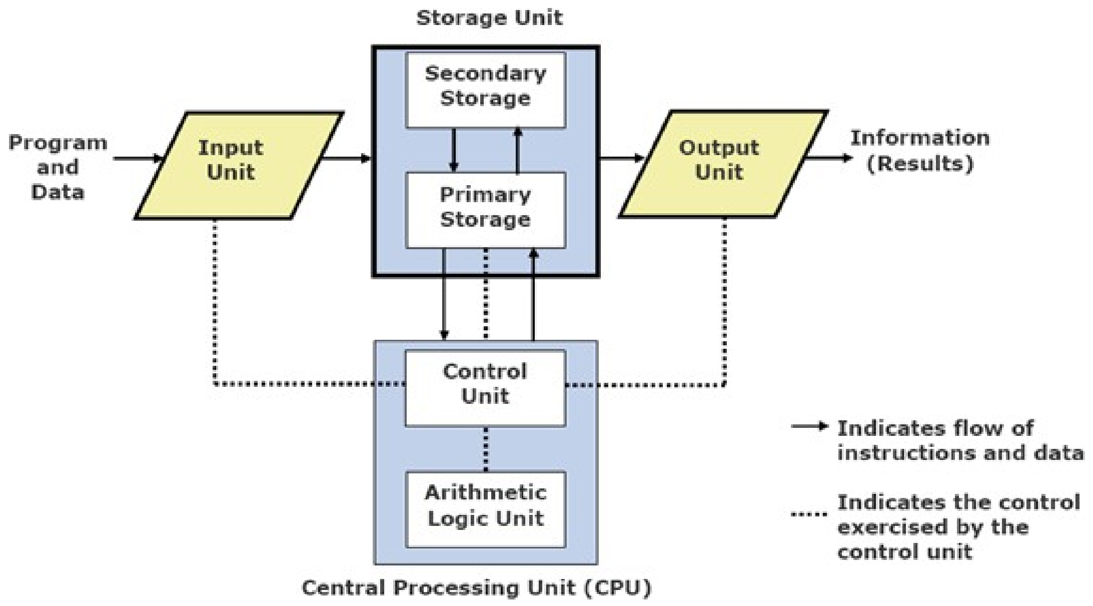
PPS QUESTION ANSWER

Que: 1 Explain basic block diagram of computer system.

Ans:



- **Input Unit (Input devices) :**
  - Input devices of a computer are used to feed information to the computer.
  - Example : keyboard , mouse , scanner , cd , floppy
- **Output Unit (Output devices):**
  - Output devices are used to display the data or result on the output screen or device.
  - Example : monitor , printer , cd , floppy
- **Storage Unit (Memory) :**
  - Computer needs memory to store the instructions and data. It also requires storing the program. (received from input devices)
  - Intermediate results of Processing
  - Final result of Processing, before they are released to output device.
  - **There are two categories of memory: *Primary* and *Secondary* memories.**
  - **Primary memory** is made from semiconductor devices is used for following
    - ▣ Use to hold running program instructions.
    - ▣ Used to hold data, intermediate results, and results of ongoing processing jobs
    - ▣ Fast in operation
    - ▣ Small Capacity

- ▣ Expensive
- ▣ Volatile (Looses data on power failure)

- Primary memory is further classified into two groups :
  1. RAM (Read/write memory)
  2. ROM (Read only memory)
- **Secondary memory** is made from magnetic material and used for following
  - ▣ Used to store program instructions
  - ▣ Used to hold data and information of stored jobs
  - ▣ Slower than Primary memory
  - ▣ Large Capacity
  - ▣ Lot Cheaper than Primary storage
  - ▣ Retains data even without power

- **Central Processing Unit :**
  - CPU is referred as a brain of computer, converts data into meaningful information.
  - A CPU controls all the internal and external devices, performs arithmetic and logical operations, and operates only on binary data (0 and 1).
  - CPU consist of main three subsystems :
    1. Arithmetic and Logical unit
    2. Control unit
    3. Registers
  - **ALU :**
    1. It performs all the arithmetic and logical operations of computer.
    2. It takes the data to perform operations from designated register.
  - **Control Unit :**
    1. This unit checks the correctness of sequence of operation.
    2. It fetches program instruction from the primary storage unit, interprets them, and ensures correct execution of program.

Que: 2 difference between:

A) Hardware and Software

B) Compiler and Interpreter

C) Application software and system software

Ans:

A) Hardware and Software

Hardware	Software
Devices that are required to store and execute (or run) the software.	Software is a program that enables a computer to perform a specific task.
Input,storage,processing,control,and output devices.	System software, Programming software, and Application software.
CD-ROM, monitor, printer, video card, scanners, label makers, routers , and modems.	Adobe Acrobat, Microsoft Word , Microsoft Excel
Hardware starts functioning once software is loaded.	To deliver its set of instructions, Software is installed on hardware.
Hardware failure is random. Hardware does have increasing failure at the last stage.	Software failure is systematic. Software does not have an increasing failure rate.
Hardware is physical in nature.	Software is logical in nature.

B) Compiler and Interpreter

Compiler	Interpreter
Checks the whole program at a time and then display the list of errors.	Checks the program line by line and stops checking whenever error occurs.
Converts whole source code into object code.	Converts source code into object code line by line.
After completion, source code is not required.	For, every execution of line, source code is required.
The object code of program generated when the program is error free.	It generates the object code for each line immediately if that line is error free.
The execution process of program is faster.	The execution process of program is faster.
Memory Requirement : More	Memory Requirement is Less

C) Application software and system software

System software	Application software
A system software runs the system	An application system runs over the system software.
A system software are programs that run & control the hardware units of the system	an application software doesn't.
System programs are written using dll, exe files for windows & rpm files for linux etc,	Application software are developed on the basis these files or by using different language files.
U can't create applications using system software	Application software are specially made to create applications for users.

Que: 3 Explain machine level, Assembly level and high level programming language.

Ans:

A) Machine language programming:

•It is written using binary language (0's and 1's).

•Computer can understand only low level language. So it is not required to convert it, so low level languages are faster.

•It is also known as Binary Language and Low Level language.

Advantages of Machine language programming:

- The programs are compact and small
- Occupied less memory
- No need to translate the program
- Suitable for low volume application
- Not require any translator to run (like assembler)

Disadvantages of Machine language programming:

- difficult to understand and debug
- required more time to write the program because it is written in bitform (0's and 1's)
- The programmer often makes errors, which are difficult to find
- Not suitable for large application
- cannot describe because it is written in bits
- Not portable (Machine dependent)

B) Assembly language:

- Written using set of instructions (mnemonics).
- It corresponds to symbolic instruction and executable machine codes and was created to use letters instead of 0s and 1s to run a machine. Ex ADD A,B for Addition.

Advantages of Assembly language:

- Easy to describe because of instructions.
- Very easy to understand or debug.
- Required less time to write the program.

Disadvantages of Assembly language:

- Required assembler to translate assembly language into machine language.
- Programmer needs to learn the structure of assembly language and syntax of every statement.

- Not portable (Machine dependent)

C) high level programming

- High-level languages are similar to English language.
- Programs written using these languages can be machine dependent.
- A single high-level statement can substitute several instructions in machine or assembly language.

Advantages:

i. Readability
ii. Machine dependent
iii. Easy debugging
iv. Easier to maintain
v. Low development cost

Disadvantages:

i. Poor control on hardware
ii. Less efficient

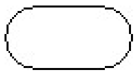
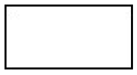
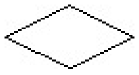
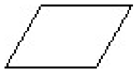
Que: 4 what is Flow chart and algorithm? Explain with any one example.

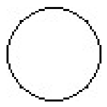
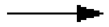
ANS: Flow Chart:

A flowchart is a graphical representation of algorithm or program.

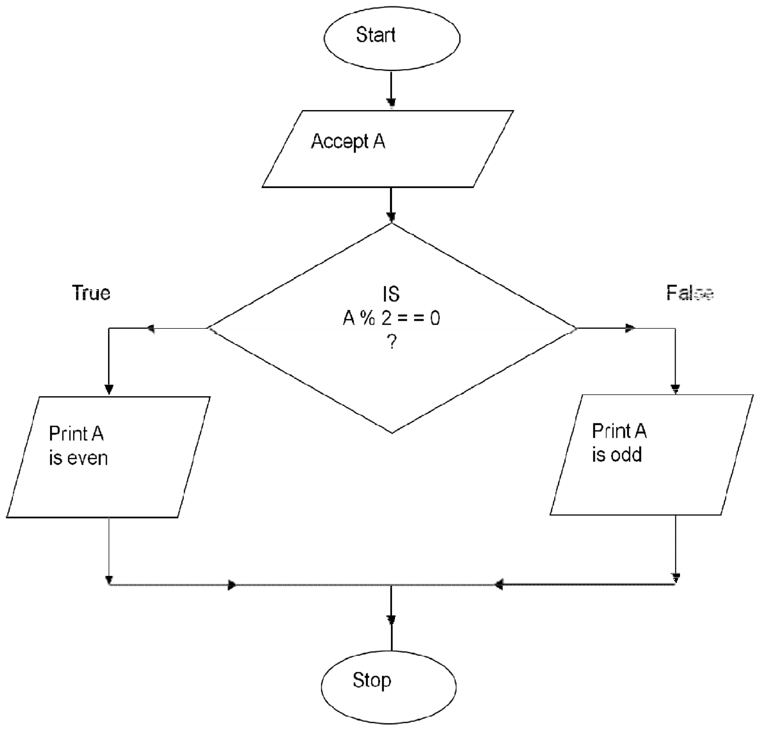
•It represents the sequence of operation to be performed to solve given task.

•Different symbols are used to represent an algorithm.

	**Start/End** The terminator symbol marks the starting or ending point of the system. It usually contains the word "Start" or "End."
	**Action or Process** A box can represent a single step ("add two cups of flour"), or and entire sub-process ("make bread") within a larger process.
	**Decision** A decision or branching point. Lines representing different decisions emerge from different points of the diamond.
	**Input/Output** Represents material or information entering or leaving the system, such as customer order (input) or a product (output).

	**Connector** Indicates that the flow continues where a matching symbol (containing the same letter) has been placed.
	**Flow Line** Lines indicate the sequence of steps and the direction of flow.

<u>Example:</u> W.A.P. to accept any number and find out whether it is even or odd ?.



Algorithm:

An algorithm is set of sequential instructions to solve a specific task. And It generates step by step solution of the problem. After a finite number of steps, solution of problem is achieved.

Advantages:

- Very easy to write.
- Easy to understand.
- Easy to detect any mistakes.

Disadvantages:

- Time consuming.
- Difficult to show branching and looping.
- Helpful for only small application.

Example: Write a algorithm to find out number is odd or even?

```
step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
            print "number even"
         else
            print "number odd"
         endif
step 5 : stop
```

Que: 5 write the importance of C language.

- It is a robust language.
- It has rich set of built-in functions and operators which can be used to write any complex programs.
- C combines the capabilities of high-level language and assembly language so is suitable for writing both system software's and application software's.
- C programs are efficient and fast because of variety of data types and operators.
- C is highly portable. C program written on one computer can be run on another with little or no modification.
- C language is well suited for structured programming which makes program debugging, testing and maintenance easier.
- C has ability to extend itself. We can continuously add our own functions in C library.

Que: 6 Explain Basic Structure of C Program.

- C program can be viewed as a group of building blocks called functions.
- A function is a collection of one or more statements that perform a specific task.
- A C program may contain one or more section as shown below:

Documentation Section
Link Section
Definition Section
Global Declaration Section

```
main()
{
                    Declaration part;
                    Executable part;
}
---------------------------------------------
Sub-program Section
                    function1()     User-
                    function2()    Defined
                    function3()    Functions
                    ….
```

Documentation Section:

- It contains set of comments lines. Documentation section will not give any effect at a runtime.
- It gives information like author of the program, time of creation of program, functionality of the program, etc.
- Comments lines are ignored by the complier and do not increase the program execution time.
- Comments can be given using // for single line and /* ….*/ for multiple lines.

Link Section:

- It consists of instructions to be given to the compiler to link functions from the C library to the C program.
- It is done using #include derivative.
- #include<Stdio.h> and #include<conio.h> are necessary for basic built in function.
- For example, if we want to use some mathematical functions then we have to include math.h file in link section as shown below.

  # include<math.h>

Definition Section:

- This section defines all the symbolic constants.
- It is done using #define derivative. Value of a symbolic constant remain constant throughout application (Program)

- By defining symbolic constant one can use these symbolic constant instead of constant value.

  # define PI 3.14 and # define TEMP 35

Global Declaration Section:

- It contains the declaration of variables which are used by more than one function known as global variables.
- It is also used for declaring user-defined functions.
- Variable declare in this section can be used by any available function in program.
- It is Return before void main() and after the link section.

Main Function Section:

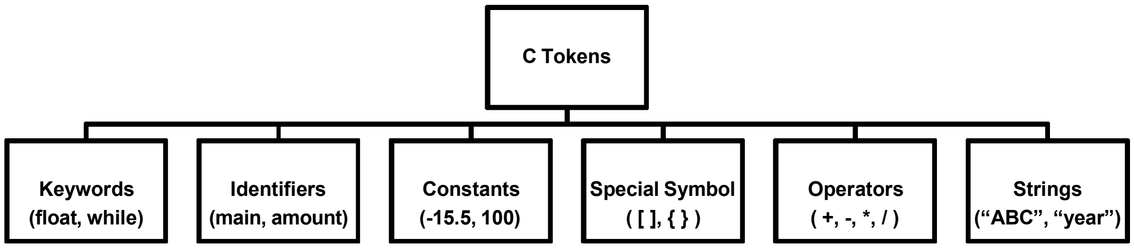
- Every C program must have one main( ) function. It's heart of c program.
- main() contains two parts, declaration part and executable part.
- There is at least one statement in executable part.
- These two parts must appear between the opening { and closing braces }.
- Closing brace of main function section is the logical end of the program.
- All statements in the declaration and executable parts end with a semicolon (;).

Sub-Program or Sub-Function Section:

- It contains all user-defined functions.

Que: 7 Write a short note on C tokens.

- In a passage of text, individual words and punctuation marks are called tokens.
- In C program smallest individual units are known as C tokens.
- C has 6 types of tokens.



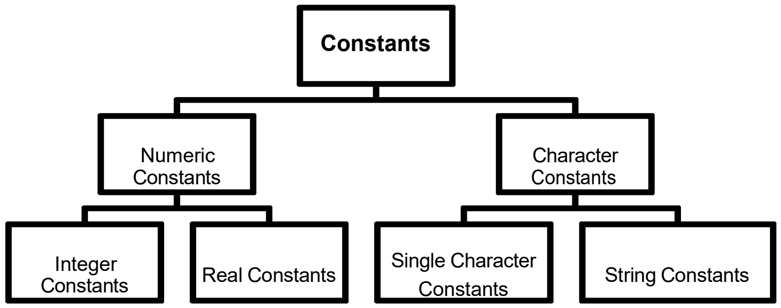
Keywords and Identifiers

- Every C word is classified as either a keyword or identifiers.
- Keywords have fixed meanings cannot be changed.
- Eg. if, else, break, char, int, float.
- Identifiers are used to give names of variables, functions and arrays.

- They are user defined names made up of sequence of letters and digits.
- First character is always a letter (underscore is also allowed).

Constants

- Constants have fixed values and do not change during execution of program.
- There are two different types of constants; numeric constants and character constants.



Integer Constants

- It refers to a sequence of digits.
- Three types of integer's viz. decimal integer, octal integer & hexadecimal integer.
- E.g. decimal integers - +78, 123, -981.
- E.g. octal integers – 037, 0, 0343, 0667.
- E.g. hexadecimal integers – 0xbcd, 0x2, 0x.

Real Constants

- Certain quantities are represented by nos. containing fractional parts are called real constants.
- E.g. – 78.67, 12.45, -981e8

Single Character Constants

- It contains a single character enclosed within a pair of single quote marks.
- Eg. – 'd', '5', 'A'

String Constants

- It is a sequence of characters enclosed in double quotes.
- E.g. – "India", "12345", "A"

Operators

- C supports total 8 different types of operator's viz. arithmetic, logical, relational, assignment, increment and decrement, conditional, bit wise and special operators.

Special Symbols

- C supports different special symbols.

- E.g. <, >, $, %, {,}, [,].

Que: 8 Explain variables. Write down rules for declaring a variable in C language. How we can declare a variable?

- A variable is a data type that may be used to store a data value.
- Value of constant remains unchanged during the execution of a program.
- Variable may take different values at different times during execution of a program.

Rules for declaring Variables:

- Variables name may consists of letters, digits, and the underscore character, subject to the following conditions:
- They must begin with a letter. Some systems permit underscore as the first character.
- ANSI standard recognizes a length of 31 characters. However, length should not be normally more than 8 characters, since only first eight characters are treated as significant by many compilers.
- Uppercase and lowercase are significant. That is, the variable Total is not same as total or TOTAL.
- It should not be a keyword.
- White space is not allowed.

Declaration of Variables:

- It tells compiler what the variable name is.
- It specifies what type of data the variable hold.
- We can declare variable as follows

  data-type v1,v2,…. vn ;

  where v1, v2, ……. vn are the name of variables.
- Variables are separated by commas.
- Declaration statement must end with a semicolon (;).

Examples:

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

Assigning Values to Variables:

- Values can be assigned to variables using the assignment operator (=) as follows :

  variable_name = constant;

Examples:

```
initial_value = 0;
balance = 54.45;
        yes = 'x';
```

- C allows multiple assignments in one line.

Examples:

```
initial_value =0; final_value = 100;
```

- It is also possible to assign a value to a variable at the time the variable is declared.

```
data_type variable_name = constant;
```

Examples:

```
int final_value = 100;
char yes = 'x';
```

Que: 9 Explain Symbolic Constants. What are the advantages of using it?

- Symbolic Constants are defined in the Definition Section.

Defining Symbolic Constants

- Certain unique constants are used now and then in the program and may appear repeatedly in a number of places.
- For example the constant “pi”, total no. of students whose mark-sheets are to be prepared, etc….
- We face 2 problems in the subsequent use of such in programs.
  - Problem in modification of the program.
  - Problem in understanding the program.
- This problem can be solved using #define derivative.

```
#define symbolic_name value of constant
```

Examples:

```
#define ROI 10
#define TOT_STUDENTS 60
#define PI 3.14
```

Examples:

```
Area = PI * radius * radius;
Simple_Int = ROI * principal * years;
```

Que: 10 Write a short note on Primary Data Types / Fundamental Data Types of C.

- All C compilers support 5 fundamental data types, namely integer (int), character (char), floating point (float), double-precision floating point (double) and void.

Data type	**Range**
char	-128 to 127
int	-32,768 to 32,767
float	3.4e-38 to 3.4e+38
double	1.7e-308 to 3.4e+308

Integer Types

- Integers are whole numbers with a range of values supported by a particular machine.
- Generally, integers occupy one word of storage.
- For 16-bit word length, the size of the integer is -32768 (-2^{15}) to 32367 ($+2^{15}-1$) and for 32-bit word length its range will be (-2^{31}) to ($-2^{31}-1$). First bit is used to indicate the sign (positive or negative) of the stored integer value.
- C has three classes of internal storage, namely **short int, int, and long int,** in both **signed and unsigned** forms.
- Short int represents fairly requires half the amount of storage as a regular int number.
- Unsigned integers are always positive. So the range for int on a 16-bit machine will be from 0 to 65,535 ($2^{31}-1$).
- Long and unsigned integers are declared to increase the range of values.
- Qualifier signed is optional because the default declaration assumes a signed number.

Floating Point & Double Types

- Floating point (or real) numbers are stored in 32 bits (on all machines), with 6 digits of precision.
- If float data type is not sufficient to hold the data then double data type is used which uses 64-bits giving a precision of 14 digits.
- Long double which uses 80-bits.

Character Types

- A single character can be defined as a character (char) type data.
- They occupy 1 byte (8 bits) of internal storage.
- Unsigned chars have values between 0 and 255; signed chars have values from -128 to 127.

Void Types

- The void type has no values.
- This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.
- It can also play a role of a generic type, meaning that it can represent any of the other standard types.

Que: 11 Define operators. Explain all operators in C language. (Operators can be individually asked or any 3-4 can be asked in group)

Definition:

- An operator is a symbol that tells the computer to perform certain mathematical or logical operations.
- Operators are used in programs to perform some operations on data and variables.

Arithmetic Operators

Operator	**Meaning**
+	Addition or unary plus
-	Subtraction or unary minus
*****	Multiplication
/	Division
%	Modulo division

Examples:

a – b	a + b
a * b	a / b
a %b	-a * b

- Modulo division operation produces the reminder of an integer division.
- Modulo division operator % cannot be used on floating point data (real operands).
- C does not have an operator for exponentiation.

Relational Operators

Operator	**Meaning**
<	is less than
<=	is less than or equal to
>	is greater than

>=	is greater than or equal to
==	equal to
!=	is not equal to

- The value of a relational expression is either one or zero.

Syntax:

ae-1 relational operator ae-2

- ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them.

Examples:

4.5 <= 10 TRUE

4.5 <= -10 FALSE

10 7 + 5 TRUE

- a + b = c + d TRUE only if the sum of values of a and b is equal to the sum of values of c and d.
- Relational expressions are used in decision statements such as if and while.

Logical Operators

Operator	**Meaning**
&&	logical AND
| |	logical OR
!	logical NOT

- The logical operators && and || are used when we want to test more than one condition and make decisions.

Example:

a > b && x = = 10

Logical expression produces a value zero or one according to following truth table.

op-1	**op-2**	**op-1 && op-2**	**op-1 || op-2**
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Assignment Operators

- Assignment operators "=" is used to assign the result of an expression to a variable.
- 'Shorthand' assignment operator is also used in C.

Syntax:

v op = exp;

- *v* is a variable, *exp* is an expression and *op* is a operator.
- The assignment statement

  v op = exp; is equivalent to **v = v op (exp);**

- Assignment operators have three advantages:
  1. Easier to write as repetition of variable *v* on the right hand side is not required.
  2. The statement is more brief and easier to read.
  3. The statement is more efficient.

Example:

```
Value = Value + delta;
```

- Can be written as

```
Value + = delta;
```

Increment and Decrement Operators

- **++** and **--** are the increment and decrement operators.
- ++ adds 1 to the operand, while -- subtracts 1.
- ++m or m++; is equivalent to m = m +1 (or m += 1).
- --m or m--; is equivalent to m = m -1 (or m -= 1).
- m++ and ++m means the same when they form statements independently.
- They behave differently when they are used in expressions on the right-hand side assignment statement.

Example:

```
m = 5;
y = ++m;
```

- Value of y will be 6 and that of m will also be 6.
- But if,

```
m = 5;
y = m++;
```

- In this case the value of y will be 5 and that of m will be 6..

Conditional Operator

- “? :” operator is known as conditional operator in C.
- It is also known as ternary operator.

Syntax:

exp1 ? exp2 : exp3

- where exp1, exp2, exp3 are expressions.
- exp1 is evaluated first. If it is non-zero (true), then the expression exp2 is evaluated and becomes the value of the expression.
- if exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

Example:

```
a = 10;
b = 15;
x = (a > b) ? a : b; here x will be assigned with the value of b.
```

- This can be achieved using if…else statements as:

```
if (a > b)
        x = a;
else
        x = b;
```

Bitwise Operators

Operator	**Meaning**
&	bitwise **AND**
|	bitwise **OR**
^	bitwise exclusive **OR**
<<	shift left
>>	shift right

- Bitwise operators are used for manipulation of data at bit level.
- They may not be applied to float or double.

Special Operators

- C supports special operators like *comma* operator, *sizeof* operator, *pointer* operators (& and *) and *member selection operators* (. and →).

- **Comma Operator**

- It can be used to link the related expressions together.
- Expression is evaluated left to right.
- Value of right-most expression is the value of the combined expression.

Example:

```
value = (x = 10, y = 5, x + y);
```

- First assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 to value.
- Comma operator has the lowest precedence thus, brackets () are necessary.
- It is used in for and while loops and exchanging values.

Examples:

```
for (i=0, j=1; i<5; i++, j=j+2)
while (c = getchar(), c != '$')
t = x, x = y, y = t;
```

- **Sizeof Operator**

- It is a compile time operator.
- It returns the number of bytes occupied by the operand.

Example:

```
m = sizeof (sum); gives 2 if sum in of int type.
n = sizeof (long int); assigns 4 to n.
k = sizeof (235L); assigns 10 to k.
```

- It is normally used to determine the lengths of arrays and structures.
- It is also used to allocate memory space dynamically to variables.

Que: 12 what is type casting / type conversion? Explain type casting / type conversion with suitable example.

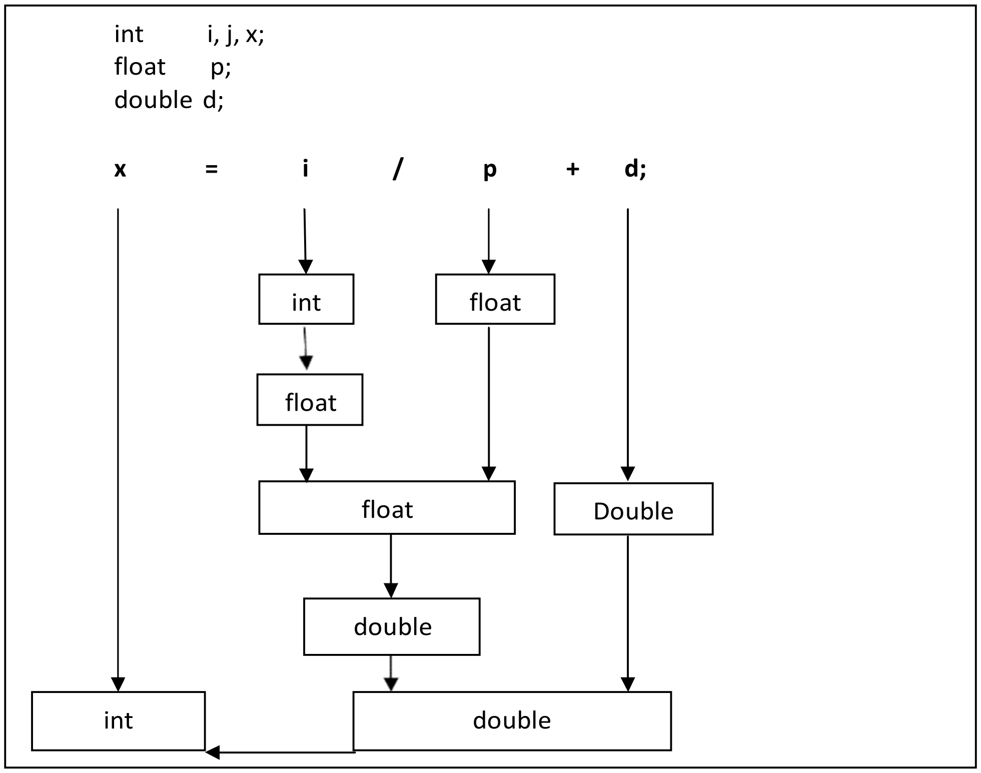
Definition:

- In C language, expression can be evaluated if variables and/or constants are of same type.
- If they are of different type, they need to be converted to other types so that they can be evaluated without losing any meaning.
- This is known as type casting / type conversion in C.
- There are two methods for type conversion: Implicit and Explicit type conversion.

Implicit Type Conversion

- C *automatically* converts any intermediate values to the proper type so that the expression can be evaluated without losing any meaning. This automatic conversion is known as implicit type conversion.
- Operand of ‘lower’ type is converted to the ‘higher’ type before it is evaluated.
- The result is of higher type.

Example:



```
int     i, j, x;
float   p;
double  d;

x   =   i   /   p   +   d;
```

- In above example, the final result of an expression is converted to the type integer before assigning the value to it.

Explicit Type Conversion

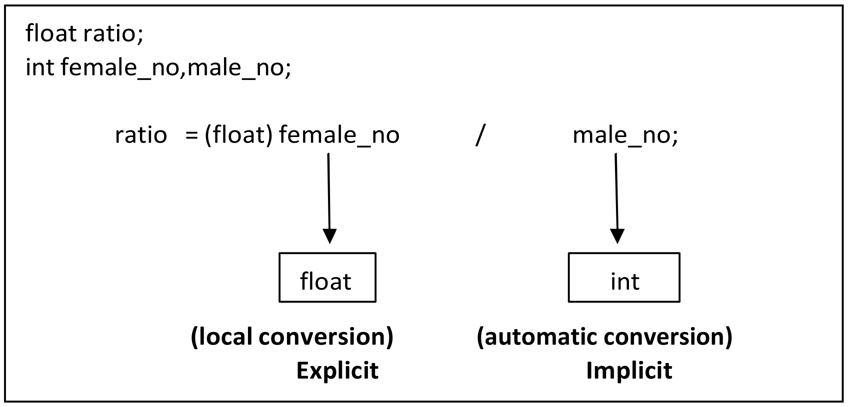
- Sometimes we want to force conversion in a way that the result obtained is of the desire type.
- For example, we want to calculate ratio of females to male in a town.

  ratio = female_no / male_no

- Since female_no and male_no are integer values the result; ratio would represent a wrong figure.

- This problem can be solved by converting locally one of the variables to floating point as follows:

  ratio = (float) female_no / male_no



```
float ratio;
int female_no,male_no;

        ratio  = (float) female_no        /        male_no;
```

- The operator (float) converts female_no to floating point, then by automatic conversion the division is performed in floating point mode and the result; ratio will be a float value.
- Value of variable female_no remains as int in other parts of the program.

Que: 13 What is Operator Precedence and Associativity? Explain with suitable example.

Operator Precedence and Associativity

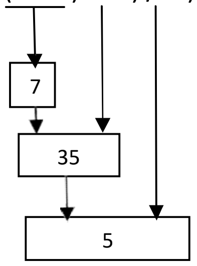
- Precedence is used to determine how an expression involving more than one operator is evaluated.
- The operators of higher level are evaluated first.
- Associativity is used to determine whether operators of the same precedence are evaluated either from 'left to right' or from 'right to left'; depending on the level.
- The Operators with the Precedence and Associativity are given in the table below:

Operartor	**Description**	**Associativity**	**Rank**
( )	Function call	Left to Right	1
[ ]	Array element reference		
+	Unary plus	Right to Left	2
-	Unary minus		
+ +	Increment		
- -	Decrement		
!	Logical negation		
~	Ones complement		

*	Pointer reference (indirection)		
&	Address		
sizeof	Size of an object		
(type)	Type cast (conversion)		
*	Multiplication	Left to Right	3
/	Division		
%	Modulus		
+	Addition	Left to Right	4
-	Subtraction		
<<	Left shift	Left to Right	5
>>	Right shift		
<	Less than	Left to Right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
= =	Equality	Left to Right	7
! =	Inequality		
&	Bitwise AND	Left to Right	8
^	Bitwise XOR	Left to Right	9
|	Bitwise OR	Left to Right	10
&&	Logical AND	Left to Right	11
| |	Logical OR	Left to Right	12
? :	Conditional expression	Right to Left	13
=	Assignment operators	Right to Left	14
*= /= %=			
+= -= &=			
^= |=			
,	Comma operator	Left to Right	15

Example:

(((3 + 4) * 5) / 6)   // generates 5 as result.



- In step 1: 3+4 is evaluated, result will be 7.
- In step 2: 7 * 5 is evaluated, result will be 35
- In step 3: 35 / 6 is evaluated, final result will be 5.

Que:14 Explain if statement with suitable example.

Ans:

Syntax:

```
if (test expression)
{
        statement-block;
}
statement-x;
```

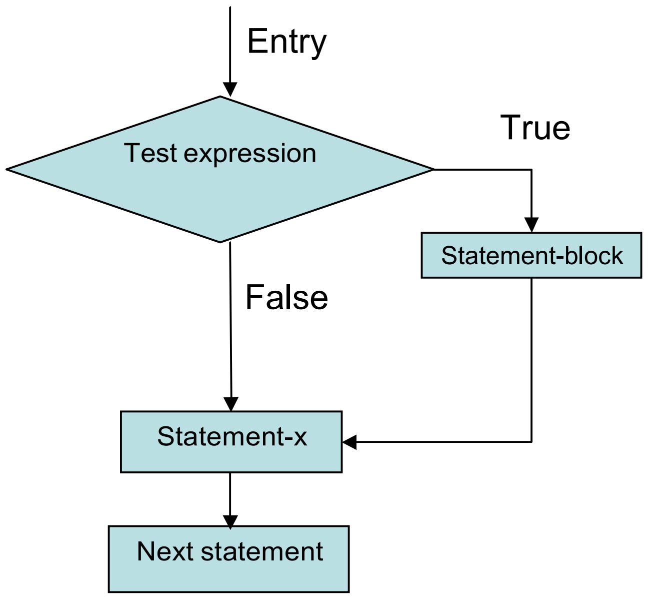
- 'Statement-block' may be single or a group or statements.
- If the test condition is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x.

Example:

```
if (category = = SPORTS)
{
      marks = marks + bonus_marks;
}
printf ("%f ", marks);
```

- The program tests the type of category of the students.
- If the student belongs to the SPORTS category, then additional bonus_marks are added to his marks before they are printed.

Flowchart:



Que: 15 Explain if….else statement with suitable example.

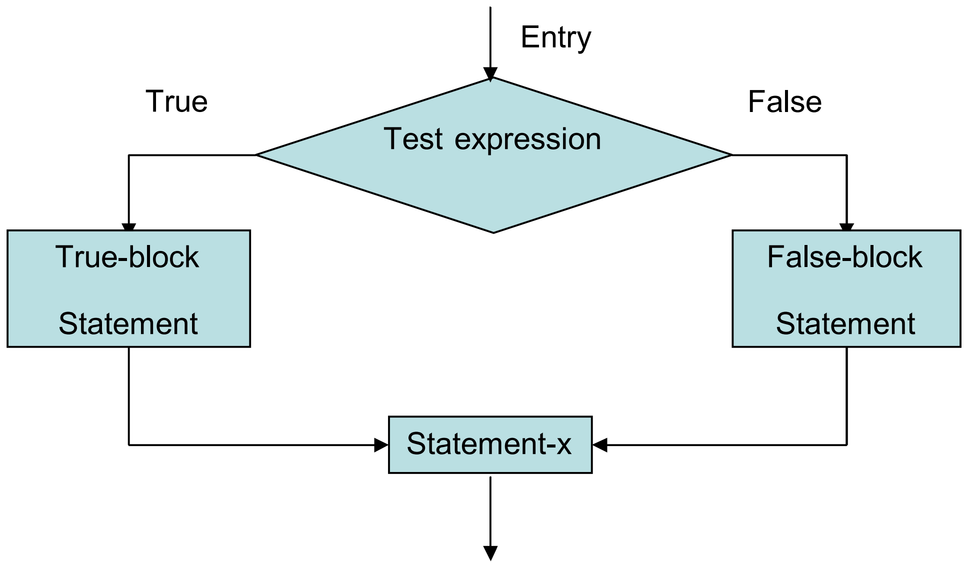
Ans:

Syntax:

```
if (test expression)
{
                True-block statement (s);
}
else
{
                False-block statement (s);
}
statement-x;
```

- If the test expression is true, then the true-block statement(s), immediately following the if statements are executed, otherwise, the false-block will be executed, not both.

- Flowchart:



Example:

```
if (code == 1)
                boy = boy + 1;
          else
                girl = girl + 1;
          statement-x;
```

- The program determines whether or not the student is a boy.
- If code is equal to 1, then the statement boy = boy + 1; is executed and the control is transferred to the statement-x.
- If code is not equal to 1 then the statement of else part girl = girl + 1; is executed before the control reaches the statement-x.

Que: 16. Explain nested if….else statement with suitable example.

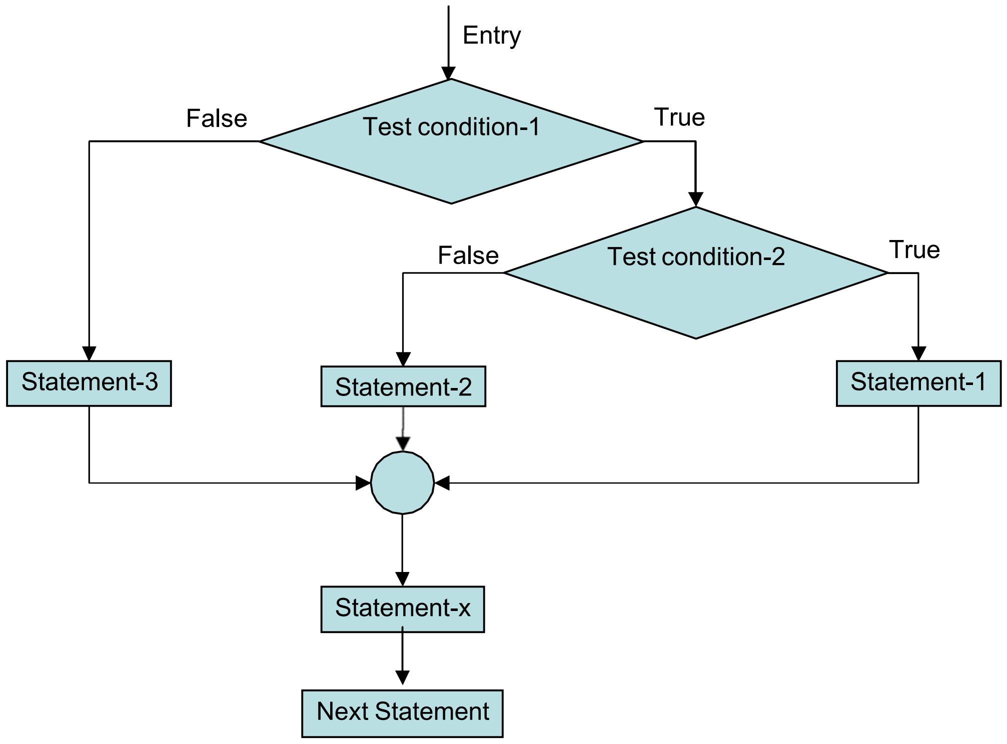
Ans:

Syntax:

```
if (test condition-1)
      {
            if (test condition-2)
            {
                  statement-1;
            }
            else
            {
                  statement-2;
            }
```

```
                }
                else
                {
                        statement-3;
                }
        statement-x;
```

- If condition-1 is false, then statement-3 will be executed; otherwise it continues to perform the second test.
- If the condition-2 is true, then statement-1 will be evaluated; otherwise statement-2 will be evaluated and then the control is transferred to the statement-x.



Example:

```
if (No = = 1)

{

        if (balance > 5000)

                bonus = 0.05 * balance;

        else

                bonus = 0.03 * balance;

}

else

{

        bonus = 0.02 * balance;
}
balance = balance + bonus;
```

- If No is equal to 1 and bank balance > 5000, then 5 % bonus is given to the employee and if balance < 5000 then 3% bonus is given.
- If No is not equal to 1 then irrespective to the balance 2% bonus is given and then the final balance is calculated.

Que: 17. Explain else if ladder with suitable example.

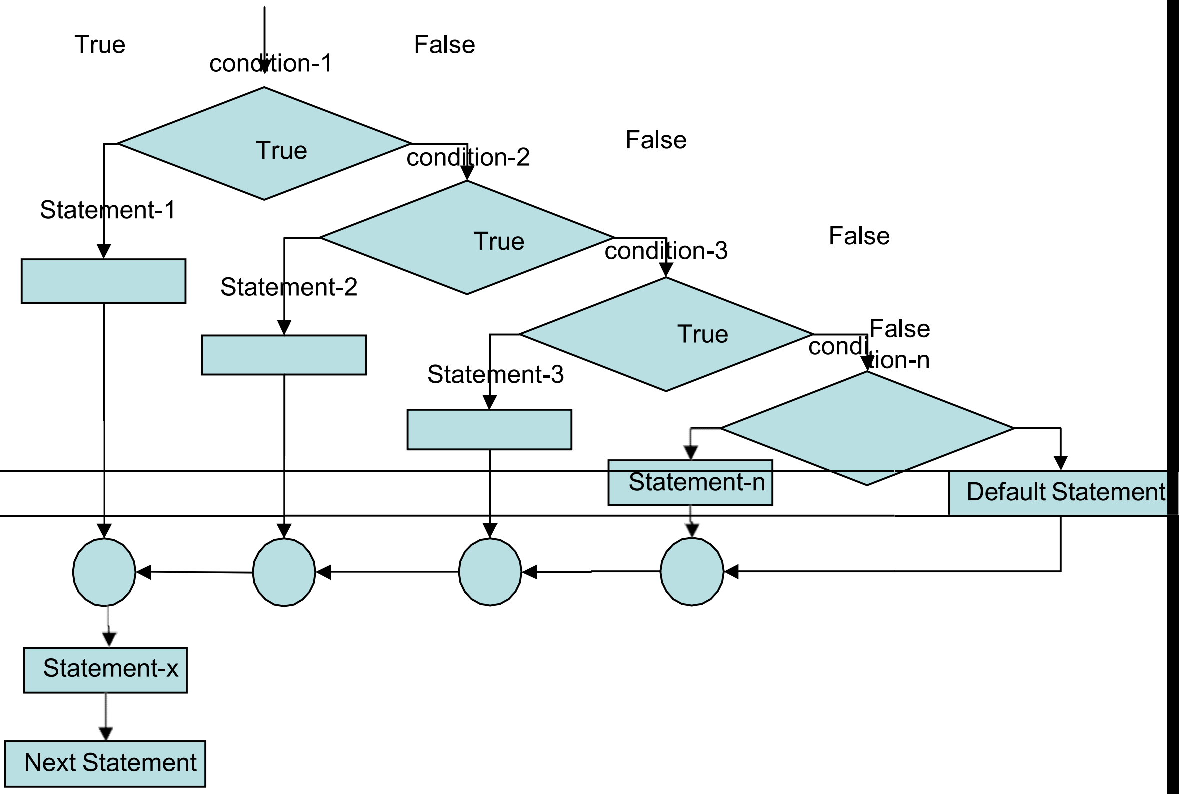
Ans:

Syntax:

```
if (condition-1)
        statement-1;
else if (condition-2)
        statement-2;
else if (condition-3)
        statement-3;
else if (condition-n)
        statement-n;
else
        default-statement;
statement-x;
```

- Conditions are evaluated from the top to bottom of the ladder.
- As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x skipping rest of the ladder.
- When all n conditions are false, then the final else containing default-statement will be executed.

Flowchart:



Example:

```
if (marks > 79)
        grade = "Honors";
else if (marks > 59)
        grade = "First";
else if ( marks > 49)
        grade = "Second";
else if (marks > 39)
        grade = "Third";
else
        grade = "Fail";
printf ("%s\n", grade);
```

- Above example calculates the grade of the student depending on the marks obtained by him.

Que: 18. Explain switch-case statement with suitable example.

Ans:

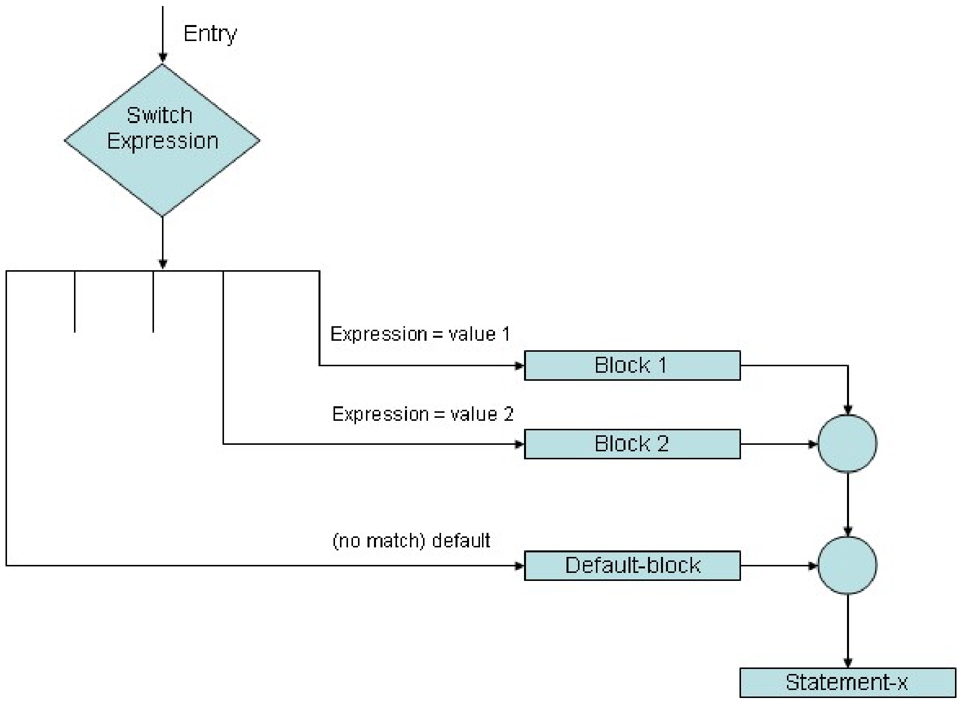
Syntax:

```
switch (expression)
        {
                case value-1:
                        block-1
                        break;
                case value-2:
                        block-2
                        break;
                ………..
                ………..
                defalult:
                        default-block
                        break;
        }
statement-x;
```

- Switch statement tests the value of given variable or expression against a list of case values.
- When a match is found, a block of statements associated with that case is executed.
- Expression is an integer expression or character expression.
- Value-1, value-2…value-n are constants or constant expressions and are known as case labels.
- Case labels should be unique.
- block-1, block-2….block-n are statements lists and may contain zero or more statements.
- Each case labels ends with a colon (:).
- The value of expression is successfully compared against the values value-1, value-2 …… If a case is found whose value matches with the value of the expression, then the block of statements associated with it is executed.
- The break statement at the end of each block signal's the end of a particular case and causes an exit from the switch statement.
- The default is an optional case.

- If it is present, it will be executed if the value of expression does not match with any of the case values.

Flowchart:



Example:

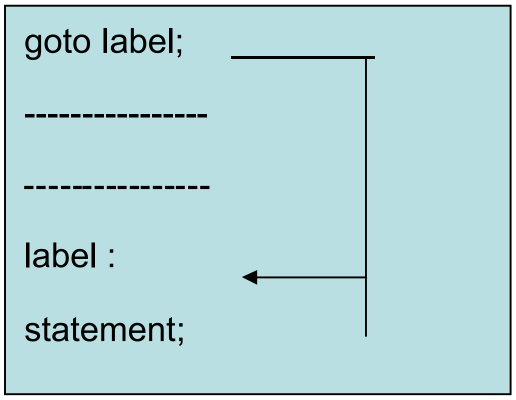
```
Printf("Enter two values a and b");
Scanf("%d %d",&a,&b);
Printf("Enter the choice 1 for + , 2 for -, 3 for *, 4 for / :");
Scanf("%d",&ch);
switch (ch)
{
        case 1:
                printf("Your addition is : %d",a+b);
                break;
        case 2:
                printf("Your substraction is : %d",a-b);
                break;
case 3:
                printf("Your multiplication is : %d",a*b);
                break;
case 4:
                printf("Your divison is : %d",a/b);
                break;
defalult:
                printf("Invalid choice");
}
```

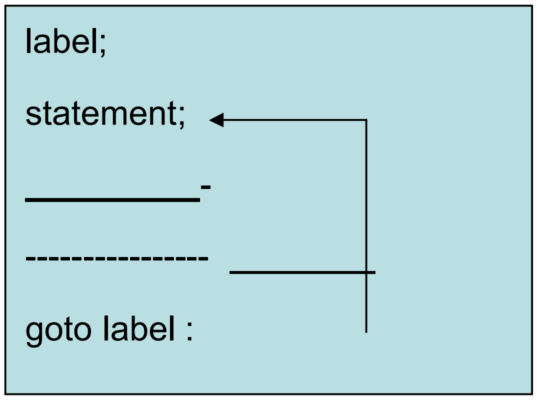
Above example calculates the arithmetic operation of two values.

Que: 19. Explain goto statement with suitable example.

Ans:

- Goto statement is used to branch unconditionally from one point to another in a program.
- It requires a label in order to identify the place where the branch is to be made.
- Label is a valid variable name, and must be followed by a colon.





FORWARD JUMP **BACKWARD JUMP**

- goto breaks the normal sequential execution of the program.
- If the label: is placed before the statement goto label ; a loop will be formed and some statements will be executed repeatedly and is known as backward jump.
- If the label : is placed after the goto label ; then some statements will be skipped and the jump is known as forward jump.

Example:

```
main()
{
        double x, y;
        read :
                scanf ("%lf ", &x);
        if ( x < 0)
                goto read;
        y = sqrt (x);
        printf ("%f %f ", x, y);
`       }
```

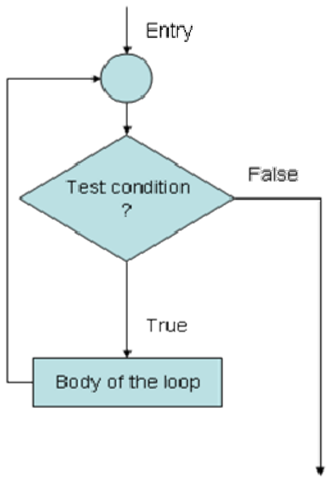
- This program will find the square root of a number if it is positive otherwise it will continue to scan a new number until a positive number is entered.

Que: 20. Explain the difference between Entry Control Loop and Exit Control Loop.

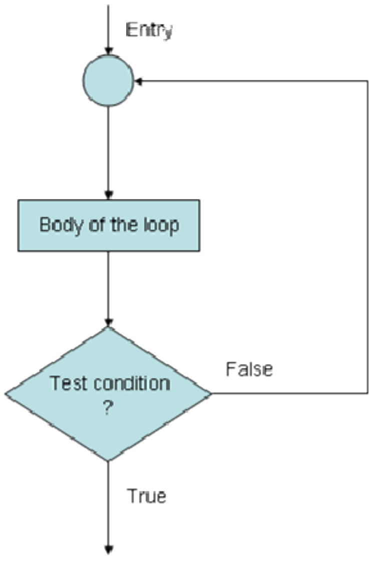
Ans: Entry Control Loop

- In entry control loop, the control conditions are tested at the start of the loop execution.
- If the conditions are not satisfied then body of the loop is not executed.
- Also known as pre-test loop.

Example: While statement, for statement.



Entry Controlled Loop



Exit Controlled Loop

Exit Control Loop

- Control conditions are tested at the end of the body of the loop.
- Body of the loop is executed unconditionally for the first time.
- Also known as post-test loop.
- **Example**: do statement.

Que: 21. Explain while loop with suitable example.

Ans:

Syntax:

```
while (test condition)
{
```

```
                    body of the loop
            }
```

- while is entry-controlled loop.
- Test-condition is checked first, and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test-condition is once again checked and if it is true, the body is executed once again.
- This process is repeated until test-condition becomes false.

Example:

```
sum = 0;
n = 1;
while ( n <= 10)
{
sum = sum + n ;
n = n + 1;
}
printf ("sum = %d ", sum);
-...............-................
```

- The program will compute the sum of all number between 1 and 10 i.e. 1 + 2 + ….. + 10.
- The loop will continue 10 times and will terminate when condition (11<=10) becomes false.

Que: 22. Explain do while loop with suitable example.

Ans:

Syntax:

```
            do
            {
                    body of the loop
            } while (test condition);
```

- do is exit-controlled loop.
- Body of the loop is evaluated first and then condition in while statement is evaluated.
- If the condition is true, the body is executed once again.

- This process is repeated as long as the test-condition is true.
- Body of the loop is evaluated at least once.

Example:

```
do
 {
            printf (“Enter a number”);
            scanf (“%d “, &no);
            sum += no;
} while ( no > 0);
printf(“%d”,sum);
```

- The program computes the sum of all positive number entered by the user.
- Loop terminates when a negative number is entered by the user.

Que: 23. Explain for loop with suitable example.

Ans:

Syntax:

```
for (initialization ; test-condition ; increment)
{
            body of loop
}
```

- for is entry-controlled loop.
- Initialization of the control variables is done first, using assignment statements like a = 1 and count = 0.
- The value of the control variable is tested using the test-condition. It is a relational expression like (i<10) or (count < 100) and determines when loop will exit.
- If the condition is true, then body of the loop is executed, otherwise loop is terminated.
- After executing body of the loop the control variable is incremented using assignment statements like (i=i + 1).
- The new value of control variable is tested again; body of the loop is executed until this value becomes false.

Example:

```
for ( x = 9; x >= 0; x = x -1)
{
                        sum += x;
}
printf ("%d \n", sum);
```

- The loop will compute the sum of the series 9 + 8 + 7 + ….. + 0.
- The loop will terminate when x becomes negative.

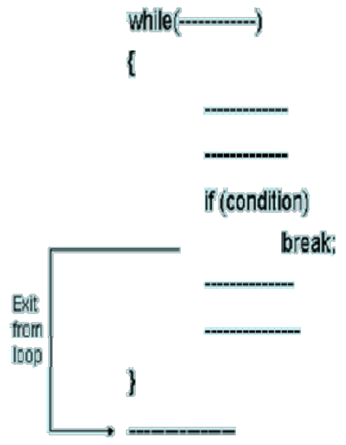
Que: 24. Explain break and continue statement with suitable example.

Ans:

- Jumping out of a loop can be done using break statement or goto statement.

Break Statement :

- When a break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.
- The effect of break statement in different looping structure is as shown in the figure given below:

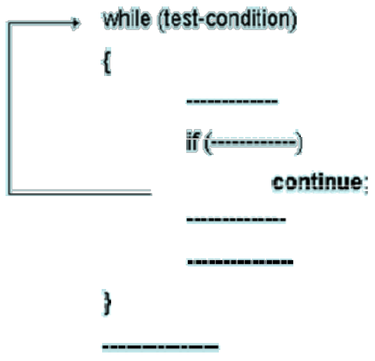


Continue statement:

- Continue statement causes the loop to be continued with the next iteration after skipping any statements in between.
- It tells the compiler to "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".
- **Syntax:**

continue;

- In while and do loops, continue causes the control to go directly to the test-condition and then continue the iteration process.
- In case of for loop, the increment section of the loop is executed before the test-condition is evaluated.
-
-



```
while (test-condition)
{
        ------------
        if (------------)
                continue;
        -------------
        --------------
}
-----------
```

Example:

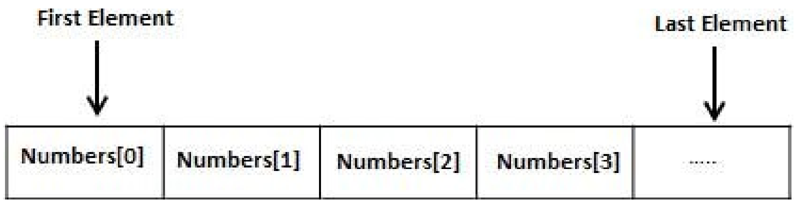
```
for(i = 0; i <= 100; i++)
        {
                if(x %2 = = 0)
                {
                        continue;
                }
                sum = sum + i;
        }
   printf("%d",sum);
```

- Above example calculates the sum of all odd positive numbers between 1 and 100.
- Loop will continue with next iteration if even number is encountered.

Que: 25 What is Array? Explain types of array and initialization of array.

Ans:

- Array is a collection of the elements of the same data type. We can store any number of elements in array variable.
- Array variable will start storing the value from 0 address.

▪ 

Types of array:

- There are basically three types of array in C.
  - One dimensional array (1-D)
  - Two dimensional array (2-D)
  - Multi dimensional array (3-D)

Declaring 1-D Arrays:

- To declare a 1-D array in C, we specifies the type of the elements and the number of elements required by an array as follows:

Syntax :

- Type variable[size];

Example :

- Int a[10];
- Here 'a' is a variable of type integer and can hold 10 integer value.

Declaring 2-D Arrays:

- To declare 2-D array in C, we specifies the type of the elements and the number of elements in the form of row and column.

Syntax:

Type variable[row_size][column_size];

Example:

Int a[3][3];

- Here 'a' is a variable of type integer and can hold 9 integer value in row and column wise.

Declaring Multi-D Arrays:

- To declare multi-D array in C, we specifies the type of the elements and the any number of size as follow :

Syntax:

Type variable[size1][size2][size3]…[size-n];

Example:

Int a[3][2][4];

- Here ‘a’ is a variable of type integer and can hold 24 integer value.

Initializing Arrays

- You can initialize array in C either one by one or using a single statement as follows:
- Int a[5]={5,10,12,15,25};
- The number of values between braces { } cannot be larger than the number of elements that we declare for the array.
- For, 2-D array the initialization will be as follows :
- Int a[2][2]={{5,8},{9,4}};

Que: 26 What is string? Explain string function.

Ans: String is collection of char data type or it is an array of char data type terminated by a special null character value '\0’.

Syntax for declaring string variable:

datatypestring_name[size];

Initialization of string:

1) char name[50]=”sanjaybhairajguru college of engineering”;

2) char s[10];

```
S[0] = 'a';
S[1] = 'b';
S[2] = 'c';
S[3] = 'd';
S[4] = 'e';
S[5] = 'g';
S[6] = '\0';
```

Here we store six character in string s but compiler automatically insert null character at the end of string so if we declare s[10] the only allow to store nine character.

List of string handling function:

Function	Syntax	Description	Example

strlen	int n = strlen(string1);	it returns number of characters of string1	int n = strlen("spit");
			n = 4

strcat	strcat(string1,string2)	string2 appends to the end of string1, and string2 remains unchanged	string1 = "very ";
			string2 = "good";
			strcat(string1,string2);
			string1 = "very good";
			string2 = "good";

strcmp	strcmp(string1,string2)	• compares string1 with string2 • if string1 = string2 then returns 0 • if string1 > string2 then returns +ve • if string1 < string2 then returns -ve	strcmp("there","there") then returns 0 strcmp("their","there") then returns -9

strcpy	strcpy(string1,string2)	string2 will copied into string1	string1 = "sardar ";
			string2 = "patel";
			strcat(string1,string2);
			string1 = "patel";
			string2 = "patel";

strlwr	strlwr(string1)	converts string1 into lower case	strlwr("SaRdaR")will return sardar
strupr	strupr(string1)	converts string1 into upper case	strupr("SaRdaR") will return SARDAR
strcmpi	strcmpi(string1,string2)	• compares string1 with string2 with ignoring case • if string1 = string2 then returns 0 • if string1 > string2 then returns +ve • if string1 < string2 then returns -ve	strcmp("there","tHere") then returns 32 strcmpi("there","tHere") then returns 0

strncmp	int n;
	strncmp(string1,string2)
		• compares string1 with string2 for only n characters
		• if string1 = string2 then returns 0
		• if string1 > string2 then returns +ve
		• if string1 < string2 then returns -ve
			int n= 3
			strncmp("there","their")
			then returns 0

strrev	strrev(string1)	returns the reverse the string of string1	strrev("there") returns ereht

strstr	strstr(string1,string2)	• it checks that string2 is contained in string1 or not
		• if yes then returns the position of first occurrence of substring
		• if no then returns NULL pointer
			string1="sardar patel"
			string2="patel"
			strstr(string1,string2)
			returns 7

strchr	strchr(string1,character1)	• it checks that character1 is contained in string1 or not
		• if yes then returns the position of first occurrence of character1
		• if no then returns NULL pointer
			string1="sardar patel"
			c1='r'
			strstr(string1,c1)
			returns 2

strrchr	strrchr(string1,character1)	• it checks that character1 is contained in string1 or not
		• if yes then returns the position of last occurrence of character1
		• if no then returns NULL pointer
			string1="sardar patel"
			c1='r'
			strrstr(string1,c1)
			returns 5

Que: 27 What is UDF? Explain elements of function and Explain category of function.
Ans:

- User defined function is group of statements and that function can be called from main function and use whenever it is required.
- Program is divided into functional parts and these parts are known as functions.
- In some cases certain operations or calculations are repeated many times in a program So the time and space will be wasted at time with help of UDF solve this problem.

There are three elements of user defined function.
1. Function declaration
2. Function call
3. Function definition

1. **Function declaration (Function prototype):**
   The calling program should declare the function before it is used. This is known as function declaration.
   ```
           int add(int, int);
   ```

2. **Function call :**
   Function call is a statement that is used to invoke the function for use it.
   A program that calls the function is known as calling function.

```
main( )
        {
                c = add(10, 5);//function call
                printf("%d", c);
        }
```

3. **Function definition :**
   The function definition is an independent program module and in that module function coding is written.
   Syntax:
   ```
           Functiondatatype function_name(list of argument)
           {
                   Function statements;
           }
   ```
   Example:
   ```
           int add(int x, int y)
           {
                   int z;
                   z = x + y;
                   return(z);
           }
   ```

Example:

```
#include<stdio.h>
#include<conio.h>
int add(int, int);                                  // Function declaration
void main()
{
        inta,b,c;
        clrscr();
        printf("enter the value of a & b :");
        scanf("%d%d",&a,&b);
        c = add(a,b);                               // Function call
        printf("sum = %d",c);
        getch();
}
int add(int x, int y)                               // Function definition
{
        int z;
        z = x + y;
        return(z);
}
```

Category of user defines function:

1. Function with NO ARGUMENTS and NO RETURN VALUES.
2. Function with ARGUMENTS but NO RETURN VALUES.
3. Function with ARGUMENTS and RETURN VALUES.
4. Function with NO ARGUMENTS but RETURN VALUES.

1. **Function with NO ARGUMENTS and NO RETURN VALUES:**
   Here function does not receive any data because function has no arguments. Here a calling function does not receive any data from called function because it is not return any value.

   Example:

```
        #include<stdio.h>
        #include<conio.h>
        Void printline(void);
        Void main( )
        {
                clrscr();
                printline();
                getch();
        }
        voidprintline(void)
        {
                printf("hello computer");
        }
```

2. **Function with ARGUMENTS but NO RETURN VALUES:**
Here called function receives data from calling function because function has arguments. Here a calling function does not receive any data from called function because it is not return any value. Example:

```
#include<stdio.h>
#include<conio.h>
Void  printline(int);
void main( )
{
        int a = 9;
        clrscr();
        printline(a);
        getch();
}
Void printline(int f)
{
        printf("answer = %d", f);
}
```

3. **Function with ARGUMENTS and RETURN VALUES:**
Here called function receive data from calling function because function has arguments. Here a calling function receive data from called function because it is return value.

Example:

```
#include<stdio.h>
#include<conio.h>
int add(int, int);
void main()
{
        inta,b,c;
        clrscr();
        printf("enter the value of a & b :");
        scanf("%d%d",&a,&b);
        c = add(a,b);
        printf("sum = %d",c);
        getch();
}
int add(int x, int y)
{
        int z;
        z = x + y;
        return(z);
}
```

4. **Function with NO ARGUMENTS but RETURN VALUES.**
Here a calling function has not arguments but called function return a value to the calling function.
Example:

```
#include<stdio.h>
#include<conio.h>
Int get_number(void);
void main( )
{
        int c;
        clrscr();
        printf(“enter number:”);
        c = get_number();
        printf(“the number = %d”, c);
        getch();
}
Int get_number(void)
{
        int  d;
        scanf(“%d”, &d);
        return(d);
}
```

QUE 28: Explain call by value and call by reference with example.
Ans:

Call by value:
When function is call from main() function at time Duplicate copy of original parameter is passing as function parameter then it is known as call by value.
In this type any change is made in UDF then it is not reflected on variable of main function.

Example:

```
#include <stdio.h>
#include <conio.h>
Void call_by_value(int x)
 {
        printf("Inside function call_by_valuex = %d before adding 10.\n", x);
        x =x+10;
        printf("Inside function call_by_value x = %d after adding 10.\n", x);
}
void main()
{
        int a=10;
        printf("a = %d before function call_by_value.\n", a);
        call_by_value(a);
        printf("a = %d after function call_by_value.\n", a);
        getch();
}
```
OUTPUT:
```
a = 10 before function call_by_value.
Inside function call_by_value x = 10 before adding 10.
Inside function call_by_value x = 20 after adding 10.
a = 10 after function call_by_value.
```

Call by Reference

When function is call from main( ) function at time reference of variable passing as function parameter then it is known as call by reference.
In this type any change is made in UDF then it is reflected on variable of main function.
Example:

```
#include <stdio.h>
#include <conio.h>
Void call_by_reference(int *x)
 {
        printf("Insidefunctioncall_by_referencex = %d before adding 10.\n", *x);
        *x =*x+10;
        printf("Inside function call_by_referencex = %d after adding 10.\n", *x);
}


void main()
{
        int a=10;
        printf("a = %d before function call_by_reference.\n", a);
        call_by_value(&a);
        printf("a = %d after function call_by_reference.\n", a);
        getch();
}
```

OUTPUT:

```
a = 10 before function call_by_reference.
Inside function call_by_referencex = 10 before adding 10.
Inside function  call_by_reference x = 20  after  adding 10.
a = 20 after function call_by_reference.
```

POINT	Call by value	Call by reference
Copy	Duplicate copy of original parameter is passed	Actual copy of original parameter is passed
Modification	No effect on original parameter after modifying parameter in function	Original parameter gets affected if value of parameter changed inside function

Que 29: What is recursion? Explain with example.
Ans:

- Function call itself then is called recursion.
- Recursion is function which calls itself repeatedly, until some condition has been satisfied. It is use for solving any iterative and complex problem.

```
#include<stdio.h>
#include<conio.h>
int fact(int);
```

```
void main( )
{
            int n,ans;
           clrscr();
            printf("enter n:");
           scanf("%d", &n);
           ans = fact(n);
           printf("%d", ans);
           getch();
}
int fact(int x)
{
int f;
if(x==1)
{
           return(1);
}


else
{
           f = x*fact(x-1);
           return(f);
}
}
```

- In above example user insert value of n=4 then function fact is call from the main method with parameter n=4 and value of n is stored in x.
- Then UDF fact is execute and check value of variable x if it is equal to 1 then return 1 other wise Else part is executed in that value of f=x*fact(x-1)
    - So f=4*fact(3)
- Again UDF fact is call from self-function fact with value of n=3 so value of f=4*3*face(2);
- This process continues until x becomes one.

Que 30: What is pointer? List advantage and disadvantage of pointer.
Ans:

- Pointer is variable which store the memory address of other variable.
- Memory addresses are the locations in the computer memory where program instructions and data are stored.
- The address of a variable can be finding with the help of the operator & in C.
- The asterisk (*) tells that the variable pointer_variableis a pointer variable.

Declaration of Pointer variable:

```
Datatype  pointer_name;
int* p;            //style 1
int*p;             //style 2
int* P;            //style 3
```

Example:

```
        int a;
        float b;
        intpointer_a=&a;        //integer pointer
        floatpointer_b=&b;      //float pointer
```

Here pointer_a store the memory address of int variable of a and pointer_b store the memory address of float variable b.

ADVANTAGES OF POINTERS:

- Pointers are more efficient in handling arrays and data tables.
- It returns more than one value from a function.
- It can pass arrays and string from one function to another.
- It can create complex data structures, such as linked lists, stacks, queues.
- It allows C to support dynamic memory management.
- It reduces length and complexity of programs.
- It increases the execution speed and thus reduces the program execution time.

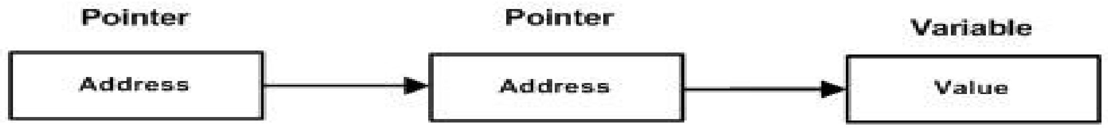
DISADVANTAGES OF POINTER:

- If the programmer is not careful and consistent with the use of pointers, the program may crash.
- It is complex program for multiple pointer declaration.

Que 31: Explain pointer to pointer with example.

Ans:

- A pointer to a pointer is a form of multiple indirections, or a chain of pointers.
- A pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



- A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int:
- int **var;
- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:

```
#include <stdio.h>
int main ()
{
int var;
int *ptr;
int **pptr;
var = 3000;
  /* take the address of var */
ptr = &var;
 /* take the address of ptr using address of operator & */
pptr = &ptr;
  /* take the value using pptr */
printf("Value of var = %d\n", var );
printf("Value available at *ptr = %d\n", *ptr );
printf("Value available at **pptr = %d\n", **pptr);
return 0;
}
```

OUTPUT:

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Que:32 What is structure? How to initialize structure member? How to access member of structure?
Ans:

- Structure is derived data type. It is the collection of variables of different types that known by common name.
- For example: You want to store the information about person about his/her name, citizenship number and salary.
- You can create this information separately but, better approach will be collection of this information under single name because all these information are related to person.
- In c language **Struct** keyword is use for declaring structure.

Syntax:

```
        Struct structure_name
        {
                data_type member1;
                data_type member2;
                 . .
                data_typememeber;
        };
```

Example:

```
        struct person
        {
                char name[50];
                intcit_no;
                float salary;
         };
```

Here struct is a keyword that declares a structure person that contains three members name, cit_no, and salary. All member inside structure having different data type and known by single name person.

For **initialize structure** member first require to create structure variable.
Syntax:
```
Struct structure_name
{
        data_type member1;
        data_type member2;
         . .
        data_typememeber;
 }p1,p2;
```
Here we create to two structure variable p1, p1 of type person.

For access structure member we use **.**(dot) operator.

Syntax for accessing structure member:

```
structure_variable.structure_member;
```
Example:
```
        Printf("Enter name");
        scanf("%s",&p1.name);
        printf("name of person p1 is %s",p1.salary);
```

Initialization of structure member:
Syntax:
1) **s**tructure_variable.structure_member=value;
2) stuctstructure_ namestructure_variable={values1,valu2, ...};

Example:
```
1)      p1.name="sanjay";
        p1.salary=20000;
        p2.name="jone";
2)  stuct person p1={"janjay",2,"20000"};
```

Que 33: What is nested structure? Explain with example.
Ans:
- Structure written inside another structure is called as nested structure or nesting of structure.
- In nested structure we can declare another structure in structure as member of structure.

- **Declare nested structure**:
```
struct salary
{
        char name;
        char department;
                struct
                {
                int  dearness;
                int house_rent;
                int city;
                }allowance;
        }employee;
```

- **For accessing structure member:**
  employee.allowance.dearness;
  employee.allowance.house_rent;
  employee.allowance.city;

```
#include <stdio.h>
#include <string.h>
struct salary
        {
                char name;
                char department;
                        struct
                        {
                        int  dearness;
                        int house_rent;
                        intcity_code;
                        }allowance;
        }emp1;

int main()
{
  struct salaryemp1 = {"Raju", “IT",100,2000,360001};
  printf(" name is: %s \n", emp1.name);
  printf(" department is: %s \n", emp1.department);
  printf("dearnessis:  %d  \n\n",emp1.allowance.dearness);

  printf("house_rent is: %d \n",emp1.allowance.house_rent);
  printf("city_codeName is: %s \n", emp1.allowance.city_code);
  return 0;
}
```

Que 34: What is dynamic memory allocation (DMA)? Explain malloc and calloc with example.
Ans:

- Dynamic memory allocation allows a program to obtain more memory space, while running or to release space when no space is required.
- In other word process of allocating memory at run time is known as dynamic memory allocation.
- Dynamic memory allocation use for allocating or releasing memory at run time. For that function are available in stdlib.h header file.

malloc( )

The name malloc stands for "memory allocation". The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form.

Syntax:

```
ptr=(cast-type*)malloc(byte-size)
```

Here, *ptr* is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

ptr=(int*)malloc(100*sizeof(int));

This statement will allocate 200 byte size and pointer points to the address of first byte of memory.

Example:

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

        int main()
        {
                int * table;
                table = (int *)malloc(sizeof(int));
                if (table == 0)
                {
                        printf("ERROR: Out of memory\n");
                        return 1;
                }
                * table = 25;
                printf("%d\n", * table);
                free(table);
                return 0;
        }
```

calloc()

The name calloc stands for "contiguous allocation". The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax of calloc()

```
                    ptr=(cast-type*)calloc(n,element-size);
```

This statement will allocate contiguous space in memory for an array of *n* elements. For example:

```
          ptr=(float*)calloc(25,sizeof(float));
```

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

Example:

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
inti, n;
int *a;
printf("Number of elements to be entered:");
scanf("%d",&n);
 a = (int*)calloc(n, sizeof(int));
printf("Enter %d numbers:\n",n);
for(i=0 ; i< n ; i++ )
{
scanf("%d",&a[i]);
}

printf("The numbers entered are: ");
for(i=0 ; i< n ; i++ ) {
printf("%d ",a[i]);
}
return(0);
}
```

Que 35: Explain file handling function in c.
Ans:

Function Name	**Operation**
fopen()	Creates a new file for use Opens a new existing file for use
Fclose()	Closes a file which has been opened for use
getc()	Reads a character from a file
putc()	Writes a character to a file
fprintf()	Writes a set of data values to a file
fscanf()	Reads a set of data values from a file
getw()	Reads a integer from a file
putw()	Writes an integer to the file
fseek()	Sets the position to a desired point in the file
ftell()	Gives the current position in the file
rewind()	Sets the position to the beginning of the file

fopen():

General format for declaring and opening a file :

*FILE *fp;*
fp= fopen("filename","mode");

here filename is any string and modes can be one of the followings :

•r-----open a file for reading only
•w----- create a file for writing only
•a ---- append (add) to file
•r+ ----open a file for read / write
•w+ ---create a file for read / write

•a+---- append a file for read / write

For example :

```
FILE *p1, *p2;
p1 = fopen("data", "r");
p2 = fopen("results", "w");
```

f**close( ):**

After completing the operations on the files, the file must be closed using the fclose( ) function.

```
fclose( FILE *fp);
```

Writing a File

Following is the simplest function to write individual characters to a stream:

```
fputc(int c, FILE *fp);
```

The function **fputc()** writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success otherwise **EOF** if there is an error. You can use the following functions to write a null-terminated string to a stream:

```
intfputs(constchar*s, FILE *fp);
```

The function **fputs()** writes the string **s** to the output stream referenced by fp. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error. You can use

```
intfprintf(FILE *fp,const char *format, ...)
```

Function as well to write a string into a file.

Example:

```
#include<stdio.h>

Void main()
{
  FILE *fp;

fp=fopen("/tmp/test.txt","w+");
fprintf(fp,"This is testing for fprintf...\n");
fputs("This is testing for fputs...\n",fp);
fclose(fp);
}
```

When the above code is compiled and executed, it creates a new file **test.txt** in /tmp directory and writes two lines using two different functions.

Reading a File

Following is the simplest function to read a single character from a file:

```
intfgetc( FILE *fp);
```

The **fgetc()** function reads a character from the input file referenced by fp. The return value is the character read, or in case of any error it returns **EOF**. The following functions allow you to read a string from a stream:

```
char*fgets(char*buf,int n, FILE *fp);
```

The functions **fgets()** reads up to n - 1 characters from the input stream referenced by fp. It copies the read string into the buffer **buf**, appending a **null** character to terminate the string.

If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including new line character.

```
intfscanf(FILE *fp, const char *format, ...)
```

function to read strings from a file but it stops reading after the first space character encounters.

```
#include<stdio.h>

main()
{
  FILE *fp;
charcontent[255];

fp=fopen("/tmp/test.txt","r");
fscanf(fp,"%s",content);
printf(" first content : %s\n",content);

for(content=fgetc(fp);c!=EOF;c++)
{
        Putc(c);
}
fclose(fp);

}
```

ftell( ) :

ftell( ) function gives the current position in the file .It takes the following form

n = ftell(fp);

Here fpis a FILE pointer associated with the file. The function ftell( ) returns a type long that gives the current file position in bytes from the start of the file (the first byte is at position 0).

In case of an error, ftell() returns -1L (a type long -1) So n will give the relative offset (in bytes) of the current position

fseek( ) :

fseek( ) function is used to set the position anywhere in the file .It takes the following form :

fseek(file_pointer, offset, position);

Here fpis a FILE pointer associated with the file.The offset specifies the number of positions to be moved from the location specified by position.
The offset may be positive, meaning moves forwards, or negative, meaning move backwards.
The position can take one of the following values :

VALUE	**MEANING**
0	Starting of file
1	Current position
2	End of file

•STATEMENT MEANING:

fseek(fp,0L,0);	go to the starting
fseek(fp,0L,1);	stay at the current position
fseek(fp,0L,2);	go to the end of file
fseek(fp,m,0);	move to (m+1)thbyte in the file
fseek(fp,m,1);	go forwards by m bytes
fseek(fp,-m,1);	go backward by m bytes from the current position
fseek(fp,-m,2);	go backward by m bytes from the end

Explain different Storage classes in C

Storage classes

To define a variable its data type and its storage class needs to be mentioned. The storage class informs the following:

-Where the variable would be stored.
-The default initial value.
-scope of the variable (availability of the variable in different files and functions)
-life of the variable

The four storage classes are

1) Automatic storage class

A variable declared without any storage class is considered to be an automatic variable. It can also be declared using the keyword auto

Eg. auto int a; // (or) int a;

Features of a variable having automatic storage class are as under

Storage	- memory
Default initial value	- garbage value if not initialized
Scope	- only in the block where it is defined
Life	- till the block is executing.

2) Register storage class

A variable declared with the keyword register is a register variable.

Features of a variable having register storage class

Storage	- CPU registers
Default initial value	- garbage value if not initialized
Scope	- only in the block where it is defined
Life	- till the block is executing.

A value stored in the CPU register can always be accessed faster than the one which is stored in memory. Therefore, if a variable is used at many places in a program it is better to declare its storage class as register. Since the registers are limited in number the variable is not necessarily stored in the register. Then the variables are considered as automatic. Loop variables may be ideally declared as register.

Eg. register int a;

3) Static Storage class

The features of a variable having static storage class are as follows

Storage	-Memory
Default initial value	- Zero (Null)
Scope	- only in the block where it is defined

Life - the variable is not destroyed after the block or function is over. The variable is initialized only once for the program, irrespective of the multiple function calls. i.e. the value persists between multiple calls to the function.

eg. static int t;
static char n;

void increment()	void increment()
{ auto int i=1;	{ static int i; // automatically gets initialized to 0
printf(“%d”,i);	printf(“%d”,i);
i=i+1;	i=i+1;
}	}
main()	main()
{ increment();	{ increment();
increment();	increment();
increment();	increment();
}	}

O/p would be 1 1 1	O/p would be 0 1 2

4) External Storage Class

The features of a variable having external storage class are as follows

Storage	- Memory
Default initial value	- Zero
Scope	- Global
Life	- As long as the **program's** execution does not end

External variables are declared outside all functions, yet are available to all functions that care to use them. Functions requiring to use the external variable may redeclare it inside the function using the **extern** keyword. This redeclaration is optional, since without the redeclaration also all the functions have access to all external variables.

Explain C Preprocessor

A preprocessor directive begins with a # symbol. Preprocessor directives may be placed anywhere in the program but are generally placed in the beginning of the program. The different preprocessor directives are

- Macro expansion (#define)
- File Expansion (#include)
- Conditional compilation (#ifdef)
- Miscellaneous directives

Macro Expansion:

Eg. #define MAX 25

In the above example MAX is called the Macro template and 25 is called the macro expansion. Whereever the macro template is present in the program the preprocessor replaces it with the expansion.

Macro expansion makes the program easier to read and modify. Since the template MAX is used in the program it indicates some maximum value. Also any modifications to the value can be made at the definition. Hence the entire program need not be changed.

Macros with arguments:

```
#define AREA(x) (3.14*x*x)
void main()
{    float a;
     a= AREA(5);
     printf("Area of circle with radius 5 is %f", a);
     printf("Area of circle with radius 10 is %f", AREA(10));
}
```

Macros	**Functions**
In a macro call the preprocessor simply replaces the template with its macro expansion	In a function call the control is passed to the function with arguments, executes the function and returns some value.
Many macro expansions makes the source larger	Many function calls does not increase the size of the program
Macros are fast to execute	Function calls are slow to execute
Macros are expanded before the program is compiled	Function calls are executed during execution of the program.

<u>File Inclusion:</u>

Eg. #include <filename>

This directive causes the filename to be included in the program. By this all the contents of the specified file are placed at the point of directive.

A large file may be divided into several different files each containing a set of related functions. These files are then included wherever the functions are required.

Also some functions may be required in most of the programs. In such a case the commonly need functions can be stored in a file and that file can be #included in the program.

#include can be written in two ways.

#include<filename>	: causes the filename to be searched in the specified library path only
#include "filename"	: causes the filename to be search in the current directory as well as the specified library path.