



## Introduction to C Language

In chapter 9 we have learnt what is a flowchart and algorithm. We also learnt how to develop an algorithm. The fact is that flowchart and algorithm form basic solution to any problem. We then convert this flowchart or algorithm into a program. For writing programs many languages are there. C language is one of them. In this chapter we will be introducing you to C language.

### Need of Programming Language

The first question we might ask is what is the need of programming language? Why not write program in the languages that we already know how to write and read in like English or Gujarati. Well the answer is, a sentence in such languages may not have just one meaning. This does not work well with computers. For computers to work properly each sentence has to be clear and precise.

Programming language when used allows us to write an instruction that has only one meaning. It consists of set of predefined rules. These rules form syntax of that language. Hence learning a programming language is only learning new syntax to represent the instructions we want to give to the computer. It is like learning new grammar when we learn new language.

### Need of Translator

Computers however do not understand the language that we speak neither does it understand the programming language. It only understands 0 and 1. The problem discussed here is similar to the problem that we face at times in our lives. Consider that there are two persons, person X and person Y who don't know each other's language. Person X knows Gujarati while person Y knows Hindi. How would these two persons communicate? The option is third person Z who knows both languages. So now when person X wants to communicate a message to person Y, he communicates the message to person Z in Gujarati first. Person Z then translates it to Hindi and then communicates the message to person Y. Same process is repeated in reverse order when person Y wants to communicate with person X. Here person Z is known as translator and the process of converting one language to another is known as translation.

The problem of computers not understanding our language is solved by using software programs called translators. This translator is known as compiler. The process of translation has been discussed at the end of this chapter.

### Program and Characteristics of Program

A program can be defined as finite set of precise and clear instructions given to a computer for performing a predefined task. The process of writing these step by step instructions using a chosen language is known as programming.

A good program should possess following characteristics :

1. A program must end after finite number of steps.
2. The instructions of program must be precisely defined, i.e. it should not have multiple meaning.
3. All the instructions must be effective, i.e. they should be carried out exactly.

4. A program may take zero or more inputs.
5. A program may produce one or more outputs.

As can be seen these characteristics are similar to characteristics of an algorithm. Example 10.1 gives a sample C program. Figure 10.1 shows the code listing and output of example 10.1 as can be seen in SciTE editor. We will learn how to use SciTE editor at the end of this chapter.

```

10_1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_1.c
/* Example 1: My first C program */
#include <stdio.h>
int main( )
- {
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}
>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
>Exit code: 0
>./10_1
Welcome to the world of C programming using Scite
>Exit code: 0

```

**Figure 10.1 : My first C Program**

This program is used for printing a message "Welcome to the world of C programming using SciTE". Here main() is known as user defined function, while printf() is known as inbuilt or library function. User defined functions are the functions that a user creates as per his or her requirement. The functions already available in C that a user can use in his or her program are known as inbuilt or library functions.

Before we start discussing in detail about various C language components let us have a look at another C program shown in figure 10.2 and its output shown in figure 10.3.

```

10_2.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_2.c
/* Example 2: Program to calculate Circumference of a circle */
#include <stdio.h>
#define PI 3.14
int main( )
- {
    float radius, circumference;
    printf("\nEnter the value of radius: ");
    scanf("%f", &radius);
    circumference = 2 * PI * radius;
    printf("\nCircumference of circle with radius %f is %f \n \n", radius, circumference);
    return 0;
}
/* End of program */

```

**Figure 10.2 : Code listing of Example 10.2**

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of radius: 2.5
Circumference of circle with radius 2.500000 is 15.700000
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

**Figure 10.3 : Output of Example 10.2**

### Explanation

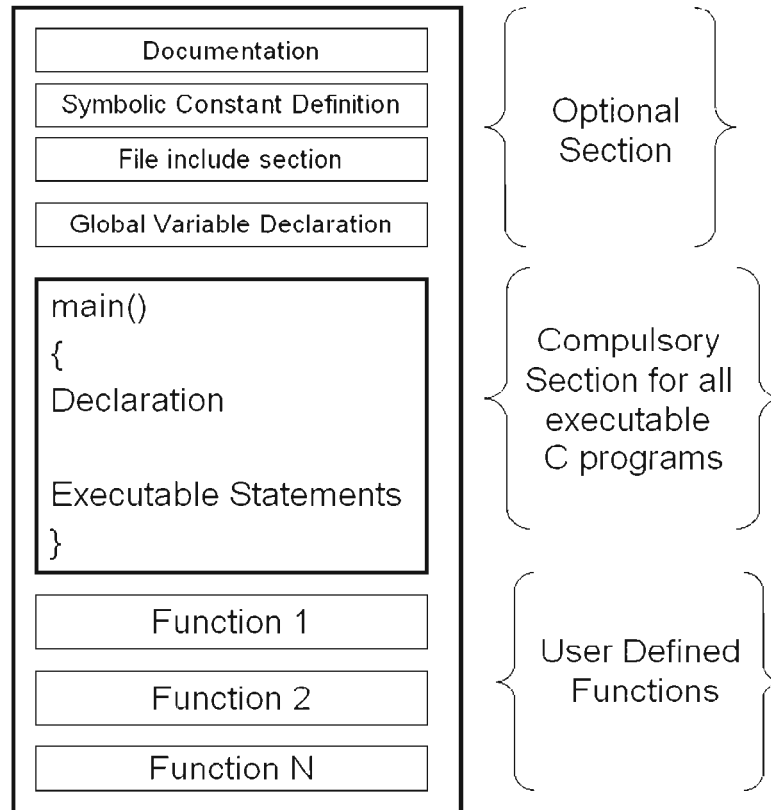
The program given here is used for calculating circumference of a circle for a given radius.

- The first statement is a comment that indicates the purpose of a program.
- The second statement instructs the compiler to include the contents of header file called "stdio.h". This file contains information about the inbuilt functions to be used.
- The third statement defines a symbolic constant PI that has value 3.14. This statement is known as pre processor directive. It instructs the translator to replace all occurrence of PI with 3.14 before executing the program.
- The fourth statement is a user defined function main(), a must for all executable C programs. Here this function returns an integer value hence the statement int main().
- The fifth statement is beginning of function main( ).
- The sixth statement defines two variables radius and circumference. These variables are capable of storing real values. An entity in C that is capable of storing different values is known as variable.
- The seventh statement uses an inbuilt function printf to print a message.
- The eighth statement uses inbuilt function scanf to read a value of "radius" from user.
- The ninth statement is a C expression; it is a formula for calculating the circumference of circle.
- The tenth statement prints a message along with "radius" and "circumference" on the screen.
- The eleventh statement provides normal exit from the function. If we don't write this statement we will get a warning message "Function should return a value". This may not affect the working of the program, but a normal exit is always a better option.
- The twelfth statement indicates the end of the function main( ).

As can be seen from example 10.1 and 10.2 main() function forms an integral part of C programs. Rather we should say that all executable C programs will have one user defined function named as main(). The execution of every C program starts from the first open curly bracket when main() is encountered.

## Structure of C Program

As seen from the previous examples a C program is a set of blocks called functions. A function is made up of one or more statements used for performing a predefined task. A complete C program structure is given in figure 10.4. Let us discuss these sections in detail.



**Figure 10.4 : Structure of complete C program**

### Documentation Section

This section is an optional section. As the name indicates it is used for the purpose of documentation. It is enclosed within `/*` and `*/`. Whenever the text is enclosed between `/*` and `*/`, it is considered as a comment in C. Comments are not processed by C compiler and are left as it is.

This section is normally used to tell about the purpose of program, author, date of creation etc. Comments in C program can be added anywhere. It is always a good practice to use comments within the functions as it improves the readability and understanding of the program.

### Symbolic Constant Definition

As seen in example 10.2 we have used a symbolic constant `PI` in our program. To use such symbolic constant in our program we need to use `#define` as a prefix to it. Here `#define` is known as pre processor directive. It instructs the compiler to replace all occurrences of symbolic constants with the values specified against it. Normally symbolic constants are defined using capital letters. Doing this differentiates them from normal variables.

### File include Section

C provides inbuilt or library functions. Some examples are `pow( )`, `sqrt( )` etc. These functions have a predefined purpose like `pow( )` is used for calculating value of `x` raised to given power,

sqrt( ) is used to find square root of a given number. We can use these functions in our program if needed. To use them we have to include files that hold information about these functions. These files are known as header files in C. The extension of header file is ".h". We use the syntax #include <filename.h> to include header files in our program. Appendix IV gives the list of header files available in C and their uses.

### Global Variable Declaration Section

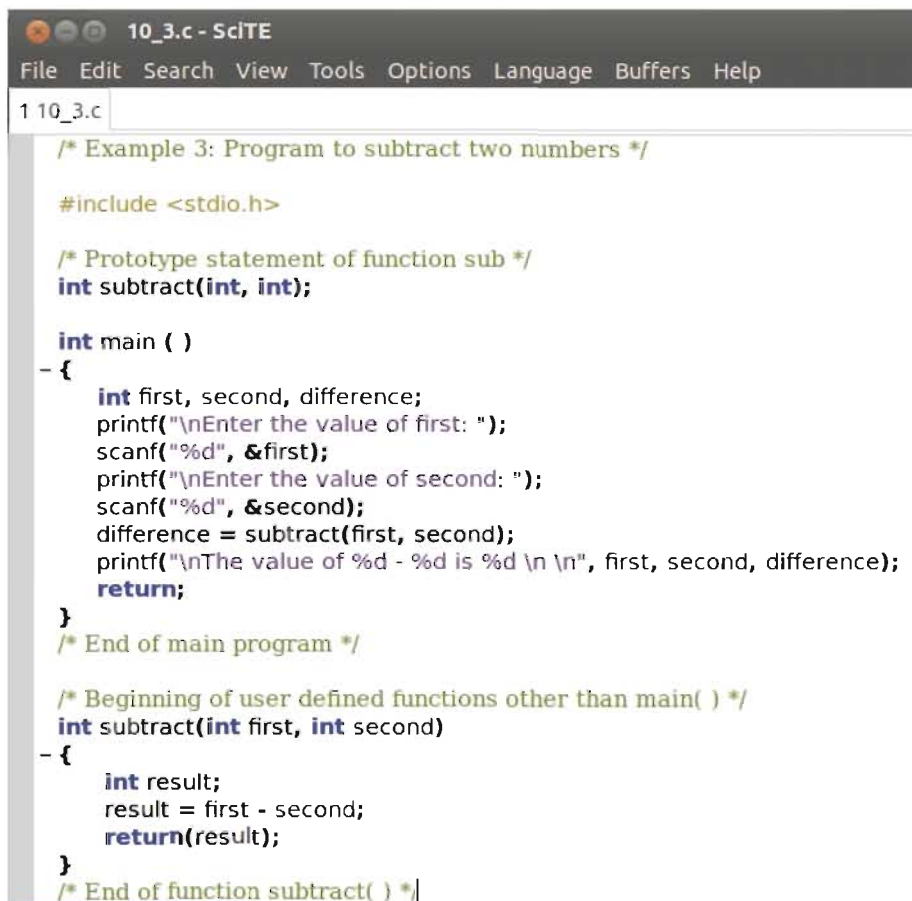
C variables are governed by scope. A scope of C variable is decided by using opening and closing curly braces { }. The variables defined within curly braces are known as local variables. For example the variables "radius" and "circumference" used in example 10.2 are local variables of function main(). These variables cannot be used outside the scope. At times we need to use a variable in all the functions, such a variable is known as global variable. This variable is defined before defining all the functions.

### Main Function

All C programs contain one function with the name main(). This is the function from where the execution of any C program starts. The control is first transferred to this function and from here rest of the operations are carried out.

### User Defined Function

C provides us a facility of breaking a single program into set of small pieces. These pieces are known as functions. In this section we define all the additional functions used in our program. Example 10.3 shown in figure 10.5 shows a program that consists of two user defined functions.



```
10_3.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_3.c
/* Example 3: Program to subtract two numbers */
#include <stdio.h>

/* Prototype statement of function sub */
int subtract(int, int);

int main ( )
- {
    int first, second, difference;
    printf("\nEnter the value of first: ");
    scanf("%d", &first);
    printf("\nEnter the value of second: ");
    scanf("%d", &second);
    difference = subtract(first, second);
    printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
    return;
}
/* End of main program */

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
- {
    int result;
    result = first - second;
    return(result);
}
/* End of function subtract( ) */
```

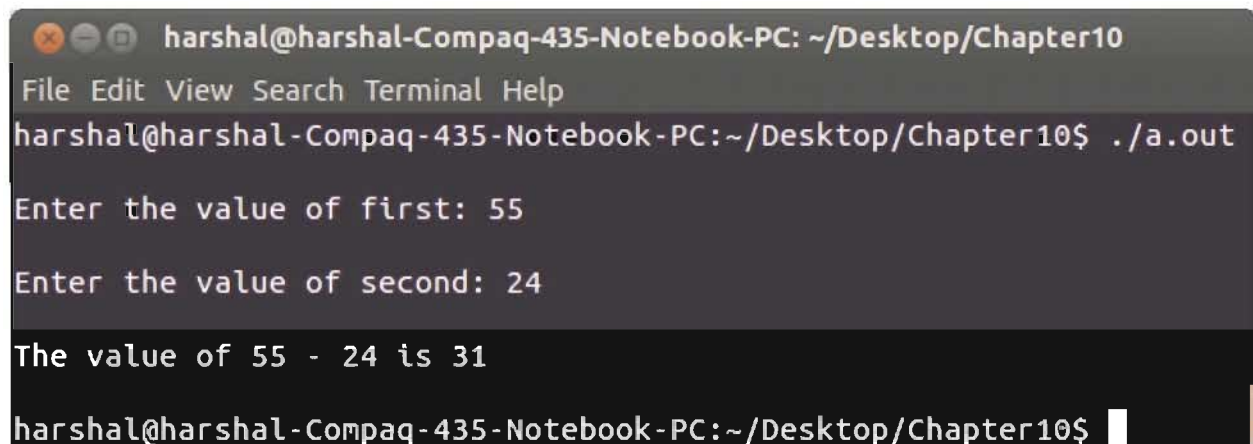
Figure 10.5 : Code listing of Example 10.3



## Explanation

This program is used to calculate subtraction of two given numbers. Although it is a small program we have divided it into two user defined functions. The first function `main()` accepts the values to be subtracted and second function `subtract()` performs the actual subtraction.

Observe that we have written a prototype statement before actually using the function `subtract`. It is needed in C if we write a user defined function after `main()`. The main function declares three integer variables, and asks user to input value of two of them. These values are then passed to function `subtract()` by using statement `difference = subtract(first, second)`. Here `first` and `second` are known as parameters of function `subtract()`. After encountering this statement the control is transferred to function `subtract()`. This function then subtracts the value and stores it in "result". The value of result is then returned to statement in `main()` and assigned to "difference". Finally `main()` function prints the value of subtraction and exits the program. The output of example 10.3 is shown in figure 10.6.



```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of first: 55
Enter the value of second: 24
The value of 55 - 24 is 31
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

**Figure 10.6 : Output of Example 10.3**

The structure of C program shown in figure 10.4 although used by almost all programmers is not a standard. It is possible that some sections may not be there at all as can be seen in example 10.1. It is also possible to interchange certain section with others. For example user defined function section can be interchanged with main program section as can be seen in code listing 10.1. Here we have kept the user defined function before the `main()` function.

```
/* Example 4: Program to subtract two numbers */

#include <stdio.h>

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
{
    int result;
    result = first - second;
return(result);
}
/* End of function subtract( ) */
```

```

/* Beginning of function main() */
int main ( )
{
    int first, second, difference;
    printf("\nEnter the value of first: ");
    scanf("%d", &first);
    printf("\nEnter the value of second: ");
    scanf("%d", &second);
    difference = subtract(first, second);
    printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
    return;
}
/* End of main program */

```

### Code Listing 10.1 : Code of Example 10.4

This program is similar to the one shown in example 10.3 except for two differences. First the function `subtract()` is defined before `main()` and second, we have not used the prototype statement. In example 10.3 on encountering the statement `difference = subtract(first, second);` the control flow goes below `main()`, in case of example 10.4 the control flow will go above `main()`.

## C Character Set

Recall that when we learnt a new language in school, we first had to learn its alphabets. These alphabets form a basic set in constructing words and sentences in that language. C language also has its own character set. The characters here can be divided into four categories.

1. Letter
2. Digits
3. White spaces
4. Special Characters.

Table 10.1 lists the characters that fall under these categories.

Letters	Digits	White Spaces
A to Z a to z	0 to 9	Blank Space
		Form feed
		Horizontal tab
		New line
		Vertical tab

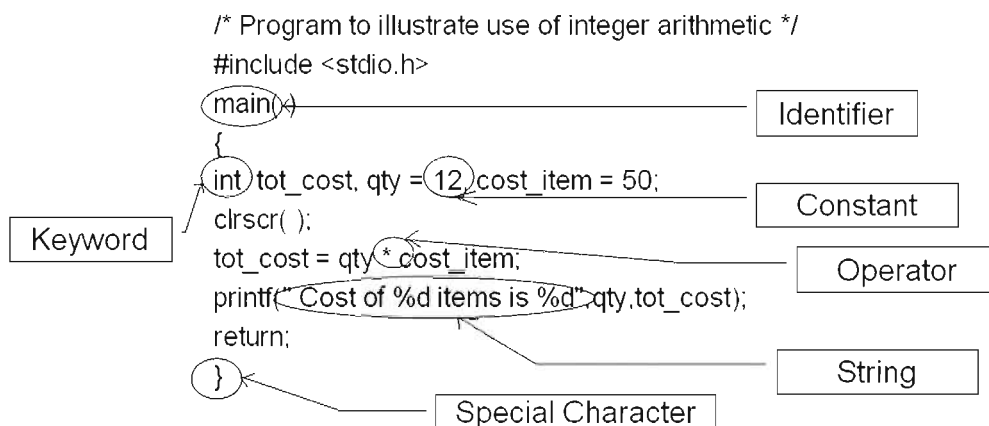
Special Characters			
Operator	Use	Operator	Use
&	Ampersand	>	Greater than
'	Apostrophe	#	Hash sign
*	Asterisk	<	Less than
@	At the rate	-	Minus
\	Backward slash	( )	Parenthesis left / right
{ }	Braces left / right	%	Percent
[ ]	Bracket left / right	.	Period
^	Caret	+	Plus
:	Colon	?	Question mark
,	Comma	"	Quotation mark
\$	Dollar	;	Semi colon
=	Equal to	~	Tilde
!	Exclamation	_	Under score
/	Forward slash		Vertical bar

**Table 10.1 : C Character set**

As can be seen in table 10.1, we have used characters that belong to English language. Hence now we would learn to use these characters as per C language syntax.

### C Words

The character set given in table 10.1 is used to form words in C language. These words are used to construct a C statement. Set of logically sequential C statements thus is identified as a C program. A word in C is known as a token of C. Each character discussed in table 10.1 is itself a token. C basically identifies six types of tokens keyword, identifier, constant, string, operator and special character. Figure 10.7 shows how these tokens are used in C program.



**Figure 10.7 : Use of words in C program**



## Keyword

All of us must have used a dictionary at some point of time. We used it for finding a meaning of a predefined word of that language. C also has such predefined words. To be specific ANSI C standard supports 32 predefined words. These predefined words in C language are known as keyword. Each keyword has a predefined meaning associated to it. Table 10.2 gives the list of the keywords available in ANSI C.

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 10.2 : ANSI C keywords**

The keywords and their use have been discussed in subsequent chapters throughout the course of study.

## Identifier

A word that a user can construct by making use of C character set is known as identifier. It consists of set of letters, digits and some special characters. Some example of identifiers are "difference", "main()", "PI", "float" etc. Here "difference" is name of a variable, "main()" is name of a function, "PI" is name of a symbolic constant and "float" is a predefined word. Before continuing further let us see what is variable.

## Variable

A C program generally accepts input, this input comes in a form of data. To store and manipulate a data we use memory space. The data within this memory space can vary based on the operations performed on it. The data within the memory space is referred by a name known as variable name. The data as it is capable of changing itself is called variable. To define a variable name in C we have to adhere to certain rules. These rules have been mentioned below :

1. Variable name cannot be same as keyword.
2. Variable name consists of letter, digit and under score. No other special character is allowed.
3. The first character of variable name must be a letter or under score.
4. The maximum length of variable name as per ANSI standards is 31 characters. However, it is dependent on compiler.
5. The variable names are case sensitive hence num, nuM, nUM, nUm and Num are considered as different variables.

Table 10.3 gives some examples valid and invalid variables of C language.

<b>Valid</b>
Double → Double is not same as keyword double.
INTEREST → Capital letters are part of C character set.
total_quantity → Underscore and characters are allowed.
mark100 → As first letter is character it is allowed.
_file → First letter can be underscore.
<b>Invalid</b>
char → Use of keyword not allowed.
Total Value → Blank space not allowed.
total&value → Special character not allowed.
10mark → First character has to be character or underscore.

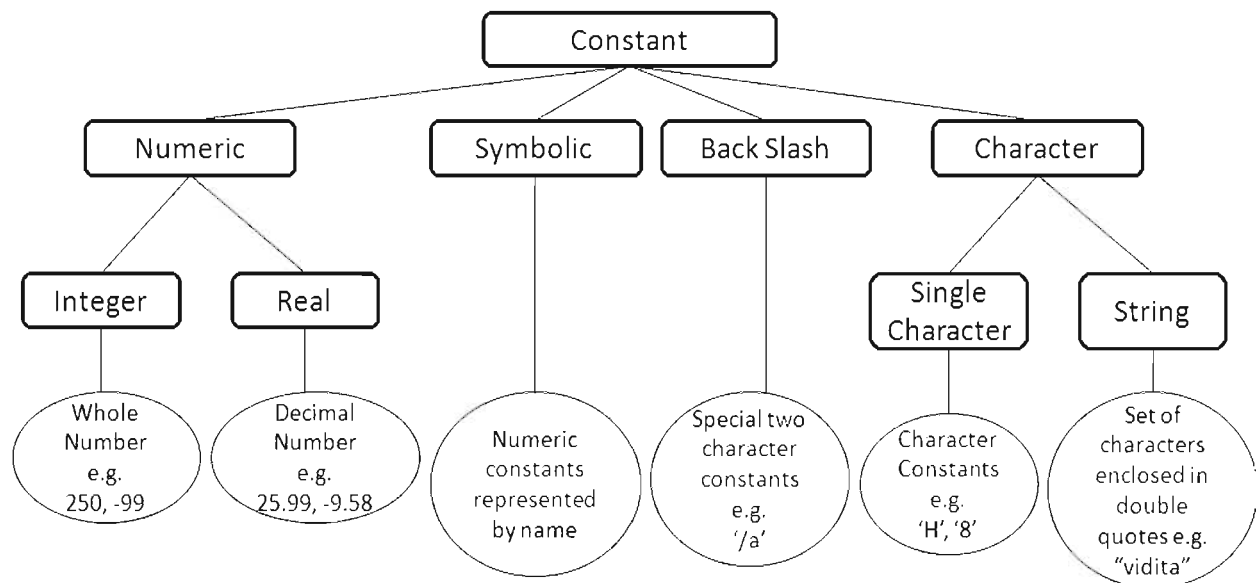
**Table 10.3 : Example of C variable**

### Constant

The entities of C that do not change its value throughout the execution of program are known as constant. C constants can be divided into different categories as shown in figure 10.7. Let us discuss these constants in detail.

### Numeric Constant

The constant that store numeric value is known as numeric constant. The numeric constants are divided in two categories, integer constants and real constants.



**Figure 10.7 : C Constants**

## Integer Constant

Integer constant refers to whole numbers. We have three types of integer constants decimal, hexadecimal and octal. Example of various number systems is given in Appendix I.

The decimal constants use numeric digits from 0 to 9. We can use as prefix unary plus or minus with such numbers.

The hexadecimal constants use numeric digits from 0 to 9 and letters A to F. Here A to F refers to numeric values 10 to 15 respectively. The hexadecimal numbers have base 16. When using this value in C we differentiate it from decimal numbers by using 0x or 0X as prefix.

The octal constants use numeric digits from 0 to 7. The base of this number system is 8. When using this value in C we differentiate it from decimal numbers by using 0 as prefix. Table 10.4 gives a list of some valid and invalid integer constants

<b>Valid</b>
40000 → Valid decimal positive whole number.
-120 → Valid decimal negative whole number.
79999L → Valid decimal positive long whole number.
0xB5 → Valid Hexadecimal number.
0X79 → Valid Hexadecimal number.
045 → Valid Octal number.
<b>Invalid</b>
25,000 → comma is not allowed.
-100.0 → it is real constant.
Rs79999 → Alphabets are not allowed.
0xG → G is not part of hexadecimal numbers.
0X8,9 → Comma is not allowed.
096 → 9 is not part of octal numbers

**Table 10.4 : Integer Constants**

## Real Constant

Real constant refer to decimal numbers that have fractional part. Example of real constant are 10.50, -65.75 etc. Real constant can also be represented by using scientific form. In this form a number is represented using a mantissa and exponent. For example the value 25.75 can be represented as 0.2575 e 2 or 0.2575 E 2. Here 0.2575 is known as mantissa and 2 is known as exponent, also "e" and "E" is equivalent. Table 10.5 gives a list of some valid and invalid real constants.

<b>Valid</b>
250.00 → Valid decimal fractional number.
295 → Valid decimal number. Automatically converted to 295.00
-15.55 → Valid negative decimal number.
-20.5e5 → Valid scientific value.
15E-2 → Valid scientific value.
<b>Invalid</b>
19,800.00 → comma is not allowed.
\$786 → Special character not allowed.
-9.4 e 5.0 → Exponent should be integer.
51E -2 → Blank spaces not allowed.

**Table 10.5 : Real Constants**

### Character Constant

Character constants as the name suggests contain characters. There are two types of character constants in C. Single character constant and String constant.

#### Single Character Constant

A single character constant is represented by using single character enclosed within single quotes. Some example of single character constants are 'H', 'v', '7', '\$' etc. Here each character constant is associated with a numeric value known as ASCII (American Standard Code for Information Interchange) value. Appendix II gives detail of ASCII values associated with different characters.

#### String Constant

String constants are denoted by using sequence of characters enclosed within double quotes. Some examples of string constants are "C Language", "Bye", "V". String constants don't have any ASCII value associated with it. Here string constant "V" is not equal to single character constant 'V'. Here string "V" will be allocated two memory spaces while character 'V' will be allocated only one memory space. This is due to the fact that strings in C are ended by adding '\0', a null character.

#### Back Slash Characters

The single character constant as the name suggests use single character, while string character uses sequence of characters. C also provides a special character constant that uses two characters. These constants are known as back slash characters or escape sequences. They are known as back slash characters as the first character is always a back slash i.e. "\", while they are known as escape sequence characters as the output of these constants is not a character but white spaces. Similar to single character constants, these constants also have an ASCII value associated with them. Table 10.6 lists the back slash constant available in C along with their use and ASCII values.

Back Slash Character	Use	ASCII Value
\0	Inserting null value	0
\a	Inserting audible alert	7
\b	Inserting backspace	8
\t	Inserting horizontal tab	9
\n	Inserting new line	10
\v	Inserting vertical tab	11
\f	Inserting form feed	12
\r	Inserting carriage return	13
\"	Inserting double quotes	34
\'	Inserting single quote	39
\?	Inserting question mark	63
\\	Inserting back slash	92

**Table 10.6 : Back Slash characters**

The escape sequences are mostly used for formatting purpose. For example '\n' is used for inserting new line at the time of input or output. Throughout the book you will be able to see the use of these characters.

### Symbolic Constant

Certain numeric and character constants can be defined using symbolic identifier. Such constants are known as symbolic constant. To define a symbolic constant we use the given syntax:

*#define identifier value*

Here #define is known as a preprocessor directive. Identifier is symbolic name of the constant that we want to define, and value is the constant itself. Some examples of symbolic constants are

```
#define PI 3.14
```

```
# define MAXVALUE 100
```

```
#define f float
```

The first statement here defines a symbolic constant called "PI" that refers to a real value "3.14". Second statement defines a symbolic constant called "MAXVALUE" that refers to an integer value "100" and the last statement defines a symbolic constant called "f" that refers to a keyword "float" in C language.

The preprocessor directive statement instructs the compiler that the occurrence of the symbolic constant used in program should be replaced by the constant value specified in the definition. Example 10.5 shows a program to find surface area of a sphere and uses preprocessor directives. Figure 10.8 shows the code listing of example 10.5 while Figure 10.9 shows the output of the program.



```
10_5.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_5.c
/* Example 5: Program to find surface area of a sphere */

#include <stdio.h>

/* Definition of a symbolic constant */

#define F float
#define P printf
#define S scanf
#define PI 3.14

int main( )
- {
    F radius, sarea;
    P("\nEnter the value of radius: ");
    S("%f", &radius);
    sarea = 4 * PI * radius * radius;
    P("\nSurface Area of sphere with radius %.2f is %.2f\n\n",radius,sarea);
    return 0;
}
/* End of program */
```

Figure 10.8 : Code listing of Example 10.5

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of radius: 2.5

Surface Area of sphere with radius 2.50 is 78.50

harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

Figure 10.9 : Output of Example 10.5

### Explanation

In this program we are defining four symbolic constants F that refers to a keyword float, P that refers to a function printf, S that refers to a keyword scanf and PI that refers to a float value 3.14. The first statement within function main defines two variables of type F. As F represents float this statement actually defines two float variables. The third statement uses P to print a message. Then we read the value of radius using S and calculate the surface area of sphere. Finally we again use P to print a message and value of surface area.

### Points to Remember in C Program

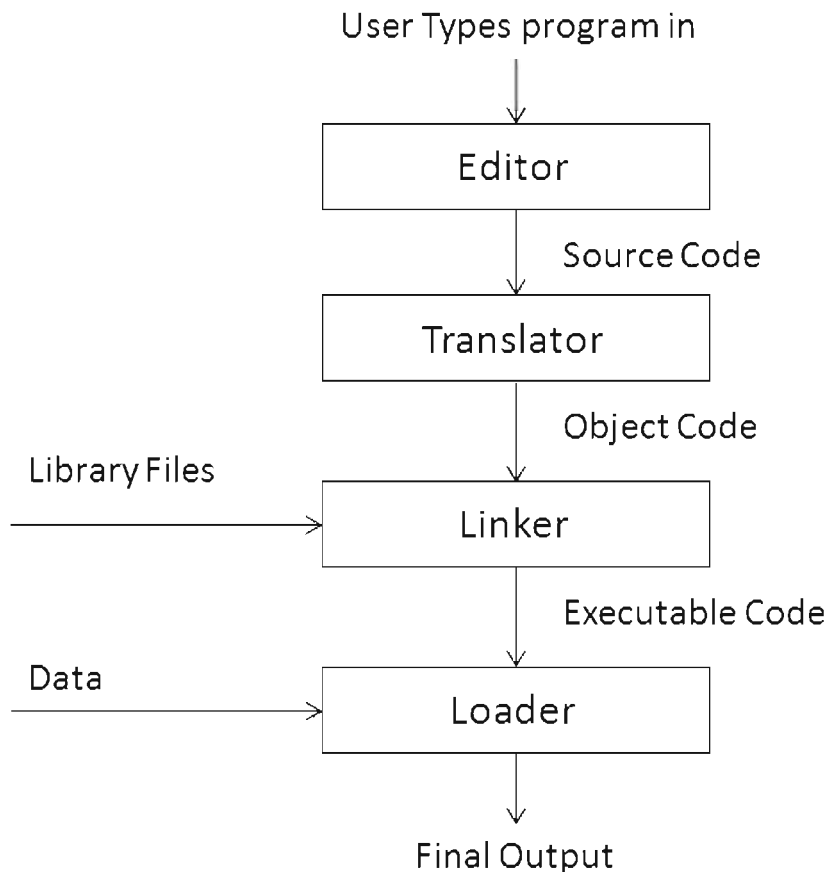
C programming though easy needs a bit of a practice. The points given in this section needs to be remembered by all C programmers.

- Header file should be included before the main() function.
- An executable C program should always contain a main() function.

- Execution of C program starts from the first opening curly brackets after main().
- Two functions with same name are not allowed in C program.
- Normally C programs consist of lower letter alphabets. Identifiers are case sensitive.
- There should be atleast one blank space between two words of a C program.
- Usually C statements end with a semi colon.
- Comment can be inserted wherever a blank space can be inserted.
- Every opening curly bracket should have a corresponding closing curly bracket.

### Execution of C Program

Till now we have seen so many C programs and their output. Let us now learn to execute the C program. Before taking hands on experience let us understand the actual steps that we need to perform. Figure 10.9 shows the entire process.

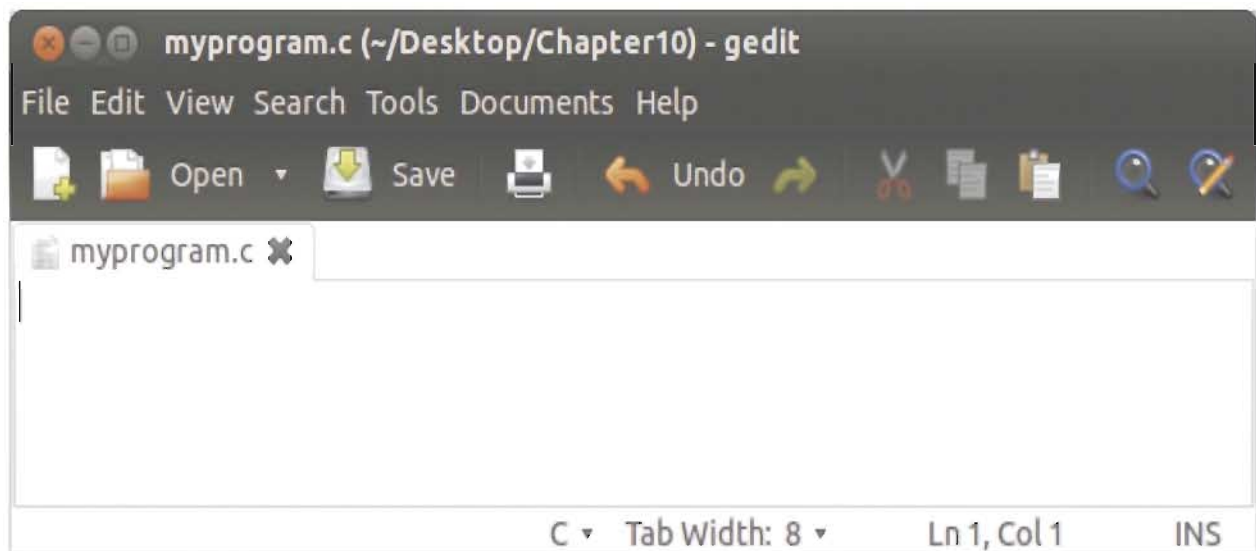


**Figure 10.9 : Steps of executing C program**

As seen in figure 10.9, the first step is to write a program using any text editor, the program that we write using an editor is known as source code or source program. The extension of the source file written in C language is ".c". We then compile the source program using a compiler. A compiler is the translator that converts a source program into machine language program known as object code or object program. Different formats of object code are available. A program called linker is then used to link the object code with the library functions giving executable program or executable code. Finally executable code is loaded on to memory by a program called loader along with the data required to give us the output.

Throughout this book we have discussed a compiler known as gcc compiler. The gcc compiler is provided by Free Software Foundation, it is a Unix/Linux based ANSI C compiler. It is usually operated via the command line but can also be integrated with a text editor like SciTE or an Integrated Development Environment (IDE) like Eclipse. It often comes bundled with almost all Linux installations hence we can simply start using it without any problem.

Let us try to use the compiler, First we need to write a program using a text editor. We can use any good text editor to create a C program. In Linux environment, we have editors such as vi, gedit, emacs and many more. Choose the one you are comfortable with. The only thing that should be kept in mind is the fact that the file created should be saved with an extension c. Make sure that the character c is written in lower case. Let us try to use gedit to create a C program. Open terminal, at the prompt type gedit myprogram.c this will open a blank editor as shown in figure 10.10.



**Figure 10.10 : Blank editor screen**

Now type the contents given in code listing 10.2 in the blank editor screen.

```
/* Example 6: My C program */

#include <stdio.h>
int main()
{
    printf("\nWelcome to the world of C programming using gcc\n");
    return 0;
}
```

**Code Listing 10.2 : Code of Example 10.6**

Once the program is ready your editor will look similar to the figure 10.11, save it and close the editor.



```
myprogram.c (~/Desktop/Chapter10) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
myprogram.c x
/* Example 6: My C program */

#include <stdio.h>
int main()
{
    printf("\nWelcome to the world of C programming using gcc\n");
    return 0;
}
C Tab Width: 8 Ln 8, Col 2 INS
```

**Figure 10.11 : Editor with code listing of Example 10.6**

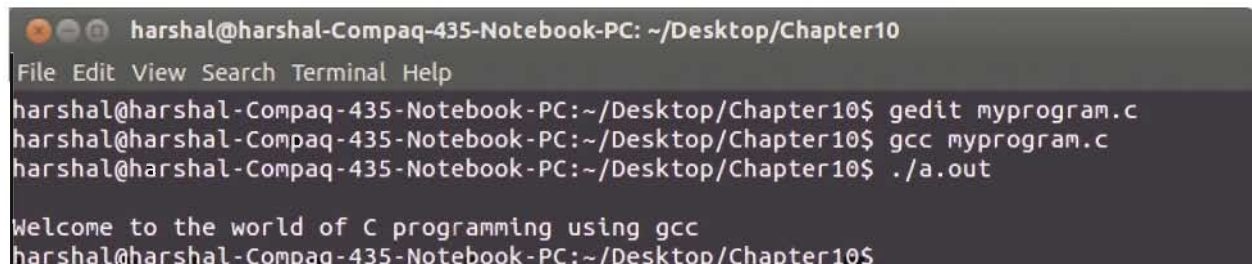
Now we are ready to compile it, go back to the terminal and type the command mentioned:

```
$ gcc myprogram.c
```

If there is no error in the program, a new prompt line will be visible. It means that the compilation is successful and an executable file with the default name a.out is created in the same directory as the source file. In case, we have errors, then appropriate error message would appear on the screen. If error occurs, we need to rectify the program and recompile. To see the output of the program, type the command as mentioned and press Enter key:

```
$ ./a.out
```

This command will display the output of myprogram.c on the screen as shown in figure 10.12.



```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gedit myprogram.c
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gcc myprogram.c
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out

Welcome to the world of C programming using gcc
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

**Figure 10.12 : Output of Example 10.6**

As compilation of all C programs generates a default output file named a.out, it is a good practice to give a name to this output file. Thus, an output file with the given name once created can be reused many times until there is a change in the actual source code. The following command shows how to give a name to output file using gcc command :

```
$ gcc -o myprogram.o myprogram.c
```

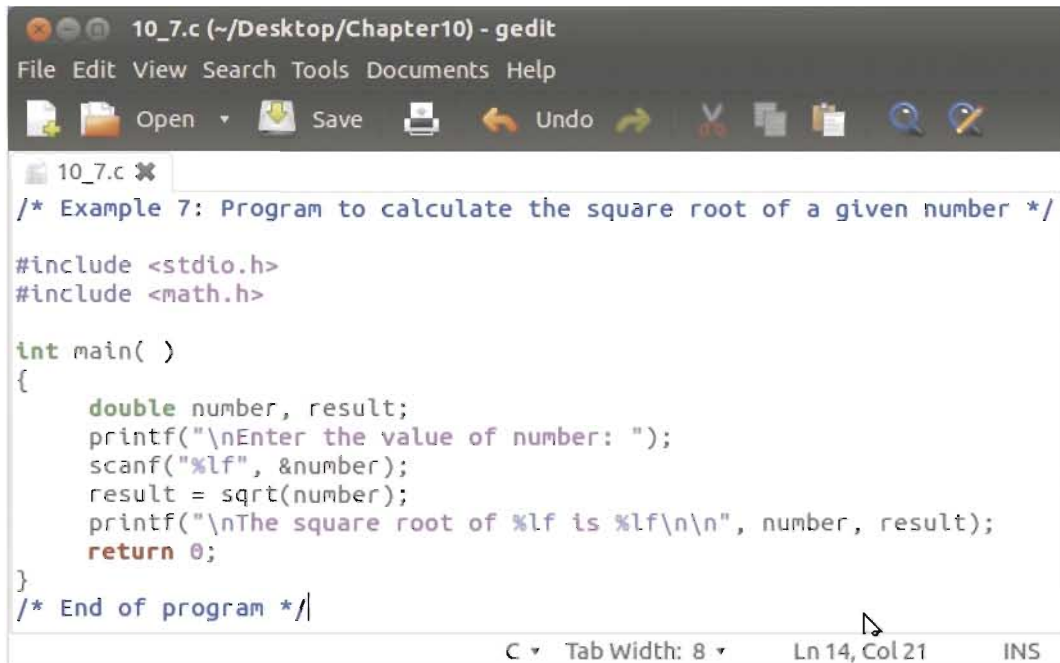
In this command the first parameter myprogram.o represents the output file name, while the second parameter represents the source file name. Observe that we have written .o as an extension for the first parameter, it is just an indication that this file is an output file. The compiler though does not require this extension. We may simple write myprogram instead of myprogram.o. After a successful execution of the preceding command, the compiler will now create a file named myprogram.o instead of a.out. We can now see the output of the program by using the command mentioned herewith :

```
$ ./myprogram.o
```

Many other parameters can also be used along with gcc command. To get the detailed help of gcc command type the help command as mentioned :

```
$ man gcc
```

Let us try to write and compile one more program. Example 10.7 uses an inbuilt function to calculate the square root of a number entered by the user. Figure 10.13 gives the code listing of the example.



```
10_7.c (~/Desktop/Chapter10) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
10_7.c
/* Example 7: Program to calculate the square root of a given number */
#include <stdio.h>
#include <math.h>

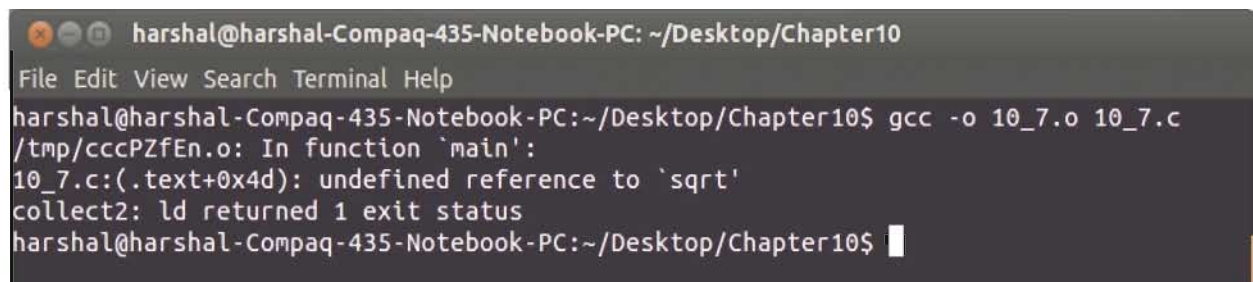
int main( )
{
    double number, result;
    printf("\nEnter the value of number: ");
    scanf("%lf", &number);
    result = sqrt(number);
    printf("\nThe square root of %lf is %lf\n\n", number, result);
    return 0;
}
/* End of program */|
C Tab Width: 8 Ln 14, Col 21 INS
```

**Figure 10.13 : Code listing of Example 10.7**

Let us now try to compile this program using gcc. Open terminal and type the command mentioned below at the prompt.

```
$ gcc -o 10_7.o 10_7.c
```

Figure 10.14 gives the output of this gcc command.



```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gcc -o 10_7.o 10_7.c
/tmp/cccPZfEn.o: In function `main':
10_7.c:(.text+0x4d): undefined reference to `sqrt'
collect2: ld returned 1 exit status
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

**Figure 10.14 : Output of Example 10.7**

Observe that we are getting a new prompt along with a message "undefined reference to 'sqrt'" on screen. As we have used an inbuilt function sqrt() in our program we need to link the math library at the time of compilation. The following command will let us link the math library.

```
$ gcc -o 10_7.o 10_7.c -lm
```

Once we issue this command we will get a new prompt as our program did not have errors. Now at the prompt type ./10\_7.o and observe the output. Figure 10.15 gives us the correct output.



```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gcc -o 10_7.o 10_7.c -lm
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./10_7.o

Enter the value of number: 25

The square root of 25.000000 is 5.000000
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

**Figure 10.15 : Correct Output of Example 10.7**

The method mentioned here is the most basic method for creating and executing a C program on Linux environment. Instead of using any normal text editor we will use an editor named SciTE to create and execute our program in this book. SciTE allows us to compile and run simple programs with one window itself. For programs with user inputs we may compile the program using SciTE, but will use terminal to execute the program. Let us try to see the usage of SciTE editor.

Open the SciTE Text Editor from appropriate location of your machine. We have already seen the look the SciTE editor window in previous examples. Type the contents of example 10.1 shown in figure 10.1 in the blank screen of the SciTE editor. Now save the file with the name 10\_1.c. You can use CTRL + S or use **File** → **Save**. The window will now look similar to the one shown in figure 10.16.

```
10_1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_1.c
/* Example 1: My first C program */
#include <stdio.h>
int main( )
- {
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}
|
```

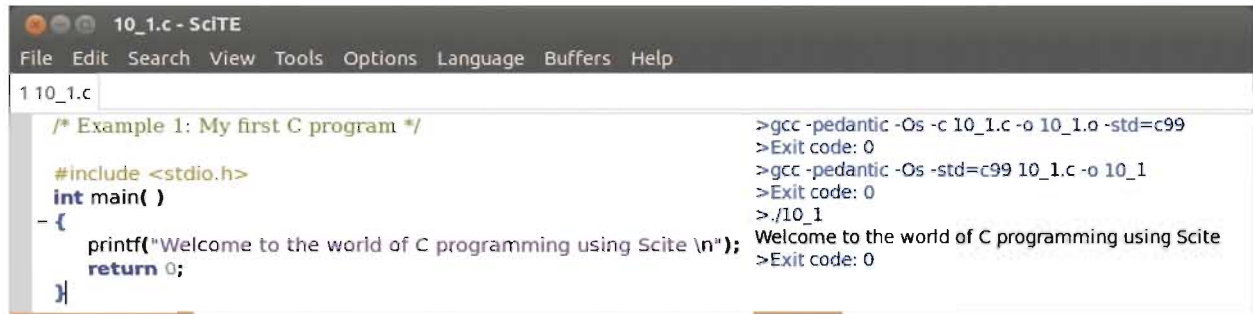
**Figure 10.16 : Example 10.1 as written in SciTE window**

Once the program has been written and saved we need to find syntax errors if present. To find syntax errors we need to compile the program. Press CTRL key along with F7 key (CTRL + F7) or select Tools → Compile from the menu. If no errors are there in the program then we will see a screen as shown in figure 10.17.

```
10_1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_1.c
/* Example 1: My first C program */
#include <stdio.h>
int main( )
- {
    printf("Welte \n");
    return 0;
}
|
>gcc -pedantic -Os -c 10_1.c -o 10_1.o -std=c99
>Exit code: 0
```

**Figure 10.17 : Successful compilation message**

Now we need to execute this program. For this press the F5 key or select Tools → Go. We will now be able to see the output window along with source code window as shown in figure 10.18.



**Figure 10.18 : SciTE editor with two windows**

Observe that we are able to see output of both the activities that we have performed. Observe the first output shown below that we had got when we just compiled the program.

```

> gcc -pedantic -Os 10_1.c -o 10_1.o -std=c99
> Exit code: 0

```

Here the file 10\_1.o although an output file is not executable. Now observe the output shown below that we got when we used Go.

```

>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
>Exit code: 0
>./10_1
Welcome to the world of C programming using scite
>Exit code: 0

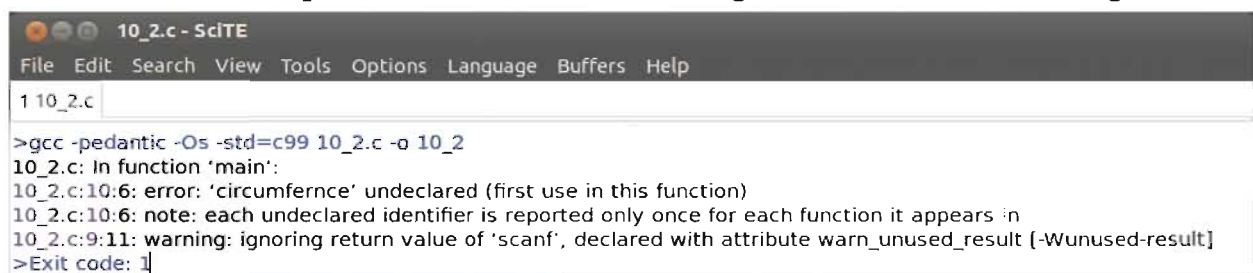
```

Here the file was processed in two phases. First the file was compiled and an executable file with the name 10\_1 was created. This operation is represented by first two lines. In the second phase the output file was executed using command ./10\_1 (as shown in the third line). The last two lines are outcome of this execution.

Note: the text succeeding > symbol refers to actions taken by compiler, while the text shown without a preceding > symbol is output that user wants to display on screen.

It is also possible to enable and disable the output window by pressing F8 key or by selecting **View → Output**. To clear all the previous outputs press Shift + F5 or select **Tools → Clear Output**.

In this example we have taken a program that worked properly. Hence the process of compilation gave us no error. Let us now see what would happen if we have entered a wrong program. Assume that we have entered example 10.2 in the editor with a small change. Instead of writing the statement "circumference = 2 \* PI \* radius;" we did a typing mistake and wrote "circumfernce = 2 \* PI \* radius;". Now if we press CTRL + F7 or F5 we will get the screen as shown in figure 10.19.



**Figure 10.19 : Compilation of program containing error**

As complete image could not be shown properly we have shown only the output window. To see only output select Options → Vertical Split. Observe that we are able to see errors in figure 10.19. Observe the line "10\_2.c:10:6: error: 'circumfernce' undeclared (first use in this function)" in the figure. It informs us that the variable with a name circumfernce is not declared in the program. Rectify the error and repeat the process of compilation. If no errors are found by the compiler, it will try to execute the program if we have pressed F5. Although we will not get any output in our case as we have used scanf() function. The Ubuntu version of SciTE seems to have a bug as of now as it does not wait for user inputs. We can execute the program using the name of the file without any extension from the terminal.

## History of C

Having learnt how to program in C, let us now have some historical overview of C. The origin of C has been dated back to 1972 in Bell laboratories. The man who owns the credit of creating C language is Dennis M. Ritchie. It was derived from *Basic Combined Programming Language* commonly known as BCPL. The aim of developing C was to build robust system software. But it became a pet of programmers in coming years and has been used for developing all kind of software. Hence it has come to be known as general purpose programming language.

Although born in 1972, it was standardized in 1989 by American National Standards Institute (ANSI). From there on it came to be known as ANSI C. Different operating systems and compilers support this standard.

C is a structured language. It allows our program to be broken into small pieces known as functions. These functions once generated are reusable. Set of such functions then becomes a C program. Taking such an approach helps us to solve problem arising in the program easily, as we have to concentrate on only one function rather than an entire program. Usually programs written in C can be ported to different machines having different operating systems and compilers with almost negligible modification. Thus it is considered to be portable language. C is also considered to be a middle level language by some and higher level language by others. In any case it has all the features that a programmer would like to have.

## Summary

In this chapter we learnt about the structure of C program. We looked at different components that can be part of C program. We then learnt about C character as mentioned can be divided into four categories namely letter, digits, white spaces and special characters. We saw how to use these characters and form valid words in C language. Then we looked at the steps of creating and executing a C program. We then learnt how to use gcc compiler to compile and execute our programs. Finally we learnt how to use the SciTE Text Editor to create and execute the C program.

## Instruction for Teachers

The discussion of SciTE Text Editor has been made based on the assumptions that the editor is installed and its short cut would be available on the computer. Before starting to write C programs in your SciTE editor make sure you select Language → C/C++ at least once. This will ensure the highlighting of C keywords when programming.

Also you may need to add the line mentioned below in properties file of SciTE to allow direct execution of C programs by using F5 or Tools → Go.

```
command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)
```

The steps mentioned here will allow you to change the properties file.

1. Find location of `cpp.properties` file in your machine. (In our computer it is `/usr/share/scite/cpp.properties`)
2. Open terminal and type `sudo gedit your_file_path/cpp.properties` and press Enter key.
3. You will be prompted for administrator's password. Key in the correct password and press Enter key.
4. Now `cpp.properties` file will be opened in gedit window. Locate the code given in table 10.7 in the file.

```
ccopts=-pedantic -Os
cc=g++ $(ccopts) -c $(FileNameExt) -o $(FileName).o
ccc=gcc $(ccopts) -c $(FileNameExt) -o $(FileName).o

make.command=make
command.compile.*.c=$(ccc) -std=c99
command.build.*.c=$(make.command)
command.build.*.h=$(make.command)
#command.go.*.c=./a.out
command.go.*.c=$(FileName)
```

**Table 10.7 : Code to be searched in `cpp.properties`**

The make sure that the last two lines look similar on your computers. If they are different, then change it as can be seen in table 10.9. This change makes sure that the executable (output) file will always be saved as the filename that you specify for your source code when using SciTE.

5. Add the following line to this file and save it.  

```
# To make the Go command both compile (if needed) and execute, use this setting:
command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)
```
6. Close the gedit and terminal windows. You are now ready to use the SciTE editor.

The screens given in this chapter are sample screens. These may differ based on the version of Ubuntu available in the school. The functionality of the screens though would remain same.

### EXERCISE

1. List and explain the characteristics of a program.
2. Explain the significance of `main()` function.
3. What is the purpose of file include section in a C program ?
4. What is an identifier ? How is it useful in C program ?
5. What is variable ? State the rules for defining a variable.
6. Differentiate between single character constant and string constant.



7. State whether true or false :

- (a) The extension of C program is ".h".
- (b) Usually C statements end with a comma.
- (c) Amount is a valid variable name.
- (d) #define PI 3.24 includes a file named PI in C program.
- (e) "X" is a valid single character constant.

8. Choose the correct option from the following :

(1) Which of the following is an extension of C program file ?

- (a) c
- (b) h
- (c) s
- (d) t

(2) Which of the following number refers to number of C character categories?

- (a) 0
- (b) 2
- (c) 4
- (d) 8

(3) Which of the following C character categories does the symbol = belong ?

- (a) Letter
- (b) Blank Space
- (c) Special Character
- (d) Digit

(4) Which of the following is a valid keyword of C ?

- (a) ofsize
- (b) sizeof
- (c) forsize
- (d) sizefor

(5) Which of the following is an invalid variable name in C ?

- (a) Register
- (b) RegIster
- (c) registre
- (d) register

(6) Which of the following is an invalid integer constant in C ?

- (a) 0xG
- (b) 0xA
- (c) 0xB
- (d) 0xD

(7) Which of the following is valid real constant in C ?

- (a) -2.0.5e5
- (b) -20.5e5.5
- (c) -20.5e5
- (d) -20.5e.5

(8) Which of the following is valid single character constant in C ?

- (a) 'a'
- (b) '\a'
- (c) "a"
- (d) Both a and b

(9) The preprocessor directive #define is used to define which of the following in C ?

- (a) String constant
- (b) Symbolic constants
- (c) Integer constant
- (d) Single character constant

(10) Which of the following function key is used to directly execute a program ?

- (a) F7
- (b) F9
- (c) F5
- (d) F8



## LABORATORY EXERCISE

1. Write a C program to display your name, school name, the standard you study in and school address in the center of the screen.

```
*****
* Name :           *
* School Name :    *
* Standard :       *
* Address :        *
*                  *
*****
```

2. Write a C program to print first letter of your surname on the screen. For example if the first letter of your surname is P then the output should be.

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

3. Write a C program to print a greeting of your choice on the screen. For example if you want to wish someone on Diwali, print "Wishing you a happy and prosperous Diwali".

