



Principles of Programming Languages (POPL)

RAJIV CHOPRA



Principles of Programming Languages

Dr. Rajiv Chopra

Asst. Professor

CSE/IT

GTBIT (GGSIPU)

New Delhi



I.K. International Publishing House Pvt. Ltd.

NEW DELHI

Published by

I.K. International Publishing House Pvt. Ltd.

S-25, Green Park Extension

Uphaar Cinema Market

New Delhi-110 016 (India)

E-mail: info@ikinternational.com

Website: www.ikbooks.com

ISBN: 978-93-84588-01-4

© 2015 I.K. International Publishing House Pvt. Ltd.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means: electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from the publisher.

Published by Krishan Makhijani for I.K. International Publishing House Pvt. Ltd., S-25, Green Park Extension, Uphaar Cinema Market, New Delhi-110 016 and Printed by Rekha Printers Pvt. Ltd., Okhla Industrial Area, Phase II, New Delhi-110 020

Principles of Programming Languages

PREFACE

We all marvel at the beautiful rose. The rose, with all its beauty and grandeur. But seldom we pause and thank the gardener who patiently matured and watered it. The book, I have written is the fruit of such unseen hands.

‘Principles of Programming Languages includes the entire gamut of the principles, rules and guidelines that must be followed both by the new programmer and an experienced programmer in order to deliver a good quality software.

Writing programs is an art. This artifact starts from scratch and then prepares you to understand what a program is and how it should be developed? Believe me while writing of this book, I imagined myself to be a student who wants to learn basic programming skills. So, you could be a very good programmer if you learn these techniques of programming.

First of all I would like to thanks the Almighty GOD who showered his blessings to write such a manuscript. I also acknowledge my parents, my wife and my twin babies for their loving and understanding support given to me during the writing of this book.

Dr. Rajiv Chopra

CONTENTS

Preface

v

1. Computer Preliminaries	1
1.1 Introduction	1
1.2 Introduction to Problem Solving	1
1.3 Overview of Computers and Computer Languages	6
1.4 Computer Hardware and Software	7
1.5 Programming Languages	9
1.6 Structured Programming	10
2. Introduction to DOS	15
2.1 Introduction	15
2.2 Setting up DOS	15
2.3 Some Basic DOS Commands	16
2.4 WATFOR-77 Commands	19
2.5 Function Keys	21
3. FORTRAN Fundamentals	23
3.1 Introduction	23
3.2 Typing FORTRAN Statements	23
3.3 FORTRAN-77 Character Set	24
3.4 Constants and Variables	25
3.5 Variables	26
3.6 Rule for Naming Variables	28
3.7 Precedence Rule for Arithmetic Operations	30
3.8 Arithmetic Assignment Statement	31
4. Numerical Input/Output	42
4.1 Introduction	42
4.2 Unformatted Input/Output	42
4.3 Formatted Input/Output	45
4.4 Input Field Specifications	47
4.5 Output Field Specifications	51
4.6 Repetition Factor	57

5. Transfer of Control	68
5.1 Introduction	68
5.2 Unconditional Transfer of Control	68
5.3 Conditional Transfer of Control	70
5.4 Double Alternatives	76
5.5 Block IF Statement for Multiple Alternatives	78
5.6 Arithmetic IF	79
5.7 Computed GO TO Statement	82
6. Looping	102
6.1 DO Loops	102
6.2 While-DO ENDWHILE Statement	108
6.3 FOR Statement	109
7. Arrays	115
7.1 Introduction	115
7.2 Subscripted Variables	115
7.3 Implied DO Loop	128
8. Function and Subroutines	150
8.1 Introduction	150
8.2 Function Subprograms	151
8.3 The Function Invocation (Calling Function Subprogram)	152
8.4 Comparison of Functions and Subroutines	154
8.5 The Common Statement	155
8.6 Multiple Entries in Subprograms	159
8.7 Multiple Returns from Subroutines	160
8.8 Equivalence Statement: (Sharing Memory)	162
8.9 The External and Intrinsic Statement	163
8.10 Block Data	165
9. Files and File Handling	180
9.1 Introduction	180
9.2 File Processing Statements	181
9.3 Rewind Statement	186
9.4 Backspace Statement	187
9.5 Endfile Statement	188
Index	215

1

COMPUTER PRELIMINARIES

1.1 INTRODUCTION

Computers are extensively used to solve variegated problems from various walks of life, like research, defence, hospitals, banks, business, education, industries to name a few. The types of problems are innumerable and each problem requires a different computational procedure to solve it. Though the computational procedure (algorithm) differs from problem to problem, still we can develop a general procedure that can be adopted to solve any of the problems. Below, we discuss this general procedure to solve the problems.

1.2 INTRODUCTION TO PROBLEM SOLVING

Problem solving on computer requires a thorough analysis of the problem. Once the problem is analyzed, developing a solution to the problem becomes much easier. Usually, the following general steps are followed to solve problems, using the computer:

1. Defining the problem
2. Analysis and design of the problem
3. Algorithm development
4. Flow charting
5. Pseudocode
6. Code or program
7. Compiling, debugging and testing

1. Defining the Problem

Making a clear, precise and accurate statement about the problem is essential. Any error in the problem definition leads to erroneous solutions.

2. Analysis and Design of the Problem

Problem analysis is essential to develop an algorithm. This step determines the variables involved in the problem and identifies the relationship between them. Once this is over, the precise specifications of variables and input/output data can be decided. Then, the analysis of problem proceeds for developing and analyzing of the model. This model is test run manually using the

actual input conforming to the data specifications decided earlier. If the results are satisfactory, then the design part moves on to the design of layout for the output, i.e., how an output should appear in the printouts or in display. This may also include output media, like line printer, computer monitor, number of copies to be printed, etc.

NOTE: All variables and I/O data in a problem require a precise detailing about them, called specifications. Some such specifications include:

1. variable types, such as **INTEGER**, **REAL**, **COMPLEX**, etc.
2. number of digits and decimal places, required to represent a numerical value accurately.
3. +ve or -ve value (signed or unsigned value)
4. range of values, etc.

3. Algorithm Development

An algorithm is a procedure for solving a problem. It consists of a finite set of precisely defined and ordered steps for solving the problem. There may be different algorithms for the same problem. The programmer should choose the one which is efficient, accurate and clear. An algorithm is considered efficient if it takes less computational time, consumes less computer memory and has response time. The accuracy can be decided by the programmer at the time of problem design. The problem is said to be clear if it is easy to understand.

For complicated problems, algorithm development becomes more complex. The common approach for such problems is top-down design using modular programming technique method. In the third chapter, you will study little more about modular programming technique, later when you study **FUNCTIONS** and **SUBROUTINES**, you will gain complete understanding of the technique.

The following example illustrates an algorithm which determines the **GREATEST** algebraic element of the given three numbers. The logic of the algorithm is simple. The comments included in the algorithm will help you in understanding the algorithm.

EXAMPLE 1.1 GREATEST algorithm: This algorithm determines the largest element of the given three numbers and assigns the value to the new variable named '**MAX**'.

1. **Read three** numbers and assign them to **A**, **B** and **C** respectively.
2. **MAX ← A** We initially assume that **A** is the largest element and its value is assigned to another variable, called **MAX**.
3. **IF MAX > B Then, MAX** (i.e., **A**) is compared with **B**.
MAX ← A else, if **MAX** is greater than **B** then **MAX** retains its earlier value, i.e., **A**.
MAX ← B. Otherwise **A** is replaced by **B**.
4. Repeat step 3 with value of element **C**.
5. Assign the greatest of the three numbers to **MAX**.
6. Finished.
7. Exit.

In the above algorithm the symbol ' \leftarrow ' is used to indicate that the **L.H.S.** element is assigned with the value of **R.H.S.** element. Below, we summarize the basic format conventions used in the formulation of an algorithm.

I. Name of an algorithm: Every algorithm should be given an identifying name written in capital letters (**GREATEST** in the above example).

II. Introductory comment: The algorithm name is followed by a chief description of the task, the algorithm performs and any other assumptions made. It also gives the names and types of variables used in the algorithm.

III. Steps: The algorithm consists of a sequence of numbered steps, describing the actions to be taken or tasks to be performed. The statements in each step are executed in left to right order.

IV. Comments: Comments are enclosed within the parentheses immediately after each step. It helps the reader in better understanding each of the individual steps.

Besides these basic format conversions, an algorithm consists of types statements and control structures at various steps in the algorithm. These are listed below without any explanation. These statements will become more clear when you actually start writing programs after learning control structures.

Statements and Control Structures

1. Assignment statement
2. IF statement
3. Case statement
4. Repeat statement
5. Go to and exit loop statement
6. Exit statement
7. Variable names
8. Data structure
9. Arithmetic operations and expressions
10. Strings and string operations
11. Relations and relation operations
12. Logical operations and expressions
13. Input and output
14. Subalgorithms (functions and subroutines)

4. Flow Charting

As programs become more complex, a flow chart is most helpful in planning, designing and structuring a program. A flow chart is a graphical representation of an algorithm which shows the logical relationship between steps involved in it. With the help of lines and arrows, it also shows the flow of control between the steps. Following guidelines can be followed for drawing flow charts:

4 Principles of Programming Languages

1. The usual direction of flow is from left to right or top to bottom. Arrowhead is required if this convention is not followed.
2. The first operation symbol is written at the top and the subsequent operation symbols are written below the previous operation symbol.
3. The flow chart must be clear, neat and easy to follow.
4. A separate flow chart for a portion of a problem can be written if the situation demands.
5. Use a test data to cross-check the correctness of the flow chart.

Some of the symbols which are used in flow charting are given below.




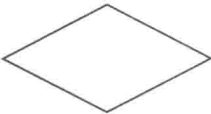

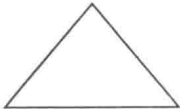
SYMBOL	USED FOR
	Start, stop, end or interrupt
	General processing of expressions
	Input or output operation
	Decision
	DO loop
	Off-line storage

Fig. 1.1 Flow chart symbols and their use

EXAMPLE 1.2 Draw a flow chart to find the sum of first 50 natural numbers.

Solution: The flow chart for summation of 50 natural numbers.

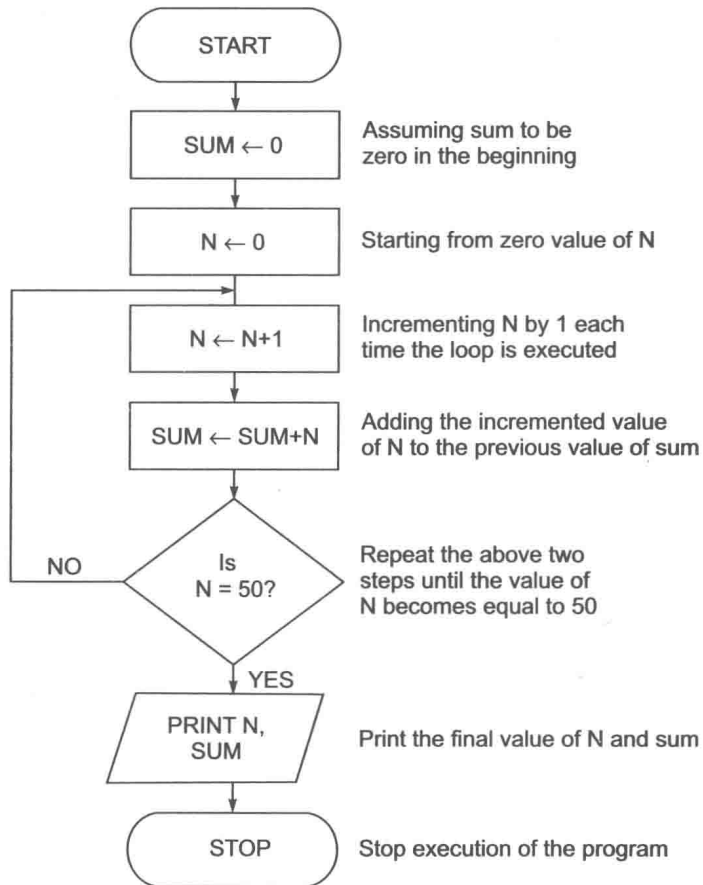


Fig. 1.2 Flow chart for summation

5. Pseudocode

It is a semiformal description of each step to be carried out by the computer to solve a particular problem. It is more readable and consists of English-like statements. It is a way of expressing an algorithm to a problem. The pseudocode should include steps that are to be repeated and decision that are to be taken. There is more than one ways of writing an algorithm in pseudocode. For example, consider a simple problem of calculating and printing the sum (S) and average (AVE) of a set of numbers (N). The basic steps in this calculation are:

1. Add all the given numbers in the set to find the sum (S).
2. Divide the sum (S) by total numbers (N) in the set to find the average (AVE).
3. Print the result.

The algorithm in pseudocode for this problem would be as follows.

```
Sum(S) ← 0
Count(N) ← 0
For each number
Sum ← Sum(S) + number
Count(N) ← Count + 1
End for
Average ← Sum(S)/Count(N)
Print the Average
Stop
```

Code or Program: It is a set of instructions to solve a problem in a computer language. Usually, the program is stored as binary information in the computer memory. The processor is then able to read, understand and perform the desired operations according to the instructions. This set of instructions is usually called a **Source Program** or **Source Code** or simply a **Program**.

6. Compiling, Debugging and Testing

The process of translation of source program into machine language (**code**) is called **compilation**. This is the first step in testing a program on the computer. During this process the computer checks whenever the program is accurately written, if so, the program is translated into machine language (**code**).

If there are any errors in the source program the process of compilation halts. These errors, may be syntax errors, logical errors, etc., and are called **bugs** in the computer jargon. The process of elimination of these bugs is called **debugging**. **Syntax error** is a mistake that occurs in the formulation of an instruction to a computer. Usually, such errors occur due to mistakes in typing the statements, like missing commas, spelling mistakes, etc. These errors have to be eliminated for successful compilation and subsequent running of the program. The **logical errors** are due to poor design of logic. These errors are hard to detect because the computer generally prints no error message during the execution of the program.

Testing is a process to find any logical error in the program. This must be done systematically. The program must give the correct answer to correct input. If there are any modules in the program, they have to be tested individually for the correct input before put together to make a complete program.

1.3 OVERVIEW OF COMPUTERS AND COMPUTER LANGUAGES

A block diagram of a computer is shown in Fig. 1.3. It consists of five major components, interconnected as shown (Fig. 1.3).

1. Input unit
2. Output unit
3. Memory unit
4. Control unit
5. Arithmetic unit

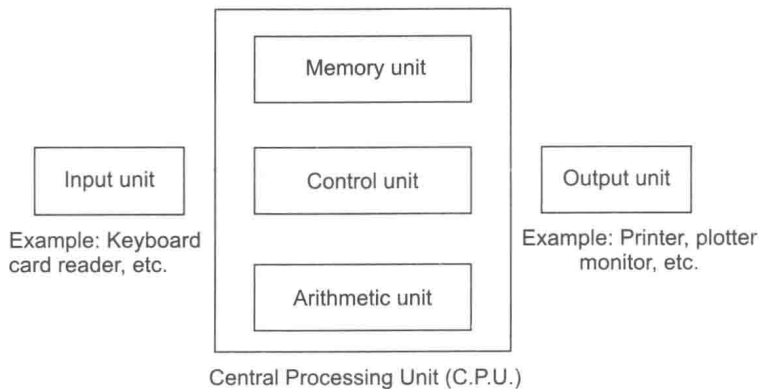


Fig. 1.3 Block diagram of a computer system

1. Input Unit: It consists of devices that permit a computer to receive information (**data**). It may be card readers, magnetic tapes, keyboard, scanner, mouse, digitizer, etc.

2. Output Unit: It consists of devices that permit a computer to display the information (**result**). It may be a printer, plotter, C.R.T. (monitor), etc.

3. Memory Unit: It is a device which stores the data input, the results or output of a program, and the program itself. You can modify these data and retrieve them whenever required. It consists of huge numbers of cells each capable of storing a unit information. To distinguish cells they all numbered sequentially. These numbers are called **addresses**, but for user these addresses are given by names and/or symbols.

4. Control Unit: It is a part of central processing unit. It controls and directs the operations of the entire computer system. The control unit retrieves computer instructions in a proper sequence, interprets each instruction and then directs the other parts of the computer system for implementation. In simple words, it coordinates with the functions of all the units of a computer.

5. Arithmetic Unit: It is also a part of a central processing unit. It performs various arithmetic operations like addition, subtraction, multiplication, etc., and logic operations like operation of values, etc.

The Central Processing Unit (CPU): The three units, viz., memory unit, control unit and arithmetic unit together constitute the central processing unit.

1.4 COMPUTER HARDWARE AND SOFTWARE

1. Hardware

The various physical components of a computer system such as keyboard, mouse, monitor, printer, disk drives, memory boards, etc., constitute the hardware. These hardware components are also

called devices, which are controlled by the programs called device drivers present in the operating systems. For example, **MS-DOS** uses a built-in device driver to control how information is read and written to a floppy disk drive.

Software

The term software is generally used to describe the set of programs that enable the computer hardware to function. There are three basic software categories:

1. Translation programs
2. Operating system programs
3. Application programs

The translation programs (compilers) are programs used by computer system to translate high level languages or problem-oriented language into machine language.

Operating system programs assist in the overall operation of the computer system. They are used to regulate and supervise the sequence of activities going on in the system at any time. More detailed discussion about the OS is made in chapter 2.

Application programs are written by individual users to solve particular problems, such as payroll software, general purpose accounting software, etc.

Storage of Information (Memory Types): In digital computers there are two types of memory units, namely, operational units and **storage units**. A **register** (also called **accumulator**) is used for temporary storage and manipulation of data, which are contained in the **CPU**. Besides storing the data, they also temporarily store program instructions and control information concerning which instruction is to be executed next. Because of their highly specialized nature, registers are expensive relative to storage type memory. This type of memory is popularly called **Random Access Memory (RAM)**.

Read Only Memory (ROM): ROM is used for **permanent storage** of information. It contains the information that CPU needs when you first turn on the computer as well as during the execution this information is stored at the time of manufacture itself by the manufacturer. The information on this can only be read, no new information can be stored or written on to it. **RAM** and **ROM** are together known as **primary storage devices**.

The **storage-type memory** unit is designed to store information which is more permanent in nature. Because the memory units do not have the necessary logic associated with them, the value of the variable stored in the memory unit must be transferred to the register unit, before arithmetic computations involving that variable is performed. When a program is executed, its instructions and data generally reside in storage units. The entire set of storage units in the mainframe is often called **main memory**. In some instances, programs can also reside in storage units which do not belong to the main memory (**secondary memory**), for example, in devices like magnetic disk, magnetic tape, magnetic drum, etc.