SCS1102 FUNDAMENTALS OF PROGRAMMING

INTRODUCTION TO PROGRAMMING

CONTENT

UNIT- I

Introduction: Algorithms & flowcharts, Overview of C, features of C, Structure of C program, Compilation & execution of C program. Identifiers, variables, expression, keywords, data types, constants, scope and life of variables, and local and global variables. Operators: arithmetic, logical, relational, conditional and bitwise operators. Special operators: sizeof () & comma (,) operator. Precedence and associativity of operators & Type conversion in expressions. Input and output statements.

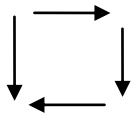
Algorithms & Flowcharts

The sequence of steps to be performed in order to solve a problem by the computer is known as an **Algorithm**. The Algorithm often refers to the logic of a program. Algorithms can be expressed in many different notations, including natural languages, pseudocode, flowcharts and programming languages.

Flowchart is a graphical or symbolic representation of an algorithm. It is the diagrammatic representation of the step-by-step solution to a given problem.

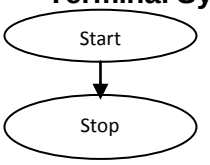
Flow Chart Symbols:

Flow Line Symbol



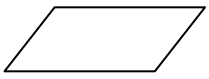
- These are the left, right, top & bottom line connection symbols
- These lines shows the flow of control through the program

Terminal Symbol



- The oval shape symbol always begins and ends the flowchart
- The Start symbol have only one flow line but not entering flow line
- The stop symbol have an entering flow line but not exit flow line

Input / Output Symbol

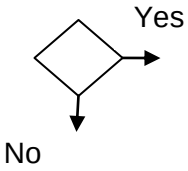


- The parallelogram is used for both Input(read) and output(write) operations

Process symbol

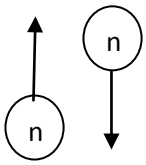
- The rectangle symbol is used primarily for calculations and initialization of memory location, all the arithmetic operations, data movements and initializations

Decision Symbol



- The diamond symbol is used in a flowchart to indicate the point at which a decision has to be made and a branch of two or more alternatives are possible
- There are always two exits from a decision symbol - one is labeled Yes or True and other labeled No or False.

Connector Symbol



- A connector symbol is represented by a circle with a letter or digit inside to specify the link.
- Used if the flowchart is big and needs continuation in next page

Let us take a small problem and see how can we write an algorithm using natural language and draw flowchart for the same.

Illustration

Consider the problem of finding the largest number in a given set of three numbers.

Algorithm

1. Get three numbers from the user
2. Compare the first two numbers
3. The larger of the first two numbers is compared with the third number
4. The larger number obtained as a result of the above execution is the largest number
5. Print the that number as output

Flowchart

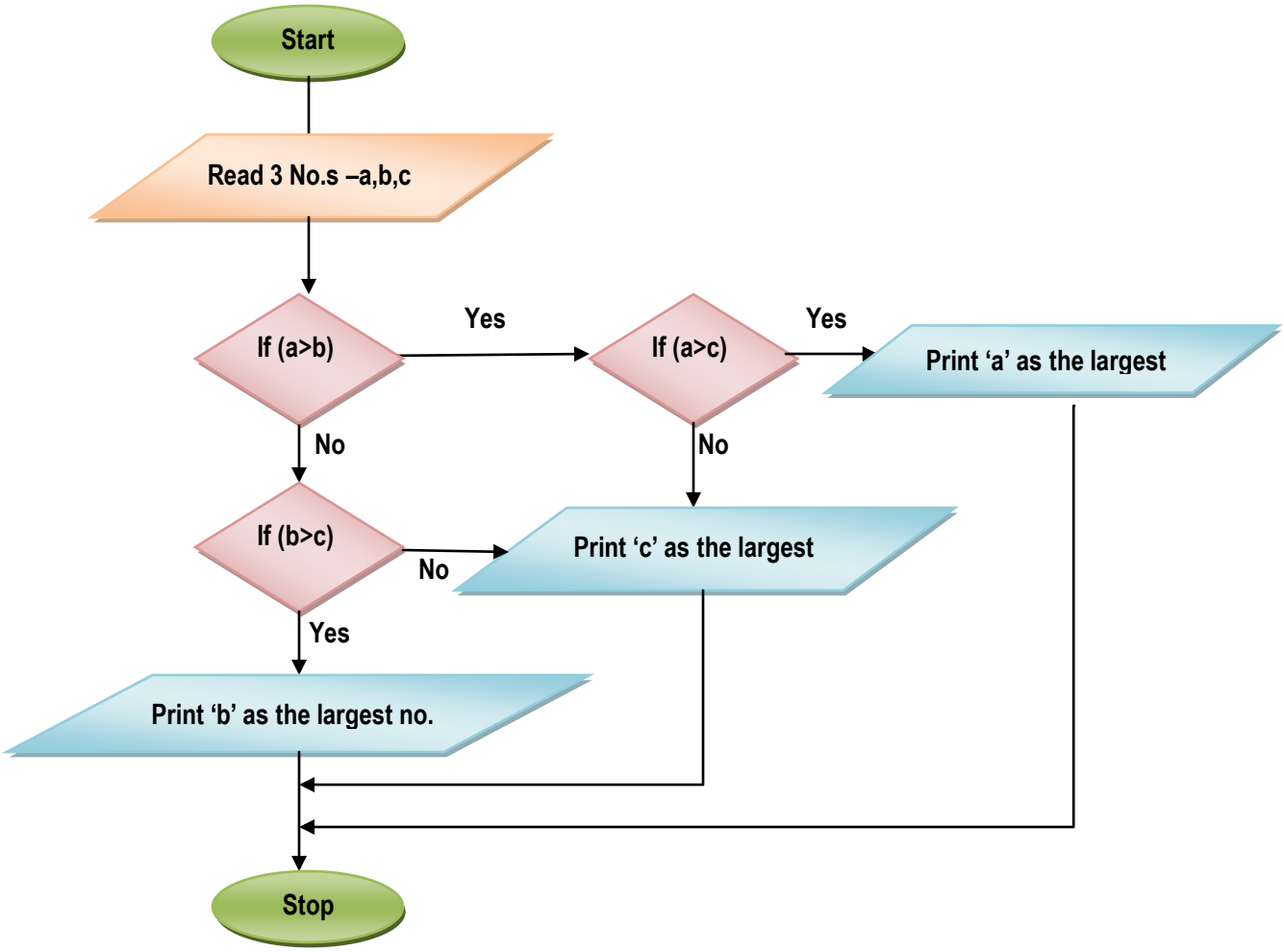


Fig 1. Flow chart for the program-finding the largest of 3 given no.s

Overview of C

A brief history

C is a programming language developed at “AT & T’s Bell Laboratories” of USA in 1972. It was written by Dennis Ritchie (Fig 2).

Fig 2. Dennis Ritchie

The programming language C was first given by Kernighan and Ritchie, in a classic book called “The C Programming Language, 1st edition”. For several years the book “The C Programming Language, 1st edition” was the standard on the C programming. In 1983 a committee was formed by the American National Standards Institute (ANSI) to develop a modern definition for the programming language C . In 1988 they delivered the final standard definition ANSI C.

Features of C

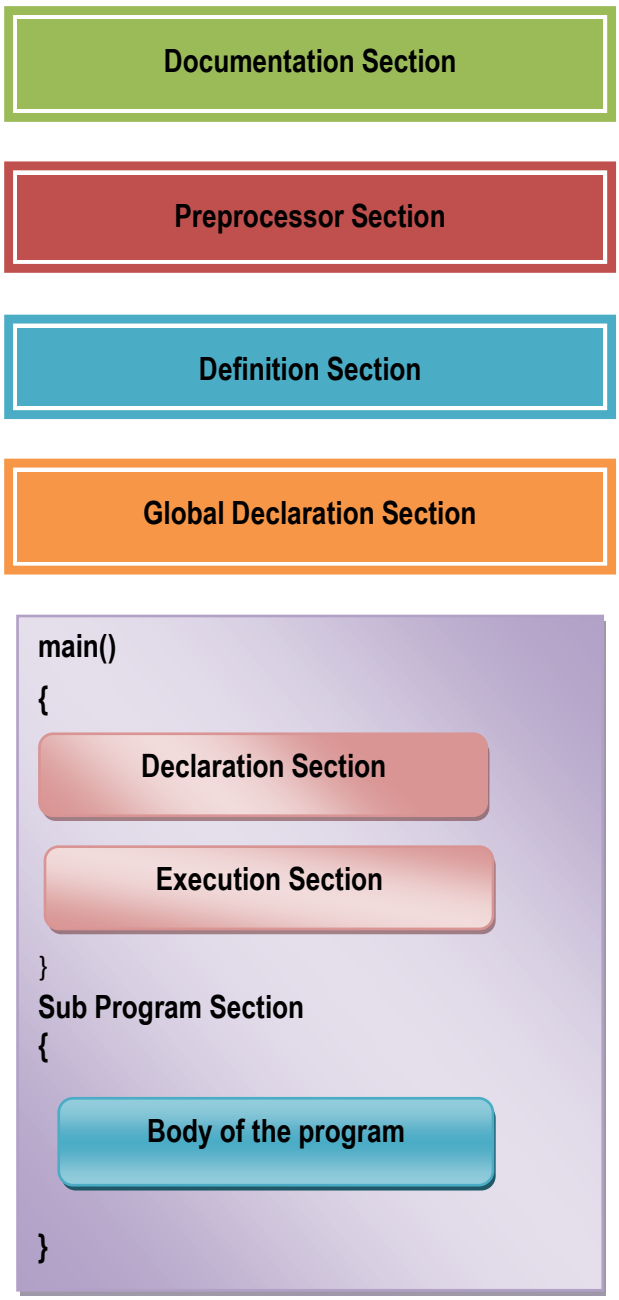
- Portability
- Modularity
- Extensible
- Speed
- Mid-level programming language
- Flexibility
- Rich Library

Advantages of C

1. C programming is the building block for many other high level programming languages existing today

2. A C program written in one computer can easily run on another computer without making any change
3. It has variety of data types and powerful operators
4. A C program is a collection of functions supported by the C library. So we can easily add our own functions to C library. Hence we can extend any existing C program according to the need of our applications
5. Since C is a structured language, we can split any big problem into several sub modules. Collection of these modules makes up a complete program. This modular concept makes the testing and debugging easier

Structure of a C program



Documentation Section:

- It consist of a set of comment lines
- The comment lines begins with /* and ends with */ or a single set of // in the beginning of the line
- These lines are not executable
- Comments are very helpful in identifying the program features

Preprocessor Section:

- It is used to link system library files, for defining the macros and for defining the conditional inclusion
- Eg: #include<stdio.h>, #include<conio.h>, #define MAX 100, etc.,

Global Declaration Section:

- The variables that are used in more than one function throughout the program are called global variables
- Should be declared outside of all the functions i.e., before main().

main():

Every 'C' program must have one main() function, which specifies the starting of a 'C' program. It contains the following two parts

Declaration Part:

- This part is used to declare the entire variables that are used in the executable part of the program and these are called local variables

Execution Part:

- It contains at least one valid C Statement.
- The Execution of a program begins with opening brace '{' and ends with closing brace '}'
- The closing brace of the main function is the logical end of the program

Sub Program section:

- Sub programs are basically functions are written by the user (user defined functions)
- They may be written before or after a main () function and called within main () function
- This is optional to the programmer

Constraints while writing a C program

- All statements in ‘C’ program should be written in lower case letters. Uppercase letters are only used for symbolic constants
- Blank space may be inserted between the words. Should not be used while declaring a variable, keyword, constant and function
- The program statements can be written anywhere between the two braces following the declaration part
- All the statements should end with a semicolon (;)

Example Program

```
/* addition.c – To find the average of two numbers and print them out
together with their average */

#include <stdio.h>

void main( )
{
  int first, second;
  float avg;
  printf("Enter two numbers: ");
  scanf("%d %d", &first, &second);
  printf("The two numbers are: %d,  %d", first, second);
  avg = (first + second)/2;
  printf("Their average is %f", avg);
}
```

Compilation and Execution of C program

1. Creating the program
2. Compiling the Program
3. Linking the Program with system library
4. Executing the program

Creating the program:

- Type the program and edit it in standard 'C' editor and save the program with **.c** as an extension.
- This is the source program

Compiling (Alt + F9) the Program:

- This is the process of converting the high level language program to Machine level Language (Equivalent machine instruction) -> Compiler does it!
- Errors will be reported if there is any, after the compilation
- Otherwise the program will be converted into an object file (.obj file) as a result of the compilation
- After error correction the program has to be compiled again

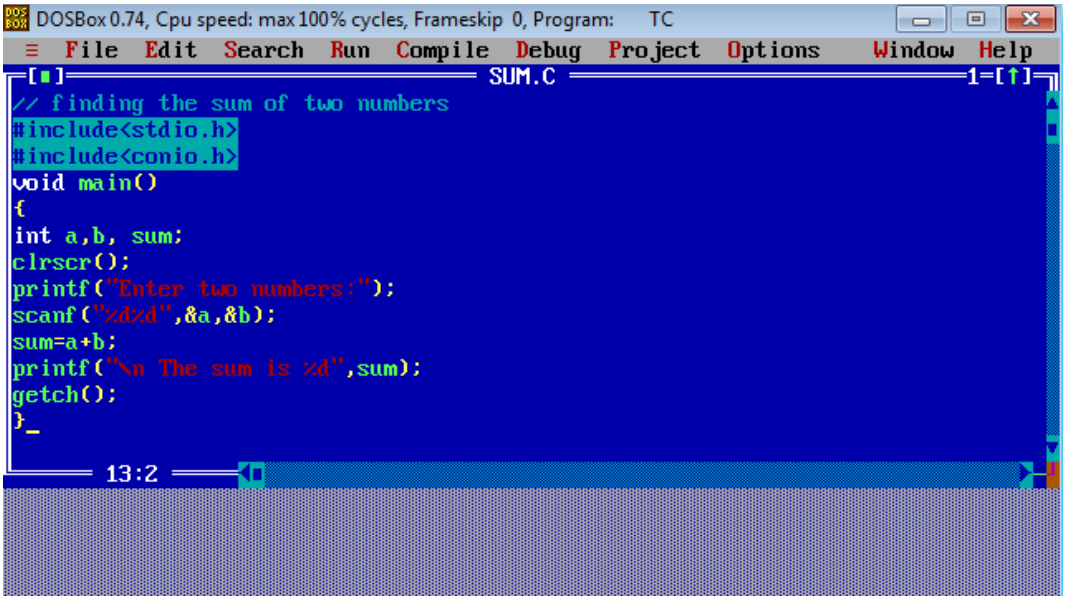
Linking the program with system Library:

- Before executing a c program, it has to be linked with the included header files and other system libraries -> Done by the Linker

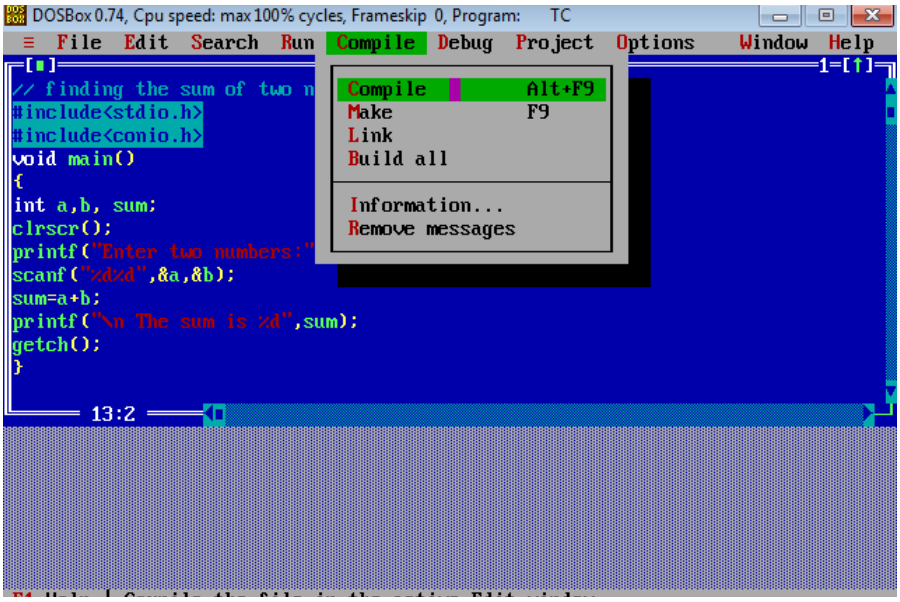
Executing the Program:

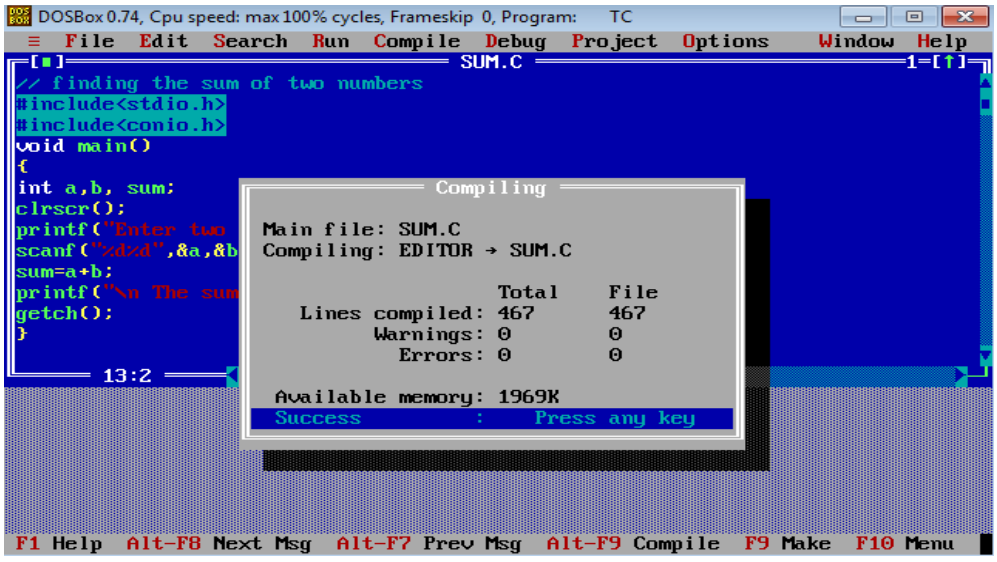
- This is the process of running (Ctrl + F9) and testing the program with sample data. If there are any run time errors, then they will be reported.

Creating the program

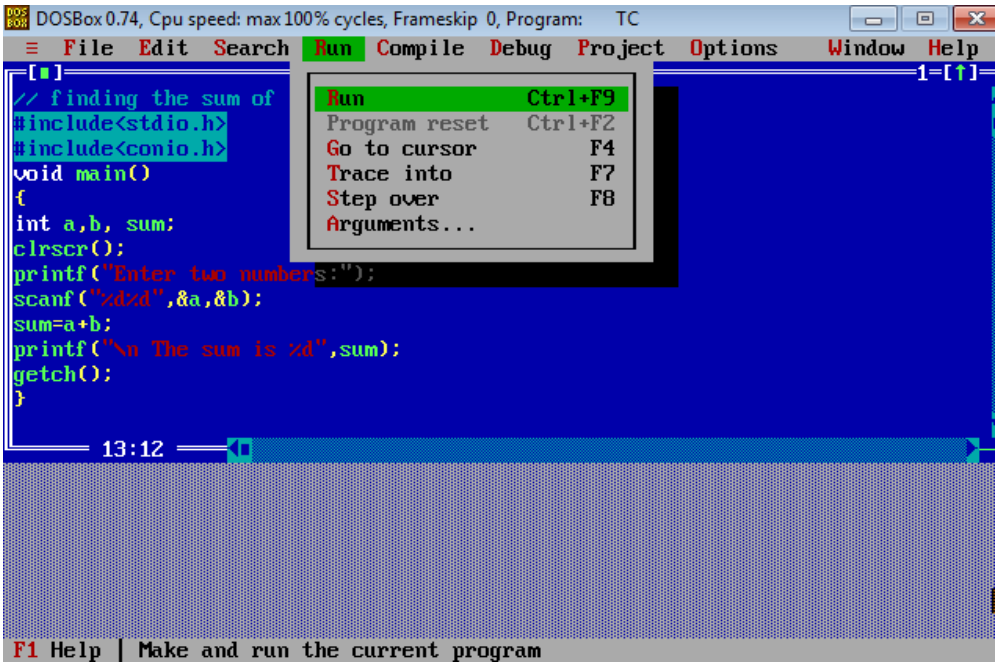


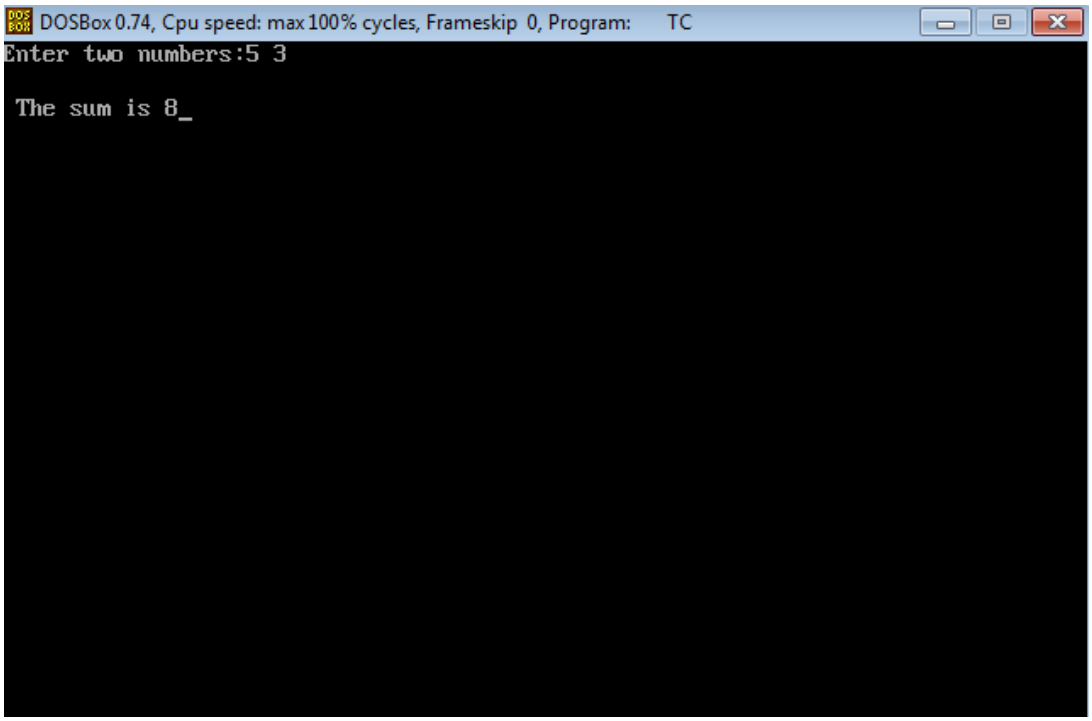
Compilation and Linking





Executing


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
File Edit Search Run Compile Debug Project Options Window Help
Run Ctrl+F9
Program reset Ctrl+F2
Go to cursor F4
Trace into F7
Step over F8
Arguments...
// finding the sum of
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b, sum;
clrscr();
printf("Enter two numbers:");
scanf("%d%d",&a,&b);
sum=a+b;
printf("\n The sum is %d",sum);
getch();
}
13:12
F1 Help | Make and run the current program


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter two numbers:5 3
The sum is 8_

The above illustration provides a lucid description of how to compile and execute a C program.

C Tokens

C tokens, Identifiers and Keywords are the basic elements of a C program. C tokens are the basic buildings blocks in C. Smallest individual units in a C program are the C tokens. C tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: “total”, “hello”),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

1. Keywords

Keywords are those words whose meaning is already defined by Compiler. They cannot be used as **Variable Names.** There are **32 Keywords** in C. C Keywords are also called as **Reserved words**. There are 32 keywords in C. They are given below:

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

2. Identifiers

Identifiers are the names given to various program elements such as variables , arrays & functions. Basically identifiers are the sequences of alphabets or digits.

Rules for forming identifier name

- The first character must be an alphabet (uppercase or lowercase) or an underscore.
- All succeeding characters must be letters or digits.
- No space and special symbols are allowed between the identifiers.
- No two successive underscores are allowed.
- Keywords shouldn't be used as identifiers.

3. Constants

The constants refer to fixed values that the program may not change or modify during its execution. Constants can be of any of the basic data types like an integer constant, a floating constant and a character constant. There is also a special type of constant called enumeration constant.

Eg:

Integer Constants- 45, 215u

Floating Constants- 3.14, 4513E-5L

Character Constants- \t, \n

4. Strings

A string in C is actually a one-dimensional array of characters which is terminated by a null character '\0'.

Eg:

char str = {'S', 'A', 'T', 'H', 'Y', 'A', 'B', 'A', 'M', 'A'}

5. Special Symbols

The symbols other than alphabets, digits and white spaces for example - [] () {} , ; : * … **= #** are the special symbols.

6. Operators

An Operator is a symbol that specifies an operation to be performed on the operands. The data items that operators act upon are called operands. Operators which require two operands are called Binary operators. Operators which require one operand are called Unary Operators.

Types of Operators

Depending upon their operation they are classified as

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators

5. Increment and Decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Sizeof() Operators

Arithmetic Operators

Arithmetic Operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

S.NO	Operators	Operation	Example
1	+	Addition	A+B
2	-	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

Rules For Arithmetic Operators

1. C allows only one variable on left hand side of = eg. c=a*b is legal, but a*b=c is not legal.
2. Arithmetic operations are performed on the ASCII values of the characters and not on characters themselves
3. Operators must be explicitly written.
4. Operation between same type of data yields same type of data, but operation between integer and float yields a float result.

Example Program

```
 #include <stdio.h>
int main()
{
int m=40,n=20, add,sub,mul,div,mod;
add = m+n;
sub = m-n;
mul = m*n;
div = m/n;
mod = m%n;
```

```
printf("Addition of m, n is : %d\n", add);
printf("Subtraction of m, n is : %d\n", sub);
printf("Multiplication of m, n is : %d\n", mul);
printf("Division of m, n is : %d\n", div);
printf("Modulus of m, n is : %d\n", mod);
}
```

Output

Addition of m, n is : 60
Subtraction of m, n is : 20
Multiplication of m, n is : 800
Division of m, n is : 2
Modulus of m, n is : 0

Relational Operators

Relational Operators are used to compare two or more operands. Operands may be variables, constants or expression

S.NO	Operators	Operation	Example
1	>	is greater than	m > n
2	<	is less than	m <n
3	>=	is greater than or equal to	m >= n
4	<=	is less than or equal to	m <= n
5	==	is equal to	m == n
6	!=	is not equal to	m!=n

Example Program

```
#include <stdio.h>
int main()
{
int m=40,n=20;
if (m == n)
{
printf(“m and n are equal”);
}
else
{
printf(“m and n are not equal”);
}
}
```

Output

m and n are not equal

Logical Operators

Logical Operators are used to combine the results of two or more conditions. It is also used to test more than one condition and make decision.

S.NO	Operators	Operation	Example	Description
1	&&	logical AND	(m>5)&&(n<5)	It returns true when both conditions are true
2	||	logical OR	(m>=10)||(n>=10)	It returns true when at-least one of the condition is true
3	!	logical NOT	!((m>5)&&(n<5))	It reverses the state of the operand “((m>5) && (n<5))” If “((m>5) && (n<5))” is true, logical NOT operator makes it false

Example Program

```
#include <stdio.h>
int main()
{
int a=40,b=20,c=30;
if ((a>b )&& (a >c))
{
printf(“ a is greater than b and c”);
}
else
if(b>c)
printf(“b is greater than a and c”);
else
prinf(“c is greater than a and b”);
}
```

Output

a is greater than b and c.

Conditional Operator

It itself checks the condition and executed the statement depending on the condition.

Syntax:

Condition? Exp1:Exp2

Example:

X=(a>b)?a:b

The ‘?:’ operator acts as ternary operator. It first evaluate the condition, if it is true then exp1 is evaluated, if condition is false then exp2 is evaluated. The drawback of Assignment operator is that after the ? or : only one statement can occur.

Example Program

```
#include <stdio.h>
int main()
{
int x,a=5,b=3;
x = (a>b) ? a : b ;
printf("x value is %d\n", x);
}
```

Output

x value is 5

Bitwise Operators

Bitwise Operators are used for manipulation of data at bit level. It operates on integer only.

S.NO	Operators	Operation	Example	Description
1	&	Bitwise AND	X & Y	Will give 1 only when both inputs are 1
2	|	Bitwise OR	X | Y	Will give 1 when either of input is 1
3	^	Bitwise XOR	X ^ Y	Will give 1 when one input is 1 and other is 0
4	~	1's Complement	~X	Change all 1 to 0 and all 0 to 1
5	<<	Shift left	X<<Y	X gets multiplied by 2^Y number of times
6	>>	Shift right	X>>Y	X gets divided by 2^Y number of times

Example Program

```
#include <stdio.h>
main()
{
int c1=1,c2;
c2=c1<<2;
printf(“Left shift by 2 bits c1<<2=%d”,c2);
}
```

Output

Left shift by 2 bits c1<<2=4

Special operators:

sizeof () operator:

1. Sizeof operator is used to calcualte the size of data type or variables.
2. Sizeof operator will return the size in integer format.
3. Sizeof operator syntax looks more like a function but it is considered as an operator in c programming

Example of Size of Variables

```
#include<stdio.h>
int main()
{
   int ivar = 100;
   char cvar = 'a';
   float fvar = 10.10;
   printf("%d", sizeof(ivar));
   printf("%d", sizeof(cvar));
   printf("%d", sizeof(fvar));
   return 0;
}
```

Output :

2 1 4

In the above example we have passed a variable to size of operator. It will print the value of variable using sizeof() operator.

Example of Sizeof Data Type

```
#include<stdio.h>
int main()
{
   printf("%d", sizeof(int));
   printf("%d", sizeof(char));
   printf("%d", sizeof(float));
   return 0;
}
```

Output :

2 1 4

In this case we have directly passed an data type to an sizeof.

Example of Size of constant

```
#include<stdio.h>
int main()
{

   printf("%d", sizeof(10));
   printf("%d", sizeof('A'));
   printf("%d", sizeof(10.10));
   return 0;
}
```

Output :

2 1 4

In this example we have passed the constant value to a sizeof operator. In this case sizeof will print the size required by variable used to store the passed value.

Example of Nested sizeof operator

```
#include<stdio.h>
int main()
{
   int num = 10;
   printf("%d", sizeof(sizeof(num)));
   return 0;
}
```

Output:

2

We can use nested sizeof in c programming. Inner sizeof will be executed in normal fashion and the result of inner sizeof will be passed as input to outer sizeof operator.
Innermost Sizeof operator will evaluate size of Variable “num” i.e 2 bytes Outer Sizeof will evaluate Size of constant “2” .i.e 2 bytes

Comma(,) Operator:

1. Comma Operator has Lowest Precedence i.e it is having lowest priority so it is evaluated at last.
2. Comma operator returns the value of the rightmost operand when multiple comma operators are used inside an expression.
3. Comma Operator Can acts as –

- **Operator :** In the Expression
- **Separator:** Function calls, Function definitions, Variable declarations and Enum declarations

Example:

```
#include<stdio.h>
void main()
{
   int num1 = 1, num2 = 2;
   int res;
   res = (num1, num2);
   printf("%d", res);
}
```

Output
2

Consider above example
int num1 = 1, num2 = 2;// In variable Declaration as separator
res = (num1, num2);// In the Expression as operator

In this case value of rightmost operator will be assigned to the variable. In this case value of num2 will be assigned to variable res.

Examples of comma operator:

Type 1 : Using Comma Operator along with Assignment

```
#include<stdio.h>
int main()
{
```

```
        int i;
        i = 1,2,3;
        printf("i:%d\n",i);
        return 0;
}
```

Output:

i:1

Explanation:

i = 1,2,3;

1. Above Expression contain 3 comma operator and 1 assignment operator.
2. If we check precedence table then we can say that “Comma” operator has lowest precedence than assignment operator
3. So Assignment statement will be executed first .
4. 1 is assigned to variable “i”.

Type 2 : Using Comma Operator with Round Braces

```
#include<stdio.h>
int main()
{
        int i;
        i = (1,2,3);
        printf("i:%d\n",i);
        return 0;
}
```

Output:

i:3

Explanation:

i = (1,2,3);

1. Bracket has highest priority than any operator.
2. Inside bracket we have 2 comma operators.
3. Comma operator has associativity from Left to Right.
4. Comma Operator will return rightmost operand

i = (1,2,3) Assign 3 to variable i.

Type 3 : Using Comma Operator inside printf statement

```
#include<stdio.h>
#include< conio.h>
void main()
```

```
{
clrscr();
printf("Computer","Programming");
getch();
}
```

Output:

Computer

You might feel that answer of this statement should be “Programming” because comma operator always returns rightmost operator, in case of printf statement once comma is read then it will consider preceding things as variable or values for format specifier.

Type 4 : Using Comma Operator inside Switch cases.

```
#include<stdio.h>
#include< conio.h>
void main()
{
 int choice = 2 ;
 switch(choice)
 {
  case 1,2,1:
        printf("\nAllas");
        break;

  case 1,3,2:
        printf("\nBabo");
        break;

  case 4,5,3:
        printf("\nHurray");
        break;
 }
}
```

Output :

Babo

Type 5 : Using Comma Operator inside For Loop

```
#include<stdio.h>
```

```
int main()
{
int i,j;
for(i=0,j=0;i<5;i++)
  {
  printf("\nValue of J : %d",j);
  j++;
  }
return(0);
}
```

Output:

Value of J : 0
Value of J : 1
Value of J : 2
Value of J : 3
Value of J : 4

Type 6 : Using Comma Operator for multiple Declaration

```
#include<stdio.h>
int main()
{
int num1,num2;
int a=10,b=20;
return(0);
}
```

Note : Use of comma operator for multiple declaration in same statement.

Variable:

- A variable is an identifier that is used to represent some specified type of information within a designated portion of the program.
- A variable may take different values at different times during the execution

Rules for naming the variable

- A variable name can be any combination of 1 to 8 alphabets, digit, or underscore
- The first character must be an alphabet or an underscore (_).
- The length of variable should not exceed 8 characters length, and some of the 'C' compiler can be recognize upto 31 characters.

Data Types in C

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.

The value of a variable can be changed any time.

C has the following basic built-in datatypes.

- int
- float
- double
- char

The bytes occupied by each of the primary data types are

Data type	Description	Memory bytes	Control String	Example
Int	Integer Quantity	2 bytes	%d or %i	int a=12;
Char	Single Character	1 bytes	%C	char s='n';
float	Floating Point	4 bytes	%f	float f=29.777
Double	Double precision floating pointing no's	8 bytes	%lf	double d= 5843214

Scope of a variable

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed. There are three places where variables can be declared in C programming language:

1. Inside a function or a block is called **local** variable,
2. Outside of all functions is called **global** variable.
3. In the definition of function parameters which is called **formal** parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>
```

```
main ()
{
  /* local variable declaration */
  int a, b, c;


  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;
  printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
 }
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <stdio.h>
/* global variable declaration */
int g;
main ()
{
  /* local variable declaration */
  int a, b;
   /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;
```

```
  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
 }
```

PRECEDENCE AND ASSOCIATIVELY OF OPERATORS

If an arithmetic expression is given, there are some rules to be followed to evaluate the value of it. These rules are called as the priority rules. They are also called as the hierarchy rules. According to these rules, the expression is evaluated as follows;

Rule 1 :- If an expression contains parentheses , the expression within the parentheses will be performed first. Within the parentheses , the priority is to be followed.
Rule 2 :- If it has more than parentheses , the inner parenthesis is performed first.
Rule 3:- If more than one symbols of same priority , it will be executed from left to right.

C operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied

Operator	Operation	Associativity	Priority
() [] . -> ++ --	Parentheses Brackets (array subscript) Dot operator Structure operator Postfix increment/decrement	left-to-right	1
++ -- + - ! ~ (type) * & sizeof	Prefix inc/decrement Unary plus/minus Not operator,complement Type cast Pointer operator Address operator Determine size in bytes	right-to-left	2
* / %	Multiplication/division/modulus	left-to-right	3
+ -	Addition/subtraction	left-to-right	4
<< >>	Bitwise shift left Bitwise shift right	left-to-right	5
< <= > >=	Relational less than less than or equal to Relational greater than greater than or equal to	left-to-right	6
== !=	Relational is equal to is not equal to	left-to-right	7
&	Bitwise AND Bitwise exclusive	left-to-right	8
^	Bitwise exclusive OR	left-to-right	9
|	Bitwise inclusive OR	left –to-right	10
&&	Logical AND	left-to-right	11
||	Logical OR	left-to-right	12
?:	Ternary conditiona	right-to-left	13

= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left	14
,	Comma	left-to-right	15

Example for evaluating an expression

Let X = 2 , Y =5 then the value of the expression
(((Y - 1) / X) * (X + Y)) is calculated as:-
(Y - 1) = (5 - 1) = 4 = T1
(T 1 / X) = (4 / 2) = 2 = T2
(X + Y) = (2 + 5) = 7 = T3
(T2 * T3) = (2 * 7) = 14
The evaluations are made according to the priority rule.

Type conversion in expressions.

Type conversion is the method of converting one type of data into another data type.

There are two types of type conversion.

1. Automatic type conversion

2. Type casting

Automatic type conversion

- This type of conversion is done automatically. The resultant value of an expression depends upon the operand which occupies more space, which means the result value converted into highest data type.
- The compiler converts all operands into the data type of the largest operand.
- This type of type conversion is done implicitly,this method is called as implicit type conversion.

Eg.1

```
float a,b,c;
```

a=10,b=3;

c=a/b

output= > c= 3.3 {4 bytes(float) (All the variables are same datatype}

Eg.2

int a,b,c;

a=10,b=3;

c=a/b;

output= >c=3{2 bytes(int)}

Eg.3

int a;

float b,c;

a=10,b=3;

c=a/b;

output=> c=3.3 {4 bytes(float) highest datatype is float}

Type casting

- This method is used,when user wants to change the type of the data.

General Format for type casting is

(datatype)operand

Eg.1

int x=10, y=3;

z=(float)x/y;(ie z=10.0/3;)

output=>z=3.3(float)

Eg:2

int x=10,y=3;

z=x/(float)y;(ie z=10/3.0;)

output=>3.3(float)

- The type of the x is not changed,only the type of the value can be changed
- Since the type of conversion is done explicitly,this type conversion is called as explicit type conversion

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.
3. All character types to be converted to integer

Input and Output statements

In 'c' language several functions ara available for input/output operations.These functions are collectively known as the standard I/O library.

1.Unformatted input /output statements
2. Formatted input **/**output statements

Unformatted Input /Output statements

These statements are used to input /output a single /group of characters from/to the input/output devices .Here the user cannot specify the type of data that is going to be input/output.

The following are the Unformatted input /output statements available in 'C'.

Input	Output
getchar( )	putchar( )
getc( )	putc( )
gets( )	Puts( )

single character input-getchar( ) function:

A getchar( ) function reads only one character through the keyboard.

Syntax: char variable=getchar( );

Example:

```
char x;
```

x=getchar( );

single character output-putchar( ) function:

A putchar( ) function is used to display one character at a time on the standard output device.

Syntax: putchar(charvariable);

Example:

```
char x;
putchar(x);
```

the getc( ) function

This is used to accept a single character from the standard input to a character variable.

Syntax: character variable=getc( );

Example:

```
char c;
c=getc( );
```

the putc( ) function

This is used to display a single character variable to standard output device.

Syntax: putc(character variable);

Example:

```
char c;
putc(c );
```

the gets( ) and puts( ) function

The gets( ) function is used to read the string from the standard input device.

Syntax: gets(string variable);

Example:

```
gets( s);
```

The puts( ) function is used to display the string to the standard output device.

Syntax: puts(string variable);

Example:

```
puts( s);
```

Proram using gets and puts function

```
#include<stdio.h>
```

```
main()
{
char scientist[40];
puts("Enter name");
gets(scientist);
puts("Print the Name");
puts(scientist);
}
```

output:

Enter Name:Abdul Kalam

Print the Name:Abdul Kalam

Formatted input /output statements

The function which is used to give the value of variable through keyboard is called ***input function***. The function which is used to display or print the value on the screen is called ***output function***.

Note : - In C language we use two built in functions, one is used for reading and another is used for displaying the result on the screen. They are scanf() and printf() functions. They are stored in the header file named stdio.h.

General format for scanf( ) function

scanf("control string", &variable1, &variable2,……)

The control sting specifies the field format in which the data is to be entered.

%d –integer

%f – float

%c- char

%s – string

% ld – long integer

%u – Unsigned Integer

Example:

scanf("%d",&x) – reading an integer value, the value will be stored in x

scanf("%d%f",&x,&a) - reading a integer and a float value In the above scanf ( ) function , we don't use any format. This type of Input is called as the Unformatted Input function.

Formatted Input of Integer

The field speciation for reading the integer number is:

%wd

Where The percentage sign(%) indicates that a conversion specification follows. w – is the field width of the number to be read. d will indicates as data type in integer number.

Example:

scanf("%2d %5d", &num1,&num2);

data line is 50 31425

the value 50 is assigned to num1 and 31425 is assigned to num2. suppose the input data is as follows

31425 50 , then the variable num1 will be assigned 31 and num2 will be assigned to 425 and the 50 is unread.

An input field may be skipped by specifying * in the place of field width.

Example the statement scanf("%d %*d %d),&a,&b); will assign the data 123 456 789

as follows: 123 is assigned to a , 456 skipped because of * and 789 to b

Output Function : To print the value on the screen or to store the value on the file, the output functions are used. printf() is the function which is use to display the output on the screen.

The General format of the printf() function is

printf("control string",variable1,variable2,…..);

Example

printf("%d",x) – printing the integer value x.

printf("%d%f", x,a)- printing a integer and float value using a single printf function.

Formatted Output of Integer : Similar to formatted input , there is a formatted output also to have the output in a format manner.

In this control string consists of three types of items.

- Characters that will be printed on the screen as they appear
- Format specification that define the output format for display of each item
- Escape sequence characters such as

  \n – new line

\b – back space

\f – form feed

\r – carriage return

\t - horizontal tab

\v – vertical tab

The format speciation is as follows

%wd

Where w – is the field width of the number to be write . d will indicates as data type in integer number.

Examples:

Printf(“%d”,9876); // output: 9876

printf(“%6d”,9876);

output:

1	2	3	4	5	6
		9	8	7	6

printf(“%-6d”,9876);

output:

1	2	3	4	5	6
9	8	7	6		

printf(“%06”,9876);

output:

1	2	3	4	5	6
0	0	9	8	7	6

Formatted input of Real(float) Numbers:

. The field speciation for reading the real number is:

%w.pf

Where w – is the field width of the number to be read . p indicates the number of digits to be read after the decimal point f – indicates that data type in float(real) number.

Example

scanf(“%2.1f %5.2f”,&num1,&num2);

data line is 50.1 31425.20

the value 50.1 is assigned to num1 and 31425.20 is assigned to num2.

An input field may be skipped by specifying * in the place of field width.

Example: the statement scanf("%f %*f %f), &a,&b); will assign the data 12.3 4.56 78.9 as follows: 12.3 is assigned to a , 4.56 skipped because of * and 78.9 to b.

Formatted output of Real(float) Numbers:

The field speciation for reading the real number is:

%w.pf

Where w – is the field width of the number to be read . p indicates the number of digits to be displayed after the decimal point f – indicates that data type in float(real) number.

Example:

Float y = 98.7682

Printf(" %f ", y); // output: 98.7682

printf("%7.2f ",y);

output:

1	2	3	4	5	6	7
		9	8	.	7	6

printf("%-7.2f ",y);

output:

1	2	3	4	5	6	7
9	8	.	7	6		

Formatted input of Single characters or strings:

The field speciation for reading the character strings:

%ws or %wc

where,

%c is used to read a single character.

Example:

Char name;

Scanf("%c", &name); \\ I / P : a

Char name[20];

Scanf(%s",&name); \\ I / P : sathyabama

Printing of a Single Character:

The field speciation for reading the character strings:

%ws or %wc

where,

%s – A sequence of characters can be displayed.

%c – A single character can be displayed.

The character will be displayed right-justified in the field of w, left-justified by placing a minus sign before the integer w.

Example:

```
Char x = 'a';
Char name[20] = "anil kumar gupta";
Printf("%c", x); // output: a
Printf("%s",name); // output: anil kumar gupta
Printf("%20s", name);
```

Output:

1	2	3	4	5	6	6	8	9	10	11	12	13	14	15	16	17	18	19	20
				a	n	i	l		k	u	m	a	r		g	u	p	t	a

Printf("%-20.10s", name);

Output:

1	2	3	4	5	6	6	8	9	10	11	12	13	14	15	16	17	18	19	20
a	n	i	l		k	u	m	a	r										

Printf("%.5s", name);

Output:

g	u	p	t	a

Questions for practice

1. Write the algorithm a) to calculate the average of three numbers. b) to check whether an entered number is odd or even.
2. Draw flowchart to find whether the given number is a prime number.
3. Write a program in 'C' to display your details like Name, Regno, Branch, 10^{th} marks, 12^{th} marks and your native state.
4. Write a program in 'C' to get two numbers as input from the user and then swap them.
5. Write a program in 'C' to print the numbers from 1 to 10 and their squares
6. Find the value for the following expression with a=10,b=5 and c=1

   i) a>9 && b!=3
   ii) a==5 || b!=3
   iii) !(a>14)
   iv) !(a>9 && y!=23)
   v) % &&b != 8|| 0
   vi) 3<<2
   vii) 4>>3
7. What will be output of the following program?

```
#include<stdio.h>
int main()
{
    printf("%d %d %d",sizeof(3.14),sizeof(3.14f),sizeof(3.14L));
    return 0;
}
```

8. What will be output of the following program?

```
#include<stdio.h>
int main()
{
    int a;
    a=sizeof(!5.6);
    printf("%d",a);
    return 0;
}
```

9. Write a c program to find the grade of a student using operators.
10. Write a c program to perform calculator operation using operators.
11. Write a c program to find the roots of the Quadratic Equation.

12. Define Type conversion.
13. What are the different types of type conversion?
14. Write about implicit conversion with example.
15. Write about Type casting with example.
16. Explain in detail about Type conversion with example.
17. What are the formatted input\output statements?
18. Write the syntax for scanf() and printf() functions.
19. What are the unformatted input\output functions?
20. Write the difference between getc() and getchar(), putc() and putchar().
21. What is the use of gets() and puts().
22. Explain in detail about all the input\output statements with example.

Links for Reference

1. http://www.tutorialspoint.com/
2. http://a4academics.com/
3. http://www.programiz.com/c-programming