

Chapter

9

ELEMENTS OF C

9.1 OVERVIEW

Computer does not do anything on its own. The most important skill to learn is how intelligently one can program it. It can be used to solve multidimensional problems. The C being a magic tool for writing programs is used to solve variety of problems; a beginner just need practice and more practice of writing programs to master it.

Here we shall introduce the basic elements of C language. These are actually the building blocks of every C program. The proper use of these basic elements helps a lot to write effective C programs.

9.1.1 Identifiers

Identifiers are the names used to represent variables, constants, types, functions, and labels in the program. Identifiers in C can contain any number of characters, but only the first 31 are significant to C compiler. There are two types of identifiers in C: standard identifiers and user-defined identifiers.

9.1.2 Standard Identifiers

Like reserved words, standard identifiers have special meanings in C, but these can be redefined to use in the program for other purposes, however this practice is not recommended. If a standard identifier is redefined, C no longer remains able to use it for its original purpose. Examples of standard identifiers include *printf* and *scanf*, which are names of input/output functions defined in standard input/output library i.e., *stdio.h*.

9.1.3 User-defined Identifiers

In a C program, the programmer may need to access memory locations for storing data and program results. For this purpose memory cells are named that are called *user-defined identifiers*.

C is a *case sensitive* language. This means that C compiler considers uppercase and lowercase letters to be distinct characters. For example, the compiler considers `SQUARE_AREA` and `Square_Area` as two different identifiers referring to different memory locations.

9.2 KEYWORDS

Keywords or *reserved words* are the words, which have predefined meaning in C. There are 32 words defined as keywords in C. These have predefined uses and cannot be used or redefined for any other purpose in a C program. They are used by the compiler as an aid to compile the program. They are always written in lower case. A complete list of ANSI C keywords is given at the end of this chapter.

Keywords or reserved words cannot be redefined in the program.

Variables

Variables are named memory locations (memory cells), which are used to store program's input data and its computational results during program execution.

As the name suggests, the value of a variable may change during the program execution. We are familiar with the concept of *variable* with reference to algebra. The variables are created in memory (RAM); therefore the data is stored in them temporarily. One should not mix the contents of a variable with its address. These are entirely different concepts. The contents of a variable can be thought of as the residents of your neighboring house, while the address of the variable can be thought of as the address of that house (Fig. 9.1).

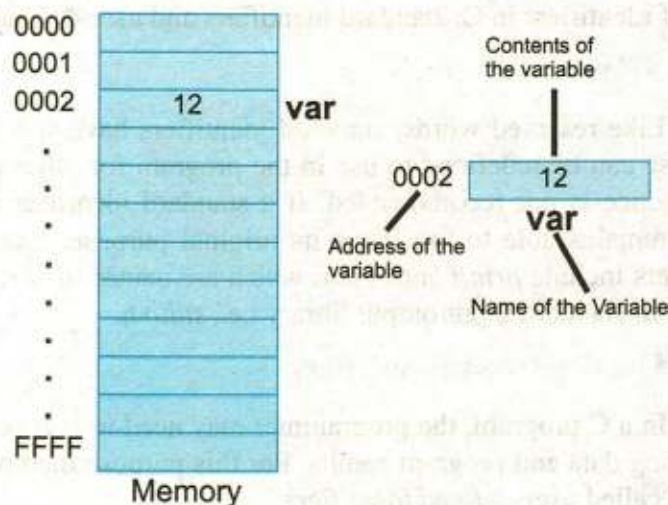


Fig. 9.1: Variable in memory

9.2.1 Declaring a Variable in C

C is a *strongly typed* language i.e. all variables must be declared before being used. The compiler will report an error if an undeclared variable is used

in a program. A variable is declared in C by specifying its type (data type) and name. The syntax takes the form:

```
data type    variable_name;
```

data type refers to the type of the data being stored in the variable. For example

```
int    kgs;  
double length;
```

A list of names of variables separated by commas is specified with the *data type* to declare more variables of the same data type such as:

```
int    marks, total_students, no_of_rooms;  
double kgs, length, volume, height;
```

9.2.2 Declaring vs Defining a Variable

Variable declaration tells the compiler the name of the variable to be used in the program and the type of information stored in it. In a C program, the

```
int volume;  
char ch;
```

variable declarations tell the compiler the name of two variables (volume and ch) used to store an integer and character data respectively.

A variable declaration does not set aside memory location for the data to be stored. It just informs the compiler the name of the variable and the type of data to be stored in it, while the *definition of the variable* that set aside memory location for the variable.

However in C, the variable declaration statement not only declares the variable but also defines it as in case of above two statements. It does not mean that the declaration of a variable can not be separated from its definition in C. C is such a powerful language that provides us all what we need for developing a program. But the discussion is out of scope of this book. We will learn more on this topic in next classes.

9.2.3 Initializing a Variable

Assigning a value to a variable at the time of its declaration is called initializing a variable. In a C program, when a variable is declared, the compiler set aside some memory space for it. This allocated memory space may contain data meaningless to the program (also called garbage). The computations involving this variable may produce unexpected results. To

avoid this situation, all variables should be declared before being used. In C, a variable can also be initialized at the time of its declaration e.g.,

```
int    count;    (variable declaration and definition)
count = 125;     (variable initialization)
char   ch = 'z'; (variable declaration, definition and initialization)
float  weight = 75.8;
```

9.2.4 Rules for Naming Variables

To use variables in C programs, we must know the rules to choose names for them. Here are some important rules for naming a variable in C:

- A variable name can consists of letters, digits, and the underscore character (_)
- The first character of the name must be a letter. The underscore is also a legal first character, but its use is not recommended. Therefore, 9winner, #home, and 2kgms are invalid names in C.
- C is a case sensitive language. Thus, the names *count* and *COUNT* refers to two different variables.
- C keywords can't be used as variable names e.g., we can not use **int**, **void**, *signed*, or *while* as variable names.
- For many compilers, a C variable name can be up to 31 characters long. (It can actually be longer than that, but the first 31 characters of the name are significant) e.g., Turbo C++ restricts the maximum length of a variable name to 31 characters. Hence, the names *problem_solving_techniques_in_C* (31 characters) and *problem_solving_techniques_in_C_language* (40 characters) would appear to be the same to the compiler. The compiler does not differentiate these two names because the first 31 characters of both are the same.
- Blank spaces are not allowed in the name e.g., *problem solving* is an invalid variable name in C.
- A variable can only be declared for one data type.

C programmers commonly use only lowercase letters in variable names, although this isn't required. Using all-uppercase letters is usually reserved for the names of constants.

Variable Names should be Readable

Let's consider a program that calculates loan payments could store the value of the prime interest rate in a variable named *interest_rate*. The variable name helps make its usage clear. We could also create a variable named *x* or even *Ahmed*; it doesn't matter to the C compiler. Many naming conventions are used for variable names. We've seen one style: *interest_rate*. Using an underscore to separate words in a variable name makes it easy to interpret. Another style is to capitalize the first letter of each word. Instead of *interest_rate*, the variable would be named *InterestRate*.

9.4 CONSTANTS

A constant is a quantity whose value can not be changed. Unlike a variable, the value stored in a constant can't be changed during program execution. As discussed in chapter 1, the *define directive* can be used to define constant macros, for example:

```
#define PI 3.142857
```

defines a constant i.e. *PI*, whose value will remain unchanged during the program execution. C has two types of constants; numeric constants and character constants, each with its own specific uses.

9.4.1 Numeric Constants

Numeric constants consist of numbers. It can be further divided into integer and floating point constants. Integer constants represent values that are counted and do not have a fractional part e.g., +56, -678, 8, etc. Floating point constants represent values that are measured e.g., - 4.786, 5.0, 0.45 etc.

9.4.2 Character Constants

A character constant is a single alphabet, a single digit or a single symbol enclosed within apostrophes e.g., 'A' is a valid character constant. e.g., 'A', 'P', '5', '=' etc. The maximum length of a character constant is 1 character.

9.5 DATA TYPE

In C, the *data type* defines a set of values and a set of operations on those values. We know that computer program manipulates various types of data. The data is given to the program as input. The data is processed according to the program instruction and output is returned. In program designing, the data and its type are defined before designing the actual program used to process the data. The type of each data value is identified at the beginning of program design. The values or data used in a program may be of different types.

In a C program, we may need to process different type of data. Therefore, variables should be capable of storing data of different types. This is done by associating certain data types to the variables.

To work with different types of data, C defines some standard data types and also allows us to define our own data types known as *user-define data types*. In C, a standard data type is one that is predefined in the language such as *int*, *double*, *char*, *float* etc. Let's look at some of the important standard data types:

9.5.1 Data Types for Integers (*int*, *short*, *long*)

In C, the data type **int** is used to represent integers – the whole numbers. This means that *int* variables store number that have no fractional parts such as 1128, 1010, 32432 etc. Along with standard type *int*, the compiler also supports two more types of integer i.e. *short int* and *long int*, often abbreviated to just *short* and *long*. In addition to these types, an integer variable can be *signed* or *unsigned*. If not mentioned, all integer variables are considered to be signed. The data types *signed int* and *int* can handle both signed and unsigned whole numbers such as 245, 1010, and -232 etc, where as the data type *unsigned* can not handle negative numbers.

Because of the limited size of the memory cell, all integers can not be represented by *int*, *short* or *long*. When a variable of type *int* is declared, the compiler allocates two bytes of memory to it. Therefore, only the numbers ranging from -2^{15} through $2^{15} - 1$ (i.e. -32768 to 32767) can be represented with the *int* type variables. The variable of type *unsigned int* can handle numbers ranging from 0 through $2^{16}-1$ (i.e. 0 to 65535). The *long* is used to represent larger integers. It occupies four bytes of memory and can hold numbers ranging from -2^{31} through $2^{31}-1$ (i.e. -2147483648 to 2147483647).

Data Type	Bytes	Other Names	Range of Values
Int	2	Signed	-32768 to 32767
unsigned int	2	Unsigned	0 to 65535
Short	2	short int	-32768 to 32767
unsigned short	2	Unsigned short int	0 to 65535
Long	4	Long int	-2147483648 to 2147483647
unsigned long	4	Unsigned long int	0 to 4294967295

Table 9.1 Data types for integers

9.5.2 Data Types for Floating Point Numbers (*float*, *double*, *long double*)

Floating point numbers are the numbers that have a fractional part e.g. 12.35874, 0.54789, and -8.64, 101.003 etc. The floating point numbers are represented in computer in a format that is analogous to scientific notation (floating-point format). The storage area occupied by the number is divided into two sections: the *mantissa* and the *exponent*. Mantissa is the value of the number and the exponent is the power to which it is raised.

For example, in exponential notation the number 245634 would be represented as 2.45634×10^5 , where 2.45634 is the mantissa and 5 is exponent. However in C, the representation of scientific notation is slightly different e.g. the above number will be represented as 2.45634e5. We don't express the exponent as the power of 10, rather the letter e or E is used to

separate exponent from the mantissa. If the number is smaller than 1 (one), then exponent would be negative. For example, the number 0.00524 will be represented in computer as 5.24E-3.

ANSI C specifies three floating point types that differ in their memory requirements: *float*, *double*, and *long double*. These data types provide higher precision than the data types used for integers. The following table shows memory requirement, precision and the range of values that can be represented by data types *float*, *double* and *long double*.

Data Type	Memory Req. (in bytes)	Precision (in digits)	Range of Values
float	4	6	10^{-38} to 10^{38}
Double	8	15	10^{-308} to 10^{308}
long double	10	19	10^{-4932} to 10^{4932}

Table 9.2 data types for floating point numbers

9.5.3 Data Type for Characters – char

The data type *char* is used to represent a letter, number, or punctuation mark (plus a few other symbols). A char type variable occupies 1 byte in memory and can represent individual characters such as 'a', 'x', '5', and '#' etc. (the character '5' is manipulated quite differently than the integer 5 in the computer, so one should not consider both the same. we shall thoroughly discuss the topic in next chapters). In C, a character is expressed as enclosed in apostrophes such as 'a', 'e', 'i', 'o', and 'u' etc.

Like numbers, characters can also be compared, added and subtracted. Let's look at the following program to understand the concept:

```
#include <stdio.h>
void main(void)
{
    char  ch1, ch2, sum;
    ch1 = '2';
    ch2 = '6';
    sum = ch1 + ch2;
    printf("Sum = %d", sum);
}
```

Output:
Sum = 104

In fact, characters are stored in the form of ASCII (American Standard Code for Information Interchange) code. When we add, subtract or compare two

characters, then instead of characters their ASCII codes are manipulated. In above example, because the ASCII codes of characters '2' and '6' are 50 and 54 respectively therefore the sum is 104. In ASCII, the printable characters have codes from 32 (code for a blank or space) to 126 (code for the symbol ~). The other codes represent nonprintable control characters. The complete list of ASCII characters with their codes is given at the end of this chapter.

The *signed* and *unsigned* keywords can be used with *char* like they can with *int*. Signed characters can represent numbers ranging from -128 though 127, while unsigned characters can represent numbers from 0 to 255. English alphabets, numbers and punctuation marks are always represented with positive numbers.

Working with floating point numbers

While working with floating point numbers, we may encounter some problems. For example, manipulation of very large and very small floating point numbers may show unexpected results. When we add a large number and a small number, the larger number may *cancel out* the smaller number; resulting in a *cancellation error* e.g. the result of addition of 1970.0 and 0.0000001243 may compute to 1970.000000 on some computers.

When two very small numbers are manipulated, the result may be too small to be represented accurately, so it will be represented as zero. This phenomenon is called *arithmetic underflow*. Similarly, manipulation of two very large numbers may result to a too large number to be represented. This phenomenon is called *arithmetic overflow*.

9.6 OPERATORS IN C

Operators are symbols, which are used to perform certain operations on data. C is equipped with a rich set of operators. These include arithmetic operators, relational operators, logical operators, bitwise operators, and many others. We shall discuss only the first three of these operators.

9.6.1 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on values (numbers). The C language incorporates the following standard arithmetic operators:

Operation	Symbol	Algebraic Expression	Expressions in C
Addition	+	$A + b$	<code>a + b</code>
Subtraction	-	$A - b$	<code>a - b</code>
Multiplication	*	$A \times b$	<code>a * b</code>
Division	/	A / b	<code>a / b</code>
Modulus	%	$a \bmod b$	<code>a % b</code>

The use of first four operators is straightforward. The last operator is *modulus* (also called *remainder* operator). Contrary to the *division* operator, which returns the quotient, it returns the remainder of an integral division. For example, if *a*, and *b* are two integers having values 8 and 3 respectively, then the express *a % b* will be evaluated to 2, which is the remainder of integral division.

9.6.2 Relational Operators

Relational operators are used to compare two values. These operators always evaluates to *true* or *false*. They produce a *non-zero* value (in most cases 1) if the relationship evaluates to *true* and a 0 if the relationship evaluates to *false*. There are the following six basic relational operators in C:

Suppose *a*, *b* and *c* are three integer variables having values 123, 215 and 123 respectively then:

Operation	Symbol	Expression	Evaluation
Equal to (comparison)	==	<i>a == c</i>	true (non-zero)
Less than	<	<i>b < a</i>	false (zero)
Greater than	>	<i>a > c</i>	false (zero)
Less than or Equal to	<=	<i>a <= b</i>	true (non-zero)
Greater than or Equal to	>=	<i>b <= a</i>	false (zero)
Not Equal to	!=	<i>a != b</i>	true (non-zero)

9.6.3 Logical Operators

Logical operators combine two or more relational expressions to construct compound expressions. The logical operators are && (logical AND), || (logical OR), and ! (logical NOT).

The first logical operator && (logical AND) when combines two conditions, evaluates to *true* if both the conditions are true, otherwise it evaluates to *false*. The second logical operator || (logical OR) when combines two conditions, evaluates to *true* if any one of the conditions evaluates to true, otherwise evaluates to *false*. Similarly, the third logical operator ! (logical NOT) when applied to a condition, reverse the result of the evaluation of the condition, this means that if the condition evaluates to true, the logical NOT operator evaluates to *false* and vice versa. The three logical operators can take the following general forms:

```
exp1 && exp2
exp1 || exp2
!(expression)
```

Example:

Suppose, if the salary of an employee in an organization is less than Rs.10,000 and he/she is married then he/she will be given an additional relief allowance. In a C program, let we have two variables; *salary* (an *int* type variable and *status*) a *char* type variable representing the marital status of the employee. To evaluate this condition, we can use the `&&` operator:

```
salary < 10000 && status == 'M'
```

Suppose, the organization changes his policy and decides to give relief allowance to all those employees whose salary is less than 5000 or who are married. To find out all those employees who fulfill the criteria, we can evaluate the following condition:

```
salary < 5000 || status == 'M'
```

In the meanwhile, the organization feels that their employees are facing problems because of low salary packages and also the organization has to spend a huge amount in the form of pension and retirement bonus. Keeping in view all these problems, the organization introduces revised pay scales in which it offers high salary packages to all those employees whose service period has not exceeded 10 years. In a C program, let we have a variable *service_period* (a float type variable). To find out all those employees whose service period has not exceeded 10 years we can evaluate the following condition:

```
!(service_period > 10)
```

9.6.4 Assignment Operator

In addition to basic C operators (arithmetic, logical, and relational) there are some other important operators, which one should know to write C programs. These includes assignment operator, increment and decrement operators.

The assignment operator is used to store a value or a computational result in a variable. In C, the symbol `=` represents the assignment operator e.g. in the following statement, values of the two variables, *height* and *width*, are multiplied and the result is assigned to the variable **Area**.

```
Area = height * width
```

The value to the right side of the operator is assigned to the variable on the left side of the assignment operator. This statement is also called assignment statement.

Assignment Statement

The assignment statement takes the general form:

variable = *expression*

The *expression* on the right side of the operator is evaluated first and the result is then assigned to the variable on the left side of the operator. The *expression* can be a variable, a constant or arithmetic, relational or logical expression.

Note: Writing variable to the right side and the expression to the left side of assignment operator will cause a syntax error.

9.6.5 Increment and Decrement Operators

The **increment operator** increases the value of its operand by one. It is denoted by the symbol ++ e.g. `count++`, where `count` is a variable. The effect of this expression is equivalent to the following expression:

```
count = count + 1;
```

When ++ precedes its operand, it is called *prefix increment*. When the ++ follows its operand, it is called *postfix increment*. Consider the following statements:

```
j = 10;  
i = ++j;
```

The first statement assigns the value of 10 to the variable *j*. In the second statement, first the value of *j* will be incremented by one (i.e. the value of *j* will be 11) and then the result will be assigned to the variable *i*. Hence, the variable *i* will be assigned the value of eleven (11). Therefore, after execution of the two statements, both the variables will have the same value i.e. eleven (11). Now, consider the following statements:

```
j = 10;  
i = j++;
```

The execution of the first statement will take place as in above case. In the second statement, first the value of *j* (i.e. 10) will be assigned to the variable *i*, and then the value of *j* will be incremented by one. Hence, the variable *i* will be assigned the value of 10. Therefore, after execution of the two statements, the variable *j* will have the value of 11 and the variable *i* will have the value of 10.

C also provides a *decrement operator* denoted by the symbol -- e.g. `count--`. The effect of this expression is equivalent to the following expression:

```
count = count - 1;
```

It can be used as either the *prefix* operator or *postfix* operator in the same way as the increment operator is used. Consider the effect of the following set of statements:

```
j = 10;  
i = --j;
```

After Execution of statements:

- The value of *j* will be 9 and
- The value of *i* will also be 9

```
j = 10;  
i = j--;
```

After Execution of statements:

- The value of *j* will be 9 and
- The value of *i* will be 10

Note: Both *increment* and *decrement* operators are unary operators.

Compound Assignment Operators

The ++ and -- operators respectively increment and decrement the value of their operand by one. There are four other compound assignment operators that can increment or decrement the value of their operand by other than one. These are +=, -=, *= and /= operators.

For example, the statement *j* += 5 increases the value of *j* by 5 and the statement *j* -= 5 decreases the value of *j* by 5. Similarly, we can also use the operators *= and /= in the same way:

```
j = 10;  
j *= 5;
```

After Execution of statements:

- The value of *j* will be 50

```
j = 10;  
j /= 5;
```

After Execution of statements:

- The value of *j* will be 2

The operator *= multiplies the value of *j* by 5 and assign the result to it, whereas the operator /= divide the value of *j* by 5 and assign the result to it. The statement *j* *= 5 is equivalent to *j* = *j* * 5, and the statement *j* /= 5 is equivalent to *j* = *j* / 5.

9.6.6 Operator Precedence

An operator's precedence determines its order of evaluation in an expression. Table 9.3 lists the precedence of all C operators discussed so far, from highest to lowest.

Operator	Precedence
	Highest
! (logical NOT)	
*, / , %	
+, -	
<, <=, >, >=	
==, !=	
&&	
= (assignment operator)	Lowest

Table 9.3 Operator Precedence

The table shows that the logical NOT operator has the highest priority. It is a unary operator – unary operators are those operators that have just one operand. Then comes arithmetic operators, relational operators, logical AND, logical OR and the assignment operators, which are all binary operators – *binary operators* are those operators that have two operands.

9.7 EXPRESSION

An *expression* is the combination of operators and operands. The operand may either be a constant or a variable e.g., $a + b$, $7 + m$ etc.

An expression, in which only arithmetic operators operate on operands, is known as *arithmetic expression*. To solve different mathematical problems, one needs to write arithmetic expressions. Arithmetic expressions involve integers and floating point numbers, which are manipulated with arithmetic operators.

9.7.1 Data Type of an Expression

The data type of expression (in fact, the data type of the result of expression) depends on the types of its operands. For example, consider the type of expression involving int or double type operands is of the form:

exp1 operator exp2

If both the operands are of type int, the result of the expression will be evaluated to an int type value. However, in case of mixed-type expression, one must be careful. A *mixed-type expression* is one in which operands are of different data types e.g. if the type of operand1 is int and the type of operand2 is double then the expression will always be evaluated to a double type value.

Arithmetic Operator	Examples
+	2 + 3 is 5 2.0 + 3.0 is 5.0
-	3 - 2 is 1 3.0 - 2.0 is 1.0
*	2 * 3 is 6 2.0 * 3.0 is 6.0
/	5 / 2 is 2 (integral division, because both the operands are integers) 5.0 / 2.0 is 2.5
%	5 / 2 is 2

Working with Division Operator

The manipulation of division operation is slightly different from other arithmetic operations in C. One should be careful while dividing numbers, as the result of division of two integers may not be an integer. C handles the division intelligently.

When the divisor and the dividend both are integers, the fractional part of the quotient (if exists) is truncated. For example the value of $7.0/2.0$ is 3.5 but the value of $7/2$ is the integral part of the result i.e. 3. Similarly, the value of $198.0/100.0$ is 1.98, but the value of $198/100$ is the integral part only i.e. 1. That's how C performs the division operation. So, to get the accurate result, at least one floating point number should be involved in division operation. Otherwise, integral division will take place and we will get the integral value.

Note: When a type double expression or value is assigned to a type int variable, the fractional part is truncated since it can not be represented in int type variable. For example, let **a** and **b** be two variables of type double and int respectively. Let,

a = 5 * 0.5;

b = 5 * 0.5;

a
2.5

b
2

9.8 Comments

Comments are used to increase the readability of the program. With comments, informative notes are inserted in the program's code, which helps in debugging and modifying the program. In C, there are two ways to comment the code; one can insert single line comments by typing two (forward) slashes at the start of the note such as:

```
// This program calculates the factorial of the given number
```

These are called *single-line* comments. The other method is referred to as *multi-line* comments. Multi-line comments are used to add such informative notes which extend to multiple lines.

Multi-line comments can be inserted to the code (program code) by placing `/*` characters at the beginning of the comments (informative note), this is called opening of the multi-line comments. Each multi-line comment must end with letters `*/`. Omitting ending letters (`*/`) will cause the whole program code beneath the opening letters (`/*`) for comments to be commented.

Note: It is worth mentioning that comments are completely ignored by the compiler while generating object code.

Example:

```
/*  
    This program prints a single line message  
    Author:    Student  
    Date:      14-05-2005  
*/  
void main (void)  
{  
    // the following line of code prints a message  
  
    printf(" This is my first program");  
}
```

Exercise 9c

1. Fill in the blanks:

- (i) The first character of a variable name must be a _____.
- (ii) _____ operates on two operands.
- (iii) Named memory cells which are used to store program's input and output are called _____.
- (iv) The maximum length of a character constant is _____ character.
- (v) The value of a variable of type *int* ranges from _____ to _____.
- (vi) The value of an *unsigned int* variable ranges from 0 to _____.
- (vii) The value of a float variable ranges from _____ to _____.
- (viii) The symbol for logical OR operator is _____.
- (ix) _____ are used to increase the readability of the program.
- (x) Multi-line comments start with _____.

2. Choose the correct option:

- (i) Variables are created in:
 - a) RAM
 - b) ROM
 - c) Hard Disk
 - d) Cache
- (ii) Which of the following is a valid character constant?
 - a) a
 - b) "b"
 - c) '6'
 - d) =
- (iii) Which of the following data type offers the highest precision?
 - a) float
 - b) long int
 - c) long double
 - d) unsigned long int
- (iv) When the result of the computation of two very small numbers is too small to be represented, this phenomenon is called:
 - a) Arithmetic overflows
 - b) Arithmetic underflow
 - c) Truncation
 - d) Round off
- (v) The symbol '=', represents:
 - a) Comparison operator
 - b) Assignment operator
 - c) Equal-to operator
 - d) None of these
- (vi) Which of the following operators has lowest precedence?
 - a) !
 - b) +
 - c) =
 - d) ==
- (vii) Relational operators are used to:
 - a) Establish a relationship among variables
 - b) Compare two values

- c) Construct compound condition
 - d) Perform arithmetic operations
- (viii) C is a strongly typed language, this means that:
- e) Every program must be compiled before execution
 - f) Every variable must be declared before it is being used
 - g) The variable declaration also defines the variable
 - h) Sufficient data types are available to manipulate each type of data
- (ix) The logical not operator, denoted by `!`, is a:
- a) Ternary operator
 - b) Unary operator
 - c) Binary operator
 - d) Bitwise operator
- (x) `a += b` is equivalent to:
- a) `b += a`
 - b) `a += b`
 - c) `a = a + b`
 - d) `b = b + a`
3. Write T for true and F for false statement.
- (i) `printf` and `scanf` are standard identifiers.
 - (ii) In C language, all variables must be declared before being used.
 - (iii) Standard data types are not predefined in C language.
 - (iv) The double data type required 4 bytes in memory.
 - (v) In Scientific notation, the exponent represents the value of the number and mantissa represents the power to which it is raised.
 - (vi) The symbol for modulus operator is `%`.
 - (vii) The symbol `=` is used to compare two values.
 - (viii) Operator precedence determines the order of evaluation of operators in an expression.
 - (ix) For many compilers a C variable name can be up to 31 characters.
 - (x) C program can only use lowercase letters in variable names.
4. What is an identifier? Discuss the two types of identifiers in C.
5. What is a variable? Discuss the difference between declaring and defining a variable.
6. Write down rules for naming variables in C.
7. Differentiate the following:
- (i) Constant and variable
 - (ii) Character constant and Numeric constant
 - (iii) Standard data type and User defined data type
 - (iv) Keyword and Identifier
8. What is a data type? Discuss various C data types to manipulate integers, floating point numbers and characters.
9. How many types of operators are available in C? Describe briefly. Also mention their precedence.

10. What data type would you use to represent the following items: number of children at your school, a letter grade on an exam, and the average marks of your class?
11. Which of the following are valid variable names in C?
 - (i) income
 - (ii) total marks
 - (iii) double
 - (iv) average-score
 - (v) room#
 - (vi) _area
 - (vii) no_of_students
 - (viii) long
 - (ix) Item
 - (x) MAX_SPEED
12. Write a note on the following:
 - (i) Arithmetic Expression
 - (ii) Comments in C
13. Let w, x, y, and z be the name of four type float variables, and let a, b and c be the names of three type int variables. Each of the following statements contains one or more violations of the rules for forming arithmetic expressions. Rewrite these statements so that it is consistent with these rules.
 - (i) $z = 4.0 \ w * y;$
 - (ii) $y = yz;$
 - (iii) $a = 6b4;$
 - (iv) $c = 3(a + b);$
 - (v) $z = 7w + xy;$
14. Assume that you have the following variable declarations:

```
int  a, b, c, d, p;  
float v, w, x, y, z;
```

Evaluate each of the following statements assuming a is 2, z is -1.3, c is 1, d is 3, y is 0.3E+1.

- (i) $v = a * 2.5 / y;$
- (ii) $w = a / y;$
- (iii) $p = a / d;$
- (iv) $x = (a + c) / (z + 0.3);$
- (v) $b = d / a + d \% a;$
- (vi) $y = c / d * a;$