Textbook of

COMPUTER SCIENCE

GRADE 10

National Book Foundation
as
Federal Textbook Board
Islamabad

Textbook of

COMPUTER SCIENCE

Grade

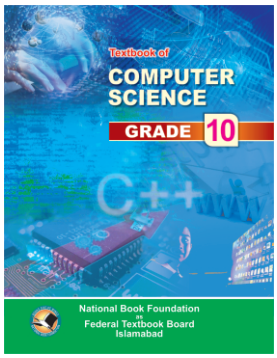
10

National Book Foundation
as
Federal Textbook Board
Islamabad

OUR MOTTO

• Standards • Outcomes • Access • Style

Textbook of
Computer Science Grade -10



Authors	:	Muhammad Sajjad Heder Muhammad Khalid
Designing	:	Hafiz Rafiuddin (Late), Shahzad Ahmed
Desk Officer, NCC	:	Dr. Zulfiqar Ali Cheema, CD & TP Wing
Management	:	Ishtiaq Ahmad Malik, Secretary NBF
Incharge Textbooks	:	Muhammad Rafique, Assistant Director, NBF
New Developed Edition	:	2018 Qty. 42000
3rd Print	:	March 2019 Qty. 40,000
4th Print	:	February 2020 Qty. 45,000
Price	:	Rs. 160/-
Code	:	STE-573
ISBN	:	978-969-37-1092-2
Printer	:	Ishaq al Fateh Printers, lahore

for Information about other National Book Foundation Publications,
visit our Web site **http://www.nbf.org.pk** , phone: **92-51-9261124, 9261125**
Email: **nbftextbooks@gmail.com** / **books@nbf.org.pk**

PREFACE

The textbook of **COMPUTER SCIENCE for GRADE - 10** has been developed according to the National Curriculum 2009. This book consists of seven chapters. It is based on C Programming Language, Computer Logic and Gates, and HTML.

Computers are changing our world and driving our economy. We are living in a world controlled by computer software. Students must learn how to design and write computer programs to solve problems related with any field. No matter what students' future plan may be, learning computer programming is essential for them. It enables them to succeed in the technology-driven world of 21st century. Programming allows putting ideas into reality in any industry. It provides skills about how to make use of programming language to solve various computational problems. There will be high demand of programmers in tomorrow's world to build useful applications and websites.

This textbook is developed with due care using simple language, supporting pictures to make it understandable by secondary school students. Keen efforts have been made to minimize the errors and omissions. This will surely enhance the confidence and comprehension of the students and general readers in the subject. Yet there is always room for improvement and any suggestion with regard to the quality of work will be warmly welcomed at our end.

Quality of Standards, Pedagogical Outcomes, Taxonomy Access and Actualization of Style is our motto. With these elaborations, this series of new development was presented for use. After educational feedback and necessary changes, the book is being published again.

National Book Foundation

TABLE OF CONTENTS

1

PROGRAMMING TECHNIQUES

After completing this lesson, you will be able to:

- Define and analyze a problem
- Plan the solution of a problem
- Define candid solutions of a problem and select the best solution
- Define algorithm
- Describe role of algorithm in problem solving
- Describe the criteria for measuring efficiency of an algorithm
- Write algorithms for solving various mathematical problems
- Define flowchart
- Describe the importance of flowchart
- Determine the flowchart requirements of a given problem
- Understand the use of flowchart symbols
- Draw flowcharts for solving various mathematical problems

Reading

UNIT INTRODUCTION

A computer is a general-purpose electronic machine invented to help people solve various problems. Computer must be programmed by human beings to perform various tasks. Various programming techniques are used for solving problems on computer. This unit explains how to write algorithms and draw flowcharts which are essential to develop a computer program.

1.1 UNDERSTANDING THE PROBLEM

Solving problems is the main task of computer science which is the job of computer programmer. Programmer must first understand how a human solves a problem then translate it into a set of instructions in a programming language that a computer can understand and execute.

The following five steps are involved in problem solving on the computer.

1.1.1 DEFINING THE PROBLEM

Defining the problem is initial stage of problem solving. It is very important to understand the problem before the programmer starts working on its solution. The following are the steps to properly define and understand the problem.

- Carefully read the problem to understand what it tells.
- Find out what the problem asks to do.
- What information can be obtained from the problem?
- What is required to be calculated as the solution of the problem?

For example,

Problem 1: A person is unhappy with a product or service he/she has purchased from a company.

After reading the above statement, we understand that a customer has purchased any product or service from a Company and now the customer is not satisfied with it. This statement defines the problem clearly and requires the solution. It is a general type of problem.

Problem 2: Finding average marks of a student.

It is clear from the problem statement that average marks of a student have to be found. This is numeric problem.

1.1.2 ANALYZING THE PROBLEM

At this stage of problem solving, the programmer investigates the problem and gathers as much information as possible to find a solution. The following questions are to be asked to analyze the problem.

- Is it possible to solve the problem on a computer?
- What is to be done to find the solution of the problem?
- What is the proper sequence of steps to solve the problem?
- What are the inputs and what output is required?
- How many solutions are possible?
- Which solution is the best and why?
- How solution will be implemented?

To analyze the **Problem 1** (given above), the following points are to be kept in mind and some information is to be gathered.

Know the outcome you want:

- Think about what is important to you and what you want to achieve. For example, do you want a refund, a replacement?
- Would you like them to provide the service again, this time meeting agreed standards?
- The more specific you are the more likely it is that you will get the outcome you want.

Prepare for the discussion:

- Find out about your legal rights.
- Write down exactly what your complaint is, including details such as dates and times.
- Gather any relevant paperwork, e.g. advertisement that misled you, or a quote that the trader gave you.

Find out what the organisation's complaint process is and follow this:

- Have a look on their website or on any documentation they have given you.

To analyze **Problem 2** the following information is required.

- What are the subject marks of the student in each subject?
- How many subjects are there?
- What is the formula to find the average marks?

1.1.3 PLANNING THE SOLUTION OF THE PROBLEM

Planning the solution of the problem is a creative stage of problem solving. It refers to dividing the solution into steps and arranging them into proper order that will solve the problem.

To plan the solution of the **Problem 1** (given above), after analyzing the problem, the following planning is required:

How to discuss a problem:

- Talk to the right person. The person you speak must have the ability to resolve the issue, e.g. a store manager, business owner or supervisor.
- Focus on talking about the problem with the product or service, rather than taking issue with a person.
- Stay calm and reasonable. Explain the problem in detail and provide any evidence you may have.
- Tell them what outcome you want. It is the store or service provider's responsibility to resolve the problem, but it can be helpful to ask them for a specific solution.
- Expect questions. A store or service provider may ask you for more details.
- Ask to speak with someone else if you are not happy with the way the conversation is going. It is okay to walk away and come back later, or to follow up in writing.
- Listen to their response. Ask for time to consider it if you need to.

Consider their response:

- Does the response resolve your issue or is it a reasonable compromise?
- Do you want to take the time and effort to carry on if the response is unsatisfactory?
- Based on what you know of your rights, do you think you have a solid basis to take the matter further?

To solve the **Problem 2**, the following planning is required:

- All subject marks are to be added together to find the sum of all the marks.
- The sum is to be divided by the total number of subjects to find the average marks.

1.1.4 CANDID SOLUTIONS OF A PROBLEM

All the possible solutions of a problem that produce correct result are known as candid solutions. To find candid solutions of a problem, programmer has to look for different methods to solve the problem and come up with several solutions.

For example, many solutions can be considered to resolve the **Problem 1**.

Solution 1: The customer can lodge the complaint personally visiting the company.

Solution 2: The customer can also use the courier services to send the product back.

Solution 3: The customer can use company's website to file the complaint.

1.1.5 SELECT THE BEST SOLUTION

After finding the candid solutions, only one solution can be selected. The selection of final solution of a problem should be based on the following criteria.

Speed: The selected solution of the problem should be efficient. In other words, it means when the solution is implemented in a programming language, the program should run fast.

Cost: The selected solution of the problem should provide a cost-effective way of implementation.

Complexity: The selected solution of the problem should not be complicated. It should contain minimum number of instructions/simple steps.

1.2 ALGORITHM

1.2.1 THE ALGORITHM

Algorithm means method, procedure, technique or plan. Algorithm is a step-by-step problem solving method that is easy to understand and follow. It is a set of steps that clearly defines a sequence of operations to solve a problem.

1.2.2 ROLE OF ALGORITHM IN PROBLEM SOLVING

Algorithm plays an important role in computer programming. Computer programming is the process of taking an algorithm and coding it in a programming language. Formulating an algorithm is the first step for developing a computer program.

1.2.3 MEASURING EFFICIENCY OF AN ALGORITHM

The efficiency of an algorithm is the property which relate the algorithm to the amount of computational resources used in it. An algorithm must be analyzed to determine its resource usage

(e.g. time, memory and storage space). For maximum efficiency we wish to minimize resource usage. However, the various resources cannot be compared directly, so which of two algorithms is considered to be more efficient often depends on which measure of efficiency is considered the most important, e.g. the requirement for high speed, minimum memory usage is considered.

There may be several algorithms for solving a particular problem. How fast a problem can be solved, depends on efficient algorithm. Therefore, analysis of algorithms is very important to compare algorithms and conclude that which one is better than the other. Analyzing the efficiency of algorithms means comparing efficiency of different methods of solutions of a problem.

1.2.4 ALGORITHMS FOR SOLVING PROBLEMS

This section presents algorithms to solve various mathematical problems. This develops skills in finding solutions of problems.

Problem 1: Find the sum, product and average of five given numbers.

Planning the Solution:

Input: Five given numbers
Required Output: Sum, product and average of five numbers
Processing: Addition, multiplication and division of numbers

Algorithm:

Step 1: Start
    Let the five numbers be A=2, B=5, C=8, D=4 and E=12
Step 2: FIND the sum (SUM)
        SUM=A+B+C+D+E
Step 3: FIND the product (PROD)
        PROD=A*B*C*D*E
Step 4: FIND the average (AVG)
        AVG=SUM/5
Step 5: Output SUM, PROD, AVG
Step 6: Stop

Problem 2: Find the largest of three unequal numbers.

Planning the Solution:

Input: Three unequal numbers
Required Output: The largest of three numbers
Processing: Comparison of each number with the other two one by one

Algorithm:

Step 1: Start
    Let the three numbers be A=10, B=20 and C=30
Step 2: Check if A is the largest number
    IF A>B and A>C THEN LARGEST=A otherwise GOTO Step 4
Step 3: GOTO Step 7

Step 4: Check if B is the largest number
IF B>A and B>C THEN LARGEST=B otherwise GOTO Step 6
Step 5: GOTO Step 7
Step 6: LARGEST=C
Step 7: Output LARGEST
Step 8: Stop

Problem 3: Find acceleration of a moving object for given mass and the force applied.

Planning the Solution:

Input: Mass and the Force
Required Output: Acceleration of moving object
Processing: Divide the force by the mass

Algorithm:

Step 1: Start
Let the mass (M) be 50 and the force (F) be 12
Step 2: CALCULATE the acceleration (A)
A=F/M
Step 3: Output A
Step 4: Stop

Problem 4: Find the volume of a cube.

Planning the Solution:

Input: Length of side of cube
Required Output: Volume of cube
Processing: Side of cube raised to the power of three

Algorithm:

Step 1: Start
Let the length of one side of cube, L be 12
Step 2: CALCULATE the volume, V
$V=L^3$
Step 3: Output V
Step 4: Stop

Problem 5: Find the area of a triangle when the lengths of height and base are given.

Planning the Solution:

Input: Height and base of triangle
Required Output: Area of triangle
Processing: Multiply height with base and divide by two

Algorithm:

Step 1: Start

Let the height, H be 10 and the base, B be 15

Step 2: CALCULATE the area, A

A = (B*H)/2

Step 3: Output A

Step 4: Stop

Problem 6: Read marks (M) and print letter grade according to the following scheme.

Marks	Letter Grade
M≥80 and M≤100	A1
M≥70 and M<80	A
M≥60 and M<70	B
M≥50 and M<60	C
M≥40 and M<50	D
M≥33 and M<40	E
M≥0 and M<33	Ungraded

Planning the Solution:

Input: Marks

Required Output: Letter Grade

Processing: Check in which range the marks fall and accordingly print the letter grade.

Algorithm:

Step 1: Start

Let the marks, be M

Step 2: Compare the marks

IF M≥80 THEN print “Grade A1” otherwise GOTO Step 4

Step 3: GOTO Step 13

Step 4: Compare the marks

IF M≥70 THEN print “Grade A” otherwise GOTO Step 6

Step 5: GOTO Step 13

Step 6: Compare the marks

IF M≥60 THEN print “Grade B” otherwise GOTO Step 8

Step 7: GOTO Step 13

Step 8: Compare the marks

IF M≥50 THEN print “Grade C” otherwise GOTO Step 10

Step 9: GOTO Step 13

Step 10: Compare the marks

IF M≥40 THEN print “Grade D” otherwise GOTO Step 12

Step 11: GOTO Step 13

Step 12: Compare the marks

IF M≥33 THEN print “Grade E” otherwise print “Ungraded”

Step 13: Stop

Problem 7: Find the exponent of a given number.

Exponent or power of a number means how many times to use the number in a multiplication. In other words it is the product of a number that is multiplied as many times as its exponent.

Planning the Solution:

Input: A number and its exponent

Required Output: Exponent of given number

Processing: Multiply the number as many times as its exponent

Algorithm:

Step 1: Start

Let the number, N be 8 and its exponent, E be 5

Step 2: Initialize product(P) and K to 1

P=1, K=1

Step 3: FIND the product(P)

P=P*N

Step 4: Increment K by 1

K=K+1

Step 5: Check if the value of K is less than or equal to E

IF K≤E THEN GOTO Step 3 otherwise GOTO Step 6

Step 6: Output P

Step 7: Stop

Problem 8: Print odd numbers from 1 to 100.

1 3 5 7 9 11 . . . 99

Planning the Solution:

Input: This problem has no input

Required Output: Printing odd numbers from 1 to 100

Processing: Initialize a variable to 1 and keep printing it with an increment of 2 till 99

Algorithm:

Step 1: Start

Initialize variable K to 1

K=1

Step 2: Output K

Step 3: Increment K by 2

K=K+2

Step 4: Check if the value of K is less than 100

IF K<100 THEN GOTO Step 2 otherwise GOTO Step 5

Step 5: Stop

Problem 9: Print the following sequence of numbers in descending order.

27 24 21 18 15 12 9 6 3 0 -3 -6

Planning the Solution:

Input: This problem has no input

Required Output: Printing numbers from 27 to -6 in descending order with a step of -3

Processing: Initialize a variable to 27 and then keep printing it with a decrement of 3 till -6

Algorithm:

Step 1: Start

Initialize variable K to 27

K=27

Step 2: Output K

Step 3: Decrement K by 3

K=K-3

Step 4: Check if the value of K is greater than or equal to -6

IF K≥-6 THEN GOTO Step 2 otherwise GOTO Step 5

Step 5: Stop

Problem 10: Find the sum of even numbers up to 100.

SUM = 2 + 4 + 6 + 8 + 10 + 12 + 14 + . . . + 100

Planning the Solution:

Input: This problem has no input

Required Output: Sum of even numbers up to 100

Processing: Add all the even numbers from 2 to 100

Algorithm:

Step 1: Start

Initialize SUM to 0 and K to 2

SUM=0, K=2

Step 2: ADD K to SUM

SUM=SUM+K

Step 3: Increment K by 2

K=K+2

Step 4: Check if the value of K is less than or equal to 100

IF K≤100 THEN GOTO Step 2 otherwise GOTO Step 5

Step 5: Output SUM

Step 6: Stop

Problem 11: Print a multiplication table of a given number.

Planning the Solution:

Input: Number whose table is required

Required Output: Printing table of given number

Processing: Initialize a variable to 1 and print its product with the given number. Increment the variable by 1 and print the product. Continue the process till printing the product of 12 and the given number.

Algorithm:

Step 1: Start

Let the number, N be 7

Step 2: Initialize K to 1

K=1

Step 3: FIND the product

P=N*K

Step 4: Output N, K, P

Step 5: Increment K by 1

K=K+1

Step 6: Check the value of K

IF K≤12 THEN GOTO Step 3 otherwise GOTO Step 7

Step 7: Stop

Problem 12: Convert temperature from Fahrenheit to Celsius.

Planning the Solution:

Input: Temperature in Fahrenheit

Required Output: Temperature in Celsius

Processing: Compute the temperature in Celsius from Fahrenheit using conversion formula

Algorithm:

Step 1: Start

Let the temperature in Fahrenheit, F be 100

Step 2: CALCULATE temperature in Celsius (C)

C = 5/9(F-32)

Step 3: Output C

Step 4: Stop

Problem 13: Find factorial of a given number.

Planning the Solution:

Input: Given Number

Required Output: Factorial of given number

Processing: Find the product of all the numbers from 1 to the given number.

Algorithm:

Step 1: Start

Let the number, N be 5

Step 2: Initialize loop variable K and factorial F to 1

K=1, F=1

Step 3: CALCULATE the product

F=F*K

Step 4: Increment K by 1

K=K+1

Step 5: Check the value of K

IF K≤N THEN GOTO Step 3 otherwise GOTO Step 6

Step 6: Output F

Step 7: Stop

1.3 FLOWCHART

1.3.1 THE FLOWCHART

Flowchart is a diagrammatic representation of algorithm. It describes what operations are required to solve a given problem.

1.3.2 IMPORTANCE OF FLOWCHART IN SOLVING A PROBLEM

Flowchart illustrates the sequence of operations to be performed to solve a problem in the form of a diagram. Computer programmers draw flowcharts before writing computer programs. It provides an easy way to analyze and find solutions of problems. Once, the flowchart is drawn, it becomes very easy to write the program in any high level language. It is very helpful in communicating the problem solving method to other people. It also helps in finding and removing errors in computer programs.

1.3.3 STEPS FOR DRAWING FLOWCHART

The flowchart developer must determine the following requirements for the given problem or algorithm before drawing a flowchart.

- Input to the flowchart
- Type of processing required
- Decisions to be taken
- The output to be produced after processing

Input: The flowchart developer must know what exactly the input to the flowchart is. The input is determined from the problem statement. For example, the given problem is to convert temperature from Fahrenheit to Celsius. Here, the input will be the temperature in Fahrenheit.

Processing: The flowchart developer must decide what type of calculation is to be performed or which formula is to be applied to obtain the required result. For example, to find the area of a triangle, the following formula is to be used.

$$Area = (Base * Height)/2$$

Decision: The flowchart developer must decide which control structures (sequence, repetition or selection) are to be applied for the solution of the problem. For example, selection structure must be applied to print letter grade of a student based on the marks obtained. The selection structure will check in which range the marks fall and accordingly print the grade.

Output: The flowchart must provide the required output.

1.3.4 FLOWCHART SYMBOLS

Flowcharts are drawn using standard symbols. These symbols have specific meaning and are connected by arrows indicating the flow from one step to another. Flowchart symbols are illustrated in Table 1-1.

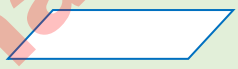
Symbol Description	Symbol Shape	Symbol Description	Symbol Shape	
Flow Line		Process		
Start/Stop		Decision		
Input/Output		Connector	 On-Page	 Off-Page

Table 1-1 Flowchart symbols

Flow Line: It is a line with arrow head used to connect various flowchart symbols and indicates the flow of control in the flowchart.

Start/Stop Symbol: It is a rounded rectangular shaped symbol. It is used to indicate the start or end of a flowchart. We can only write the words Start or Stop inside this symbol. A flowchart can only have one start but it may have many ends.

Input/Output Symbol: Parallelogram represents input or output operations in a flowchart. It contains the word READ or INPUT along with the variables for input operation or PRINT or OUTPUT along with the output data for output operation.

Process Symbol: A rectangular block is used for any data processing operation. All the calculations appear inside the processing symbol, such as "SUM= A + B". Variables are also initialized inside the process symbol, such as “K=1”.

Decision Symbol: A diamond shaped symbol represents decision in a flowchart and it contains a condition. If the condition is true, the path marked YES is to be followed. If the condition is false, the path marked NO is to be followed. The words TRUE or FALSE can also be used instead of YES or NO.

Connector Symbols: These symbols are used to connect one part of a flowchart to the other on the same page (On-Page connector) or on the new page (Off-Page connector).

1.3.5 FLOWCHARTS TO SOLVE PROBLEMS

In this section algorithms presented in the previous section will be converted into flowcharts for better understanding of the logic of problem solving.

Flowchart 1: Flowchart to find the sum, product and average of five numbers

Before drawing the flowchart to solve this problem, the input to the flowchart, type of calculation to be performed and the required output must be defined. From the statement of the flowchart, it is clear that sum, product and average are to be calculated and given are five numbers.

Flowchart to find sum, product and average of five numbers is shown in Fig.1-1.

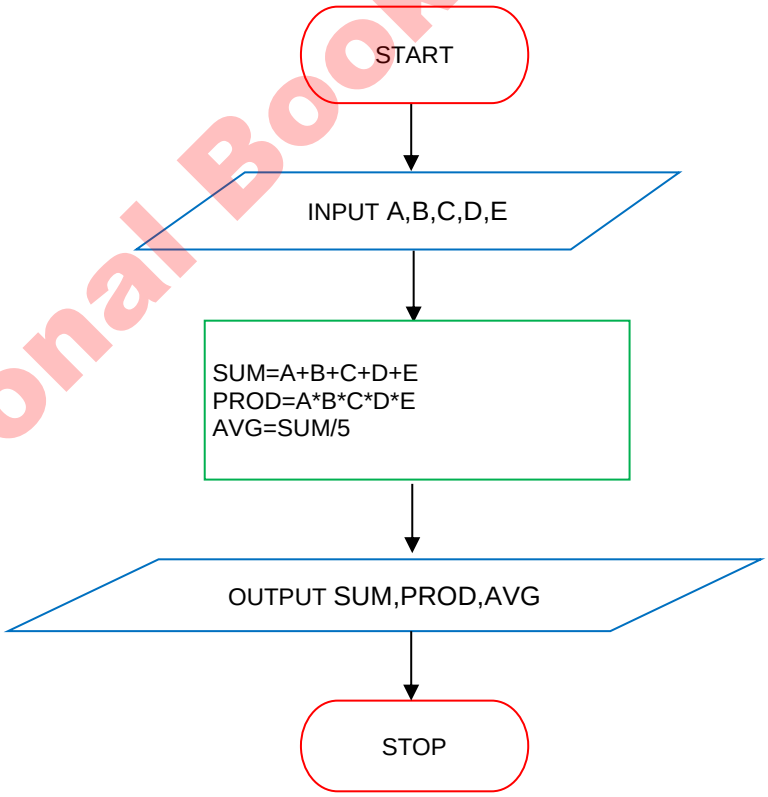


Fig.1-1 Flowchart to find sum, product and average

Flowchart 2: Flowchart to find the largest of three unequal numbers

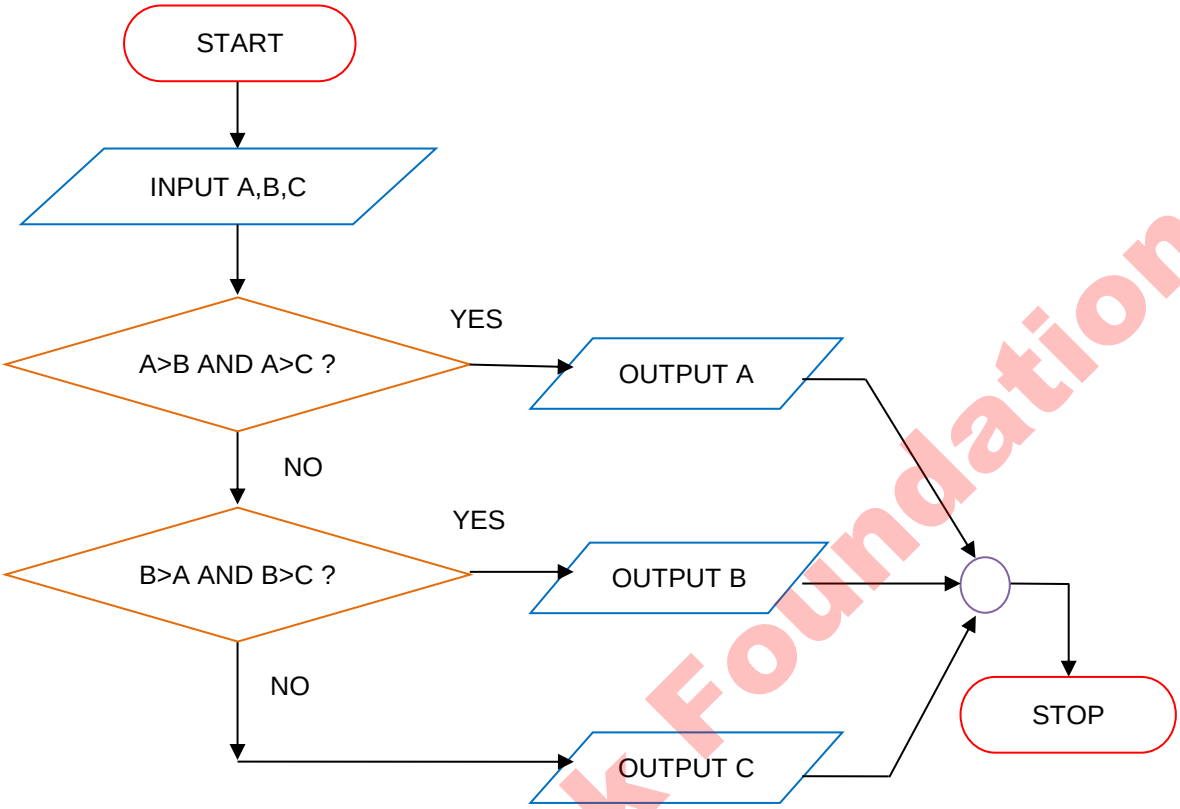


Fig.1-2 Flowchart to find largest of three numbers

Flowchart 3: Flowchart to find acceleration of a moving object given mass and the force applied

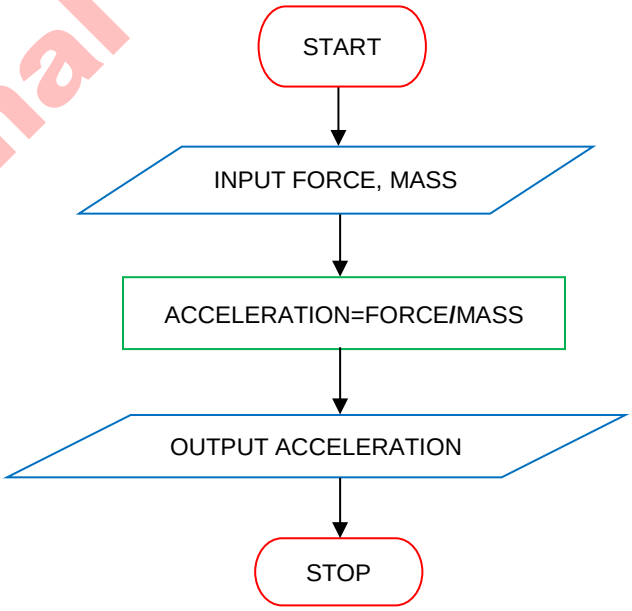


Fig.1-3 Flowchart to find acceleration

Flowchart 4: Flowchart to input the length of one side of cube and print its volume

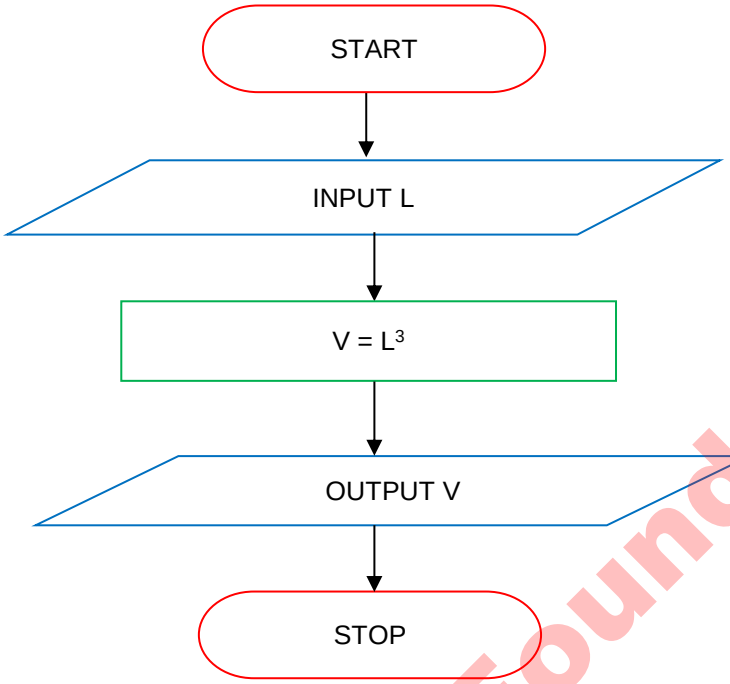


Fig.1-4 Flowchart to find volume of a cube

Flowchart 5: Flowchart to find the area of a triangle when the lengths of height and base are given

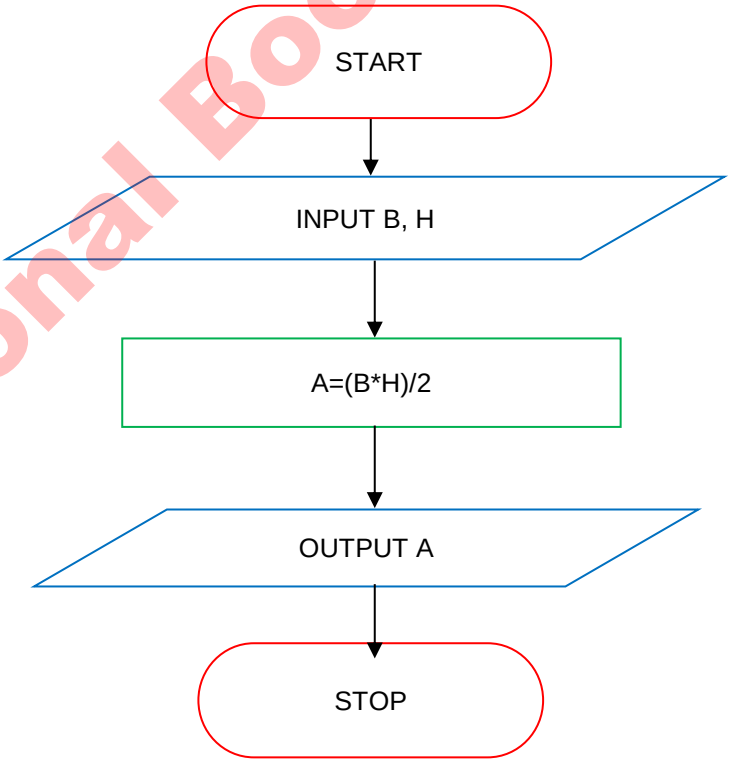


Fig.1-5 Flowchart to find area of a triangle

Flowchart 6: Flowchart to read marks (M) and print letter grade according to the following criteria.

Marks	Letter Grade
M≥80 and M≤100	A1
M≥70 and M<80	A
M≥60 and M<70	B
M≥50 and M<60	C
M≥40 and M<50	D
M≥33 and M<40	E
M<33	Ungraded

Flowchart to print letter grade is shown in Fig. 1-6.

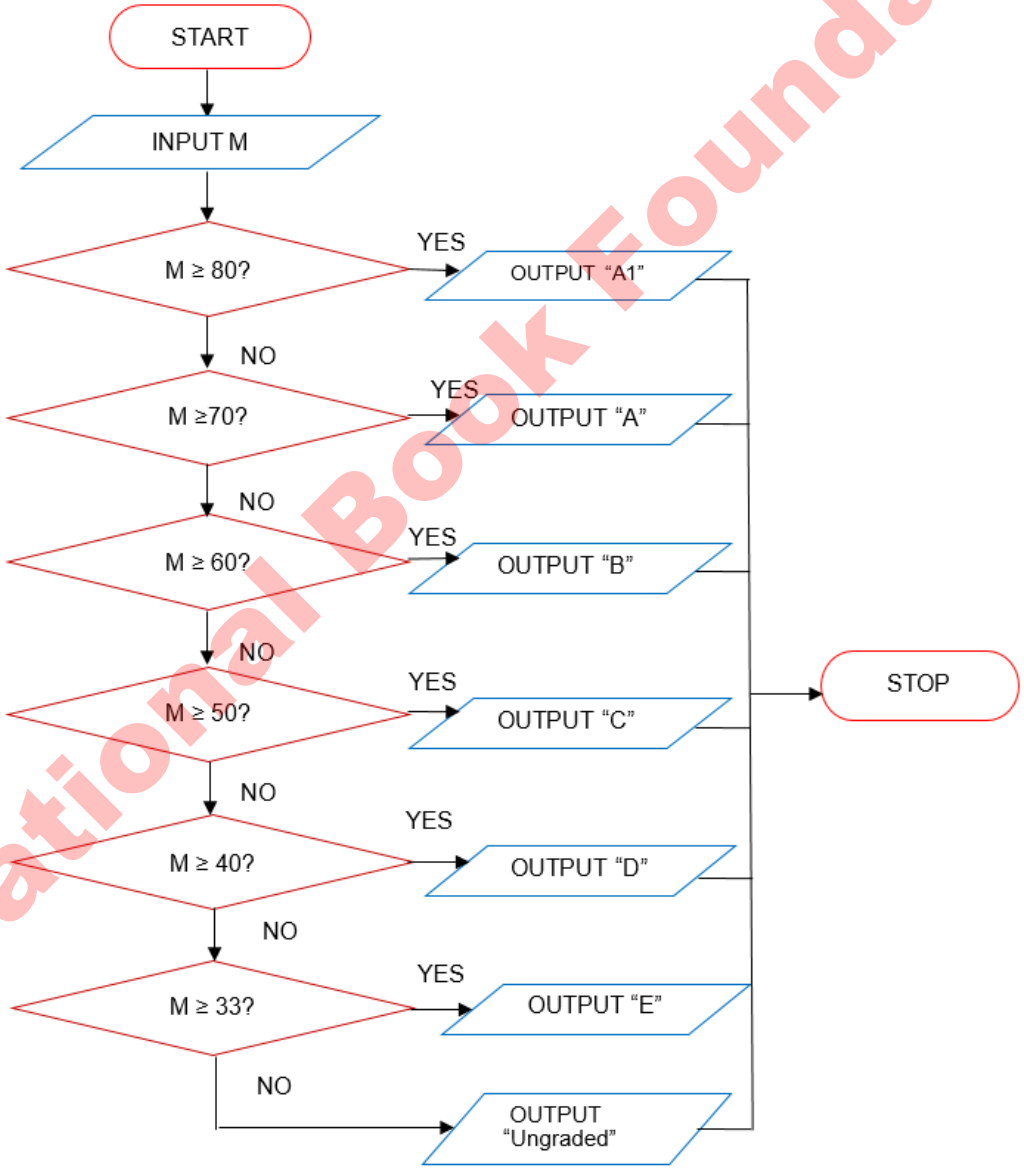


Fig.1-6 Flowchart to print letter grade

Flowchart 7: Flowchart to find the exponent (E) of a given number (N)

In this flowchart, the value P represents N raised to the power E.

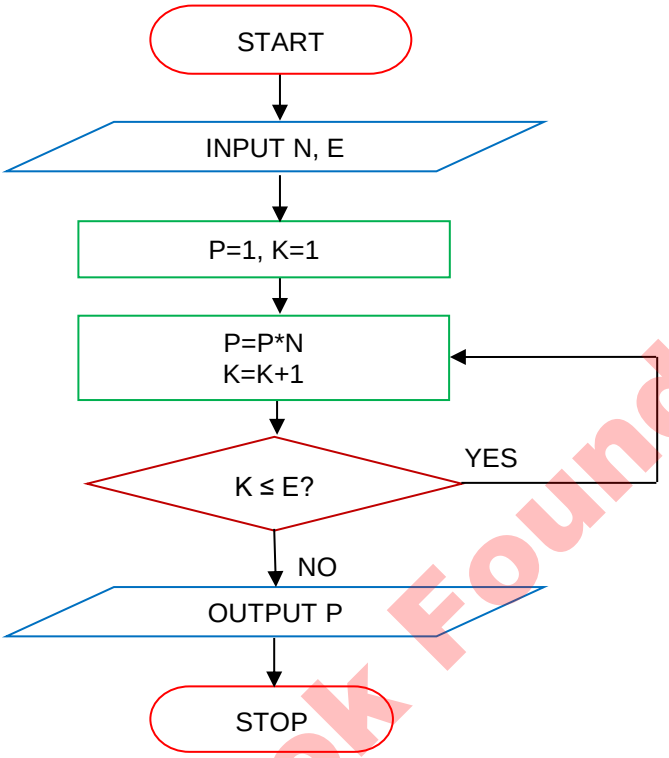


Fig.1-8 Flowchart to find exponent of a number

Flowchart 8: Flowchart to print odd numbers from 1 to 100

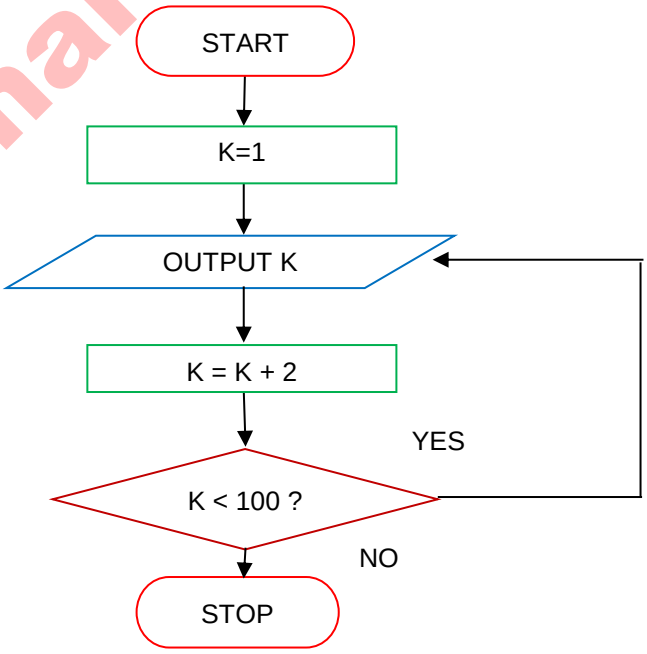


Fig.1-9 Flowchart to print odd numbers

Flowchart 9: Flowchart to print the given sequence of numbers in descending order 27 24 21 18 15 12 9 6 3 0 -3 -6

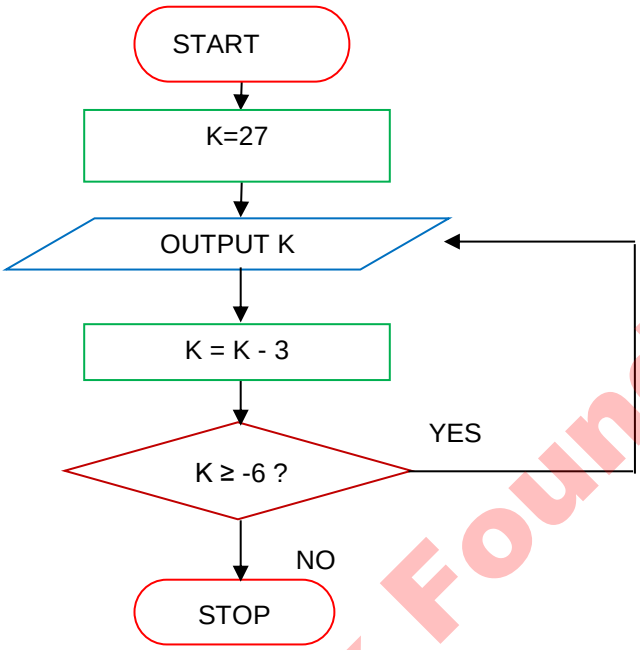


Fig.1-10 Flowchart to print numbers in descending order

Flowchart 10: Flowchart to find the sum of even numbers up to 100

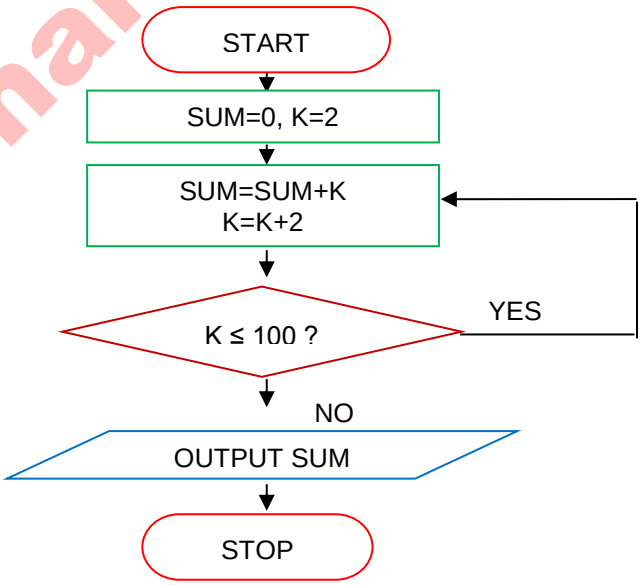


Fig.1-11 Flowchart to find the sum of even numbers

Flowchart 11: Flowchart to print a multiplication table of a given number

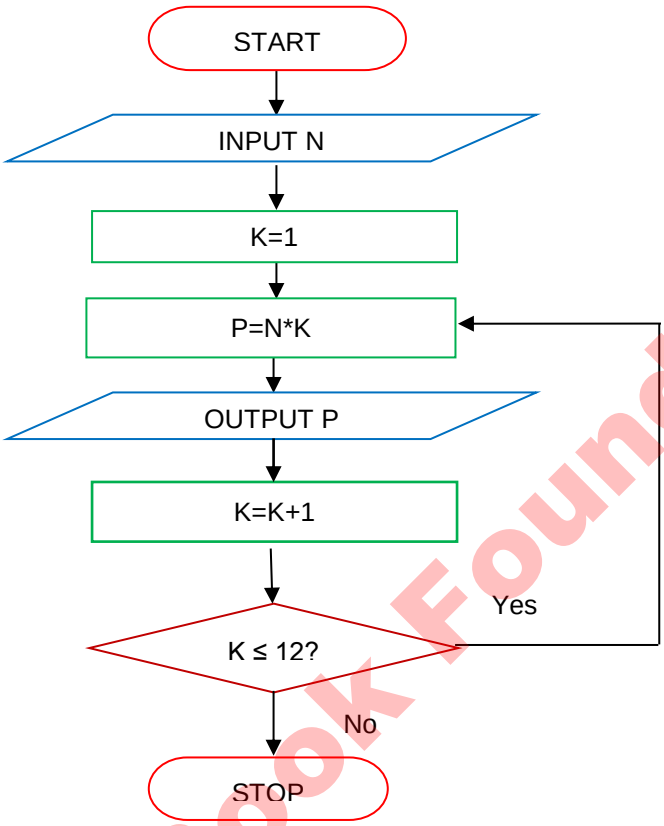


Fig.1-12 Flowchart to print multiplication table of a number

Flowchart 12: Flowchart to convert temperature from Fahrenheit to Celsius

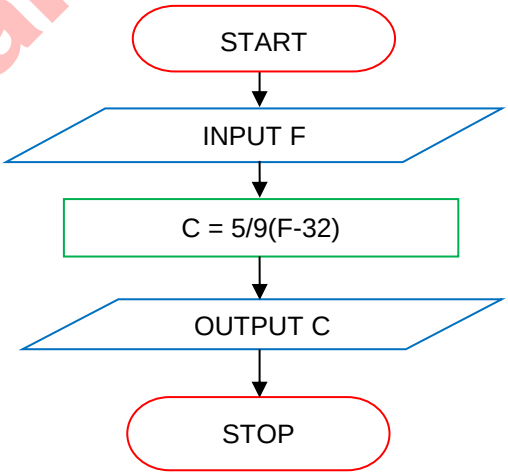


Fig.1-13 Flowchart to convert temperature

Flowchart 13: Flowchart to find factorial of a number

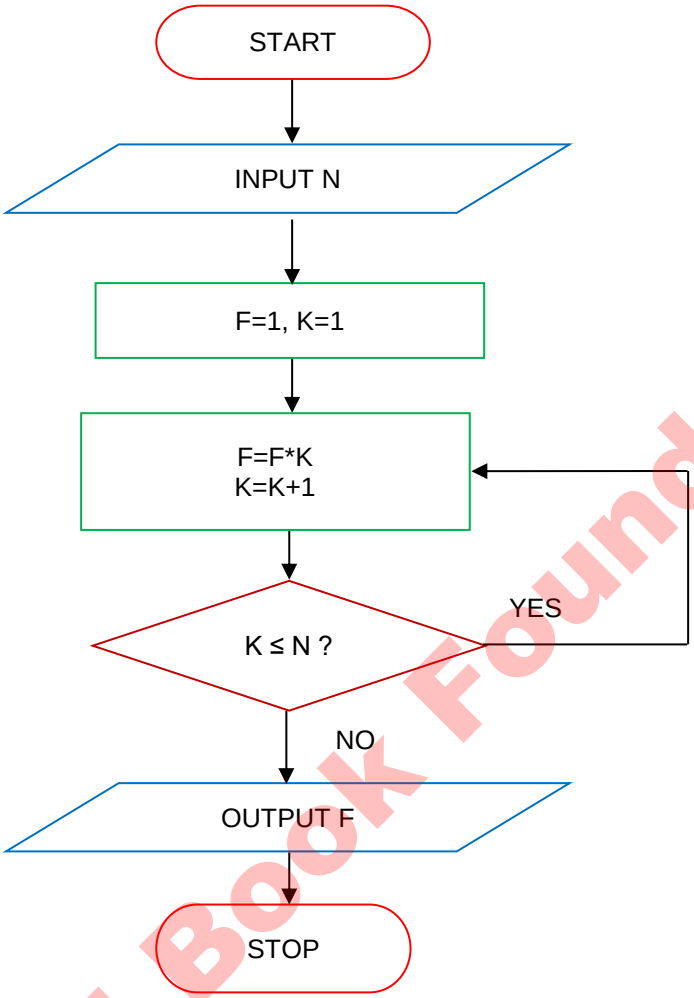


Fig.1-14 Flowchart to print factorial of a number

Key Points

- **Computer** is a general-purpose electronic machine invented to help people solve various problems.
- **Candid solutions** of a problem are all the possible solutions for a problem.
- **Algorithm** is a step by step problem solving method that is easy to understand and follow. It is a set of steps that clearly defines a sequence of operations to solve a problem.
- **Flowchart** is a diagrammatic representation of algorithm.
- **Sequential Structure** refers to the execution of operations in the order in which they appear.
- **Repetition Structure** or **loop** provides a way to repeat one or more operations as many times as required to solve a problem.
- **Selection Structure** allows a choice among various options while solving a problem

Exercise

Q1. Select the best answer for the following MCQs.

i. Which of the following structures repeats one or more operations?
   A. Sequence B. Selection
   C. Loop D. Decision

ii. Which of the following structures allows a choice among various options?
   A. Sequence B. Selection
   C. Loop D. Decision

iii. Which of the following is a sequence of instructions written in a computer language to solve a problem?
   A. Algorithm B. Flowchart
   C. Program D. Problem Analysis

iv. What illustrates a sequence of operations to be performed to solve a problem in the form of a diagram?
   A. Algorithm B. Flowchart
   C. Program D. Problem Analysis

v. What is represented by parallelogram in a flowchart?
   A. Input/Output B. Processing
   C. Start/Stop D. Decision

vi. What is represented by a small circle in a flowchart?
   A. Start/Stop B. Decision
   C. Processing D. Connector

vii. Which symbol is used for decision in a flowchart?
   A. Rectangle B. Parallelogram
   C. Diamond D. Oval

viii. Which symbol is used for processing in a flowchart?
   A. Rectangle B. Parallelogram
   C. Diamond D. Oval

Short Questions

Q2. Give short answers to the following questions.

i. Define a computer.
ii. What is algorithm and what is the role of algorithm in problem solving?
iii. What is a flowchart?
iv. What are the advantages of using flowcharts?
v. Draw any four graphical symbols used in flowchart and explain them.

Extensive Questions

Q3. Describe the steps involved in problem solving.

Q4. Write an algorithm to calculate the area of a rectangle for given breadth and length.

Q5. Write an algorithm that inputs length in inches and calculates and prints it in centimeters.

Q6. Write an algorithm that inputs marks and prints the message “PASS” or “FAIL”. Passing marks are 33.

Q7. Write an algorithm to find the sum of given sequence.

SUM = 20 + 25 + 30 + 35 + 40 + 45 + 50 + 55 + 60

Q8. Write an algorithm to find the product of given numbers.

PRODUCT = 1x3x5x7x9x11x13x15

Q9. Write an algorithm to print multiplication table of a number in reverse order.

Q10. Convert the algorithms of questions Q4 to Q9 to flowcharts.

2

PROGRAMMING IN C

After completing this lesson, you will be able to:

- Define computer program, syntax and semantic
- Explain low level and high level languages
- Elaborate the characteristics of a high level language
- Know popular programming languages
- Differentiate between compiler and interpreter
- Describe the concept of Integrated Development Environment
- Define C language (Editor, Compiler, Linker, Loader and Debugger)
- Define header files
- Identify reserved words
- Describe basic structure of a C program
- Explain purpose and syntax of comments
- Differentiate between constant and variable
- Describe the rules for specifying variable names
- Know the data types of C language

Reading

UNIT INTRODUCTION

This unit provides basic knowledge about low level and high level languages so that student can understand their purpose and characteristics in programming the computer. It describes the main features of popular high level languages. This teaches student which programming language is suitable for developing a particular type of program. The last part of the unit describes the structure of a C program and how to write, compile and execute it program using Integrated Development Environment. This explains how a program is written and executed using a user-friendly graphical user interface.

2.1 INTRODUCTION TO COMPUTER PROGRAMMING

Computer programming is the process of writing a computer program in computer language to solve a particular problem. Computer program controls the operation of a computer and it is developed in a computer language to perform a task. Therefore, it is essential for students to learn computer languages to solve various problems. Computer languages are also known as programming languages.

2.1.1 COMPUTER PROGRAM

A computer program is a set of instructions (statements) written in a programming language to solve a particular problem and achieving specific results. Any task performed by a computer is controlled by a set of instructions that are executed by the microprocessor. A large variety of programming languages have been developed for writing computer programs to use the computer as a problem-solving tool. Each statement of a programming language has syntax and semantic.

- **Syntax**

Syntax refers to the rules of a programming language according to which statements of a program are to be written. It describes the way to write correct statements in a program. Syntax of a programming language is similar to the grammar of a natural language.

For example, an assignment statement consists of a variable and an expression separated by equal sign as an assignment operator. This is the syntax of assignment statement and it can be expressed as given below.

variable = expression;

- **Semantic**

Semantic gives meaning to statements of a programming language. It describes the sequence of operations to be performed by a computer when executing the statements of a computer program.

For example, in the assignment statement:

sum = a + b;

the semantic of the statement is to perform the expression, that is, add the values stored in variables **a** and **b** and then store the result in variable **sum**.

2.1.2 PROGRAMMING LANGUAGES

A programming language is a language which is understood by computer. It is designed to give instructions to a computer to perform a specific task. It is used to write computer programs. Programming languages can be classified into two categories, that is, low level languages and high level languages.

Low Level Languages

Low level language is machine-oriented language. To understand low level language, detailed knowledge of internal working of computer is required. Low level languages include machine language and assembly language.

- **Machine Language**

Programming language that is directly understood by computer hardware is known as machine language. Machine language is associated with architecture of computer. Therefore, programs written in machine language for one computer will not work on another because of design differences. It consists of zeroes and ones. It is almost impossible for humans to use machine language because it entirely consists of numbers. Therefore, practically no programming is done in machine language. Instead, assembly languages and high level languages are used.

- **Assembly Language**

Assembly language consists of symbolic codes or abbreviations known as mnemonics. It was developed to make computer programming easier than machine language. The abbreviations used in assembly language make it easier to learn and write programs compared to machine language. A program written in assembly language must be converted into machine language before it is executed by computer. A program known as assembler is used to translate assembly language into machine language. Some important characteristics of Assembly language are:

- Assembly language allows programmers to have access to all the special features of the computer they are using. Certain types of operations which are not possible in high level languages are easily programmed using assembly language.
- Generally a program written in assembly language will require less storage and less running time than one prepared in a high level language.
- Assembly languages are still the best choice in some applications but their use is gradually declining.

High Level Languages (HLLs)

High level languages are English-oriented languages and they are commonly used for writing computer programs. These languages use English language words such as **print**, **go to**, **if, end**, etc. Therefore, they are easy to learn and use. Some examples of high level languages are **Visual Basic**, **C**, **Java** and **Pascal**.

A program known as compiler/interpreter is required to translate a high level program into machine language. Coding and debugging of a high level language program is much easier than a program written in a low level language.

High-level languages can be classified into procedural, structured and object-oriented programming languages.

- **Procedural Languages**

Procedural programming is based upon the concept of modular programming. In modular programming, programs are divided into smaller parts known as modules. Modular programs

consist of one or more modules. A module is a group of statements that can be executed more than once in a program. Each module in a program performs a specific task.

It is easy to design, modify and debug a program in a procedural language since it provides better programming facilities.

Some examples of procedural languages are FORTRAN, Pascal, C and BASIC.

- **Structured Languages**

Structured languages consist of three fundamental elements, which are sequence, selection and repetition.

Sequence: It means, writing program statements in a logical sequence. Each step in the sequence must logically progress to the next without producing any undesirable effects.

Selection: It allows the selection of any number of statements based on the result of evaluation of a condition which may be true or false. Examples of statements that implement selection in programming are **if**, **else-if**, **switch**, etc.

Repetition (loop): It means executing one or more statements a number of times until a condition is satisfied. Repetition is implemented in programs using statements, such as **for** and **while** loops.

Some examples of structured languages are ALGOL, PL/1, Ada and Pascal.

- **Object-Oriented Programming Languages (oops)**

Object-oriented programming (oops) refers to a programming method that is based on objects such as student, vehicle, building, etc. Object-oriented programming language provides a set of rules for defining and managing objects. An object can be considered a thing that can perform a set of activities. For example, the object vehicle can be defined as an object that has number of wheels, number of doors, color, number of seats, etc. The set of activities that can be performed on this object include Steer, Accelerate, Brake, etc.

Complicated and large computer programs are difficult to design, develop, maintain and debug. The concept of object-oriented programming solves this problem.

The most widely used object-oriented programming languages are C++, C#, php and Java.

2.1.3 CHARACTERISTICS OF HIGH LEVEL LANGAUGES

High-level languages have the following characteristics.

1) These languages were developed to make computer programming simple, easier and less prone to errors.

2) High level languages are not machine dependent. They enable programmers to write programs that are independent of a particular type of computer.

3) Programs written in high-level languages must be translated into machine language by a compiler or an interpreter before execution by the computer.

4) The process of finding and removing errors in programs (debugging) is easier in high-level languages compared to low level language.

5) High-level language programs are highly structured. They allow programmers to break lengthy programs into a number of modules which can be written and tested independently. This makes writing and testing of programs easier.

2.1.4 POPULAR HIGH LEVEL LANGUAGES

C/C++

C language was developed in early 1970s by Dennis Ritchie at Bell Laboratories. C has become one of the most popular programming languages today. It is a highly structures programming language that is easy to understand and use. In the past, it was mainly used for writing system programs such as operating systems, compilers, assemblers, etc. Today, it is used for writing all types of application programs as well, such as word-processing programs, spreadsheet programs, database management systems, educational programs, games, etc.

C++ was developed by Bjarne Stroustrup also at Bell Laboratories during 1983-1985. C++ is a superset of C, meaning that any valid C program is also a valid C++ program. The purpose of developing C++ was to provide programming facilities to easily and quickly write more powerful programs.

Visual Basic

Visual Basic (VB) is a high level language which evolved from the earlier version called BASIC. BASIC stands for Beginner's All-purpose Symbolic Instruction Code. VB is a very popular programming language for writing Windows and Web applications. It provides a graphical development environment to programmers to develop powerful Windows and Web applications. VB is commonly used for developing business programs such as payroll system and inventory control program. The user can also write programs related with engineering, science, arts, education, games, etc.

C#

C# (pronounced as C-sharp) is a language developed in 2000 by Microsoft Corporation. It is a simple, modern, general-purpose programming language. Syntax of C# is very similar to C and C++. It also has some features of Java. It is a language that makes computer programming easy and efficient. It provides facilities to write Web applications that can be used across the Internet. All types of programs including games, utilities, operating systems, compilers, business applications and Web based applications can be developed in C#.

Java

Java is a high-level language developed by Sun Microsystems. It is very similar in syntax to C and C++. In Java, the user can write all types of programs as those written in other programming languages and small programs that can be embedded in a Web page accessed through Internet. Java is an ideal language for network computing. It is used for writing programs for a wide range of devices, computers and networks. It is widely used in Web applications. The current versions of most of the Web browsers are made Java enabled. A few browsers that support Java are Microsoft's Internet Explorer, Firefox and Mozilla.

2.1.5 COMPILER AND INTERPRETER

Compiler

A compiler is computer software that translates source program into object program as shown in Fig.2-1.

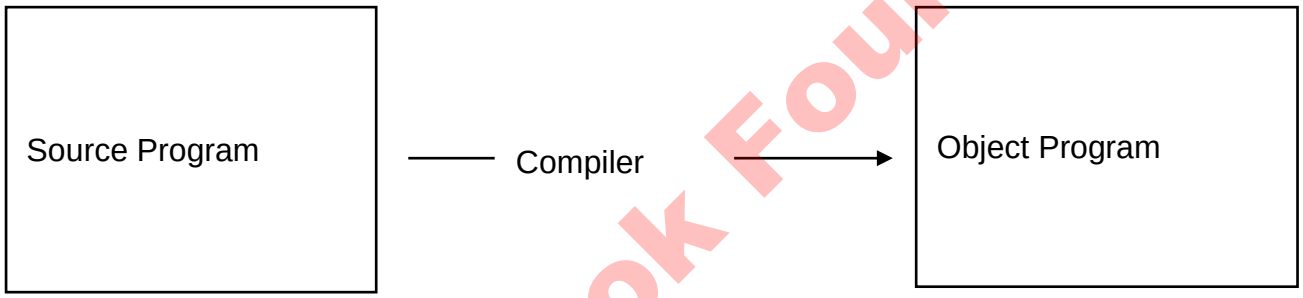


Fig.2-1 Translation of source program into object program

Source program consists of statements written in a high level language such as C, Pascal, Java, etc. For example, a program written in C language by a programmer to print table of a number is known as source program. When it is translated with a compiler into machine language, the resulting program is known as object program. The object program is understandable by computer processor but difficult for a human to read and understand because it consists of zeroes and ones.

Interpreter

Interpreter translates high level language programs into machine language but it translates one instruction at a time and executes it immediately before translating the next instruction. Examples of programming languages that use interpreter are Java Script, BASIC, Visual Basic and Perl.

Interpreter reads each statement of source program, one at a time and determines what it means as it executes it. It means each time a statement is read, it must be translated into machine language before execution. Compiler translates the entire program into object program before execution by computer. Therefore, a compiled program runs fast.

2.2 PROGRAMMING ENVIRONMENT

Programming environment is the set of processes and programming tools used to develop computer programs. In the past, programmers used various standalone programming tools for developing computer programs. These included editor, compiler, linker, debugger, etc. Using separate programs provided a difficult and time consuming environment for creating computer programs. Today, programmers use Integrated Development Environment for developing computer programs. Integrated Development Environment is an application package that consists of editor, compiler, linker and debugger with a graphical user interface.

2.2.1 INTEGRATED DEVELOPMENT ENVIRONMENT

Most of the new programming languages use Integrated Development Environment (IDE) to create, compile and run programs. IDE is computer software that brings all the processes and tools required for program development into one place. IDE's aim is to make the life of programmers easier by grouping together all the tasks needed for building applications into one environment. Today's modern IDEs have user-friendly Graphical User Interface (GUI).

2.2.2 PROGRAMMING ENVIRONMENT OF C LANGUAGE

A C language IDE consists of the following modules.

- Text Editor
- Compiler
- Linker
- Loader
- Debugger

Text Editor

A text editor is a simple word-processor that is used to create and edit source code of a program as shown in Fig.2-2. Files created by a text editor are plain text files. Most of the editors automatically highlight compile errors to simplify removing them.

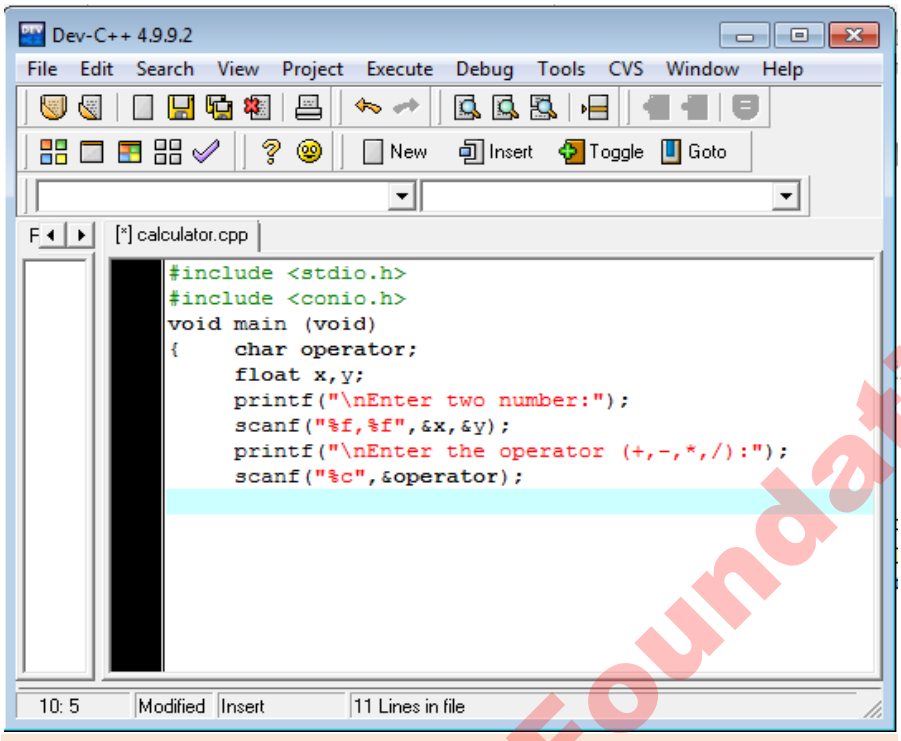


Fig.2-2 Text editor used for creating and editing a program

Compiler

A compiler is a computer software that translates C language program (source program) into machine code (object program) that can be understood and executed by the computer as shown in Fig.2-3. It also detects syntax errors in the source program and gives hints to the programmer to correct them. A compiler will only create object program if all the syntax errors in a program have been corrected if they existed. A program written in C language has the extension **.c** (for example first.c) and after compilation the object program extension will be **.obj** (for example first.obj)

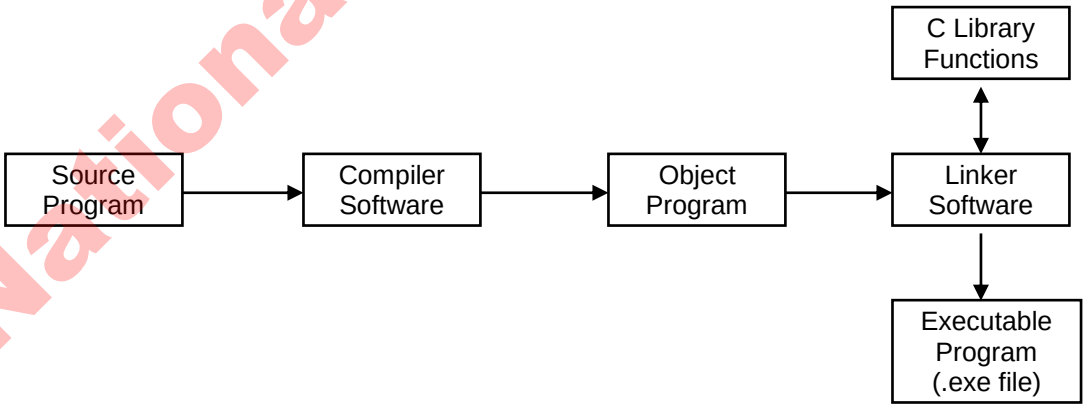


Fig.2-3 The process of creating an executable program

Linker

Linker is a software that translates object program into a single executable program as shown in Fig.2-3. During this process, if it could not find the definition of a particular function that

is used in the program, then it would assume that it is defined in C library. It will replace this function in the object program with the code from C library and then create a single executable program.

Loader

It is a software that loads programs into memory and then executes them.

Debugger

It is a software that executes a program line by line, examines the values stored in variables and helps in finding and removing errors in programs.

2.3 PROGRAMMING BASICS

C is a popular and widely used programming language for creating computer programs. A large variety of application programs and many modern operating systems are written in C.

2.3.1 C LANGUAGE CHARACTER SET

The C Language character set includes:

Letters

C language comprises the following set of letters to form a standard program. They are:

- A to Z in Capital letters.
- a to z in Small letters.
- In C programming, small latter and caps latter are distinct.

Digits

- C language comprises the following sequence of numbers to associate the letters. 0 to 9 digits.

Special Characters

C language contains the following special character in association with the letters and digits.

Symbol	Meaning	Symbol	Meaning	Symbol	Meaning
~	Tilde	_	Underscore	]	Right bracket
!	Exclamation mark	+	Plus sign	:	Colon
#	Number sign	|	Vertical bar	"	Quotation mark
$	Dollar sign	\	Backslash	;	Semicolon
%	Percent sign	`	Apostrophe	<	Opening angle bracket
^	Caret	-	Minus sign	>	Closing angle bracket
&	Ampersand	=	Equal to sign	?	Question mark
*	Asterisk	{	Left brace	,	Comma
(	Lest parenthesis	}	Right brace	.	Period
)	Right parenthesis	[	Left bracket	/	Slash

2.3.2 RESERVED WORDS

The words that are part of programming language and have special purposes in computer programs are called reserved words or keywords. They have predefined use and cannot be used for any other purpose. Reserved words are always written in lowercase. There are 32 words defined as reserved words in C. A complete list of reserved words used in C language is given in Fig.2-5

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

Fig.2-5 List of reserved words of C

2.3.3 HEADER FILES

C language contains a number of standard functions in library file that perform various tasks in C programs. These tasks include all the input/output operations and all the math operations. Library file contains header files and each header file contains a set of functions. Some commonly used header files are stdio.h, conio.h and math.h.

- Some commonly used functions that are included in stdio.h file are printf(), scanf(), etc.
- Some of the functions that are included in conio.h file are clrscr(), getch() and getche().
- The math.h header file contains mathematical function such as sqrt(), pow(), log(), sin(), cos() and tan().

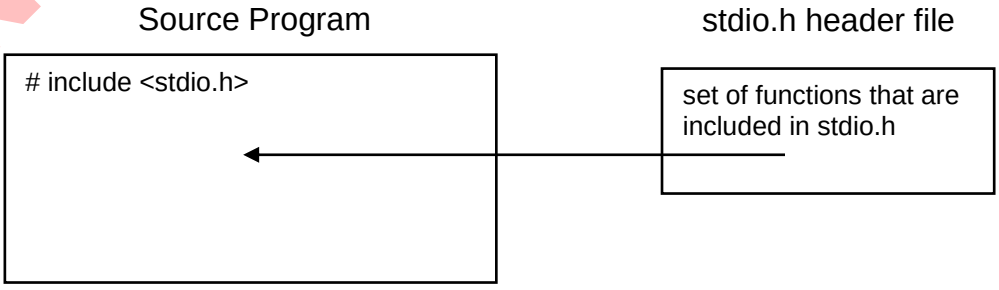


Fig.2-4 Use of header file in a C program

In order to use a library function, the programmer must include appropriate header file at the beginning of the program as shown in Fig.2-4. When source program is translated into executable file, a copy of code of function is pasted in the source program by the linker from the header file before creating the executable file.

2.3.4 STRUCTURE OF A C PROGRAM

The format according to which a program is written is called the structure of program. The following is the structure of a C program.

```
Preprocessor directives
Main function
{
        Local declarations      } Body of main() function
        Statements              }
}
User-defined function
Function 1                      }
Function 2                      } (option to user)
Function 3                      }
```

To introduce the structure of a C program, consider a very simple program that will print the message:

I Love Pakistan

The program may be written as shown in Fig.2-6.

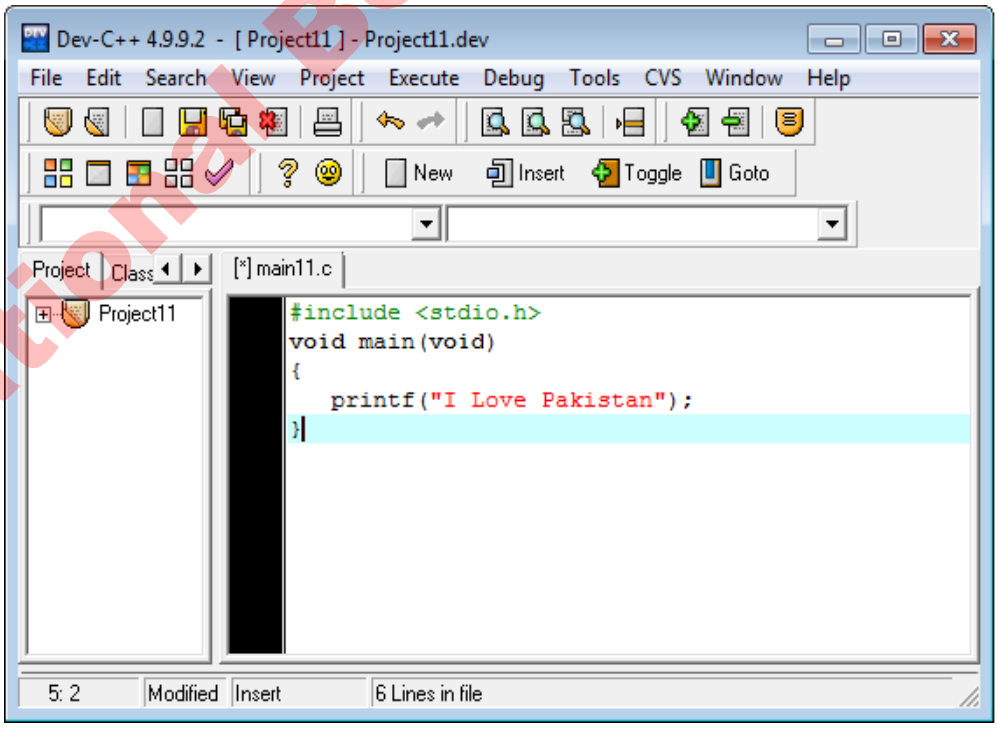


Fig.2-6 Program to print a message on the screen

This program consists of three main parts which are preprocessor directive, the main() function and the body of main().

Preprocessor Directives

Preprocessor directives are instructions for the C compiler. Every C language program contains certain preprocessor directives at the beginning of the program. Before translating a C language program into machine language, the compiler of C language carries out the processor directives. These directives start with number sign (#). The most commonly used preprocessor directives are **#include** and **#define**.

#include Preprocessor Directive

It has the following syntax.

#include<header file name>

When this preprocessor is carried out by the C compiler, it will search for the header file that is written within the less than (<) and greater than (>) symbols and copy it into the source file.

In the above program the header file **stdio.h** is used. It tells the C compiler to copy the **stdio.h** header file into the program. The **stdio.h** header file stands for standard input-output header. It includes the standard **printf( )** and **scanf()** function prototypes. In the above program the **printf()** functions is used. Therefore, it is required to include this header file in the include preprocessor directive.

main() Function

C programs consist of one or more functions. A function performs a single well-defined task. Every C program must have the function **main()** which is the first section to be executed when the program runs. The general form of main() is:

void main(void)

The word **void** before the function **main()** means that this function does not return a value and the second **void** inside the brackets means it does not have any argument.

Body of main() Function

The body of the function **main()** is surrounded by braces (curly brackets { and }). The left brace indicates the start of the body of the function and the matching right brace indicates the end of the body of the function.

The body of the function in the program (see Fig. 2.6) consists of a single statement **printf()** which ends with a semicolon(;). **printf()** is the standard output function. The text to be printed is enclosed in double quotes. A statement in C is terminated with a semicolon. Therefore, a semicolon is used at the end of **printf()** statement. This program will print the message **I Love Pakistan** on the screen.

2.3.5 COMMENTS IN C LANGUAGE

It is a good programming practice to add comments in program to make it easy for others to understand it. Comments in the source code are ignored by the compiler.

Comments are added in programs when a fact is necessary to be brought to the attention of program's reader. Generally, programmers write comments at the beginning of the program explaining the reader what the program is intended to achieve and inside the program code where something is to be clarified about the structure of the program.

There are two types of comments.

- **Single line comment**
- **Multiple line comment**

Single line comment

The // is used as single line comment. The syntax of single line comment is:

// comment

An example is:

// Programmer: M. Sajjad Heder

Multiple line comment

The /* …. */ is used for multiple line comments.

Comment can span to multiple lines as shown in the following example.

```
/* Programmer: M. Sajjad Heder
 Programming Language: C
 Date Program was Written: 7-01-2019 */
```

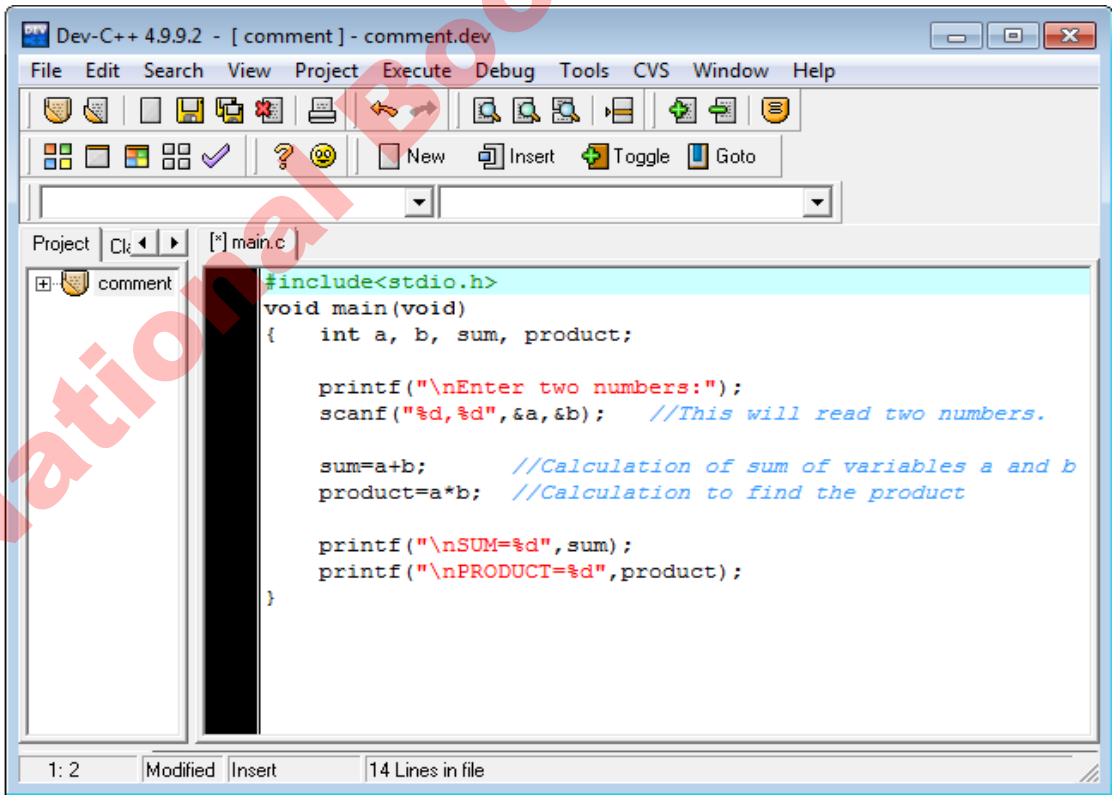


Fig.2-7 Program that demonstrates the use of comment

The program in Fig.2-7 demonstrates how single line comments are used in a program. In this program comments describe the purpose of statements.

2.4 CONSTANTS AND VARIABLES

Constant and variables are used in expressions in computer programs. After an expression is evaluated, the result of the expression is also stored in a variable.

2.4.1 CONSTANT AND VARIABLE

Constant

Constants are quantities whose values do not change during program execution. They may be numeric, character or string.

Numeric Constants are of two types, integer and floating-point numbers.

Integer constants represent values that are counted, like the number of students in a class. Some examples of integer constants are 7145, -234, 26, etc.

Floating-point constants are used to represent values that are measured, like the height of a person which might have a value of 166.75 cm or the weight such as 82.6 kilograms.

Character Constant is one of the symbols in C character set. It includes digits 0 to 9, upper-case letters A to Z, lower-case letters a to z, punctuation symbols such as semicolon (;), comma (,), period (.) and special symbols such as +, -, =, >, etc. A character constant is enclosed by single quotes such as ‘a’, ‘s’, etc.

String Constant contains a string of characters within double quotes such as “Hello Ahmed”, “a”, etc.

Variable

A variable is a symbolic name that represents a value that can change during execution of a program. A variable has a name, known as variable name and it holds data of other types. A number or any other type of data held in a variable is called its value. It also indicates the types of value a variable can represent. Variables are of two types, numeric and character.

Numeric variables are used to represent numeric values in computer programs. They represent integer and floating-point values. Some examples of numeric variables are sum, avg, length, salary, marks, etc.

Character variables represent character values in computer programs. It can represent a single character or a string of characters. Some examples of character variables are name, city, gender, etc.

When a variable is used in a computer program, the computer associates it with a particular memory location. The value of a variable at any time is the value stored in the associated memory location at that time. Variables are used so that the same space in memory can hold different values at different times.

2.4.2 RULES FOR SPECIFYING VARIABLE NAMES

The following are the rules for specifying variable names in C language.

- A variable begins with a letter or underscore (_) and may consist of letters, underscores and/or digits.
- The underscore may be used to improve readability of the variable name. For example, **over_time**.
- There is no restriction on the length of a variable name. However, only the first 31 characters of a variable are significant. This means that if two variables have the same first 31 characters they are considered to be the same variables.
- Both upper and lower case letters are allowed in naming variables. An upper case letter is considered different from a lower case letter. For example the variable AVG is different from Avg or avg.
- Special characters cannot be used as variable name. e.g., #, ?, @ etc.
- Reserved words of C language such as **int**, **case**, **if**, etc., cannot be used as variable names.
- Space is not allowed in the name of variable. For example **ma ss** is not correct.

2.4.3 DATA TYPES USED IN C PROGRAMS

C provides three main data types for variables, that is, integer, floating-point and character variables.

- **Integer Data Type**

Integer variable declaration statement has the form:

Type specifier Variable;

For example:

int sum;

The declaration consists of the type name, **int**, followed by the name of the variable, **sum**. All the variables used in a C program must be declared. If there are more than one variable of the same type, separate the variable names with commas as shown in the following declaration statement.

int a,b,sum,product;

Declaring a variable tells the computer the name of the variable and its type. This enables the compiler to recognize the valid variable names in program. This is very helpful if the user types the wrong spellings of a variable name in his program, the compiler will give an error message indicating that the variable is not declared.

Declaration of an integer variable can begin with the type qualifiers, **short**, **long**, **unsigned** or **signed**. Some examples are:

short int num;
long int sum, avg;
unsigned int count;

short unsigned int count;

Type qualifiers refer to the number of bytes used for storing the integer on a particular computer. Refer to computer manual to find out the sizes of the computer being used. Generally, the number of bytes set aside by the compiler for the above type qualifiers are as given in Fig.2-8.

Variable Declaration	No. of Bytes	Range
int or short int	2	-32,768 ~ +32,767
long int	4	-2,147,483,648 ~ +2,147,483,647
unsigned int	2	0 ~ 65,535

Fig.2-8 Integer data types used in C

Here, **unsigned** implies that the value of variable is greater than or equal to zero. The qualifier, **signed**, is rarely used since by default, an **int** declaration assumes a **signed** number. Note that in the above examples, the word **int** may be omitted when it begins with **long** or **unsigned** type qualifiers in the declaration statements. For example you can write:

short num;

- **Floating-point Data Type**

Floating-point variables are used for storing floating-point numbers. Floating point numbers are stored in memory in two parts. The first part is the **mantissa** and the second part is the **exponent**. The **mantissa** is the value of the number and the **exponent** is the power to which it is raised. Some examples of floating-point variable declaration statements are:

float base,height;
double float a,b,total;
long double float size,x,y;

Usually the number of bytes set aside by the compiler for the above floating-point type qualifiers are as given in Fig.2-9.

Variable Declaration	No. of Bytes	Range
float	4	10^{-38} ~ 10^{38} (with 6 or 7 digits of precision)
double float	8	10^{-308} ~ 10^{308} (with 15 digits of precision)
long double float	10	10^{-4932} ~ 10^{4932} (with twice the precision of double)

Fig.2-9 Floating-point data types used in C

Here, precision means the number of digits after the decimal point. The word **float** may be omitted when it is preceded by **double** or **long double** type qualifiers in the declaration statements.

There is another method of representing floating-point numbers known as exponential notation. For example, the number 23,688 would be represented as 2.3688e4. Here, 2.3688 is the value of the number (mantissa) and 4 is the exponent(e). The exponent can also be negative. For example, the number 0.0005672 is represented as 5.672e-4 in exponential notation. Exponential notation is used to represent very large and very small numbers. In C, a floating point number of type **float** is stored in four bytes. Three bytes for the **mantissa** and one byte for the **exponent** and provides 6 or 7 digits of precision.

- **Character Data Type**

Character variables are used to store character constants. Examples of declaration of character variables are:

char ch, letter,a;

A character variable can only store one character. The compiler sets aside one byte of memory for storing a character in a character variable.

Key Points

- **Syntax** refers to the rules of a programming language according to which statements of a program are to be written.
- **Semantic** gives meaning to statements of a programming language.
- **Machine language** is a programming language that is directly understood by computer hardware. It consists of zeroes and ones.
- **Assembly language** consists of symbolic codes or abbreviations. It was developed to make computer programming easier than machine language. A program called assembler is required to convert assembly language to machine language for execution by the computer.
- **High level language** is an English-oriented language. It is commonly used for writing computer programs. A program called compiler/interpreter is required to translate high level language into machine language for execution by the computer.
- **Compiler** is a computer program that translates a program written in a high level language into machine language equivalent program. It converts the entire program into machine language before execution by the computer.
- **Interpreter** is a computer program that translates a program written in a high level language into machine language. It translates one instruction at a time and executes it immediately before translating the next instruction.

- **Programming environment** is the set of processes and programming tools used to develop computer programs.
- **Integrated Development Environment** (IDE) is a computer software that is used to create, compile and run programs. It brings all the processes and tools required for program development into one place.
- **Text editor** is a simple word-processor that is used to create and edit source code of a program.
- **Linker** is a software that takes one or more object files, generated by a compiler and combines them into a single executable program.
- **Loader** is a software that loads programs into memory and then executes them.
- **Debugger** is a software that executes a program line by line, examines the values stored in variables and helps in finding and removing errors in programs.
- **Header file** is a file that contains predefined functions of C language. Including a header file in a C source file produces the same result as copying the header file into the source file.
- **Reserved words** are those words that are part of a programming language and have special purpose in computer programs.
- **Preprocessor directives** are instructions for the C compiler at the beginning of program.
- **Comments** are notes that are included in programs when a fact is necessary to be brought to the attention of program's reader.
- **Constant** is a quantity whose value does not change during program execution.
- **Variable** is a symbolic name that represents a value that can change during execution of a program.

Exercise

Q1. Select the best answer for the following MCQs.

i. What defines the rules of valid statements in programming?

A. Compiler
B. Interpreter
C. Syntax
D. Semantic

ii. Which language is directly understood by the computer?

A. Machine language
B. Assembly language
C. High level language
D. C language

iii. When was C language developed?

A. Late 1960s
B. Early 1970s

C. 1980s D. 1990s

iv. Who developed Java language?

A. Dennis Ritchie B. Microsoft

C. Sun Microsystems D. IBM

v. What is the other word used for Reserved Words?

A. Compiler words B. Keywords

C. Special programming words D. Mnemonics

vi. How many bytes are set aside by the compiler for a variable of type int?

A. 2 B. 3

C. 4 D. 5

vii. How many bytes are set aside by the compiler for a variable of type float?

A. 2 B. 3

C. 4 D. 5

viii. What is the range of numbers that can be stored in a variable of type double float?

A. -32,768 ~ +32,767 B. $10^{-38} \sim 10^{38}$

C. $10^{-308} \sim 10^{308}$ D. $10^{-4932} \sim 10^{4932}$

ix. Which program translates high level language into machine language?

A. Compiler B. Linker

C. Loader D. Debugger

x. Which software helps in finding and removing errors in programs?

A. Linker B. Text Editor

B. Loader D. Debugger

Short Questions

Q2. Give short answers to the following questions.

i. Define computer program.

ii. Differentiate between syntax and semantic.

iii. Write three differences between assembly language and HLLs.

iv. Write four characteristics of HLLs.

v. Define Integrated Development Environment (IDE).

vi. Differentiate between constant and variable.

vii. Which of the following are valid C variables? Give the reason if not a valid variable.
area, 5x, Sum, net pay, float, _age, else, case, size22, my_weight

viii. What are reserved words? Why they should not be used as variable names?

ix. Why comments are used in programs?

x. What is the purpose of header files in C language?

Extensive Questions

Q3. Describe the following High Level Languages.

a) C/C++ b) Visual Basic

c) C# d) Java

Q4. What is C language IDE? Explain its modules in detail.

Q5. What are the rules for specifying a variable name in C language?

Q6. What is a preprocessor directive? Explain **include#** preprocessor directive in detail.

3

INPUT AND OUTPUT HANDLING

After completing this lesson, you will be able to:

- Use output functions (printf() and puts())
- Use input functions (scanf(), getche(), getch(), getchar() and gets())
- Use statement terminator (semicolon)
- Define format specifier
- Define escape sequence and know how to use them in programs
- Define arithmetic operators and know how to use them in programs
- Know the use of assignment operators
- Define relational operators and know how to use them in programs
- Define logical operators and know how to use them in programs
- Differentiate between assignment operator and equal to operator
- Differentiate between unary and binary operators
- Define expression ternary (conditional) operator
- Define and know the order of precedence of operators

Reading

UNIT INTRODUCTION

Input and output operations provide communication between a computer program and a computer user. Therefore, students should learn how to program the computer to perform input/output operations. This unit teaches how to accept data from an input device such as keyboard and how to output result on an output device such as monitor. It also teaches the use of various types of operators in expressions so that students learn how to perform calculations in C programs.

3.1 INPUT AND OUTPUT FUNCTIONS

In programming, input means to enter or feed data in a computer program and output is what the computer produces after processing the data. C language provides many input/output functions to provide interaction between a program and user.

3.1.1 OUTPUT FUNCTIONS

In computer programming, output means to display information on screen or print on printer. C language provides different functions to output information on computer screen. The most common is the ***printf()*** function.

The printf() Function

The **printf()** function is used to print text, constants, values of variables and expressions on the screen in a specified format.

The general syntax of this function is:

printf(control_string, list of arguments);

The '**control_string'** consists of text, format specifiers and escape sequences. It is written within double quotes. Text specifies the message that is to be printed along with the values. Format specifier specifies the format according to which a value is to be printed. Escape sequence controls the printing on the output device.

The '**list of arguments'** consists of a list of variables, constants and arithmetic expressions, separated by commas, whose values are to be printed. The values are printed according to the corresponding format specifier. The first format specifier applies to the first argument, the second to the second argument and so on. The arguments in the **printf()** function are optional. When **printf()** function is used to print only a message, then the arguments are omitted.

Following is an example of **printf()** function.

printf("\nThe value of a is %d and the value of b is %d.",a ,b);

Assuming that **a** has the value of 3 and **b** has the value of 4, then the following message will be printed by the above statement on a new line.

The value of a is 3 and the value of b is 4.

In the above print statement, **\n** is an escape sequence and **%d** is a format specifier. The escape sequence **\n** is used to start printing on a new line. The format specifier **%d** is replaced by the values stored in variables **a** and **b** in the order in which they are written. The format specifier **%d** is used for printing values stored in integer variables.

3.1.2 INPUT FUNCTIONS

In computer programming input means to feed data into a program through an input device. C language provides different functions to input data. The most common is the ***scanf()*** function.

The scanf() Function

The scanf() function is used to get values into variables from the keyboard during the execution of a program. The value is input into a variable in a specified format.

Its syntax is:

scanf("format specifiers", list of variables);

The '**format specifiers'** specify the format of the variables. These are written within double quotes with or without spaces.

The **'list of variables'** consists of a list of variables, separated by commas, into which the values are to be entered. In C language, ampersand (**&)** symbol is used in **scanf()** function before the name of the variable to which a value is to be assigned. Ampersand (&) sign refers to the memory location where the variable is going to store.

For example, to input values into two integer variables, **a** and **b**, the **scanf()** function is written as:

scanf("%d %d", &a, &b);

Here, the format specifier **%d** is used twice in control string for the variables **a** and **b**.

The program in Fig.3-4, reads two numbers and prints their sum.

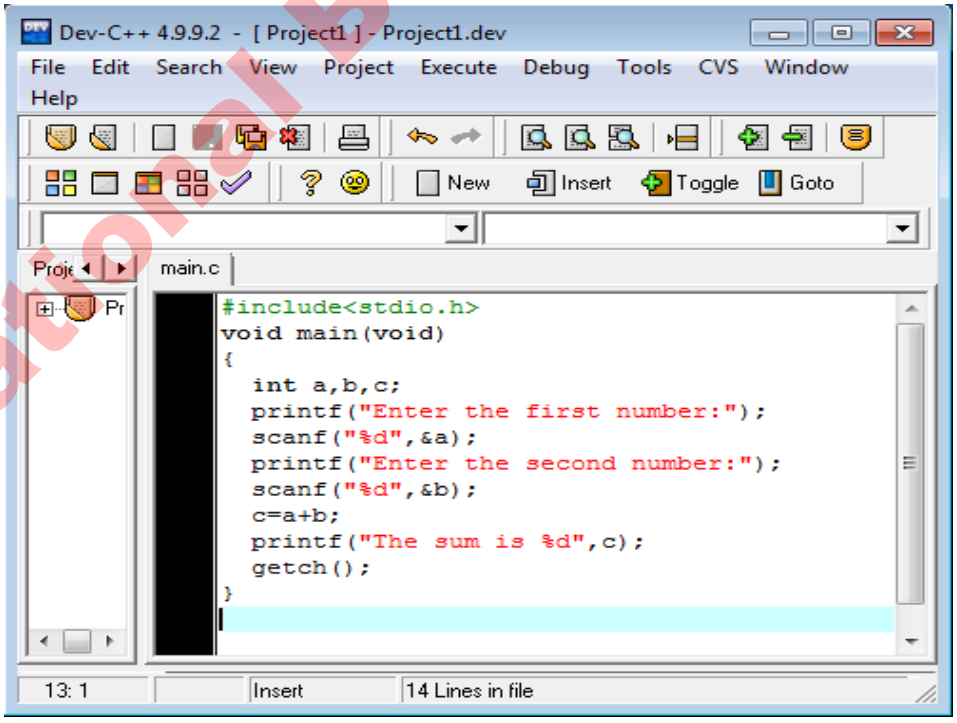


Fig.3-4 Program to find sum of two numbers.

In the above program, the statement

int a,b,c;

is used to declare that **a, b** and **c** are integer variables. Note that the statement ends with a semicolon(;) as all the C statements do.

The statement

printf("Enter the first number: ");

prompts the user for the first number.

The statement

scanf("%d", &a);

reads the value for the variable **a** and stores it in the computer's memory. In this statement, **%d** format specifier specifies that an integer is to be input.

The next two statements prompt the user to input the second number and store it in variable named **b**. The expression **c=a+b** calculates the sum of values stored in variables **a** and **b** and stores it in variable **c**. The last statement prints the sum on the screen.

The execution of this program is shown Fig.3-5.

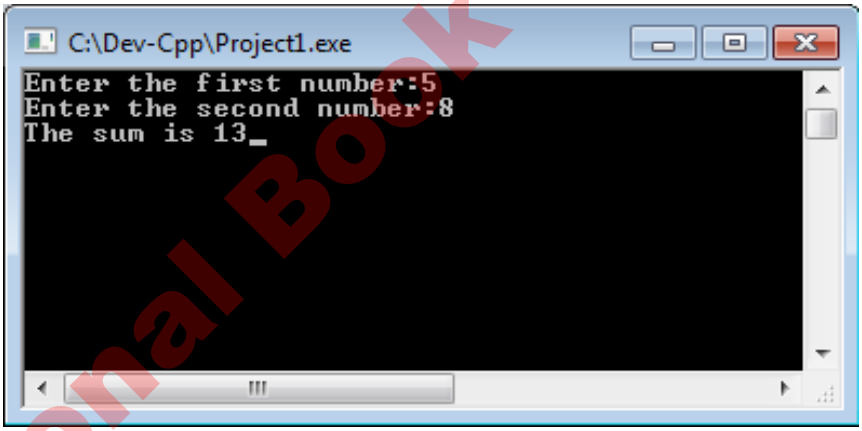


Fig.3-5 Execution of program to print sum of two numbers.

3.1.3 STATEMENT TERMINATOR

Semicolon (;) is entered at the end of a statement in C language. It indicates the compiler that the statement ends here. If a statement is not terminated with semicolon, C compiler will give an error message during compilation and the program will not be compiled.

3.1.4 FORMAT SPECIFIERS

A format specifier is computer code that tells about the data type, field width and the format according to which a value is to be printed or read from an input device. A list of commonly used format specifiers is given below.

%d decimal integer
%i integer
%ld long decimal integer
%f floating-point (decimal notation)
%e floating-point (exponential notation)
%c single character

- **Integer Format Specifiers (%d, %ld and %i)**

The format specifier **%d** is used to read or print a decimal integer and the format specifier **%ld** is used with long integers.

The program in Fig.3-12, demonstrates the use of integer format specifiers.

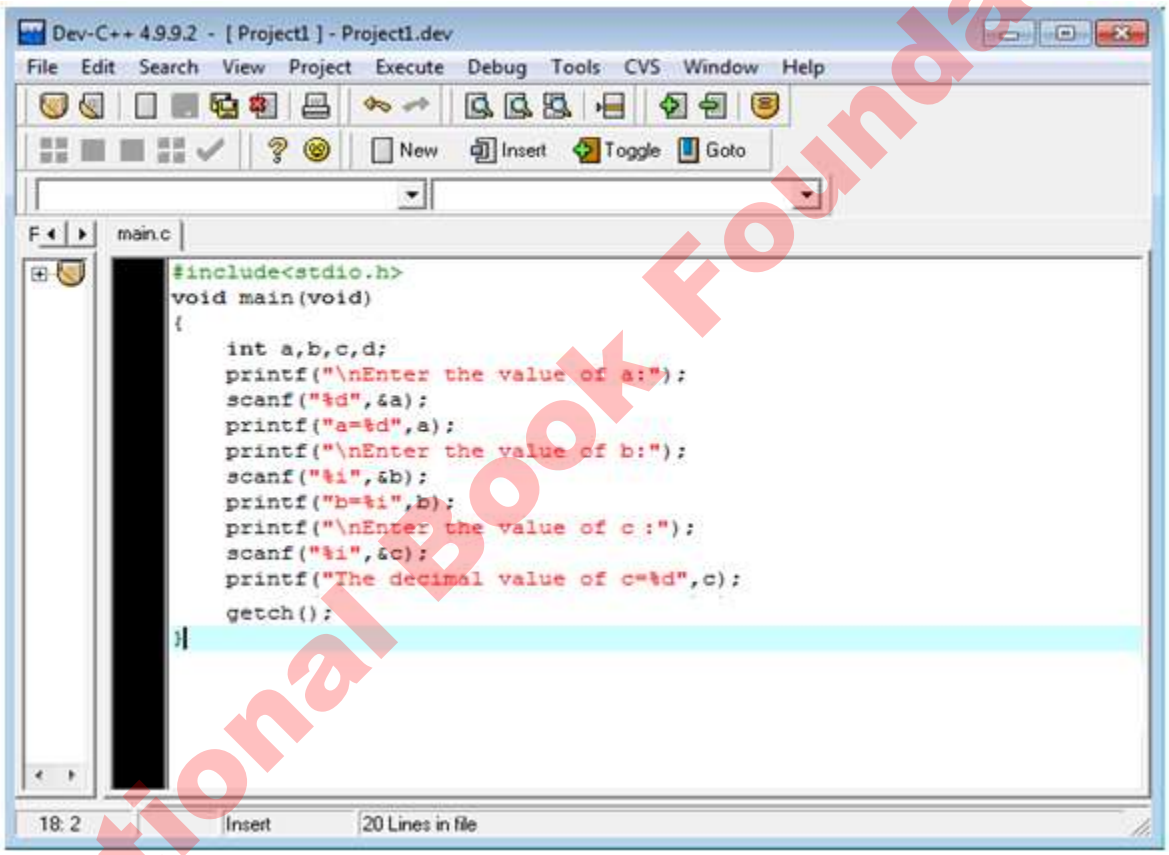


Fig.3-12 Program that uses integer format specifiers

The format specifiers **%d** and **%i** are same when they are used with the **printf()** function but different when used with **scanf()** function. For **printf()** function, both **%d** and **%i** are used for decimal integer as shown in the program.

The execution of the program is shown in Fig.3-13

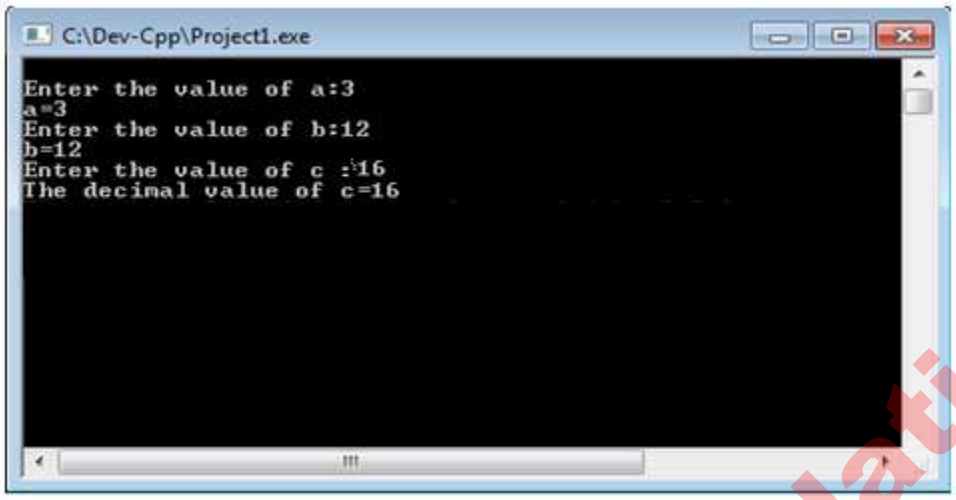


Fig.3-13 Reading decimal, hexadecimal and octal integers

- **Floating-point Format Specifiers (%f, %e)**

The format specifier **%f** is used to read and print floating-point numbers in decimal notation with a precision of 6 digits after the decimal point.

The format specifier **%e** is used to read and print floating-point numbers in exponential notation.

The program in Fig.3-14, demonstrates the use of floating-point format specifiers.

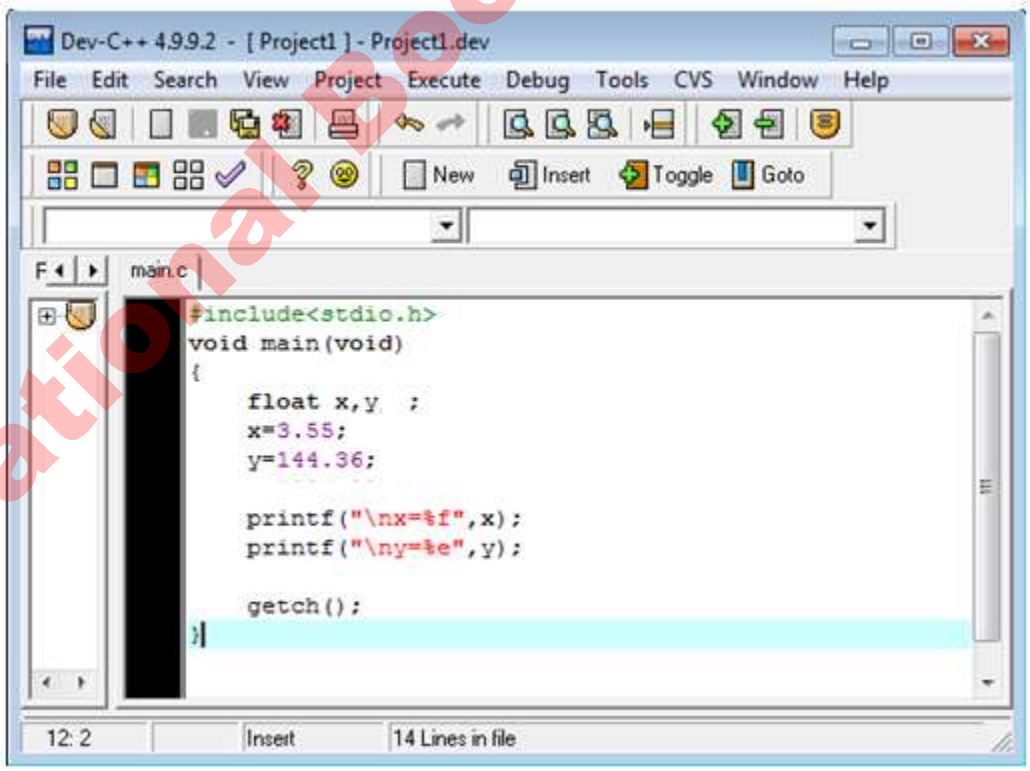


Fig.3-14 Using floating-point format specifiers in a program

The execution of the program is shown in Fig.3-15.

Fig.3-15 Printing floating-point numbers

- **The Character Format Specifier (%c)**

  The character format specifier, **%c**, is used to read or print a single character.

  The program in Fig.3-16 demonstrates the use of **%c** format specifier.

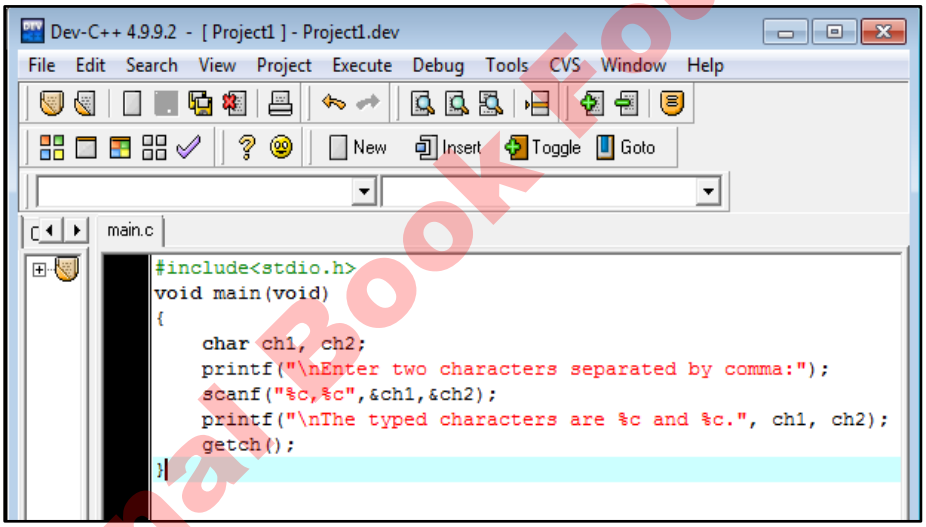


Fig.3-16 Using character format specifier in a program

The execution of the program is shown in Fig.3-17.

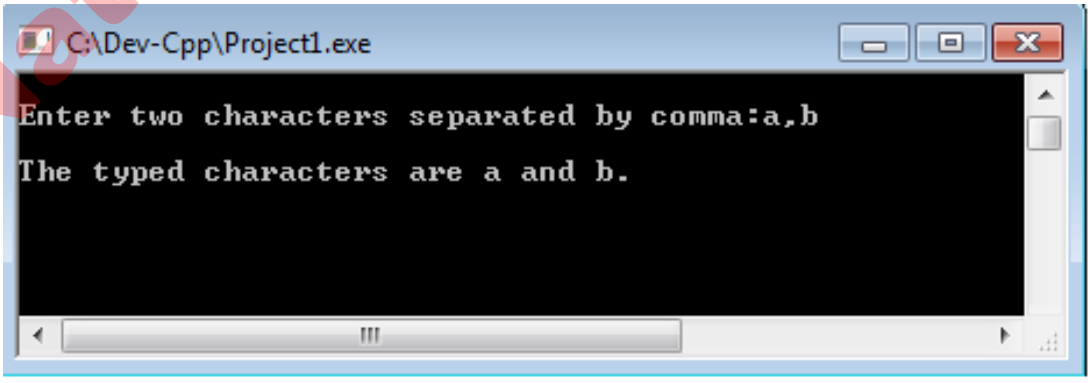


Fig.3-17 Reading and printing characters

3.1.5 ESCAPE SEQUENCE

The special characters used in C language to control printing on the output device are called escape sequences. These characters are not printed. These are used inside the control string. An escape sequence is a combination of a backslash (****) and a code character. The backslash is called the control character. A list of commonly used escape sequences is given in Fig.3-18 with their meanings.

Escape Sequence	Meaning
\a	Produces alert (bell) sound
\b	Moves cursor backward by one position
\n	Moves cursor to the beginning of next line
\r	Moves cursor to the beginning of current line
\t	Moves cursor to the next horizontal tabular position
\\	Produces a backslash
\’	Produces a single quote
\”	Produces a double quote

Fig.3-18 Escape sequences used in C language

The program in Fig.3-19, demonstrates the use of escape sequence.

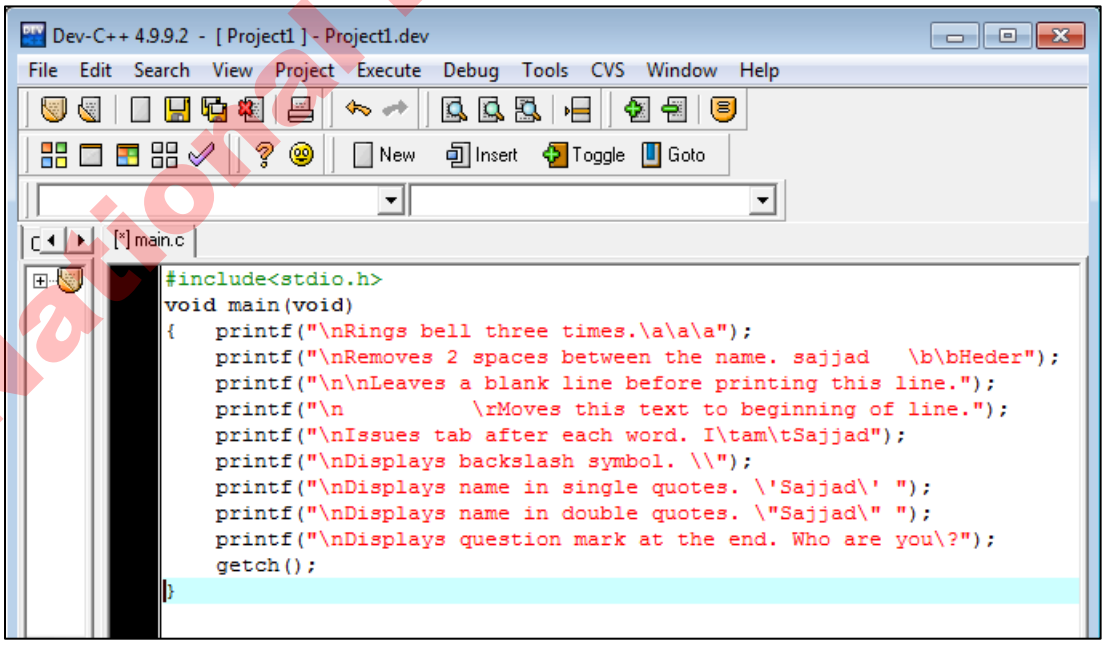


Fig.3-19 Using escape sequences in a program

The output of the program is shown in Fig.3-20

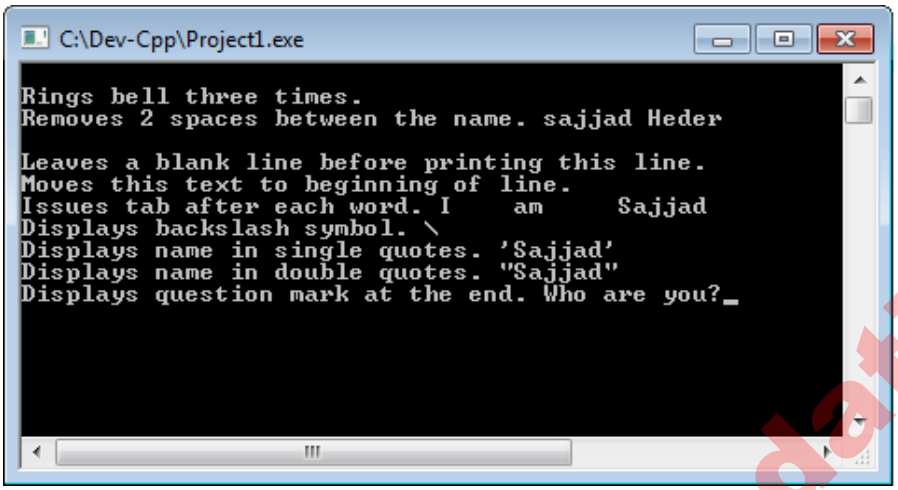


Fig.3-20 Printing strings using escape sequence

3.2 OPERATORS OF C LANGUAGE

Expressions consist of constants and variables combined together with operators. An operator is a symbol used to command the computer to perform a certain mathematical or logical operation. Operators are used to operate on data and variables.

The following types of operators are commonly used in C language.

- Arithmetic operators
- Assignment operators
- Relational operators
- Logical operators
- Increment and decrement operators

3.2.1 ARITHMETIC OPERATORS

Arithmetic operators are used to perform arithmetic operations that include addition, subtraction, multiplication, division and also to find the remainder obtained when an integer is divided by another integer.

TYPES OF ARITHMETIC OPERATORS

The types of arithmetic operators used in C programming are described with their operations in Fig.3-21

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (Modulus) Operator

Fig.3-21 Types of arithmetic operators

The program in Fig.3-22, demonstrates the use of arithmetic operators.

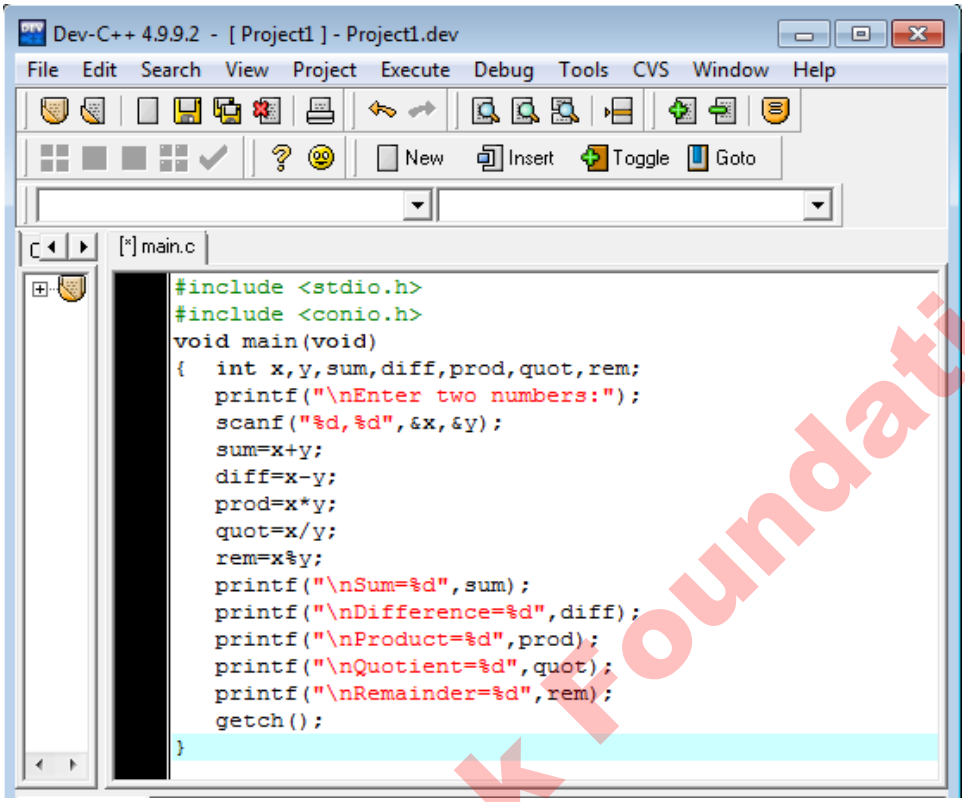


Fig.3-22 Using arithmetic operators in a program

In this program when **x/y** is performed it will give an integer result because the fractional part is truncated when the two operands are of type integer. Moreover, the remainder operator will give the remainder after dividing **x** by **y** when **x%y** is performed. The output of the program is shown in Fig.3-23.

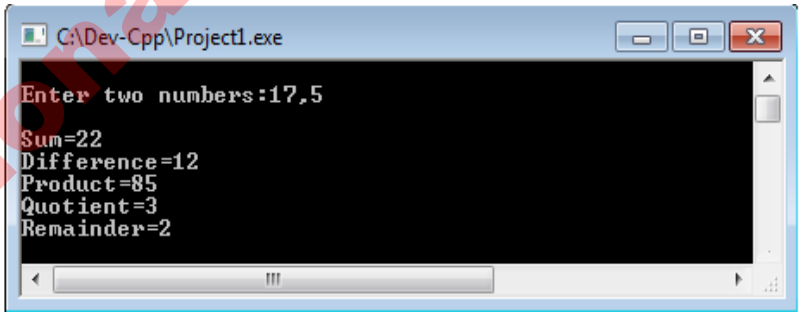


Fig.3-23 Use of arithmetic operators to perform calculations

3.2.2 ASSIGNMENT OPERATORS

Assignment operators are used to assign values to variables used in computer programs. C language provides three types of assignment operators. These are basic assignment operator, compound assignment operators and increment/decrement operators

Basic Assignment Operator

The basic assignment operator is **=**. This is used to assign value of an expression to a variable. It has the general form:

variable = expression

where expression may be a constant, another variable to which a value has previously been assigned or a formula to be evaluated. For example:

sum = a + b;

Compound Assignment Operators

In addition to **=**, there are a number of assignment operators unique to C. These include **+=**, **- =**, ***=**, **/=** and **%=**. Suppose **op** represents an arithmetic operator. Then, the compound assignment operator has the following general form to assign value of an expression to a variable.

variable op = expression

This is equivalent to:

variable = variable op expression

For example, consider the following statement:

sum = sum + n;

This assignment statement could be written using a compound assignment operator as:

sum += n;

The effect is exactly the same but the expression is more compact. Some more examples are:

sum - = n	is equivalent to	**sum = sum – n**
prod *= n	is equivalent to	**prod = prod * n**
a /= b	is equivalent to	**a = a / b**
a %= b	is equivalent to	**a = a % b**

3.2.3 RELATIONAL OPERATORS

Relational operators are used to compare two values of the same type. These are used in expressions when a decision is to be based on a condition. After evaluation of a relational expression, the result produced is either True or False. Relational operators are used in programming for decision making.

TYPES OF RELATIONAL OPERATORS

Six types of relational operators are available in C language. These are described in Fig.3-25.

Operator	Definition
= =	equal to
!=	not equal to

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Fig.3-25 Types of relational operators

The following are some examples of relational operators.

c>=a+b
x<5.3
n==20
count!=10

If **a** has the value 25, **b** has the value 10 and **c** has the value 28 then first expression is false since 28 is not greater than or equal to 35.

In the second expression, if **x** has the value 4.5, the expression is true because **x** is less than 5.3.

In the third expression, if **n** is equal to 20 then the expression will be true. For any value other than 20, the expression will be false.

In the last expression if **count** is any number other than 10 then the expression will be true. It will only be false when **count** is equal to 10.

3.2.4 LOGICAL OPERATORS

Logical operators are used for building compound conditions. We have seen before that a single condition is built using a relational operator in an expression. If we need to build more than one condition for some action to take place in programming, then we have to form compound condition.

TYPES OF LOGICAL OPERATORS

There are three types of logical operators. These are described in Fig.3-26.

Operator	Definition
&&	AND
||	OR
!	NOT

Fig.3-26 Types of logical operators

Logical AND (&&) Operator

It is used to form compound condition in which two relational expressions are evaluated. One relational expression is to the left and the other to the right of the operator. If both of the

relational expressions (conditions) are true then the compound condition is considered true otherwise it is false.

Syntax:

Expression1 && Expression2

Truth table for AND operator is shown here under:

Expression-1	Expression-2	Expression-1 && Expression-2
True	True	True
True	False	False
False	True	False
False	False	False

Example:

Consider the following compound condition:

(a>=1) && (a<=10)

This expression is considered true if both conditions **(a>=1)** and **(a<=10)** are true, that is, if the value of **a** is between 1 to 10.

In the next example, the compound condition is considered true if the value stored in **a** is greater than the value stored in **b** and the value stored in **c** is equal to 15.

(a>b) && (c= =15)

The following compound condition will check whether the character stored in variable **ch** is a lowercase letter or not.

(ch>='a') && (ch<='z')

Logical OR (||) Operator

Logical **OR** operator is also used to form a compound condition. Just like the logical **AND** operator, one relational expression is to the left and the other to the right of the **OR** operator. The compound condition is true if either of the conditions is true or both conditions are true. It is considered false only if both of the conditions are false.

Syntax:

Expression1* || *Expression2

Truth table for AND operator is shown here under:

Expression-1	Expression-2	Expression-1 || Expression-2
True	True	True
True	False	True
False	True	True
False	False	False

Example:

(n<10)||(n>25)

Suppose, the value of **n** is 5, then the expression will be considered true because one of the two conditions is true. If the value of **n** is 28 then also the compound condition will be true. If the value of **n** is 12 then the expression will be false since both conditions are false.

The next compound condition will be true if **a** is greater than **b** or **c** is equal to 10. It will also be true if both conditions are true, that is, **a** is greater than **b** and **c** is equal to 10. It will only be false if **a** is not greater than **b** and at the same time **c** is not equal to 10.

(a>b) || (c= =10)

Logical OR condition is used when we wish to perform an operations if one of the two conditions is true or both of the conditions are true.

Logical NOT (!) Operator

The logical NOT operator is used with a single expression (condition) and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words, it reverses the result of a single expression.

Syntax:

!*Expression1*

Truth table for AND operator is shown here under:

Expression	!Expression
True	False
False	True

For example, the expression:

!(a<b)

will be true if **a** is not less than **b.** In other words, the condition will be true if **a** is greater than or equal to **b**. The same condition can also be written as given below which is easy to understand.

(a>=b)

3.2.5 INCREMENT AND DECREMENT OPERATORS

Increment operator is **++** and decrement operator is **- -**. These are defined in Fig.3-24.

Operator	Definition
++	Increment by 1
- -	Decrement by 1

Fig.3-24 Increment and decrement operators

Examples:

++n and n++	are both equivalent to	**n = n + 1 (or n+=1)**
- - n and n- -	are both equivalent to	**n = n – 1 (or n-=1)**

When increment or decrement operator is written before the variable, it is known as **prefix** and when it is written after the variable, it is known as **postfix**.

In certain situations, **++n** and **n++** have different effect. This is because **++n** increments **n** before using its value whereas **n++** increments **n** after it is used.

As an example, suppose **n** has the value 3. The statement:

a = ++n;

will first increment **n** and then assigns the value 4 to **a**. But the statement:

a = n++;

will first assign the value 3 to **a** and then increments **n** to 4. In both cases **n** has the value 4.

The same rule applies to **- -n** and **n- -** as well.

3.2.6 DIFFERENCE BETWEEN ASSIGNMENT OPERATOR AND EQUAL TO OPERATOR

The assignment operator (=) is used to assign a value to a variable whereas the equal to operator (==) is used to compare two values of same data type.

For example:

```
a=5;
c=b;
z=x+y;
```

In the above statements, variable **a** is assigned the value 5, **c** is assigned the value stored in variable **b** and **z** is assigned the sum of values stored in variable **x** and **y.**

The relational operator (= =) is used to build a condition based on which computer takes some action.

For example:

```
a= =1
c= = a+b
```

In the first condition, if the value of **a** is equal to 1 then the condition is true otherwise it is false. In the second condition, the equal to operator is used to check whether the value of **c** is equal to the sum of **a** and **b**. If it is equal then the condition is true otherwise it is false.

3.2.7 DIFFERENCE BETWEEN UNARY AND BINARY OPERATORS

The operators that work with a single operand are known as unary operators whereas operators that work with two operands are known as binary operators. Unary operators are -, ++, -- and the logical operator ! (NOT). Binary operators are -, +, *, \, % and logical operators && (AND) and || (OR).

Some examples of unary operators are:

```
a= -b;
k++;
- -x;
```

Some examples of binary operators are:

```
a=b+c;
z=x*y;
k=d%e;
```

3.2.8 CONDITIONAL (TERNARY) OPERATOR

A conditional operator is a decision-making operator. It has the following form.

```
condition? expression1 : expression2;
```

When this statement is executed, the condition is evaluated. If it is true, the entire conditional expression takes on the value of expression1. If it is false, the conditional expression takes on the value of expression2. The entire conditional expression takes on a value and can therefore be used in an assignment statement. Consider the following example.

```
a = (k>15)? x*y : x+y;
```

This statement will assign the product of **x** and **y** to the variable **a**, if **k** is greater than 15, otherwise **a** will be assigned the sum of **x** and **y**. This expression is equivalent to the following if-else statement which will be explained in the next unit.

```
if (k>15)
        a = x*y;
else
        a = x+y;
```

Some programmers may prefer to use the above if-else statement rather than using the conditional operator because it is easy to understand.

The program in Fig.3-27, demonstrate the use of conditional operator for finding the larger of two numbers.

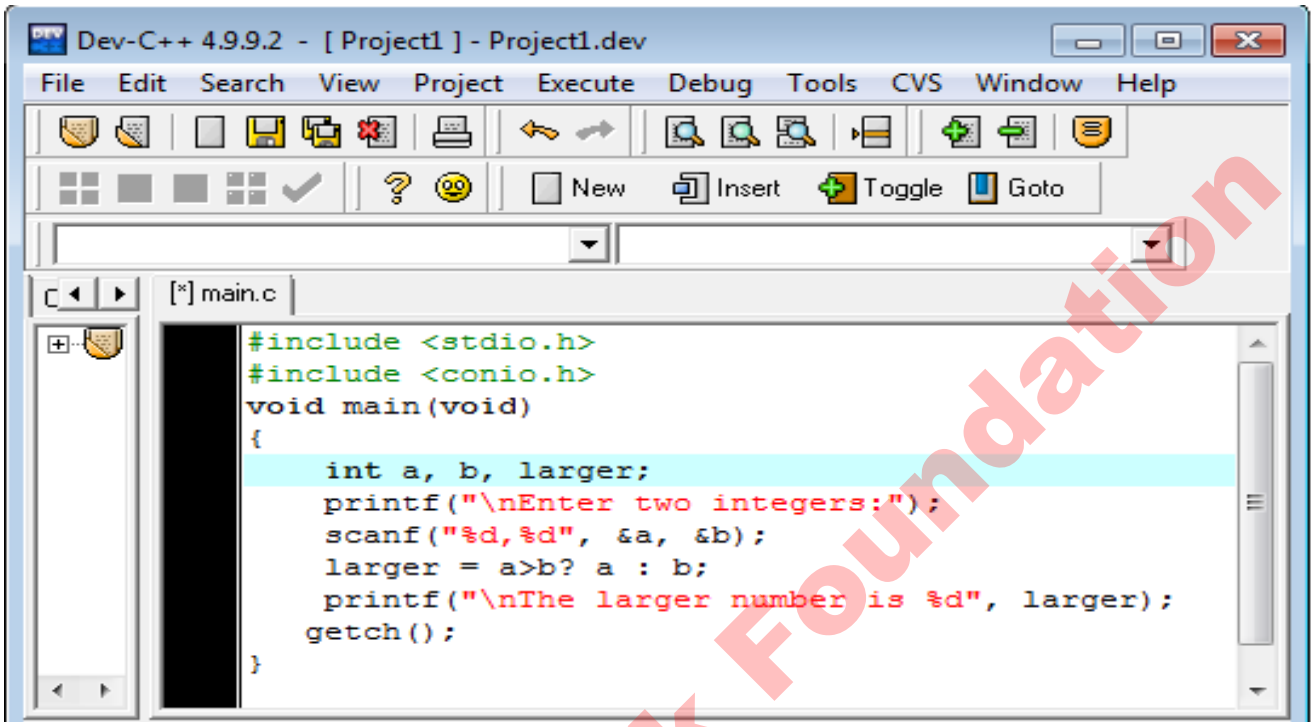


Fig.3-27 Program to find larger of two numbers

The execution of the program is shown in Fig.3-28.

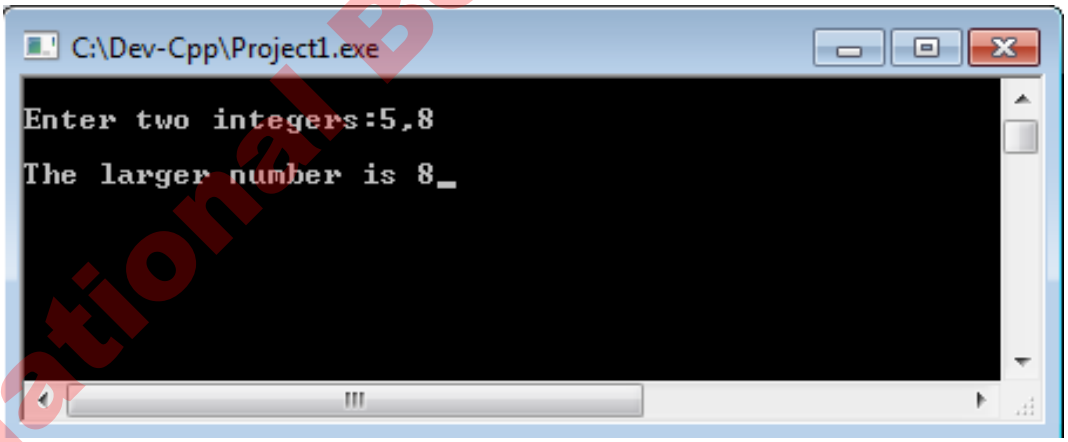


Fig.3-28 Finding larger of two numbers

3.2.9 ORDER OF PRECEDENCE OF OPERATORS

Order of precedence of operators is the rule that specifies the order in which operations are to be performed in an expression. The order of precedence is similar to that used in algebraic formulas.

The order of precedence of operators is shown in Fig.3-29. The operator that has the highest precedence is written at the top and the one with the lowest precedence is written at the bottom.

Precedence	Operator	Description
1.	++, --	Increment and Decrement (Prefix or Postfix)
2.	*, /, %	Multiplication, Division and Remainder
3.	+, -	Addition and Subtraction
4.	<, <=, >, >=	Relational Operators
5.	= =, !=	Equal to and Not Equal to
6.	!	Logical NOT
7.	&&	Logical AND
8.	||	Logical OR
9.	=, *=, /=, %=, +=, -=	Assignment Operators

Fig.3-29 Order of precedence of operators

As shown in Fig.3-29, the increment/decrement operators are performed first, then arithmetic operators, then relational operators, after that logical operators and at the end assignment operators.

If an expression contains two or more operators of the same precedence then the operations are performed from left to right in the order in which they appear.

To illustrate this, consider the expression

7 * 10 – 5 % 3 * 4 + 9 / 3

The leftmost multiplication is performed first, giving the intermediate result

70 – 5 % 3 * 4 + 9 / 3

The next high-priority operation encountered is % which gives

70 – 2 * 4 + 9 / 3

Multiplication is performed next, giving

70 – 8 + 9 / 3

The last high-priority operation, division is performed next. It gives

70 – 8 + 3

Next the low-priority operations are performed in the order in which they occur, from left to right. The subtraction is thus performed first giving

62 + 3

and then the addition is carried out, giving the final result 65.

The standard order of evaluation can be modified by using brackets to enclose part of an expression. The part of the expression within brackets is first evaluated in the standard manner and then the result is combined to evaluate the complete expression. For example in the expression,

a / (b + c)

b+c will be performed first and then **a** will be divided by the sum of **b** and **c**.

If the brackets are nested, that is, if one set of brackets is contained within another, the computations in the innermost brackets are performed first.

Key Points

- The **printf()** functions is used to print text and values on the screen in a specified format.
- The **scanf()** function is used to get values into variables from the keyboard during execution of a program.
- A semicolon is entered at the end of each C statement to terminate it.
- A format specifier tells about the data type, field-width and the format according to which a value is to be printed or read from an input device.
- Escape sequence is a combination of backslash (\) and a code character to control printing of data on the screen.
- Assignment operator is used to assign a value to a variable.
- Relational operator is used to compare two values of the same type. It is used in an expression when a decision is to be based on a condition in a program.
- Logical operator is used for building compound condition.
- The order of precedence of operators is the rule that specifies the order in which operations are to be performed in an expression.

Exercise

Q1. Select the best answer for the following MCQs.

i. Which function is used for output purpose in C language?

A. printf() B. scanf()
C. input() D. getch()

ii. Which character terminates a C statement?
A. Colon B. Semicolon
C. Period D. Comma

iii. Which format specifier is used to print or read a floating-point value?
A. %d B. %i
C. %f D. %e

iv. Which escape sequence is used to move cursor to the beginning of current line?
A. \a B. \r
C. \n D. \b

v. Which of the following is an arithmetic operator?
A. % B. <=
C. && D. +=

vi. Which of the following is a logical operator?
A. % B. <=
C. && D. +=

vii. Which statement is equivalent to "k = k + a;"?
A. k+=a; B. k=+a;
C. k++a; D. k=a++;

viii. Which of the following is an increment operator?
A. + B. +=
C. ++ D. =+

ix. Which of the following operator has the highest precedence?
A. && B. <=
C. = D. *

x. What will be the output of the expression, 5+3*3-1?
A. 16 B. 13
C. 23 D. 12

Short Questions

Q2. Give short answers to the following questions.

i. Why format specifier is used? Explain with examples.

ii. Why escape sequence is used? Explain with examples.

iii. What is the purpose of printf() function? Explain with an example.

iv. Differentiate between printf() and scanf() functions.

v. Evaluate the following expressions.

a) 7+5*(3+4)
b) 100/10/4
c) 50%13%3
d) 30/7*3-6

vi. What will be the output of the following program?

```
# include <stdio.h>
void main(void)
{
        int x,y,z1,z2,z3,z4;
        x=17;
        y=5;
        z1=x/y;
        printf("\nz1=%d",z1);
        z2=x%y;
        printf("\nz2=%d",z2);
        z3=++x;
        printf("\nz3=%d",z3);
        z4=y++;
        printf("\nz4=%d",z4);
}
```

vii. What will be the output of the following program?

```
# include <stdio.h>
void main(void)
{
        int b;
        float a,c,d,e,f;
        a=14.84;
        b=7;
        c=a-b;
        printf("\nc=%f",c);
        d=a/b;
        printf("\nd=%f",d);
        e=a-b*3;
        printf("\ne=%f",e);
```

```
    f=(a+b)/2;
    printf("\nf=%f",f);
}
```

Extensive Questions

Q3. Describe how basic and compound assignment operators are used?

Q4. Describe the functions of the following operators?

i) Relational operators

ii) Logical operators

iii) Conditional operator

Q5. Write a program that reads three numbers and prints their sum, product and average.

Q6. Write a program that reads the length and width of a rectangle and prints its area.

Q7. Write a program that reads the length of one side of a cube and prints its volume.

Q8. Write a program that reads temperature in Celsius, converts it into Fahrenheit and prints on the screen.

4

CONDITIONAL CONTROL STRUCTURE

After completing this lesson, you will be able to:

- Define a control statement
- Define a conditional statement
- Know the structure and use of **if** statement
- Know the structure and use of **if-else** statement
- Know the structure and use of **else-if** statement
- Know the structure and use of **switch** statement
- Know the role of **break** in **switch** statement
- Know the use of nested selection structures
- Differentiate among all the selection structures
- Write codes for flowcharts discussed in Unit 1

Reading

UNIT INTRODUCTION

In programming, it is required to make choices and decisions based on conditions. It is important to know how to program the computer to implement choices and decisions. Programming languages provide control structures to achieve this. In this unit, students will learn **if**, **if-else**, **else-if** and **switch** statements that implement control structure in C language programs.

4.1 CONTROL STRUCTURE

Everybody makes different decisions in the daily life. For example, if my friend is at home then I will visit him. If the weather is good tomorrow, I will go for picnic. Similarly, computer programmers also have to make decisions and perform some operations depending on the user input to the program. Control structures are used in programs to implement decisions.

4.1.1 CONROL STATEMENT

A control statement is an instruction which determines the sequence of execution of other statements. In other words, it controls the flow of execution of program statements.

4.1.2 CONDITIONAL STATEMENT

A conditional statement is an instruction in a programming language that contains a condition. When a conditional statement is executed, first the condition is evaluated and then based on the result (true or false), a particular statement or a set of statements is executed. Conditional statements of C language are **if**, **if-else**, **else-if** and **switch** statements.

4.1.3 STRUCTURE OF IF STATEMENT

The **if** statement has the following general form.

```
if (condition)
{
    Block of statements
}
```

When this statement is executed, the condition is evaluated. If the condition is true then the block of statements within the braces will be executed. If the condition is false then the block of statements within the braces will be skipped and the control will be transferred to the next statement if any exists. If there is only one statement to be executed if the condition is true then braces are not required.

4.1.4 USE OF IF STATEMENT

Program 1: The program in Fig.4-1, demonstrates the use of **if** statement.

main.c

```
# include <stdio.h>
# include <conio.h>
void main(void)
{
    int marks;
    printf("\nEnter your marks:");
    scanf("%d",&marks);
    if (marks>32)
    {
        printf("\nCongratulations");
        printf("\nYou have passed");
    }
    getch();
}
```

Fig.4-1 Program to demonstrate the use of if statement

- When **if** statement of this program is executed, the condition inside the brackets is evaluated.
- If the condition is true, that is, the marks entered are greater than 32, then the two statements following the keyword **if** are executed.
- If the condition is false, that is, the marks entered are less than 33, then the following two statements will be skipped and the program will terminate and there will be no output.
- The output of the program is shown in Fig.4-2, if the marks entered are more than 32.

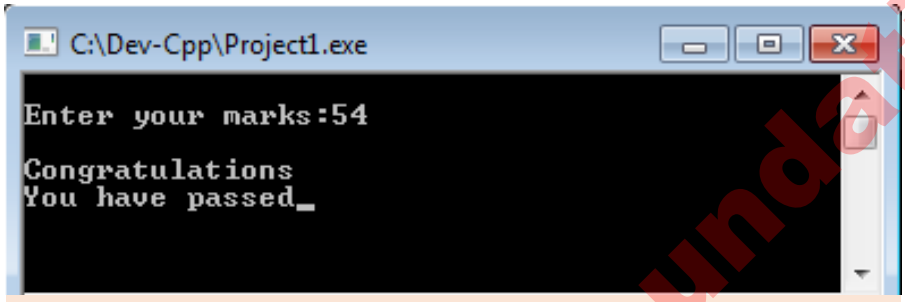


Fig.4-2 Output of Program 1

Program 2: The program in Fig.4-3 prints square of a number.

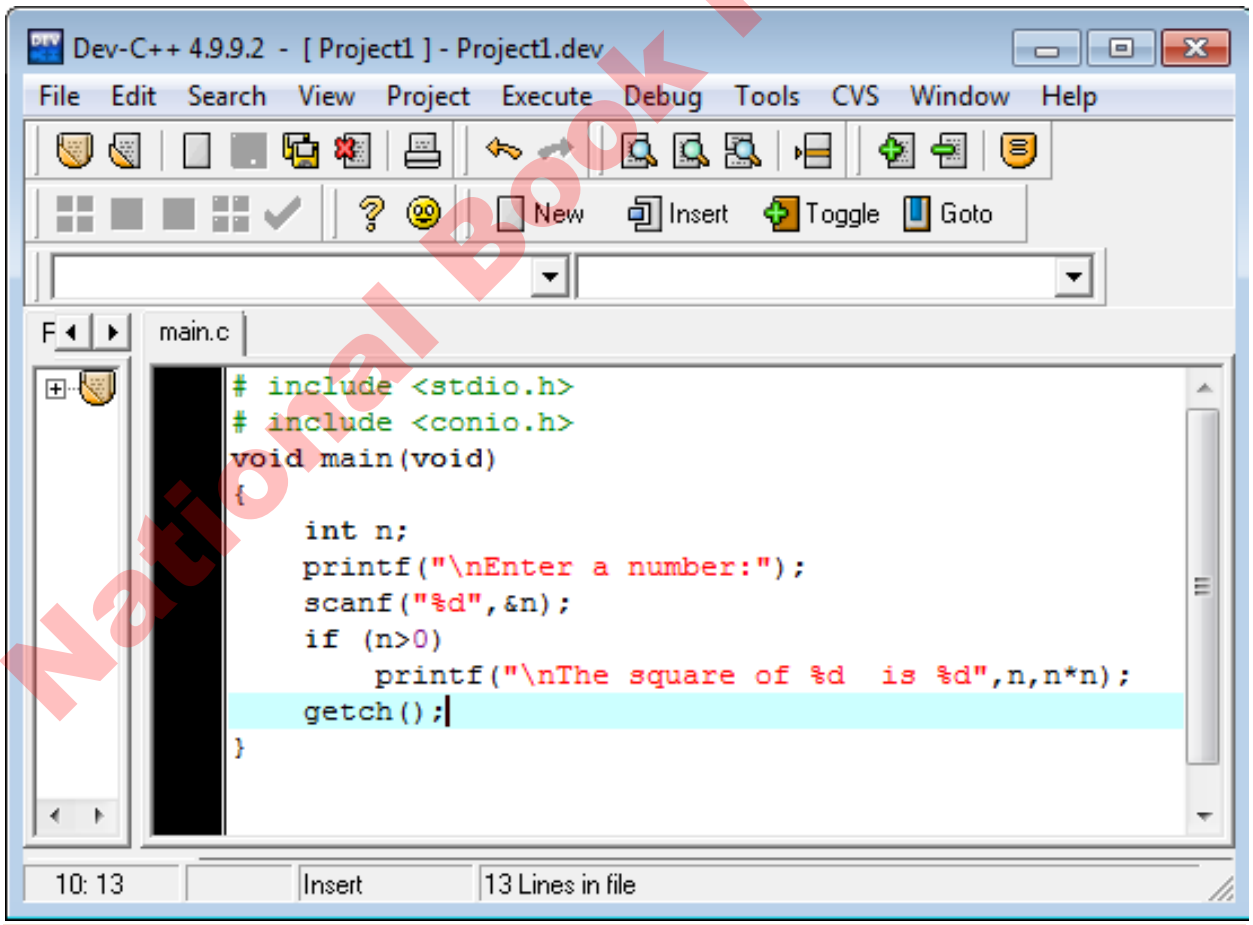


Fig.4-3 Program to print square of a number

- When the above program is executed, it will prompt the user to enter a number.
- If the user enters a number greater than zero, it will print the square of the number as shown in Fig.4-4.

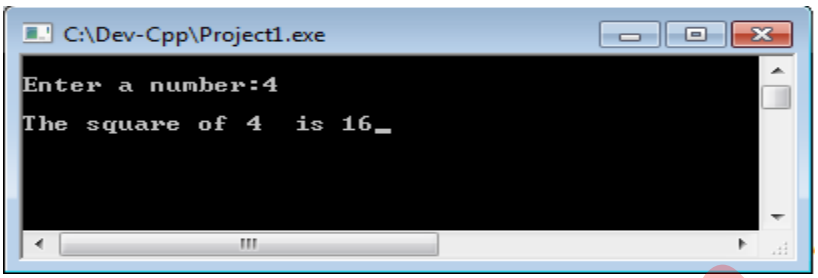


Fig.4-4 Output of Program 2

4.1.5 STRUCTURE OF IF-ELSE STATEMENT

The **if-else** statement is used in situation where some code is to be executed if a condition is true and some other code is to be executed if the condition is false.

The **if-else** statement has the following general form.

```
if (condition)
{
      Block of statements
}
else
{
      Block of statements
}
```

- When **if-else** statement is executed, the condition is evaluated.
- If the condition is true then the block of statements following **if** will be executed and the block of statements following **else** will be skipped.
- If the condition is false then the block of statements following **if** will be skipped and the block of statements following **else** will be executed.
- If a single statement is to be executed after **if** or **else** then braces are not required.

4.1.6 USE OF IF-ELSE STATEMENT

Program 3: The program in Fig.4-5, reads marks and prints the message whether the student passed or failed.

main.c

```
# include <stdio.h>
# include <conio.h>
void main(void)
{
     int marks;
     printf("\nEnter your marks:");
     scanf("%d",&marks);
     if (marks>32)
     {
          printf("\nCongratulations");
          printf("\nYou have passed");
     }
     else
          printf("\nYou have failed");
     getch();
}
```

Fig.4-5 Program to demonstrate the use of if-else statement

- In this program, if the marks entered are above 32 then the statements following **if** are executed and the message shown in Fig.4-6 (a) will be printed.
- If the marks entered are below 33 then the statement following **else** are executed and the message shown in Fig.4-6 (b) will be printed.
- Also note that, there is a single statement to be executed if the condition is false. Therefore, braces are not required after **else**.

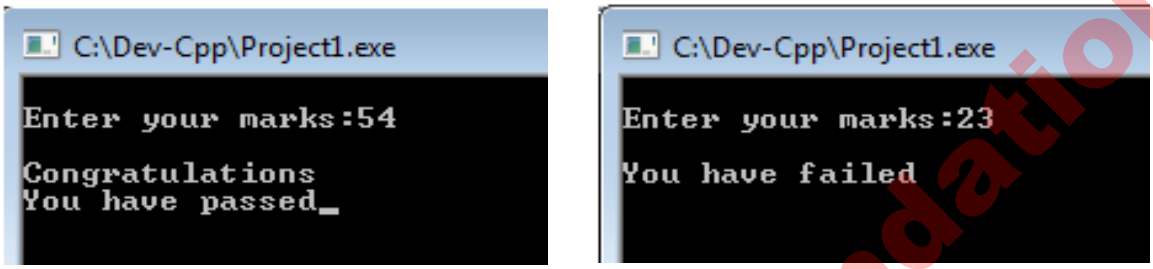


Fig.4-6 (a) Output when marks are grater than 32
(b) Output when marks are less than 33

Program 4: The program in Fig.4-7 reads a number from the keyboard and prints the message whether it is an even or an odd number.

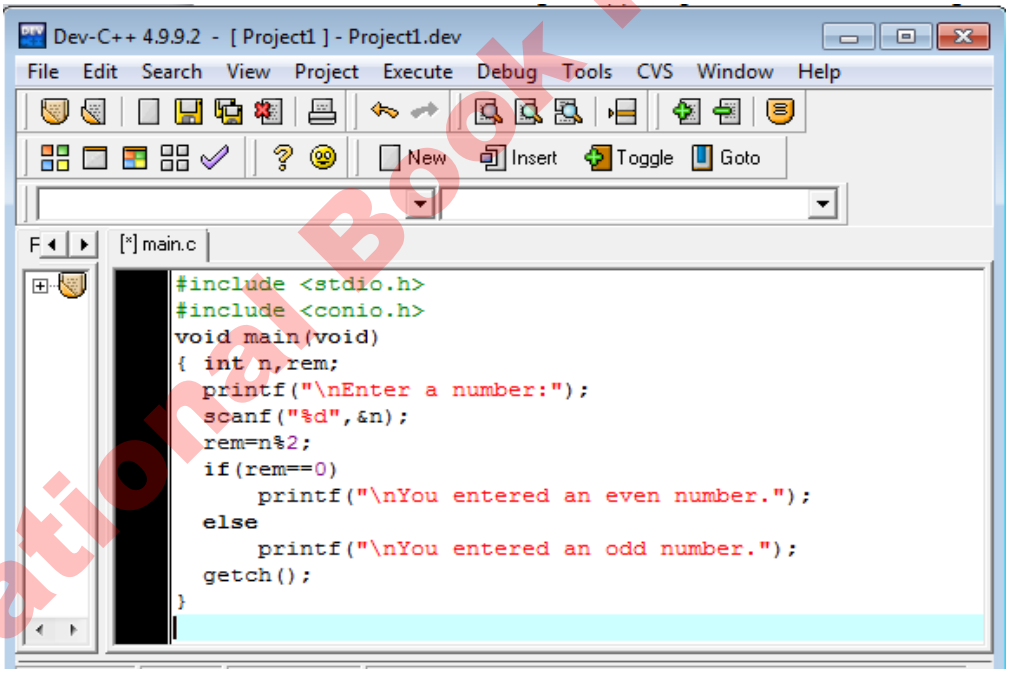


Fig.4-7 Program to demonstrate the use of if-else statement

- When this program is executed it prompts the user to enter the value of **n**.
- The remainder operator **%** is used to divide the number by 2 and store the remainder in variable **rem**.
- The value of **rem** is checked. If it is equal to 0 then the number **n**, is an even number as shown in Fig.4-8 (a).

- If the value of **rem** is not equal to 0, in other words, it is equal to 1 then the number **n** is an odd number as shown in Fig.4-8 (b).

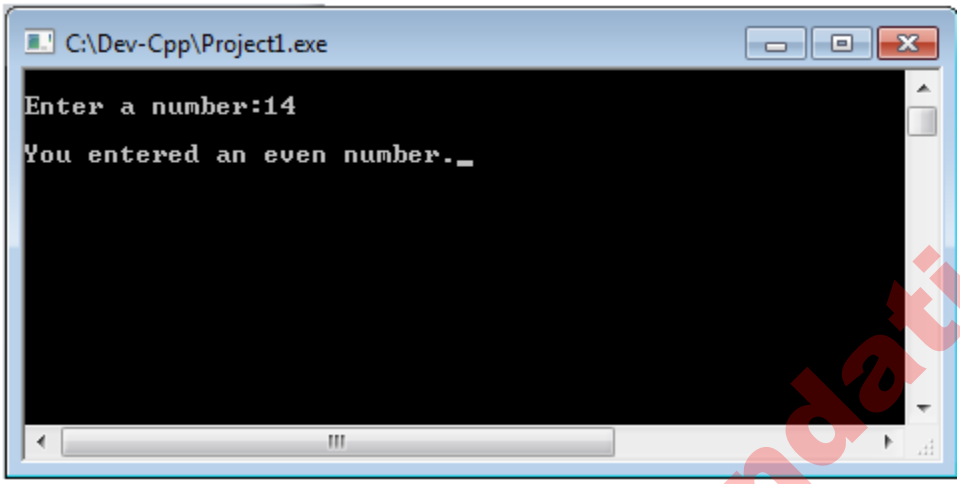


Fig.4-8 (a) Execution of program when value of n is 14.

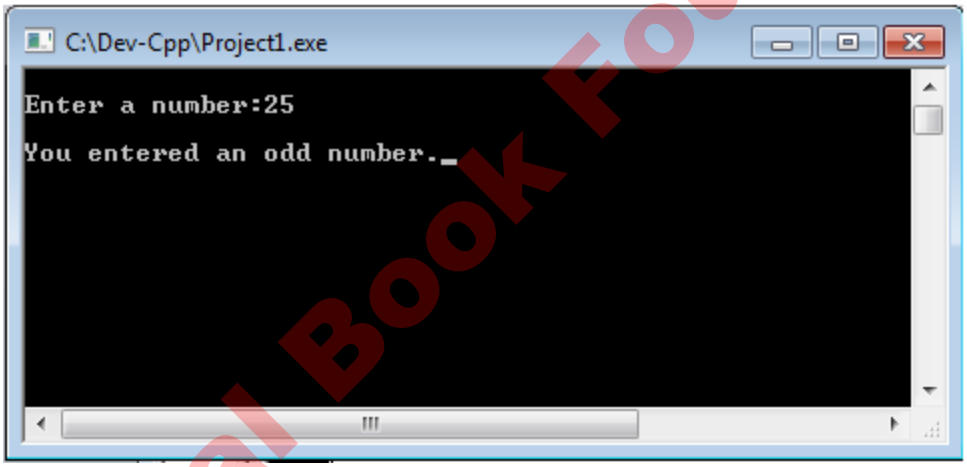


Fig.4-8 (b) Execution of program when value of n is 25.

4.1.7 STRUCTURE OF IF-ELSE-IF STATEMENT

The **else-if** is a type of conditional statement that combines more than two conditions. It allows the programmer to make a decision based on several conditions.

The **else-if** statement has the following general form.

```
if(condition-1)
{
        Block of statements
}
else if(condition-2)
{
        Block of statements
}
```

```
else if(condition-3)
{
        Block of statements
}
                .
                .
                .
else
{
        Block of statements to be executed
        when none of the conditions is true.
}
```

- When this statement is executed, condition-1 is evaluated, if it is true then the block of statements following **if** is executed and if it is false, the next condition is evaluated.
- If any condition is true then the following block of statements is executed.
- If none of the conditions is true then the block of statements following **else** is executed automatically.
- If a single statement is to be executed after **if**, **else if** or **else**, instead of a set of statements then the braces are not required.

4.1.8 USE OF ELSE-IF STATEMENT

Program 5: The program in Fig.4-9 performs an arithmetic operation based on the choice of user.

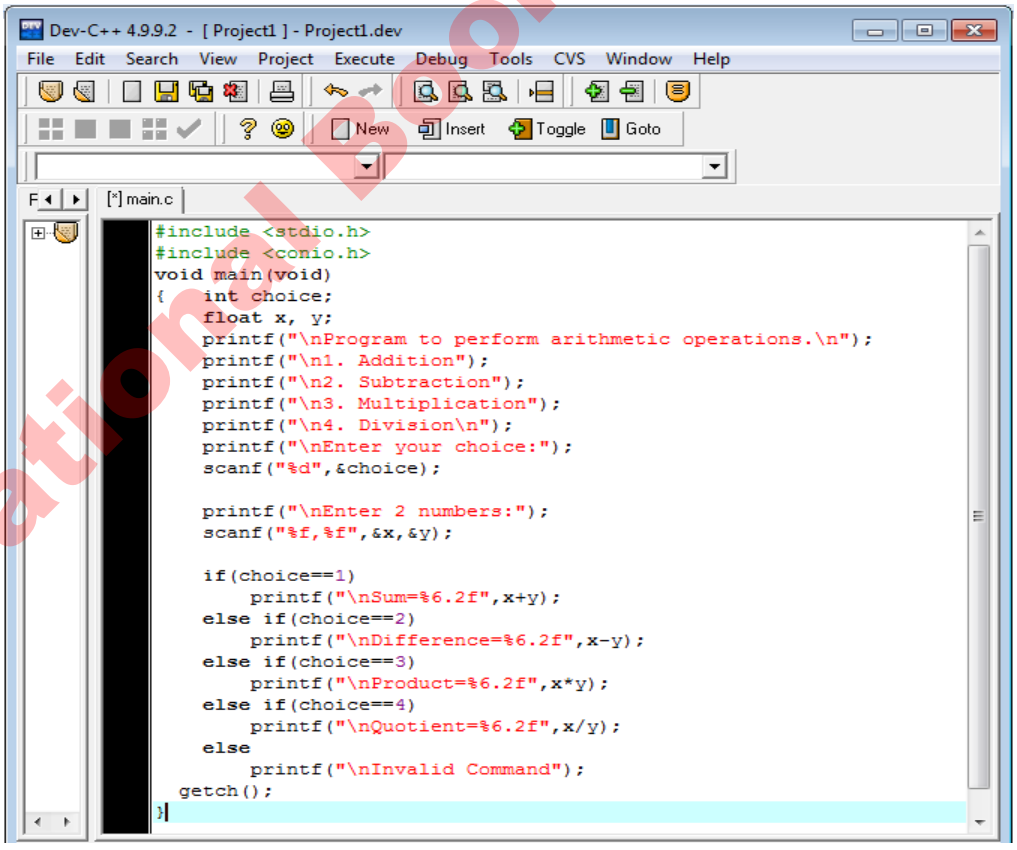


Fig.4-9 Program to demonstrate the use of else-if statement

- When this program is executed, it will display four choices and prompt the user to enter his choice. User's choice will be stored in the variable **choice**.
- After this, it will again prompt the user to enter two numbers which will be stored in variables **x** and **y**.
- Now, the **if-else-if** statement will be executed. First condition will be checked first, if it is true, that is, the value stored in the variable **choice** is **1** then sum of **x** and **y** will be evaluated and printed.
- If the first condition is false then the second condition will be checked and if it is true, then the difference **(x-y)** will be evaluated and printed and so on.
- If all the conditions are false then the statement following **else** will the executed and the message "**Invalid Command**" will be printed.

In this program, the format specifier **%6.2f** is used for printing floating-point value. Here, 6 is the total field width for printing the number and reserves 6 spaced for printing it. The digit 2 means 2 digits are to be printed after the decimal point. If the floating-point number to be printed requires less than 6 spaces then it will be right justified and extra spaces will appear at the left side of the number.

The execution of this program is shown in Fig.4-10, when user choice is 3.

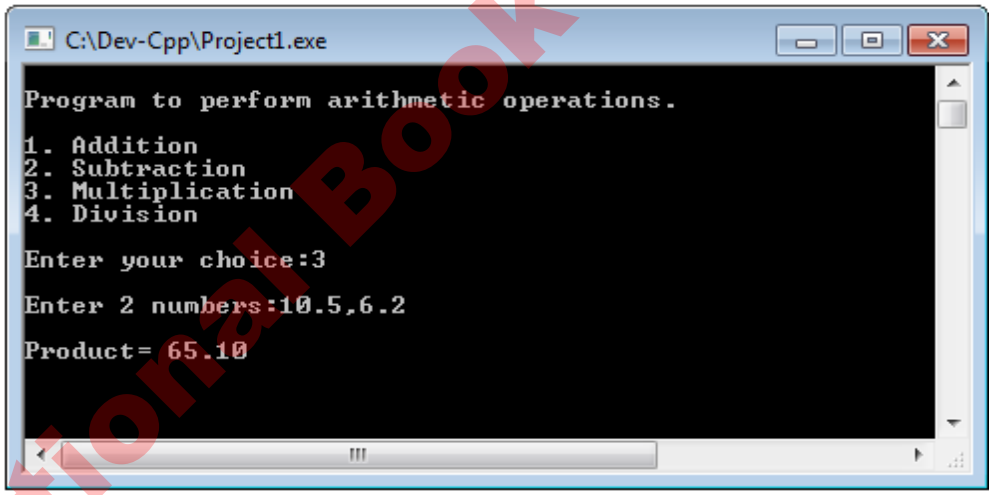


Fig.4-10 Execution of Program 5

Program 6: The following program reads marks of a student and prints his letter grade according to the following scheme.

Marks	Grade
80~100	A
70~79	B
60~69	C
50~59	D
Below 50	F

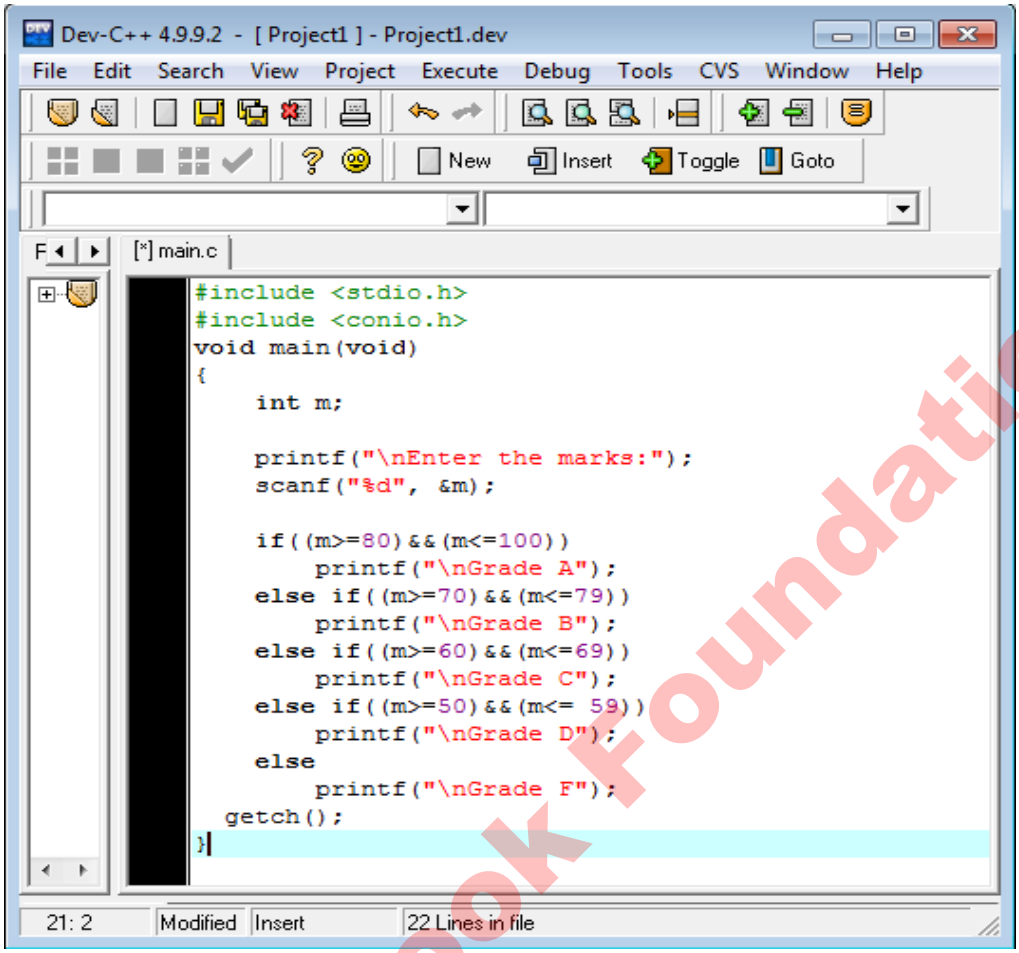


Fig.4-11 Program to find Grade

- When the above program is executed, it will prompt the user to enter marks.
- It will check in which range the marks fall and then accordingly print the grade.
- If none of the conditions is true then the statement following **else** will be executed and it will print the message “Grade F”. The execution of the program is shown in Fig.4-12, if the marks entered by the user are 66. Since 66 falls in the range 60 to 69, so the grade displayed is C.

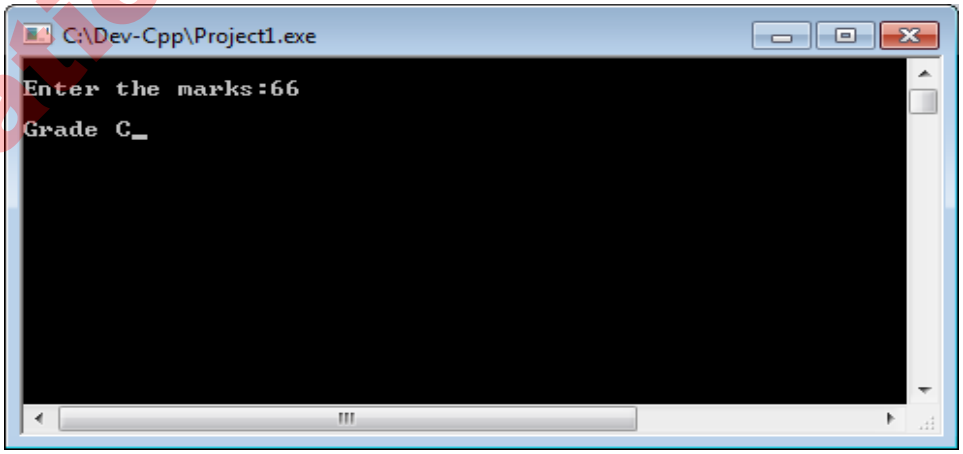


Fig.4-12 Execution of Program 6

4.1.9 SWITCH STATEMENT

The **switch** statement has the following general form.

```
switch (expression)
{
 case const-1:
        statements;
        break;
 case const-2:
        statements;
        break;
        .
        .
        .
 default:
        statements;
}
```

- The **switch** statement is similar to the **else-if** statement. It is used when multiple choices are given and one choice is to be selected.
- When **switch** statement is executed, the expression is evaluated. Based on the result of expression one of the cases in the **switch** statement is executed. The result of expression is compared with the constant values given after the keyword **case**. If the result matches the constant value after any **case** then the statements under that case are executed.
- In switch statement, it is allowed to use a variable within the parenthesis instead of an expression based on which statements under a **case** can be executed.
- The purpose of **break** statement is to exit the body of the **switch** statement after executing the statements under a **case** and transfer control to the first statement following the end of the **switch** statement.
- If no **case** is matched then the statements under the **default** keyword are executed. Its use is optional. If it is not used then the control exits from the body of the **switch** statement and goes to the first statement following the end of the **switch** statement.
- The expression should be of type **int**, **char** but not **float**.

Program 7: The program in Fig.4-13 uses a variable of type integer as switch variable.

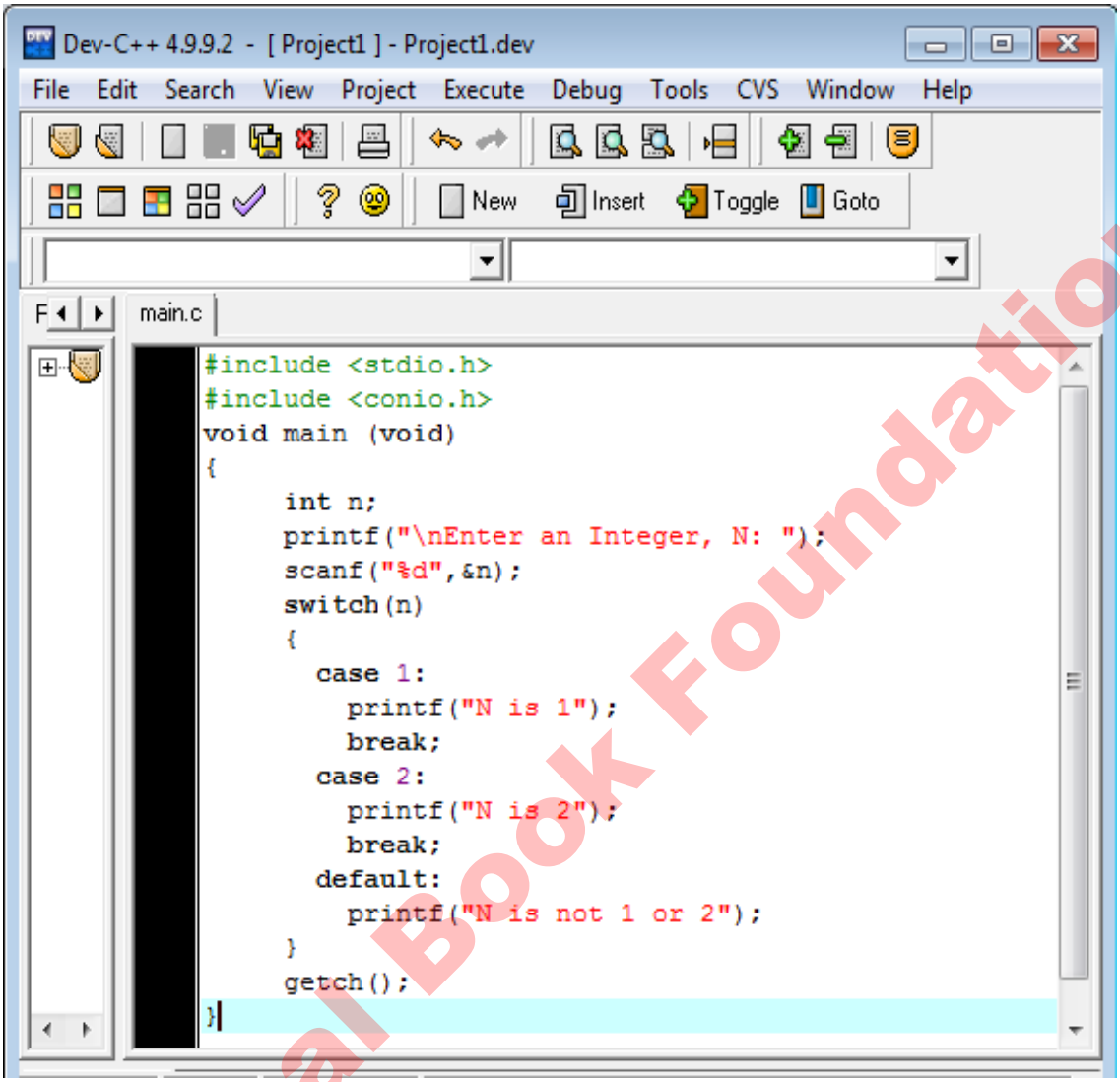


Fig.4-13 Program to demonstrate use of switch statement

- When this program is executed, the switch variable must have an integer value. The value of switch variable **n** is compared with the constant values following the **case** keyword and if it matches, control is transferred to the statements following that particular **case**.
- If the switch variable does not match any of the **case** constants, control goes to the keyword **default** which is at the end of the switch statement.
- Notice the use of **break** statement in this program. It terminates the **switch** statement when the body of the statements in a particular **case** has been executed.

Program 8: The program in Fig.4-14 prompts the user to enter a grade and prints appropriate message. It uses a variable of type character as switch variable.

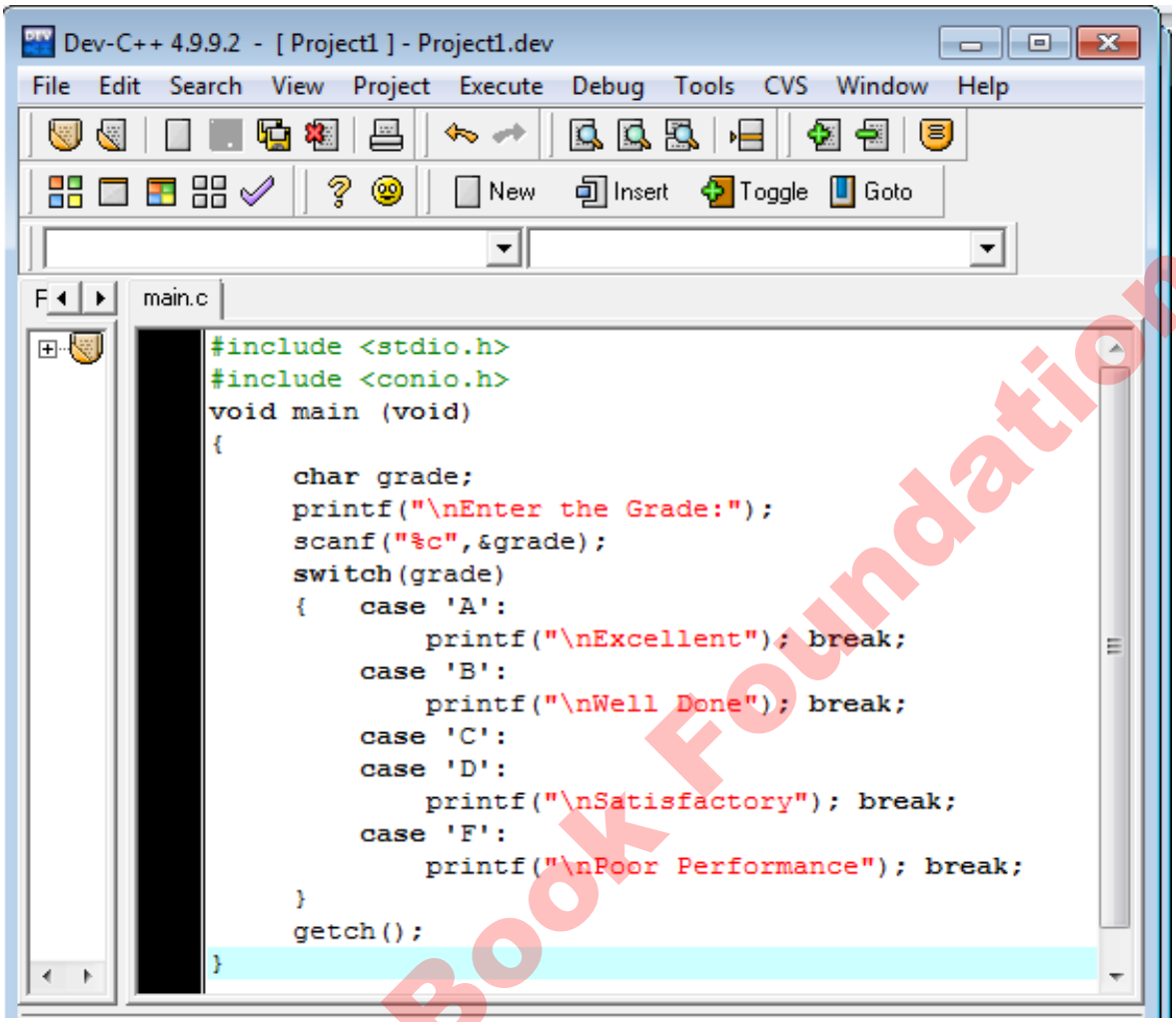


Fig.4-14 Program to print a message based on grade

- This program prompts the user to enter a grade, that is, '**A**', '**B**', '**C**', '**D**' or '**F**' and stores it in the variable **grade**. It will print the message "**Excellent**" for grade '**A**' and "**Well Done**" for grade '**B**'.
- If the user enters grade '**C**' or '**D**', the same message "**Satisfactory**" will be printed. In the absence of statements after a **case**, the control falls right through one case to the case below and this makes it easy for several values of switch variable to execute the same code.
- Finally, if the user enters grade '**F**', the program will print the message "**Poor Performance**".
- In this program there is no **default** keyword. Therefore, the whole switch statement simply terminates when there is no match.

Program 9: The program in Fig.4-15 creates a simple calculator to perform addition, subtraction, multiplication or division.

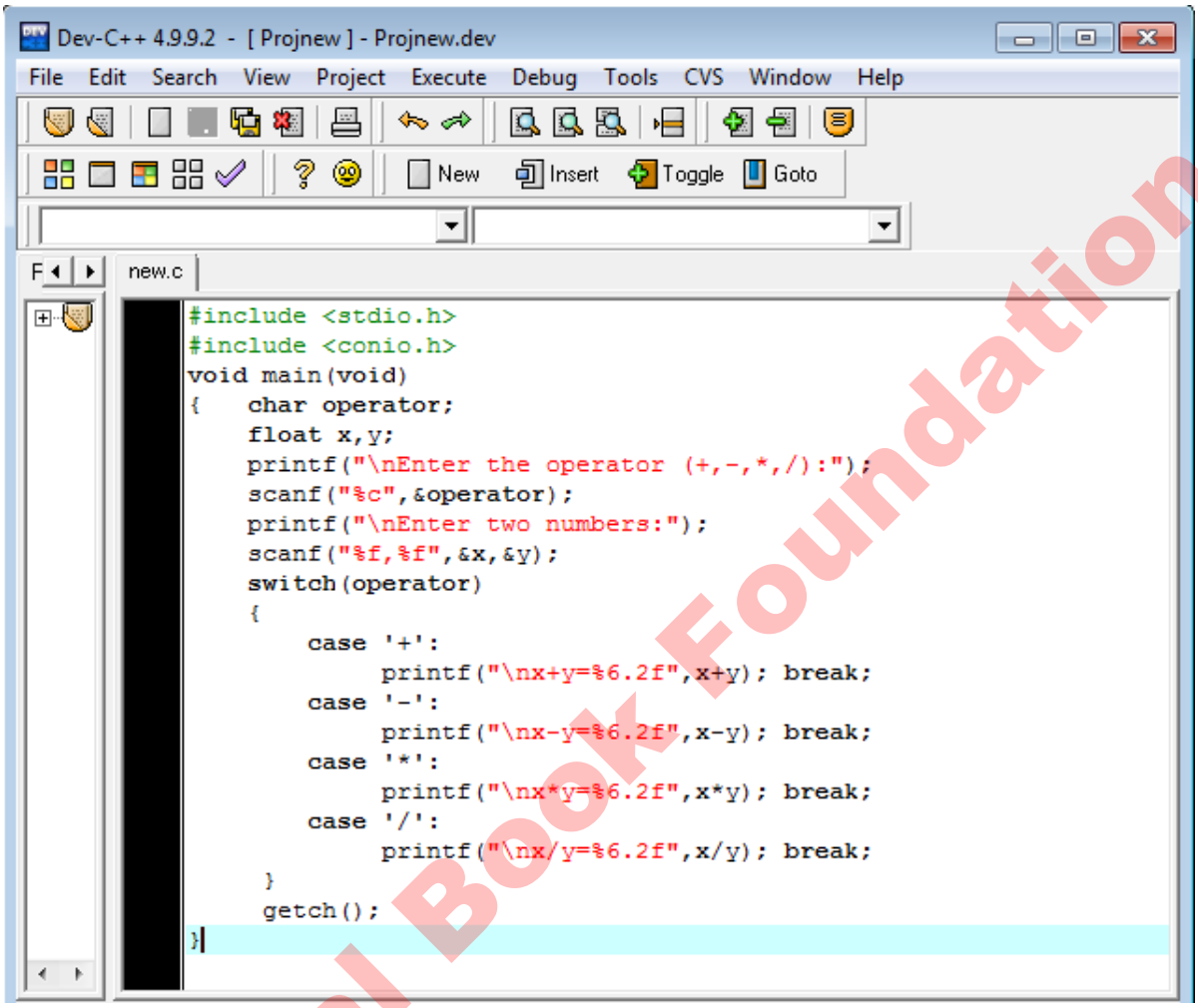


Fig.4-15 Program to create a simple calculator

- When the above program is executed, it will ask the user to enter the type of operation to be performed on two numbers. Then it will ask for two numbers to be entered.
- Based on the operator, it will perform addition, subtraction, multiplication or division and print the result.

Advantage and Limitation of switch Statement

- The **switch** statement allows a variable to be compared against a list of constant values. When there is a match to a **case**, the statements following that **case** will execute until a **break** statement is reached. This makes the logic of program simple and easy to understand.
- The **switch** statement has a limitation. It is not allowed to use relational operators in the expression of **switch** statement. For example, to check for passing marks, the condition,

(marks>32) cannot be used in **switch** statement. It is also not possible to check for a range such as ((marks>=70)&&(marks<=80)) in a **switch** statement, for this, the programmer has to use **if**, **if-else** or **else-if** statement.

Using Conditional Operator in Program

Program 10: The program in Fig.4-16 reads marks and prints the message “PASS” or “FAIL”, using conditional operator instead of **if-else** statement. Conditional operator was explained in the previous unit.

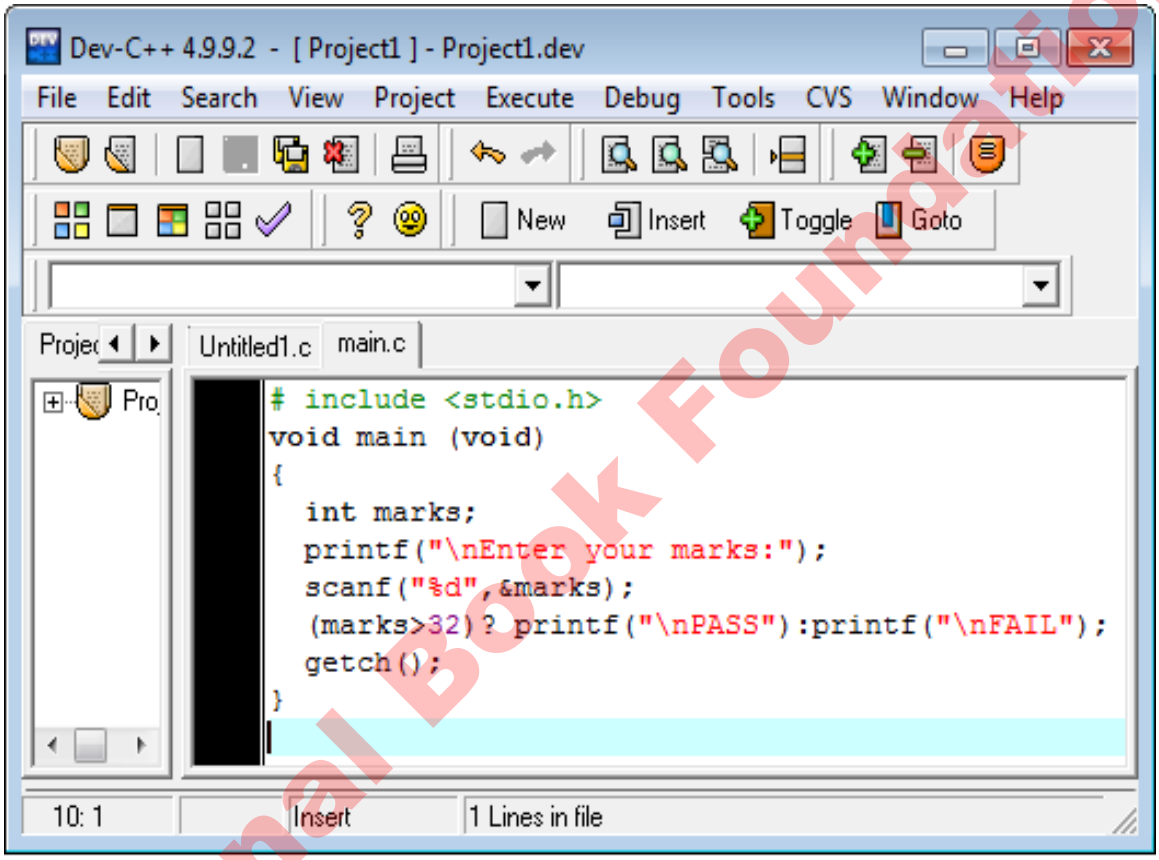


Fig.4-16 Using conditional operator in a program

- When this program is executed, it will ask the user to enter the marks and store it in the variable **marks**.
- The condition **(marks>32)** will be evaluated.
- If the condition is true, that is, marks are greater than 32 then the first print statement will be executed and the message “PASS” will be printed.
- If the condition is false then the second print statement will be executed and the message “FAIL” will be printed.

Execution of the program is shown in Fig.4-17.

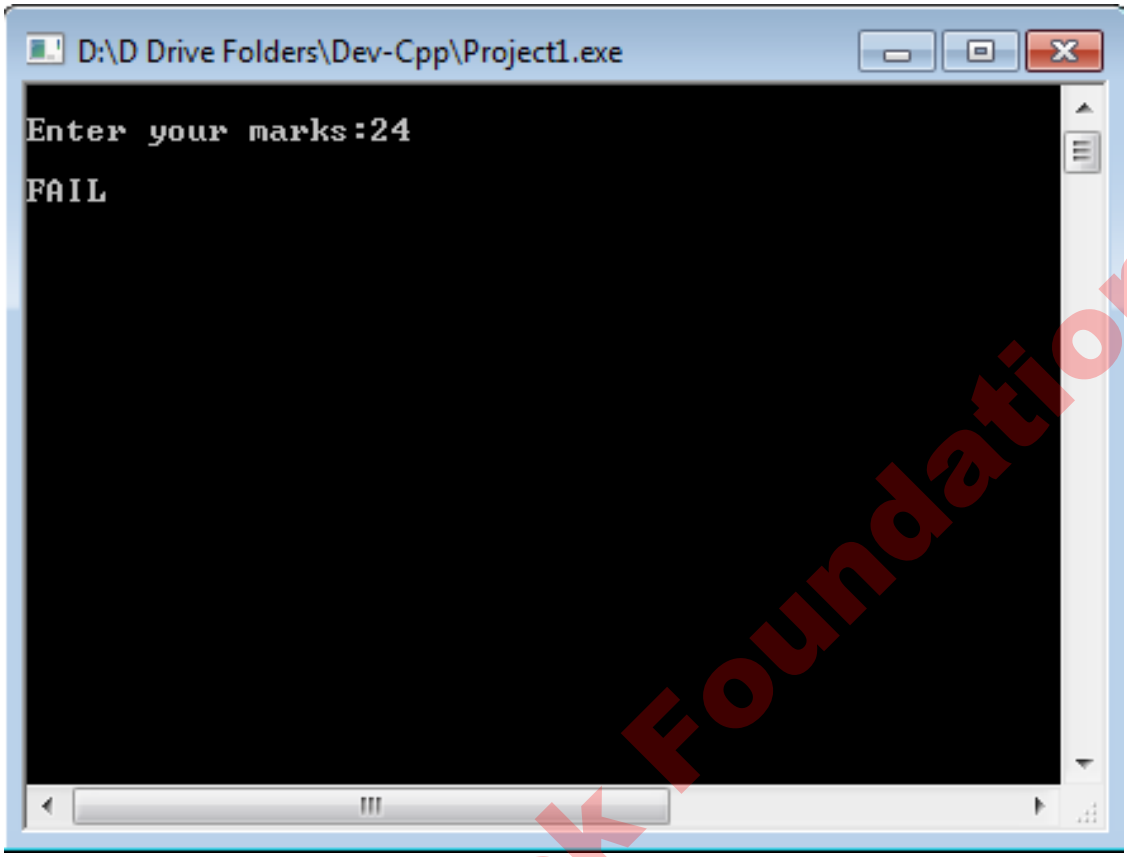


Fig.4-17 Execution of Program 10

4.1.10 NESTED SELECTION STRUCTURE

The selection structure that is within another selection structure is known as nested selection structure. Sometimes, in computer programming, it is required to use a selection structure within another selection structure. This is also supported in C language. In C language, the programmer can have a selection structure (**if**, **if-else**, **else-if** or **switch** statement) within another selection structure.

Program 11: The program in Fig.4-18 demonstrates the use of nested **if-else** statement within another **if-else** statement.

- When this program is executed, it reads an integer number, stores it in the variable **n** and then the condition (n>0) is evaluated.
- If it is true, it prints the message that the entered number is a positive number and then the nested **if-else** statement is executed.
- The nested **if-else** statement prints whether **n** is an even number or an odd number.
- If the condition (n>o) is false then the statements following the first **if** are skipped and the last statement after the **else** is executed which prints the message that the user entered a negative number or zero.

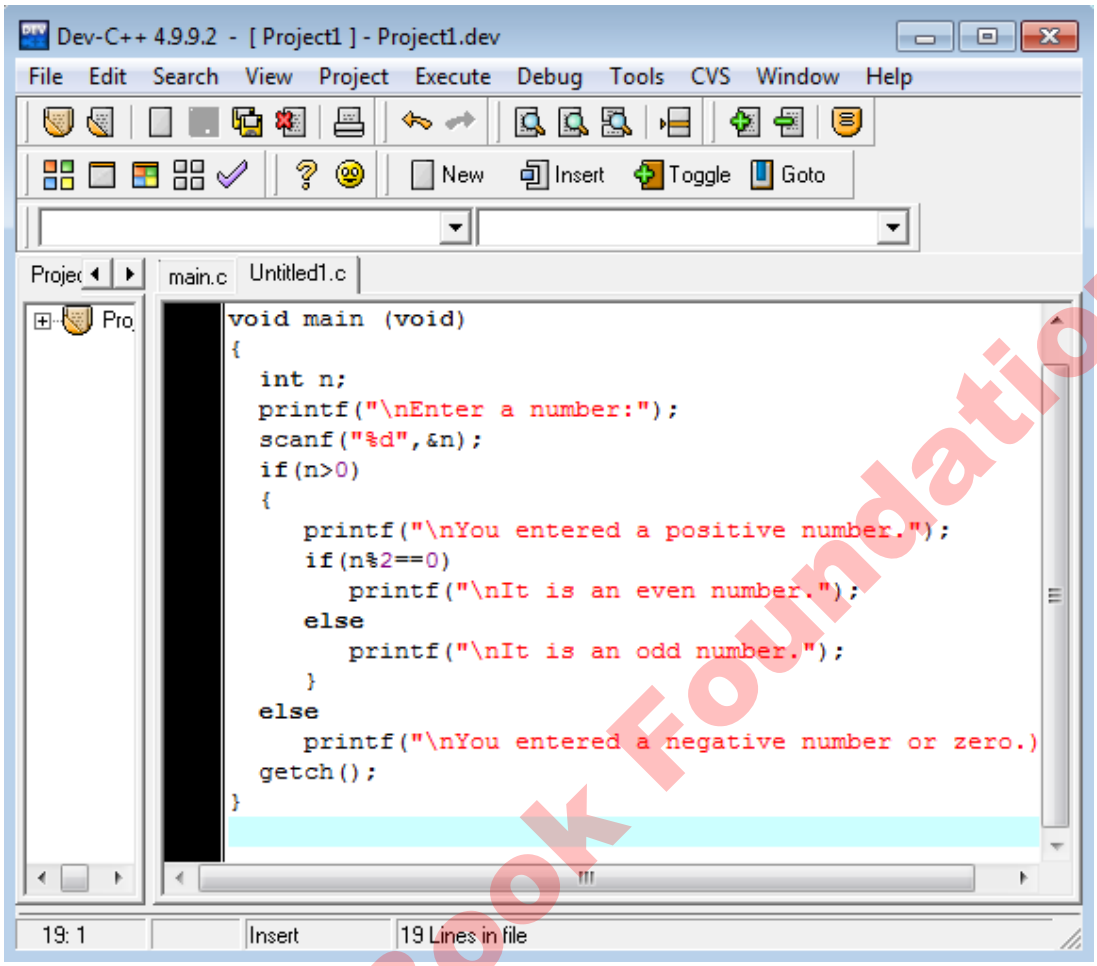


Fig.4-18 Using nested if-else statement in a program

Execution of the program is shown in Fig.4-19.

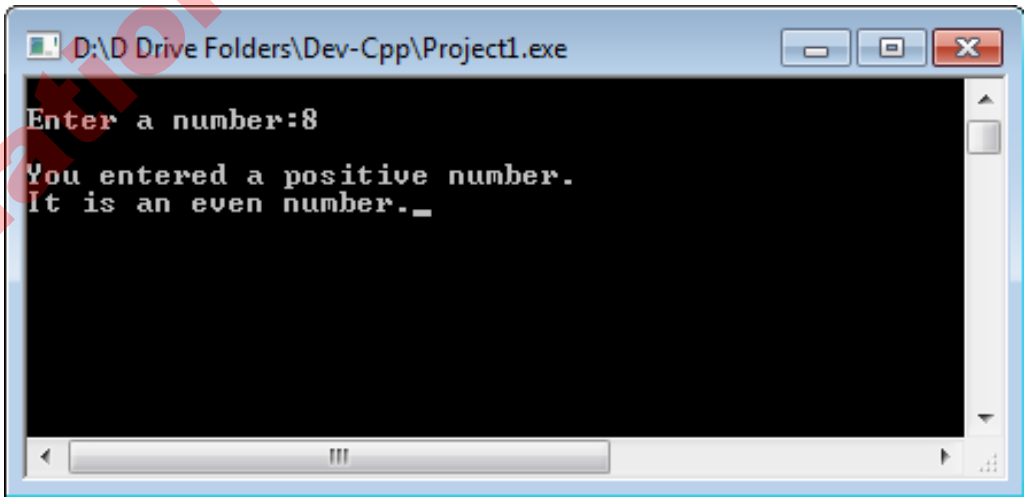


Fig.4-19 Execution of Program 11

Key Points

- In a programming language, a control statement is an instruction which determines the sequence of execution of other statements in a program.
- A conditional statement is an instruction in a programming language that contains a condition based on which flow of execution of program statements is controlled.
- The **if** statement is a control statement. When it is executed, the condition is evaluated. If it is true then the block of statement within the braces will be executed otherwise control will be transferred to the next statement.
- The **if-else** statement is a control statement. When it is executed, the condition is evaluated. If it is true then the block of statements following **if** will be executed otherwise the block of statements following **else** will be executed.
- The **if-else-if** statement is a control statement. When it is executed, the first condition is evaluated. If it is true then the block of statements following **if** will be executed. It the condition is false then the second condition after **else if** will be evaluated. If it is true, then the block of statements following **else if** will be executed. If any condition is true the block of statements following that **else if** will be executed. If none of the conditions is true then the block of statements following **else** will be executed automatically if it exists otherwise the control will be transferred to the next statement.
- The **switch** statement is used when multiple choices are given and one choice is to be selected. It is similar to **else-if** statement.
- A selection structure within another selection structure is known as nested selection structure.

Exercise

Q1. Select the best answer for the following MCQs.

i. For which purpose if structure is used in programming?

A. Repetition
B. Selection
C. Sequence
D. Input of data

ii. Which statement is suitable to use in a situation where there are only two choices based on a condition?

A. if statement
B. if-else-if statement
C. if-else statement
D. switch statement

iii. Which statement can be used in place of switch statement?

A. if statement
B. if-else statement
C. if-else-if statement
D. conditional operator

iv. Which statement can be used in place of conditional operator?

A. if statement
B. if-else statement
C. else-if statement
D. switch statement

v. Which statement is used to exit from the body of switch statement?

A. default
B. continue
C. exit
D. break

vi. Which of the following is a multiple selection statement?

A. if statement
B. if-else statement
C. if-else-if statement
D. none of these

vii. Which of the selection structures tests only for equality?

A. if statement
B. if-else statement
C. else-if statement
D. switch statement

viii. What will be printed when the following code is executed?

```
x=1;
switch(x)
{       case 1:
        case 2:
        case 3:
                printf("\n x is a positive number");
                break;
        default
                printf("\n value of x is 1");
}
```

A. value of x is 1
B. x is a positive number
C. Nothing will be printed
D. It will give error

Short Questions

Q2. Give short answers to the following questions.

i. Differentiate between if and if-else selection structures.
ii. Differentiate between else-if and switch selection structures.
iii. What is nested selection structure?
iv. Write the following statement using if-else statement.

```
k = (a+b>20)? a+3*b: a-b;
```

v. Write the following statement using conditional operator.

```
if (x>y)
        z=(x+y)/3;
else
        z=x-5*y;
```

vi. What will be the output of the following code?

```
int n, count, sum;
n=28; count=15; sum=30;
if (n<25)
{       count=count+5;
        printf("\nCount=%d",count);     }
else
{       count=count-5;
        sum=sum+n;
        printf("\nCount=%d",count);
        printf("\nSum=%d",sum);     }
```

vii. What will be the output of the following code?

```
char ch;
ch='c';
switch(ch)
{       case 'a':
                printf("\n   Good Morning!   "); break;
        case 'b':
                printf"\n   Have a Nice Day!   "; break;
        case 'c':
        case 'd':
        case 'e':
                printf("\n   Good Bye!   "); break;
}
```

Extensive Questions

Q3. What is control structure? Explain conditional control structure with examples.

Q4. What is the purpose of switch() statement? Explain with the help of one example.

Lab Activities

1. Write a program that reads a number and prints its square if it is greater than 10.
2. Write a program that reads two numbers and prints the larger.
3. Write the above program using conditional operator.
4. Write a program that reads a number (***n***) and prints a message based on its value as given below.

Value of *n*	**Message to print**
n is greater than zero	It is a positive number
n is less than zero	It is a negative number
n is equal to zero	It is equal to zero

5. Write a program that reads temperature (**t**) in Celsius and prints a message as given below.

Temperature	**Message to print**
t>35	It is hot!
t≥20, **t**≤35	Nice day!
t< 20	It is cold!

6. Write a program that reads marks (m) of a subject and prints the letter grade as given below.

Marks obtained	**Grade**
m≥80	A+
m≥70, m<80	A
m≥60, m<70	B
m≥50, m<60	C
m≥40, m<50	D
m≥33, m<40	E
m<33	Ungraded

7. Write a program that reads basic pay of an employee and calculates and prints his net pay as given below.

Net Pay = Basic Pay + House Rent

Basic Pay	**House Rent**
< 25,000	30% of Basic Pay
≥25,000 and ≤40,000	40% of Basic Pay
> 40,000	50% of Basic Pay

About Authors

Mohammad Sajjad Heder

Mohammad Sajjad Heder has done Master (MS) in Computer Science from George Mason University, Virginia, USA and B.E. Mechanical Engineering from Kim Check Engineering College, Korea. He has profound knowledge of computers and is highly experienced in teaching the subject of Computer Science. He has been teaching Computer Science to students of SSC and HSSC for the last twenty years. He is the author of previous HSSC-I & II textbooks of Computer Science. These books were approved by Ministry of Education, Islamabad and prescribed by the Federal Board of Intermediate and Secondary Education for about twenty years.

Mohammad Khalid

Mohammad Khalid is Head of Computer Science Department at OPF Boys College, Islamabad. He has very outstanding educational career. He did his MSc in Computer Science from the University of Peshawar and earned his B.Ed. degree in Science from Institute of Education and Research, University of Peshawar. He has been instructor and teacher in Computer Science for the last eighteen years. He has attended many national and international level seminars, workshops and meetings for the promotion of teaching Computer Science at different levels.

He has been member of the National Curriculum development team since 2007. He has contributed a great deal in writing the Curriculum of Computer Education from Grades VI-VIII and Curriculum of Computer Science from Grades IX-XII.

As an author he has written many textbooks regarding computer for different Textbook Boards of Pakistan from grade VI to XII. For queries about book in question, please contact following Emails.

mkhalidopf@yahoo.com / sajjadheder@yahoo.com

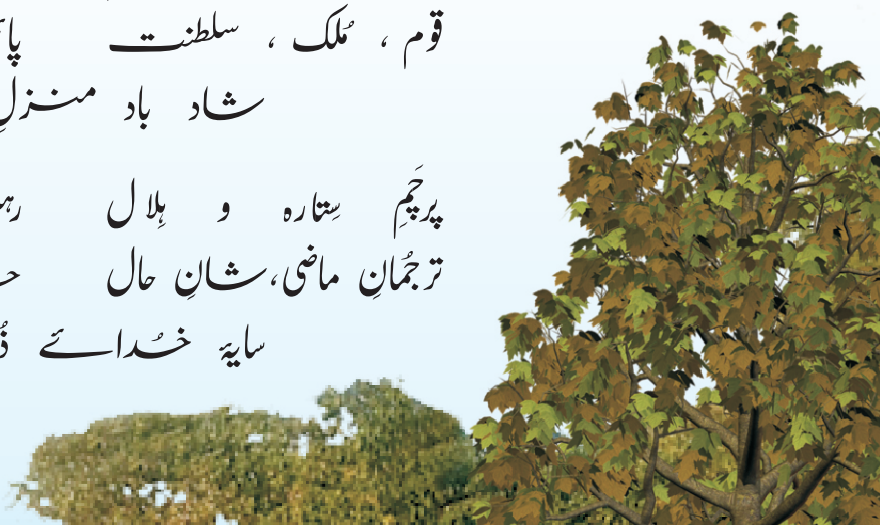
Approved by the Capital Administration & Development Division
(Curriculum Development & Textbook Production Wing),
Government of Pakistan, Islamabad,
vide letter No. 1-2/2014-AEA(BS) Dated: 11-12-2015

قومی ترانہ

پاک سَر زمین شاد باد! کشورِ حسین شاد باد!
تو نِشانِ عزمِ عالی شان ارضِ پاکستان
مرکزِ یقین شاد باد!

پاک سر زمین کا نِظام قُوّتِ اُخُوّتِ عوام
قوم ، مُلک ، سلطنت پائندہ تابندہ باد!
شاد باد منزلِ مُراد!

پرچَمِ سِتارہ و ہِلال رہبرِ ترقی و کمال
ترجُمانِ ماضی، شانِ حال جانِ اِستقبال
سایۂ خُدائے ذُوالجلال!



National Book Foundation
as
Federal Textbook Board
Islamabad