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

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.

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.

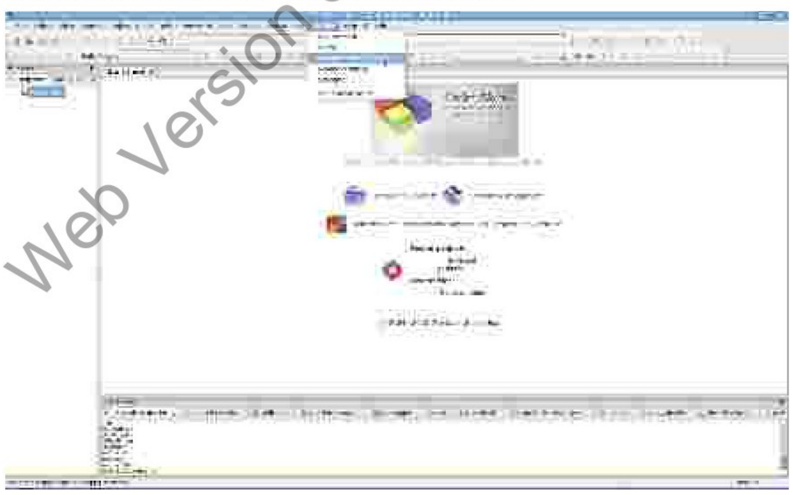


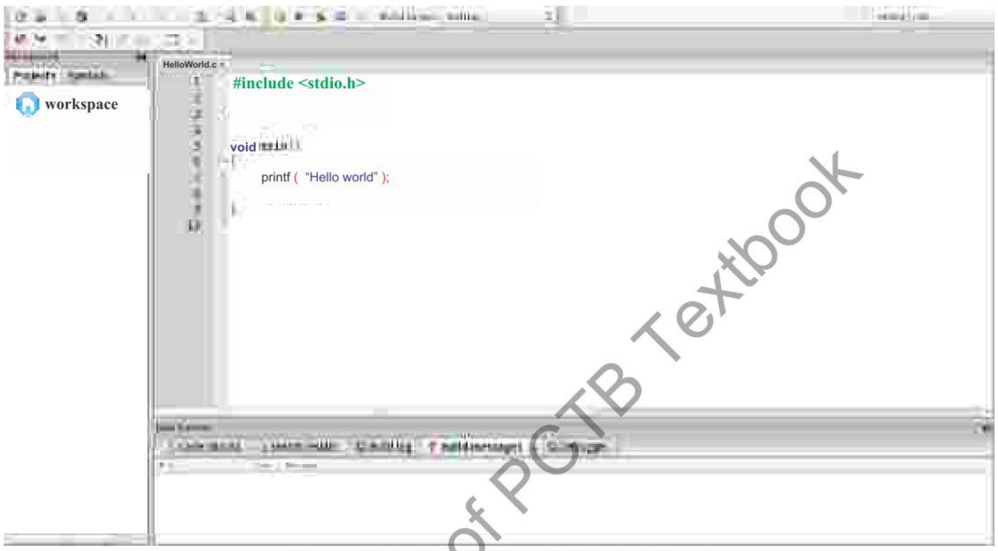
Figure 1.1: Main interface of Code::Blocks

ACTIVITY 1.1

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

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.

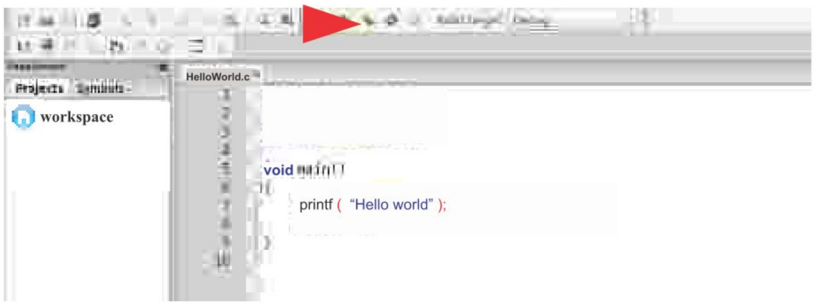


```
#include <stdio.h>

void main()
{
    printf ( "Hello world" );
}
```

Figure 1.2: Text editor in Code::Blocks

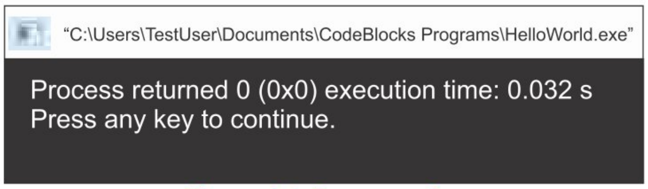
Figure 1.2 shows a basic C language program written in the text editor of IDE Code::Blocks. When executed, this program displays *Hello World!* on computer screen. We have to save our file before it can be executed. We have named our program file as "*HelloWorld.c*". We can click on the *build and run*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



```
void main()
{
    printf ( "Hello world" );
}
```

Figure 1.3: Running program in Code::Blocks

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



```
"C:\Users\TestUser\Documents\CodeBlocks Programs\HelloWorld.exe"

Process returned 0 (0x0) execution time: 0.032 s
Press any key to continue.
```

Figure 1.4: Program Output

ACTIVITY 1.2

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

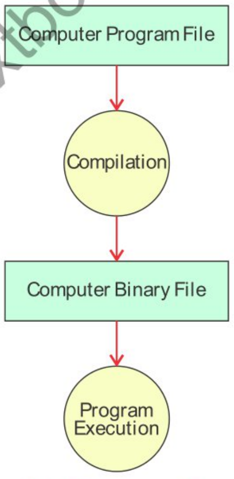


Figure 1.5: Program Execution

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)**.

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*.

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

ACTIVITY 1.3

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.

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.

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,

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

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	'\'

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.

- **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.

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.

ACTIVITY 1.7

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;
```

```
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
}
```

ACTIVITY 1.8

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

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.

- ***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.

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

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)** { }	

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