Chapter 2 Constants, Variables and Data Types

1. Character Set:

The characters used to form words, numbers, and expressions depend upon the computer on which the program runs. The characters in C are represented by alphabets in lower case or upper case, digits, or special symbols.

They are classified into the following categories:

i. Alphabets both in lower and upper case (A, B, C……Z and a, b, c……z)

ii. Digits (numbers from 0 to 9)

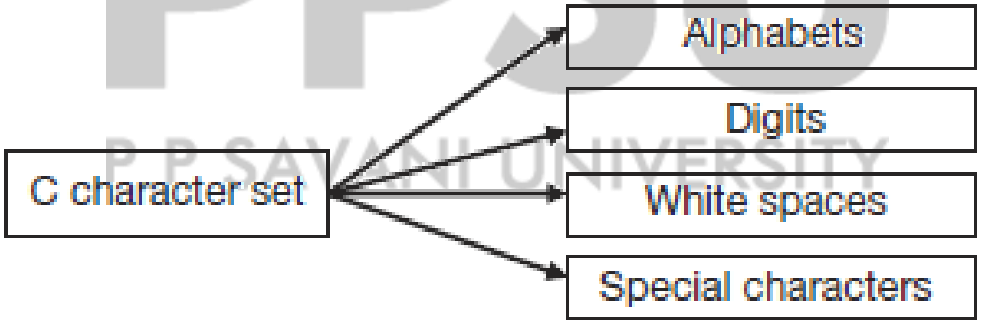
iii. White spaces (blank space, tabs, etc.)

iv. Special characters (^, *, (,), ^, ~, {, }, etc.)

Letters	Digits	Escape Sequences
Upper case letters **A** to **Z**	All decimal digits **0** to **9**	Back space '\b'
Lower case letters **a** to **z**		Horizontal tab '\t'
		Vertical tab '\v'
		New line '\n'
		Form feed '\f'
		Backslash '\\'
		Alert bell '\a'
		Carriage return '\r'
		Question mark '\?'
		Single quote '\''
		Double quote '\"'
		Octal number '\ooo'
		Hexadecimal number '\xhh'

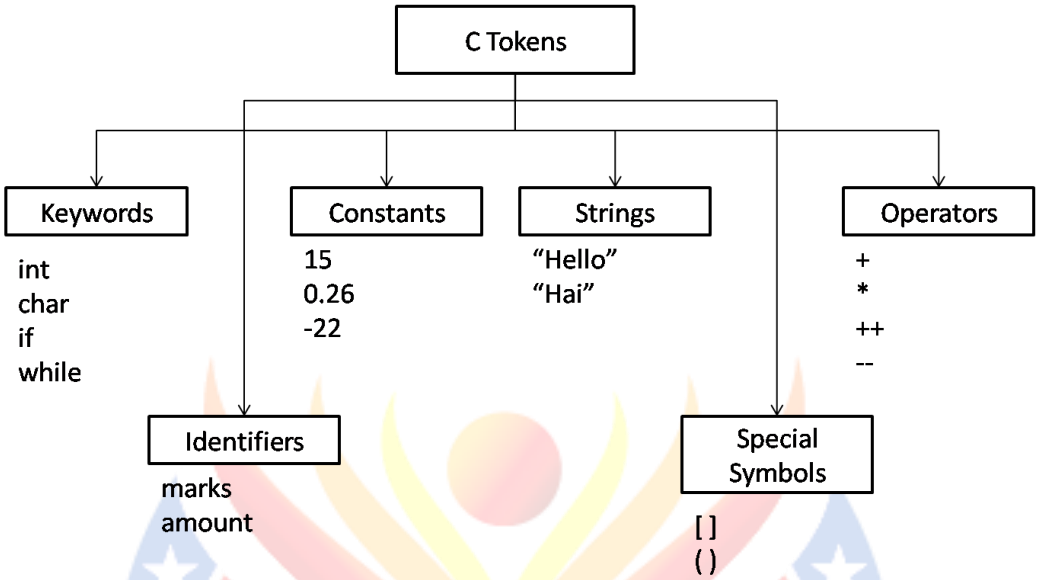
List of Special Characters

Symbol	Name	Symbol	Name
,	Comma	&	Ampersand
.	Period or dot	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
'	Apostrophe	+	Plus
"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	()	Parenthesis left /right
/	Slash	[]	Bracket left/right
\	Back slash	{}	Curly Braces left/right
~	Tilde	%	Percent
_	Underscore	#	Number sign or Hash
$	Dollar	=	Equal to
?	Question Mark	@	At the rate



2. C Tokens:

In a passage of text, individual words and punctuation marks are called token. Similarly, in a C program the smallest individual units are known as C tokens.



3. C keywords:

The C keywords are reserved words by the compiler. All C keywords have been assigned a fixed meaning. They cannot be used as variable names because they have been assigned fixed jobs. However, few C compilers allow the construction of variable names, which exactly coincide with the keywords. It is suggested not to mix up keywords with variable names. For utilizing the keywords in a program, no header file is to be included. The C keywords are listed in Table.

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

Additional keywords for Borland C are as follows:

asm	cdecl	far	huge	interrupt	near	pascal

4. Identifiers:

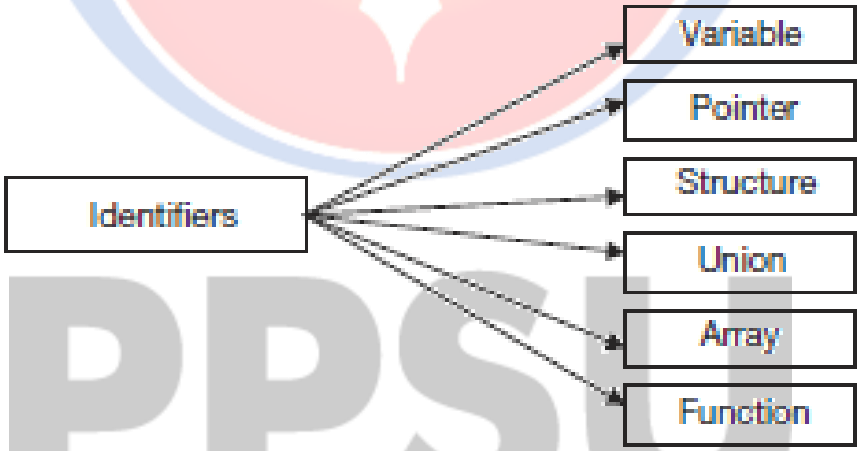
A symbolic name is used to refer to a variable. This is done using identifiers. In other words, identifiers refer to variety of entities, such as structures, unions, enumeration, constants, typedef names, functions, and objects. The C identifier does not allow blank spaces, punctuation, signs, etc. Identifiers are user-defined names, consisting of sequence of letters and digits, with the letter as the first character. They are used to name variables, functions, and arrays. They are generally defined in lower case letters. However, upper case letters are also permitted. The (_) underscore symbol can also be used as and in an identifier. In general, underscore is used as a link between two words in long identifiers. Examples of valid identifiers are length, area, volume, sUM, Average, etc. Examples of Invalid identifiers are length of line, S+um, year's, etc. Figure shows the various identifiers.

Some identifiers in C can be as follows:

(a) #define N 10

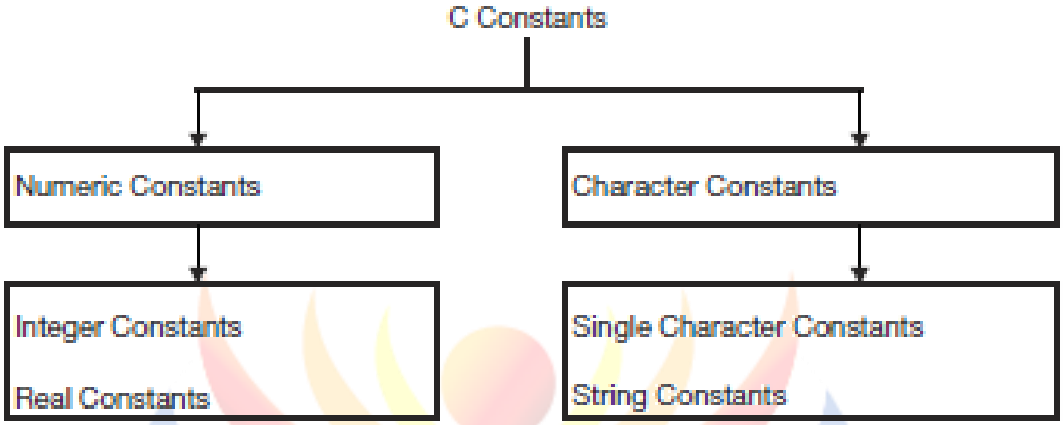
(b) #define a 15

Here, 'N' and 'a' are user-defined identifiers.



5. Constant;

The constants in C are applicable to the values that do not change during the execution of a program. There are several types of constants in C. They are classified into the following groups as given in Figure.



a. Numerical Constants:

- **Integer Constants:** These constants are represented with whole numbers. They require a minimum of 2 bytes and a maximum of 4 bytes of memory.

The following concepts are essential to follow the numerical constants:

(a) Numerical constants are represented with numbers. At least one digit is needed for representing the number.

(b) The decimal point, fractional part, or symbols are not permitted. Neither blank spaces nor commas are permitted.

(c) Integer constant could be either positive or negative or may be zero.

(d) A number without a sign is assumed as positive.

- Real Constants: Real constants are often known as floating point constants. Real constants can be represented in exponential or fractional form. Integer constants are unfit to represent many quantities. Many parameters or quantities are defined not only in integers but also in real numbers. For example, length, height, price, distance, etc. are also measured in real numbers. The following concepts are essential to follow the real numbers:

(a) The decimal point is permitted.

(b) Neither blank spaces nor commas are permitted.

(c) Real numbers could be either positive or negative.

(d) The number without a sign is assumed as positive.

b. Character Constant:

(a) Single character constants: A character constant is a single character. Characters are also represented by a single digit, single special symbol, or white space enclosed within a pair of single quote marks.

Example: 'a', '8', '"', etc.

Character constants have integer values known as ASCII values. For example, the statement

printf ("%c %d",65,'B'); will display the characters, 'A' and 66.

(b) String constants String constants are sequence of characters enclosed within double quote marks. The string may be a combination of all kinds of symbols.

Example: "Hello", "India", "444", "a".

Constant	Meaning
'\n'	Audible alert (bell)
'\b'	Back space
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\v'	Vertical tab
'\'	Single quote
'\"'	Double quote
'\?'	Question mark
'\\'	Backslash
'\0'	null

6. Variables:

During program execution, the value of variables can be changed. Variables can be of different data types. They can be integer, real, or character data types. These data are stored in the memory and at the time of execution different operations are performed on them.

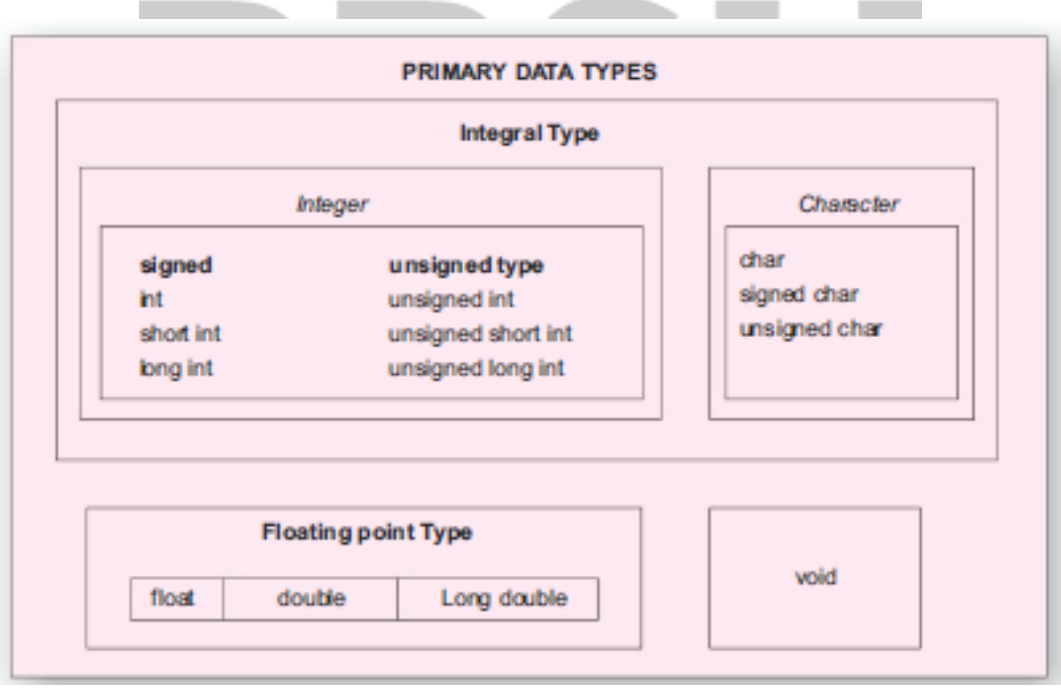
In C, a variable is a data name used for storing a data value. Its value may be changed during program execution. The value of variables keeps on changing during the execution of a program. In other words, a variable can be assigned different values at different times during the execution of a program. A variable name may be declared based on the meaning of the operation. Some meaningful variable names are height, average, sum, etc.

RULES FOR DEFINING VARIABLES

1. They must begin with a character without spaces but underscore is permitted.
2. The length of the variable varies from one compiler to another. Generally, most of the compilers support eight characters excluding extension. However, the ANSI standard recognizes the maximum length of a variable up to 31 characters.
3. The variable should not be a C keyword.
4. The variable names may be a combination of upper and lower characters. For example, suM and sum are
not the same variables.
5. The variable name should not start with a digit.

7. Data Types:

All C compilers support a variety of data types. This enables the programmer to select the appropriate data type according to the need of the application. Generally, data is represented using numbers or characters. The numbers may be integers or real. The C compiler supports different data types, which are explained here under.



Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

Data type	*Keyword equivalent*
Character	char
Unsigned character	unsigned char
Signed character	signed char
Signed integer	signed int (or int)
Signed short integer	signed short int (or short int or short)
Signed long integer	signed long int (or long int or long)
Unsigned integer	unsigned int (or unsigned)
Unsigned short integer	unsigned short int (or unsigned short)
Unsigned long integer	unsigned long int (or unsigned long)
Floating point	float
Double-precision floating point	double
Extended double-precision floating point	long double

8. **Constant and Volatile Variables:**

(a) Constant variable: If we want the value of a certain variable to remain the same or remain unchanged during the execution of a program, it can be done only by declaring the variable as a constant. The keyword const is added before the declaration. It tells the compiler that the variable is a constant. Thus, variables that are declared as constant are protected from modification.

Example:

const int m=10;

Where, const is a keyword, m is a variable name, and 10 is a constant value.

The compiler protects the value of 'm' from modification. The user cannot assign any value to m, but using scanf() statement the value can be replaced. The compiler protects the value of 'm' from modification. The user cannot assign any value to m, but using pointer the value can be changed. When the user attempts to modify the value of constant variable, the message 'cannot modify the constant object in function main' will be displayed.

(b) Volatile variable: The volatile variables are those variables that are changed at any time by other external

program or the same program. The syntax is as follows:

volatile int d;

If the value of a variable in the current program is to be maintained constant and desired not to be changed by any other external operation, then the declaration of variable will be as follows:

volatile const d=10;

9. **typedef:**

By using typedef we can create new data type. The statement typedef is to be used while defining the new data type. The syntax is as given under. typedef type dataname; Here, type is the data type and dataname is the user-defined name for that type.

typedef int hours;

Here, hours is another name for int and now we can use hours instead of int in the program as follows.

hours hrs;

10. **Enumerated data type:**

The enum is a keyword and it is one of the user defined data type. It is used for declaring enumerated data type. The programmer can create his/her own data type and define

what values the variables of these data types can hold. This enumeration data type helps in reading the program.

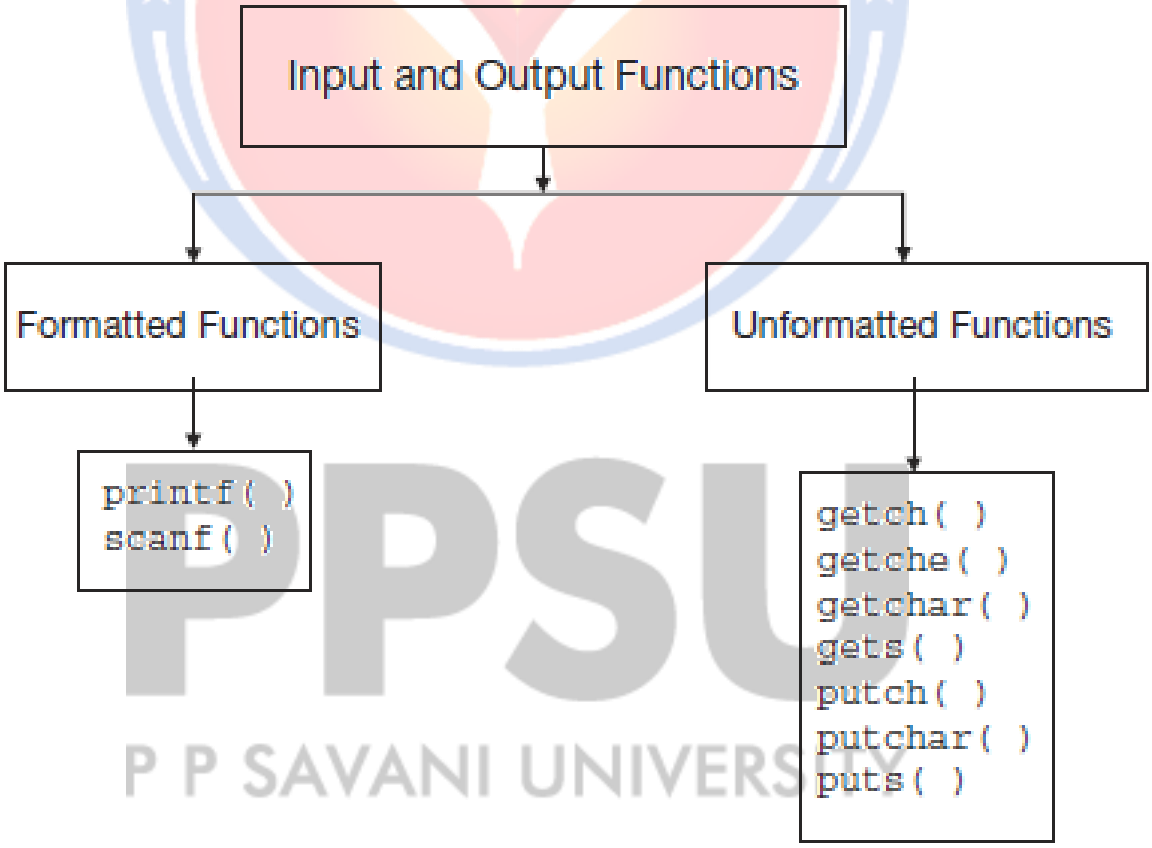
Consider the example of 12 months of a year.

enum month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

This statement creates a user-defined data type. The keyword enum is followed by the tag name month. The enumerators are the identifiers Jan, Feb, Mar, Apr, May and so on. Their values are constant unsigned integers and start from 0. The identifier Jan refers to 0, Feb to 1 and so on. The identifiers are not to be enclosed within quotation marks. Also, note that the integer constants are not permitted as identifier.

11. Input and Output Operation:

There are a number of I/O standard functions in C, based on the data types. The I/O functions are classified into two types: a) formatted functions and b) unformatted functions. Various functions of these categories are listed in Figure.



a. **Formatted functions:** With formatted functions, the input data or output results are formatted as per requirement. They improve the readability of the I/O. For example, with formatted functions one can decide how the result should be displayed on the screen. Formatted functions take care of the appearance of the results. For example, the result can either be displayed in the second line or after leaving some space; if the result

is a real number then a decision can be made even on number of digits before and after the decimal point. All I/O functions are defined in stdio.h header file, which can be initialized at the start of a program. The formatted I/O functions can be used to read and write all types of data values. They require a format specifier to identify the data type. The formatted functions return the values after execution. The return value is equal to the number of variables successfully read/written. Using this value, the user can find out the errors occurring during reading or writing the data. Using this function, the given numeric data can be represented in floating point, integer, and double integer to the available limits of the language. The syntax of the input function for inputting the data is scanf().

The format specifiers and their meanings are given below.

%d: The data is taken as integer.

%c: The data is taken as character.

%s: The data string.

%f: The data is taken as float.

For displaying the result, prinf() formatted function is used.

b. **Unformatted functions:** Formatting is not possible with unformatted functions. The unformatted I/O functions only work with character data types. They do not require format specifiers for identification of data types because they work only with character data types. There is no need to convert the data. In case values of other data types are passed to these functions, they are treated as character data. The unformatted functions also return values, but the returned value of unformatted functions is always the same.