COMPUTER SCIENCE

2021-22

NOT FOR SALE

EDUCATION REFORMS PROGRAMME OF GOVERNMENT OF THE PUNJAB

Scanned with CamScanner

COMPUTER SCIENCE

PUNJAB CURRICULUM AND TEXTBOOK BOARD, LAHORE

Scanned with CamScanner

All rights reserved with the **Punjab Curriculum and Textbook Board, Lahore.**

No part of this book can be copied, translated, reproduced or used for preparation of test papers, guide books, key notes, helping books, etc.

Contents

Unit	Topic	Page
1	Introduction to Programming	1
2	User Interface	21
3	Conditional Logic	52
4	Data and Repetition	77
5	Functions	101
	Glossary	118
	Index	121
	Answers	122

AUTHOR

Dr. Muhammad Adnan Hashmi
Assistant Professor
Department of Computer Science and IT
The University of Lahore

EDITOR

Dr. Mudasser Naseer
Associate Professor
Department of Computer Science and IT
The University of Lahore

Artist

Ms. Aisha Waheed

Designer

Mr. Uzair Ahmad
BSCS (continued), COMSATS University Islamabad (Lahore Campus)

Layout Setting

Mr. Aleem Ur Rehman

Supervision

Director Manuscripts
Dr. Abdullah Faisal

Mr. Jahanzaib Khan
S.S Computer Science
PCTB, Lahore

Punjab Curriculum & Textbook Board **Kutab Khana Khurshidia, Lahore**

Date	PMIU	FEF	PWWB	Total Quantity
Jan.2021	177,795	1,350	973	180,118

Scanned with CamScanner

Table of Contents

1: Introduction to Programming 1

- 1.1 Programming Environment 2
  - 1.1.1 Integrated Development Environment (IDE)....3
  - 1.1.2 Text Editor....4
  - 1.1.3 Compiler....5
- 1.2 Programming Basics 5
  - 1.2.1 Reserved Words....6
  - 1.2.2 Structure of a C Program....6
  - 1.2.3 Purpose and Syntax of comments in C Programs....8
- 1.3 Constants and Variables 9
  - 1.3.1 Constants....10
  - 1.3.2 Variables....11
  - 1.3.3 Data type of a Variable....11
  - 1.3.4 Name of a variable....12
  - 1.3.5 Variable Declaration....13
  - 1.3.6 Variable Initialization....14

2: User Interface 21

- 2.1 Input/ Output (I/O) Functions 23
  - 2.1.1 printf()....23
  - 2.1.2 Format Specifiers....24
  - 2.1.3 scanf()....26
  - 2.1.4 getch()....28
  - 2.1.5 Statement Terminator....29
  - 2.1.6 Escape Sequence....29
- 2.2 Operators 31
  - 2.2.1 Assignment Operators....31
  - 2.2.2 Arithmetic Operators....32
  - 2.2.3 Relational Operators....37
  - 2.2.4 Assignment Operator (=) and equal to Operator (==)....38
  - 2.2.5 Logical Operators....39
  - 2.2.6 Unary vs Binary Operators....41
  - 2.2.7 Operators' Precedence....42

3: Conditional Logic 52

- 3.1 Control Statements 52
- 3.2 Selection Statements 52

Scanned with CamScanner

Table of Contents

- 3.2.1 If Statement....53
- 3.2.2 If-else Statement....59
- 3.2.3 Nested If-else Structures....64
- 3.2.4 Solved Example Problems....67

4: Data and Repetition **77**

- 4.1 Data Structures 78
  - 4.1.1 Array....78
  - 4.1.2 Array Declaration....79
  - 4.1.3 Array Initialization....79
  - 4.1.4 Accessing array elements....80
  - 4.1.5 Using variables as array indexes....82
- 4.2 Loop Structures 83
  - 4.2.1 General structure of loops....83
  - 4.2.2 General syntax of for loop....83
  - 4.2.3 Nested Loops....87
  - 4.2.4 Solved Example Problems....91
  - 4.2.5 Loops and Arrays....93
  - 4.2.6 Solved Example Problems....95

5: Functions **101**

- 5.1 Functions 102
  - 5.1.1 Types of Functions....102
  - 5.1.2 Advantages of Functions....103
  - 5.1.3 Structure of a Function....103
  - 5.1.4 Defining a Function....104

6: Glossary **118**

7: Indexes **121**

8: Answers **122**

Scanned with CamScanner

Unit # 1

INTRODUCTION TO PROGRAMMING

Students Learning Outcomes

After completing this unit students will be able to

- Describe the concept of Integrated Development Environments (IDE)
- Explain the following modules of the C programming environment
  - Text Editor
  - Compiler
- Identify the reserved words
- Describe the structure of a C program covering
  - Include
  - main () function
  - Body of main { }
- Explain the purpose of comments and their syntax
- Explain the difference between a constant and a variable
- Explain the rules for specifying variable names
- Know the following data types offered by C and the number of bytes taken by each data type
  - Integer – int (signed/unsigned)
  - Floating point – float
  - Character – char
- Explain the process of declaring and initializing variables

Scanned with CamScanner

Unit Introduction

Computers have become an important part of our daily lives. They can help us to solve several problems ranging from complex mathematical problems and searching on the internet to controlling and operating satellites and rocket launchers. In reality, computers are not very smart on their own. In order to perform all the tasks, they have to be fed a series of instructions by humans which tell them how to behave and perform when faced with a particular type of problem. These series of instructions are known as a ***computer program*** or ***software***, and the process of feeding or storing these instructions in the computer is known as ***computer programming***. The person who knows how to write a computer program correctly is known as a ***programmer***.

Computers cannot understand English, Urdu or any other common language that humans use for interacting with each other. They have their own special languages, designed by computer scientists. Programmers write computer programs in these special languages called ***programming languages***. Java, C, C++, C#, Python are some of the most commonly used programming languages. In this book, we are using C language to write computer programs. This chapter discusses some basics of computer programming using C language.

DID YOU KNOW?

C language was developed by Dennis Ritchie between 1969 and 1973 at Bell Labs.

1.1 Programming Environment

In order to correctly perform any task, we need to have proper tools. For example for gardening we need gardening tools and for painting we need a collection of paints, brushes and canvas. Similarly we need proper tools for programming.

A collection of all the necessary tools for programming makes up a programming environment. It is essential to setup a programming environment before we start writing programs. It works as a basic platform for us to write and execute programs.

Scanned with CamScanner

1.1.1 Integrated Development Environment (IDE)

A software that provides a programming environment to facilitate programmers in writing and executing computer programs is known as an **Integrated Development Environment (IDE).**

An IDE has a graphical user interface (GUI), meaning that a user can interact with it using windows and buttons to provide input and get output. An IDE consists of tools that help a programmer throughout the phases of writing, executing and testing a computer program. This is achieved by combining text editors, compilers and debuggers in a single interface. Some of the many available IDEs for C programming language are:

1) Visual Studio
2) Xcode
3) Code::Blocks
4) Dev C++

Figure 1.1 shows the main screen of Code::Blocks IDE.

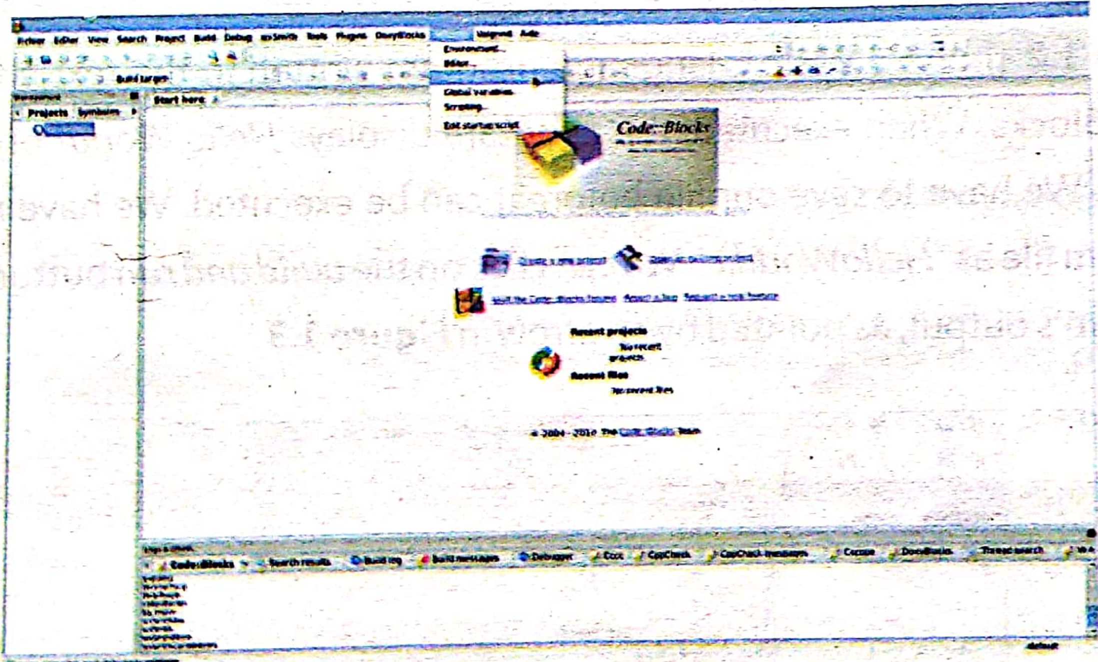


Figure 1.1: Main interface of Code::Blocks

Use your web browser to find out the names of three different IDEs that can be used for C programming language.

Scanned with CamScanner

1.1.2 Text Editor

An **text editor** is a software that allows programmers to write and edit computer programs. All IDEs have their own specific text editors. It is the main screen of an IDE where we can write our programs.

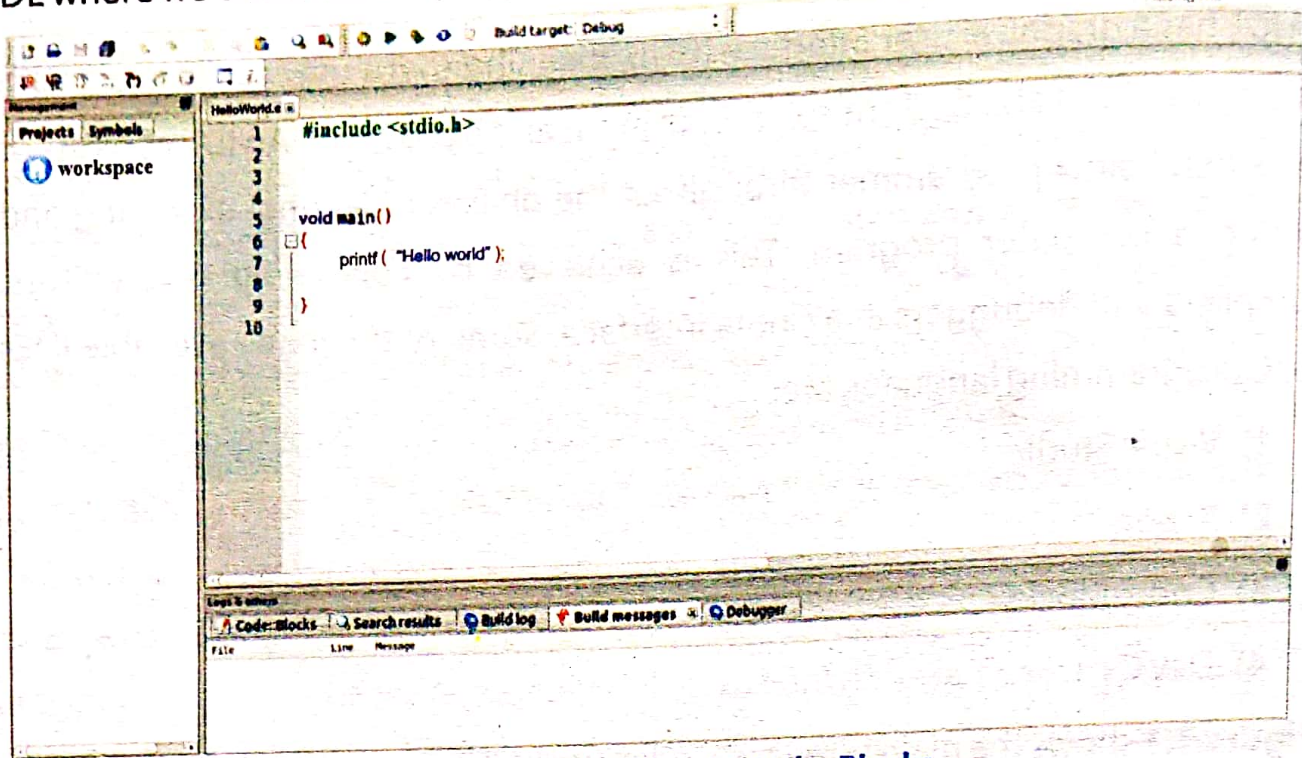


Figure 1.2: Text editor in Code::Blocks

Figure 1.2 shows a basic C language program written in the text editor of IDE Code::Blocks. When executed, this program displays *Hello World!* on computer screen. We have to save our file before it can be executed. We have named our program file as "*HelloWorld.c*". We can click on the *build and run* button to see the program's output, as pointed by an arrow in **Figure 1.3**.

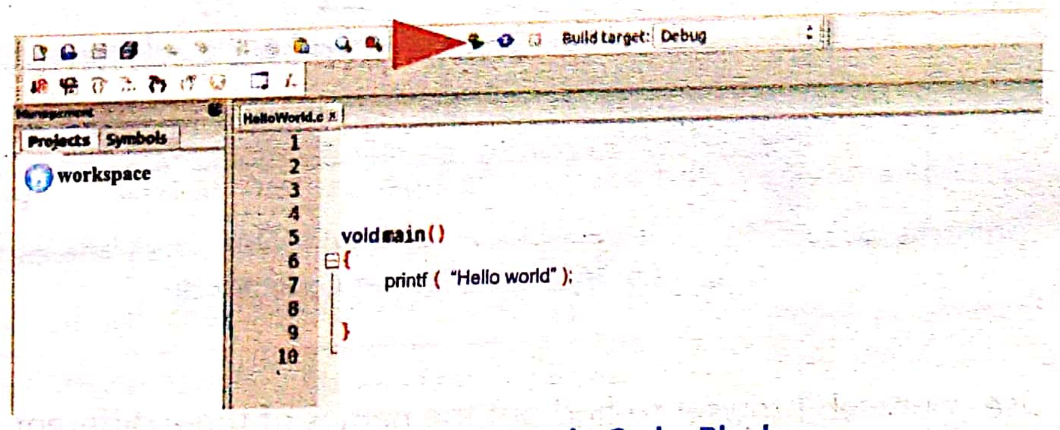


Figure 1.3: Running program in Code::Blocks

Scanned with CamScanner

A console screen showing the output is displayed, as shown in **Figure 1.4**.

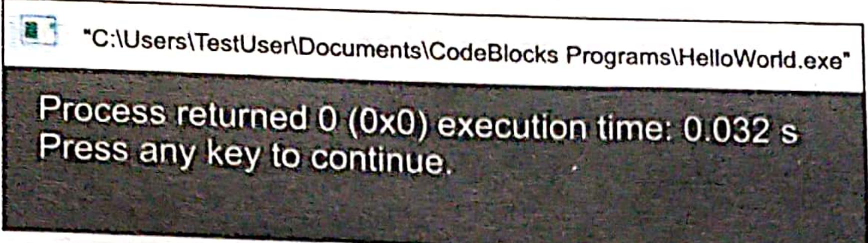


Figure 1.4: Program Output

Open the IDE installed on your lab computer. Write the program written in Figure 1.2 in the text editor of your IDE and execute it.

1.1.3 Compiler

Computers only understand and work in machine language consisting of 0s and 1s. They require the conversion of a program written in *programming language* to *machine language*, in order to execute it. This is achieved using a compiler. A **compiler** is a software that is responsible for conversion of a computer program written in some high level programming language to machine language code **(Figure 1.5)**.

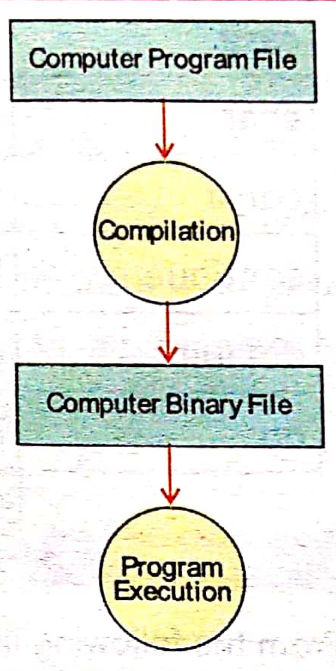


Figure 1.5: Program Execution

1.2 Programming Basics

Each *programming language* has some primitive building blocks and provides some rules in order to write an accurate program. This set of rules is known as **syntax** of the language. Syntax can be thought of as grammar of a programming language. While programming, if proper syntax or rules of the programming language are not followed, the program does not get compiled. In this case, the compiler generates an error. This kind of errors are called *syntax errors*.

Not For Sale- PESRP

Scanned with CamScanner

1.2.1 Reserved Words

Every programming language has a list of words that are predefined. Each word has its specific meaning already known to the compiler. These words are known as **reserved words** or **keywords.** If a programmer gives them a definition of his own, it causes a syntax error. Table 1.1 shows the list of reserved words in C programming language.

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

Table 1.1: Reserved words in C language

From the following list, encircle the reserved words in C language:
int, pack, create, case, return, small, math, struct, program, library.

1.2.2 Structure of a C Program

We can understand the structure of a C language program, by observing the program written in **Figure 1.2**. We can see that a program can be divided into three main parts:

1. **Link section or header section:** While writing programs in C language, we make extensive use of functions that are already defined in the language. But before using the existing functions, we need to include the files where these functions have been defined. These files are called header files. We include these header files in our program by writing the *include* statements at the top of program.

Scanned with CamScanner

General structure of an *include* statement is as follows:

```
#include<header_file_name>
```

Here *header_file_name* can be the name of any header file.

In the above example **(Figure 1.2)**, we have included file *stdio.h* that contains information related to input and output functions. Many other header files are also available, for example file *math.h* contains all predefined mathematics functions.

2. **Main section:** It consists of a *main()* function. Every C program must contain a *main()* function and it is the starting point of execution.
3. **Body of *main()* function:** The body of *main()* is enclosed in the curly braces { }. All the statements inside these curly braces make the body of *main* function. In the above program **(Figure 1.2)**, the statement `printf("Hello world!");` uses a predefined function *printf* to display the statement *Hello World!* on computer screen. We can also create other functions in our program and use them inside the body of *main()* function.

Important Note:

Following points must be kept in mind in order to write syntactically correct C language programs.

- The sequence of statements in a C language program should be according to the sequence in which we want our program to be executed.
- C language is case sensitive. It means that if a keyword is defined with all small case letters, we cannot capitalize any letter i.e. *int* is different from *Int*. Former is a keyword, whereas latter is not.
- Each statement ends with a semi-colon ; symbol.

Not For Sale- PESRP

Scanned with CamScanner

ACTIVITY 1.4

Identify different parts of the following C program:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        printf("I am a student of class 10");
        getch();
}
```

1.2.3 Purpose and Syntax of Comments in C Programs

Comments are the statements in a program that are ignored by the compiler and do not get executed. Usually comments are written in natural language e.g. in English language, in order to provide description of our code.

Purpose of writing comments

Comments can be thought of as documentation of the program. Their purpose is twofold.

1) They facilitate other programmers to understand our code.

2) They help us to understand our own code even after years of writing it.

We do not want these statements to be executed, because it may cause syntax error as the statements are written in natural language.

Syntax of writing comments

In C programming language, there are two types of comments.

1- Single-line Comments

2- Multi-line Comments

Single-line comments start with //. Anything after // on the same line, is considered a comment. For example, `//This is a comment.`

Multi-line comments start with /* and end at */. Anything between /* and */ is considered a comment, even on multiple lines. For example,

Scanned with CamScanner

```
/*this is
a multi-line
comment*/
```

Following example code demonstrates the usage of comments:

</> EXAMPLE CODE 1.1

```
#include <stdio.h>
/*this program displays "I am a student of class 10" on the
output screen*/
void main()
{ //body of main function starts from here
      printf("I am a student of class 10");
} //body of main function ends here
```

ACTIVITY 1.5

Tick valid comments among the following:

- *comment goes here*
- /comment goes here/
- %comment goes here%
- /* comment goes
  here*/
- /*comment goes here/
- //comment goes here */

1.3 Constants and Variables

Each language has a basic set of alphabets (character set) that are combined in an allowable manner to form words, and then these words can be used to form sentences. Similarly in C programming language we have a *character set* that includes:

1) Alphabets (A, B, ……, Y, Z), (a, b….y, z)

2) Digits (0 - 9)

3) Special symbols (~ ` ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /)

Scanned with CamScanner

These alphabets, digits and special symbols when combined in an allowable manner, form *constants*, *variables* and *keywords* (also known as reserved words). We have already discussed the concept of reserved words. In the following, we discuss the concept of *constants* and *variables*.

1.3.1 Constants

Constants are the values that cannot be changed by a program e.g. 5, 75.7, 1500 etc. In C language, primarily we have three types of constants:

1- **Integer Constants:** These are the values without a decimal point e.g. 7, 1256, 30100, 55555, -54, -2349 etc. They can be positive or negative. If the value is not preceded by a sign, it is considered as positive.

2- **Real Constants:** These are the values including a decimal point e.g. 3.14, 15.3333, 75.0, -1575.76, -7941.2345 etc. They can also be positive or negative.

3- **Character Constants:** Any single small case letter, upper case letter, digit, punctuation mark, special symbol enclosed within ' ' is considered a character constant e.g. '5', '7', 'a', 'X', '!', ';' etc.

IMPORTANT TIP

A digit used as a character constant i.e. '9', is different from a digit used as an integer constant i.e. 9. We can add two integer constants to get the obvious mathematical result e.g. 9 + 8 = 17, but we cannot add a character constant to another character constant to get the obvious mathematical result e.g. '9' + '8' ≠ 17.

ACTIVITY 1.6

Identify the type of constant for each of the following values:

12	1.2	'*'	-21	32.768
'a'	-12.3	41	40.0	'\'

Scanned with CamScanner

1.3.2 Variables

A variable is actually a name given to a memory location, as the data is physically stored inside the computer's memory. The value of a variable can be changed in a program. It means that, in a program, if a variable contains value 5, then later we can give it another value that replaces the value 5.

Each variable has a ***unique name*** called ***identifier*** and has a ***data type***. Data type describes the type of data that can be stored in the variable. C language has different data types such as *int*, *float*, and *char*. The types *int*, *float* and *char* are used to store integer, real and character data respectively. **Table 1.2** shows the matching data types in C language, against different types of data.

Type of Data	Matching Data Type in C language	Sample Values
Integer	*int*	123
Real	*float*	23.5
Character	*char*	'a'

Table 1.2: Matching data types against different types of data

In the following, we discuss in detail the possible data types and names of variables.

1.3.3 Data Type of a Variable

Each variable in C language has a data type. The data type not only describes the type of data to be stored inside the variable but also the number of bytes that the compiler needs to reserve for data storage. In the following, we discuss different data types provided by C language.

DID YOU KNOW?

Some compilers use two bytes of memory to store an *int* value. In such compilers, an int value ranges from -32,768 to 32,768.

Scanned with CamScanner

- **Integer – int (signed/unsigned)**

  Integer data type is used to store integer values (whole numbers). Integer takes up 4 bytes of memory. To declare a variable of type integer, we use the keyword *int*.

  Signed int: A *signed int* can store both positive and negative values ranging from -2,147,483,648 to 2,147,483,647. By default, type *int* is considered as a signed integer.

  Unsigned int: An *unsigned int* can store only positive values and its value ranges from 0 to +4,294,967,295. Keyword *unsigned int* is used to declare an unsigned integer.

- **Floating Point – float**

  Float data type is used to store a real number (number with floating point) up to six digits of precision. To declare a variable of type float, we use the keyword *float*. A *float* uses 4 bytes of memory. Its value ranges from $3.4 X 10^{-38}$ to $3.4 X 10^{38}$.

- **Character – char**

  To declare character type variables in C, we use the keyword *char*. It takes up just 1 byte of memory for storage. A variable of type *char* can store one character only.

1.3.4 Name of a Variable

Each variable must have a unique name or identifier. Following rules are used to name a variable.

1. A variable name can only contain alphabets (uppercase or lowercase), digits and underscore _ sign.
2. Variable name must begin with a letter or an underscore, it cannot begin with a digit.
3. A reserved word cannot be used as a variable name.

Scanned with CamScanner

4. There is no strict rule on how long a variable name should be, but we should choose a concise length for variable name to follow good design practice.

Some examples of valid variable names are height, AverageWeight, _var1.

Encircle the valid variable names among the following:

_Hello,	1var	roll_num	Air23Blue	float
Case	$car	name	=color	Float

Important Note:

Good programming practice suggests that we should give appropriate name to a variable, that describes its purpose e.g. in order to store salary of a person, appropriate variable name could be *salary* or *wages*.

1.3.5 Variable Declaration

We need to declare a variable before we can use it in the program. Declaring a variable includes specifying its data type and giving it a valid name. Following syntax can be followed to declare a variable.

data_type variable_name;

Some examples of valid variable declarations are as follows:

```
unsigned int age;
float height;
int salary;
char marital_status;
```

Multiple variables of same data type may also be declared in a single statement, as shown in the following examples:

```
unsigned int age, basic_salary, gross_salary;
int points_scored, steps;
```


Not For Sale - PESRP

Scanned with CamScanner

```
float height, marks;
char marital_status, gender;
```

A variable cannot be declared unless we mention its data type. After declaring a variable, its data type cannot be changed. Declaring a variable specifies the type of variable, the range of values allowed by that variable, and the kind of operations that can be performed on it. Following example shows a program declaring two variables:

</> EXAMPLE CODE 1.2

```
void main()
{
        char grade;
        int value;
}
```

1.3.6 Variable Initialization

Assigning value to a variable for the first time is called variable initialization. C language allows us to initialize a variable both at the time of declaration, and after declaring it. For initializing a variable at the time of declaration, we use the following general structure.

```
data_type  variable_name = value;
```

Following example shows a program that demonstrates the declaration and initialization of two variables.

</> EXAMPLE CODE 1.3

```
#include<stdio.h>
void main()
{
        char grade; //Variable grade is declared
        int  value  =  25;  /*Variable  value  is  declared  and
        initialized.*/
        grade = 'A'; //Variable grade is initialized
}
```


Scanned with CamScanner

Write a program that declares variables of appropriate data types to store your personal data. Initialize these variables with the following data:

- initial letter of your name
- initial letter of your gender
- your age
- your marks in 8th class
- your height

Scanned with CamScanner

SUMMARY

- Computers need to be fed a series of instructions by humans which tell them how to perform a particular task. These series of instructions are known as a ***computer program*** or ***software***.
- The process of feeding or storing the instructions in the computer is known as ***computer programming*** and the person who knows how to write a computer program correctly is known as a ***programmer***.
- Computer programs are written in languages called ***programming languages***. Some commonly known programming languages are Java, C, C++, Python.
- A collection of all the necessary tools for programming makes up a ***programming environment***. Programming environment provides us the basic platform to write and execute programs.
- A software that provides a programming environment which facilitates the programmer in writing and executing computer programs is known as an ***Integrated Development Environment (IDE).***
- A ***text editor*** is a software that allows programmers to write and edit computer programs. All IDEs have their own specific editors.
- A **compiler** is a software that is responsible for conversion of a computer program written in some programming language to machine language code.
- Every programming language has some primitive building blocks and follows some grammar rules known as its **syntax**.
- Every programming language has a list of words that are predefined. Each word has its specific meaning already known to the compiler. These words are known as **reserved words** or **keywords.**
- A program is divided into three parts. ***Header section*** is the part where header files are included. ***Main section*** corresponds to the main function and the ***body of the main function*** includes everything enclosed in the curly braces.

Scanned with CamScanner

- ***Comments*** are the statements that are ignored by the compiler and do not get executed. To include additional information about the program, comments can be used.
- **Constants** are the values that do not change. The three types of constants are *integer constants*, *real constants* and *character constants.*
- ***Variables*** is a name given to a memory location as the data is physically stored inside the computer's memory. Each variable has a ***unique name*** or ***identifier*** by which we can refer to that variable, and an associated ***data type*** that describes the type of constant that can be stored in that variable.
- A variable must be declared before its use. ***Variable declaration*** includes specifying variable's data type and giving it a valid name.
- Assigning value to a variable for the first time is called ***variable initialization***. The variable can be initialized at the time of declaration or after declaration.

Scanned with CamScanner

Exercise

Q1 Multiple Choice Questions

1) A software that facilitates programmers in writing computer programs is known as ______.
   a) a compiler b) an editor c) an IDE d) a debugger
2) ________ is a software that is responsible for the conversion of program files to machine understandable and executable code.
   a) Compiler b) Editor c) IDE d) Debugger
3) Every programming language has some primitive building blocks and follows some grammar rules known as its ______.
   a) programming rules b) syntax c) building blocks d) semantic rules
4) A list of words that are predefined and must not be used by the programmer to name his own variables are known as ______.
   a) auto words b) reserved words
   c) restricted words d) predefined words
5) *include* statements are written in ______ section.
   a) header b) main c) comments d) print
6) ________ are added in the source code to further explain the techniques and algorithms used by the programmer.
   a) Messages b) Hints c) Comments d) Explanations
7) ________ are the values that do not change during the whole execution of program.
   a) Variables b) Constants c) Strings d) Comments
8) A *float* uses ____ bytes of memory.
   a) 3 b) 4 c) 5 d) 6
9) For initializing a variable, we use ______ operator.
   a) → b) = c) @ d) ?
10) ______ can be thought of as a container to store constants.
   a) box b) jar c) variable d) collection

Scanned with CamScanner

Q2 True or False

1) An IDE combines text editors, libraries, compilers and debuggers in a single interface. T/F
2) Computers require the conversion of the code written in program file to machine language in order to execute it. T/F
3) *Column* is a reserved word in C programming language. T/F
4) **comment goes here** is a valid comment. T/F
5) *float* can store a real number upto six digits of precision. T/F

Q3 Define the following.

1) IDE **2)** Compiler **3)** Reserved Words **4)** Main section of a program **5)** *char* data type

Q4 Briefly answer the following questions.

1) Why do we need a programming environment?
2) Write the steps to create a C program file in the IDE of your lab computer.
3) Describe the purpose of a compiler.
4) List down five reserved words in C programming language.
5) Discuss the main parts of the structure of a C program.
6) Why do we use comments in programming?
7) Differentiate between constants and variables.
8) Write down the rules for naming variables.
9) Differentiate between *char* and *int*.
10) How can we declare and initialize a variable?

Q5 Match the columns.

A	B	C
1) IDE	a) Machine executable code	
2) Text Editor	b) include statement	
3) Compiler	c) Python	
4) Programming Language	d) CLion	
5) Reserved words	e) /* (a+b) */	
6) Link Section	f) Notepad	
7) Body of main()	g) struct	
8) Comment	h) { }	

Scanned with CamScanner

Programming Exercises

Exercise 1

- With the help of your teacher open the IDE installed on your lab computer for writing C programs.
- Write the following program in the editor and save it as "welcome.c".

```
#include <stdio.h>
#include <conio.h>
void main()
{
        /*A simple C language program*/
        printf("Welcome to C language");
        getch();
}
```

- Run the program to see *Welcome to C language* printed on the screen as output.

Exercise 2

Write a program that declares variables of appropriate data types to store the personal data about your best friend. Initialize these variables with the following data:

- initial letter of his name
- initial letter of his gender
- his age
- his height

Scanned with CamScanner

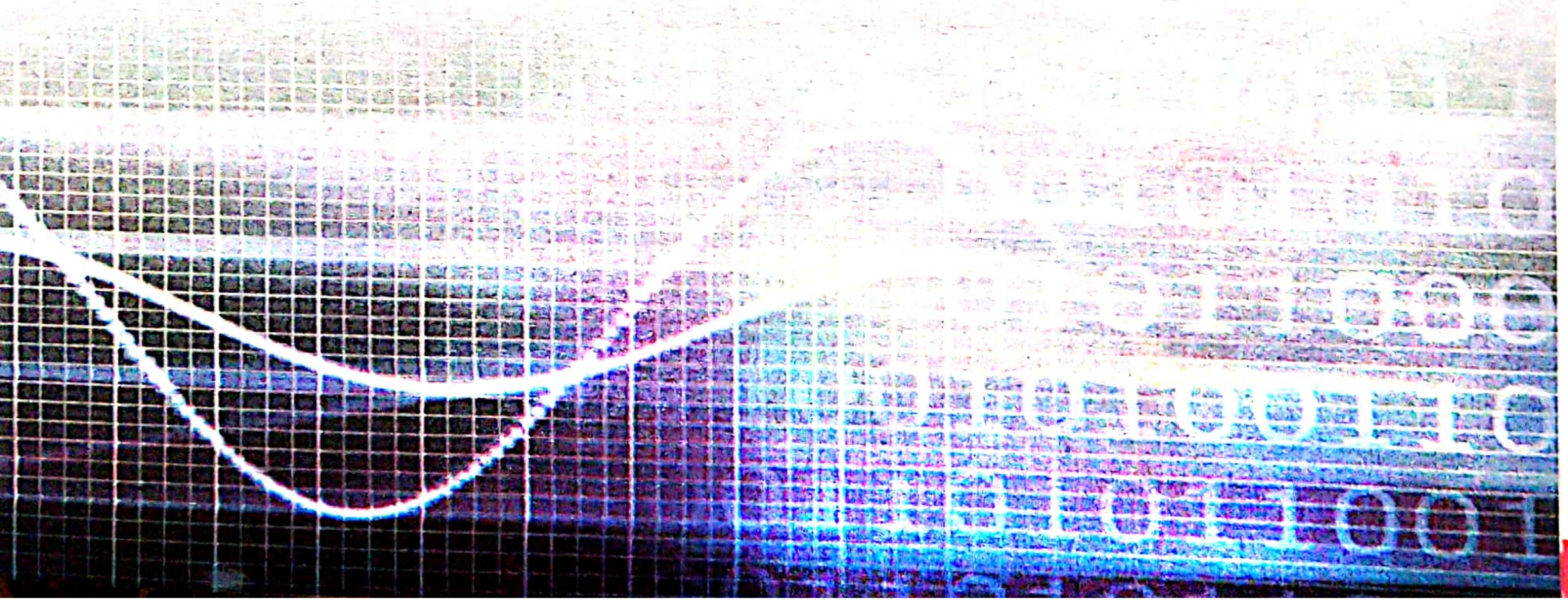
Unit # 2

USER INTERACTION

Students Learning Outcomes

After completing this unit students will be able to

- Use output functions like printf()
- Use input functions like
  - scanf()
  - getch()
- Use statement terminator (semicolon)
- Define format specifiers
  - Integer - %i
  - Decimal - %d
  - Float - %f
  - Char - %c
- Define an escape sequence
- Explain the use of the following escape sequences using programming examples:
  - Newline - \n
  - Tab - \t
- Define an arithmetic operator
- Use the following arithmetic operators:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Remainder (%)

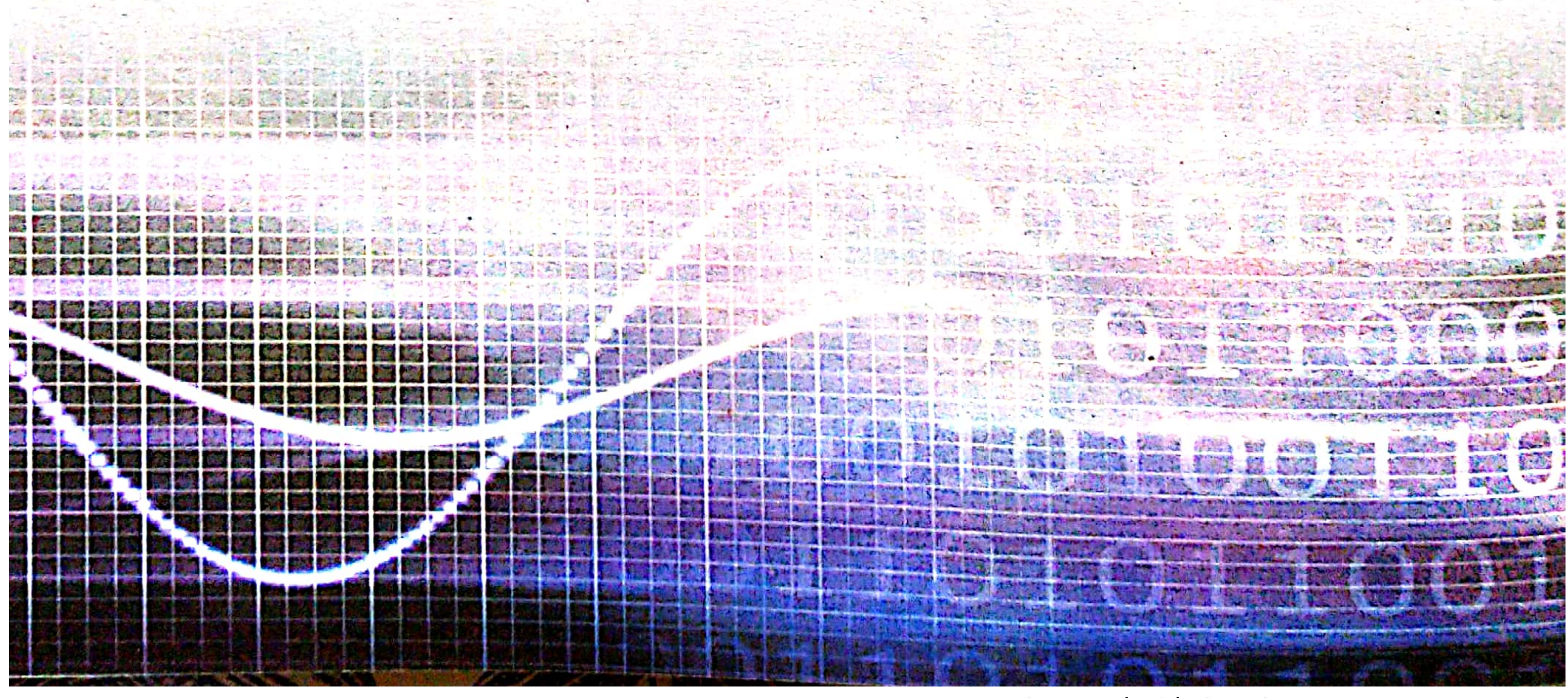


Scanned with CamScanner

Unit # 2

USER INTERACTION

- Use the assignment operator
- Define relational operators
- Use the following relational operators:
  - Less than (<)
  - Greater than (>)
  - Less than or equal to (<=)
  - Greater than or equal to (>=)
  - Equal to (==)
  - Not Equal to (!=)
- Define a logical operator
- Use the following logical operators:
  - AND (&&)
  - OR (||)
  - NOT (!)
- Differentiate between the assignment operator (=) and equal to operator (==)
- Differentiate between the unary and binary operators
- Define and explain the order of precedence of operators



Scanned with CamScanner

Unit Introduction:

A computer is a device that takes data as input, processes that data and generates the output. Thus all the programming languages must provide instructions to handle input, output and processing of data. In this chapter, we discuss different pre-built input/output functions available in C language. We also discuss different operators that we can apply to process the data.

2.1 Input/output (I/O) functions:

We need a way to provide input and show output while writing programs. Each programming language has its keywords or standard library functions for I/O operations. C language offers *printf* function to display the output, and *scanf* function to get input from user. In the following section, we discuss these two functions.

2.1.1 printf()

printf is a built-in function in C programming language to show output on screen. Its name comes from "print formatted" that is used to print the formatted output on screen. All data types discussed in the previous chapter can be displayed with *printf* function. To understand the working of *printf* function, let's look at the following example program:

</> EXAMPLE CODE 2.1

```
#include<stdio.h>
void main ()
{
        printf("Hello World");
}
Output:
Hello World
```

In this example, *printf* function is used to display *Hello World* on screen. Whatever we write inside the double quotes " and " in the *printf()* function, gets displayed on screen.

Scanned with CamScanner

ACTIVITY 2.1

Write down the output of following code:

```
# include<stdio.h>
void main ()
{
        printf("I am UPPERCASE and this is lowercase");
}
```

ACTIVITY 2.2

Write a program that shows your first name in Uppercase and your last name in lower case letters on screen.

2.1.2 Format Specifiers

What if we want to display the value of a variable? Let's declare a variable and then check the behavior of printf.

```
int age = 35;
```

Now I want to display the value of this variable *age* on screen. So, I write the following statement:

```
printf("age");
```

But, it does not serve the purpose, because it displays the following on screen.

```
age
```

It does not display the value stored inside the variable age, instead it just displays whatever was written inside the double quotes of *printf*. In fact, we need to specify the format of data that we want to display, using format specifiers. Table 2.1 shows format specifiers against different data types in C language.

Scanned with CamScanner

Data Type	Format Specifier
int	% d or % i
float	% f
char	% c

Table 2.1: Format specifiers for I/O operations

Suppose we want to show *int* type data, we must specify it inside the *printf* by using the format specifier *%d* or *%i*. In the same way, for *float* type data we must use *%f*. It is illustrated in the following example.

EXAMPLE CODE 2.2

```
#include<stdio.h>
void main()
{
	float height = 5.8;
	int age = 35;
	printf("My age is %d and my height is %f", age, height);
}
```

Output:

```
My age is 35 and my height is 5.800000
```

We can observe that while displaying output, first format specifier is replaced with the value of first variable/data after the ending quotation mark i.e. *age* in the above example, and second format specifier is replaced with the second variable/data.

IMPORTANT TIP

When we use *%f* to display a float value, it displays 6 digits after the decimal point. If we want to specify the number of digits after decimal point then we can write *%.nf* where *n* is the number of digits. In the above example, if we write the following statement:

```
printf("My age is %d and my height is %.2f", age, height);
```

The output is

```
My age is 35 and my height is 5.80
```

Important Note:

Format specifiers are not only used for variables. Actually they are used to display the result of any expression involving variables, constants, or both, as illustrated in the following example.

</> EXAMPLE CODE 2.3

```
# include <stdio.h>
void main ()
{
        printf("Sum of 23 and 45 is  %d", 23 + 45);
}
```

Output

```
Sum of 23 and 45 is 68
```

2.1.3 scanf()

scanf is a built-in function in C language that takes input from user into the variables. We specify the expected input data type in *scanf* function with the help of format specifier. If user enters integer data type, format specifier mentioned in *scanf* must be *%d* or *%i*. Consider the following example:

</> EXAMPLE CODE 2.4

```
#include <stdio.h>
void main ()
{
        char grade;
        scanf("%c", &grade);
}
```

In this example, *%c* format specifier is used to specify character type for the input variable. Input entered by user is saved in variable *grade*.

There are two main parts of *scanf* function as it can be seen from the above code. First part inside the double quotes is the list of format specifiers and second part is the list of variables with & sign at their left.

Scanned with CamScanner

EXAMPLE CODE 2.5

Example

```
# include <stdio.h>
void main ()
{
        int number;
        printf("Enter a number between 0-10: ");
        scanf("%d", &number);
        printf("The number you entered is: %d", number);
}
```

Output:

```
Enter a number between 0-10: 4
The number you entered is: 4
```

Important Note

We can take multiple inputs using a single scanf function e.g. look at the following statement.

```
scanf("%d%d%f", &a, &b, &c);
```

It takes input into two integer type variables *a* and *b*, and one float type variable *c*. After each input, user should enter a *space* or press *enter* key. After all the inputs user must press *enter* key.

ACTIVITY 2.3

Write a program that takes roll number, percentage of marks and grade from user as input. Program should display the formatted output like following:

```
Roll No      :          input value
Percentage   :          input value %
Grade        :          input value
```

Important Note

It is a very common mistake to forget & sign in the *scanf* function. Without & sign, the program gets executed but does not behave as expected.

Scanned with CamScanner

2.1.4 getch()

getch() function is used to read a character from user. The character entered by user does not get displayed on screen. This function is generally used to hold the execution of program because the program does not continue further until the user types a key. To use this function, we need to include the library *conio.h* in the header section of program.

</> EXAMPLE CODE 2.6

```
# include<stdio.h>
# include<conio.h>
void main ()
{
        printf("Enter any key if you want to exit program ");
        getch();
}
```

In the above, program prompts user to enter a character and then waits for the user's input before finishing the execution of program.

</> EXAMPLE CODE 2.7

```
# include<stdio.h>
# include<conio.h>
void main ()
{
        char key;
        printf("Enter any key : ");
        key = getch(); //Gets a character from user into variable key
}
```

If we run this program, we notice a difference between reading a character using *scanf* and reading a character using *getch* functions. When we read character through *scanf*, it requires us to press enter for further execution. But in case of *getch*, it does not wait for enter key to be pressed. Function reads a character and proceeds to the execution of next line.

Scanned with CamScanner

2.1.5 Statement Terminator

A statement terminator is identifier for compiler which identifies end of a line. In C language *semi colon (;)* is used as statement terminator. If we do not end each statement with a ; it results into error.

```
printf("Hello World");
```

; → Statement Terminator!

2.1.6 Escape Sequence

Purpose

Escape sequences are used in *printf* function inside the " and ". They force *printf* to change its normal behavior of showing output. Let's understand the concept of an escape sequence by looking at the following example statement:

```
printf("My name is \"Ali\"");
```

The output of above statement is

```
My name is "Ali"
```

In the above example \" is an escape sequence. It causes *printf* to display " on computer screen.

Formation of escape sequence

Escape sequences consist of two characters. The first character is always back slash (\) and the second character varies according to the functionality that we want to achieve. Back slash (\) is called escape character which is associated with each escape sequence to notify about escape. Escape character and character next to it are not displayed on screen, but they perform specific task assigned to them.

DID YOU KNOW?

Besides different escape sequences discussed in this section, following escape sequences are also commonly used in C language.

Sequence	Purpose	Sequence	Purpose
\'	Displays Single Quote(')	\a	Generates an alert sound
\\	Displays Back slash(\)	\b	Removes previous char

Scanned with CamScanner

New Line (\n)

After escape character, *n* specifies movement of the cursor to start of the next line. This escape sequence is used to print the output on multiple lines. Consider the following example to further understand this escape sequence:

EXAMPLE CODE 2.8

```
#include <stdio.h>
void main ()
{
        printf("My name is Ali. \n");
        printf("I live in Lahore.");
}
```

Output

```
My name is Ali.
I live in Lahore.
```

Important Note:

In the absence of an escape sequence, even if we have multiple *printf* statements, their output is displayed on a single line. Following example illustrates this point..

EXAMPLE CODE 2.9

```
#include<stdio.h>
void main()
{
        printf("My name is");
        printf(" Ahmad");
}
```

Output

```
My name is Ahmad
```

Tab (\t)

Escape sequence \t specifies the I/O function of moving to the next tab stop horizontally. A tab stop is collection of 8 spaces. Using \t takes cursor to the next tab stop. This escape sequence is used when user presents data with more spaces

Scanned with CamScanner

EXAMPLE CODE 2.10

```
#include<stdio.h>
void main ()
{
    printf("Name: \tAli\nFname: \tHammad\nMarks: \t1000");
}
```

Output

```
Name:   Ali
Fname:  Hammad
Marks:  1000
```

2.2 Operators

The name *computer* suggests that computation is the most important aspect of computers. We need to perform computations on data through programming. We have a lot of mathematical functions to perform calculations on data. We can also perform mathematical operations in our programs. C language offers numerous operators to manipulate and process data. Following is the list of some basic operator types:

- Assignment operator
- Arithmetic operators
- Logical operators
- Relational operators

2.2.1 Assignment Operator

Assignment operator is used to assign a value to a variable, or assign a value of variable to another variable.

Equal sign (=) is used as assignment operator in C. Consider the following example:

```
int sum = 5;
```

Value 5 is assigned to a variable named *sum* after executing this line of code.

Scanned with CamScanner

Let's have a look at another example:

```
int sum = 6;
int var = sum;
```

First, value 6 is assigned to variable *sum*. In the next line, the value of *sum* is assigned to variable *var*.

PROGRAMMING TIME 2.1

Write a program that swaps the values of two integer variables.

Program:

```
void main()
{
        int a = 2, b = 3, temp;
        temp = a;
        a = b;
        b = temp;
        printf("Value of a after swapping: %d\n", a);
        printf("Value of b after swapping: %d\n", b);
}
```

2.2.2 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on data. Table 2.2 represents arithmetic operators with their description.

Operator	Name	Description
/	Division Operator	It is used to divide the value on left side by the value on right side.
*	Multiplication Operator	It is used to multiply two values.
+	Addition Operator	It is used to add two values.
-	Subtraction Operator	It is used to subtract the value on right side from the value on left side.
%	Modulus Operator	It gives remainder value after dividing the left operand by right operand.

Table 2.2: Arithmetic Operators

Division

Division operator (/) divides the value of left operand by the value of right operand. e.g. look at the following statement.

Scanned with CamScanner

```
float result = 3.0 / 2.0;
```

After this statement, the variable *result* contains the value 1.5.

Important Note

If both the operands are of type *int*, then result of division is also of type *int*. Remainder is truncated to give the integer answer. Consider the following line of code:

```
float result = 3 / 2;
```

As both values are of type *int* so answer is also an integer which is 1. When this value 1 is assigned to the variable *result* of type *float*, then this 1 is converted to *float*, so value 1.0 is stored in variable *result*. If we want to get the precise answer then one of the operands must be of floating type. Consider the following line of code:

```
float result = 3.0 / 2;
```

In the above example, the value stored in variable *result* is 1.5.

PROGRAMMING TIME 2.2

```
/*This program takes as input the price of a box of chocolates and
the total number of chocolates in the box. The program finds and
displays the price of one chocolate.*/
# include <stdio.h>
void main ()
{
    float box_price, num_of_chocolates, unit_price;
    printf ("Please enter the price of whole box of chocolates: ");
    scanf("%f", &box_price);
    printf ("Please enter the number of chocolates in the box: ");
    scanf("%f", &num_of_chocolates);
    unit_price = box_price / num_of_chocolates;
    printf("The price of a single chocolate is %f", unit_price);
}
```

Output:

```
Please enter the price of whole box of chocolates: 150
Please enter the number of chocolates in the box: 50
The price of a single chocolate is 3.000000
```


Scanned with CamScanner

Multiplication

Multiplication operator (*) is a binary operator which performs the product of two numbers. Look at the following statement:

```
int multiply = 5 * 5;
```

After the execution of statement, the variable *multiply* contains value 25.

PROGRAMMING TIME 2.3

```
/* Following program takes as input the length and width of a rectangle. Program calculates and displays the area of rectangle on screen. */
# include<stdio.h>
void main ()
{
        float length, width, area;
        printf("Please enter the length of rectangle: ");
        scanf("%f", &length);
        printf("Please enter the width of rectangle: ");
        scanf("%f", &width);
        area = length * width;
        printf("Area of rectangle is : %f", area);
}
```

Output

```
Please enter the length of rectangle: 6.5
Please enter the length of rectangle: 3
Area of rectangle is : 19.500000
```

ACTIVITY 2.4

Write a program that takes as input the length of one side of a square and calculates the area of square.

Scanned with CamScanner

Addition

Addition operator (+) calculates the sum of two operands. Let's look at the following statement:

```
int add = 10 + 10;
```

Resultant value in variable *add* is 20.

PROGRAMMING TIME 2.4

```
/* This program takes marks of two subjects from user and displays
the sum of marks on console. */
#include <stdio.h>
void main ()
{
        int sum, math, science;
        printf ("Enter marks of Mathematics: ");
        scanf ("%d", &math);
        printf ("Enter marks of Science: ");
        scanf ("%d", &science);
        sum = math + science;
        printf ("Sum of marks is : %d", sum);
}
```

Output

```
Enter marks of Mathematics:  90
Enter marks of Science: 80
Sum of marks is : 170
```

ACTIVITY 2.5

Write a program that takes as input the number of balls in jar A and the number of balls in jar B. The program calculates and displays the total number of balls.

IMPORTANT TIP

The statement `a = a + 1;` is used to increase the value of variable *a* by 1. In C language, this statement can also be written as `a++;` or `++a;`. Similarly, `a--;` or `--a;` is used to decrease the value of *a* by 1.

Scanned with CamScanner

Subtraction

Subtraction operator (–) subtracts right operand from the left operand. Let's look at the following statement:

```
int result = 20 - 15;
```

After performing subtraction, value 5 is assigned to the variable *result*.

ACTIVITY 2.6

Write a program that takes original price of a shirt and discount percentage from user. Program should display the original price of shirt, discount on price and price after discount.

Modulus operator

Modulus operator (%) performs division of left operand by the right operand and returns the remainder value after division. Modulus operator works on integer data types.

```
int remaining = 14 % 3;
```

As, when we divide 14 by 3, we get a remainder of 2, so the value stored in variable *remaining* is 2.

PROGRAMMING TIME 2.5

```
/* This program finds and displays the right most digit of an input
number. */
#include <stdio.h>
void main()
{
        int num, digit;
        printf("Enter a number: ");
        scanf("%d", &num);
        digit = num % 10;
        printf("Right most digit of number you entered is: %d",
        digit);
}
```

Output

```
Enter a number: 789
Right most digit of number you entered is : 9
```


Scanned with CamScanner

ACTIVITY 2.7

Write a program that takes 2 digit number from user, computes the product of both digits and show the output.

ACTIVITY 2.8

Write a program that takes seconds as input and calculates equivalent number of hours, minutes and seconds.

Important Note:

While writing arithmetic statements in C language, a common mistake is to follow the usual algebraic rules e.g. writing 6 * y as 6y, and writing x * x * x as x^3 etc. It results in a compiler error.

ACTIVITY 2.9

Convert the following algebraic expressions into C expressions.

$$x = 6y + z$$

$$x = yz^3 + 3y$$

$$z = x + \frac{y^2}{3x}$$

$$z = (x - 2)^2 + 3y$$

$$y = (x + \frac{3z}{2}) + z^3 + \frac{x}{z}$$

2.2.3 Relational Operators

Relational operators compare two values to determine the relationship between values. Relational operators identify either the values are equal, not equal, greater than or less than one another. C language allows us to perform relational operators on numeric and char type data. Table 2.3 presents relational operators in C language and their descriptions:

Scanned with CamScanner

Relational Operator	Description
= =	Equal to
! =	Not equal
>	Greater than
<	Less than
> =	Greater than equal to
< =	Less than equal to

Table 2.3: Basic relational operators with their description

Relational operators perform operations on two operands and return the result in Boolean expression (*true* or *false*). A *true* value is represented by 1, whereas a *false* value is represented by a 0. This concept is further illustrated in Table 2.4.

Relational Expression	Explanation	Result
5 = = 5	5 is equal to 5?	True
5 ! = 7	5 is not equal to 7?	True
5 > 7	5 is greater than 7?	False
5 < 7	5 is less than 7?	True
5 > = 5	5 is greater than or equal to 5?	True
5 < = 4	5 is less than or equal to 4?	False

Table 2.4: Illustration of relational operators with examples

2.2.4 Assignment operator (=) and equal to operator (==):

In C language, == operator is used to check for equality of two expressions, whereas = operator assigns the result of expression on right side to the variable on left side. Double equal operator (==) checks whether right and left operands are equal or not. Single equal operator (=) assigns right operand to the variable on left side.

Important Note

We can also use printf function to show the results of a relational expression, e.g. look at the following examples:

```
printf("%d", 5 == 5); // This statement displays 1
printf("%d", 5 > 7);  // This statement displays 0
```


Scanned with CamScanner

ACTIVITY 2.10

Consider the variables x=3, y=7. Find out the Boolean result of following expressions.

(2 + 5) > y	*(x + 4) == y*
x != (y - 4)	*(y / 2) >= x*
-1 < x	*(x * 3) <= 20*

2.2.5 Logical Operators

Logical operators perform operations on Boolean expressions and produce a Boolean expression as a result.

As we have studied that result of a relational operation is a Boolean expression, so logical operators can be performed to evaluate more than one relational expressions. Table 2.5 shows the logical operators offered by C language.

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Table 2.5: Basic logical operators and their description

AND operator (&&):

AND operator && takes two Boolean expressions as operands and produces the result *true* if both of its operands are *true*. It returns *false* if any of the operands is *false*. Table 2.6 shows the **truth table** for AND operator.

Expression	Result
False && False	False
False && True	False
True && False	False
True && True	True

Table 2.6: Truth table for AND operator

Scanned with CamScanner

OR operator (||):

OR operator accepts Boolean expression and returns *true* if at least one of the operands is *true*. Table 2.7 shows the truth table for OR operator.

<table>
<tr><th>Expression</th><th>Result</th></tr>
<tr><td>False || False</td><td>False</td></tr>
<tr><td>False || True</td><td>True</td></tr>
<tr><td>True || False</td><td>True</td></tr>
<tr><td>True || True</td><td>True</td></tr>
</table>

Table 2.7: Truth table for OR operator

NOT operator (!):

NOT operator negates or reverses the value of Boolean expression. It makes it *true*, if it is *false* and *false* if it is *true*. Table 2.8 presents the truth table for Not operator.

Expression	Result
!(True)	False
!(False)	True

Table 2.8: Truth table for NOT operator

Examples of Logical Operators:

Table 2.9 illustrates the concept of logical operators with the help of examples.

<table>
<tr><th>Logical Expression</th><th>Explanation</th><th>Result</th></tr>
<tr><td>3 < 4 && 7 > 8</td><td>3 is less than 4 AND 7 is greater than 8?</td><td>False</td></tr>
<tr><td>3 == 4 || 3 > 1</td><td>3 is equal to 4 OR 3 is greater than 1?</td><td>True</td></tr>
<tr><td>! (4 > 2 || 2 == 2)</td><td>NOT (4 is greater than 2 OR 2 is equal to 2)?</td><td>False</td></tr>
<tr><td>6 <= 6 && !(1 > 2)</td><td>6 is less than or equal to 6 AND NOT (1 is greater than 2)?</td><td>True</td></tr>
<tr><td>8 > 9 || !(1 <=0)</td><td>8 is greater than 9 OR NOT (1 is less than or equal to 0)?</td><td>True</td></tr>
</table>

Table 2.9: Illustration of logical operators with examples

Scanned with CamScanner

DID YOU KNOW?

C language performs **short-circuit evaluation**. It means that:

1- While evaluating an AND operator, if sub expression at left side of the operator is *false* then the result is immediately declared as *false* without evaluating complete expression.

2- While evaluating an OR operator, if sub expression at left side of the operator is *true* then the result is immediately declared as true without evaluating complete expression.

ACTIVITY 2.11

Assume the following variable values x=4, y=7, z=8. Find out the resultant expression.

```
x== 2  ||  y==8          7>= y   &&  z<5
z>=5   ||  x<= -3        y==7   &&  !(true)
x!=y   ||  y<5           !(z>x)
```

2.2.6 Unary vs Binary Operators:

All the operators discussed in this chapter can be divided into two basic types, based on the number of operands on which the operator can be applied.

Unary Operators: Unary operators are applied over one operand only e.g. logical not (!) operator has only one operand. Sign operator (-) is another example of a unary operator e.g. -5.

Binary Operators: Binary operators require two operands to perform the operation e.g. all the arithmetic operators, and relational operators are binary operators. The logical operators && and || are also binary operators.

DID YOU KNOW?

C language also offers a ternary operator that is applied on three operands.

Scanned with CamScanner

2.2.7 Operators' Precedence:

If there are multiple operators in an expression, the question arises that which operator is evaluated first. To solve this issue, a precedence has been given to each operator (Table 2.10). An operator with higher precedence is evaluated before the operator with lower precedence. In case of equal precedence, the operator at left side is evaluated before the operator at right side.

Example:

```
result = 18  / 2 * 3 + 7 % 3 + ( 5 * 4);     // evaluate ( )
result = 18  / 2 * 3 + 7 % 3 + 20;           // evaluate /
result = 9 * 3 + 7 % 3 + 20;                 // evaluate *
result = 27 + 7 % 3 + 20;                    // evaluate %
result = 27 + 1 + 20;                        // evaluate +
result = 28 + 20;                            // evaluate +
result = 48;
```

<table>
<tr><th>Operator</th><th>Precedence</th></tr>
<tr><td>()</td><td>1</td></tr>
<tr><td>!</td><td>2</td></tr>
<tr><td>* , / , %</td><td>3</td></tr>
<tr><td>+ , -</td><td>4</td></tr>
<tr><td>> , < , >= , <=</td><td>5</td></tr>
<tr><td>== , !=</td><td>6</td></tr>
<tr><td>& &</td><td>7</td></tr>
<tr><td>| |</td><td>8</td></tr>
<tr><td>=</td><td>9</td></tr>
</table>

Table 2.10: Operators with their precedence

ACTIVITY 2.12

Find out the results of the following Expressions:

16 / (5 + 3)
7 + 3 * (12 + 2)
25 % 3 * 4
34 – 9 * 2 / (3 * 3)
18 / (15 – 3 * 2)

Scanned with CamScanner

SUMMARY

- We need a way to provide input and show output while writing programs. Each programming language has its keywords or standard built-in function for I/O operations.
- ***printf*** is a built-in function in C programming language. It's name comes from "print formatted" that is used to show the formatted output on screen.
- ***Format specifiers*** are used to specify format of data type during input and output operations. Format specifier is always preceded by a percentage (%) sign.
- ***scanf*** is a built-in function in C language that takes input from user into the variables
- ***getch()*** function is used to read a character from user. This function accepts characters only. The character entered by user does not get displayed on screen.
- A ***statement terminator*** is identifier for compiler which identifies end of a statement. In C language *semi colon (;)* is used as statement terminator.
- ***Escape sequence*** forces printf to escape from its normal behavior. It is the combination of escape character(\) and a character associated with special functionality.
- Escape sequence *n* specifies the movement of cursor to start of the next line. This escape sequence is used to display the output on multiple lines.
- Escape sequence *t* specifies the movement of cursor to the next tab stop horizontally. A tab stop is collection of 8 spaces.
- ***Basic operators*** are arithmetic operators, assignment operator, relational operators, and logical operators.
- ***Arithmetic operators*** Arithmetic operators are used to perform arithmetic operations on data to evaluate arithmetic functions. Arithmetic operators are +, -, *, /, %.

Scanned with CamScanner

- ***Modulus operator*** is also a binary operator, which performs division of left operand to the right operand and returns the remainder value after division. Modulus operator works on integer data types.
- ***Relational operators*** compare two values to determine the relationship between values.
- ***Logical operator*** performs operation on Boolean expressions and returns a Boolean value as a result.
- ***Logical AND operator*** returns *true* when the result of expressions on both sides is *true* whereas the ***Logical OR operator*** returns *true* when either of the two expressions is true.
- ***Logical NOT operator*** returns *true* if the expression is *false* and vice versa.
- ***Short circuiting*** is to deduce the result of an operation without computing the whole expression.
- There are three types of operators. ***Unary, binary and ternary operators*** require one, two and three operands respectively, to perform the operation.
- ***Precedence*** tells which operation should be performed first. Different operators have different precedence. Operators with higher precedence are evaluated first and the ones with lowest precedence are evaluated last.

Scanned with CamScanner

Exercise

Q.1 Multiple Choice Questions

1) printf is used to print __________ type of data.

a) *int* b) *float* c) *char* d) All of them

2) scanf is a __________ in C programming language.

a) Keyword b) library c) function d) none of them

3) getch() is used to take __________ as input from user.

a) *int* b) *float* c) *char* d) All of them

4) Let the following part of code, what will be the value of variable ***a*** after execution:

```
int a = 4;
float b = 2.2;
a = a * b;
```

a) 8.8 b) 8 c) 8.0 d) 8.2

5) Which of the following is a valid line of code.

a) int = 20; b) grade= 'A'; c) line= this is a line; d) none of them

6) Which operator has highest precedence among the following:

a) / b) = c) > d) !

7) Which of the following is not a type of operator:

a) Arithmetic operator c) Relational operator
b) Check operator d) Logical operator

8) The operator % is used to calculate __________.

a) Percentage b) Remainder c) Factorial d) Square

9) Which of the following is a valid character:

a) 'here' b) "a" c) '9' d) None of them

10) What is true about C language:

a) C is not a case sensitive language
b) Keywords can be used as variable names
c) All logical operators are binary operators
d) None of them

Scanned with CamScanner

Q.2 True or False

1) Maximum value that can be stored by an integer is 32000. T/F
2) Format specifiers begin with a % sign. T/F
3) Precedence of division operator is greater than multiplication operator. T/F
4) *getch* is used to take all types of data input from user. T/F
5) *scanf* is used for output operations. T/F

Q.3 Define the following.

1) Statement Terminator 2) Format Specifier 3) Escape Sequence
4) scanf 5) Modulus Operator

Q.4 Briefly answer the following questions.

1) What is the difference between *scanf* and *getch*?
2) Which function of C language is used to display output on screen?
3) Why format specifiers are important to be specified in I/O operations?
4) What are escape sequences? Why do we need them?
5) Which operators are used for arithmetic operations?
6) What are relational operators? Describe with an example.
7) What are logical operators? Describe with an example.
8) What is the difference between unary operators and binary operators?
9) What is the difference between == operator and = operator?
10) What is meant by precedence of operators? Which operator has the highest precedence in C language?

Q.5 Write down output of the following code segments.

a)
```
# include<stdio.h>
void main ()
{
        int x = 2, y = 3, z = 6;
        int ans1, ans2, ans3;
        ans1 = x / z * y ;
        ans2 = y + z / y * 2;
        ans3 = z / x + x * y;
        printf("%d %d %d", ans1, ans2, ans3 );
}
```


Scanned with CamScanner

b)
```
# include<stdio.h>
void main ()
{
    printf ( "nn \n\n nnn\nn\nt\t" ) ;
    printf ( "nn /n/n nn/n\n" ) ;
}
```

c)
```
#include<stdio.h>
void main()
{
    int a = 4, b;
    float c = 2.3;
    b = c * a;
    printf("%d", b);
}
```

d)
```
#include<stdio.h>
void main()
{
    int a = 4 * 3 / ( 5 + 1 ) + 7 % 4;
    printf("%d", a);
}
```

e)
```
#include<stdio.h>
void main()
{
    printf("%d", ((( 5 > 3 ) && (4 > 6) ) || (7 > 3)) );
}
```

Q.6 Identify errors in the following code segments.

a)
```
#include<stdio.h>
void main ()
{
    int a , b = 13;
    b = a % 2;
    printf("Value of b is : %d, b);
}
```


Not For Sale- PESI

Scanned with CamScanner

b)
```
#include<stdio.h>
void main ()
{
        int a , b , c,
        printf("Enter First Number: ");
        scanf("%d", &a);
        printf("Enter second number : ");
        scanf("%d", &b);
        a + b = c;
]
```

c)
```
#include<stdio.h>;
main ()
{
        int num;
        printf(Enter number: ");
        scanf(%d, &num);
};
include<stdio.h>
int main ()
{
        float f;
        printf["Enter value: "];
        scanf("%c", &f);
}
```

Programming Exercises

Exercise 1

The criteria for calculation of wages in a company is given below.

Basic Salary	*= Pay Rate Per Hour*	X	*Working Hours Of Employee*
Overtime Salary	*= Overtime Pay Rate*	X	*Overtime Hours Of Employee*
Total Salary	*= Basic Salary*	+	*Overtime Salary*

Write a program that should take working hours and overtime hours of employee as input. The program should calculate and display the total salary of employee.

Scanned with CamScanner

Exercise 2

Write a program that takes Celsius temperature as input, converts the temperature into Fahrenheit and shows the output. Formula for conversion of temperature from Celsius to Fahrenheit is:

$$F = \frac{9}{5}C + 32$$

Exercise 3

Write a program that displays the following output using single *printf* statement:

```
*       *       *       *
1       2       3       4
```

Exercise 4

Write a program that displays the following output using single *printf* statement:

I am a Boy
I live in Pakistan
I am a proud Pakistani

Exercise 5

A clothing brand offers 15% discount on each item. A lady buys 5 shirts from this brand. Write a program that calculates total price after discount and amount of discount availed by the lady. Original prices of the shirts are:

Shirt1 = 423
Shirt2 = 320
Shirt3 = 270
Shirt4 = 680
Shirt5 = 520

Note: Use 5 variables to store the prices of shirts.

Exercise 6

Write a program that swaps the values of two integer variables without help of any third variable.

Exercise 7

Write a program that takes a 5 digit number as input, calculates and displays the sum of first and last digit of number.

Scanned with CamScanner

Exercise 8

Write a program that takes monthly income and monthly expenses of the user like electricity bill, gas bill, food expense. Program should calculate the following:

- Total monthly expenses
- Total yearly expenses
- Monthly savings
- Yearly saving
- Average saving per month
- Average expense per month

Exercise 9

Write a program that takes a character and number of steps as input from user. Program should then jump number of steps from that character.

Sample output:

Enter character: a
Enter steps: 2
New character : c

Exercise 10

Write a program that takes radius of a circle as input. The program should calculate and display the area of circle.

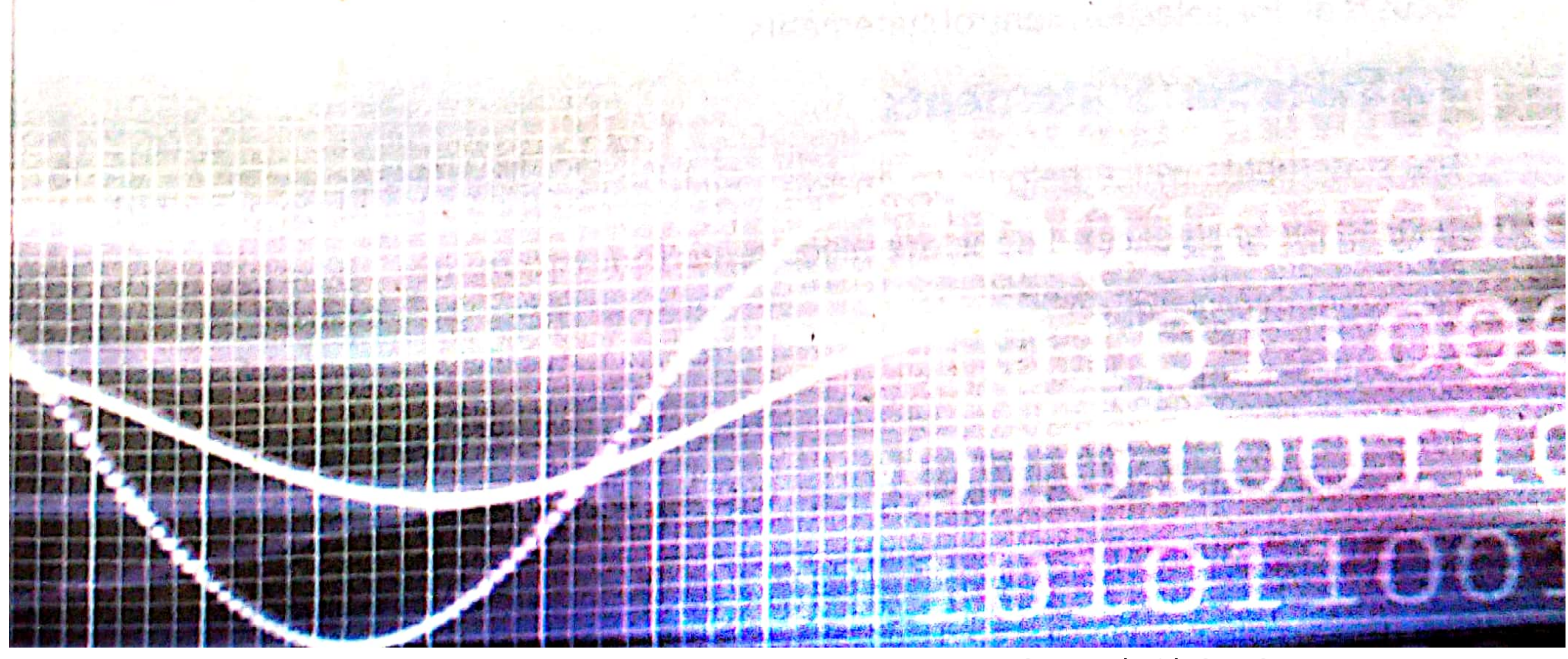
Scanned with CamScanner

CONDITIONAL LOGIC

Students Learning Outcomes

After completing this unit students will be able to

- Define a control statement
- Define a selection statement
- Know the structure of *if statement*
- Use *if statement*
- Know the structure of *if-else statement*
- Use nested selection structures



Scanned with CamScanner

Unit Introduction

In our daily life, we often do certain tasks depending upon the situation e.g. I will go for a walk if I wake up at 6 am. If the weather turns cloudy, I will take umbrella with me. If Sara passes the exam, I will gift her a watch. All these decisions are taken on the basis of condition. If the condition is true, we perform the specified task, otherwise we do not. Sometimes, if the condition is not true then we perform some other task. This is called ***conditional logic***. In this chapter, we discuss how to implement conditional logic in C programming language.

3.1 Control Statements

In order to solve a problem, there is a need to control the flow of execution of a program. Sometimes we need to execute one set of instructions if a particular condition is true and another set of instructions if the condition is false. Moreover, sometimes we need to repeat a set of statements for a number of times. We can control the flow of program execution through control statements. There are three types of control statements in C language.

1- Sequential Control Statements

2- Selection Control Statements

3- Repetition Control Statements

Sequential control is the default control structure in C language. According to the sequential control, all the statements are executed in the given sequence. Till now, we have just worked according to the sequential control. In this chapter, our focus is on the selection control statements.

3.2 Selection Statements

The statements which help us to decide which statements should be executed next, on the basis of conditions, are called ***selection statements***.

Two types of selection statements are:

1. If statement
2. If-else statement

Scanned with CamScanner

3.2.1 If statement

C language provides ***if statement*** in which we specify a condition, and associate a code to it. The code gets executed if the specified condition turns out to be *true*, otherwise the code does not get executed.

Structure of if statement

If statement has the following structure in C language:

```
if (condition)
        Associated Code
```

Here is a brief description of different components involved in the general structure of *if statement*.

1- In the given structure, ***if*** is a keyword that is followed by a *condition* inside parentheses **( )**.

2- A ***condition*** could be any valid expression including arithmetic expressions, relational expressions, logical expressions, or a combination of these. Here are a few examples of valid expressions that can be used as condition.

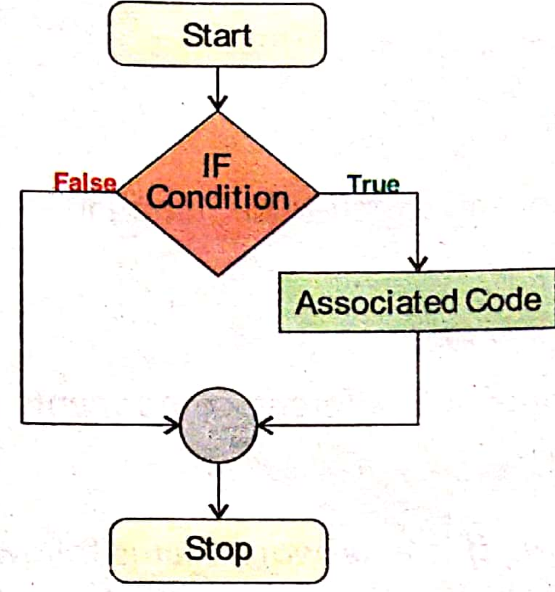
a-	5	(true)
b-	5 + 4	(true)
c-	5 – 5	(false)
d-	5 > 4	(true)
e-	5 == 4	(false)
f-	! (4 > 5)	(true)
g-	(5 > 4) && (10 < 9)	(false)
h-	(5 > 4) || (9 < 10)	(true)

Any expression that has a non-zero value calculates to *true*, e.g. expressions *a* and *b* above produce a *true* value, but the expression *c* produces a *false* value. The expression can also include variables, in that case values inside the variables are used to calculate the true/ false value of the expression.

Scanned with CamScanner

3- The ***associated code*** is any valid C language set of statements. It may contain one or more statements.

The following flow chart shows the basic flow of an *if statement*.



If we want to associate more than one statements to an *if statement*, then they need to be enclosed inside a { } block, but if we want to associate only one statement, then although it may be enclosed inside { } block, but it is not mandatory. It is demonstrated through the following examples.

</> EXAMPLE CODE 3.1

```
#include<stdio.h>
void main()
{
        int a = 12;
        if (a % 2 == 0)
        {
                printf("The variable a contains an even value.");
                printf("\nYou are doing a great job.");
        }
}
```

Output:

```
The variable a contains an even value.
You are doing a great job.
```


Scanned with CamScanner

Because, when value 12 is divided by 2, it gives a remainder equal to 0, so the condition inside if parentheses is true. As both the *printf* statements are inside { } block, so both the statements get executed.

Now look at the following example:

EXAMPLE CODE 3.2

```
#include<stdio.h>
void main()
{
        int a = 4;
        int b = 5;
        if (a > b)
                printf("The value of a is greater than b.");
        printf("\nYou are doing a great job.");
}
```

Output:

```
You are doing a great job.
```

As the condition inside the *if parentheses* is false, and the statements following the *if statement* are not inside { } block, so only the 2nd statement is executed because without a { } block, only 1st statement is considered to be associated with the *if statement*.

Using if statement in C

Let's understand the concept of if statement using different examples.

PROGRAMMING TIME 3.1

Problem:

Write a program in C language that takes the percentage of student as an input and displays "PASS" if the percentage is above 50.

```
#include <stdio.h>
void main()
{
        float percentage;
        printf ("Enter the percentage: ");
```


Scanned with CamScanner

Continued

```
    scanf ("%f", &percentage);
    if (percentage > 50)
        printf ("PASS\n");
}
```

Output:

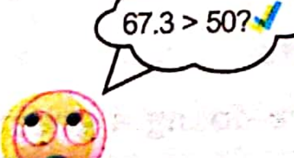
On the input 47, program simply ends because 47 is less than 50 and the condition turns *false*.

```
Enter the percentage : 47
```

47 > 50?

When 67.3 is entered as an input, "PASS" gets printed on console because condition is *true*, as 67.3 is greater than 50.

```
Enter the percentage : 67.3
PASS
```



PROGRAMMING TIME 3.2

Problem:

A marketing firm calculates the salary of its employees according to the following formula.

Gross Salary = Basic Salary + (Number of Items Sold X 8) + Bonus

If the number of sold items are more than 100 and the number of broken items are 0, then bonus is Rs. 10000, otherwise bonus is 0.

Write a program that takes basic salary, the number of sold and broken items as input from user, then calculates and displays the gross salary of the employee.

Scanned with CamScanner

Continued

Program:

```
#include<stdio.h>
void main()
{
        int  basic_salary,  items_sold,  items_broken,
        gross_salary;
        int bonus = 0;
        printf("Enter the basic salary: ");
        scanf("%d", &basic_salary);
        printf("Enter the number of items sold: ");
        scanf("%d", &items_sold);
        printf("Enter the number of items broken: ");
        scanf("%d", &items_broken);
        if (items_sold > 100 && items_broken == 0)
                bonus = 10000;
        gross_salary = basic_salary + (items_sold * 8) + bonus;
        printf("Gross  salary  of  the  employee  is  %d",
        gross_salary);
}
```

Description:

In the above example, bonus is initialized to 0 because if the number of sold items are not more than 100, then automatically bonus is considered 0. Inside the *if statement*, it is checked that whether the number of sold items are greater than 100. If so, the bonus is assigned 10000. It is to be noted that gross salary is calculated outside the if block, because whether the number of sold items are more than 100 or not, the gross salary must be calculated.

Scanned with CamScanner

Following figure explains the flow of program with the help of a flow chart.

Start

Bonus=0

INPUT
basic salary,
items sold,
items broken

ARE
items sold > 100
&& items broken == 0

bonus=10000

gross salary = basic salary + (items sold * 8) + bonus

Print
gross salary

Stop

ACTIVITY 3.1

Write a program that takes the age of a person as an input and displays "Teenager" if the age lies between 13 and 19.

ACTIVITY 3.2

Write a program that takes year as input and displays "Leap Year" if the input year is leap year. Leap years are divisible by 4.

Not For Sale- PESRP

Scanned with CamScanner

IMPORTANT TIP

Properly indent the instructions under *if statement* using tab. It improves the readability of the program.

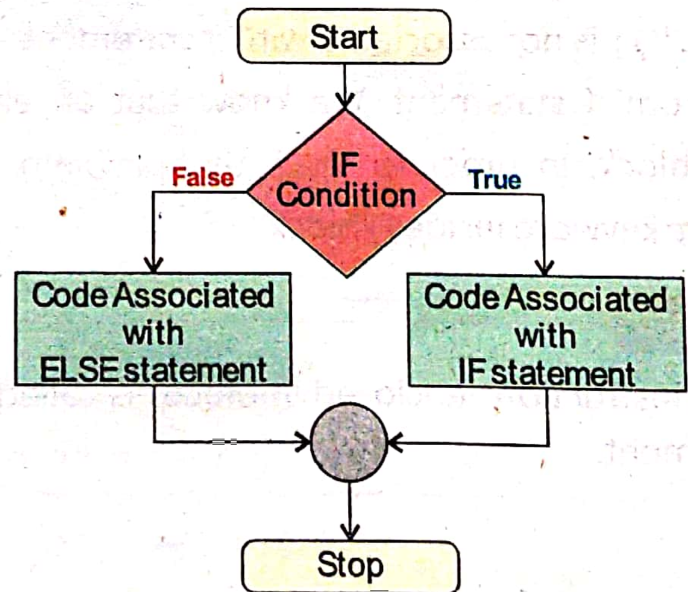
3.2.2 If-else Statement

Till now, we have demonstrated how to execute a set of instructions if a particular condition is *true*, but if the condition turns out to be *false* then we are not doing anything. What if we want to execute one set of instructions if a particular condition is *true* and another set of instructions if the condition is *false*. In such situations we use *if-else statement*. It executes the set of statements under if statement if the condition is true, otherwise executes the set of statements under else statement.

General structure of the *if-else statement* is as follows:

```
if (condition)
        Associated Code
else
        Associated Code
```

Associated code of *if statement* is executed if the condition is *true*, otherwise the code associated with *else* statement is executed. Following flow chart shows the structure of *if-else statement*.



Scanned with CamScanner

Important Note

An *if statement* may not have an associated *else* statement, but an *else* statement must have an *if statement* to which it is associated.

Before *else* keyword, if there are multiple statements under *if*, then they must be enclosed inside the { } block, otherwise compiler issues an error. In order to understand this concept, let's look at the following example.

</> EXAMPLE CODE 3.3

```
#include<stdio.h>
void main()
{
        int a = 15;

        if (a % 2 == 0)
                printf("The variable a contains an even value.");
                printf("\nYou are doing a great job.");
        else
                printf("The variable a contains an odd value.");
}
```

ERROR: else without an associated if

The above code cannot be compiled, because as discussed earlier, without a { } block only one statement is associated with *if* statement. In this case, the 1st statement i.e. `printf("The variable a contains an even value.");` is associated with *if* statement, but the 2nd statement i.e. `printf("\nYou are doing a great job.");` is not associated with *if* statement. So, the *else* part is also disconnected from if statement. We know that an else block must be associated to an *if* block. In order to solve this problem, we can put both statements before *else* keyword inside { } block.

DID YOU KNOW?

A set of multiple instructions enclosed in braces is called a block or a compound statement.

Scanned with CamScanner

Following code demonstrates this.

</> EXAMPLE CODE 3.4

```
#include<stdio.h>
void main()
{
        int a = 15;
        if (a % 2 == 0)
        {
                printf("The variable a contains an even value.");
                printf("\nYou are doing a great job.");
        }
        else
                printf("The variable a contains an odd value.");
}
```

Output:

```
The variable a contains an odd value.
```

Use of if-else statement

The following program takes a character as input and displays "DIGIT" if the character entered by user is a digit between 0 to 9, otherwise displays "NOT DIGIT".

</> EXAMPLE CODE 3.5

```
#include <stdio.h>
void main()
{
        char input;
        printf ("Please enter a character: ");
        scanf ("%c", &input);
        if (input >= '0' && input <= '9')
                printf ("DIGIT\n");
        else
                printf ("NOT DIGIT\n");
}
```


Scanned with CamScanner

Output: Continued

If user enters a digit, e.g. 5, then "DIGIT" is displayed on the screen.

```
Please enter a character: 5
DIGIT
```

If user enters another character, e.g. *k*, then "NOT DIGIT" is displayed as the condition turns *false*.

```
Please enter a character: k
NOT DIGIT
```

ACTIVITY 3.3

Write a program that takes the value of body temperature of a person as an input and displays "You have fever." if body temperature is more than 98.6 otherwise displays "You don't have fever."

IMPORTANT TIP

If there are more than one instructions under *if statement* or *else statement*, enclose them in the form of a block. Otherwise, the compiler considers only one instruction under it and further instructions are considered independent.

Important Note

C language also provides an *if-else-if* statement that has the following structure.

```
if (condition 1)
    Code to execute if condition 1 is true;
else if (condition 2)
    Code to execute if condition 1 is false but condition 2 is true;
................................
else if (condition N)
    Code to execute if all previous conditions are false but condition N is true;
else
    Code to execute if all the conditions are false;
```


Scanned with CamScanner

PROGRAMMING TIME 3.3

Problem:

Write a program that takes percentage marks of student as input and displays his grade. Following table shows grades distribution criteria.

Percentage	Grade
80% and above	A
70% - 80%	B
60% - 70%	C
50% - 60%	D
Below 50%	F

Program:

```
#include<stdio.h>
void main()
{
      float percentage;
      printf ("Enter the percentage: ");
      scanf ("%f", &percentage);
      if (percentage >= 80)
            printf ("A\n");
      else if (percentage >= 70)
            printf ("B\n");
      else if (percentage >= 60)
            printf ("C\n");
      else if (percentage >= 50)
            printf ("D\n");
      else
            printf ("F\n");
}
```


Not For Sale- PESRP
Scanned with CamScanner

3.2.3 Nested Selection Structures

Let's closely observe the general structure of an if-else statement given below:

```
if (condition)
        Associated Code
else
        Associated Code
```

The code associated with an *if statement* or with an *else statement* can be any valid C language set of statements. It means that inside an *if* block or inside an *else* block, we can have other *if statements* or *if-else statements*. It also means that inside those inner *if statements* or *if-else statements* we can have even more *if statements* or *if-else statements* and so on. Conditional statements within conditional statements are called *nested selection structures*.

All the following structures are valid nested selection structures.

if (condition1 is true)
    if (condition2 is true)
        Associated code
    else
        Associated code
	if (condition1 is true)
	    if (condition2 is true)
	        Associated code
	else
	    if (condition3 is true)
	        Associated code

if (condition1 is true)
    if (condition2 is true)
        Associated code
    else
        Associated code
else
    if (condition3 is true)
        Associated code
	if (condition1 is true)
	    if (condition2 is true)
	        Associated code
	    else
	        Associated code
	else
	    if (condition3 is true)
	        Associated code
	    else
	        Associated code

Scanned with CamScanner

Use of Nested Selection Structures

In order to understand the usage of nested selection structures, let's have a look at the following example problem.

PROGRAMMING TIME 3.4

Problem:

An electricity billing company calculates the electricity bill according to the following formula.

Bill Amount = Number of Units Consumed X Unit Price

There are two types of electricity users i.e. Commercial and Home Users. For home users the unit price varies according to the following:

Units Consumed	Unit Price
Units < = 200	Rs 12
Units > 200 but Units < = 400	Rs 15
Units > 400	Rs 20

For commercial users, the unit price varies according to the following:

Units Consumed	Unit Price
Units < = 200	Rs 15
Units > 200 but Units < = 400	Rs 20
Units > 400	Rs 24

Write a program that takes the type of consumer and number of units consumed as input. The program then displays the electricity bill of the user.

Program:

```
#include<stdio.h>
void main()
{
	int units, unit_price, bill;
	char user_type;
	printf("Please enter h for home user and c for commercial
	user: ");
	scanf("%c", &user_type);
	printf("Please enter the number of units consumed: ");
	scanf("%d", &units);
```


Scanned with CamScanner

Continued

```
        if(units <= 200)
                if(user_type == 'h')
                        unit_price = 12;
                else if(user_type == 'c')
                        unit_price = 15;
        else if(units > 200 && units <= 400)
                if(user_type == 'h')
                        unit_price = 15;
                else if(user_type == 'c')
                        unit_price = 20;
        else
                if(user_type == 'h')
                        unit_price = 15;
                else if(user_type == 'c')
                        unit_price = 24;
        bill = units * unit_price;
        printf("Your electricity bill is %d", bill);
}
```

IMPORTANT TIP

In compound statements, it is a common mistake to omit one or two braces while typing. To avoid this error, it is better to type the opening and closing braces first and then type the statements in the block.

ACTIVITY 3.4

The eligibility criteria of a university for its different undergraduate programs is as follows:

BSSE Program: 80% or more marks in Intermediate

BSCS Program: 70% or more marks in Intermediate

BSIT Program: 60% or more marks in Intermediate

Otherwise the university do not enroll a student in any of its programs. Write a program that takes the percentage of Intermediate marks and tells for which programs the student is eligible to apply.

Scanned with CamScanner

DID YOU KNOW?

C programming language also provides Switch-Case structure to deal with conditions, but Switch-Case structure is applicable only in limited scenarios. The if, if-else structures cover all the possible decision making scenarios.

ACTIVITY 3.5

Write a program that takes two integers as input and asks the user to enter a choice from 1 to 4. The program should perform the operation according to the given table and display the result.

Choice	Operation
1	Addition
2	Subtraction
3	Multiplication
4	Division

3.2.4 Solved Example Problems

PROGRAMMING TIME 3.5

Problem:

Write a program that displays larger one out of the three given numbers.

Program:

```
include <stdio.h>
void main()
{
      int n1, n2, n3;
      printf ("Enter three numbers");
      scanf ("%d%d%d", &n1, &n2, &n3);
      if (n1 > n2 && n1 > n3)
            printf ("The largest number is %d", n1);
      else if (n2 > n3 && n2 > n1)
            printf ("The largest number is %d", n2);
      else
            printf ("The largest number is %d", n3);
}
```


Scanned with CamScanner

PROGRAMMING TIME 3.6

Problem:

Write a program that calculates the volume of cube, cylinder or sphere, according to the choice of user.

Program:

```
#include<stdio.h>
void main()
{
        int choice;
        float volume;
        printf ("Find Volume\n");
        printf ("1.Cube\n2.Cylinder\n3.Sphere\nEnter your choice: ");
        scanf ("%d", &choice);
        if (choice == 1)
        {
                float length;
                printf ("Enter Length: ");
                scanf ("%f", &length);
                volume = length * length * length;
                printf ("Volume is %f", volume);
        }
        else if (choice == 2)
        {
                float length1, radius1;
                printf ("Enter Length: ");
                scanf ("%f", &length1);
                printf ("Enter Radius: ");
                scanf ("%f", &radius1);
                volume = 3.142 * radius1 * radius1 * length1;
                printf ("Volume is %f", volume);
        }
```


Scanned with CamScanner

Continued

```
    else if (choice == 3)
    {
        float radius;
        printf ("Enter Radius: ");
        scanf ("%f", &radius);
        volume = 3.142 * radius * radius * radius;
        printf ("Volume is %f", volume);
    }
    else
        printf ("Invalid Choice");
}
```

ACTIVITY 3.6

Write a program that finds and displays area of a triangle, parallelogram, rhombus or trapezium according to the choice of user.

Scanned with CamScanner

- The flow of program execution is controlled through ***control statements.***
- ***Sequential control*** is the default control structure in C language. According to the sequential control, all the statements are executed in the given sequence.
- The statements which help us to decide which statements should be executed next, on the basis of conditions, are called ***selection statements***.
- In ***if statement*** we specify a condition, and associate a code to it. The code gets executed if the specified condition turns out to be true, otherwise the code does not get executed.
- A ***condition*** could be any valid expression including arithmetic expressions, relational expressions, logical expressions, or a combination of these.
- The ***associated code*** in if statement is any valid C language set of statements.
- ***If-else statement*** executes the set of statements under *if statement* if the condition is *true* and executes the set of statements under *else* otherwise.
- An *if statement* may not have an associated *else* statement, but an *else* statement must have an *if statement* to which it is associated.
- Selection statements within selection statements are called ***nested selection structures.***

Scanned with CamScanner

Exercise

Q1 Multiple Choice Questions

1) Conditional logic helps in ______________.
   a) decisions b) iterations c) traversing d) all
2) ____________________ statements describe the sequence in which statements of the program should be executed.
   a) Loop b) Conditional c) Control d) All
3) In *if statement*, what happens if *condition* is *false*?
   a) Program crashes
   b) Index out of bound error
   c) Further code executes
   d) Compiler asks to change condition
4)
```
int a = 5;
if (a < 10)
        a++;
else
        if (a > 4)
                a--;
```
   Which of the following statements will execute?
   a) a++; b) a--; c) both (a) and (b) d) None
5) Which of the following is the condition to check *a* is a factor of *c*?
   a) *a % c == 0* b) *c % a == 0* c) a*c==0 d) a+c==0
6) A *condition* can be any ________________ expression.
   a) *arithematic* b) *relational*
   c) logical d) arithematic, relational or logical
7) An *if* statement inside another *if* statement is called __________ structure.
   a) *nested* b) *boxed* c) repeated d) decomposed
8) A set of multiple instructions enclosed in braces is called a ____________.
   a) box b) *list* c) block d) job

Q2 Define the following.

1) Control Statements 2) Selection Statements 3) Sequential Control
4) Condition 5) Nested Selection Structures

Scanned with CamScanner

Q3 Briefly answer the following questions.

1) Why do we need selection statements?
2) Differentiate between sequential and selection statements.
3) Differentiate between *if statement* and *if else* statement with an example.
4) What is the use of nested selection structures?
5) Write the structure of *if statement* with brief description.

Q4 Identify errors in the following code segments. Assume that variables have already been declared.

a)
```
if (x ≥ 10)
        printf ("Good");
```

b)
```
if (a < b && b < c);
        sum = a + b + c;
else
        multiply = a * b * c;
```

c)
```
if (a < 7 < b)
        printf ("7");
```

d)
```
if (a == b &| x == y)
        flag = true;
else
        flag = false;
```

e)
```
if (sum == 60 || product == 175)
        printf ("Accepted %c), sum);
else
        if (sum >= 45 || product > 100)
                printf ("Considered %d" + sum);
        else
                printf ("Rejected");
```


Scanned with CamScanner

Q5 Write down output of the following code segments.

a)
```
int a = 7, b = 10;
a = a + b;
if ( a > 20 && b < 20 )
        b = a + b;
printf ( " a = %d, b = %d", a, b);
```

b)
```
int x = 45;
if (x + 20 * 7 == 455)
        printf ("Look's Good");
else
        printf ("Hope for the Best");
```

c)
```
char c1 = 'Y', c2 = 'N';
int n1 = 5, n2 = 9;
n1 = n1 + 1;
c1 = c2;
if (n1 == n2 && c1 == c2)
        printf ("%d = %d and %c = %c", n1, n2, c1, c1);
else
        if (n1 < n2 && c1 == c2)
                printf ("%d < %d and %c = %c", n1, n2, c1, c2);
        else
                printf ("Better Luck Next Time!");
```


Scanned with CamScanner

d)
```
int a = 34, b = 32, c = 7, d = 15;
 a = b + c + d;
 if ( a < 100 )
        a = a * 2;
 b = b * c;
 c = c + d;
 if ( a > b && c == d )
 {
        c = d;
        b = c;
        a = b;
 }
 else
        if (a > b && c > d || b >= d + c)
        {
               d = c * c;
               a = b * b;
        }
 printf ("a=%d, b=%d, c=%d, d=%d", a, b, c, d);
```

e)
```
int x = 5, y = 7, z = 9;
if ( x % 2 == 0)
       x++;
else
       x = y + z;
printf (" x = %d\n", x);
if ( x % 2 == 1 && y % 2 == 1 && z % 2 == 1)
       printf ("All are Odd");
if ( x > y || x < z )
{
       if ( x > y )
              y++;
       else
              if ( x < z )
                     x++;
}
printf ("x = %d, y = %d, z = %d", x, y, z);
```


Scanned with CamScanner

Programming Exercises

Exercise 1

Write a program that takes two integers as input and tells whether first one is a factor of the second one?

Exercise 2

Write a program that takes a number as input and displays "YES" if the input number is a multiple of 3, and has 5 in unit's place e.g. 15, 75.

Exercise 3

Following is the list of discounts available in "Grocery Mart".

Total Bill	Discount
1000	10%
2500	20%
5000	35%
10000	50%

Write a program that takes *total bill* as input and tells how much discount the user has got and what is the discounted price.

Exercise 4

Write a program that takes as input, the *original price* and *sale price* of a product and tells whether the product is sold on *profit* or *loss*. The program should also tell the profit/loss percentage.

Exercise 5

Write a program that takes as input, the lengths of 3 sides of a triangle and tells whether it is a right angle triangle or not. For a right angled triangle,

$$\text{hypotenuse}^2 = \text{base}^2 + \text{height}^2.$$

Exercise 6

Following is the eligibility criteria for admission in an IT University.

- At least 60% marks in Matric.
- At least 65% marks in Intermediate (Pre-Engineering or ICS).
- At least 65% marks in entrance test.

Scanned with CamScanner

Write a program that takes as input, the obtained and total marks of Matric, Intermediate and Entrance Test. The program should tell whether the student is eligible or not.

Exercise 7

Write a program that calculates the bonus an employee can get on the following basis:

Salary	Experience with Company	Bonus Tasks	Bonus
10000	2 Years	5	1500
10000	3 Years	10	2500
25000	3 Years	4	2000
75000	4 Years	7	3500
100000	5 Years	10	5000

The program should take as input, the salary, experience and number of bonus tasks of the employee. The program should display the bonus on the screen.

Scanned with CamScanner

Unit # 4

DATA AND REPETITION

Students Learning Outcomes

After completing this unit students will be able to

- Understand the structure of array
- Declare and use one dimensional arrays
- Use variable as an index in array
- Read and write values in array
- Explain the concept of loop structure
- Know that for loop structure is composed of:
  - For
  - Initialization expression
  - Test expression
  - Body of the loop
  - Increment/decrement expression
- Explain the concept of a nested loop
- Use loops to read and write data in array

Scanned with CamScanner

Unit Introduction

While writing computer programs, we may find situations where we need to process large quantities of data. The techniques that we have learnt so far may not seem suitable in these situations. So we need to have better mechanisms for storage and processing of large amounts of data. Another common problem that we face is how to repeat a set of instructions for multiple times without writing them again and again. This chapter discusses the ways that C programming language provides in order to deal with data and repetitions.

4.1 Data Structures

In the previous chapters, we learnt how to store pieces of data in the variables. What if we need to store and process large amount of data e.g. the marks of 100 students? Probably, we need to declare 100 variables, which does not seem an appropriate solution. So, high level programming languages provide ***data structures*** in order to store and organize data. A data structure can be defined as follows:

> Data structure is a container to store collection of data items in a specific layout.

Different data structures are available in C programming language, however this chapter discusses only one of them, which is called ***Array***. It is one of the most commonly used data structures.

4.1.1 Array

An array is a data structure that can hold multiple values of same data type e.g. an *int* array can hold multiple integer values, a *float* array can hold multiple real values and so on. An important property of array is that it stores all the values at consecutive locations inside the computer memory.

Scanned with CamScanner

4.1.2 Array Declaration

In C language, an array can be declared as follows:

type of data to be stored in array | maximum number of elements that can be stored

data_type array_name[array_size];

identifier of array

If we want to declare an array of type *int* that holds the daily wages of a laborer for seven days, then we can declare it as follows:

```
int daily_wage[7];
```

Following is the example of the declaration of a *float* type array that holds marks of 20 students.

```
float marks[20];
```

4.1.3 Array Initialization

Assigning values to an array for the first time, is called array initialization. An array can be initialized at the time of its declaration, or later. Array initialization at the time of declaration can be done in the following manner.

data_type array_name[N] = {value1, value2, value3,........, valueN};

Following example demonstrates the declaration and initialization of a float array to store the heights of seven persons.

Scanned with CamScanner

```
float height[7] = {5.7, 6.2, 5.9, 6.1, 5.0, 5.5, 6.2};
```

Here is another example that initializes an array of characters to store five vowels of English language.

```
char vowels[5] = {'a', 'e', 'i', 'o', 'u'};
```

Important Note:

If we do not initialize an array at the time of declaration, then we need to initialize the array elements one by one. It means that we cannot initialize all the elements of array in a single statement. This is demonstrated by the following example.

</> EXAMPLE CODE 4.1

```
void main()
{
    int array[5];
    array[5] = {10, 20, 30, 40, 50};
}
```

ERROR initializing whole array after declaration not allowed

The compiler generates an error on the above example code, as we try to initialize the whole array in one separate statement after declaring it.

4.1.4 Accessing array elements

Each element of an array has an *index* that can be used with the array name as *array_name[index]* to access the data stored at that particular *index*.

First element has the index 0, second element has the index 1 and so on. Thus *height[0]* refers to the first element of array *height*, *height[1]* refers to the second element and so on. Figure 4.1 shows graphical representation of array *height* initialized in the last section.

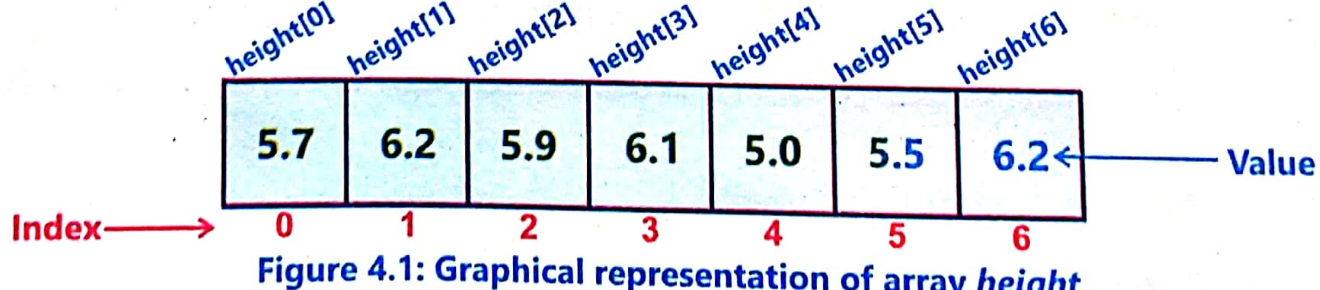


Figure 4.1: Graphical representation of array *height*

Scanned with CamScanner

PROGRAMMING TIME 4.1

Write a program that stores the ages of five persons in an array, and then displays on screen.

Solution:

```
#include<stdio.h>
void main()
{
	int age[5];
	/* Following statements assign values at different indices of array age. We can see that the first value is stored at index 0 and the last value is stored at index 4 */
	age[0] = 25;
	age[1] = 34;
	age[2] = 29;
	age[3] = 43;
	age[4] = 19;
	/* Following statement displays the ages of five persons stored in the array */
	printf("The ages of five persons are: %d, %d, %d, %d, %d", age[0], age[1], age[2], age[3], age[4]);
}
```

PROGRAMMING TIME 4.2

Write a program that takes the marks obtained in 4 subjects as input from the user, calculates the total marks and displays on screen.

Solution:

```
#include<stdio.h>
void main()
{
	float marks[4], total_marks;
	printf("Please enter the marks obtained in 4 subjects: ");
	scanf("%f%f%f%f",&marks[0], &marks[1], &marks[2], &marks[3]);
	total_marks = marks[0] + marks[1] + marks[2] + marks[3];
	printf("Total marks obtained by student are %f", total_marks);
}
```


Scanned with CamScanner

4.1.5 Using variables as array indexes

A very important feature of arrays is that we can use variables as array indices e.g. look at the following program:

</> EXAMPLE CODE 4.2

```
#include<stdio.h>
void main()
{
        int array[5] = {10, 20, 30, 40, 50};
        int i;
        /* Following statements ask the user to input an index
        into variable i. */
        printf("Please enter the index whose value you want to
        display");
        scanf("%d", &i);
        /* Following statement displays the value of the array
        at the index entered by user. */
        printf("The value is %d", array[i]);
}
```

Following program demonstrates that when we change the value of a variable, its later usage uses the updated value.

</> EXAMPLE CODE 4.3

```
#include<stdio.h>
void main()
{
        int array[5] = {10, 20, 30, 40, 50};
        int i = 2;
        /* Following statement displays value 30, as i contains
        2 and the value at array[2] is 30 */
        printf("%d", array[i]);
        i++;
        /* Following statement displays value 40, as i has been
        incremented to 3 and the value at array[3] is 40. */
        printf("\n%d", array[i]);
}
```


Scanned with CamScanner

4.2 Loop Structure

If we need to repeat one or more statements, then we use loops. For example, if we need to write Pakistan thousand times on the screen, then instead of writing `printf("Pakistan");` a thousand times, we use loops. C language provides three kind of loop structures:

1- For loop

2- While loop

3- Do While loop

In this chapter, our focus is on *for loops*.

4.2.1 General structure of loops

If we closely observe the process that humans follow for repeating a task for specific number of times then it becomes easier for us to understand the loop structures that C language provides us for controlling the repetitions.

Let's assume that our sports instructor asks us to take 10 rounds of the running track. How do we perform this task? First we set a counter to zero, because we have not yet taken a single round of the track. Then we start taking the rounds. After each round we increase our counter by 1 and check whether we have completed 10 rounds or not yet. If we have not yet completed the 10 rounds then we again take a round, increase our counter by 1, and again check whether we have taken 10 rounds or not. We repeat this process till our counter reaches 10.

Different programming languages follow similar philosophy in the loop structures for repeating a set of instructions.

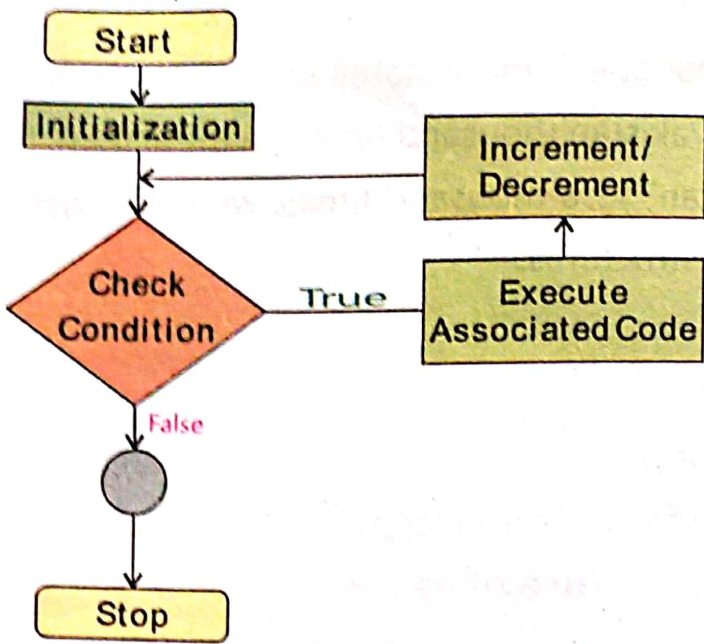
4.2.2 General syntax of for loop

In C programming language, for loop has the following general syntax.

```
for(initialization; condition; increment/decrement)
{
        Code to repeat
}
```


Scanned with CamScanner

In order to understand the *for* loop structure let's look at the following flow chart.



From the flow chart, we can observe the following sequence:

1- *Initialization* is the first part to be executed in a *for* loop. Here we initialize our counter variable and then move to the *condition* part.

2- *Condition* is checked, and if it turns out to be *false*, then we come out of loop.

3- If the *condition* is *true*, then *body of the loop* is executed.

4- After executing the body of loop, the counter variable is increased or decreased depending on the used logic, and then we again move to the step 2.

After executing the *body of loop*, the counter variable is increased or decreased depending on the used logic, and then we again move to the step 2.

</> EXAMPLE CODE 4.4

```
for(int i = 0; i < 3; i++)
{
     printf("Pakistan\n");
}
```

Output:

```
Pakistan
Pakistan
Pakistan
```


Scanned with CamScanner

Continued

Description:

If we observe the written code and compare it to the flowchart description, we can see the following sequence of execution.

1- *Initialization* expression is executed, i.e. *int i = 0*. Here counter variable *i* is declared and initialized with value *0*.

2- *Condition* is tested, i.e. *i<3*. As variable *i* has value *0* which is less than *3* so *condition* turns out to be *true*, and we move to the *body of the loop*.

3- *Loop body* is executed, i.e. *printf("Pakistan\n");* thus *Pakistan* is displayed on screen.

4- *Increment/decrement* expression is executed, i.e. *i++*. Thus the value of *i* is incremented by 1. As variable *i* had value *0*, so after this statement *i* contains value *1*.

5- Now *condition* is again tested. Because, value of *i* is *1* which is less than *3* so *condition* again turns out to be *true* and *loop body* is executed again i.e. *Pakistan* is again displayed on screen. The value of *i* gets incremented to *2*.

6- Now *condition* is again tested. Because, value of *i* is *2* which is less than *3* so *Pakistan* is again displayed on screen. The value of *i* gets incremented to 3.

7- *Now condition is again tested. Because, value of i is 3 which is not less than 3, so the condition turns out to be false and control comes out of the loop.*

PROGRAMMING TIME 4.3

Write a program that displays the values from 1 – 10 on the computer screen.

Program:

```
for(int i = 1; i <= 10; i++)
{
    printf("%d\n", i);
}
```


Scanned with CamScanner

Continued

Description:

Consider the example program given above.

- First of all, the value of *i* is set to *1* and then *condition* is checked.
- As the *condition* is *true (1<=10)* so *loop body* executes. As in the *loop body*, we are displaying the value of the counter variable, so *1* is displayed on console.
- After increment, the value of *i* becomes *2*. The *condition* is again checked. It is *true* as *(2<=10)* so this time *2* is printed.
- The procedure continues till *10* is displayed and after increment the value of *i* becomes *11*. *Condition* is checked and it turns out to be *false (11>10)* so the loop finally terminates after printing the numbers from *1* to *10*.

ACTIVITY 4.1

Write a program that displays the table of 2.

IMPORTANT TIP

Always make sure that the condition becomes false at some point, otherwise the loop repeats infinitely and never terminates.

DID YOU KNOW?

Each run of a loop is called an iteration.

PROGRAMMING TIME 4.4

Write a program that calculates the factorial of a number input by user.

Program Logic:

When we want to solve a problem programmatically, first we need to know exactly what we want to achieve. In this example, we are required to find the factorial of a given number, so first we need to know the formula to find factorial of a number.

$$N! = 1 * 2 * 3 * 4 * ..... * (N-1) * N$$

We can see the pattern that is being repeated, so we can solve the problem using for loop.

Scanned with CamScanner

Continued

Program:

```
#include<stdio.h>
void main()
{
        int n, fact = 1;
        printf("Please enter a positive number whose factorial
        you want to find");
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
        {
                fact = fact * i;
        }
        printf("The factorial of input number %d is %d", n,
        fact);
}
```

Description:

Following table shows the working of program, if the input number is 5. It demonstrates the changes in the values of variables at each iteration.

Iteration	Value of counter	Condition	Loop body	Result
				fact=1
1	i = 1	TRUE (1<=5)	fact = fact * i	fact=1*1=1
2	i = 2	TRUE (2<=5)	fact = fact * i	fact=1*2=2
3	i = 3	TRUE (3<=5)	fact = fact * i	fact=2*3=6
4	i = 4	TRUE (4<=5)	fact = fact * i	fact=6*4=24
5	i = 5	TRUE (5<=5)	fact = fact * i	fact=24*5=120
6	i = 6	FALSE (6>5)		

4.2.3 Nested Loops

Let's carefully observe the general structure of a loop.

```
for(initialization; condition; increment/decrement)
{
        Code to repeat
}
```


Scanned with CamScanner

We can observe that *Code to repeat* could be any valid C language code. It can also be another *for* loop e.g. the following structure is a valid loop structure.

```
for(initialization; condition; increment/decrement)
{
        for(initialization; condition; increment/decrement)
        {
                Code to repeat
        }
}
```

When we use a loop inside another loop, it is called nested loop structure.

When do we use nested loops?

When we want to repeat a pattern for multiple times, then we use nested loops, e.g. if 10 times we want to display the numbers from 1 – 10. We can do this by writing the code of displaying the numbers from 1 – 10 in another loop that runs 10 times.

PROGRAMMING TIME 4.5

Problem:

Write a program that 5 times displays the numbers from 1 – 10 on computer screen.

Program:

```
#include<stdio.h>
void main()
{
        for(int i = 1; i <= 5; i++)
        {
                for(int j = 1; j <= 10; j++)
                {
                        printf("%d ", j);
                }
                printf("\n");
        }
}
```


Scanned with CamScanner

Continued

Output:

Here is the output of above program.

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Description:

As we understand the working of inner loop, so here let's focus on outer loop.

1- For the value *i = 1, condition* in outer loop is checked which is *true* (1 <= 5), so whole inner loop is executed and numbers from 1 – 10 are displayed.

2- When control gets out of inner loop, `printf("\n");` is executed which inserts a new line on console.

3- Then *i* is incremented and it becomes *2*. As it is less than 5, so condition is *true*. The whole inner loop is executed, and thus numbers from 1 – 10 are again displayed on screen. Coming out of the inner loop *new line* is inserted again.

4- After five times displaying the numbers from 1 – 10 on screen, the value of *i* gets incremented to *6* and *condition* of outer loop turns *false*. So outer loop also terminates.

PROGRAMMING TIME 4.6

Problem:

Write a program to display the following pattern of stars on screen.

```
*
**
***
****
*****
******
```

Continued

Program:

```
#include<stdio.h>
void main()
{
      for(int i = 1; i <= 6; i++)
      {
            for(int j = 1; j <= i; j++)
                  printf("*");
            printf("\n");
      }
}
```

Description:

Here is the description of above code.

1- As we have to display 6 lines containing stars, so we run the outer loop from 1 to 6.

2- We can observe that in the given pattern we have 1 star on 1st line, 2 stars on 2nd line, 3 stars on 3rd line and so on. So, the inner loop is dependent on the outer loop, i.e. if counter of outer loop is 1 then inner loop should run 1 time, if the counter of outer loop is 2 then inner loop should run 2 times and so on. So, we use the counter of outer loop in the termination condition of inner loop i.e. j <= i.

3- When outer loop counter *i* has value 1, inner loop only runs 1 time, so only 1 star is displayed. When outer loop counter is 2, the inner loop runs 2 times, so 2 stars are displayed and the process is repeated until six lines are complete.

ACTIVITY 4.2

Write a program that displays the tables of 2, 3, 4, 5 and 6.

IMPORTANT TIP

We can use *if* structures inside loop structures or loop structures inside *if* structures in any imaginable manners.

Scanned with CamScanner

4.2.4 Solved Example Problems

PROGRAMMING TIME 4.7

Problem:

Write a program that counts multiples of a given number lying between two numbers.

Program:

```
#include <stdio.h>
void main ()
{
        int n, lower, upper, count = 0;
        printf ("Enter the number: ");
        scanf ("%d", &n);
        printf ("Enter the lower and upper limit of
        multiples:\n");
        scanf ("%d%d", &lower, &upper);
        for(int i = lower; i <= upper; i++)
                if(i % n == 0)
                        count++;
        printf ("Number of multiples of %d between %d and %d are
        %d", n, lower, upper, count);
}
```

PROGRAMMING TIME 4.8

Problem:

Write a program to find even numbers in integers ranging from *n1* to *n2* (where *n1* is greater than *n2*).

Program:

```
#include <stdio.h>
void main ()
{
        int n1, n2;
        printf ("Enter the lower and upper limit of even
        numbers:\n");
        scanf ("%d%d", &n2, &n1);
```

Continued

```
    if(n1 > n2)
    {
        for (int i = n1; i >= n2; I--)
        {
            if(i % 2 == 0)
                printf ("%d ", i);
        }
    }
}
```

PROGRAMMING TIME 4.9

Problem:

Write a program to determine whether a given number is prime number or not.

Program:

```
#include <stdio.h>
void main ()
{
    int n;
    int flag = 1;
    printf ("Enter a number: ");
    scanf ("%d", &n);
    for (int i = 2; i < n; i++)
    {
        if (n % i == 0)
            flag = 0;
    }
    if (flag == 1)
        printf ("This is a prime number");
    else
        printf ("This is not a prime number");
}
```


Scanned with CamScanner

PROGRAMMING TIME 4.10

Problem:

Write a program to display prime numbers ranging from 2 to 100.

Program:

```
# include<stdio.h>
int main ()
{
        int flag;
        for (int j = 2; j <= 100; j++)
        {
                flag = 1;
                for (int i = 2; i < j; i++)
                {
                        if(j % i == 0)
                        {
                                flag = 0;
                        }
                }
                if (flag == 1)
                {
                        printf ("%d ", j);
                }
        }
}
```

4.2.5 Loops and Arrays

As variables can be used as array indexes, so we can use loops to perform different operations on arrays. If we want to display the whole array, then instead of writing all the elements one by one, we can loop over the array elements by using the loop counter as array index.

Scanned with CamScanner

In the following, we discuss how loops can be used to read and write values in arrays.

1) **Writing values in Arrays using Loops:** Using loops, we can easily take input in arrays. If we want to take input from user in an array of size 10, we can simply use a loop as follows:

</> EXAMPLE CODE 4.5

```
int a[10];
for (int i = 0; i < 10; i++)
        scanf ("%d", &a[i]);
```

PROGRAMMING TIME 4.11

Problem:

Write a program that assigns first 5 multiples of 23 to an array of size 5.

Program:

```
#include<stdio.h>
void main()
{
        int multiples[5];
        for (int i = 0; i < 5; i++)
                multiples[i]= (i + 1) * 23 ;
}
```

2) **Reading values from Arrays using Loops:** Let's see how loops help us in reading the values from array. The following code can be used to display the elements of an array having 100 elements:

</> EXAMPLE CODE 4.6

```
        for (int i = 0; i < 100; i++)
                printf("%d ", a[i]);
```

The following code can be used to add all the elements of an array having 100 elements.

Scanned with CamScanner

EXAMPLE CODE 4.7

```
int sum = 0;
for(int i = 0; i < 100; i++)
      sum = sum + a[i];
printf("The sum of all the elements of array is %d", sum);
```

ACTIVITY 4.3

Write a program that takes as input the marks obtained in matriculation by 30 students of a class. The program should display the average marks of the class.

4.2.6 Solved Example Problems

PROGRAMMING TIME 4.12

Problem:

Write a program that adds corresponding elements of two arrays.

Program:

```
#include <stdio.h>
void main ()
{
      int a[] = {2, 3, 54, 22, 67, 34, 29, 19};
      int b[] = {65,73,26, 10, 4, 2, 84, 26};
      for (int i=0; i<8; i++)
            printf ("%d   ", a[i] + b[i]);
}
```


Scanned with CamScanner

- ***Data structure*** is a container to store collection of data items in a specific layout.
- An ***Array*** is a data structure that can hold multiple values of same data type. It stores all the values at contiguous locations inside the computer's memory.
- In C language, an array can be declared as follows:

  data_type array_name[array_size];

  - ***Data Type*** is the type of data that we want to store in the array.
  - ***Array Name*** is the unique identifier that we use to refer to the array.
  - ***Array Size*** is the maximum number of elements that the array can hold.
- Assigning values to an array for the first time, is called ***Array Initialization***. An array can be initialized at the time of its declaration, or later. Array initialization at the time of declaration can be done in the following manner.

  data_type array_name[N] = {value1, value2, value3,..., valueN};
- Each element of an array has an ***index*** that can be used with the array name as *array_name[index]* to access the data stored at that particular *index*. Variables can also be used as array indices.
- ***Loop Structure*** is used to repeat a set of statements. Three types of loops are for loop, while loop, do-while loop.
- In C programming language, *for loop* has the following general structure.

```
for(initialization; condition; increment/decrement)
{
        Code to repeat;
}
```

- When we use a loop inside another loop, it is called nested loop structure. We use nested loops to repeat a pattern multiple times.
- Loops make it easier to read and write values in arrays.

Scanned with CamScanner

Exercise

Q1 Multiple Choice Questions

1) An array is a ______ structure.
   a) Loop b) Control c) Data d) Conditional
2) Array elements are stored at ______ memory locations.
   a) Contiguous b) Scattered c) Divided d) None
3) If the size of an array is 100, the range of indexes will be ______.
   a) 0-99 b) 0-100 c) 1-100 d) 2-102
4) ______ structure allows repetition of a set of instructions.
   a) Loop b) Conditional c) Control d) Data
5) ______ is the unique identifier, used to refer to the array.
   a) Data Type b) Array name c) Array size d) None
6) Array can be initialized ______ declaration.
   a) At the time of b) After c) Before d) Both a & b
7) Using loops inside loops is called ______ loops.
   a) For b) While c) Do-while d) Nested
8) ______ part of *for* loop is executed first.
   a) Condition b) Body
   c) Initialization d) Increment/Decrement
9) ______ make it easier to read and write values in array.
   a) Loops b) Conditions c) Expressions d) Functions
10) To initialize the array in a single statement, initialize it ______ declaration.
   a) At the time of b) After c) Before d) Both a & b

Scanned with CamScanner

Q2 Define the following terms.

1) Data Structure
2) Array
3) Array Initialization
4) Loop Structure
5) Nested Loops

Q3 Briefly answer the following Questions.

1) Is loop a data structure? Justify your answer.
2) What is the use of nested loops?
3) What is the advantage of initializing an array at the time of declaration?
4) Describe the structure of a *for* loop.
5) How can you declare an array? Briefly describe the three parts of array declaration.

Q4 Identify the errors in the following code segments.

a)
```
int a[] = ({2},{3},{4});
```

b)
```
for (int i = 0, i < 10, i++)
        printf ("%d\n", i);
```

c)
```
int a[] = {1,2,3,4,5};
for (int j = 0; j < 5; j++)
        printf ("%d ", a(j));
```

d)
```
float f[] = {1.4, 3.5, 7.3, 5.9};
int size = 4;
for (int n = -1; n < size; n--)
        printf ("%f\n", f[n]);
```

e)
```
int count = 0;
for (int i = 4; i < 6; i--)
        for (int j = i, j < 45; j++)
        {
                count++;
                printf ("%count", count)
        }
```

Scanned with CamScanner

Q5 Write down output of the following code segments.

a)
```
int sum = 0, p;
for (p = 5; p <= 25; p = p + 5)
        sum = sum + 5;
printf ("Sum is %d", sum);
```

b)
```
int i;
for (i = 34; i <= 60; i = i * 2)
        printf ("* ");
```

c)
```
for (int i = 50; i <= 50; i++)
{
        for (j = i; j >= 48; j--)
                printf ("j = %d \ n", j);
        printf ("i = %d\n", i);
}
```

d)
```
int i, arr[] = {2, 3, 4, 5, 6, 7, 8};
for (i = 0; i < 7; i++)
{
        printf ("%d\n", arr[i] * arr[i]);
        i++;
}
```

e)
```
int i,j;
float ar1[] = {1.1, 1.2, 1.3};
float ar2[] = {2.1, 2.2, 2.3};
for (i = 0; i < 3; i++)
        for (j = i; j < 3; j++)
                printf ("%f\n", ar1[i] * ar2[j] * i * j);
```


Scanned with CamScanner

Programming Exercises

Exercise 1

Use loops to print the following patterns on console.

a)
```
*****
*****
*****
```

b)
```
A
BC
DEF
GHIJ
KLMN
```

Exercise 2

Write a program that takes two positive integers *a* and *b* as input and displays the value of a^b.

Exercise 3

Write a program that takes two numbers as input and displays their Greatest Common Divisor (GCD) using Euclidean method.

Exercise 4

Write a program to display factorials numbers from 1 to 7. (Hint: Use Nested Loops)

Exercise 5

Write a program that takes 10 numbers as input in an array and displays the product of first and last element on console.

Exercise 6

Write a program that declares and initializes an array of 7 elements and tells how many elements in the array are greater than 10.

Scanned with CamScanner

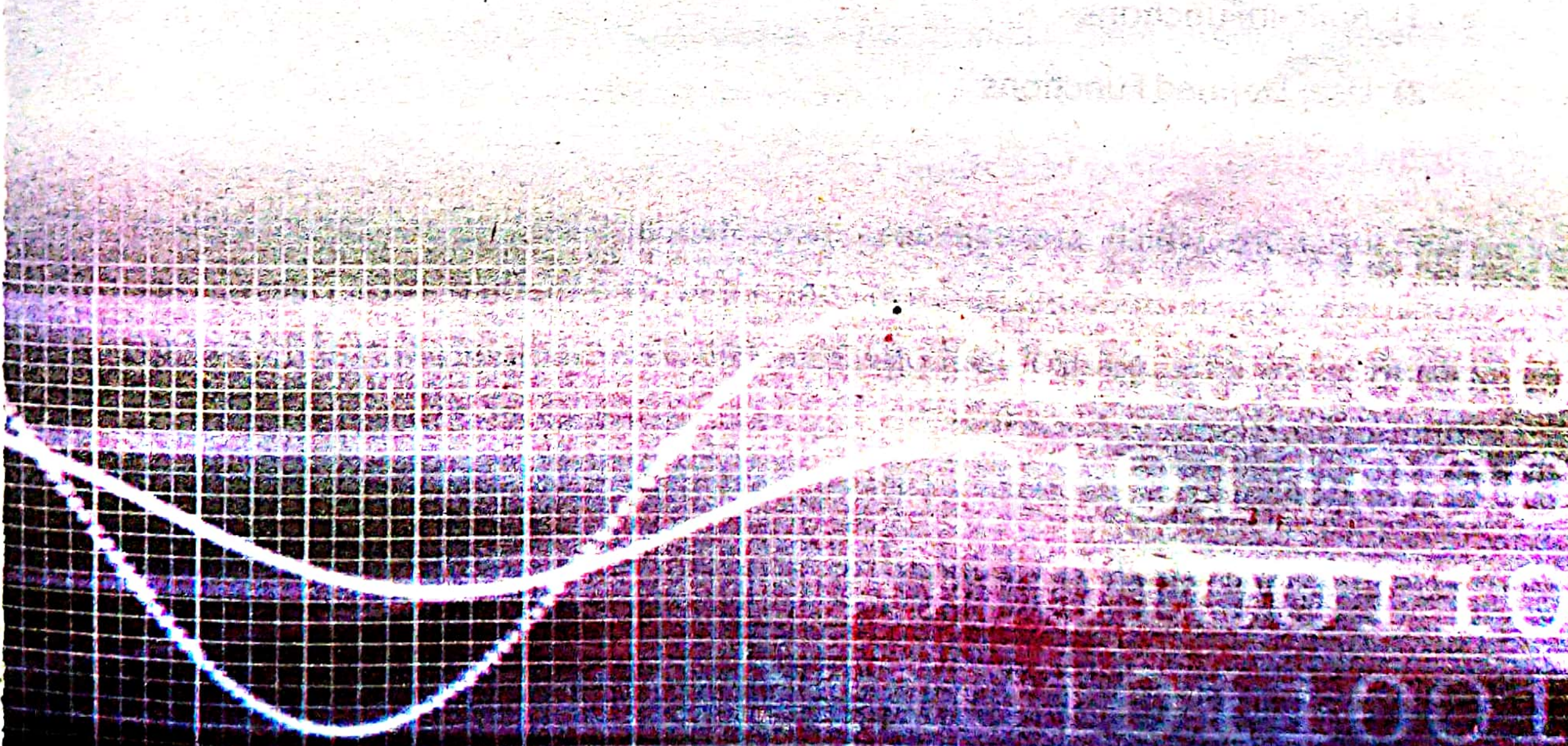
Unit # 5

FUNCTIONS

Students Learning Outcomes

After completing this unit students will be able to

- Explain the concept and types of functions
- Explain the advantages of using functions
- Explain the signature of function (Name, Arguments, Return type)
- Explain the following terms related to functions
    - Definition of a function
    - Use of a function



Scanned with CamScanner

Unit Introduction

A good problem solving approach is to divide the problem into multiple smaller parts or sub-problems. Solution of the whole problem thus consists of solving the sub-problems one by one, and then integrating all the solutions. In this way, it becomes easier for us to focus on a single smaller problem at a time, instead of thinking about the whole problem all the time. This problem solving approach is called *divide and conquer*. C programming language provides us with *functions* that allow us to solve a programming problem using the divide and conquer approach. In this chapter, we will learn the concept of functions, their advantages, and how to work with them.

5.1 Functions

A function is a block of statements which performs a particular task, e.g. *printf* is a function that is used to display anything on computer screen, *scanf* is another function that is used to take input from the user. Each program has a *main* function which performs the tasks programmed by the user. Similarly, we can write other functions and use them multiple times.

5.1.1 Types of Functions

There are basically two types of functions:

1) Built-inFunctions
2) User Defined Functions

Built-in Functions

The functions which are available in C Standard Library are called built-in functions. These functions perform commonly used mathematical calculations, string operations, input/output operations etc. For example, *printf* and *scanf* are built-in functions.

User Defined Functions

The functions which are defined by a programmer are called user-defined functions. In this chapter we will learn how to write user defined functions.

Scanned with CamScanner

5.1.2 Advantages of Functions

Functions provide us several advantages.

1) **Reusability:** Functions provide reusability of code. It means that whenever we need to use the functionality provided by the function, we just call the function. We do not need to write the same set of statements again and again.

2) **Separation of tasks:** Functions allow us to separate the code of one task from the code of other tasks. If we have a problem in one function, then we do not need to check the whole program for removing the problem. We just need to focus at one single function.

3) **Handling the complexity of the problem:** If we write the whole program as a single procedure, management of the program becomes difficult. Functions divide the program into smaller units, and thus reduce the complexity of the problem.

4) **Readability:** Dividing the program into multiple functions, improves the readability of the program.

5.1.3 Signature of a Function

A function is a block of statements that gets some inputs and provides some output. Inputs of a function are called **parameters** of the function, and output of the function is called its **return value**. A function can have multiple parameters, but it cannot return more than one values.

Function signature is used to define the inputs and output of a function. The general structure of a *function signature* is as follows:

function identifier

```
return_type function_name(data_type1, data_type2,...,data_typeN);
```

data type of return value

data types of function parameters

Scanned with CamScanner

Example Function Signatures:

Table 5.1 shows the descriptions of some functions and their signatures.

Function Description	Function Signature
A function that takes an integer as input and returns its square.	`int square (int);`
A function that takes length and width of a rectangle as input and returns the perimeter of the rectangle.	`float perimeter (float, float);`
A function that takes three integers as input and returns the largest value among them.	`int largest (int, int, int);`
A function that takes radius of a circle as input and returns the area of circle.	`float area (float);`
A function that takes a character as input and returns 1, if the character is a vowel, otherwise returns 0.	`int isVowel (char);`

Table 5.1: Some functions and their Signatures

5.1.4 Defining a Function

The function signature does not describe how the function performs the task assigned to it. Function definition does that. A function definition has the following general structure.

```
return_type function_name (data_type var1, data_type var2,.., data_type varN)
{
        Body of the function
}
```


Scanned with CamScanner

Body of the function is the set of statements which are executed in the function to perform the specified task. Just after the function's signature, the set of statements enclosed inside { } form the body of the function.

Following example defines a function *showPangram()* that does not take any input and does not return anything, but displays *A quick brown fox jumps over the lazy dog.* on computer screen.

</> EXAMPLE CODE 5.1

```
void showPangram()      // function name
{
    printf("\nA quick brown fox jumps over the lazy dog.\n");
}
```

As the above function does not return anything thus return type of the function is *void*.

Let's take another example of a function that takes as input two integers and returns the sum of both integers.

</> EXAMPLE CODE 5.2

```
int add(int x, int y)   // return type, function name, parameters of function
{
    int result;
    result = x + y;
    return result;
}
```

Inside the function, *return* is a keyword that is used to return a value to the calling function.

Important Note:

A function cannot return more than one values. e.g the following statement results in a compiler error.

```
return (4, 5);
```


Scanned with CamScanner

Important Note:

There may be multiple *return* statements in a function but as soon as the first *return* statement is executed, the function call returns and further statements in the body of function are not executed.

Using a Function

We need to call a function, so that it performs the programmed task. Following is the general structure used to make a function call.

```
function_name(value1, value2,…, valueN);
```

For example, let's observe the following program.

</> EXAMPLE CODE 5.3

```
void main()
{
        printf("Hello from main()");
        showPangram();  ← function call
        printf("Welcome back to main()");
}
```

Output:

```
Hello from main()
A quick brown fox jumps over the lazy dog.
Welcome back to main()
```

We can see that the program starts its execution from *main()* function. When it encounters a function call (inside the rectangle), it transfers the control to called function. After executing the statements of called function, the control is transferred back to the calling function, i.e. *main()* in the above example.

The following program inputs two numbers and displays their sum.

The statement inside the rectangle in the following code includes a call to the *add* function defined in previous section.

Scanned with CamScanner

</> EXAMPLE CODE 5.4

```
void main ()
{
    int n1, n2, sum;
    scanf ("%d%d", &n1, &n2);
    sum = add (n1, n2);
    printf ("Sum is %d", sum);
}
```


- In the function call *n1* and *n2* are arguments to the function *add()* discussed in **Example 5.2**.
- Variable *sum* is declared to store the result returned from the function *add()*.
- The variables passed as arguments are not altered by the function. The function makes a copy of the variables and all the modifications are made to that copy only.
- In the above example when *n1* and *n2* are passed, the function makes copies of these variables. The variable *x* is the copy of *n1* and the variable *y* is the copy of *n2*.

Important Note:

The values passed to the function are called ***arguments***, whereas variables in the function definition that receive these values are called ***parameters*** of the function.

In the above example, values of variables *n1* and *n2* are arguments to the function *add()*, whereas the variables x and y inside function *add()* are parameters of the function.

Scanned with CamScanner

Important Note:

It is not necessary to pass the variables with same names to the function as the names of the parameters. However, we can also use same names. Here another important point is that even if we use same names, still the variables used in the function are a copy of the original variables. This is illustrated here through following example:

</> EXAMPLE CODE 5.5

```
#include<stdio.h>
void fun(int x, int y)
{
        x = 20;
        y = 10;
        printf("Values of x and y in fun(): %d %d", x, y);
}
void main()
{
        int x = 10, y = 20;
        fun(x, y);
        printf("Values of x and y in main(): %d %d", x, y);
}
```

Output:

```
Values of x and y in fun(): 20 10
Values of x and y in main(): 10 20
```


Scanned with CamScanner

Important Note:

Following points must be kept in mind for the arrangement of functions in a program.

1- If the definition of called function appears before the definition of calling function, then function signature is not required.

2- If the definition of called function appears after the definition of calling function, then function signature of called function must be written before the definition of calling function.

Both the following code structures are valid.

a)
```
int add(int, int);
void main()
{
    printf(”%d ”add(4, 5));
}
int add(int a, int b)
{
    return a + b;
}
```

b)
```
int add(int a, int b)
{
    return a + b;
}
void main()
{
    printf(”%d ”add(4, 5)
}
```

PROGRAMMING TIME 5.1

Problem:

Write a function isPrime() that takes a number as input and returns 1 if the input number is prime, otherwise returns 0. Use this function in main().

Program:

```
#include <stdio.h>
int prime (int n)
{
        for (int i = 2; i < n; i++)
                if(n % i == 0)
                        return 0;
        return 1;
}
```


Scanned with CamScanner

Continued

```
void main()
{
        int x;
        printf ("Please enter a number: ");
        scanf ("%d", &x);
        if(prime(x))
                printf ("%d is a Prime Number",x);
        else
                printf ("%d is not a Prime Number",x);
}
```

PROGRAMMING TIME 5.2

Problem:

Write a function which takes a positive number as input and returns the sum of numbers from 0 to that number.

Program:

```
int digitsSum(int n)
{
        int sum = 0;
        for(int i = 0; i <= n; i++)
        {
                sum = sum + i;
        }
        return sum;
}
void main()
{
        int number;
        printf("Please enter a positive number: ");
        scanf("%d", &number);
```


Scanned with CamScanner

Continued

```
    if(number >= 0)
    {
        int sum = digitsSum(number);
        printf("The sum of numbers upto given number is %d",sum);
    }
    else
        printf("You entered a negative number.");
}
```


Scanned with CamScanner

- A ***function*** is a block of statements that performs a particular task.
- The functions which are available in C Standard Library are called ***built-in functions***.
- The functions which are defined by a programmer are called ***user-defined functions***.
- Some advantages of using functions are: reusability of code, separation of tasks, reduction in the complexity of problem, and readability of code.
- **Function signature** describes the name, inputs and output of the function.
- We can define a function as follows

```
return_type name (Parameters)
{
    Body of the Function
}
```

- The ***return type*** of the function is the data type of the value returned by function.
- The ***name*** of the function should be related to its task.
- ***Parameters*** are variables of different data types, that are used to receive the values passed to the function as input.
- ***Body*** of the function is the set of statements which are executed in the function to fulfil the specified task.
- ***Calling a function*** means to transfer the control to that particular function.
- During the function call, the values passed to the function are called ***arguments***.
- We can call a user-defined function from another user defined function, same as we call other functions in main function.

Scanned with CamScanner

Exercise

Q1 Multiple Choice Questions

1) Functions could be built-in or ____________.

   a) admin defined b) server defined c) user defined d) Both a and c

2) The functions which are available in C Standard Library are called ________.

   a) user-defined b) built-in c) recursive d) repetitive

3) The values passed to a function are called ____________.

   a) bodies b) return types c) arrays d) arguments

4) char cd() { return 'a'}. In this function "char" is ____________.

   a) body b) return type c) array d) arguments

5) The advantages of using functions are ______________

   a) readability b) reusability c) easy debugging d) all

6) If there are three return statements in the function body, ________ of them will be executed.

   a) one b) two c) three d) first and last

7) Readability helps to ____________ the code.

   a) understand b) modify c) debug d) all

8) ______________ means to transfer the control to another function.

   a) calling b) defining c) re-writing d) including

Q2 Define the following.

1) Functions 2) Built-in functions 3) Functions Parameters
4) Reusability 5) Calling a function

Q3 Briefly answer the following questions.

1) What is the difference between arguments and parameters? Give an example.
2) Enlist the parts of a function definition.
3) Is it necessary to use compatible data types in function definition and function call? Justify your answer with an example.
4) Describe the advantages of using functions.
5) What do you know about the *return* keyword?

Scanned with CamScanner

Q4 Identify the errors in the following code segments.

a)
```
void sum (int a, int b)
{
        return a + b;
}
```

b)
```
void message ();
{
        printf ("Hope you are fine :)");
        return 23;
}
```

c)
```
int max (int a; int b)
{
        if (a > b)
                return a;
        return b;
}
```

d)
```
int product (int n1, int n2)
        return n1*n2;
```

e)
```
int totalDigits (int x)
{
        int count = 0;
        for (int i = x; i >= 1, i = i/10)
                count++;
        return count
};
```


Scanned with CamScanner

Q5 Write down output of the following code segments.

a)
```
int xyz (int n)
{
    return n + n;
}
int main()
{
    int p = xyx(5);
    p = xyz(p);
    printf ("%d ", p);
}
```

b)
```
void abc (int a, int b, int c)
{
    int sum = a + b + c;
}
int main()
{
    int x = 4, y = 7, z = 23, sum1 = 0;
    abc (x, y, z);
    printf ("%d %d %d" x, y , z);
}
```

c)
```
int aa (int x)
{
    int p = x / 10;
    x++;
    p = p + (p * x);
    return p;
}
int main()
{
    printf ("We got %d ", aa(aa(23)));
}
```


Scanned with CamScanner

d)
```
float f3(int n1, int n2)
{
        n1 = n1 + n2;
        n2 = n2 - n1;
        return 0;
}
int main()
{
        printf ("%f\n", f3(3, 2));
        printf ("%f\n", f3(10, 6));
}
```


Scanned with CamScanner

Programming Exercises

Exercise 1

Write a function ***int square(int x);*** to calculate the square of an integer *x*.

Exercise 2

Write a function ***int power(int x, int y);*** to calculate and return x^y.

Exercise 3

Write a function to calculate factorial of a number. .

Exercise 4

Write a function which takes values for three angles of a triangle and prints whether the given values make a valid triangle or not. A valid triangle is the one, where the sum of three angles is equal to 180.

Exercise 5

Write a function which takes the amount and the interest percentage and return the interest amount.

Exercise 6

Write a function which takes a number as input and displays its digits with spaces in between.

Exercise 7

Write a function to print the table of a number.

Scanned with CamScanner