any error in the logic of the
procedure can be detected more easily than in the case of a program. Once the flowchart is ready, the programmer can forget about the logic and can concentrate only on coding the operations in each box of the flowchart in terms of the statements of the programming language. This will normally ensure an error-free program.

A flowchart, therefore, is a picture of the logic to be included in the computer program. It is simply a method of assisting the program to lay out, in a visual, two dimensional format, ideas on how to organize a sequence of steps necessary to solve a problem by a computer. It is basically the plan to be followed when a program is written. It acts like a road map for a programmer and guides him/her how to go from starting point to the final point while writing a computer program.

Experienced programmers sometimes write programs without drawing the flowchart. However, for a beginner it is recommended that a flowchart be drawn first in order to reduce the number of errors and omissions in the program. It is a good practice to have a flowchart along with a computer program because a flowchart is very helpful during the testing of the program as well as while incorporating further modifications in the program.

**Flowchart Symbols:**
A flowchart uses boxes of different shapes to denote different types of instructions. The communication of program logic through flowcharts is made easier through the use of symbols that have standardized meanings. For example, a diamond always means a decision. Only a few symbols are needed to indicate the necessary operations in a flowchart. These symbols are standardized by the American National Standard Institute (ANSI). These symbols are listed below:
Terminal: The terminal symbol, as the name implies is used to indicate the beginning (START), ending (STOP) and pauses (HALT) in the program logic flow. It is the first symbol and last symbol in the program logic. In addition, pause (HALT) used with a terminal symbol in program logic in order to represent some error condition.

Input/Output: The input/output symbol is used to denote any function of an input/output device in the program. If there is a program instruction to input data from a disk, tape, card-reader, terminal or any other type of input device, that step will be indicated in the flowchart with an input/output symbol. Similarly, all output instructions whether it is output on a printer, magnetic tape, magnetic disk, terminal screen or any output device, are indicated in the flowchart with an input/output symbol.

Processing: A processing symbol is used in a flowchart to represent arithmetic and data movement instructions. Thus all arithmetic processes of adding, subtracting, multiplying and dividing are shown by a processing symbol. The logical process of moving data from one location of the main memory to another is also denoted by this symbol. When more than one arithmetic and data movement instructions are to be executed consecutively, they are normally placed in the same processing box and they are assumed to be executed in the order of their appearance.

Flowlines: Flowlines with arrowheads are used to indicate the flow of operations, that is, the exact sequence in which the instructions are to be executed. The normal flow of flowchart is from top to bottom and left to right. Arrowheads are required only when the normal top to bottom flow is not to be followed. However, as a good practice and in order to avoid ambiguity, flowlines are usually drawn with an arrowhead at the point of entry to a symbol. Good practice also dictates that flowlines should not cross each other and that such intersections should be avoided whenever possible.

Decision: The decision symbol is used in a flowchart to indicate a point at which a decision has to be made and a branch to one of two or more alternative points is possible. The criteria for making the decision should be indicated clearly within the decision box.

Connector: If a flowchart becomes very long, the flowlines start criss-cross at many places that causes confusion and reduces understandability of the flowchart. Whenever a flowchart becomes complex enough that the number and direction of flowlines is confusing or it spreads over more than one pages it is useful to utilize the connector symbol as a substitute for flowlines.

Problem: A student appears in an examination that consists of total 10 subjects, each subject having maximum marks of 100. The roll number of the students, his name, and the marks obtained by him in various subjects is supplied as input data. Such collection of related data items that is treated as a unit is k/a a record. Draw a flowchart for the algorithm to calculate the percentage marks obtained by the student in this examination and then to print it along
with his roll number and name.

Fig : Flowchart
Coding:
In order to make a program in any programming language, what you have written is known as code. The act of writing code in a computer language is known as coding. In other words, code is a set of instructions that a computer can understand.

Compilation & Execution:
The process by which source codes of a computer (programming language) are translated into machine codes is known as compilation. After compilation if everything is ok, the code is going under other process, that is known as execution. You can get the required output after execution process.

Debugging & Testing:
The process of finding and removing errors from a program is known as debugging. One simple method of debugging is to place print statements throughout the program to display the values of variables. It displays the dynamics of a program and allows us to examine and compare the information at various points. Once the location of an error is identify and the error is corrected, the debugging statements may be removed.

Testing is the process of reviewing and executing a program with the intent of detecting errors. Testing can be done manually and computer based testing.

Manual Testing is an effecting error-detection process and is done before the computer based testing begins. Manual testing includes code inspection by the programmer, code inspection by a test group and a review by a peer group. Computer based testing is done by computer with the help of compiler (a program that changes source codes into machine codes word by word).

Program Documentation:
Program Documentation refers to the details that describe a program. Some details may be built-in as an integral part of the program. These are known as internal documentation. Two important aspects of internal documentation are; selection of meaningful variable names and the use of comments. Selection of meaningful names is crucial for understanding the program. For example,

\[
\text{Area} = \text{Breadth} \times \text{Length};
\]

is more meaningful than

\[
\text{A} = \text{B} \times \text{L}
\]

And comments are used to describe actions parts and identification in a program. For example,

\[
/ * \text{ include file } * \ / \text{ describes parts of program}
\]

\[
/ * \text{ header file } * \ / \text{ describes parts of program.}
\]
Introduction to C

Introduction to C:
C is a general-purpose, structured programming language. Its instructions consist of terms that resemble algebraic expression, augmented by certain English keywords such as if, else, for, do and while, etc. C contains additional features that allow it to be used at a lower level, thus bridging the gap between machine language and the more conventional high level language. This flexibility allows C to be used for system programming (e.g. for writing operating systems as well as for applications programming such as for writing a program to solve mathematical equation or for writing a program to bill customers). It also resembles other high level structure programming language such as Pascal and Fortran.

Historical Development of C:
C was an offspring of the ‘Basic Combined Programming Language’ (BCPL) called B, developed in 1960s at Cambridge University. B language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named C. Since it was developed along with the UNIX operating system, it is strongly associated with UNIX. This operating system was developed at Bell Laboratories and was coded almost entirely in C.

C was used mainly in academic environments for many years, but eventually with the release of C compiler for commercial use and the increasing popularity of UNIX, it began to gain widespread support among compiler professionals. Today, C is running under a number of operating systems including Ms-DOA. C was now standardized by American National Standard Institute. Such type of C was named ANSI C.

Importance of C:
Now-a-days, the popularity of C is increasing probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used of built-in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assemble language with the features of a high-level language and therefore it well suited for writing both system software and business packages. In fact, many of the C compilers available in the market are written in C.
Programs written in C are efficient and fast. This is due to its variety of data types and powerful operators. It is many times faster than BASIC (Beginners All Purpose Symbolic Instruction Code – a high level programming language).

There are only 32 keywords and its strength lies in its built-in functions. Several standard functions are available which can be used for developing programs. C is highly portable. This means that C programs written for one computer can be seen on another with little or no modification. Portability is important if we plan to use a new computer with a different operating system.

C Language is well suited for structure programming thus requiring the user to think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. This modular structure makes program debugging, testing and maintenance.

Another important feature of C is its ability to extend itself. A C program is basically a collection of functions that are supported by the C library. We can continuously add our own function to the C library. With the availability of a large number of functions, the programming task becomes simple.

**Basic Structure of C programs:**

Every C program consists one or more modules called function. One of the function must be called main( ). A function is a sub-routine that may include one or more statements designed to perform a specific task. A C program may contain one or more sections shown in fig:

<table>
<thead>
<tr>
<th>Documentation Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Section</td>
</tr>
<tr>
<td>Global Declaration Section</td>
</tr>
<tr>
<td>Main() Function Section</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>Declaration Part</td>
</tr>
<tr>
<td>Execution Part</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Subprogram Section

| Function 1 |  |
| Function 2 | (User-defined Functions) |
|           |   |
|           |   |
| Function n |   |

Fig : Basic Structure of a C program

The documentation section 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. The link section provides instructions to the compiler to link function from the system library. The definition defines all the symbolic constants. There are some variables that are used in more than one function. Such variables are called global variables and are declared in global declaration section that is outside of all the function.

Every C program must have one main( ) function section. This section consists two parts: declaration part and executable part. The declaration part declares all the variables used in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening braces and ends at the closing brace. The closing brace of the main( ) function section is the logical end of the program. All the statements in the declaration and executable parts ends with a semicolon.

The subprogram section contains all the user-defined functions that are called in the main( ) function. User-defined functions are generally placed immediately after the main( ) function, although they may appear in any order. All section, except the main( ) function section may be absent when they are not required.

2.4 Executing a C Program

Executing a program written in C involves a series of steps:
1. Creating the program;
2. Compiling the program;
3. Linking the program with functions that are needed from the C library; and
4. Executing the program.

The program can be created using any word processing software in non-document mode. The file name should end with the characters “.c” like program.c, lab1.c, etc. Then the command under Ms DOS operating system would load the program stored in the file program.c i.e.
MSC pay .C
and generate the object code. This code is stored in another file under name ‘program.obj’. In case any language errors are found, the compilation is not completed. The program should then be corrected and compiled again.

The linking is done by the command

LINK program.obj

which generates the executable code with the filename program.exe. Now the command

./program

would execute the program and give the results.

/* First Program written in C */
/* Save it as hello.c */

#include <stdio.h> /* header file */

main ( ) /* main ( ) function */
{
    printf (“hello, world in”), /* statement */
}
output : hello, world

/* Program to calculate the area of a circle */
/* area.c */
#include <stdio.h> /* library file access */

main( ) /* function heading */
{
    float radius, area; /* variable declaration */
    printf (“Enter radius=?=”), /* output statement */
    scanf (“%f”, & radius); /* input statement */
    area = 3.14159 * radius; /* assignment statement */
    printf (“Area=%f”, area); /* output statement */
}
C Fundamentals

C Fundamentals is concerned with the basic elements used to construct simple C statements. These elements include the C character set, identifiers and keywords, data types, constants, variables and arrays, declaration expressions and statements. The purpose of this material is to introduce certain basic concept and to provide some necessary definitions.

A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of programming of data is accomplished by executing a sequence of precise instruction called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules.

Character Set:

C uses the uppercase letters A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements (e.g. constants, variables, operators, expressions, etc). The special characters are listed below:

+ - * / = % & # ! ? ^ " ' ~ \ | < > ( ) [ ] { } ; . , -

(blank space) (Horizontal tab)

(White Space)

Most versions of the language also allow certain other characters, such as @ and $ to be included with strings & comments.

Identifiers & Keywords:

C Tokens:

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are also known as C tokens. C has six types of tokens:

1. Identifiers e.g. x area _ temperature PI
2. Keywords e.g. int float for while
3. Constants e.g. -15.5 100
4. Strings e.g. “ABC” “year”
5. Operators e.g. + - *
6. Special Symbols e.g. ( ) [ ] { }

Character Set:

C uses the uppercase letters A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements (e.g. constants, variables, operators, expressions, etc). The special characters are listed below:

+ - * / = % & # ! ? ^ " ' ~ \ | < > ( ) [ ] { } ; . , -

(blank space) (Horizontal tab)

(White Space)

Most versions of the language also allow certain other characters, such as @ and $ to be included with strings & comments.

Identifiers & Keywords:

C Tokens:

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are also known as C tokens. C has six types of tokens:

1. Identifiers e.g. x area _ temperature PI
2. Keywords e.g. int float for while
3. Constants e.g. -15.5 100
4. Strings e.g. “ABC” “year”
5. Operators e.g. + - *
6. Special Symbols e.g. ( ) [ ] { }
Identifiers:

Identifiers are names that are given to various program elements, such as variables, functions and arrays. Identifiers consisted of letters and digits, in any order, except that first character must be a letter. Both upper and lower case letters are permitted, though common usage favors the use of lowercase letters for most type of identifiers. Upper and lowercase letters are not interchangeable (i.e. an uppercase letter is not equivalent to the corresponding lowercase letters). The underscore ( _ ) can also be included, and considered to be a letter. An underscore is often used in middle of an identifier. An identifier may also begin with an underscore.

Rules for Identifier:

1. First character must be an alphabet (or Underscore).
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.

The following names are valid identifiers

x a12 sum_1 _temp name area tax_rate TABLE

The following names are not valid identifier for the reason stated

4th The first character must be letter
“x” Illegal characters (“”)
Order-no Illegal character (-)
Error-flag Illegal character (blank space)

Keywords:

There are certain reserved words, called keywords that have standard, predefined meanings in C. These keywords can be used only for their intended purpose, they cannot be used as programmer-defined identifiers. The standard keywords are

auto break case char const
continue default do double else
enum extern float for goto
if int long register return
short signed sizeof static struct
switch typedef union unsigned void
volatile while

The keywords are all lowercase. Since upper and lowercase characters are not equivalent, it is possible to utilize an uppercase keyword as an identifier. Normally, however, this is not done, as it is considered a poor programming practice.

Data Types:

C language is rich in its data types. C supports several different types of data, each of which may be represented differently within the computer memory. There are three cases of data types:
1. Basic data types (Primary or Fundamental) e.g. int, char
2. Derived data types e.g. array, pointer, function
3. User defined data types e.g. structure, union, enum

The basic data types are also k/a built in data types. The basic data types are listed below. Typical memory requirements are also given:

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Description</th>
<th>Typical Memory Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>single character</td>
<td>1 byte</td>
</tr>
<tr>
<td>int</td>
<td>integer quantity</td>
<td>2 bytes</td>
</tr>
<tr>
<td>float</td>
<td>floating-point number</td>
<td>4 bytes (1 word)</td>
</tr>
<tr>
<td>double</td>
<td>double-precision floating point number</td>
<td>8 bytes (2 word)</td>
</tr>
</tbody>
</table>

In order to provide some control over the range of numbers and storage space, C has following classes: signed, unsigned, short, long.

<table>
<thead>
<tr>
<th>Types</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char or signed char</td>
<td>1 byte</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
</tr>
<tr>
<td>int</td>
<td>2 bytes</td>
</tr>
<tr>
<td>short int</td>
<td>1 byte</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>1 byte</td>
</tr>
<tr>
<td>signed int</td>
<td>2 bytes</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2 bytes</td>
</tr>
<tr>
<td>long int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>signed long int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long double</td>
<td>10 bytes</td>
</tr>
</tbody>
</table>

void is also a built-in data type used to specify the type of function. The void type has no values.

**Constants, Variables**

Constants in C refer to fixed values that do not change during the execution of a program. There are four basic types of constants in C. They are integer constants, floating point constants, character constants and string constants.

Integer and floating point constants represent numbers. They are often referred to collectively as numeric _ type constants. The following rules apply to all numeric type constants.

1. Commas and blank spaces cannot be included within the constants.
2. The constant can be preceded by a minus (-) if desired. The minus sign is an operator that changes the sign of a positive constant though it can be thought of as a
part of the constant itself.

3. The value of a constant cannot exceed specified minimum and maximum bounds. For each type of constant, these bounds will vary from one C compiler to another.

**Integer Constants:**

An integer constant is an integer-valued number. Thus it consists of a sequence of digits. Integer (number) constants can be written in three different number systems: decimal (base 10), octal (base 8) and hexadecimal (base 16). Beginning programmers rarely however use anything other than decimal integer constants.

A decimal integer constant can consists of any combination of digits taken from the set 0 through 9. If the constant contains two or more digits, the first digit must be something other than 0. Several valid decimal integer constants are shown below:

```
0   1   143   5280   12345   9999
```

The following decimal integer constants are written incorrectly for reason stated:

- 12,452 illegal character (,)
- 36.0 illegal character (.)
- 10  20  30 illegal character ( blank space )
- 123_45_6743 illegal character (-)
- 900 the first digit cannot be zero.

An octal integer constant can consist of any combination of digits taken from the set 0 through 7. However, the first digit must be 0, in order to identify the constant as an octal number.

Valid octal number (integer) constants are shown below:

```
0   01  0743  07777
```

The following octal integer constants are written incorrectly for the reason stated:

- 743 does not begin with 0.
- 05280 illegal character (8)
- 777.777 illegal character (.)

A hexadecimal integer constant must begin with either 0x or 0X. It can then be followed by any combination of digits taken from the sets 0 through 9 and a through f (either upper or lower case). The letters a through f (or A through F) represent the (decimal) quantities 10 through 15 respectively. Several valid hexadecimal integer constants are shown below:

```
0x   0X1  0X7FFF  0xabcd
```

The following hexadecimal integer constants are written incorrectly for the reason stated:

- 0X12.34 illegal character (.)
- 013E38 doesn’t begin with 0x or 0X.
- 0x.4bff illegal character (.)
- 0XDEFG illegal character(G)

Unsigned and Long Integer Constants:
Unsigned integer constants may exceed the magnitude of ordinary integer constants by approximately a factor of 1, though they may not be negative. An unsigned integer constant can be identified by appending the letter (U) (either upper or lowercase) to the end of the constant.

Long integer constants may exceed the magnitude of ordinary integer constants, but require more memory within the computer. A long integer constant can be identified by appending the letter L (either upper or lowercase) to the end of the constant.

An unsigned long integer may be specified by appending the letters UL to the end of the constant. The letters may be written in either upper or lowercase. However, the U must precede the L.

Several unsigned and long integer constants are shown below:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Number System</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000 U</td>
<td>decimal (unsigned)</td>
</tr>
<tr>
<td>123456789 L</td>
<td>decimal (long)</td>
</tr>
<tr>
<td>123456789 UL</td>
<td>decimal (unsigned long)</td>
</tr>
<tr>
<td>0123456 L</td>
<td>octal (long)</td>
</tr>
<tr>
<td>0777777 U</td>
<td>octal (unsigned)</td>
</tr>
<tr>
<td>0X50000 U</td>
<td>hexadecimal (unsigned)</td>
</tr>
<tr>
<td>0xFFFFFUL</td>
<td>hexadecimal (unsigned long)</td>
</tr>
</tbody>
</table>

Floating Point Constants:

A floating point constant is a base 10 number that contains either a decimal point or an exponent (or both).

Several valid floating point constants

<table>
<thead>
<tr>
<th>0.</th>
<th>1.</th>
<th>0.2</th>
<th>827.602</th>
</tr>
</thead>
<tbody>
<tr>
<td>500.</td>
<td>0.000743</td>
<td>12.3</td>
<td></td>
</tr>
<tr>
<td>2E.8</td>
<td>0.006e.3</td>
<td>1.6667e+8</td>
<td></td>
</tr>
</tbody>
</table>

The following are not valid floating point constants for the reason stated.

| 1    | Either a decimal point or an exponent must be present. |
| 1,000.0 | Illegal character (,) |
| 2E+10.2 | The exponent must be an integer (it cannot contain a decimal point) |
| 3E 10   | Illegal character (blank space) in the exponent. |

The quantity $3 \times 10^5$ can be represented in C by any of the following floating point constants:

| 300000. | 3e5 | 3e+5 | 3E5 | 3.0e+5 |
| .3e5    | 0.3E6 | 30E4 | 30.3E4 | 300e3 |
Character Constants:
A character constant is a single character, enclosed in apostrophes (i.e. single quotation marks).

Several character constants are shown below:

`'A' 'X' '3' '?' ' '``

Character constants have integer values that are determined by the computer’s particular character set. Thus, the value of a character constant may vary from one computer to another. The constants themselves, however, are independent of the character set. This feature eliminates the dependence of a C program on a particular character set.

Most computers, and virtually all personal computer make use of ASCII (i.e. American Standard Code for Information Interchange) character set, in which each individual character is numerically encoded with its own unique 7-bit combination (hence a total of $2^7$=128 difference characters).

Several character constant and their corresponding values, as defined by ASCII character set are shown below:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>65</td>
</tr>
<tr>
<td>'X'</td>
<td>120</td>
</tr>
<tr>
<td>'3'</td>
<td>51</td>
</tr>
<tr>
<td>'?'</td>
<td>63</td>
</tr>
<tr>
<td>' '</td>
<td>32</td>
</tr>
</tbody>
</table>

These values will be the same for all computer that utilize the ASCII character set.

String Constants:

A string consists of any number of consecutive characters (including none), enclosed in (double) quotation marks. Several string constants are shown below:

```
"green" "Washinton, D.C. 2005" "207-32-345"
"$19.95" "THE CORRECT ANSWER IS" "2*(I+3"
" " "Line 1\n Line 2\n line 3" ""
```

The string constants “Line 1\n Line 2\n line 3” extends over three lines, because of the newline characters that are embedded within the string. Thus, the string would be displayed as

```
Line 1
Line 2
Line 3
```

The compiler automatically places a null character (\0) at the end of every string constant, as the last character within the string (before the closing double quotation mark). This character is not visible when the string is displayed.

A character constant (e.g. ‘A’) and the corresponding single-character string constant (‘“A”’) are not equivalent. A character constant has an equivalent integer value, whereas a single character string constant does not have an equivalent integer value and in fact, consists of two characters – the specified character followed by the null character (\0).
Variables:

A variable is an identifier that is used to represent some specified type of information within a designated portion of the program. In its simplest form, a variable is an identifier that is used to represent a single data item, i.e., a numerical quantity or a character constant. The data item must be assigned to the variable at some point in the program. A given variable can be assigned different data items at various places within the program. Thus, the information represented by the variable can change during the execution of the program. However, the data type associated with the variable are not change.

A C program contains the following lines:

```
int a,b,c ;  
char  d ;  
- - - - -  
a = 3 ;  
b = 5 ;  
c = a + b ;  
d = ‘a’ ;  
- - - - -  
a = 4 ;  
b = 2 ;  
c = a – b ;  
d= ‘w’
```

The first two lines are not type declaration which state that a, b and c are integer variables, and that d is a character type. Thus, a, b and c will each represent an integer-valued quantity, and d will represent a single character. The type declaration will apply throughout the program.

The next four lines cause the following things to happen: the integer quantity 3 is assigned to a, 5 is assigned to b and the quantity represented by the sum a+b (i.e. 8) is assigned to c. The character ‘a’ is assigned then assigned to d.

In the third line within this group, the values of the variables a and b are accessed simply by writing the variables on the right-hand side of the equal sign.

The last four lines redefine the values assigned to the variables as the integer quantity 4 is assigned to a, replacing the earlier value, 3; then 2 is assigned to b, replacing the earlier value, 5; The difference between a and b (i.e. 2) is assigned to c, replacing the earlier value 8. Finally the character ‘w’ is assigned to d, replacing the earlier character, ‘a’.
**Declarations**

A declaration associates a group of variables with a specific data type. All variables must be declared before they can appear in executable statements.

A declaration consists of a data type, followed by one or more variable names, ending with a semicolon. Each array variable must be followed by a pair of square brackets, containing a positive integer which specifies the size (i.e. the number of elements) of the array.

A C program contains the following type declarations:

```c
int a, b, c ;
float root1, root2 ;
char flag, text [80],
```

Thus, a, b and c are declared to be integer variables, root1 and root2 are floating variables, flag is a char-type variable and text is an 80-element, char-type array. Square brackets enclosing the size specification for text.

These declarations could also have been written as follows:

```c
int a ;
int b ;
int c ;
float root1 ;
float root2 ;
char flag ;
char text [80] ;
```

A C program contains the following type declarations:

```c
short int a, b, c ;
long int r, s, t ;
    int p, q ;
```

Also written as

```c
short a, b, c ;
long r, s, t ;
int p, q ;
```

short and short int are equivalent, as are long and long int.
A C program contains the following type declarations:

- float c1, c2, c3;
- double root1, root2;

Also written as:
- long float root1, root2;

A C program contains the following type declarations:

- int c = 12;
- char star = ‘*’;
- float sum = 0.;
- double factor = 0.21023e-6

Thus, c is an integer variable whose initial value is 12, star is a char type variable initially assigned the character ‘*’, sum is a floating point variable whose initial value is 0., and factor is double precision variable whose initial value is \(0.21023 \times 10^6\).

A C program contains the following type declarations:

- char text[ ] = "California";

This declaration will cause text to be an 11-element character array. The first 10 elements will represent the 10 characters within the word California, and the 11th element will represent the null character (\0) which automatically added at the end of the string.

The declaration could also have been written char

where size of the array is explicitly specified. In such situations, it is important, however, that the size be specified correctly. If the size is too small, eg.,
char text[10] = “California”;

the character at the end of the string (in this case, the null character) will be lost. If the size is too large eg.
char text[20] = “California”;

the extra array elements may be assigned zeros, or they may be filled with meaningless characters.

The array is another kind of variable that is used extensively in C. An array is an identifier that refers to collection of data items that have the same name. The data items must all be of the same type (e.g. all integers, all characters). The individual data items are represented by their corresponding array element (i.e. the first data item is represented by the first array element, etc). The individual array elements distinguished from one another by the value that is assigned to a subscript.

```
c a l i f o r n i a \0
```

Subscript: 0 1 2 3 4 5 6 7 8 9 10
Escape Sequence:

Certain nonprinting character, as well as the backslash (\) and apostrophe (’), can be expressed in terms of escape sequences. An escape sequence always begins with a backslash and is followed by one or more special characters. For example, a linefeed (LF), which is referred to as a newline in C, can be represented as \n. Such escape sequences always represent single characters, even though they are written in terms of two or more characters.

The commonly used escape sequences are listed below:

<table>
<thead>
<tr>
<th>Character</th>
<th>Escape Sequence</th>
<th>ASCII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bell (alast)</td>
<td>\a</td>
<td>007</td>
</tr>
<tr>
<td>backspace</td>
<td>\b</td>
<td>008</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>\t</td>
<td>009</td>
</tr>
<tr>
<td>vertical tab</td>
<td>\v</td>
<td>011</td>
</tr>
<tr>
<td>newline (line feed)</td>
<td>\n</td>
<td>010</td>
</tr>
<tr>
<td>form feed</td>
<td>\f</td>
<td>012</td>
</tr>
<tr>
<td>carriage return</td>
<td>\r</td>
<td>013</td>
</tr>
<tr>
<td>quotation mark (”)</td>
<td>\”</td>
<td>034</td>
</tr>
<tr>
<td>apostrophe (‘)</td>
<td>’</td>
<td>039</td>
</tr>
<tr>
<td>question mark (?)</td>
<td>?</td>
<td>063</td>
</tr>
<tr>
<td>backslash (/)</td>
<td>\</td>
<td>092</td>
</tr>
<tr>
<td>null</td>
<td>\0</td>
<td>000</td>
</tr>
</tbody>
</table>

Several character constants are expressed in terms of escape sequences are

‘\n’  ‘\t’  ‘\b’  ‘\’  ‘\’  ‘\”  ‘\’

The last three escape sequences represent an apostrophe, backslash and a quotation mark respectively.

Escape Sequence ‘\0’ represents the null character (ASCII 000), which is used to indicate the end of a string. The null character constant ‘\0’ is not equivalent to the character constant ‘0’.

The general form ‘\000’ represents an octal digit (0 through 7). The general form of a hexadecimal escape sequence is ‘\xhh’, where each h represents a hexadecimal digit (0 through 9 and a through f).
Preprocessors Directives:
The C preprocessor is a collection of special statements, called directives, that are executed at the beginning of compilation process. Preprocessors directives usually appear at the beginning of a program. A preprocessor directive may appear anywhere within a program. Preprocessor directives follow special syntax rules that are different from the normal C syntax. They all begin with the symbol # in column one and do not require a semicolon at the end. We have already used the directives #define and #include to a limited extent. A set of commonly used preprocessor directives and their functions are listed below:

<table>
<thead>
<tr>
<th>Directives</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define</td>
<td>Defines a macro substitution</td>
</tr>
<tr>
<td>#undef</td>
<td>Undefines a macro</td>
</tr>
<tr>
<td>#include</td>
<td>Species the files to be included</td>
</tr>
<tr>
<td>#ifdef</td>
<td>Test for a macro definition</td>
</tr>
<tr>
<td>#endif</td>
<td>Specifies the end of #if</td>
</tr>
<tr>
<td>ifndef</td>
<td>Tests whether a macro is not defined</td>
</tr>
<tr>
<td>if</td>
<td>Tests a compile-time condition</td>
</tr>
<tr>
<td>else</td>
<td>Specifies alternatives when #if test fails.</td>
</tr>
</tbody>
</table>

These directives can be divided into three categories:
1. Macro substitution directives
2. File inclusion directives
3. Compiler control directives

Macro substitution directives:
Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens. The general format

```
#define identifier string
#define PI 3.1415926
#define FALSE 0
#define COUNT 100
#define CAPITAL “KATHMANDU”
```

File inclusion directives
We use File inclusion in order to include a file where functions or macros are defined.

```
#include <filename> or #include “filename”
```

Compiler control directives:
In order find files based switch or (on or off) particular line or groups of lines in a program, we use conditional compilation. For that, we use following preprocessor directives such as #ifdef, #endif, #ifndef, #if,#else.
**Typedef Statement:**

C supports a feature known as “type definition” that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables. It takes the general form:

```c
typedef type identifier;
```

where type refers to an existing data type and identifier refers to the “new” name given to the data type. The existing data type may belong to any class of type, including the user defined ones. The new type is new only in name, but not the data type. typedef cannot create a new type. Some examples of type definition are:

```c
typedef int units;
typedef float marks;
```

Here, units represent int and marks represents float. They can be later used to declare variables as follows:

```c
units batch1, batch2;
marks name1[50]; name2[50];
```

batch1 and batch2 are declared as int variable and name1[50] and name2[50] are declared as 50 element floating point array variables. The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

**Symbolic Constants:**

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent numeric constant, a character constant or a string constant. Thus, a symbolic constant allows a name to appear in place of a numeric constant, a character constant or a string. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence.

Symbolic constants are usually defined at the beginning of a program. The symbolic constants may then appear later in the program in place of the numeric constants, character constant, etc that the symbolic constants represent.

A symbolic constant is defined by writing

```c
#define name text
```

where name represents a symbolic name, typically written in uppercase letters, and text represents the sequence of characters that is associated with the symbolic name. It doesn’t require semicolon. For example

```c
#define TAXRATE 0.13
#define PI 3.141593
#define TRUE 1
#define FALSE 0
#define FRIEND “Susan”
```

```c
area = PI * radius * radius; is equivalent to
area = 3.141593 * radius * radius
```
Input and Output Operations

A program is a set of instructions that takes some data and provides some data after execution. The data that is given to a program is known as input data. Similarly, the data that is provided by a program is known as output data. Generally, input data is given to a program from a keyboard (a standard input device) or a file. The program then proceeds the input data and the result is displayed on the screen (monitor – a standard output device) or a file. Reading input data from keyboard and displaying the output data on screen, such input output system is considered as console input output.

To perform input/output operation in console mode, C has a number of input and output functions. When a program needs data, it takes the data through the input functions and sends the results to output devices through the output functions. Thus the input/output functions are the link between the user and the terminal.

As keyboard is a standard input device, the input functions used to read data from keyboard are called standard input functions. The standard input functions are scanf(), getchar(), getch(), gets(), etc. Similarly, the output functions which are used to display the result on the screen are called standard output functions. The standard output functions are printf(), putchar(), putch(), puts(), etc. The standard library stdio.h provides functions for input and output. The instruction #include<stdio.h> tells the compiler to search for a file named stdio.h and place its contents at this point in the program. The contents of the header file become part of the source code when it is compiled.

Types of I/O

The input/output functions are classified into two types:

i. Formatted functions
ii. Unformatted functions

Formatted Functions:
Formatted functions allow the input read from the keyboard or the output displayed on screen to be formatted according to our requirements. Tie input function scanf() and output function: printf() fall under this category. While displaying a certain data on screen, we can...
specify the number of digits after decimal point, number of spaces before the data, the position where the output is to be displayed, etc, using formatted functions.

Formatted Input:
Formatted input refers to an input data the has been arranged in a particular format. For example, consider the following data.

20 11.23 Ram

The above line contains three types of data and must be read according to its format. The first be read into a variable int, the second into float, and the third into char. This is possible in C using the scanf function. scanf stands for scan formatted.

The input data can be entered into the computer from a standard input device keyboard by means of the C library function scanf. This function can be used to enter any combination of numerical values, single characters and strings. The function returns the number of data items that have been entered successfully. The general syntax of scanf function is

```c
scanf (control string, arg1, arg2, ………, argn)
```

where,

- control string refers to a string containing certain required formatting information so also known as format string and arg1, arg2, ………, argn are arguments that represent the individual input data items. Actually, the arguments represent pointers that indicate the addresses of the data items within the computer’s memory.

The control string consists of individual groups of characters, with one character group for each input data item. Each character group must begin with a percent sign (%). In its simplest form, a single character group will consist of the percentage sign, followed by a conversion character which indicates the types of corresponding data item. Within the control string, multiple character groups can be contiguous, or they can be separated by whitespace (i.e. blank space), tabs or newline characters. If whitespace characters are used to separate multiple character groups in the control string, then all consecutive white-space characters in the input data will be read but ignored. The use of blank spaces as character group separators is very common.

<table>
<thead>
<tr>
<th>Conversion Characters</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Data item is a single character</td>
</tr>
<tr>
<td>d</td>
<td>Data item is a decimal integer</td>
</tr>
<tr>
<td>e</td>
<td>Data item is a floating-point value</td>
</tr>
<tr>
<td>f</td>
<td>Data item is a floating-point value</td>
</tr>
<tr>
<td>g</td>
<td>Data item is a floating point value</td>
</tr>
<tr>
<td>h</td>
<td>Data item is short integer</td>
</tr>
<tr>
<td>i</td>
<td>Data item is decimal, hexadecimal or octal integer</td>
</tr>
<tr>
<td>o</td>
<td>Data item is octal integer</td>
</tr>
</tbody>
</table>
s

Data item is a string followed by a white space character (the null character (\0) automatically be added at the end

u

Data item is an unsigned decimal integer

x

Data item is a hexadecimal integer

[...]

Data item is string which may include white space

l

Data item is long integer or double

L

Data item is long double

Example: 1
#include<stdio.h>

main()
{
    char item[20] ;
    int partno ;
    float cost ;
    ---------
    scanf (“%s %d %f”, item, &partno, &cost) ;
}

The following data items could be entered from the standard input device when the program is executed.

Biratnagar 1245 0.05
Biratnagar 1245 0.05
1245 1245 0.05
0.05
Biratnagar 1245
0.05

Example: 2
#include<stdio.h>

main()
{
    char line [80]
    ---------
    scanf (“%*[ABCDEFHIJKLMNOPQRSTUVWXYZ]”, line) ;
    ---------
}

If the string KATHMANDU ENGINEERING COLLEGE
Is entered from the standard input device when the program is executed, the entire string will be assigned to the array line since the string is comprised entirely of uppercase letters & blank spaces.

If the string were written as
Kathmandu Engineering College
then only the single letter K would be assigned to line. Since the first lowercase letter ( in this case a) would be interpreted as the first character beyond the string.

Example: 3
#include<stdio.h>
main()
{ char line [80],
  - - - - -
  scanf (“u%[^\n]”, line) ;
  - - - - -
}

A variation of this feature which is often more useful is to precede the characters within the square brackets by a circumflex (or caret). If the character within the brackets are simply the circumflex followed by a newline character, then string entered from the standard input device can contain any ASCII characters except the newline characters (line feed). Thus, the user may enter whatever he or she wishes and then press the Enter Key. The Enter Key will issue the newline character, thus signifying the end of the string.

The consecutive nonwhitespace characters that define a field. It is possible to limit the number of such characters by specifying a maximum field width for that data item. To do so, an unsigned integer indicating the field width is placed within the control spring between the percent sign (%) and the conversion character.

The data item may contain fewer characters than the specified filed width. However, the number of characters in the actual data item cannot exceed the specified field width. Any characters that extend beyond the specified filed width will not be read. Such leftover characters may be incorrectly interpreted as the components of the next data item.

Example: 4
#include<stdio.h>
main()
{
    int a, b, c ;
    - - - - -
Suppose the input data items that are entered as
1 2 3
Then the following assignment will result:
a = 1, b = 2, c = 3

If the data had been entered as
123 456 789
Then the assignment would be
a = 123, b = 456, c = 789

Now suppose that the data had been entered as
123456789
Then the assignments would be
a = 123, b = 456, c = 789

Finally, suppose that the data had been entered as
1234 5678 9
The resulting assignments would now be
a = 123, b = 4, c = 567

Example: 5
#include<stdio.h>
main()
{
    int i;
    float x;
    char c;
    scanf("%3d %5f %c", &i, &x, &c);
}

If the data items are entered as
10 256.875 T
The output would now be
10, 256.8, 7
The remaining two input characters (5 and T) will be ignored.

Example: 6
#include<stdio.h>
main()
{ short ix, iy;
  long lx, ly,
  double dx, dy,
  
  scanf ("%hd %ld %lf", &ix, &ly, &dx);
  
}

The control string in the first scanf function indicates that the first data item will be assigned to a short decimal integer variable. The second will be assigned to a long decimal integer variable, and the third will be assigned to a double precision variable.

The control string in the second scanf function indicates that the first data item will have a maximum field width of 3 characters and it will be assigned to short octal integer variable, the second data item will have a maximum field width of 7 characters and it will be assigned to a long hexadecimal integer variable, and the third data item will have a maximum field width of 15 characters and it will be assigned to double precision variable.

In most version of C, it is possible to skip over a data item, without assigning it to the designated variable or array. To do so, the % sign within the appropriate control group is followed by an asterisk (*). This feature is referred to as assignment suppression.

Example: 7

```c
#include<stdio.h>
main()
{
  char item[20] ;
  int partno ;
  float cost
  
  scanf ("%s %*d %f", item, &partno, &cost),
  
}
```

If the corresponding data item are input
fasterner 12345 0.05

fasterner is assigned to item and 0.05 will be assigned to cost. However 12345 will not be assigned partno because of asterisk, which is interpreted as an assignment suppression character.
Formatted Output:

Formatted output refers to the output of data that has been arranged in a particular format. The printf() is a built-in function which is used to output data from the computer onto a standard output device i.e. screen, This function can be used to output any combination of numerical values, single character and strings. The printf() statement provides certain features that can be used to control the alignment and spacing of print-outs on the terminals. The general form of printf() statement is printf (control string, arg1, arg2, …., argn) where control string refers to a string that contains formatting information, and arg1, arg2, …., argn are arguments that represent the individual output data item. The arguments can be written as constants, single variable or array names, or more complex expressions. Function references may also be included. In contrast to scanf() function, the arguments in a printf() function do not represent memory addresses and therefore are not preceded by ampersands. An individual character group in control string will consist of the percent sign, followed by a conversion character indicating the type of the corresponding data item.

Multiple character group can be contiguous, or they can be separated by other characters, including whitespace character. These “other” characters are simply transferred directly to the output device, where they are displayed. The use of blank spaces are character group separators is particularly common.

Several of the more frequently used conversion characters are listed below:

<table>
<thead>
<tr>
<th>Conversion Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Data item is displayed as a single data</td>
</tr>
<tr>
<td>d</td>
<td>Data item is displayed as a signed decimal in</td>
</tr>
<tr>
<td>e</td>
<td>Data item is displayed as floating point value with an exponent</td>
</tr>
<tr>
<td>f</td>
<td>Data item is displayed as floating point without an exponent</td>
</tr>
<tr>
<td>g</td>
<td>Data item is displayed as floating point value using either e-type or f-type conversion</td>
</tr>
<tr>
<td>i</td>
<td>Data item is displayed as a signed decimal integer</td>
</tr>
<tr>
<td>o</td>
<td>Data item is displayed as an octal integer without a leading zero</td>
</tr>
<tr>
<td>s</td>
<td>Data item is displayed as a string</td>
</tr>
<tr>
<td>u</td>
<td>Data item is displayed as an unsigned decimal integer</td>
</tr>
<tr>
<td>x</td>
<td>Data item is displayed as a hexadecimal integer without the leading 0x.</td>
</tr>
</tbody>
</table>

l for long int, h for signed/unsigned short, L for double.

For example: 1
#include<stdio.h>
main()
{  
    char item [20];  
    int partno;  
    float cost;  
    - - - - -  
    printf ("%s%d%f", item, partno, cost);  
}

Suppose fastener, 12345 and 0.5 have been assigned to name, partno and cost. So, the output generated will be  
    fastener 123450.5  

Example: 2
#include<stdio.h>
main()
{
    double x = 5000.0, y = 0.0025
    printf ("%f%f%f%f%f%f", x, y, x*y, x/y);  
    printf ("%e%e%e%e", x, y, x*y, x/y);  
}

The output are  
    5000.000000  0.002500  12.500000  2000000   .000000  
    5.000000e+03  2.5000000e-03  1.250000e+01  2.000000+06  

Example: 3
    / * read and write a line of text */
#include<stdio.h>
main()
{
    char line [80];  
    scanf ("%[^\n]", line);  
    printf ("%s", line);  
}

Arun Kumar  
Arun Kumar  

The general syntax of control string as  
    %[Flag] [Field width] [.precision] conversion character  

% Flags [Optional]

The flag affect the appearance of the output. They must be placed immediately after the percent sign. The flags may be -, +, 0, blank space or #.
<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Data item is left justified within the field. The blank spaces required to fill the minimum field width will be added after the data item rather than the data item.</td>
</tr>
<tr>
<td>+</td>
<td>A sign (either + or -) will precede each signed numerical data item. Without this flag, only negative data items are preceded by a sign.</td>
</tr>
<tr>
<td>0</td>
<td>causes leading zeros to appear instead of leading blanks. Applies only to data items that are right justified within a field whose minimum size is larger than the data item.</td>
</tr>
<tr>
<td>(blank space)</td>
<td>A blank space will precede each positive signed numeric data item. This flag is overwritten by the + flag if both are present.</td>
</tr>
<tr>
<td>#</td>
<td>When # flag is used with %0 or %x, it causes octal and hex items to be preceded by 0 and 0x respectively. When it is used with %e, %f or %g, it causes a decimal point to be presented in all floating point numbers, even if it is a whole number.</td>
</tr>
</tbody>
</table>

% Field Width [Optional]

The field width is an integer specifying the minimum output field width. If the number of characters in the corresponding data item is less than the specified field width then the data item will be preceded by enough leading blanks to fill the specified field. If the number of characters in the data item exceeds the specified field width, then additional space will be allocated to the data item so that the entire data item will be displayed.

% Precision [Optional]

The operation of precision field depends on the types of conversion. It must start with a period (.)

Format for Integer Output:

%wd

where w is the integer number specifying the minimum field width of output data. If the length of the variable is less than the specified field width, then the variable is right justified with leading blanks.

For example: integer number
i.e. int n = 1234

### Format vs. Output

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printf(“%d”, n);</code></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td><code>printf(“%6d”, n)</code></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td><code>printf(“%2d”, n)</code></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td><code>printf(“%-6d”, n)</code></td>
<td>0 0 1 2 3 4</td>
</tr>
</tbody>
</table>

For floating point output: The general form:

- `%w.pf`
- `%w.pe`

where `w` is the integer width including decimal point.

- `p` is the precision.
- `f` and `e` are conversion characters.

### Format vs. Output

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printf(“%.4d”, 12);</code></td>
<td>0 0 1 2</td>
</tr>
<tr>
<td><code>printf(“%.40d”, 12);</code></td>
<td>0 0 1 4</td>
</tr>
<tr>
<td><code>printf(“%-4x”, 12);</code></td>
<td>0 0 0</td>
</tr>
<tr>
<td><code>printf(“%.2f”, 12.3456);</code></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td><code>printf(“%.2e”, 12.3456);</code></td>
<td>1 2 3 e + 0 1</td>
</tr>
<tr>
<td><code>printf(“%.4g”, 12.3456);</code></td>
<td>1 2 . 3 5</td>
</tr>
<tr>
<td><code>printf(“%.3s”, “NEPAL”);</code></td>
<td>N E P</td>
</tr>
</tbody>
</table>
Output of Strings:
The general form of control string is \%w.p.s
where
w specifies the field width for display and p instructs that only the first P characters of the
string are to be displayed. The display is right justified
For example
char str[10] = “MY NEPAL”

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf (“%s”, str) ;</td>
<td>M Y N E P A L</td>
</tr>
<tr>
<td>printf (“%10s”, str) ;</td>
<td>M Y N E P A L</td>
</tr>
<tr>
<td>printf (“%4s”, str) ;</td>
<td>M Y N</td>
</tr>
<tr>
<td>printf (“%10.6s”, str) ;</td>
<td>M Y N E P</td>
</tr>
<tr>
<td>printf (“%4s”, str) ;</td>
<td>M Y N E P A L</td>
</tr>
</tbody>
</table>

Unformatted Functions:

Unformatted function do not allow the user to read or display data in desired format. These
library functions basically deal with a single character or a string of characters. The functions
getchar(), putchar(), gets(), puts(), getch(), getche(), putch() are considered as unformatted
functions.
getcchar() and putchar()

The getchar() function reads a character from a standard input device. The general syntax is

c = getchar();

where c is a valid C char type variable. When this statement is encountered,
the computer waits until a key is pressed and assign this character to c.

The putchar() function displays a character to the standard output device. The general
syntax of putchar() function is

c = putchar(character_variable);

where character_variable is a char type variable containing a character.

#include<stdio.h>
#include<conio.h>
main()
```c
{ 
    char gender ;
    printf ("Enter gender M or F : ") ;
    putchar (gender) ;
}

Output:
Enter gender M or F : M
Our gender is : M

getch(), getche() and putch() :

The functions getch() and getche() reads a single character the instant it is typed without waiting for the enter key to be hit. The difference between them is that getch() reads the character typed without echoing it on the screen, while getche() reads the character and echoes (displays) it on the screen. The general syntax of getch() :
    character_variable = getch() ;
Similarly, the syntax of getche() is
    character_variable = getche() ;
The putch() function prints a character onto the screen. The general syntax is
    putch(character_variable) ;

These three functions are defined under the standard library function conio.h and hence we should include this in our program using the instruction #include<conio.h>

#include<stdio.h>
#include<conio.h>
    main()
    {
        char ch1, ch2 ;
        printf ("Enter first character: ") ;
        ch1 = getch() ;
        printf ("\n Enter second character: ") ;
        ch2 = getche() 
        printf ("\n First character: ") ;
        putch(ch1) ;
        printf("\n Second character: ") ;
        putch(ch2) ;
        getch() ;
    }

Output:
Enter first character : a

```

Enter second character : b
First character : a
Second character : b

Since the first input is taken using getc() function, the character ‘a’ entered is not echoed. However, using getche() function, we can see what we have typed. In both cases, input accepted as soon as the character typed. The last getc() simply takes a character but does not store it anywhere. So, its work is merely to hold the output screen until a key is pressed.

gets() and puts():

The gets() function is used to read a string of text containing whitespaces, until a newline character is encountered. It offers an alternative function of scanf() function for reading strings. Unlike scanf() function, it does not skip whitespaces. The general syntax of gets() is

```
gets(string_variable);
```

The puts() function is used to display the string onto the terminal. The general syntax of puts() is

```
puts (string_variable);
```

This prints the string value of string_variable and then moves the cursor to the beginning of the next line on the screen.

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

void main()
{
    char name[20];
    printf (“Enter your name:”);
    gets (name);
    printf (“your name is:’’);
    puts (name);
}
```

Output:

```
Enter your name : Ram Kumar
Your name is : Ram Kumar
```
**Practice Questions**

**Chapter 1**

**Overview of Computer Software and Programming Languages**

1. Draw the functional block diagram of computer system and describe it.
2. What do you understand by Computer Generation? List salient features of each generation.
3. What do you mean by programming languages? Explain in details.
4. What is computer software? How many types of software are there? Explain all in details.
5. Explain about the recent trends with key features in software development.
6. Define the following terms:
   a) Source code    b) Object Code    c) Assembler    d) Linker
7. Differentiate between compiler, interpreter and assembler.
8. What do you mean by compilation and execution process? Explain in detail.

**Chapter 2**

**Problem Solving Using Computer**

1) Explain about the different types of errors that can occur in C programming.
2) What is debugging and testing? How it is done?
3) What are the steps that need to be followed for developing the application software?
4) What is a flowchart? List the various commonly used flowchart symbols. How does a flowchart help a computer programming?
5) Define problem. What are various steps to solve the problem?
7) What is algorithm development and flowcharting? What is its role in software development?
8) What is pseudo code? How it is different from algorithm?
9) In how many ways can we convert high level languages to the machine level language? Explain in detail with block diagram.
10) Write an algorithm and draw a flowchart to add and multiply two numbers.
11) Write an algorithm and draw a flowchart to find the lowest and maximum of two numbers.
12) Write an algorithm and draw a flowchart to swap the values of two variables.
13) Write an algorithm and draw a flowchart to find the distance and slope between two coordinates.
14) Write an algorithm and draw a flowchart to check whether the positive integer number entered by the user is prime or not.

15) Write a pseudo code to calculate the simple interest.

16) Write a pseudo code to read the time in hh:mm:ss format and display it in seconds.

Chapter 3 Variables, Operators and Expressions

1. What is identifier? Mention the rules to be followed while forming identifiers.

2. Differentiate between identifiers and keywords with examples.

3. Why character proceeded by \ are called an escape sequences? Why are they required?

4. Differentiate between a single character constant and a string constant.

5. What is variable? Write the syntax of declaration of variables.

6. What are the data types available in C? Explain about their memory size and range with examples. How could we extend the range of values they represent?

7. What do you mean by qualifiers? Explain with examples. Describe the purpose of the qualifiers const and volatile.

8. What are the ways of giving values to variables? Explain with examples.

9. Explain in brief about the different types of operators available in C language.

10. Describe the precedence and associativity of arithmetic operations with example.

11. What is the role of sizeof operator in C program? Explain operator precedence and associativity with suitable example.

12. What is the different between automatic type conversion and type casting?

13. Write a program to find the value of \( f(x) = x^4 + 3 \) if the value of \( x \) is given.

14. Write the output of the following and explain:-

```c
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=100, b=200, c=300,d;
    d=++a + b--;  
    d++; 
    c=a++ - d--; 
    printf("%d\n%d\n%d\n%d",a,b,c,d); 
    getch();
}
```
Chapter 4

Formatted Input Output

1. Explain about the input/output functions available in C with the syntax and example.
2. Why is formatted output required? Write a general format specification of printing different data types. Briefly discuss about each part.
3. Why are gets and puts functions used? WAP to read a string using these functions.
4. WAP that reads name, roll, section and marks percentage of 7 students and display in tabular form.
5. What is the main limitation of scanf() function to read strings?
6. Explain the way of formatted input of integer number, float number, character and string with suitable examples.
7. Explain the way of formatted output of integer number, float number, character and string with suitable examples.
8. How can a string with white spaces be read? Justify your answer with an example.
9. What do you mean by search set? Explain format specification %[characters] and %[^characters]. What is the main advantage of using these specifications in reading string?
10. Explain about getch(), getchar() and getche() functions.
11. Explain about putch() and putchar() functions.
12. What do you mean by interactive programming? Explain.