GPCET
Pioneering Innovative Education

G. PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Accredited by NAAC with 'A' Grade of UGC, Approved by AICTE, New Delhi

Permanently Affiliated to JNTUA, Ananthapuramu

(Recognized by UGC under 2(f) and 12(B) & ISO 9001:2008 Certified Institution)

Nandikotkur Road, Venkayapalli, Kurnool – 518452

Department of Computer Science and Engineering

Bridge Course
On
Computer Programming

By

P. Rama Rao

GPCET
Pioneering Innovative Education

Table of Contents

Sno	Topic	Page Number
1	Introduction	1
2	Algorithms	2
	2.1 Simple Algorithms	2
	2.2 Type of Algorithms	3
	2.3 Properties of Algorithm	6
3	Flowcharts	6
	3.1 Flowchart Symbols	7
	3.2 General Rules for Flowcharting	8
	3.3 Some examples of Flowcharts	8
	3.4 Advantages Of Using Flowcharts	10
4	C- Language	11
	4.1 A Brief History of C	11
	4.2 C Is a Middle-Level Language	12
	4.3 C Is a Structured Language	12

1 Introduction

Intelligence is one of the key characteristics which differentiate a human being from other living creatures on the earth. Basic intelligence covers day to day problem solving and making strategies to handle different situations which keep arising in day to day life. One person goes Bank to withdraw money. After knowing the balance in his account, he/she decides to with draw the entire amount from his account but he/she has to leave minimum balance in his account. Here deciding about how much amount he/she may with draw from the account is one of the examples of the basic intelligence. During the process of solving any problem, one tries to find the necessary steps to be taken in a sequence. In this Unit you will develop your understanding about problem solving and approaches.

Problem Solving

Can you think of a day in your life which goes without problem solving? Answer to this question is of course, No. In our life we are bound to solve problems. In our day to day activity such as purchasing something from a general store and making payments, depositing fee in school, or withdrawing money from bank account. All these activities involve some kind of problem solving. It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving. To make it clearer, let us see some other examples.

Example1: If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. move to that 2 channel by pressing that channel number on your remote. This is a kind of problem solving.

Example 2: One Monday morning, a student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable. So here picking up books and copies as per timetable is a kind of problem solving.

Example 3: If someone asks to you, what is time now? So seeing time in your watch and telling him is also a kind of problem solving.

Example 4: Some students in a class plan to go on picnic and decide to share the expenses among them. So calculating total expenses and the amount an individual have to give for picnic is also a kind of problem solving.

Now, broadly we can say that problem is a kind of barrier to achieve something and problem solving is a process to get that barrier removed by performing some sequence of activities. Here it is necessary to

mention that all the problems in the world can not be solved. There are some problems which have no solution and these problems are called Open Problems. If you can solve a given problem then you can also write an algorithm for it. In next section we will learn what is an algorithm.

2 Algorithms

Algorithm can be defined as: “A sequence of activities to be processed for getting desired output from a given input.”

Webopedia defines an algorithm as: “A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point”. There may be more than one way to solve a problem, so there may be more than one algorithm for a problem.

Now, if we take definition of algorithm as: “A sequence of activities to be processed for getting desired output from a given input.” Then we can say that:

1. Getting specified output is essential after algorithm is executed.
2. One will get output only if algorithm stops after finite time.
3. Activities in an algorithm to be clearly defined in other words for it to be unambiguous.

Before writing an algorithm for a problem, one should find out what is/are the inputs to the algorithm and what is/are expected output after running the algorithm. Now let us take some exercises to develop an algorithm for some simple problems: While writing algorithms we will use following symbol for different operations:

‘+’ for Addition

‘-’ for Subtraction

‘*’ for Multiplication

‘/’ for Division and

‘ ’ for assignment. For example A X*3 means A will have a value of X*3.

2.1 Simple Algorithms

Problem 1: Find the area of a Circle of radius r.

Inputs to the algorithm:

Radius r of the Circle.

Expected output:

Area of the Circle

Algorithm:

Step1: Read\input the Radius r of the Circle

Step2: Area PI*r*r // calculation of area

Step3: Print Area

Problem2: Write an algorithm to read two numbers and find their sum.

Inputs to the algorithm:

First num1.

Second num2.

Expected output:

Sum of the two numbers.

Algorithm:

Step1: Start

Step2: Read\input the first num1.

Step3: Read\input the second num2.

Step4: Sum num1+num2 // calculation of sum

Step5: Print Sum

Step6: End

Problem 3: Convert temperature Fahrenheit to Celsius

Inputs to the algorithm:

Temperature in Fahrenheit

Expected output:

Temperature in Celsius

Algorithm:

Step1: Start

Step 2: Read Temperature in Fahrenheit F

Step 3: C = 5/9*(F32)

Step 4: Print Temperature in Celsius: C

Step 5: End

2.2 Type of Algorithms

The algorithm and flowchart, classification to the three types of control structures. They are:

1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

These three control structures are sufficient for all purposes. The sequence is exemplified by sequence of statements place one after the other – the one above or before another gets executed first. In flowcharts, sequence of statements is usually contained in the rectangular process box.

The branch refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken. This is usually represented by the 'if-then' construct in pseudo-codes and programs. In flowcharts, this is represented by the diamond-shaped decision box. This structure is also known as the selection structure.

Problem1: write algorithm to find the greater number between two numbers

Step1: Start
Step2: Read/input A and B
Step3: If A greater than B then C=A
Step4: if B greater than A then C=B
Step5: Print C
Step6: End

Problem 2: A algorithm to find the largest value of any three numbers.

Step1: Start
Step2: Read/input A,B and C
Step3: If (A>=B) and (A>=C) then Max=A
Step4: If (B>=A) and (B>=C) then Max=B
Step5:If (C>=A) and (C>=B) then Max=C
Step6: Print Max
Step7: End

The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the 'while' and 'for' constructs in most programming languages, for unbounded loops and bounded loops respectively. (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied; bounded loops refer to those whose number of iterations is known before-hand.) In the flowcharts, a back arrow hints the presence of a loop. A trip around the loop is known as iteration. You must ensure that the condition for

the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the repetition structure.

Examples:

Problem1: An algorithm to calculate even numbers between 0 and 99

1. Start
2. I ← 0
3. Write I in standard output
4. I ← I+2
5. If (I <=98) then go to line 3
6. End

Problem2: Design an algorithm which gets a natural value, n,as its input and calculates odd numbers equal or less than n. Then write them in the standard output:

1. Start
2. Read n
3. I ← 1
4. Write I
5. I ← I + 2
6. If (I <= n) then go to line 4
7. End

Problem3: Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

1. Start
2. I ← 1000 and S ← 0
3. Write I
4. S ← S + I
5. I ← I + 2
6. If (I <= 2000) then go to line 3 else go to line 7
7. Write S
8. End

Problem4: Design an algorithm with a natural number, n, as its input which calculates the following formula and writes the result in the standard output:

S = ½ + ¼ + … +1/n

1. Start
2. Read n
3. I ← 2 and S ← 0
4. S= S + 1/I
5. I ← I + 2
6. If (I <= n) then go to line 4 else write S in standard output
7. End

Combining the use of these control structures, for example, a loop within a loop (nested loops), a branch within another branch (nested if), a branch within a loop, a loop within a branch, and so forth, is not uncommon. Complex algorithms may have more complicated logic structure and deep level of nesting, in which case it is best to demarcate parts of the algorithm as separate smaller modules. Beginners must train themselves to be proficient in using and combining control structures appropriately, and go through the trouble of tracing through the algorithm before they convert it into code.

2.3 Properties of Algorithm

Donald Ervin Knuth has given a list of five properties for an algorithm, these properties are:

1) Finiteness: An algorithm must always terminate after a finite number of steps. It means after every step one reach closer to solution of the problem and after a finite number of steps algorithm reaches to an end point.

2) Definiteness: Each step of an algorithm must be precisely defined. It is done by well thought actions to be performed at each step of the algorithm. Also the actions are defined unambiguously for each activity in the algorithm.

3) Input: Any operation you perform need some beginning value/quantities associated with different activities in the operation. So the value/quantities are given to the algorithm before it begins.

4) Output: One always expects output/result (expected value/quantities) in terms of output from an algorithm. The result may be obtained at different stages of the algorithm. If some result is from the intermediate stage of the operation then it is known as intermediate result and result obtained at the end of algorithm is known as end result. The output is expected value/quantities always have a specified relation to the inputs.

5) Effectiveness: Algorithms to be developed/written using basic operations. Actually operations should be basic, so that even they can in principle be done exactly and in a finite amount of time by a person, by using paper and pencil only.

3 Flowcharts

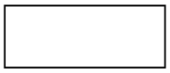
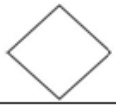
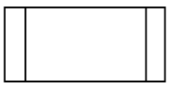
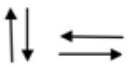
The flowchart is a diagram which visually presents the flow of data through processing systems. This means by seeing a flow chart one can know the operations performed and the sequence of these

operations in a system. Algorithms are nothing but sequence of steps for solving problems. So a flow chart can be used for representing an algorithm. A flowchart, will describe the operations (and in what sequence) are required to solve a given problem. You can see a flow chart as a blueprint of a design you have made for solving a problem.

For example suppose you are going for a picnic with your friends then you plan for the activities you will do there. If you have a plan of activities then you know clearly when you will do what activity. Similarly when you have a problem to solve using computer or in other word you need to write a computer program for a problem then it will be good to draw a flowchart prior to writing a computer program. Flowchart is drawn according to defined rules.

3.1 Flowchart Symbols

There are 6 basic symbols commonly used in flowcharting of assembly language Programs: Terminal, Process, input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols, it is the ones used most often in the structure of Assembly language programming.

Symbol	Name	Function
	**Process**	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

3.2 General Rules for Flowcharting

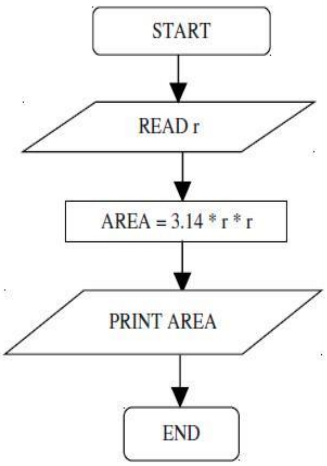
1. All boxes of the flowchart are connected with Arrows. (Not lines)
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
4. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
5. Connectors are used to connect breaks in the flowchart. Examples are:
    - From one page to another page.
    - From the bottom of the page to the top of the same page.
    - An upward flow of more then 3 symbols
6. Subroutines and Interrupt programs have their own and independent flowcharts.
7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
8. All flowcharts end with a terminal or a contentious loop.

Flowcharting uses symbols that have been in use for a number of years to represent the type of operations and/or processes being performed. The standardized format provides a common method for people to visualize problems together in the same manner. The use of standardized symbols makes the flow charts easier to interpret, however, standardizing symbols is not as important as the sequence of activities that make up the process.

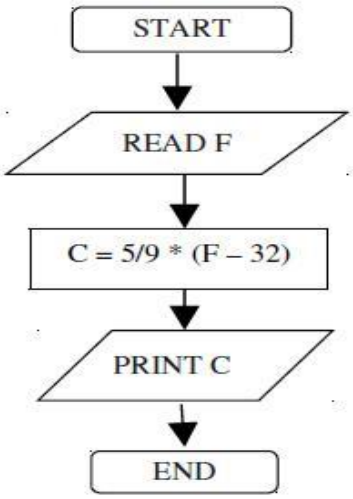
3.3 Some examples of Flowcharts

Now, we will discuss some examples on flowcharting. These examples will help in proper understanding of flowcharting technique. This will help you in program development process in next unit of this block.

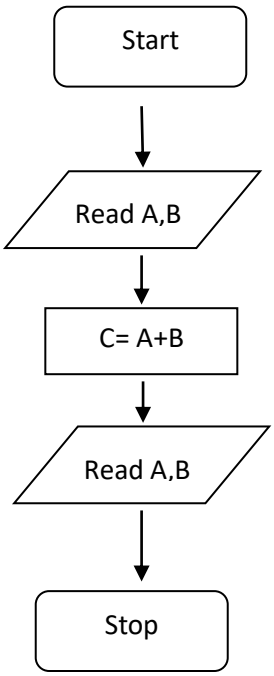
Problem1: Find the area of a circle of radius r.



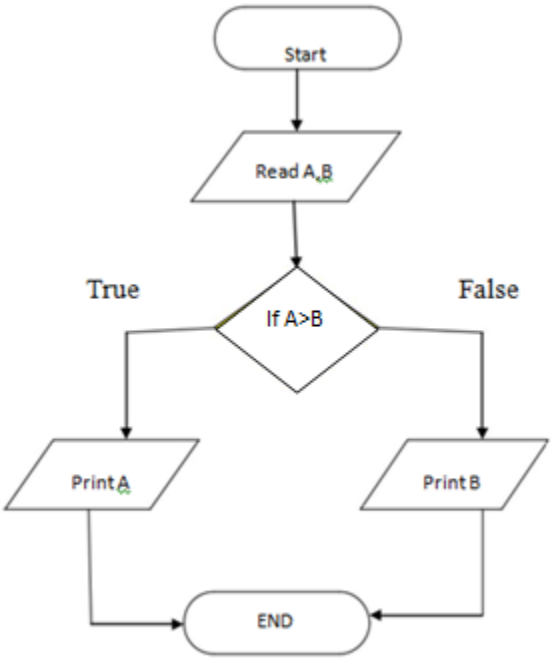
Problem 2: Convert temperature Fahrenheit to Celsius.



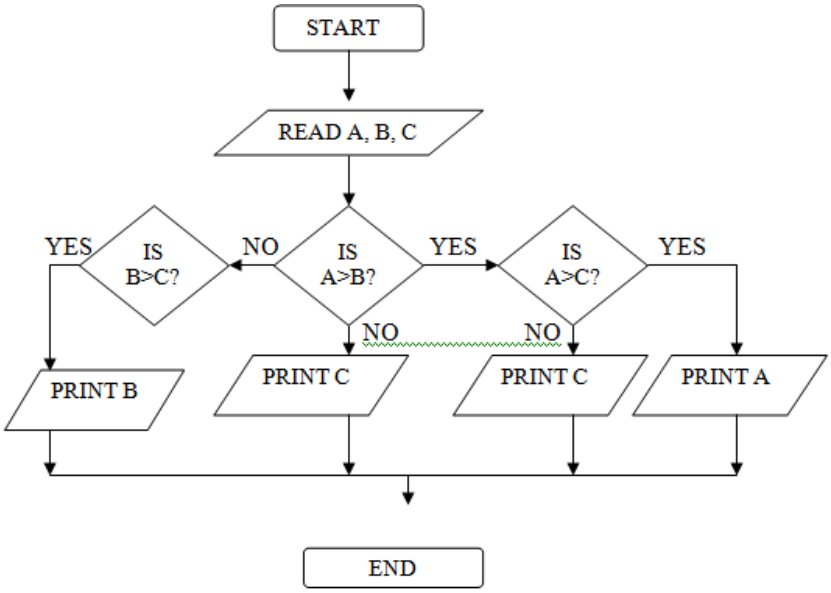
***Problem3*: Flowchart for an algorithm which gets two numbers and prints sum of their value.**



***Problem 4*: Algorithm for find the greater number between two numbers.**



***Problem 5*: Draw a flowchart to find the largest of three numbers A,B, and C.**



3.4 Advantages Of Using Flowcharts

Flow chart is used for representing algorithm in pictorial form. This pictorial representation of a solution/system is having many advantages. These advantages are as follows:

1) **Communication:** A Flowchart can be used as a better way of communication of the logic of a system and steps involve in the solution, to all concerned particularly to the client of system.

2) **Effective analysis:** A flowchart of a problem can be used for effective analysis of the problem.

3) **Documentation of Program/System:** Program flowcharts are a vital part of a good program documentation. Program document is used for various purposes like knowing the components in the program, complexity of the program etc.

4) **Efficient Program Maintenance:** Once a program is developed and becomes operational it needs time to time maintenance. With help of flowchart maintenance become easier.

5) **Coding of the Program:** Any design of solution of a problem is finally converted into computer program. Writing code referring the flowchart of the solution become easy.

4 C- Language

4.1 A Brief History of C

C was invented and first implemented by Dennis Ritchie on a DEC PDP-11 that used the Unix operating system. C is the result of a development process that started with an older language called BCPL. BCPL was developed by Martin Richards, and it influenced a language called B, which was invented by Ken Thompson. B led to the development of C in the 1970s. For many years, the de facto standard for C was the version supplied with the Unix operating system. It was first described in *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Englewood Cliffs, N.J.: Prentice-Hall, 1978). In the summer of 1983 a committee was established to create an ANSI (American National Standards Institute) standard that would define the C language. The standardization process took six years (much longer than anyone reasonably expected).

The ANSI C standard was finally adopted in December 1989, with the first copies becoming available in early 1990. The standard was also adopted by ISO (International Standards Organization), and the resulting standard was typically referred to as ANSI/ISO Standard C. In 1995, Amendment 1 to the C standard was adopted, which, among other things, added several new library functions. The 1989 standard for C, along with Amendment 1, became a *base document* for Standard C++, defining the *C subset* of C++. The version of C defined by the 1989 standard is commonly referred to as C89.

During the 1990s, the development of the C++ standard consumed most programmers' attention. However, work on C continued quietly along, with a new standard for C being developed. The end result was the 1999 standard for C, usually referred to as C99. In general, C99 retained nearly all of the features of C89. Thus, C is still C! The C99 standardization committee focused on two main areas: the addition of several numeric libraries and the development of some special-use, but highly innovative, new features, such as variable-length arrays and the **restrict** pointer qualifier. These innovations have once again put C at the forefront of computer language development.

4.2 C Is a Middle-Level Language

C is often called a *middle-level* computer language. This does not mean that C is less powerful, harder to use, or less developed than a high-level language such as BASIC or Pascal, nor does it imply that C has the cumbersome nature of assembly language (and its associated troubles). Rather, C is thought of as a middle-level language because it combines the best elements of high-level languages with the control and flexibility of assembly language As a middle-level language, C allows the manipulation of bits, bytes, and addresses— the basic elements with which the computer functions. Despite this fact, C code is also very portable. *Portability* means that it is easy to adapt software written for one type of computer or operating system to another type. For example, if you can easily convert a program written for DOS so that it runs under Windows 2000, that program is portable.

High level

1. Ada
2. Modula-2
3. Pascal
4. COBOL
5. FORTRAN
6. BASIC
7. Java

Middle level

1. C++
2. C
3. FORTH
4. Macro-assembler

4.3 C Is a Structured Language

The term *block-structured* applied to a computer language. Although the term block-structured language does not strictly apply to C, C is commonly referred to simply as a *structured* language. It has many similarities to other structured languages, such as ALGOL, Pascal, and Modula-2. A structured language offers a variety of programming possibilities. For example, structured languages typically support several loop constructs, such as **while**, **do-while**, and **for**.

In a structured language, the use of **goto** is either prohibited or discouraged and is not the common form of program control (as is the case in standard BASIC and traditional FORTRAN, for example). A structured language allows you to place statements anywhere on a line and does not require a strict field concept (as some older FORTRANs do).

Here are some examples of structured and nonstructured languages:

Nonstructured

1. FORTRAN
2. BASIC
3. COBOL
4. C
5. Java
6. Modula-2

Structured

1. Pascal
2. Ada
3. C++