GATE

Be sure with Vidyalankar Since 1960

Pearl Centre, S.B. Marg, Dadar (W), Mumbai – 400 028. Tel. 4232 4232

CS : COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

Programming in C

Index

Topic 1 : C–Fundamentals and Input Output

INTRODUCTION TO C – PROGRAMMING

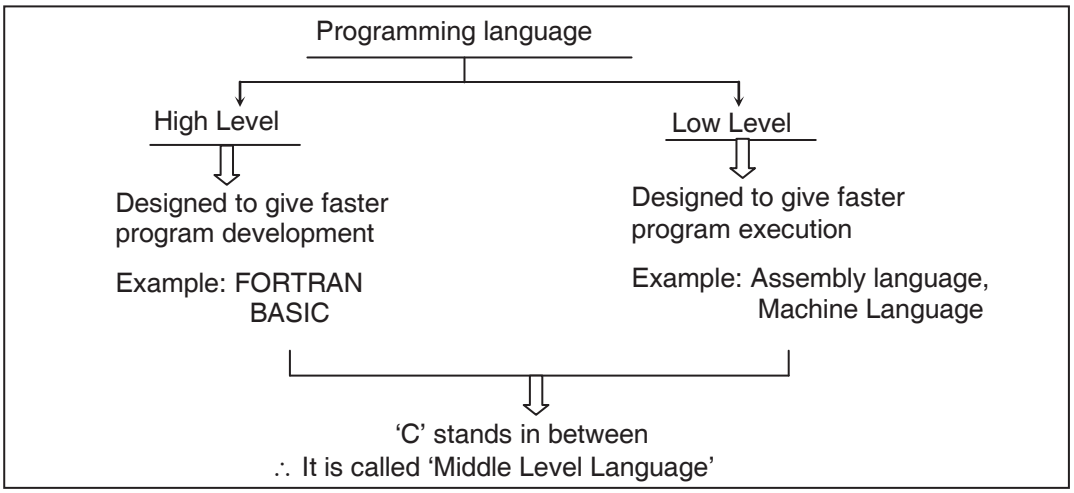
'C' seems a strange name for a programming language. But this strange sounding language is one of the most popular computer languages today. C was an offspring of the 'Basic Combined Programming Language' (BCPL) called B. B Language was modified by Dennis Ritchie and was implemented at Bell Laboratories in 1972. The new language was named C. Since it was developed along with the UNIX operating system, it is strongly associated with Unix.

Importance of C

1. It is robust language whose rich set of built-in functions and operators can be used to write any complex programs.
2. The 'C' compiler combines the capabilities of an Assembly language with the features of the high–level language and therefore well suited for writing both system software and business packages.
3. Programs written in C are efficient and fast due to variety of data types and powerful operators.
4. It is a highly portable means 'C' program written for one machine can easily be run on another machine with little or no–modification.
5. 'C' language can extend itself i.e. we can add our own function to its library.

Where 'C' stands

Programming languages are divided into high level and low level language



C – FUNDAMENTALS

CHARACTER SET

- The characters that can be used to form words, numbers and expressions depend upon the computer on which the program is run.
- The characters in 'C' are grouped into :

1. C supports A ..Z and a…z letters
2. C supports 0…9 digits
3. C supports 29 special characters
4. C supports white space like blank space or tab etc.
    - Compiler ignores white spaces unless they are part of string
    - It is also used to separate the words

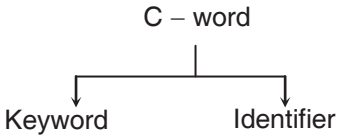
Special Characters

, comma
. period
; semicolon
: colon
? question mark
′ apostrophe
″ quotation mark
! exclamation mark
| vertical bar
/ slash
\ back slash
~ tilde
_ underscore
$ dollar sign
% per cent sign
# number sign
& ampersand
^ caret
* asterisk
– minus sign
+ plus sign
< opening angle bracket (or less than sign)
> closing angle bracket (or greater than sign)
(left parenthesis
) right parenthesis
[left bracket
] right bracket
{ left brace
} right brace

White Spaces

Blank space
Horizontal tab
Carriage return
New line
Form feed

KEYWORDS AND IDENTIFIERS



- Every C word is classified as either a keyword or an identifier
- All keywords have fixed meaning and their meanings cannot be changed.
- Keywords serves as basic building blocks for programming statement.
  Example : void, for, if etc.

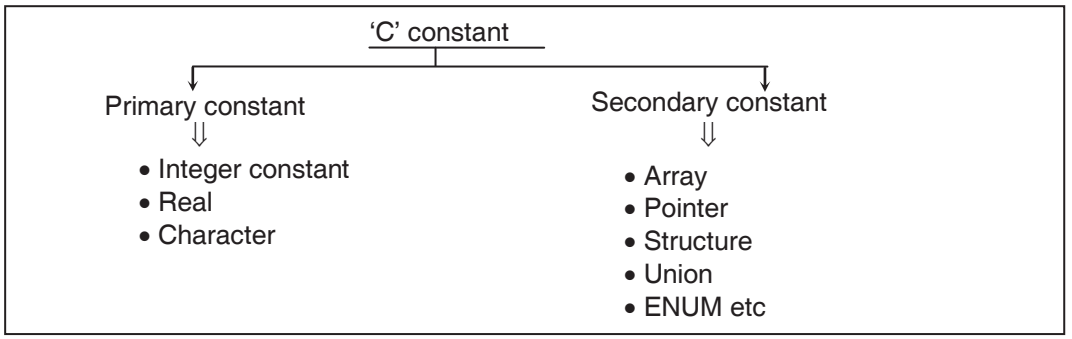
IDENTIFIER

- It refers to the names of variables, functions and arrays
- These are user defined and consists of letter / digit, with a letter as first character. It accepts both lowercase and uppercase letters.
  Example : max10, m1023 etc.

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	go to	sizeof	volatile
do	if	static	while

CONSTANTS

It refers to the fixed values that do not change during the execution of program



Integer constants

It refers to the sequence of digits. There are three types of integers, namely decimal, octal and hexa–decimal. Rules for constructing integer constants are:

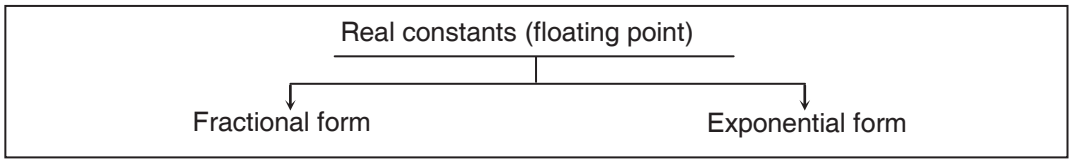
(a) At least one digit must be present
(b) No decimal point
(c) It is either +ve or –ve
(d) If no sign mention, then it is considered as positive.
(e) No comma or blanks are allowed within constant.
(f) Allowable range for integer constant is –32768 to +32767

Examples :

Valid	Invalid
+ 123	IS 7500
0	$ 15
–78	20,000
12	
654321	

Real constant

It is also called as floating point constant. Real numbers are represented into two forms.



(i) Fractional form

Rules for constructing Real constants in fractional form

(a) It must have at least one digit
(b) It must have decimal point
(c) It is either positive or negative
(d) By default it is positive.
(e) No comma or blank space is allowed.

Examples :

Valid	Invalid
426.0	326
–36.78	$ 4.76

(ii) Exponential form

Real number is represented as follows :

Mantissa e exponent

Mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional plus or minus sign.

Example : 3.2e –5, 4.2e + 4

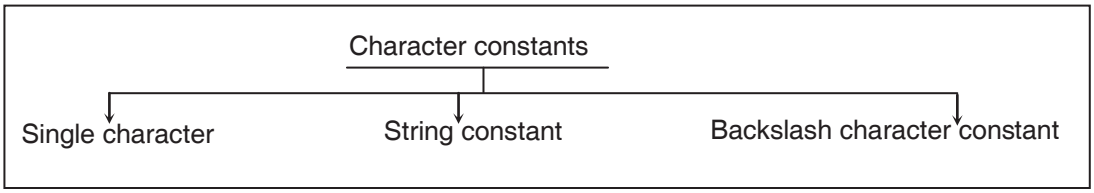
Rules for constructing Real constants in Exponential form

(a) Mantissa and Exponent should be separated by e.
(b) Mantissa is either positive or negative.
(c) By default positive.
(d) Exponent must have one digit either positive or negative. By default positive.
(e) Exponent can be written in either lowercase or uppercase.
(f) Embedded white space is not allowed.
(g) Exponent must be an integer.
(h) Range : –3.4e38 to 3.4e38

Example :

+3.2e – 5
4.1e8
–0.2e + 3
–3.2e – 5

Character constants



(i) Single character :
- Single character enclosed within a pair of single quote marks.
- Example : '5', 'x' etc.

(ii) String character
- String constants is a sequence of characters enclosed in double quotes.
- Characters may be letters, numbers, special characters and blank spaces.
- Example : "Hello!", "1987" etc.

(iii) Backslash character constant.

'C' supports some special backslash character constants that are used in output functions

Constant	Meaning	Constant	Meaning
'\a'	audible alert (bell)	'\v'	vertical tab
'\b'	back space	'\' '	single quote
'\f'	form feed	'\"'	double quote
'\n'	new line	'\?'	question mark
'\r'	carriage return	'\\'	backslash
'\t'	horizontal tab	'\0'	null

These character combinations are known as **"escape sequences"**.

VARIABLES

- A variable is a data name that may be used to store a data value.
- Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.

Rules for constructing variable names:

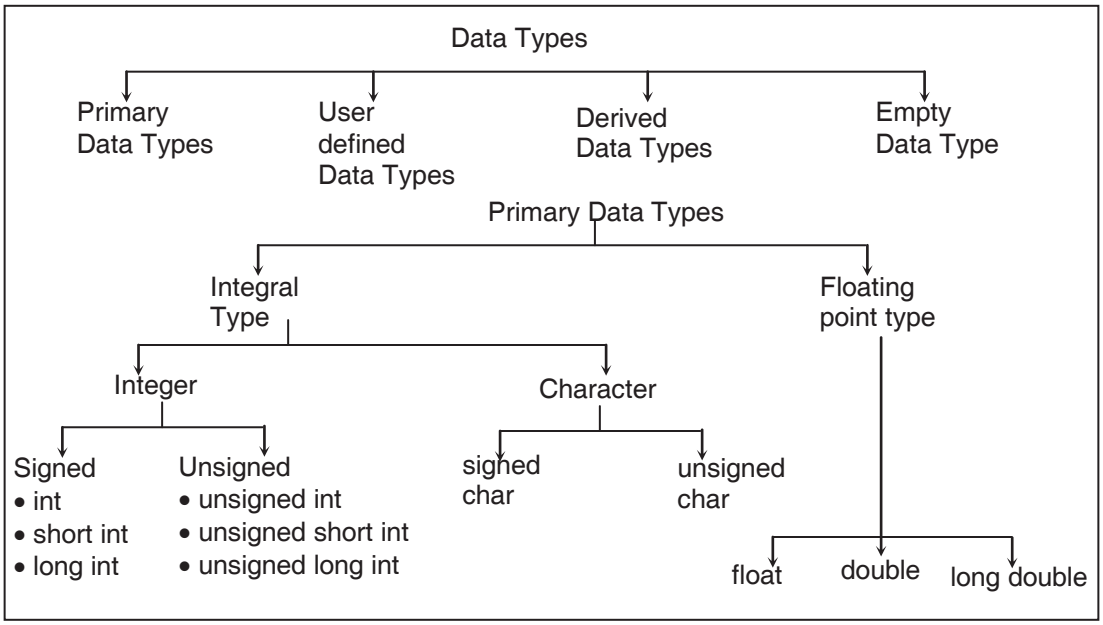
1. They must begin with a letter. Some systems permit underscore as the first character.
2. ANSI standard recognizes a length of 31 characters. However, the length should not be normally more than 8 characters, since only the first eight characters are treated as significant by many compilers.
3. Uppercase and lowercase are significant. That is, the variable Total is not the same as *total* or TOTAL.
4. The variable name should not be a keyword.
5. White space is not allowed.

Examples : Value, T_raise, XI etc.

DATA TYPES

'C' language is rich in its 'data types'. Storage representations and machine instructions to handle constants differ from machine to machine.

The variety of data types are available to allow the programmer to select the type appropriate to the needs of application as well as the machine. ANSI C supports four classes of data types.



Type	Size(bits)	Range
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147, 483, 647
unsigned long int	32	0 to4,294,967,295
float	32	3.4E -38 to 3.4E + 38
double	64	1.7E–308 to 1.7E+308
long double	80	3.4E–4932 to 1.1E+4932

Declaration of Variables

After designing suitable variable names, we must declare them to the compiler. A declaration associates a group of variables with a specific data type. Declaration does two things:

(1) It tells the compiler what the variable name is.
(2) It specifies what type of data the variable will hold.

Syntax :

```
data type v1, v2, …..vn;
```

where v1, v2, …….vn are the names of variables. Variables are separated by commas.

Declaration sentences must end with a semicolon.

Example :

```
int count ;
int number, total ;
double ratio ;
```

Assigning values to variables

Initial values can be assigned to variables within a type declaration. To do so, the declaration must consist of a data type, followed by a variable name, an equal sign (=) and a constant of the appropriate type. A semicolon must appear at the end, as usual.

Example :

```
initial_value = 0 ;
Final_value = 100 ;
```

The process of giving initial values to variables is called initialization. C permits the initialization of more than one variables in one statement using multiple assignment operators.

Example :

```
p = q = s = 0 ;
x = y = z = MAX ;
```

External and static variables are initialized to zero default.

SYMBOLIC CONSTANTS

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent a numeric constant, a character constant or a string constant. Thus, a symbolic constant allows a name to appear in place of a numeric constant, a character constant or a string. When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence. Symbolic constants are usually defined at the beginning of a program.

Syntax :

```
# define   name   text
```

where name represents a symbolic name, typically written in uppercase letters and text represents the sequence of characters associated with the symbolic name.

Example :

```
# define PI  3.1415
# define MAX 200
```

Symbolic names are sometimes called constant identifiers
The following rules apply to a # define statement which define a symbolic constant.

1. Symbolic names have the same form as variable names
2. No blank space between the pound sign '#' and the word define is permitted.
3. '#' must be the first character in the line.
4. A blank space is required between #define and symbolic name and between the symbolic name and the constant.
5. # define statements must not end with a semicolon.
6. Symbolic names are NOT declared for data types. Its data type depends on the type of constant.
7. #define statements may appear anywhere in the program but before it is referenced in the program.

BASIC STRUCTURE OF C – PROGRAM

Documentation Section
Link Section
Definition Section
Global Declaration
Main ( ) function section **{** **Declaration part** **Executable part** **}**
Sub program section **Function 1** **Function 2** **:** **:** **Function n**

User defined functions

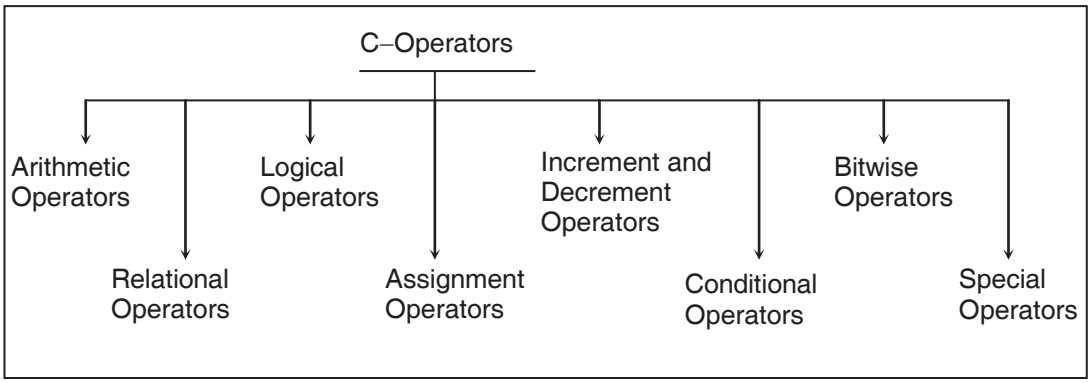
1. Every C program requires a main ( ) function (use of more than one main ( ) is illegal). The place main is where the program execution begins.
2. The execution of a function begins at the opening brace of the function and ends at the corresponding closing brace.
3. C programs are written in lowercase letters. However, uppercase letters are used for symbolic names and output strings.
4. All the words in a program line must be separated from each other by at least one space, or a tab, or a punctuation mark.
5. Every program statement in a C program must end with a semicolon.

6. All variables must be declared for their types before they are used in the program.
7. We must make sure to include header files using # include directive when the program refers to special names and functions that it does not define.
8. Compiler directives such as define and include are special instructions to the compiler to help it compile a program. They do not end with a semicolon.
9. The sign # of compiler directives must appear in the first column of the line.
10. When braces are used to group statement, make sure that the opening brace has a corresponding closing brace.
11. C is a free – form language and therefore a proper form of indentation of various sections would improve legibility of the program.
12. A comment can be inserted almost anywhere a space can appear. Use of appropriate comments in proper places increases readability and understandability of the program and helps users in debugging and testing. Remember to match the symbols /* and */ appropriately.

OPERATORS and EXPRESSIONS

'C' supports a rich set of operators.

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations
- Operators are used in programs to manipulate data and variable. They usually form a part of the mathematical or logical expression.
- The data items that operators act upon are called operands. Some operators require two operands, while other act upon only one operand.



ARITHMETIC OPERATORS

- C provides all the basic arithmetic operators.
  They are as follows:

Operator	Meaning
+	Addition or Unary plus
–	Subtraction or unary minus
*	Multiplication
/	Integer Division
%	Modulo Division

- **Examples:**

  a – b a + b
  a × b a / b
  a % b –a × b

  } a & b are variables and are known as operands

Integer Arithmetic :

- When both the operands in a single arithmetic expression such as a + b are integers, the expression is called an integer expression and the operation is called integer arithmetic.
- It always gives integer values.
- **Examples :**

  Let a = 14, b = 4
  a – b = 10
  a + b = 18
  a * b = 56
  a / b = 3 ⇒ (14/4 ⇒ It truncates fractional part)
  a % b = 2 ⇒ (14 % 4 ⇒ It gives remainder)

  During integer division

  → If both the operands are of the same sign, the result is truncated towards zero.
  e.g. 6/7 = 0 or –6 / –7 = 0

  → If one of them is negative, the direction of truncation is implementation dependent.
  e.g. : –6/7 = 0 or –1 ⇒ (machine dependent)

  During modulo division

  → The sign of the result is always the sign of the first operand
  e.g. –14 % 3 = –2
  – 14 % –3 = –2
  14 % –3 = 2

Real Arithmetic

- An arithmetic operation with real operands is called Real arithmetic.
- A Real operand may assume values either in decimal or exponential notation.
- If x, y, z are floats, then we will have
  x = 6.0/7.0 = 0.857143
  y = 1.0/3.0 = 0.333
  z = –2.0/3.0 = –0.66667
- The % operator cannot be used with Real operands

Mixed mode Arithmetic:

- When one of the operands is real and the other is integer, the expression is called a mixed–mode arithmetic expression.
- If either operand is of the real type, then only the real operation is performed and the result is always a real number.
  e.g. : 15/10.0 = 1.5
  where as 15/10 = 1

RELATIONAL OPERATORS

- These operators are generally used for comparisons. Expression containing relational operators is termed as relational expression
- The value / Result of the Relational expression is either one if the specified relation is true or zero if the specified relation is false.
  e.g. $10 < 20 \Rightarrow$ True (1)
  $20 < 10 \Rightarrow$ False (0)
- Relational operators are :

Operator	Meaning
<	less than
< =	less than or equal to
>	greater than
> =	greater than or equal to
= =	equal to
! =	not equal to

Syntax :

Arithmetic Expression 1 Relational operator Arithmetic expression 2

In this, arithmetic expression may contain simple constant, variables or combination of them.

Examples :

4.5 < = 10 $\Rightarrow$ True
4. 5 < – 10 $\Rightarrow$ False
10 < 7 + 5 $\Rightarrow$ True
a + b = = c + d $\Rightarrow$ True (only if sum of (a + b) = sum of (c + d))

- Arithmetic operators having higher priority than relational operators.
  $\therefore$ Arithmetic expressions are evaluated first and then compared.

LOGICAL OPERATORS

- The logical operators are used to evaluate logical expression.

Operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Each of the logical operators falls into its own precedence group. *Logical and* has a higher precedence than *logical or*. Both precedence groups are lower than the group containing the equality operators. The associativity is left to right. C also includes the unary operator ! that negates the value of a logical expression, i.e., it causes an expression that is originally true to become false, and vice versa. This operator is referred to as the *logical negation* (or *logical not*) operator.

Example : a > b && x = = 10

An expression of this kind which combines two or more relational expression is termed as a logical expression or compound relational expression.

Examples :

(1) if (age > SS && salary < 1000)
(2) if (number < 0 || number > 100)

ASSIGNMENT OPERATORS

- Assignment operator is used to assign the result of an expression to a variable. We usually use '='
- We can also use shorthand assignment operator.

  Syntax: `Vop = exp;` equivalent to `V = Vop(exp) ;`

  where, V → variable
  op → operator
  exp → expression

- *Assignment operator* = and the *equality operator* = = are *distinctly different*. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value.
- If the two operands in an assignment expression are of different data types. Then the value of the expression on the right (i.e., the right hand operand) will automatically be converted to the type of the identifier on the left. The entire assignment expression will then be of this same data type.

  Under some circumstances, this automatic type conversion can result in an alteration of the data being assigned.

For Example :

- A floating point value may be truncated if assigned to an integer identifier.
- A double–precision value may be rounded if assigned to a floating point (single precision) identifier.

Assignment operators have a lower precedence than any of the other operators that have been discussed so far. Therefore unary operations, arithmetic operations, relational operations, equality operations and logical operations are all carried out before assignment operations.

Examples :

Statement with simple assignment operator	statement with shorthand operators
a = a + 1	a + = 1
a = a – 1	a – = 1
a = a*(n + 1)	a* = n + 1
a = a / (n + 1)	a / = n + 1
a = a % b	a % = b

- **Advantages of shorthand operators :**

1. What appears on the left–hand side need not be repeated and therefore it becomes easier to write
2. The statement is more concise and easier to read.
3. The statement is more efficient.

Example :

value (5 * j – 2) = value (5 * j – 2) + delta ;

This can be written as, value (5 * j – 2) + = delta

INCREMENT AND DECREMENT OPERATORS

- 'C' Supports ⇒ Increment operator → + + ⇒ adds 1
  ⇒ Decrement operator → –– ⇒ subtracts 1
- Both are unary operators. It takes the following form :

Preorder Operator { ++ m or m ++ ; } Postorder Operator
Preorder Operator { –– m or m –– ; } Postorder Operator

- **++** m & m ++ means
  → The same thing when they form statement independently.
  → It behave differently when they are used in expression on the RHS of an assignment statement.

- **Example** :

(1) m = 5;
y = ++m ; } ⇒ A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left
O/P :
y = m = 6

(2) m = 5;
y = m ++ ; } ⇒ A postfix operator first assigns the value to the variable on left and then increment its value by 1.

O/P :
y = 5
m = 6

CONDITIONAL OPERATOR

- A ternary operator pair " ? : " is available in C to construct a conditional expression
- Syntax :

  Exp = exp1 ? exp 2 : exp 3 ;

  → In this case, exp 1 is calculated first
  If exp 1 = true
  then exp 2 is evaluated and it becomes the value of Exp.
  → If exp 1 = false
  then exp 3 is evaluated and it becomes the value of Exp. i.e. only one exp. is evaluated at a time.

- **Example :**
  a = 10 ;
  b = 15 ;
  x = (a > b) ? a : b ;
  Exp^{n} (a > b) is evaluated first, in this case (10 > 15) is false.
  ∴ x = b i.e. x = 15 is the result.

BITWISE OPERATORS

- 'C' has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level.
- These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied for float or double.
-

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
Λ	Bitwise Ex–OR
< <	Shift – Left
> >	Shift – right
∼	One's complement

SPECIAL OPERATORS

- C supports some special operators such as comma, sizeof operators.
- The comma operator can be used to link the related expressions together. A comma–linked list of expressions are evaluated left to right and the value of right–most expression is the value of the combined expression.
  Example : value = (x = 10, y = 5, x +y) ;
  It first assigns the value 10 to x, then assign 5 to y, and finally assigns 15(i.e. 10 + 5) to value. Since, comma operator has the lowest operator precedence of all operators, the parentheses are necessary.
  Use of comma operator:
  (1) In for loops :
  for (n = 1, m = 10; n < = m; n++, m++)
  (2) In while loops :
  while (c = getchar ( ), c! = '10')

THE sizeof OPERATOR

The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.

Examples :

```
m = sizeof (sum) ;
n = sizeof (long int) ;
```

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

EXPRESSIONS AND ITS COMPONENTS

- An expression represents a single data item, such as a number or a character. The expression may consist of a single entity, such as a constant, a variable, an array element or a reference to a function.

- It may also consist of some combination of such entities interconnected by one or more operators.
  The use of expressions involving operators is particularly common in C, as in most other programming languages.

- Expression can also represent logical conditions that are either true or false. However, in C the conditions true and False are represented by the integer values 1 and 0, respectively. Hence, logical type expressions represent numerical quantities.

Arithmetic Expressions

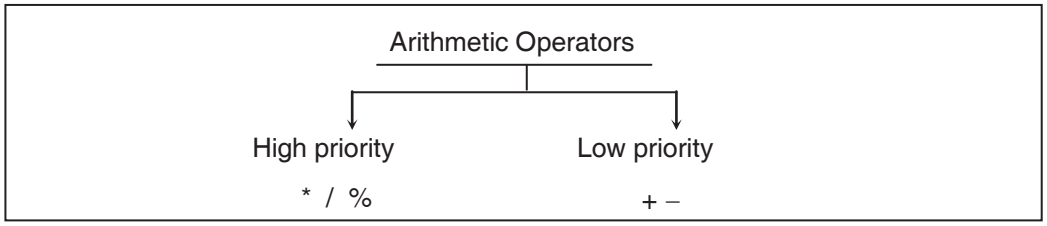
An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.

- Evaluation of expressions
  Expressions are evaluated using an assignment statement of the form :

Variable = expression ;

Algebraic expression	C – Expression
$a \times b - c$	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
$\frac{ab}{c}$	a*b/c
$3x^2 + 2x + 1$	3 * x * x + 2 * x +1
$\frac{x}{y} + c$	x / y + c

PRECEDENCE OF ARITHMETIC OPERATORS



The basic evaluation procedure includes two left – to – right passes through expression.
During first pass : High priority operators are applied as they are encountered
During second pass : Low priority operators are applied as they are encountered.

Example to study operator precedence

```
# include  < stdio.h >
   main  ( )
   { float a, b, c, x, y, z ;
     a = 9 ;
      b = 12;
      c = 3;
     x = a – b / 3 + c * 2 – 1 ;
     y = a – b / (3 + c) * (2 – 1) ;
     z = a – (b / (3 + c) * 2) – 1 ;
     printf ("x = % f \n", x) ;
     printf ("y = % f \n", y) ;
     printf ("z = %f \n", z) ;
   }
```

Note :

For solving such kind of problem, apply all basic rules of operators and expression.

O/P :

x =10.000
y = 7.000
z = 4.000

AUTOMATIC TYPE CONVERSION

'C' permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversions.

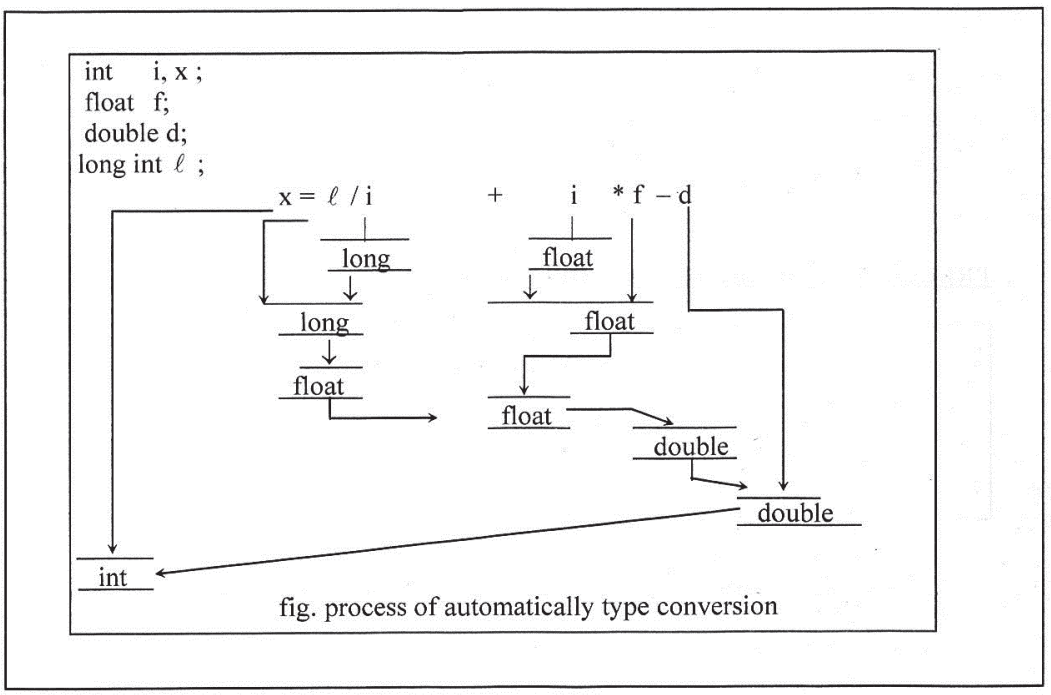


fig. process of automatically type conversion

Summary of C operators

Operators	Description	Associativity	Ranks
( ) [ ]	function call Array element reference	Left to Right	1
+ – ++ –– ! ~ * & size of (type)	Unary plus unary minus increment decrement logical not one's complement pointer reference address size of an object Type cast	Right to left	2
* / %	multiplication Division Modulus	Left to right	3
+ – < < > >	Addition Subtraction Left shift Right shift	Left to Right Left to Right	4 5
< < =	less than less than or equal to	Left to Right	6
> > =	Greater than Greater than or equal to		
= = ! =	Equality Inequality	Left to Right	7
& ^ | && | | ? :	Bitwise AND Bitwise XOR Bitwise OR Logical AND Logical OR Conditional Exp^{n}	Left to Right Left to Right Left to Right Left to Right Left to Right Right to Left	8 9 10 11 12 13
= * = / = % = + = – = & = ^ = | = < < = > > =	Assignment operators	Right to left	14
Comma (,) Operator		Left to Right	15

DATA INPUT/OUTPUT

Unlike other high – level language, C does not have any built in input /output statements as part of its syntax. All input / output operations are carried out through function calls such as printf and scanf. There exist several functions that have more or less become standard for input and output Operations in C. These functions are collectively known as the standard I /O Library.

The file name stdio.h is an abbreviation for standard input – output header file. The instruction # include < stdio. h > tells the compiler 'to search for a file named stdio. h and place its contents at this point in the program. The contents of the header file become part of the source code when it is compiled.

READING A CHARACTER

- The simplest of all input / output operations is reading a character from the standard input unit (usually the key board) and writing it to the standard output unit (usually the screen)
- Reading a single character can be done by using the function getchar.
  Syntax :

  Variable – name = getchar ( ) ;

  where ¸ variable – name is a valid C name that has been declared as char type.
- When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to getchar function.
- Since getchar is used on the R. H. S. of an assignment statement, the character value of getchar is in turn assigned to the variable name on the left.

For example :

```
char  name ;
name = getchar ( ) ;
```

The **getchar( )** function accepts any character keyed in. This includes RETURN and TAB. This means that when we enter single character input, the newline character is waiting in the input queue after **getchar( )** returns.

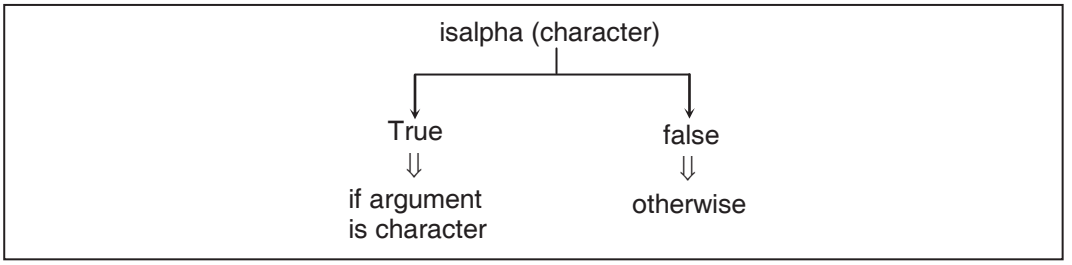
Character Test Functions

We can use different functions to test a given character is an alphabet or digit or any other special character.
Various functions are :

Functions	Test
isalnum (C)	is C an alphanumeric character ?
isalpha (C)	is C an alphabetic char ?
isdigit (C)	is C a digit ?
islower (C)	is C a lower case letter ?
isupper (C)	is C an upper case letter ?
isprint (C)	is C a printable character ?
ispunct (C)	is C a punctuation mark ?
isspace (C)	is C a white space character ?

All these functions are included in ctype . h

Example :



WRITING A CHARACTER

Like getchar, there is an analogous function putchar for writing characters one at a time to the terminal.

Syntax :

```
putchar (Variable–name);
```

where variable – name is a type char variable containing a character. This statement displays the character contained in the variable–name at the terminal.

Example :

```
ans = 'y' ;
putchar (ans) ;
```

getchar function

- Single character can be entered into the computer using the C library function getchar. The getchar function is a part of the standard C language I/O Library. It returns a single character from a standard input device.
- The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.
  A reference to the getchar function is written as
  Character – var = getchar ( ) ;
- If an end – of – file conditions is encountered when reading a character with the getchar function, the value of the symbolic constant EOF will automatically be returned.
- The detection of EOF in this manner offers a convenient way to detect an end–of–file, whenever and wherever it may occur.
- The getchar function can also be used to read multi character strings, by reading one character at a time within a multipass loop.

putchar function

- Single characters can be displayed using the C library function putchar. This function is complementary to the character input function getchar.
- The putchar function, like getchar, is a part of the standard C language I/O library. It transmits a single character to a standard output device.
- The character being transmitted will normally be represented as a character type variable. It must be expressed as an argument to the function, enclosed in parenthesis, following the word putchar. It is written as :

  putchar (char – variable) ;

  where, char–variable is valid C–character variable.
- The putchar function can be used to output a string constant by storing the string within a one dimensional, character type array. Each character can then be written separately within a loop.

gets ( )

It receives a string from the input device. We can do the same by scanf function but it has some limitation. Let's consider following example :

```
main ( )
{    char name [50] ;
     printf (" /n Enter Name" ) ;
     scanf (" % s", name) ;
          printf (" % s", name) ;
}
```

O / P {Expected }
Enter Name
Jonty Rhodes
Actual O/p after execution
Jonty

Due to the blank space, it never stored in an array. scanf assumes that it is end of string once a blank space is encountered. So to avoid this problem we use gets ( ). It accepts the string from keyboard and terminate when Enter key is pressed. Therefore, space and tab is acceptable.

Syntax :

```
gets (string) ;
```

puts ( )

It works exactly opposite to gets ( ). i.e. it outputs the string on monitor.

Syntax :

```
puts (string) ;
```

write a program to study gets ( ) and puts ( ) function

```
# include < stdio.h >
main ( )
{
     char cricket [40] ;
     puts ("Enter Name") ;
     gets (cricket) ; / *  sends bare address of an array * /
```

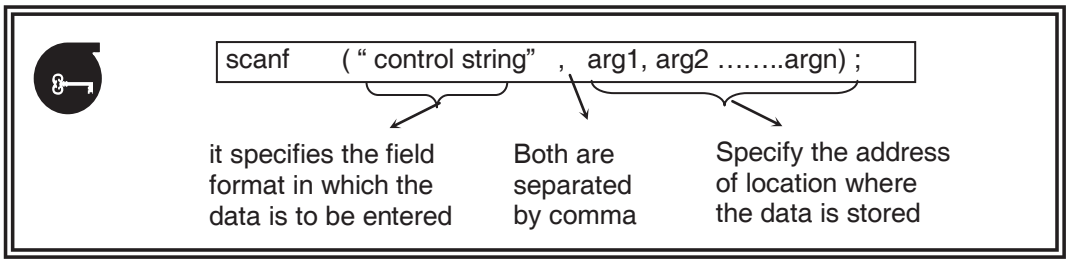
```
        puts ("Cricket is a funny game") ;
        puts (cricket) ;
    }
```

Output :

```
Enter Name
Sachin Tendulkar
Cricket is a funny game
Sachin Tendulkar
```

FORMATTED INPUT

Formatted input refers to an input data that has been arranged in a particular format.
General syntax :

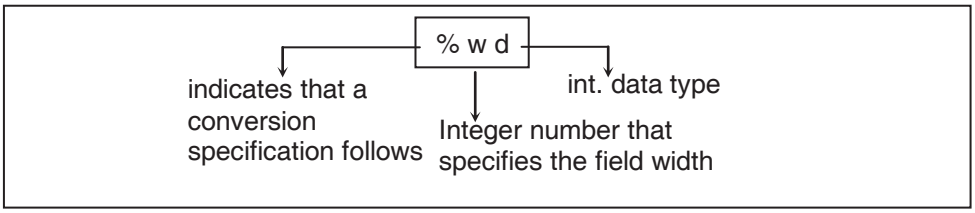


- **Control string** contains field specification which direct the interpretation of input data. It may include :

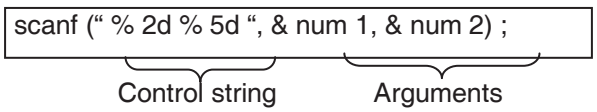
→ Field (or format) specifications, consisting of the conversion character %, a data type character, and an optional number specifying the field width.

→ Blanks, tabs or newlines.
Blanks, tables and newlines are ignored. The data type character indicates the type of data that is to be assigned to the variable associated with the corresponding argument.
The field width specifier is optional.

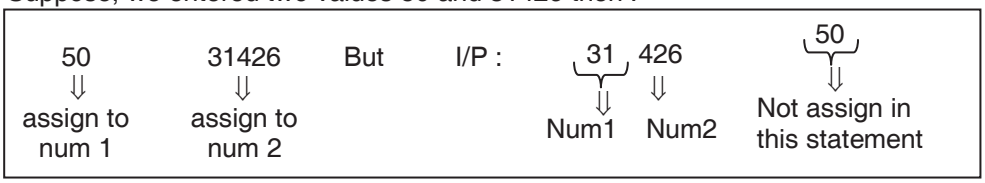
(a) Integer numbers :
The field specification for reading an integer number is :



Example :



Suppose, we entered two values 50 and 31426 then :



50 is not assigned in this scanf statement. 50 will be assigned to the first variable in the next scanf call.

- To avoid this types of errors we can write a scanf statement without w.

```
scanf (" % d  % d ",  & num1, & num2) ;
```

- An I/P field may be skipped by specifying * in the place of field width.

Example :

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

We will assign the data : 50 100 150
It is as follows : 50 to a
100 is skipped
150 to b

(b) Real numbers :

- field width is not required in Real number.
- scanf reads real numbers using the simple specification %f
- **Example** :

```
scanf  (" %f  %f  %f", &x, &y, &z) ;
```

with i/p data : 475.89 43.21E –1 678
⇓ ⇓ ⇓
x y = 4.321 z

- If the number to be read is of double type, the specification should be % ℓ f instead of simple %f
- An input field may be skipped by specifying * in the place of field width.

(c) Character string :

- Single character can be read by getchar function, the same can be done by scanf function.
- In addition, a scanf function can input strings containing more than one character.
- We can use

```
% ws
```

or

```
% wc
```

Example. : scanf (" %ws", & name) ;
scanf (" %wc", & name) ;

(d) Reading Mixed data type :

- It is possible to use one scanf statement to input a data line containing mixed more data.

Example :

```
                int     char    float   string
scanf (" %d     %c      %f      %s",  &count,  &code,  &ratio,  &name ) ;
                                      ⇓        ⇓        ⇓        ⇓
data type             int      char     float    string
```

Points to Remember while using scanf

1. All functions arguments, except the control string, must be pointer to variable.
2. Format specifications contained in the control string should match the arguments in order.
3. I/P data items must be separated by spaces and must match the variables receiving the input in the same order.
4. The reading will be terminated, when scanf encounters an 'invalid mismatch' of data or a character that is not valid for the value being read.
5. When searching for a value, scanf ignores line boundaries and simply looks for the next appropriate character.
6. Any unread data items in a line will be considered as a part of the data input line to the next scanf call.
7. When the field width specifier w is used, it should be large enough to contain the i/p data size.

FORMATTED OUTPUT

It is highly desirable that the outputs are produced in such a way that they are understandable and are in an easy–to–use form. The printf statement provides certain features that can be effectively exploited to control the alignment and spacing of print–outs on the terminals.

- Syntax

  printf ("control string", arg1, arg2, ……argn) ;

  Control string consists of three types of items :
  1. Characters that will be printed on the screen as they appear.
  2. Format specifications that define the output format for display of each item.
  3. Escape sequence characters such as \n , \t and \b.
     - The control string indicates how many arguments follow and what their types are.
     - The arguments arg1, arg2 …argn are the variables whose values are formatted and printed according to the specification of the control string.
     - A simple format specification has the following form :

       % w. p type–specifier

where, w → an integer number that specifies the total number of columns for the output value

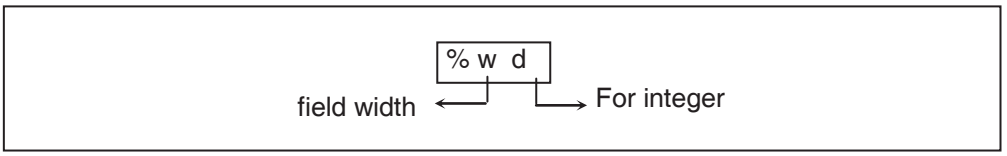
p → an integer number that specifies the number of digits to the right of the decimal point or the number of characters to be printed from a string.

- Examples :

```
printf ("programming in C") ;
printf ( "    " ) ;
printf (" \n" ) ;
printf (" % d", x) ;
printf ("a = %f \n b = %f", a, b) ;
printf ("sum = %d", 1234) ;
printf ("\n \n ") ;
```

Output of Integer numbers

- The format specification for printing an integer number is



- The number is written right–justified in the given field width.
- Example :

Format	output
(1) printf (" % d", 3846)	3 8 4 6
(2) printf ("%6d", 3846)	3 8 4 6 By default right justified.
(3) printf ("%2d", 3846)	3 8 4 6
(4) printf ("%-6d", 3846)	3 8 4 6 Left justified.
(5) printf ("%06d", 3846)	0 0 3 8 4 6

- The long integer may be printed by specifying ℓ d in the place of d in the format specification.

Output of Real Numbers

- The output of a real number may be displayed in decimal notation using the following format specification:

% w. pf

where : w $\rightarrow$ indicates the minimum number of positions that are to be used for the display of the value.
p $\rightarrow$ indicates the number of digits to be displayed after the decimal point.
(Precision)

- The value, when displayed, is rounded to p decimal places and printed right–justified in the field of w columns.
- Leading blanks and trailing zeros will appear as necessary. The default precision is 6 decimal places. The negative numbers will be printed with the minus sign.
- We can also display a real number in exponential notation by using the specification.

% w. pe

The field width w should satisfy the condition

$$w \geq p + 7$$

Example : y = 98.7654

Format	output
(1) printf (" % 7.4f ", y)	9 8 • 7 6 5 4
(2) printf ("%7.2f", y)	9 8 • 7 7
(3) printf ("%-7.2f",y)	9 8 • 7 7
(4) printf ("% f", y)	9 8 • 7 6 5 4
(5) printf ("% 10.2 e", y)	9 • 8 8 e + 0 1

Output of a single character

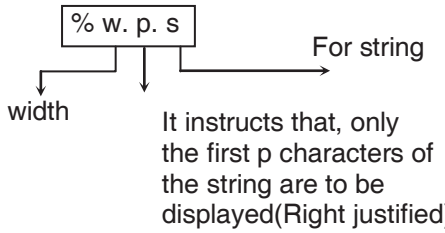
- A single character can be displayed in a desired position using the format,

% w c

- The character will be displayed right – justified in the field of w columns
- Left justified by placing minus sign (–) before w. The default value of w = 1

Printing of Strings :

The format specification for outputting strings is similar to that of real numbers.



Example : *Hello 123*

1) % s — H e ℓ ℓ o 1 2 3
2) % 10s — H e ℓ ℓ o 1 2 3
3) % .5s — H e ℓ ℓ o
4) % 10.5s — H e ℓ ℓ o
5) % –10.5s — H e ℓ ℓ o

Mixed data output :

It is permitted to mix data types in one printf statement.

```
printf (“ % d %f % s  % c ″, a, b, c, d ) ;
```

scanf format codes

Code	Meaning
% c	Read a single character
% d	Read a decimal integer
% e	Read a floating pt. value
% f	Read a floating pt. value
% g	Read a floating pt. value
% h	Read a short integer
% i	Read a decimal, hexadecimal or octal integer
% o	Read an octal integer
% s	Read a string
% u	Read an unsigned decimal integer
% x	Read a hexadecimal integer
% [..]	Read a string of word (s)

printf format code

Code	Meaning
% c	print a single character
% d	print a decimal integer
% e	print a floating pt. value in exponent form
% f	print a floating pt. value without exponent
% g	print a floating pt. value either e–type or f– type depending on value
% i	print a signed decimal integer
% o	print an octal integer, without leading zero
% s	print a string
% u	print an unsigned decimal integer
% x	print a hexadecimal integer, without leading 0’s

LMR (LAST MINUTE REVISION)

- C was an offspring of BCPL and discovered by Dennis Ritchie in 1972.
- C was developed along with Unix Operating system, so strongly associated with Unix.
- C is rich with built in functions and operations and combines the capabilities of an assembly language with the features of the high-level language.
- C program written for one machine can easily be run on another machine with little or no-modification. Also one can add our own function to its library.
- While space separate the words and compiler ignore them unless they are part of string.
- Keywords have fixed meaning which cannot be changed whereas identifiers are the name of variables, function and arrays.
- Constants refer to the fixed value that do not change during the execution of program.
- Allowable range for integer constant is -32768 to + 32767, for exponential form is -3.4e 38 to 3.4e 38.
- A variable may take different values at different times during execution and they are not keywords.
- Symbolic constant are unique constants in a program which may appear repeatedly in a number of places in the program.
- # define statements may appear anywhere in the program but before it is referenced in the program and must not end with a semicolon.
- Compiler directives such as define and include are special instructions to the compiler to help it compile a program.
- C is a free – form language and therefore a proper form of indentation of various sections would improve legibility of the program.
- Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.
- 'C' permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversions.
- sizeof operator is used to allocate memory space dynamically to variables during execution of a program.

- 'C' has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level.
- The comma operator can be used to link the related expressions together.
- Ternary operator pair " ? : " is available in C to construct a conditional expression.
- Assignment operator is used to assign the result of an expression to a variable.
- The logical operators are used to evaluate logical expression.
- The instruction # include < stdio. h > tells the compiler to search for a file named stdio.h and place its contents at this point in the program. The contents of the header file becomes part of the source code when it is compiled.
- Reading a single character can be done by using the function getchar.
- putchar function is use for writing characters one at a time to the terminal.
- getchar and putchar function does not require any arguments, though a pair of empty parentheses must follow those words.
- gets function receives a string from the input device and terminate when Enter key is pressed.
- puts outputs the string on monitor.
- Formatted input refers to an input data that has been arranged in a particular format.
- It is possible to use one scanf statement to input a data line containing mixed mode data.

❑ ❑ ❑ ❑ ❑ ❑

ASSIGNMENT – 1

Duration : 45 Min. **Max. Marks : 30**

Q 1 to Q 6 carry one mark each

1. We want to round off x, a float to an int value. What is the correct way to do so?
   (A) y = (int)(x + 0.5) (B) y = int(x + 0.5)
   (C) y = (int) x + 0.5 (D) y = (int)((int) x + 0.5)

2. By default any real number in 'C' is treated as
   (A) a float
   (B) a long double
   (C) a double
   (D) depends upon the memory model that you are using

3. Trace the output :
```
main ( )
{
    printf("%c", "abcdefgh"[4]);
}
```
   (A) Error (B) d
   (C) e (D) abcd

4. Trace the output :
```
main ( )
{
    printf("\n %x", –1 >> 4);
}
```
   (A) ffff (B) 0fff
   (C) 000 (D) fff0

5 The declarations
```
typedef float ht[100];
ht men, women;
```
defines _____
   (A) men and women as 100 element floating point arrays
   (B) men and women as floating point variables
   (C) ht, men and women as floating point variables
   (D) None of these

6. If a variable can take any integral values from 0 to n, where n is a constant integer, then the variable can be represented as a bit field whose width is the integral part of _____
   (A) $\log_2(n) + 1$ (B) $\log_2(n + 1) + 1$
   (C) $\log_2(n - 1) + 1$ (D) None of the above

Q7 to Q18 carry two marks each

7. Trace the output :
```
main ( )
{
    int i = 32, j = 0×20, k, l, m;
```

```
    k = i | j ;
    l = i & j
    m = k ^ l;
    printf("%d %d %d %d %d", i, j, k, l, m);
}
```

(A) 32 32 32 32 0 (B) 0 0 0 0 0
(C) 0 32 32 32 32 (D) 32 32 32 32 32

8. Trace the output :

```
main ( )
{
    unsigned int a = 0xffffffff ;
    a = ~ a;
    printf("%x", a);
}
```

(A) ffff (B) 0
(C) 00ff (D) None of these

9. If y is of integer type, then the expression
3 * (y – 8) / 9 and (y – 8) / 9 * 3
yield the same value if
(A) y is an even number
(B) y is an odd number
(C) (y – 8) is an integral multiple of 9
(D) (y – 8) is an integral multiple of 3

10. If y is of integer type, then the expressions
3 * (y – 8) / 9 and (y – 8) /9 * 3
(A) must yield the same value
(B) must yield different values
(C) may or may not yield the same value
(D) None of the above

11. Trace the output :

```
    int x, y = 2, z, a;
    x = (y*= 2) + (z = a = y);
    printf("%d", x);
```

(A) prints 7 (B) prints 6
(C) prints 8 (D) is syntactically wrong

12. What is the result of execution of the following 'C' program fragment?

```
    int i = 107, x = 5;
    printf((x > 7) ? "%d" : "%c", i);
```

(A) an execution error (B) a syntax error
(C) printing k (D) None of the above

13. Consider the following program fragment:

```
main ( )
{
    int a, b, c;
    b = 2;
    a = 2 * (b++);
```

```
    c = 2 * (++b);
}
```

Which of the following is correct?

(A) a = 4, c = 6 (B) b = 3, c = 6
(C) a = 3, c = 8 (D) a = 4, c = 8

14. Trace the output:

```
main ( )
{
    int y = 128;
    const int x = y;
    printf("%d", x);
}
```

(A) 128 (B) Garbage value
(C) Error (D) 0

15. Match the following :

(a) \a	(i)	tab
(b) \v	(ii)	apostrophe
(c) \"	(iii)	carriage return
	(iv)	bell
	(v)	quotation mark

(A) a – ii, b – iv, c – iii (B) a – iv, b – i, c – v
(C) a – i, b – iv, c – v (D) a – v, b – i, c – iii

16. What will be the values of a, b, and c after execution?

```
int a, b, c;
b = 10;
c = 15;
a = ++b + c++;
```

(A) a = 25, b = 10, c = 15 (B) a = 27, b = 10, c = 15
(C) a = 26, b = 11, c = 16 (D) a = 27, b = 11, c = 16

17. Consider a following declaration:

```
i = 3.3, j = 3.9, k = –3.9;
printf("%d %d %d", i, j, k) ;
```

The select the correct the statement?

(A) 3 4 – 4 (B) 3 3 –3
(C) 3 3.9 –3.9 (D) None of these

18. Consider a following declaration

```
int i = 7;
float f = 8.5;
```

then which of the following statement is correct?

(i) (i + f) % 4 (ii) ((int)(i +f)) % 4
(iii) ((float)(f + i)) % 4 (iv) ((int)f) % 2
(A) Only (ii) (B) (i) and (iii)
(C) (ii) and (iv) (D) All of the above

❑ ❑ ❑ ❑ ❑ ❑

TEST PAPER – 1

Duration : 30 Min. **Max. Marks : 25**

Q 1 to 5 Carry one mark each

1. The rule for implicit type conversion in 'C' is :
   (A) int < unsigned < float < double
   (B) unsigned < int < float < double
   (C) int < unsigned < double < float
   (D) unsigned < int < double < float

2. Which of the following statements is/are correct ?
   (A) enum variables can be assigned new values
   (B) enum variables can be compared
   (C) enumeration feature does not increase the power of C
   (D) all of the above.

3. main ( )
```
{
    int i;
    for (i = '64'; i <= '127'; i++)
    {
        printf("%d", i);
    }
}
```
In the above declaration, the value of first and last i is _____
   (A) A and ~
   (B) ? and DEL
   (C) A and DEL
   (D) None of these

4. The difference between rand( ) and srand( ) is _____
   (A) There is no difference between rand( ) and srand(u) both do same function.
   (B) The return type of rand( ) function is int and that of srand( ) is void.
   (C) The srand( ) function is used to initialize the random number generator whereas rand( ) function returns a random positive integer.
   (D) Both (B) and (C)

5. A _____ is the value of a variable or an expression which is displayed continuously as the program executes.
   (A) breakpoint value
   (B) watch value
   (C) step value
   (D) error value

Q6 to 13 Carry two marks each

6. Consider a following declaration

```
main ( )
{
    int i, n;
    float p;
    scanf("%d %d %f", &i, &n, &p);
    printf("%f", p*(pow((1 + i), n)));
}
```

Select the correct statement :

(A) The above program is used for finding the Fibbonacci series with n numbers and n power.

(B) The above program is use for finding compound interest.

(C) The above program is use for finding $(a+(a+1)^1+(a+2)^2+.....)$

(D) The above program is also use for $(1+i^1+i^2+i^3+.....+i^n)^n$

7. Consider the following declaration :

```
# include "goto.c"
# include <goto.c>
```

From which of the following statement is correct ?

(A) Both declaration are same and valid

(B) Both declaration are not same and not valid

(C) Both declaration are not same but both are valid

(D) Only second declaration is valid and both are same

8. Trace the final value of s and i

```
void main ( )
{
    int c = 1, s = 0, i = 1;
    while(c <= 5)
    {
        s = s + i;
        i = i + 2;
        c += 1;
    }
    printf("%d  %d", s, i);
}
```

(A) 36 13 (B) 25 11

(C) 64 17 (D) 81 21

9. Match the following :

1. Integrity	a. enhances the accuracy and clarity of program.
2. Efficiency	b. execution speed and proper memory utilization
3. Modularity	c. clarity and accuracy of a program are usually enhanced by it.
	d. refers to the accuracy of the calculations.

(A) 1 – a, 2 – c, 3 – d (B) 1 – b, 2 – d, 3 – c

(C) 1 – c, 2 – d, 3 – b (D) 1 – d, 2 – b, 3 – a

10. Trace the output :

```
main ( )
{
    float a = –4.9, b = –8.6; int c;
    c = floor (a) * a – (–b);
    printf("%d", c);
}
```

(A) 28.2 (B) 28
(C) 33.1 (D) 33

11. Consider the following declaration :

```
main ( )
{
    int a, b, c;
    c += (a > 0 && a <= 10) ? ++a : a | b;
}
```

Which of the following statements is false?

(A) If (a > 0 && a <= 10) is true then ++a with right to left associativity is evaluated.
(B) If (a > 0 && a <= 10) is false then a | b with left to right associativity is evaluated.
(C) ++, \, and += operators are always right to left associative.
(D) The value of C is increased by the value of conditional expression.

12. Select the correct statement :

(i) The sign # of compiler directives must appear in the first column of the line
(ii) C is free form of language
(iii) All statements, variables, constants are always written in lowercase in a C program.

(A) Only (i)
(B) Only (iii)
(C) (i) and (ii)
(D) (i), (ii) and (iii)

13. Consider the following declaration :

```
int i = 8, j = 5;
float x = 0.005, y = –0.01;
char c = 'c', d = 'd';
f = 2 * ((i/5) + (4 *(j – 3)) % (i + j – 2));
```

Then, the data type of f is _____ and value is _____

(A) float, 13
(B) int, 18
(C) float, 18
(D) int, 13

Q14(a) & (b) carry two marks each

Linked Answer Question

14(a).
```
main ( )
{
        int a = 6, b = 10, x;
        x = a & b;
        printf("%d", x);
}
```
Find the value of x.

(A) 2 (B) 7

(C) 10 (D) 5

14(b).
```
main ( )
{
    int z;
    float p;
    p = 6.8;
    z = (++x)*floor(p) × 3.5;
    printf("%d", z);
}
```
Find the value of z.

(A) 73 (B) 73.5

(C) 63 (D) 66.5

❑ ❑ ❑ ❑ ❑ ❑