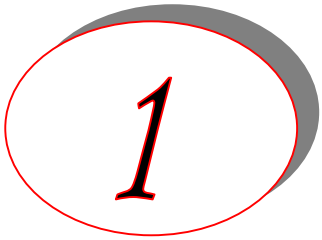
Table of Contents



Data Structures

Table 3.1. The relative complexity of all the algorithms

S No	*Algorithm*	*Best*	*Average*	*Worst*
1	Quick sort	*O(N log N)*	*O(N log N)*	*O(N^2/2)*
2	Merge sort	*O(N longN)*	*O(N log N)*	*O(N log N)*
3	Radix sort	*O(N)*	*O(M+N)*	*O(M+N)*
4	Linear search	O(1)		*O(N)*
5	Binary search	*O(log $_2$N)*	*O(log $_2$N)*	O(N)
6	Hashing	*O(N/M)**		
7	Digital trie	*O(1)***		
8	Heap sort	O(NlogN)	O(NlogN)	O(NlogN)
9	Selection sort		*O(N^2)*	*O(N^2)*
10	Bubble sort		*O(N^2/4)*	*O(N^2)*
11	Insertion sort	*O(N^2)*	*O(N^2)*	*O(N^2)*
12	Shell sort	*O($N^{1.2}$)*	*O($N^{1.5}$)*	*O(N^2)*
13	Linked list (insertion,deletion,search)	O(1)	O(N/2)	O(N)
14	Bucket sort	O(N)	O(N)	

Full binary tree of height h (number of levels)has $l=2^h$ *where*
l= number of leaves and internal nodes m= 2^h –1. Total nodes n=2^{h+1} -1
The full binary tree with n nodes height h=log(n+1) –1
The depth of a complete binary tree with n nodes is d=log(n+1) –1

1)Pick up the best average behavior of sorting technique
a)Quick sort b) Selection sort c) Heap sort d) Merge sort e)Radix sort
Ans) a
2)Which of the following is best if number of swapping done is only measure of efficieny?
a)Quick sort b) Selection sort c) Insertion sort d) Bubble sort
Ans) b
3)Quick sort algorithm requires a max of *O(n) stack space* in the worse case
4)For merging two sorted lists of sizes m and n into a sorted list of size m+n then *O(m+n)* comparisions are required.
5)The average successful search time for sequential search on n times is *(n+1)/2*
6)Maximum number of interchanges done in selection sort is *n-1*
7)No of comparisions are done in bubble sort in worst case is *n(n-1)/2*
8)If a double linked list contains n nodes then it contains *2n* no of address parts.
9)Best sorting method for elements in ascending order is selection
10) A full binary tree has 101 nodes then how many subroots it has
a)50 b)49 c)51 d)none
Ans) a

1)What is data structure?

Ans) A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.

There are two types of Data structures

Linear Data Structures: Data structures using sequential allocation are called linear data structures. A linear data structures shows the relationship of adjacency between elements.
E.g.: Arrays, Records, Stacks, and Queues etc.

Non-Linear Data Structures: Data structures using linked allocation are non-linear data structures.
E.g. Trees, Graphs, etc.

2)List out the areas in which data structures are applied extensively?

Ans)

- Compiler Design,
- Operating System,
- Database Management System,
- Statistical analysis package,
- Numerical Analysis,
- Graphics,
- Artificial Intelligence,
- Simulation

3)What are the major data structures used in the following areas: RDBMS, Network data model & Hierarchical data model?

Ans)

- RDBMS – Array (i.e. Array of structures)
- Network data model – Graph
- Hierarchical data model – Trees

4)If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

Ans) The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

5)Minimum number of queues needed to implement the priority queue?

Ans) Two, One queue is used for actual storing of data and another for storing priorities.

6)What is the data structures used to perform recursion?

Ans) Stack, Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.

7)What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

Ans) Polish and Reverse Polish notations.

8)Convert the expression ((A + B) * C – (D – E) ^ (F + G)) to equivalent Prefix and Postfix notations.

Ans)

Prefix Notation: -*+ABC^-DE+FG

Postfix Notation: AB+C*DE-FG+^-

9)Sorting is not possible by using which of the following methods?

(a) Insertion

(b) Selection

(c) Exchange

(d) Deletion

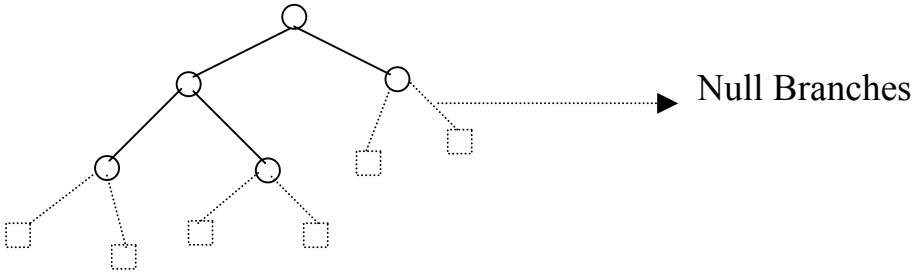
Ans) (d) Deletion.

Using insertion we can perform insertion sort, using selection we can perform selection sort, using exchange we can perform the bubble sort (and other similar sorting methods). But no sorting method can be done just using deletion.

10)A binary tree with 20 nodes has null branches?

Ans) 21

Let us take a tree with 5 nodes (n=5)

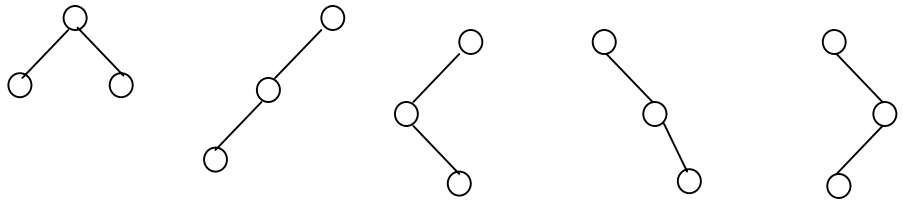


It will have only 6 (ie, 5+1) null branches. In general,

*A binary tree with **n** nodes has exactly **n+1** null nodes.*

11)What are the methods available in storing sequential files?
Ans)
- Straight merging,
- Natural merging,
- Polyphase sort,
- Distribution of Initial runs.

12)How many different trees are possible with 10 nodes ?
Ans) 1014
For example, consider a tree with 3 nodes(n=3), it will have the maximum combination of 5 different (ie, $2^3 - 3 = 5$) trees.



i ii iii iv v

In general: *If there are **n** nodes, there exist **2^n-n** different trees.*

13)List out few of the Application of tree data-structure?
Ans)
- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

14)List out few of the applications that make use of Multilinked Structures?
Ans)
- Sparse matrix,
- Index generation.

15)In tree construction which is the suitable efficient data structure?
(a) Array (b) Linked list (c) Stack (d) Queue (e) none
Ans) (b) Linked list

16)What is the type of the algorithm used in solving the 8 Queens problem?
Ans) Backtracking

17)In an AVL tree, at what condition the balancing is to be done?
Ans) If the 'pivotal value' (or the 'Height factor') is greater than 1 or less than –1.

18)What is the bucket size, when the overlapping and collision occur at same time?
Ans) One. If there is only one entry possible in the bucket, when the collision occurs, there is no way to accommodate the colliding value. This results in the overlapping of values.

19)There are 8, 15, 13, 14 nodes were there in 4 different trees. Which of them could have formed a full binary tree?

Ans) 15.

In general:

There are ***2^n-1*** *nodes in a full binary tree.*

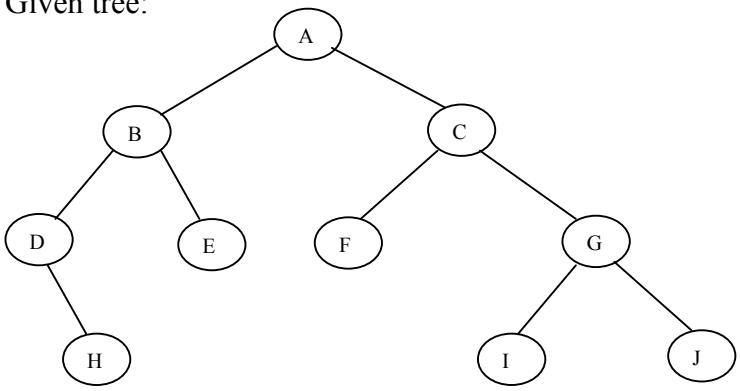
By the method of elimination:

Full binary trees contain *odd* number of nodes. So there cannot be full binary trees with 8 or 14 nodes, so rejected. With 13 nodes you can form a *complete* binary tree but not a full binary tree. So the correct answer is 15.

Note: Full and Complete binary trees are different. *All full binary trees are complete binary trees but not vice versa.*

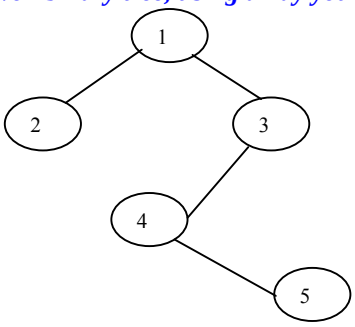
20)Traverse the given tree using Inorder, Preorder and Postorder traversals.

Given tree:



- Inorder : D H B E A F C I G J
- Preorder: A B D H E C F G I J
- Postorder: H D E B F I J G C A

21)In the given binary tree, using array you can store the node 4 at which location?



Ans) At location 6

1	2	3	-	-	4	-	-	5
root	LC1	RC1	LC2	RC2	LC3	RC3	LC4	RC4

where LCn means Left Child of node n and RCn means Right Child of node n

22)Sort the given values using Quick Sort?

6	7	7	8	8	6	5	5	4
5	0	5	0	5	0	5	0	5

Sorting takes place from the pivot value, which is the first value of the given elements, this is marked bold. The values at the left pointer and right pointer are indicated using ^{L} and ^{R} respectively.

6	7	7	8	8	6	5	5	4
5	0	5	0	5	0	5	0	5
	L							R

Since pivot is not yet changed the same process is continued after interchanging the values at ^{L} and ^{R} positions

6	4	7	8	8	6	5	5	7
5	5	5	0	5	0	5	0	0
		L					R	

6	4	5	8	8	6	5	7	7
5	5	0	0	5	0	5	5	0
			L			R		

6	4	5	5	8	6	8	7	7
5	5	0	5	5	0	0	5	0
				L	R			

6	4	5	5	6	8	8	7	7
5	5	0	5	0	5	0	5	0
				R	L			

When the L and R pointers cross each other the pivot value is interchanged with the value at right pointer. If the pivot is changed it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed, one from start of the original array to the pivot position-1 and the other from pivot position+1 to end.

6	4	5	5	***6***	**8**	8	7	7
0	5	0	5	***5***	**5**	0	5	0
L			R		L			R

5	4	5	***6***	***6***	**7**	8	7	***8***
5	5	0	***0***	***5***	**0**	0	5	***5***
L		R			R	R		

5	4	***5***	***6***	***6***	***7***	8	7	***8***
0	5	***5***	***0***	***5***	***0***	0	5	***5***

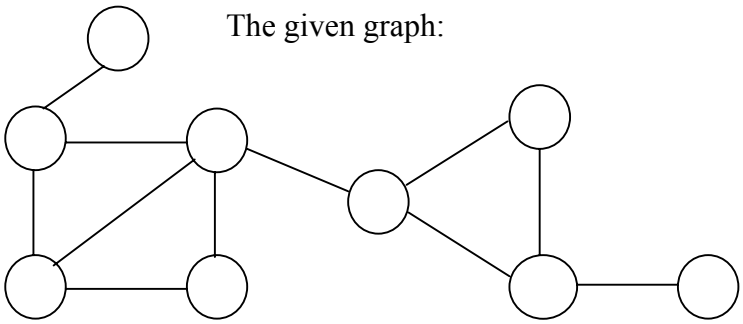
L R L R

In the next pass we get the sorted form of the array.

4	***5***	***5***	***6***	***6***	***7***	***7***	***8***	***8***
5	***0***	***5***	***0***	***5***	***0***	***5***	***0***	***5***

23)For the given graph, draw the DFS and BFS?

The given graph:



- *BFS:* A X G H P E M Y J
- *DFS:* A X H P E Y M J G

24. ***Classify the Hashing Functions based on the various methods by which the key value is found.***

Ans)

- Direct method,
- Subtraction method,
- Modulo-Division method,
- Digit-Extraction method,
- Mid-Square method,
- Folding method,
- Pseudo-random method.

25. ***In RDBMS, what is the efficient data structure used in the internal storage representation?***

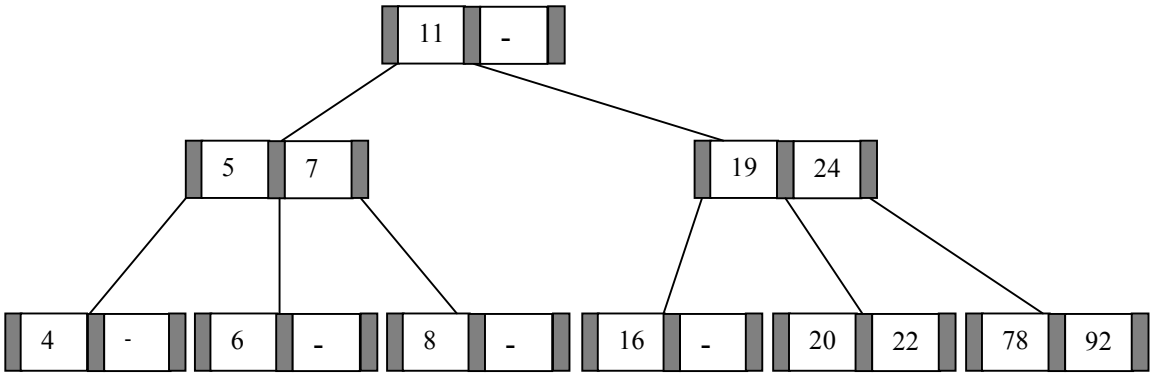
Ans) B+ tree. Because in B+ tree, all the data is stored only in leaf nodes, that makes searching easier. This corresponds to the records that shall be stored in leaf nodes.

26. ***What are the types of Collision Resolution Techniques and the methods used in each of the type?***

Ans)

- Open addressing (closed hashing),
  - The methods used include: Overflow block,linear probing,quadratic probing,double hasing,rehasing.
- Closed addressing (open hashing)
  - The methods used include: Linked list, Binary tree…

27. ***Draw the B-tree of order 3 created by inserting the following data arriving in sequence – 92 24 6 7 11 8 22 4 5 16 19 20 78***



28. ***Of the following tree structure, which is, efficient considering space and time complexities?***

1) Incomplete Binary Tree
2) Complete Binary Tree
3) Full Binary Tree

Ans) (2) Complete Binary Tree.

By the method of elimination:

Full binary tree loses its nature when operations of insertions and deletions are done. For incomplete binary trees, extra storage is required and overhead of NULL node checking takes place. So complete binary tree is the better one since the property of complete binary tree is maintained even after operations like additions and deletions are done on it.

29. ***What is a spanning Tree?***

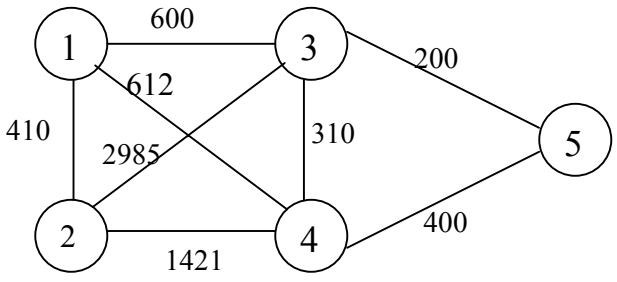
Ans) A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

30. ***Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?***

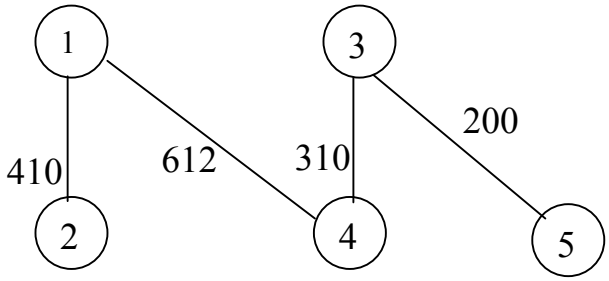
Ans) No

Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it *doesn't* mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

31. ***Convert the given graph with weighted edges to minimal spanning tree.***



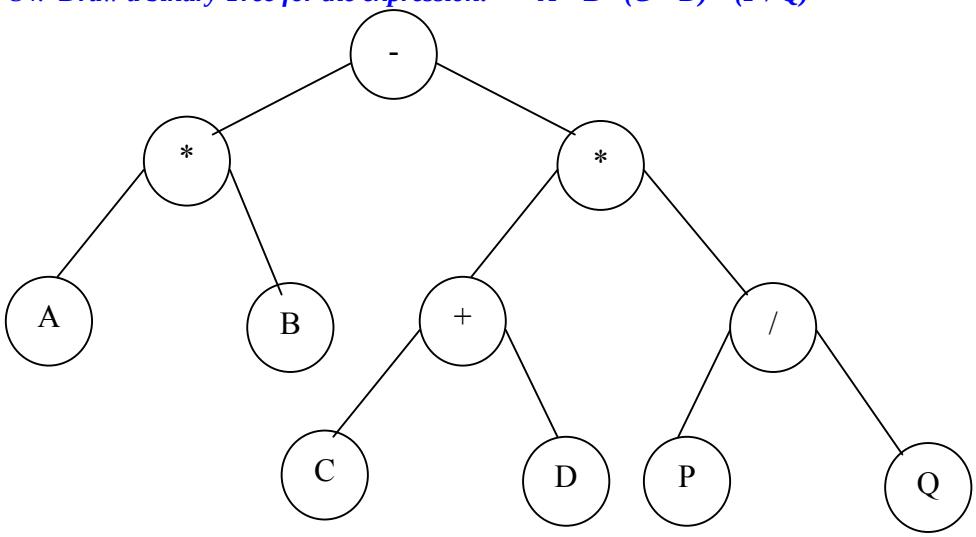
Ans) equivalent minimal spanning tree is:



32. ***Which is the simplest file structure?***
a)seqential
b)Indexed
c)Random
Ans) (a) Sequential

33. ***Whether Linked List is linear or Non-linear data structure?***
Ans) According to Access strategies Linked list is a linear one.
According to Storage Linked List is a Non-linear one.

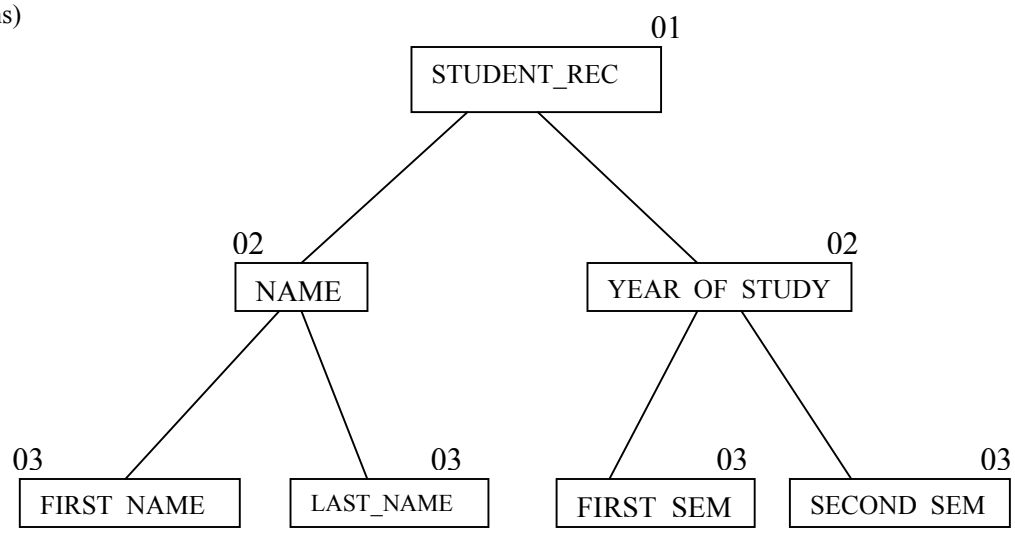
34. ***Draw a binary Tree for the expression:*** ***A * B - (C + D) * (P / Q)***



35. ***For the following COBOL code, draw the Binary tree?***

```
01 STUDENT_REC.
   02 NAME.
        03 FIRST_NAME PIC X(10).
        03 LAST_NAME PIC X(10).
   02 YEAR_OF_STUDY.
        03 FIRST_SEM PIC XX.
        03 SECOND_SEM PIC XX.
```

Ans)



36. What is Stack? What are applications of stacks?

Ans)

1. A Stack is an ordered list (linear data structure) in which all insertion and deletions are made at one end.
2. It works on the principal Last in First Out.
3. In a stack insertion operation is called "PUSH" and deletion is called "POP"
4. In a stack most accessible elements is only top element of the stack

Application of stack:

1. Stacks are used in the evaluation of arithmetic's expression
2. Stacks play an important role in parsing arithmetic expression such as a*(a-1)
3. When a function is called ,its return address and arguments are pushed onto a stack, and when the function returns they are popped off.
4. In recursion, all intermediate arguments and return values on the micro process's stack.
5. Stacks are very useful when using certain complex data structures like "Binary Trees" . In this a stack is used to traverse the nodes of a tree.

37. What are Binary tree, Complete Binary tree and Full Binary tree?

Ans)

Binary Tree: A binary tree is either empty or it consists of a node called the root together with two binary tree called the left sub-tree and right sub-tree of the root.

Complete Binary Tree: The tree T is said to be complete if all its levels, except possibly the last, have the maximum number of possible nodes, and if all the nodes at the last level appears as far left as possible.

The depth of the complete tree T with n-nodes is given by $D= \log_2(n+1)$

Full Binary Tree: The tree T is said to be Full Binary Tree if all the non leaf nodes are full.

Application of Trees:

1) Symbolic manipulations of algebraic expressions

2) Searching using binary trees (i.e. it takes minimum searching time)
3) Used in sorting
4) Examining the are of syntax analysis and its relationship to parse trees
5) It is not a random-access structure like a simple array, it provides faster and more constant access to individual nodes than does a linked list.
6) Thus it is particularly suitable for application in which search time must be minimized or the nodes will necessarily be processed in order.

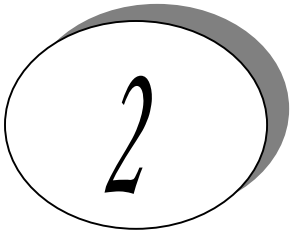
38) Characteristics of Data Structures

Ans)

S .No	DataStructure	Advantages	Disadvantages
1	Array	Quick insertion, very fast access if index known	Slow search, Slow Selection, fixed size
2	Ordered array	Quicker search than unsorted array	Slow insertion and deletion, fixed size
3	Stack	Provides LIFO access	Slow access to other Items.
4	Queue	Provides FIFO access	Slow access to other Items.
5	Linked list	Quick insertion, Quick deletion	Slow search
6	Binary tree	Quick search, insertion, deletion (if tree remains balanced)	Deletion algorithm is complex
7	Hash table	Very fast access if key known. Fast insertion	Slow deletion, access slow if key not know, Inefficient memory usage
8	Heap	Fast insertion, deletion, access to largest item	Slow access to other Items.

39)Glossary of Data Structures

Data Abstraction: The separation of logical properties of the organization of program's data from its implementation, ignoring inessential details.
Data Encapsulation: Separation of the representation of data from the applications that use the data at logical level.
Hashing: The Technique used for ordering and accessing elements in a list in relatively constant amount of time by manipulating the key to produce a unique location.



C Aptitude

Note : All the programs are tested under Turbo C/C++ compilers.
It is assumed that, Programs run under DOS environment,The underlying machine is an x86 system, Program is compiled using Turbo C/C++ compiler.
The program output may depend on the information based on this assumptions (for example sizeof(int) == 2 may be assumed).

Predict the output or error(s) for the following:

1.
```
void main()
{
int  const * p=5;
printf("%d",++(*p));
}
```
Answer:
Compiler error: Cannot modify a constant value.
Explanation:
p is a pointer to a "constant integer". But we tried to change the value of the "constant integer".

2.
```
main()
{
char s[ ]="man";
int i;
for(i=0;s[ i ];i++)
printf("\n%c%c%c%c",s[ i ],*(s+i),*(i+s),i[s]);
}
```
Answer:
mmmm
aaaa
nnnn
Explanation:
s[i], *(i+s), *(s+i), i[s] are all different ways of expressing the same idea. Generally array name is the base address for that array. Here **s** is the base address. **i** is the index number/displacement from the base address. So, indirecting it with * is same as s[i]. i[s] may be surprising. But in the case of C it is same as s[i].

3.
```
main()
{
float me = 1.1;
double you = 1.1;
if(me==you)
    printf("I love U");
    else
        printf("I hate U");
```

```
}
```

Answer:

I hate U

Explanation:

For floating point numbers **(**float, double, long double**)** the values cannot be predicted exactly. Depending on the number of bytes, the precession with of the value represented varies. Float takes 4 bytes and long double takes 10 bytes. So float stores 0.9 with less precision than long double.

Rule of Thumb:

Never compare or at-least be cautious when using floating point numbers with relational operators **(**== , >, <, <=, >=,!= **)** .

4.
```
main()
{
static int var = 5;
printf("%d ",var--);
if(var)
        main();
}
```

Answer:

5 4 3 2 1

Explanation:

When *static* storage class is given, it is initialized once. The change in the value of a *static* variable is retained even between the function calls. Main is also treated like any other ordinary function, which can be called recursively.

5.
```
main()
{
 int c[ ]={2.8,3.4,4,6.7,5};
 int j,*p=c,*q=c;
 for(j=0;j<5;j++) {
          printf(" %d ",*c);
          ++q;       }
 for(j=0;j<5;j++){
      printf(" %d ",*p);
      ++p;            }
}
```

Answer:

2 2 2 2 2 2 3 4 6 5

Explanation:

Initially pointer c is assigned to both **p** and **q**. In the first loop, since only **q** is incremented and not c , the value 2 will be printed 5 times. In second loop **p** itself is incremented. So the values 2 3 4 6 5 will be printed.

6.
```
main()
```

```
{
extern int i;
i=20;
      printf("%d",i);
}
```

Answer:
Linker Error : Undefined symbol '_i'
Explanation:
extern storage class in the following declaration,
extern int i;
specifies to the compiler that the memory for **i** is allocated in some other program and that address will be given to the current program at the time of linking. But linker finds that no other variable of name **i** is available in any other program with memory space allocated for it. Hence a linker error has occurred .

7.
```
main()
{
int i=-1,j=-1,k=0,l=2,m;
m=i++&&j++&&k++||l++;
printf("%d %d %d %d %d",i,j,k,l,m);
}
```

Answer:
0 0 1 3 1
Explanation :
Logical operations always give a result of **1 or 0** . And also the logical AND (&&) operator has higher priority over the logical OR (||) operator. So the expression '**i++ && j++ && k++'** is executed first. The result of this expression is 0 (-1 && -1 && 0 = 0). Now the expression is 0 || 2 which evaluates to 1 (because OR operator always gives 1 except for '0 || 0' combination- for which it gives 0). So the value of m is 1. The values of other variables are also incremented by 1.

8.
```
main()
{
char *p;
printf("%d %d ",sizeof(*p),sizeof(p));
}
```

Answer:
1 2
Explanation:
The sizeof() operator gives the number of bytes taken by its operand. P is a character pointer, which needs one byte for storing its value (a character). Hence sizeof(*p) gives a value of 1. Since it needs two bytes to store the address of the character pointer sizeof(p) gives 2.

9.
```
main()
{
int i=3;
switch(i)
 {
    default:printf("zero");
    case 1: printf("one");
              break;
   case 2:printf("two");
             break;
  case 3: printf("three");
             break;
 }
}
```

Answer :

three

Explanation :

The default case can be placed anywhere inside the loop. It is executed only when all other cases doesn't match.

10.
```
main()
{
  printf("%x",-1<<4);
}
```

Answer:

fff0

Explanation :

-1 is internally represented as all 1's. When left shifted four times the least significant 4 bits are filled with 0's.The %x format specifier specifies that the integer value be printed as a hexadecimal value.

11.
```
main()
{
    char string[]="Hello World";
display(string);
}
void display(char *string)
{
printf("%s",string);
}
```

Answer:

Compiler Error : Type mismatch in redeclaration of function display

Explanation :

In third line, when the function **display** is encountered, the compiler doesn't know anything about the function display. It assumes the arguments and return types to be integers, (which is the default type). When it sees the actual function **display,** the arguments and type contradicts with what it has assumed previously. Hence a compile time error occurs.

12.
```
main()
{
int c=- -2;
printf("c=%d",c);
}
```
Answer:
c=2;
Explanation:
Here unary minus (or negation) operator is used twice. Same maths rules applies, ie. minus * minus= plus.
Note:
However you cannot give like --2. Because -- operator can only be applied to variables as a **decrement** operator (eg., i--). 2 is a constant and not a variable.

13.
```
#define int char
main()
{
int i=65;
printf("sizeof(i)=%d",sizeof(i));
}
```
Answer:
sizeof(i)=1
Explanation:
Since the #define replaces the string **int** by the macro **char**

14.
```
main()
{
    int i=10;
    i=!i>14;
    Printf ("i=%d",i);
}
```
Answer:
i=0
Explanation:
In the expression **!i>14** , NOT (!) operator has more precedence than ' >' symbol. **!** is a unary logical operator. !i (!10) is 0 (not of true is false). 0>14 is false (zero).

15.
```
#include<stdio.h>
main()
{
    char s[]={'a','b','c','\n','c','\0'};
    char *p,*str,*str1;
    p=&s[3];
    str=p;
    str1=s;
    printf("%d",++*p + ++*str1-32);
}
```
Answer:
77

Explanation:
p is pointing to character '\n'. str1 is pointing to character 'a' ++*p. "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10, which is then incremented to 11. The value of ++*p is 11. ++*str1, str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98.
Now performing (11 + 98 – 32), we get 77("M");
So we get the output 77 :: "M" (Ascii is 77).

16.
```
#include<stdio.h>
main()
{
    int a[2][2][2] = { {10,2,3,4}, {5,6,7,8}  };
    int *p,*q;
    p=&a[2][2][2];
    *q=***a;
    printf("%d----%d",*p,*q);
}
```
Answer:
SomeGarbageValue---1
Explanation:
p=&a[2][2][2] you declare only two 2D arrays, but you are trying to access the third 2D(which you are not declared) it will print garbage values. *q=***a starting address of a is assigned integer pointer. Now q is pointing to starting address of a. If you print *q, it will print first element of 3D array.

17.
```
#include<stdio.h>
main()
{
    struct xx
    {
        int x=3;
        char name[]="hello";
     };
    struct xx *s;
    printf("%d",s->x);
    printf("%s",s->name);
}
```
Answer:
Compiler Error
Explanation:
You should not initialize variables in declaration

18.
```
#include<stdio.h>
main()
{
    struct xx
    {
int x;
        struct yy
        {
            char s;
        struct xx *p;
        };
        struct yy *q;
    };
}
```

Answer:

Compiler Error

Explanation:

The structure yy is nested within structure xx. Hence, the elements are of yy are to be accessed through the instance of structure xx, which needs an instance of yy to be known. If the instance is created after defining the structure the compiler will not know about the instance relative to xx. Hence for nested structure yy you have to declare member.

19.
```
main()
{
    printf("\nab");
    printf("\bsi");
    printf("\rha");
}
```

Answer:

ha

Explanation:

\n - newline
\b - backspace last character will be removed
\r - linefeed all the characters before will be removed

20.
```
main()
{
    int i=5;
    printf("%d%d%d%d%d%d",i++,i--,++i,--i,i);
}
```

Answer:

45545

Explanation:

The arguments in a function call are pushed into the stack from left to right. The evaluation is by popping out from the stack. and the evaluation is from right to left, hence the result.

21.
```
#define square(x) x*x
main()
{
    int i;
    i = 64/square(4);
    printf("%d",i);
}
```

Answer:

64

Explanation:

the macro call square(4) will substituted by 4*4 so the expression becomes i = 64/4*4 Since / and * has equal priority expression will be evaluated as (64/4)*4 i.e. 16*4 =64

22.
```
main()
{
char *p="hai friends",*p1;
p1=p;
while(*p!='\0') ++*p++;
printf("%s   %s",p,p1);
}
```

Answer:

ibj!gsjfoet

Explanation:

++*p++ will be parse in the given order

- *p that is value at the location currently pointed by p will be taken
- ++*p the retrieved value will be incremented
- when ; is encountered the location will be incremented that is p++ will be executed

Hence, in the while loop initial value pointed by p is ‘h’, which is changed to ‘i’ by executing ++*p and pointer moves to point, ‘a’ which is similarly changed to ‘b’ and so on. Similarly blank space is converted to ‘!’. Thus, we obtain value in p becomes “ibj!gsjfoet” and since p reaches ‘\0’ and p1 points to p thus p1doesnot print anything.

23.
```
#include <stdio.h>
#define a 10
main()
{
    #define a 50
    printf("%d",a);
}
```

Answer:

50

Explanation:

The preprocessor directives can be redefined anywhere in the program. So the most recently assigned value will be taken.

24. #define clrscr() 100
```
main()
{
    clrscr();
    printf("%d\n",clrscr());
}
```
Answer:
100
Explanation:
Preprocessor executes as a seperate pass before the execution of the compiler. So textual replacement of clrscr() to 100 occurs.The input program to compiler looks like this :
```
main()
{
   100;
   printf("%d\n",100);
}
```
Note:
100; is an executable statement but with no action. So it doesn't give any problem

25. main()
```
{
    printf("%p",main);
}
```
Answer:
Some address will be printed.
Explanation:
Function names are just addresses (just like array names are addresses).
main() is also a function. So the address of function main will be printed. %p in printf specifies that the argument is an address. They are printed as hexadecimal numbers.

26. main()
```
{
clrscr();
}
clrscr();
```
Answer:
No output/error
Explanation:
The first clrscr() occurs inside a function. So it becomes a function call. In the second clrscr(); is a function declaration (because it is not inside any function).

27. enum colors {BLACK,BLUE,GREEN}
```
main()
    {
    printf("%d..%d..%d",BLACK,BLUE,GREEN);

    return(1);
    }
```
Answer:
0..1..2
Explanation:
enum assigns numbers starting from 0, if not explicitly defined.

28. void main()
```
    {
    char far *farther,*farthest;
    printf("%d..%d",sizeof(farther),sizeof(farthest));
    }
```
Answer:
4..2
Explanation:
the second pointer is of char type and not a far pointer

29. main()
```
    {
    int i=400,j=300;
    printf("%d..%d");
    }
```
Answer:
400..300
Explanation:
printf takes the values of the first two assignments of the program. Any number of printf's may be given. All of them take only the first two values. If more number of assignments given in the program,then printf will take garbage values.

30) main()
```
    {
    char *p;
    p="Hello";
    printf("%c\n",*&*p);
    }
```
Answer:
H
Explanation:
* is a dereference operator & is a reference operator. They can be applied any number of times provided it is meaningful. Here p points to the first character in the string "Hello". *p dereferences it and so its value is H. Again & references it to an address and * dereferences it to the value H.

31. char *someFun1()

```
{
char temp[ ] = “string";
return temp;
}
char *someFun2()
{
char temp[ ] = {‘s’, ‘t’,’r’,’i’,’n’,’g’};
return temp;
}
int main()
{
puts(someFun1());
puts(someFun2());
}
```

Answer:
Garbage values.
Explanation:
Both the functions suffer from the problem of dangling pointers. In someFun1() temp is a character array and so the space for it is allocated in heap and is initialized with character string “string”. This is created dynamically as the function is called, so is also deleted dynamically on exiting the function so the string data is not available in the calling function main() leading to print some garbage values. The function someFun2() also suffers from the same problem but the problem can be easily identified in this case.

32) main()

```
{
  int i=1;
  while (i<=5)
  {
    printf("%d",i);
    if (i>2)
      goto here;
    i++;
  }
}
fun()
{
  here:
    printf("PP");
}
```

Answer:
Compiler error: Undefined label 'here' in function main
Explanation:
Labels have functions scope, in other words The scope of the labels is limited to functions . The label 'here' is available in function fun() Hence it is not visible in function main.

33) main()
```
{
  static char names[5][20]={"pascal","ada","cobol","fortran","perl"};
   int i;
   char *t;
   t=names[3];
   names[3]=names[4];
   names[4]=t;
   for (i=0;i<=4;i++)
      printf("%s",names[i]);
}
```
Answer:
Compiler error: Lvalue required in function main
Explanation:
Array names are pointer constants. So it cannot be modified.

34) void main()
```
{
     int i=5;
     printf("%d",i++ + ++i);
}
```
Answer:
Output Cannot be predicted exactly.
Explanation:
Side effects are involved in the evaluation of i

35) void main()
```
{
     int i=5;
     printf("%d",i+++++i);
}
```
Answer:
Compiler Error
Explanation:
The expression i+++++i is parsed as i ++ ++ + i which is an illegal combination of operators.

36) #include<stdio.h>
```
main()
{
int i=1,j=2;
switch(i)
 {
 case 1:  printf("GOOD");
          break;
 case j:  printf("BAD");
         break;
 }
```

```
        }
```

Answer:

Compiler Error: Constant expression required in function main.

Explanation:

The case statement can have only constant expressions (this implies that we cannot use variable names directly so an error).

Note: Enumerated types can be used in case statements.

37)
```
main()
        {
        int i;
        printf("%d",scanf("%d",&i));  // value 10 is given as input here
        }
```

Answer:

1

Explanation:

Scanf returns number of items successfully read and not 1/0. Here 10 is given as input which should have been scanned successfully. So number of items read is 1.

38)
```
#define f(g,g2) g##g2
        main()
        {
        int var12=100;
        printf("%d",f(var,12));
    }
```

Answer:

100

39)
```
main()
        {
        int i=0;

        for(;i++;printf("%d",i)) ;
        printf("%d",i);
        }
```

Answer:

1

Explanation:

before entering into the for loop the checking condition is "evaluated". Here it evaluates to 0 (false) and comes out of the loop, and i is incremented (note the semicolon after the for loop).

40)
```
#include<stdio.h>
        main()
        {
         char s[]={'a','b','c','\n','c','\0'};
         char *p,*str,*str1;
         p=&s[3];
         str=p;
         str1=s;
```

```
  printf("%d",++*p + ++*str1-32);
 }
```

Answer:

M

Explanation:

p is pointing to character '\n'.str1 is pointing to character 'a' ++*p meAnswer:"p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10. then it is incremented to 11. the value of ++*p is 11. ++*str1 meAnswer:"str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98. both 11 and 98 is added and result is subtracted from 32.

i.e. (11+98-32)=77("M");

41)
```
#include<stdio.h>
main()
{
 struct xx
  {
    int x=3;
    char name[]="hello";
  };
struct xx *s=malloc(sizeof(struct xx));
printf("%d",s->x);
printf("%s",s->name);
}
```

Answer:

Compiler Error

Explanation:

Initialization should not be done for structure members inside the structure declaration

42)
```
#include<stdio.h>
main()
{
struct xx
 {
       int x;
       struct yy
        {
          char s;
          struct xx *p;
        };
      struct yy *q;
  };
 }
```

Answer:

Compiler Error

Explanation:

in the end of nested structure yy a member have to be declared.

43) main()
```
{
 extern int i;
 i=20;
 printf("%d",sizeof(i));
}
```
Answer:
Linker error: undefined symbol '_i'.
Explanation:
extern declaration specifies that the variable i is defined somewhere else. The compiler passes the external variable to be resolved by the linker. So compiler doesn't find an error. During linking the linker searches for the definition of i. Since it is not found the linker flags an error.

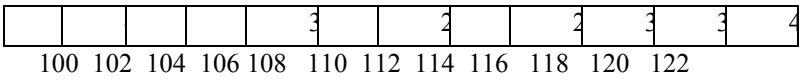
44) main()
```
{
printf("%d", out);
}
int out=100;
```
Answer:
Compiler error: undefined symbol out in function main.
Explanation:
The rule is that a variable is available for use from the point of declaration. Even though a is a global variable, it is not available for main. Hence an error.

45) main()
```
{
 extern out;
 printf("%d", out);
}
 int out=100;
```
Answer:
100
Explanation:
This is the correct way of writing the previous program.

46) main()
```
{
 show();
}
void show()
{
 printf("I'm the greatest");
}
```
Answer:
Compier error: Type mismatch in redeclaration of show.
Explanation:

When the compiler sees the function show it doesn't know anything about it. So the default return type (ie, int) is assumed. But when compiler sees the actual definition of show mismatch occurs since it is declared as void. Hence the error.
The solutions are as follows:
1. declare void show() in main() .
2. define show() before main().
3. declare extern void show() before the use of show().

47)
```
main( )
{
        int a[2][3][2] = {{{2,4},{7,8},{3,4}},{{2,2},{2,3},{3,4}}};
        printf("%u %u %u %d \n",a,*a,**a,***a);
        printf("%u %u %u %d \n",a+1,*a+1,**a+1,***a+1);
}
```
Answer:
100, 100, 100, 2
114, 104, 102, 3
Explanation:
The given array is a 3-D one. It can also be viewed as a 1-D array.



thus, for the first printf statement a, *a, **a give address of first element . since the indirection ***a gives the value. Hence, the first line of the output.
for the second printf a+1 increases in the third dimension thus points to value at 114, *a+1 increments in second dimension thus points to 104, **a +1 increments the first dimension thus points to 102 and ***a+1 first gets the value at first location and then increments it by 1. Hence, the output.

48)
```
main( )
{
 int a[ ] = {10,20,30,40,50},j,*p;
 for(j=0; j<5; j++)
     {
     printf("%d" ,*a);
     a++;
     }
     p = a;
      for(j=0; j<5; j++)
          {
                printf("%d " ,*p);
                p++;
          }
      }
 }
```

Answer:

Compiler error: lvalue required.

Explanation:
Error is in line with statement a++. The operand must be an lvalue and may be of any of scalar type for the any operator, array name only when subscripted is an lvalue. Simply array name is a non-modifiable lvalue.

49) main( )
```
{
static int  a[ ]  = {0,1,2,3,4};
int  *p[ ] = {a,a+1,a+2,a+3,a+4};
int  **ptr =  p;
ptr++;
printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
*ptr++;
printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
*++ptr;
printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
++*ptr;
printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
}
```
Answer:
111
222
333
344
Explanation:
Let us consider the array and the two pointers with some address

a

0	1	2	3	4
100	102	104	106	108

p

1	1	1	1	1
0	0	0	0	0
0	2	4	6	8
1000	1002	1004	1006	1008

ptr

1
0
0
0
2000

After execution of the instruction ptr++ value in ptr becomes 1002, if scaling factor for integer is 2 bytes. Now ptr – p is value in ptr – starting location of array p, (1002 – 1000) / (scaling factor) = 1, *ptr – a = value at address pointed by ptr – starting value of array a, 1002 has a value 102 so the value is (102 – 100)/(scaling factor) = 1, **ptr is the value stored in the location pointed by the pointer of ptr = value pointed

by value pointed by 1002 = value pointed by 102 = 1. Hence the output of the firs printf is 1, 1, 1.
After execution of *ptr++ increments value of the value in ptr by scaling factor, so it becomes1004. Hence, the outputs for the second printf are ptr – p = 2, *ptr – a = 2, **ptr = 2.
After execution of *++ptr increments value of the value in ptr by scaling factor, so it becomes1004. Hence, the outputs for the third printf are ptr – p = 3, *ptr – a = 3, **ptr = 3.
After execution of ++*ptr value in ptr remains the same, the value pointed by the value is incremented by the scaling factor. So the value in array p at location 1006 changes from 106 10 108,. Hence, the outputs for the fourth printf are ptr – p = 1006 – 1000 = 3, *ptr – a = 108 – 100 = 4, **ptr = 4.

50) main( )
```
{
 char  *q;
 int  j;
 for (j=0; j<3; j++) scanf("%s" ,(q+j));
 for (j=0; j<3; j++) printf("%c" ,*(q+j));
 for (j=0; j<3; j++) printf("%s" ,(q+j));
}
```
Explanation:
Here we have only one pointer to type char and since we take input in the same pointer thus we keep writing over in the same location, each time shifting the pointer value by 1. Suppose the inputs are MOUSE, TRACK and VIRTUAL. Then for the first input suppose the pointer starts at location 100 then the input one is stored as

M	O	U	S	E	\ 0

When the second input is given the pointer is incremented as j value becomes 1, so the input is filled in memory starting from 101.

M	T	R	A	C	K	\ 0

The third input starts filling from the location 102

M	T	V	I	R	T	U	A	L	\ 0

This is the final value stored .
The first printf prints the values at the position q, q+1 and q+2 = M T V
The second printf prints three strings starting from locations q, q+1, q+2
i.e MTVIRTUAL, TVIRTUAL and VIRTUAL.

51) main( )
```
{
 void *vp;
 char ch = 'g', *cp = "goofy";
 int j = 20;
 vp = &ch;
 printf("%c", *(char *)vp);
 vp = &j;
 printf("%d",*(int *)vp);
```

```
        vp = cp;
        printf("%s",(char *)vp + 3);}
```

Answer:

g20fy

Explanation:

Since a void pointer is used it can be type casted to any other type pointer. vp = &ch stores address of char ch and the next statement prints the value stored in vp after type casting it to the proper data type pointer. the output is 'g'. Similarly the output from second printf is '20'. The third printf statement type casts it to print the string from the 4th value hence the output is 'fy'.

52)
```
main ( )
        {
        static char *s[ ]  = {"black", "white", "yellow", "violet"};
        char **ptr[ ] = {s+3, s+2, s+1, s}, ***p;
        p = ptr;
        **++p;
        printf("%s",*--*++p + 3);
        }
```

Answer:

ck

Explanation:

In this problem we have an array of char pointers pointing to start of 4 strings. Then we have ptr which is a pointer to a pointer of type char and a variable p which is a pointer to a pointer to a pointer of type char. p hold the initial value of ptr, i.e. p = s+3. The next statement increment value in p by 1 , thus now value of p = s+2. In the printf statement the expression is evaluated *++p causes gets value s+1 then the pre decrement is executed and we get s+1 – 1 = s . the indirection operator now gets the value from the array of s and adds 3 to the starting address. The string is printed starting from this position. Thus, the output is 'ck'.

53)
```
main()
{
        int  i, n;
        char *x = "girl";
        n = strlen(x);
        *x = x[n];
        for(i=0; i<n; ++i)
          {
        printf("%s\n",x);
        x++;
          }
}
```

Answer:

(blank space)
irl
rl
l

Explanation:

Here a string (a pointer to char) is initialized with a value “girl”. The strlen function returns the length of the string, thus n has a value 4. The next statement assigns value at the nth location (‘\0’) to the first location. Now the string becomes “\0irl” . Now the printf statement prints the string after each iteration it increments it starting position. Loop starts from 0 to 4. The first time x[0] = ‘\0’ hence it prints nothing and pointer value is incremented. The second time it prints from x[1] i.e “irl” and the third time it prints “rl” and the last time it prints “l” and the loop terminates.

54) int i,j;
```
for(i=0;i<=10;i++)
{
j+=5;
assert(i<5);
}
```
Answer:
Runtime error: Abnormal program termination.
assert failed (i<5), <file name>,<line number>
Explanation:
asserts are used during debugging to make sure that certain conditions are satisfied. If assertion fails, the program will terminate reporting the same. After debugging use, #undef NDEBUG and this will disable all the assertions from the source code. Assertionis a good debugging tool to make use of.

55) main()
```
{
int i=-1;
+i;
printf("i = %d, +i = %d \n",i,+i);
}
```
Answer:
i = -1, +i = -1
Explanation:
Unary + is the only dummy operator in C. Where-ever it comes you can just ignore it just because it has no effect in the expressions (hence the name dummy operator).

56) What are the files which are automatically opened when a C file is executed?
Answer:
stdin, stdout, stderr (standard input,standard output,standard error).

57) what will be the position of the file marker?
a: fseek(ptr,0,SEEK_SET);
b: fseek(ptr,0,SEEK_CUR);
Answer :
a: The SEEK_SET sets the file position marker to the starting of the file.
b: The SEEK_CUR sets the file position marker to the current position of the file.

58) main()
```
{
char name[10],s[12];
scanf(" \"%[^\"]\"",s);
```

```
}
```
How scanf will execute?

Answer:
First it checks for the leading white space and discards it.Then it matches with a quotation mark and then it reads all character upto another quotation mark.

59) What is the problem with the following code segment?
```
while ((fgets(receiving array,50,file_ptr)) != EOF)
        ;
```
Answer & Explanation:
fgets returns a pointer. So the correct end of file check is checking for != NULL.

60)
```
main()
{
main();
}
```
Answer:
Runtime error : Stack overflow.
Explanation:
main function calls itself again and again. Each time the function is called its return address is stored in the call stack. Since there is no condition to terminate the function call, the call stack overflows at runtime. So it terminates the program and results in an error.

61)
```
main()
{
char *cptr,c;
void *vptr,v;
c=10;  v=0;
cptr=&c; vptr=&v;
printf("%c%v",c,v);
}
```
Answer:
Compiler error (at line number 4): size of v is Unknown.
Explanation:
You can create a variable of type void * but not of type void, since void is an empty type. In the second line you are creating variable vptr of type void * and v of type void hence an error.

62)
```
main()
{
char *str1="abcd";
char str2[]="abcd";
printf("%d %d %d",sizeof(str1),sizeof(str2),sizeof("abcd"));
}
```
Answer:
2 5 5
Explanation:

In first sizeof, str1 is a character pointer so it gives you the size of the pointer variable. In second sizeof the name str2 indicates the name of the array whose size is 5 (including the '\0' termination character). The third sizeof is similar to the second one.

63)
```
main()
{
char not;
not=!2;
printf("%d",not);
}
```
Answer:
0
Explanation:
! is a logical operator. In C the value 0 is considered to be the boolean value FALSE, and any non-zero value is considered to be the boolean value TRUE. Here 2 is a non-zero value so TRUE. !TRUE is FALSE (0) so it prints 0.

64)
```
#define FALSE -1
#define TRUE   1
#define NULL   0
main() {
  if(NULL)
        puts("NULL");
  else if(FALSE)
        puts("TRUE");
  else
        puts("FALSE");
  }
```
Answer:
TRUE
Explanation:
The input program to the compiler after processing by the preprocessor is,
```
main(){
if(0)
        puts("NULL");
else if(-1)
        puts("TRUE");
else
        puts("FALSE");
}
```
Preprocessor doesn't replace the values given inside the double quotes. The check by if condition is boolean value false so it goes to else. In second if -1 is boolean value true hence "TRUE" is printed.

65)
```
main()
{
int k=1;
printf("%d==1 is ""%s",k,k==1?"TRUE":"FALSE");
}
```
Answer:

1==1 is TRUE

Explanation:
When two strings are placed together (or separated by white-space) they are concatenated (this is called as "stringization" operation). So the string is as if it is given as "%d==1 is %s". The conditional operator(?:) evaluates to "TRUE".

66)
```
main()
{
int y;
scanf("%d",&y); // input given is 2000
if( (y%4==0 && y%100 != 0) || y%100 == 0 )
    printf("%d is a leap year");
else
    printf("%d is not a leap year");
}
```
Answer:
2000 is a leap year
Explanation:
An ordinary program to check if leap year or not.

67)
```
#define max 5
#define int arr1[max]
main()
{
typedef char arr2[max];
arr1 list={0,1,2,3,4};
arr2 name="name";
printf("%d %s",list[0],name);
}
```
Answer:
Compiler error (in the line arr1 list = {0,1,2,3,4})
Explanation:
arr2 is declared of type array of size 5 of characters. So it can be used to declare the variable name of the type arr2. But it is not the case of arr1. Hence an error.
Rule of Thumb:
#defines are used for textual replacement whereas typedefs are used for declaring new types.

68)
```
int i=10;
main()
{
 extern int i;
          {
    int i=20;
          {
          const volatile unsigned i=30;
          printf("%d",i);
```

```
            }
        printf("%d",i);
      }
   printf("%d",i);     }
```

Answer:

30,20,10

Explanation:

'{' introduces new block and thus new scope. In the innermost block i is declared as, const volatile unsigned hich is a valid declaration. i is assumed of type int. So printf prints 30. In the next block, i has value 20 and so printf prints 20. In the outermost block, i is declared as extern, so no storage space is allocated for it. After compilation is over the linker resolves it to global variable i (since it is the only variable visible there). So it prints i's value as 10.

69)

```
main()
{
   int *j;
   {
    int i=10;
    j=&i;
    }
    printf("%d",*j);
     }
```

Answer:

10

Explanation:

The variable i is a block level variable and the visibility is inside that block only. But the lifetime of i is lifetime of the function so it lives upto the exit of main function. Since the i is still allocated space, *j prints the value stored in i since j points i.

70)

```
main()
{
int i=-1;
-i;
printf("i = %d, -i = %d \n",i,-i);
}
```

Answer:

i = -1, -i = 1

Explanation:

-i is executed and this execution doesn't affect the value of i. In printf first you just print the value of i. After that the value of the expression -i = -(-1) is printed.

71)

```
#include<stdio.h>
      main()
       {
         const int i=4;
         float j;
         j = ++i;
         printf("%d  %f", i,++j);
       }
```

Answer:
Compiler error
Explanation:
i is a constant. you cannot change the value of constant

72) #include<stdio.h>
```
main()
{
 int a[2][2][2] = { {10,2,3,4}, {5,6,7,8}  };
 int *p,*q;
 p=&a[2][2][2];
 *q=***a;
 printf("%d..%d",*p,*q);
}
```
Answer:
garbagevalue..1
Explanation:
p=&a[2][2][2] you declare only two 2D arrays. but you are trying to access the third 2D(which you are not declared) it will print garbage values. *q=***a starting address of a is assigned integer pointer. now q is pointing to starting address of a.if you print *q meAnswer:it will print first element of 3D array.

73) #include<stdio.h>
```
main()
  {
   register i=5;
   char j[]= "hello";
    printf("%s  %d",j,i);
}
```
Answer:
hello 5
Explanation:
if you declare i as register compiler will treat it as ordinary integer and it will take integer value. i value may be stored either in register or in memory.

74) main()
```
{
 int i=5,j=6,z;
 printf("%d",i+++j);
}
```
Answer:
11
Explanation:
the expression i+++j is treated as (i++ + j)

76) struct aaa{
```
struct aaa *prev;
int i;
struct aaa *next;
};
```

```
main()
{
 struct aaa abc,def,ghi,jkl;
 int x=100;
 abc.i=0;abc.prev=&jkl;
 abc.next=&def;
 def.i=1;def.prev=&abc;def.next=&ghi;
 ghi.i=2;ghi.prev=&def;
 ghi.next=&jkl;
 jkl.i=3;jkl.prev=&ghi;jkl.next=&abc;
 x=abc.next->next->prev->next->i;
 printf("%d",x);
}
```

Answer:

2

Explanation:

above all statements form a double circular linked list;
abc.next->next->prev->next->i
this one points to "ghi" node the value of at particular node is 2.

77)
```
struct point
 {
 int x;
 int y;
 };
struct point origin,*pp;
main()
{
pp=&origin;
printf("origin is(%d%d)\n",(*pp).x,(*pp).y);
printf("origin is (%d%d)\n",pp->x,pp->y);
}
```

Answer:

```
origin is(0,0)
origin is(0,0)
```

Explanation:

pp is a pointer to structure. we can access the elements of the structure either with arrow mark or with indirection operator.

Note: Since structure point is globally declared x & y are initialized as zeroes

78)
```
main()
 {
 int i=_l_abc(10);
  printf("%d\n",--i);
 }
 int _l_abc(int i)
 {
  return(i++);
 }
```

Answer:
9
Explanation:
return(i++) it will first return i and then increments. i.e. 10 will be returned.

79) main()
```
{
char *p;
int *q;
long *r;
p=q=r=0;
p++;
q++;
r++;
printf("%p...%p...%p",p,q,r);
}
```
Answer:
0001...0002...0004
Explanation:
++ operator when applied to pointers increments address according to their corresponding data-types.

80) main()
```
{
char c=' ',x,convert(z);
getc(c);
if((c>='a') && (c<='z'))
x=convert(c);
printf("%c",x);
}
convert(z)
{
 return z-32;
}
```
Answer:
Compiler error
Explanation:
declaration of convert and format of getc() are wrong.

81) main(int argc, char **argv)
```
{
printf("enter the character");
getchar();
sum(argv[1],argv[2]);
}
sum(num1,num2)
int num1,num2;
{
return num1+num2;
}
```
Answer:

Compiler error.
Explanation:
argv[1] & argv[2] are strings. They are passed to the function sum without converting it to integer values.

82) # include <stdio.h>
```
int one_d[]={1,2,3};
main()
{
 int *ptr;
 ptr=one_d;
 ptr+=3;
 printf("%d",*ptr);
}
```
Answer:
garbage value
Explanation:
ptr pointer is pointing to out of the array range of one_d.

83) # include<stdio.h>
```
aaa() {
  printf("hi");
 }
bbb(){
 printf("hello");
 }
ccc(){
 printf("bye");
 }
main()
{
  int (*ptr[3])();
  ptr[0]=aaa;
  ptr[1]=bbb;
  ptr[2]=ccc;
  ptr[2]();
}
```
Answer:
bye
Explanation:
ptr is array of pointers to functions of return type int.ptr[0] is assigned to address of the function aaa. Similarly ptr[1] and ptr[2] for bbb and ccc respectively. ptr[2]() is in effect of writing ccc(), since ptr[2] points to ccc.

85) #include<stdio.h>
```
main()
{
FILE *ptr;
char i;
ptr=fopen("zzz.c","r");
```

```
while((i=fgetch(ptr))!=EOF)
printf("%c",i);
}
```

Answer:

contents of zzz.c followed by an infinite loop

Explanation:

The condition is checked against EOF, it should be checked against NULL.

86)
```
main()
{
 int i =0;j=0;
 if(i && j++)
      printf("%d..%d",i++,j);
printf("%d..%d,i,j);
}
```

Answer:

0..0

Explanation:

The value of i is 0. Since this information is enough to determine the truth value of the boolean expression. So the statement following the if statement is not executed. The values of i and j remain unchanged and get printed.

87)
```
main()
{
 int i;
 i = abc();
 printf("%d",i);
}
abc()
{
 _AX = 1000;
}
```

Answer:

1000

Explanation:

Normally the return value from the function is through the information from the accumulator. Here _AH is the pseudo global variable denoting the accumulator. Hence, the value of the accumulator is set 1000 so the function returns value 1000.

88)
```
int i;
main()
{
int t;
for ( t=4;scanf("%d",&i)-t;printf("%d\n",i))
      printf("%d--",t--);
 }
// If the inputs are 0,1,2,3 find the o/p
```

Answer:

4--0

3--1
2--2

Explanation:
Let us assume some x= scanf("%d",&i)-t the values during execution will be,

t	i	x
4	0	-4
3	1	-2
2	2	0

89) main(){
int a= 0;int b = 20;char x =1;char y =10;
if(a,b,x,y)
printf("hello");
}
Answer:
hello
Explanation:
The comma operator has associativity from left to right. Only the rightmost value is returned and the other values are evaluated and ignored. Thus the value of last variable y is returned to check in if. Since it is a non zero value if becomes true so, "hello" will be printed.

90) main(){
unsigned int i;
for(i=1;i>-2;i--)
printf("c aptitude");
}
Explanation:
i is an unsigned integer. It is compared with a signed value. Since the both types doesn't match, signed is promoted to unsigned value. The unsigned equivalent of -2 is a huge value so condition becomes false and control comes out of the loop.

91) In the following pgm add a stmt in the function fun such that the address of 'a' gets stored in 'j'.
main(){
int * j;
void fun(int **);
fun(&j);
}
void fun(int **k) {
int a =0;
/* add a stmt here*/
}
Answer:
*k = &a
Explanation:
The argument of the function is a pointer to a pointer.

92) What are the following notations of defining functions known as?

i.
```
int abc(int a,float b)
        {
    /* some code */
 }
```

ii.
```
int abc(a,b)
 int a; float b;
        {
        /* some code*/
        }
```

Answer:

i. ANSI C notation

ii. Kernighan & Ritche notation

93)
```
main()
{
char *p;
p="%d\n";
      p++;
      p++;
      printf(p-2,300);
}
```

Answer:

300

Explanation:

The pointer points to % since it is incremented twice and again decremented by 2, it points to '%d\n' and 300 is printed.

94)
```
main(){
 char a[100];
 a[0]='a';a[1]]='b';a[2]='c';a[4]='d';
 abc(a);
}
abc(char a[]){
 a++;
printf("%c",*a);
 a++;
 printf("%c",*a);
}
```

Explanation:

The base address is modified only in function and as a result a points to 'b' then after incrementing to 'c' so bc will be printed.

95) func(a,b)

```
int a,b;
{
 return( a= (a==b) );
}
main()
{
int process(),func();
printf("The value of process is %d !\n ",process(func,3,6));
}
process(pf,val1,val2)
int (*pf) ();
int val1,val2;
{
return((*pf) (val1,val2));
 }
```

Answer:

The value if process is 0 !

Explanation:

The function 'process' has 3 parameters - 1, a pointer to another function 2 and 3, integers. When this function is invoked from main, the following substitutions for formal parameters take place: func for pf, 3 for val1 and 6 for val2. This function returns the result of the operation performed by the function 'func'. The function func has two integer parameters. The formal parameters are substituted as 3 for a and 6 for b. since 3 is not equal to 6, a==b returns 0. therefore the function returns 0 which in turn is returned by the function 'process'.

96) void main()

```
{
    static int i=5;
    if(--i){
            main();
            printf("%d ",i);
    }
}
```

Answer:

0 0 0 0

Explanation:

The variable "I" is declared as static, hence memory for I will be allocated for only once, as it encounters the statement. The function main() will be called recursively unless I becomes equal to 0, and since main() is recursively called, so the value of static I ie., 0 will be printed every time the control is returned.

97) void main()

```
{
    int k=ret(sizeof(float));
```

```
    printf("\n here value is %d",++k);
}


int ret(int ret)
{
    ret += 2.5;
    return(ret);
}
```

Answer:

Here value is 7

Explanation:

The int ret(int ret), ie., the function name and the argument name can be the same. Firstly, the function ret() is called in which the sizeof(float) ie., 4 is passed, after the first expression the value in ret will be 6, as ret is integer hence the value stored in ret will have implicit type conversion from float to int. The ret is returned in main() it is printed after and preincrement.

98)

```
void main()
{
    char a[]="12345\0";
    int i=strlen(a);
    printf("here in 3 %d\n",++i);
}
```

Answer:

here in 3 6

Explanation:

The char array 'a' will hold the initialized string, whose length will be counted from 0 till the null character. Hence the 'I' will hold the value equal to 5, after the pre-increment in the printf statement, the 6 will be printed.

99)

```
void main()
{
    unsigned giveit=-1;
    int gotit;
    printf("%u ",++giveit);
    printf("%u \n",gotit=--giveit);
}
```

Answer:

0 65535

Explanation:

100)

```
void main()
    {
    int i;
    char a[]="\0";
    if(printf("%s\n",a))
            printf("Ok here \n");
    else
            printf("Forget it\n");
```

```
    }
```

Answer:

Ok here

Explanation:

Printf will return how many characters does it print. Hence printing a null character returns 1 which makes the if statement true, thus "Ok here" is printed.

101)
```
void main()
    {
    void *v;
    int integer=2;
    int *i=&integer;
    v=i;
    printf("%d",(int*)*v);
    }
```

Answer:

Compiler Error. We cannot apply indirection on type void*.

Explanation:

Void pointer is a generic pointer type. No pointer arithmetic can be done on it. Void pointers are normally used for,

- Passing generic pointers to functions and returning such pointers.
- As a intermediate pointer type.
- Used when the exact pointer type will be known at a later point of time.

102)
```
void main()
    {
            int i=i++,j=j++,k=k++;
            printf("%d%d%d",i,j,k);
    }
```

Answer:

Garbage values.

Explanation:

An identifier is available to use in program code from the point of its declaration.

So expressions such as i = i++ are valid statements. The i, j and k are automatic variables and so they contain some garbage value. *Garbage in is garbage out (GIGO).*

103)
```
        void main()
        {
        static int i=i++, j=j++, k=k++;
        printf("i = %d j = %d k = %d", i, j, k);
        }
```

Answer:

i = 1 j = 1 k = 1

Explanation:

Since static variables are initialized to zero by default.

```
104)    void main()
        {
        while(1){
        if(printf("%d",printf("%d")))
                break;
        else
                continue;
                }
        }
```

Answer:

Garbage values

Explanation:

The inner printf executes first to print some garbage value. The printf returns no of characters printed and this value also cannot be predicted. Still the outer printf prints something and so returns a non-zero value. So it encounters the break statement and comes out of the while statement.

```
104)    main()
        {
        unsigned int i=10;
        while(i-->=0)
        printf("%u ",i);
        }
```

Answer:

10 9 8 7 6 5 4 3 2 1 0 65535 65534…..

Explanation:

Since i is an unsigned integer it can never become negative. So the expression i-- >=0 will always be true, leading to an infinite loop.

```
105)#include<conio.h>
    main()
    {
    int x,y=2,z,a;
    if(x=y%2) z=2;
    a=2;
    printf("%d %d ",z,x);
    }
```

Answer:

Garbage-value 0

Explanation:

The value of y%2 is 0. This value is assigned to x. The condition reduces to if (x) or in other words if(0) and so z goes uninitialized.

Thumb Rule: **C**heck all control paths to write bug free code.

```
106)main()
    {
```

```
int a[10];
printf("%d",*a+1-*a+3);
}
```

Answer:
4
Explanation:
*a and -*a cancels out. The result is as simple as 1 + 3 = 4 !

```
107) #define prod(a,b) a*b
main()
{
    int x=3,y=4;
    printf("%d",prod(x+2,y-1));
}
```
Answer:
10
Explanation:
The macro expands and evaluates to as:
x+2*y-1 => x+(2*y)-1 => 10

```
108) main()
    {
    unsigned int i=65000;
    while(i++!=0);
    printf("%d",i);
    }
```
Answer:
1
Explanation:
Note the semicolon after the while statement. When the value of i becomes 0 it comes out of while loop. Due to post-increment on i the value of i while printing is 1.

```
109)        main()
            {
                int i=0;
                while(+(+i--)!=0)
                    i-=i++;
                printf("%d",i);
            }
```
Answer:
-1
Explanation:
Unary + is the only dummy operator in C. So it has no effect on the expression and now the while loop is, while(i--!=0) which is false and so breaks out of while loop. The value –1 is printed due to the post-decrement operator.

```
113)        main()
            {
            float f=5,g=10;
```

```
        enum{i=10,j=20,k=50};
        printf("%d\n",++k);
        printf("%f\n",f<<2);
        printf("%lf\n",f%g);
                printf("%lf\n",fmod(f,g));
        }
```

Answer:

Line no 5: Error: Lvalue required
Line no 6: Cannot apply leftshift to float
Line no 7: Cannot apply mod to float

Explanation:

Enumeration constants cannot be modified, so you cannot apply ++.
Bit-wise operators and % operators cannot be applied on float values.
fmod() is to find the modulus values for floats as % operator is for ints.

110)

```
main()
{
int i=10;
void pascal f(int,int,int);
     f(i++,i++,i++);
printf(" %d",i);
}
void pascal f(integer :i,integer:j,integer :k)
     {
     write(i,j,k);
     }
```

Answer:

Compiler error: unknown type integer
Compiler error: undeclared function write

Explanation:

Pascal keyword doesn't mean that pascal code can be used. It means that the function follows Pascal argument passing mechanism in calling the functions.

111)

```
void pascal f(int i,int j,int k)
        {
        printf("%d %d %d",i, j, k);
        }
        void cdecl f(int i,int j,int k)
        {
        printf("%d %d %d",i, j, k);
        }
        main()
        {
             int i=10;
             f(i++,i++,i++);
             printf(" %d\n",i);
             i=10;
             f(i++,i++,i++);
             printf(" %d",i);
        }
```

Answer:
10 11 12 13
12 11 10 13

Explanation:
Pascal argument passing mechanism forces the arguments to be called from left to right. cdecl is the normal C argument passing mechanism where the arguments are passed from right to left.

112). What is the output of the program given below

```
main()
  {
    signed char i=0;
    for(;i>=0;i++) ;
    printf("%d\n",i);
  }
```

Answer
-128
Explanation
Notice the semicolon at the end of the for loop. THe initial value of the i is set to 0. The inner loop executes to increment the value from 0 to 127 (the positive range of char) and then it rotates to the negative value of -128. The condition in the for loop fails and so comes out of the for loop. It prints the current value of i that is -128.

113)
```
main()
  {
    unsigned char i=0;
    for(;i>=0;i++) ;
    printf("%d\n",i);
  }
```

Answer
infinite loop
Explanation
The difference between the previous question and this one is that the char is declared to be unsigned. So the i++ can never yield negative value and i>=0 never becomes false so that it can come out of the for loop.

114)
```
main()
  {
      char i=0;
      for(;i>=0;i++) ;
      printf("%d\n",i);

  }
```

Answer:
Behavior is implementation dependent.
Explanation:
The detail if the char is signed/unsigned by default is implementation dependent. If the implementation treats the char to be signed by default the program will print –128

and terminate. On the other hand if it considers char to be unsigned by default, it goes to infinite loop.

Rule:

You can write programs that have implementation dependent behavior. But dont write programs that depend on such behavior.

115) Is the following statement a declaration/definition. Find what does it mean?

```
int (*x)[10];
```

Answer

Definition.
x is a pointer to array of(size 10) integers.
Apply clock-wise rule to find the meaning of this definition.

116). What is the output for the program given below

```
typedef enum errorType{warning, error, exception,}error;
 main()
{
   error g1;
   g1=1;
   printf("%d",g1);
 }
```

Answer

Compiler error: Multiple declaration for error

Explanation

The name error is used in the two meanings. One means that it is a enumerator constant with value 1. The another use is that it is a type name (due to typedef) for enum errorType. Given a situation the compiler cannot distinguish the meaning of error to know in what sense the error is used:

```
error g1;
g1=error;
// which error it refers in each case?
```

When the compiler can distinguish between usages then it will not issue error (in pure technical terms, names can only be overloaded in different namespaces).

Note: the extra comma in the declaration,
enum errorType{warning, error, exception,}
is not an error. An extra comma is valid and is provided just for programmer's convenience.

117)

```
typedef struct error{int warning, error, exception;}error;
 main()
{
   error g1;
   g1.error =1;
   printf("%d",g1.error);
 }
```

Answer

1

Explanation

The three usages of name errors can be distinguishable by the compiler at any instance, so valid (they are in different namespaces).
Typedef struct error{int warning, error, exception;}error;
This error can be used only by preceding the error by struct kayword as in:
struct error someError;
typedef struct error{int warning, error, exception;}error;
This can be used only after . (dot) or -> (arrow) operator preceded by the variable name as in :
g1.error =1;
printf("%d",g1.error);
typedef struct error{int warning, error, exception;}error;
This can be used to define variables without using the preceding struct keyword as in:
error g1;
Since the compiler can perfectly distinguish between these three usages, it is perfectly legal and valid.

Note
This code is given here to just explain the concept behind. In real programming don't use such overloading of names. It reduces the readability of the code. Possible doesn't mean that we should use it!

118)
```
#ifdef something
int some=0;
#endif

main()
{
int thing = 0;
printf("%d %d\n", some ,thing);
}
```

Answer:
Compiler error : undefined symbol some
Explanation:
This is a very simple example for conditional compilation. The name something is not already known to the compiler making the declaration
int some = 0;
effectively removed from the source code.

119)
```
#if something == 0
int some=0;
#endif

main()
{
int thing = 0;
printf("%d %d\n", some ,thing);
}
```

Answer

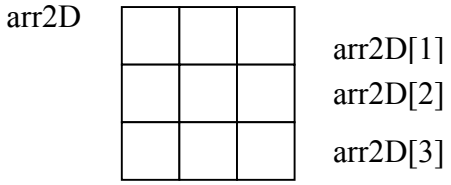
0 0

Explanation
This code is to show that preprocessor expressions are not the same as the ordinary expressions. If a name is not known the preprocessor treats it to be equal to zero.

120). What is the output for the following program

```
main()
    {
        int arr2D[3][3];
         printf("%d\n", ((arr2D==* arr2D)&&(* arr2D == arr2D[0])) );
    }
```

Answer
1
Explanation
This is due to the close relation between the arrays and pointers. N dimensional arrays are made up of (N-1) dimensional arrays.
arr2D is made up of a 3 single arrays that contains 3 integers each .



The name arr2D refers to the beginning of all the 3 arrays. *arr2D refers to the start of the first 1D array (of 3 integers) that is the same address as arr2D. So the expression (arr2D == *arr2D) is true (1).
Similarly, *arr2D is nothing but *(arr2D + 0), adding a zero doesn't change the value/meaning. Again arr2D[0] is the another way of telling *(arr2D + 0). So the expression (*(arr2D + 0) == arr2D[0]) is true (1).
Since both parts of the expression evaluates to true the result is true(1) and the same is printed.

121)
```
void main()
    {
        if(~0 == (unsigned int)-1)
printf("You can answer this if you know how values are represented in memory");
    }
```

Answer:
You can answer this if you know how values are represented in memory
Explanation:
~ (tilde operator or bit-wise negation operator) operates on 0 to produce all ones to fill the space for an integer. –1 is represented in unsigned value as all 1's and so both are equal.

122)
```
int swap(int *a,int *b)
{
 *a=*a+*b;*b=*a-*b;*a=*a-*b;
}
main()
{
    int x=10,y=20;
    swap(&x,&y);
    printf("x= %d y = %d\n",x,y);
}
```
Answer

x = 20 y = 10

Explanation

This is one way of swapping two values. Simple checking will help understand this.

123)
```
main()
{
char *p = “ayqm”;
printf(“%c”,++*(p++));
}
```
Answer:

b

124)
```
main()
{
 int i=5;
 printf("%d",++i++);
}
```
Answer:

Compiler error: Lvalue required in function main

Explanation:

++i yields an rvalue. For postfix ++ to operate an lvalue is required.

125)
```
main()
{
char *p = “ayqm”;
char c;
c = ++*p++;
printf(“%c”,c);
}
```
Answer:

b

Explanation:

There is no difference between the expression ++*(p++) and ++*p++. Parenthesis just works as a visual clue for the reader to see which expression is first evaluated.

126)

```
int aaa() {printf(“Hi”);}
int bbb(){printf(“hello”);}
iny ccc(){printf(“bye”);}

main()
{
int ( * ptr[3]) ();
ptr[0] = aaa;
ptr[1] = bbb;
ptr[2] =ccc;
ptr[2]();
}
```

Answer:

bye

Explanation:

int (* ptr[3])() says that ptr is an array of pointers to functions that takes no arguments and returns the type int. By the assignment ptr[0] = aaa; it means that the first function pointer in the array is initialized with the address of the function aaa. Similarly, the other two array elements also get initialized with the addresses of the functions bbb and ccc. Since ptr[2] contains the address of the function ccc, the call to the function ptr[2]() is same as calling ccc(). So it results in printing "bye".

127)

```
main()
{
int i=5;
printf(“%d”,i=++i ==6);
}
```

Answer:

1

Explanation:

The expression can be treated as i = (++i==6), because == is of higher precedence than = operator. In the inner expression, ++i is equal to 6 yielding true(1). Hence the result.

128)

```
main()
{
     char p[ ]="%d\n";
p[1] = 'c';
printf(p,65);
}
```

Answer:

A

Explanation:

Due to the assignment p[1] = 'c' the string becomes, "%c\n". Since this string becomes the format string for printf and ASCII value of 65 is 'A', the same gets printed.

129) void (* abc(int, void (*def) ())) ();

Answer:
abc is a ptr to a function which takes 2 parameters .(a). an integer variable.(b).
a ptrto a funtion which returns void. the return type of the function is void.

Explanation:

Apply the clock-wise rule to find the result.

130)
```
main()
{
while (strcmp("some","some\0"))
printf("Strings are not equal\n");
}
```

Answer:

No output

Explanation:

Ending the string constant with \0 explicitly makes no difference. So "some" and "some\0" are equivalent. So, strcmp returns 0 (false) hence breaking out of the while loop.

131)
```
main()
{
char str1[] = {'s','o','m','e'};
char str2[] = {'s','o','m','e','\0'};
while (strcmp(str1,str2))
printf("Strings are not equal\n");
}
```

Answer:

"Strings are not equal"
"Strings are not equal"
….

Explanation:

If a string constant is initialized explicitly with characters, '\0' is not appended automatically to the string. Since str1 doesn't have null termination, it treats whatever the values that are in the following positions as part of the string until it randomly reaches a '\0'. So str1 and str2 are not the same, hence the result.

132)
```
main()
{
int i = 3;
for (;i++=0;) printf("%d",i);
}
```

Answer:

Compiler Error: Lvalue required.

Explanation:

As we know that increment operators return rvalues and hence it cannot appear on the left hand side of an assignment operation.

133)
```
void main()
{
int *mptr, *cptr;
mptr = (int*)malloc(sizeof(int));
printf("%d",*mptr);
int *cptr = (int*)calloc(sizeof(int),1);
printf("%d",*cptr);
}
```
Answer:

garbage-value 0

Explanation:

The memory space allocated by malloc is uninitialized, whereas calloc returns the allocated memory space initialized to zeros.

134)
```
void main()
{
static int i;
while(i<=10)
(i>2)?i++:i--;
    printf("%d", i);
}
```
Answer:

32767

Explanation:

Since i is static it is initialized to 0. Inside the while loop the conditional operator evaluates to false, executing i--. This continues till the integer value rotates to positive value (32767). The while condition becomes false and hence, comes out of the while loop, printing the i value.

135)
```
main()
{
    int i=10,j=20;
    j = i, j?(i,j)?i:j:j;
    printf("%d %d",i,j);
}
```
Answer:

10 10

Explanation:

The Ternary operator (? :) is equivalent for if-then-else statement. So the question can be written as:

```
if(i,j)
    {
        if(i,j)
            j = i;
        else
           j = j;
```

```
    }
  else
        j = j;
```

136) 1. const char *a;
2. char* const a;
3. char const *a;
-Differentiate the above declarations.

Answer:

1. 'const' applies to char * rather than 'a' (pointer to a constant char)

```
*a='F'      : illegal
a="Hi"      : legal
```

2. 'const' applies to 'a' rather than to the value of a (constant pointer to char)

```
*a='F'      : legal
a="Hi"      : illegal
```

3. Same as 1.

137)
```
main()
{
    int i=5,j=10;
    i=i&=j&&10;
    printf("%d %d",i,j);
}
```

Answer:

1 10

Explanation:

The expression can be written as i=(i&=(j&&10)); The inner expression (j&&10) evaluates to 1 because j==10. i is 5. i = 5&1 is 1. Hence the result.

138)
```
main()
{
    int i=4,j=7;
    j = j || i++ && printf("YOU CAN");
    printf("%d %d", i, j);
}
```

Answer:

4 1

Explanation:

The boolean expression needs to be evaluated only till the truth value of the expression is not known. j is not equal to zero itself means that the expression's truth value is 1. Because it is followed by || and *true || (anything) => true where (anything) will not be evaluated.* So the remaining expression is not evaluated and so the value of i remains the same.

Similarly when && operator is involved in an expression, when any of the operands become false, the whole expression's truth value becomes false and hence the remaining expression will not be evaluated.

false && (anything) => false where (anything) will not be evaluated.

139)
```
main()
{
    register int a=2;
    printf("Address of a = %d",&a);
    printf("Value of a   = %d",a);
}
```
Answer:

Compier Error: '&' on register variable

Rule to Remember:

& (address of) operator cannot be applied on register variables.

140)
```
main()
{
    float i=1.5;
    switch(i)
    {
            case 1: printf("1");
            case 2: printf("2");
            default : printf("0");
    }
}
```
Answer:

Compiler Error: switch expression not integral

Explanation:

Switch statements can be applied only to integral types.

141)
```
main()
{
    extern i;
    printf("%d\n",i);
    {
            int i=20;
            printf("%d\n",i);
    }
}
```
Answer:

Linker Error : Unresolved external symbol i

Explanation:

The identifier i is available in the inner block and so using extern has no use in resolving it.

142)
```
main()
{
    int a=2,*f1,*f2;
```

```
        f1=f2=&a;
        *f2+=*f2+=a+=2.5;
        printf("\n%d %d %d",a,*f1,*f2);
    }
```

Answer:

16 16 16

Explanation:

f1 and f2 both refer to the same memory location a. So changes through f1 and f2 ultimately affects only the value of a.

143)
```
main()
{
    char *p="GOOD";
    char a[ ]="GOOD";
    printf("\n sizeof(p) = %d, sizeof(*p) = %d, strlen(p) = %d", sizeof(p),
    sizeof(*p), strlen(p));
    printf("\n sizeof(a) = %d, strlen(a) = %d", sizeof(a), strlen(a));
}
```

Answer:

sizeof(p) = 2, sizeof(*p) = 1, strlen(p) = 4
sizeof(a) = 5, strlen(a) = 4

Explanation:

sizeof(p) => sizeof(char*) => 2
sizeof(*p) => sizeof(char) => 1
Similarly,
sizeof(a) => size of the character array => 5

When sizeof operator is applied to an array it returns the sizeof the array and it is not the same as the sizeof the pointer variable. Here the sizeof(a) where a is the character array and the size of the array is 5 because the space necessary for the terminating NULL character should also be taken into account.

144)
```
#define DIM( array, type) sizeof(array)/sizeof(type)
main()
{
int arr[10];
printf("The dimension of the array is %d", DIM(arr, int));
}
```

Answer:

10

Explanation:

The size of integer array of 10 elements is 10 * sizeof(int). The macro expands to sizeof(arr)/sizeof(int) => 10 * sizeof(int) / sizeof(int) => 10.

145)
```
int DIM(int array[])
{
return sizeof(array)/sizeof(int );
}
main()
{
int arr[10];
```

```
printf("The dimension of the array is %d", DIM(arr));
}
```

Answer:

1

Explanation:

Arrays cannot be passed to functions as arguments and only the pointers can be passed. So the argument is equivalent to int * array (this is one of the very few places where [] and * usage are equivalent). The return statement becomes, sizeof(int *)/ sizeof(int) that happens to be equal in this case.

146)
```
main()
{
    static int a[3][3]={1,2,3,4,5,6,7,8,9};
    int i,j;
    static *p[]={a,a+1,a+2};
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        printf("%d\t%d\t%d\t%d\n",*(*(p+i)+j),
        *(*(j+p)+i),*(*(i+p)+j),*(*(p+j)+i));
    }
}
```

Answer:

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

Explanation:

((p+i)+j) is equivalent to p[i][j].

147)
```
main()
{
    void swap();
    int x=10,y=8;
    swap(&x,&y);
    printf("x=%d y=%d",x,y);
}
void swap(int *a, int *b)
{
  *a ^= *b,  *b ^= *a, *a ^= *b;
}
```

Answer:

x=10 y=8

Explanation:
Using ^ like this is a way to swap two variables without using a temporary variable and that too in a single statement.
Inside main(), void swap(); means that swap is a function that may take any number of arguments (not no arguments) and returns nothing. So this doesn't issue a compiler error by the call swap(&x,&y); that has two arguments.
This convention is historically due to pre-ANSI style (referred to as Kernighan and Ritchie style) style of function declaration. In that style, the swap function will be defined as follows,

```
void swap()
int *a, int *b
{
  *a ^= *b,  *b ^= *a, *a ^= *b;
}
```

where the arguments follow the (). So naturally the declaration for swap will look like, void swap() which means the swap can take any number of arguments.

148)
```
main()
{
    int i = 257;
    int *iPtr = &i;
    printf("%d %d", *((char*)iPtr), *((char*)iPtr+1) );
}
```

Answer:
1 1
Explanation:
The integer value 257 is stored in the memory as, 00000001 00000001, so the individual bytes are taken by casting it to char * and get printed.

149)
```
main()
{
    int i = 258;
    int *iPtr = &i;
    printf("%d %d", *((char*)iPtr), *((char*)iPtr+1) );
}
```

Answer:
2 1
Explanation:
The integer value 257 can be represented in binary as, 00000001 00000001. Remember that the INTEL machines are 'small-endian' machines. *Small-endian means that the lower order bytes are stored in the higher memory addresses and the higher order bytes are stored in lower addresses*. The integer value 258 is stored in memory as: 00000001 00000010.

150)
```
main()
{
    int i=300;
    char *ptr = &i;
    *++ptr=2;
    printf("%d",i);
```

```
}
```

Answer:

556

Explanation:

The integer value 300 in binary notation is: 00000001 00101100. It is stored in memory (small-endian) as: 00101100 00000001. Result of the expression *++ptr = 2 makes the memory representation as: 00101100 00000010. So the integer corresponding to it is 00000010 00101100 => 556.

151)
```
#include <stdio.h>
main()
{
char * str = "hello";
char * ptr = str;
char least = 127;
while (*ptr++)
   least = (*ptr<least ) ?*ptr :least;
printf("%d",least);
}
```

Answer:

0

Explanation:

After 'ptr' reaches the end of the string the value pointed by 'str' is '\0'. So the value of 'str' is less than that of 'least'. So the value of 'least' finally is 0.

152) Declare an array of N pointers to functions returning pointers to functions returning pointers to characters?

Answer:

```
(char*(*)( )) (*ptr[N])( );
```

153)
```
main()
{
struct student
{
char name[30];
struct date dob;
}stud;
struct date
      {
      int day,month,year;
      };
scanf("%s%d%d%d", stud.rollno, &student.dob.day, &student.dob.month, &student.dob.year);
}
```

Answer:

Compiler Error: Undefined structure date

Explanation:

Inside the struct definition of 'student' the member of type struct date is given. The compiler doesn't have the definition of date structure (forward reference is not allowed in C in this case) so it issues an error.

154)
```
main()
{
struct date;
struct student
      {
      char name[30];
      struct date dob;
      }stud;
struct date
      {
       int day,month,year;
      };
scanf("%s%d%d%d", stud.rollno, &student.dob.day, &student.dob.month,
&student.dob.year);
}
```
Answer:

Compiler Error: Undefined structure date

Explanation:

Only declaration of struct date is available inside the structure definition of 'student' but to have a variable of type struct date the definition of the structure is required.

155)There were 10 records stored in "somefile.dat" but the following program printed 11 names. What went wrong?
```
void main()
{
struct student
{
char name[30], rollno[6];
}stud;
FILE *fp = fopen("somefile.dat","r");
while(!feof(fp))
 {
                fread(&stud, sizeof(stud), 1 , fp);
puts(stud.name);
}
}
```
Explanation:

fread reads 10 records and prints the names successfully. It will return EOF only when fread tries to read another record and fails reading EOF (and returning EOF). So it prints the last record again. After this only the condition feof(fp) becomes false, hence comes out of the while loop.

156)Is there any difference between the two declarations,

- int foo(int *arr[]) and

- int foo(int *arr[2])

Answer:
No

Explanation:
Functions can only pass pointers and not arrays. The numbers that are allowed inside the [] is just for more readability. So there is no difference between the two declarations.

157) What is the subtle error in the following code segment?

```
void fun(int n, int arr[])
{
int *p=0;
int i=0;
while(i++<n)
        p = &arr[i];
*p = 0;
}
```

Answer & Explanation:
If the body of the loop never executes p is assigned no address. So p remains NULL where *p =0 may result in problem (may rise to runtime error "NULL pointer assignment" and terminate the program).

158) What is wrong with the following code?

```
int *foo()
{
int *s = malloc(sizeof(int)100);
assert(s != NULL);
return s;
}
```

Answer & Explanation:
assert macro should be used for debugging and finding out bugs. The check s != NULL is for error/exception handling and for that assert shouldn't be used. A plain if and the corresponding remedy statement has to be given.

159) What is the hidden bug with the following statement?

```
assert(val++ != 0);
```

Answer & Explanation:
Assert macro is used for debugging and removed in release version. In assert, the experssion involves side-effects. So the behavior of the code becomes different in case of debug version and the release version thus leading to a subtle bug.

Rule to Remember:
Don't use expressions that have side-effects in assert statements.

160)

```
void main()
{
int *i = 0x400;  // i points to the address 400
```

```
*i = 0;                    // set the value of memory location pointed by i;
}
```

Answer:

Undefined behavior

Explanation:

The second statement results in undefined behavior because it points to some location whose value may not be available for modification. *This type of pointer in which the non-availability of the implementation of the referenced location is known as 'incomplete type'*.

161)

```
#define assert(cond) if(!(cond)) \
  (fprintf(stderr, "assertion failed: %s, file %s, line %d \n",#cond,\
  __FILE__,__LINE__), abort())

void main()
{
int i = 10;
if(i==0)
   assert(i < 100);
else
   printf("This statement becomes else for if in assert macro");
}
```

Answer:

No output

Explanation:

The else part in which the printf is there becomes the else for if in the assert macro. Hence nothing is printed.

The solution is to use conditional operator instead of if statement,

```
#define assert(cond) ((cond)?(0): (fprintf (stderr, "assertion failed: \ %s, file %s, line %d \n",#cond, __FILE__,__LINE__), abort()))
```

Note: However this problem of "matching with nearest else" cannot be solved by the usual method of placing the if statement inside a block like this,

```
#define assert(cond) { \
if(!(cond)) \
  (fprintf(stderr, "assertion failed: %s, file %s, line %d \n",#cond,\
  __FILE__,__LINE__), abort()) \
}
```

162) Is the following code legal?

```
struct a
   {
int x;
 struct a b;
   }
```

Answer:

No

Explanation:

Is it not legal for a structure to contain a member that is of the same type as in this case. Because this will cause the structure declaration to be recursive without end.

163) Is the following code legal?

```
struct a
   {
int x;
struct a *b;
   }
```

Answer:

Yes.

Explanation:

*b is a pointer to type struct a and so is legal. The compiler knows, the size of the pointer to a structure even before the size of the structure is determined(as you know the pointer to any type is of same size). This type of structures is known as 'self-referencing' structure.

164) Is the following code legal?

```
typedef struct a
   {
int x;
 aType *b;
   }aType
```

Answer:

No

Explanation:

The typename aType is not known at the point of declaring the structure (forward references are not made for typedefs).

165) Is the following code legal?

```
typedef struct a aType;
struct a
{
int x;
aType *b;
};
```

Answer:

Yes

Explanation:

The typename aType is known at the point of declaring the structure, because it is already typedefined.

166) Is the following code legal?

```
void main()
{
typedef struct a aType;
aType someVariable;
```

```
struct a
{
int x;
     aType *b;
};
}
```

Answer:

No

Explanation:

When the declaration,
typedef struct a aType;
is encountered body of struct a is not known. This is known as 'incomplete types'.

167)
```
void main()
{
     printf("sizeof (void *) = %d \n", sizeof( void *));
     printf("sizeof (int *)    = %d \n", sizeof(int *));
     printf("sizeof (double *)  = %d \n", sizeof(double *));
     printf("sizeof(struct unknown *) = %d \n", sizeof(struct unknown *));
}
```

Answer :

```
sizeof (void *) = 2
sizeof (int *)    = 2
sizeof (double *)  =  2
sizeof(struct unknown *) =  2
```

Explanation:

The pointer to any type is of same size.

168) char inputString[100] = {0};

To get string input from the keyboard which one of the following is better?

1) gets(inputString)
2) fgets(inputString, sizeof(inputString), fp)

Answer & Explanation:

The second one is better because gets(inputString) doesn't know the size of the string passed and so, if a very big input (here, more than 100 chars) the charactes will be written past the input string. When fgets is used with stdin performs the same operation as gets but is safe.

169) Which version do you prefer of the following two,

1) printf("%s",str); // or the more curt one
2) printf(str);

Answer & Explanation:

Prefer the first one. If the str contains any format characters like %d then it will result in a subtle bug.

170)
```
void main()
{
int i=10, j=2;
int *ip= &i, *jp = &j;
int k = *ip/*jp;
```

```
printf("%d",k);
}
```

Answer:

Compiler Error: "Unexpected end of file in comment started in line 5".

Explanation:

The programmer intended to divide two integers, but by the "maximum munch" rule, the compiler treats the operator sequence / and * as /* which happens to be the starting of comment. To force what is intended by the programmer,

```
int k = *ip/ *jp;
// give space explicity separating / and *
//or
int k = *ip/(*jp);
// put braces to force the intention
will solve the problem.
```

171)

```
void main()
{
char ch;
for(ch=0;ch<=127;ch++)
printf("%c   %d \n", ch, ch);
}
```

Answer:

Implementaion dependent

Explanation:

The char type may be signed or unsigned by default. If it is signed then ch++ is executed after ch reaches 127 and rotates back to -128. Thus ch is always smaller than 127.

172) Is this code legal?

```
int *ptr;
ptr = (int *) 0x400;
```

Answer:

Yes

Explanation:

The pointer ptr will point at the integer in the memory location 0x400.

173)

```
main()
{
char a[4]="HELLO";
printf("%s",a);
}
```

Answer:

Compiler error: Too many initializers

Explanation:

The array a is of size 4 but the string constant requires 6 bytes to get stored.

174)

```
main()
{
```

```
char a[4]="HELL";
printf("%s",a);
}
```

Answer:

HELL%@!~@!@???@~~!

Explanation:

The character array has the memory just enough to hold the string "HELL" and doesnt have enough space to store the terminating null character. So it prints the HELL correctly and continues to print garbage values till it accidentally comes across a NULL character.

175)
```
main()
{
    int a=10,*j;
    void *k;
    j=k=&a;
    j++;
    k++;
    printf("\n %u %u ",j,k);
}
```

Answer:

Compiler error: Cannot increment a void pointer

Explanation:

Void pointers are generic pointers and they can be used only when the type is not known and as an intermediate address storage type. No pointer arithmetic can be done on it and you cannot apply indirection operator (*) on void pointers.

176)
```
main()
{
        extern int i;
{       int i=20;
 {
   const volatile unsigned i=30; printf("%d",i);
 }
        printf("%d",i);
}
  printf("%d",i);
}
int i;
```

Answer:

30201073828704

177) Printf can be implemented by using _____ list.

Answer:

Variable length argument lists

178)
```
char *someFun()
{
char *temp = "string constant";
```

```
return temp;
}
int main()
{
puts(someFun());
}
```

Answer:

string constant

Explanation:

The program suffers no problem and gives the output correctly because the character constants are stored in code/data area and not allocated in stack, so this doesn't lead to dangling pointers.

179)

```
char *someFun1()
{
char temp[ ] = "string";
return temp;
}
char *someFun2()
{
char temp[ ] = {'s', 't','r','i','n','g'};
return temp;
}
int main()
{
puts(someFun1());
puts(someFun2());
}
```

Answer:

Garbage values.

Explanation:

Both the functions suffer from the problem of dangling pointers. In someFun1() temp is a character array and so the space for it is allocated in heap and is initialized with character string "string". This is created dynamically as the function is called, so is also deleted dynamically on exiting the function so the string data is not available in the calling function main() leading to print some garbage values. The function someFun2() also suffers from the same problem but the problem can be easily identified in this case.

C FAQ

Q1) When is a *switch* statement better than multiple *if* statements?
Answer:
A switch statement is generally best to use when you have more than two conditional expressions based on a single variable of *numeric* type.

Q2) What is the difference between *goto* and *longjmp()* and *setjmp()*?
Answer:
A goto statement implements a *local* jump of program execution, and the longjmp() and setjmp() functions implement a *nonlocal,* or far, jump of program execution. Generally, a jump in execution of any kind should be avoided because it is not considered good programming practice to use such statements as goto and longjmp in your program.
A goto statement simply bypasses code in your program and jumps to a predefined position. To use the goto statement, you give it a labeled position to jump to. This predefined position must be within the same function. You cannot implement gotos between functions.
When your program calls setjmp(), the current state of your program is saved in a structure of type jmp_buf. Later, your program can call the longjmp() function to restore the program's state as it was when you called setjmp().Unlike the goto statement, the longjmp() and setjmp() functions do not need to be implemented in the same function. However, there is a major drawback to using these functions: your program, when restored to its previously saved state, will lose its references to any dynamically allocated memory between the longjmp() and the setjmp(). This means you will waste memory for every malloc() or calloc() you have implemented between your longjmp() and setjmp(), and your program will be horribly inefficient. It is highly recommended that you avoid using functions such as longjmp() and setjmp() because they, like the goto statement, are quite often an indication of poor programming practice.

Q3) What is an lvalue?
Answer:
An lvalue is an expression to which a value can be assigned. The lvalue expression is located on the *left side* of an assignment statement, whereas an rvalue is located on the *right side* of an assignment statement. Each assignment statement must have an lvalue and an rvalue. The lvalue expression must reference a storable variable in memory. It cannot be a constant.

Q4) Array is an lvalue or not?

Answer:
An lvalue was defined as an expression to which a value can be assigned. Is an array an expression
to which we can assign a value? The answer to this question is no, because an array is composed of several separate *array elements* that cannot be treated as a whole for assignment purposes.
The following statement is therefore illegal:

```
int x[5], y[5];
x = y;
```

Additionally, you might want to copy the whole array all at once. You can do so using a library function such as the memcpy() function, which is shown here:

```
memcpy(x, y, sizeof(y));
```

It should be noted here that unlike arrays, structures *can* be treated as lvalues. Thus, you can assign one
structure variable to another structure variable of the same type, such as this:

```
typedef struct t_name
{
char last_name[25];
char first_name[15];
char middle_init[2];
} NAME;
...
NAME my_name, your_name;
...
your_name = my_name;
...
```

Q5) What is page thrashing?

Answer:
Some operating systems (such as UNIX or Windows in enhanced mode) use virtual memory. Virtual
memory is a technique for making a machine behave as if it had more memory than it really has, by using disk space to simulate RAM (random-access memory). In the 80386 and higher Intel CPU chips, and in most other modern microprocessors (such as the Motorola 68030, Sparc, and Power PC), exists a piece of hardware called the Memory Management Unit, or MMU.
The MMU treats memory as if it were composed of a series of "pages." A page of memory is a block of
contiguous bytes of a certain size, usually 4096 or 8192 bytes. The operating system sets up and maintains a table for each running program called the Process Memory Map, or PMM. This is a table of all the pages of memory that program can access and where each is really located.
Every time your program accesses any portion of memory, the address (called a "virtual address") is processed by the MMU. The MMU looks in the PMM to find out where the memory is really located (called the "physical address"). The physical address can be any location in memory or on disk that the operating system has assigned for it. If the location the program wants to access is on disk, the page containing it must be read from disk into memory, and the PMM must be updated to reflect this action (this is called a "page fault").

Because accessing the disk is so much slower than accessing RAM, the operating system tries to keep as much of the virtual memory as possible in RAM. If you're running a large enough program (or several small programs at once), there might not be enough RAM to hold all the memory used by the programs, so some of it must be moved out of RAM and onto disk (this action is called "paging out").
The operating system tries to guess which areas of memory aren't likely to be used for a while (usually based on how the memory has been used in the past). If it guesses wrong, or if your programs are accessing lots of memory in lots of places, many page faults will occur in order to read in the pages that were paged out. Because all of RAM is being used, for each page read in to be accessed, another page must be paged out. This can lead to more page faults, because now a different page of memory has been moved to disk.
The problem of many page faults occurring in a short time, called "page thrashing," can drastically cut the performance of a system. Programs that frequently access many widely separated locations in memory are more likely to cause page thrashing on a system. So is running many small programs that all continue to run even when you are not actively using them. To reduce page thrashing, you can run fewer programs simultaneously. Or you can try changing the way a large program works to maximize the capability of the operating system to guess which pages won't be needed. You can achieve this effect by caching values or changing lookup algorithms in large data structures, or sometimes by changing to a memory allocation library which provides an implementation of malloc() that allocates memory more efficiently. Finally, you might consider adding more RAM to the system to reduce the need to page out.

Q6) What is a *const* pointer?
Answer:
The access modifier keyword const is a promise the programmer makes to the compiler that the value of a variable will not be changed after it is initialized. The compiler will enforce that promise as best it can by not enabling the programmer to write code which modifies a variable that has been declared const.

A "const pointer," or more correctly, a "pointer to const," is a pointer which points to data that is const
(constant, or unchanging). A pointer to const is declared by putting the word const at the beginning of the pointer declaration. This declares a pointer which points to data that can't be modified. The pointer itself can be modified. The following example illustrates some legal and illegal uses of a const pointer:

```
const char *str = "hello";
char c = *str /* legal */
str++; /* legal */
*str = 'a'; /* illegal */
str[1] = 'b'; /* illegal */
```

Q7) When should the register modifier be used? Does it really help?
Answer:

The register modifier hints to the compiler that the variable will be heavily used and should be kept in the CPU's registers, if possible, so that it can be accessed faster. There are several restrictions on the use of the register modifier.
First, the variable must be of a type that can be held in the CPU's register. This usually means a single value of a size less than or equal to the size of an integer. Some machines have registers that can hold floating-point numbers as well.
Second, because the variable might not be stored in memory, its address cannot be taken with the unary & operator. An attempt to do so is flagged as an error by the compiler. Some additional rules affect how useful the register modifier is. Because the number of registers is limited, and because some registers can hold only certain types of data (such as pointers or floating-point numbers), the number and types of register modifiers that will actually have any effect are dependent on what machine the program will run on. Any additional register modifiers are silently ignored by the compiler.
Also, in some cases, it might actually be slower to keep a variable in a register because that register then
becomes unavailable for other purposes or because the variable isn't used enough to justify the overhead of loading and storing it.
So when should the register modifier be used? The answer is never, with most modern compilers. Early C compilers did not keep any variables in registers unless directed to do so, and the register modifier was a valuable addition to the language. C compiler design has advanced to the point, however, where the compiler will usually make better decisions than the programmer about which variables should be stored in registers.
In fact, many compilers actually ignore the register modifier, which is perfectly legal, because it is only a hint and not a directive.

Q8) when should the *volatile* modifier be used?
Answer:
The volatile modifier is a directive to the compiler's optimizer that operations involving this variable should not be optimized in certain ways. There are two special cases in which use of the volatile modifier is desirable. The first case involves memory-mapped hardware (a device such as a graphics adaptor that appears to the computer's hardware as if it were part of the computer's memory), and the second involves shared memory (memory used by two or more programs running simultaneously).
Most computers have a set of registers that can be accessed faster than the computer's main memory. A good compiler will perform a kind of optimization called "redundant load and store removal." The compiler looks for places in the code where it can either remove an instruction to load data from memory because the value is already in a register, or remove an instruction to store data to memory because the value can stay in a register until it is changed again anyway.
If a variable is a pointer to something other than normal memory, such as memory-mapped ports on a
peripheral, redundant load and store optimizations might be detrimental. For instance, here's a piece of code that might be used to time some operation:

```
time_t time_addition(volatile const struct timer *t, int a)
{
int n;
int x;
```

```
time_t then;
x = 0;
then = t->value;
for (n = 0; n < 1000; n++)
{
x = x + a;
}
return t->value - then;
}
```

In this code, the variable t->value is actually a hardware counter that is being incremented as time passes. The function adds the value of a to x 1000 times, and it returns the amount the timer was incremented by while the 1000 additions were being performed.

Without the volatile modifier, a clever optimizer might assume that the value of t does not change during the execution of the function, because there is no statement that explicitly changes it. In that case, there's no need to read it from memory a second time and subtract it, because the answer will always be 0. The compiler might therefore "optimize" the function by making it always return 0.

If a variable points to data in shared memory, you also don't want the compiler to perform redundant load and store optimizations. Shared memory is normally used to enable two programs to communicate with each other by having one program store data in the shared portion of memory and the other program read the same portion of memory. If the compiler optimizes away a load or store of shared memory, communication between the two programs will be affected.

Q9) Can a variable be both *const* and *volatile*?

Answer:

Yes. The const modifier means that this code cannot change the value of the variable, but that does not mean that the value cannot be changed by means outside this code. For instance, in the example in

FAQ 8, the timer structure was accessed through a volatile const pointer. The function itself did not change the value of the timer, so it was declared const. However, the value was changed by hardware on the computer, so it was declared volatile. If a variable is both const and volatile, the two modifiers can appear in either order.

Q10) How reliable are floating-point comparisons?

Answer:

Floating-point numbers are the "black art" of computer programming. One reason why this is so is that there is no optimal way to represent an arbitrary number. The Institute of Electrical and Electronic Engineers (IEEE) has developed a standard for the representation of floating-point numbers, but you cannot guarantee that every machine you use will conform to the standard.

Even if your machine does conform to the standard, there are deeper issues. It can be shown mathematically that there are an infinite number of "real" numbers between any two numbers. For the computer to distinguish between two numbers, the bits that represent them must differ. To represent an infinite number of different bit patterns would take an infinite number of bits. Because the computer must represent a large range of numbers in a small number of bits (usually 32 to 64 bits), it has to make approximate representations of most numbers.

Because floating-point numbers are so tricky to deal with, it's generally bad practice to compare a floating-point number for equality with anything. Inequalities are much safer.

Q11) How can you determine the maximum value that a numeric variable can hold?
Answer:
For integral types, on a machine that uses two's complement arithmetic (which is just about any machine you're likely to use), a signed type can hold numbers from –2(number of bits – 1) to +2(number of bits – 1) – 1. An unsigned type can hold values from 0 to +2(number of bits) – 1. For instance, a 16-bit signed integer can hold numbers from –2^15 (–32768) to +2^15 – 1 (32767).

Q12) When should a type cast be used?
Answer:
There are two situations in which to use a type cast. The first use is to change the type of an operand to an arithmetic operation so that the operation will be performed properly.
The second case is to cast pointer types to and from void * in order to interface with functions that expect or return void pointers. For example, the following line type casts the return value of the call to malloc() to be a pointer to a foo structure.
struct foo *p = (struct foo *) malloc(sizeof(struct foo));

Q13) When should a type cast not be used?
Answer:
A type cast should not be used to override a const or volatile declaration. Overriding these type modifiers can cause the program to fail to run correctly.
A type cast should not be used to turn a pointer to one type of structure or data type into another. In the
rare events in which this action is beneficial, using a union to hold the values makes the programmer's
intentions clearer.

Q14) Is it acceptable to declare/define a variable in a C header?
Answer:
A global variable that must be accessed from more than one file can and should be declared in a header file. In addition, such a variable must be defined in one source file. Variables should not be defined in header files, because the header file can be included in multiple source files, which would cause multiple definitions of the variable. The ANSI C standard will allow multiple external definitions, provided that there is only one initialization. But because there's really no advantage to using this feature, it's probably best to avoid it and maintain a higher level of portability.
"Global" variables that do not have to be accessed from more than one file should be declared static and
should not appear in a header file.

Q15) What is the difference between declaring a variable and defining a variable?
Answer:

Declaring a variable means describing its type to the compiler but not allocating any space for it. Defining a variable means declaring it and also allocating space to hold the variable. You can also initialize a variable at the time it is defined.

Q16) Can *static* variables be declared in a header file?

Answer:

You can't declare a static variable without defining it as well (this is because the storage class modifiers
static and extern are mutually exclusive). A static variable can be defined in a header file, but this would cause each source file that included the header file to have its own private copy of the variable, which is probably not what was intended.

Q17) What is the benefit of using *const* for declaring constants?

Answer:

The benefit of using the const keyword is that the compiler might be able to make optimizations based on the knowledge that the value of the variable will not change. In addition, the compiler will try to ensure that the values won't be changed inadvertently.
Of course, the same benefits apply to #defined constants. The reason to use const rather than #define to define a constant is that a const variable can be of any type (such as a struct, which can't be represented by a #defined constant). Also, because a const variable is a real variable, it has an address that can be used, if needed, and it resides in only one place in memory

Q18) What is the easiest sorting method to use?

Answer:

The answer is the standard library function qsort(). It's the easiest sort by far for several reasons:
It is already written.
It is already debugged.
It has been optimized as much as possible (usually).
Void qsort(void *buf, size_t num, size_t size, int (*comp)(const void *ele1, const void *ele2));

Q19) What is the quickest sorting method to use?

Answer:

The answer depends on what you mean by quickest. For most sorting problems, it just doesn't matter how quick the sort is because it is done infrequently or other operations take significantly more time anyway. Even in cases in which sorting speed is of the essence, there is no one answer. It depends on not only the size and nature of the data, but also the likely order. No algorithm is best in all cases.
There are three sorting methods in this author's "toolbox" that are all very fast and that are useful in different situations. Those methods are quick sort, merge sort, and radix sort.

The Quick Sort

The quick sort algorithm is of the "divide and conquer" type. That means it works by reducing a sorting
problem into several easier sorting problems and solving each of them. A "dividing" value is chosen from the input data, and the data is partitioned into three sets: elements that belong before the dividing value, the value itself, and elements that come after the dividing value. The partitioning is performed by exchanging elements that are in the first set but belong in the third with elements that are in the third set but belong in the first Elements that are equal to the dividing element can be put in any of the three sets—the algorithm will still work properly.

The Merge Sort

The merge sort is a "divide and conquer" sort as well. It works by considering the data to be sorted as a
sequence of already-sorted lists (in the worst case, each list is one element long). Adjacent sorted lists are merged into larger sorted lists until there is a single sorted list containing all the elements. The merge sort is good at sorting lists and other data structures that are not in arrays, and it can be used to sort things that don't fit into memory. It also can be implemented as a stable sort.

The Radix Sort

The radix sort takes a list of integers and puts each element on a smaller list, depending on the value of its least significant byte. Then the small lists are concatenated, and the process is repeated for each more significant byte until the list is sorted. The radix sort is simpler to implement on fixed-length data such as ints.

Q20) How can I sort things that are too large to bring into memory?

Answer:

A sorting program that sorts items that are on secondary storage (disk or tape) rather than primary storage (memory) is called an *external* sort. Exactly how to sort large data depends on what is meant by "too large to fit in memory." If the items to be sorted are themselves too large to fit in memory (such as images), but there aren't many items, you can keep in memory only the sort key and a value indicating the data's location on disk. After the key/value pairs are sorted, the data is rearranged on disk into the correct order. If "too large to fit in memory" means that there are too many items to fit into memory at one time, the data can be sorted in groups that will fit into memory, and then the resulting files can be merged. A sort such as a radix sort can also be used as an external sort, by making each bucket in the sort a file. Even the quick sort can be an external sort. The data can be partitioned by writing it to two smaller files. When the partitions are small enough to fit, they are sorted in memory and concatenated to form the sorted file.

Q21) What is the easiest searching method to use?

Answer:

Just as qsort() was the easiest sorting method, because it is part of the standard library, bsearch() is the
easiest searching method to use. If the given array is in the sorted order bsearch() is the best method.
Following is the prototype for bsearch():

```
void *bsearch(const void *key, const void *buf, size_t num, size_t size, int
(*comp)(const void *, const void*));
```

Another simple searching method is a linear search. A linear search is not as fast as bsearch() for searching among a large number of items, but it is adequate for many purposes. A linear search might be the only method available, if the data isn't sorted or can't be accessed randomly. A linear search starts at the beginning and sequentially compares the key to each element in the data set.

Q22) What is the quickest searching method to use?

Answer:

A binary search, such as bsearch() performs, is much faster than a linear search. A hashing algorithm can provide even faster searching. One particularly interesting and fast method for searching is to keep the data in a "digital trie." A digital trie offers the prospect of being able to search for an item in essentially a constant amount of time, independent of how many items are in the data set.

A digital trie combines aspects of binary searching, radix searching, and hashing. The term "digital trie" refers to the data structure used to hold the items to be searched. It is a multilevel data structure that branches *N* ways at each level.

Q23) What is hashing?

Answer:

To hash means to grind up, and that's essentially what hashing is all about. The heart of a hashing algorithm is a hash function that takes your nice, neat data and grinds it into some random-looking integer.

The idea behind hashing is that some data either has no inherent ordering (such as images) or is expensive to compare (such as images). If the data has no inherent ordering, you can't perform comparison searches.

If the data is expensive to compare, the number of comparisons used even by a binary search might be too many. So instead of looking at the data themselves, you'll condense (hash) the data to an integer (its hash value) and keep all the data with the same hash value in the same place. This task is carried out by using the hash value as an index into an array.

To search for an item, you simply hash it and look at all the data whose hash values match that of the data you're looking for. This technique greatly lessens the number of items you have to look at. If the parameters are set up with care and enough storage is available for the hash table, the number of comparisons needed to find an item can be made arbitrarily close to one.

One aspect that affects the efficiency of a hashing implementation is the hash function itself. It should ideally distribute data randomly throughout the entire hash table, to reduce the likelihood of collisions. Collisions occur when two different keys have the same hash value. There are two ways to resolve this problem. In "open addressing," the collision is resolved by the choosing of another position in the hash table for the element inserted later. When the hash table is searched, if the entry is not found at its hashed position in the table, the search continues checking until either the element is found or an empty position in the table is found

The second method of resolving a hash collision is called "chaining." In this method, a "bucket" or linked list holds all the elements whose keys hash to the same value. When the hash table is searched, the list must be searched linearly.

Q24) How can I sort a linked list?

Answer:

Both the merge sort and the radix sort are good sorting algorithms to use for linked lists.

Q25) How can I search for data in a linked list?

Answer:

Unfortunately, the only way to search a linked list is with a linear search, because the only way a linked list's members can be accessed is sequentially. Sometimes it is quicker to take the data from a linked list and store it in a different data structure so that searches can be more efficient.

Q26) How do you redirect a standard stream?

Answer:

Most operating systems, including DOS, provide a means to redirect program input and output to and from different devices. This means that rather than your program output (stdout) going to the screen; it can be redirected to a file or printer port. Similarly, your program's input (stdin) can come from a file rather than the keyboard. In DOS, this task is accomplished using the redirection characters, < and >. For example, if you wanted a program named PRINTIT.EXE to receive its input (stdin) from a file named STRINGS.TXT, you would enter the following command at the DOS prompt:

C:>PRINTIT < STRINGS.TXT

Notice that the name of the executable file always comes first. The less-than sign (<) tells DOS to take the strings contained in STRINGS.TXT and use them as input for the PRINTIT program.

The following example would redirect the program's output to the prn device, usually the printer attached on LPT1:

C :> REDIR > PRN

Alternatively, you might want to redirect the program's output to a file, as the following example shows:

C :> REDIR > REDIR.OUT

In this example, all output that would have normally appeared on-screen will be written to the file
REDIR.OUT.

Redirection of standard streams does not always have to occur at the operating system. You can redirect a standard stream from *within your program* by using the standard C library function named freopen(). For example, if you wanted to redirect the stdout standard stream within your program to a file named
OUTPUT.TXT, you would implement the freopen() function as shown here:

```
...
freopen("output.txt", "w", stdout);
...
```

Now, every output statement (printf(), puts(), putch(), and so on) in your program will appear in the file
OUTPUT.TXT.

Q27) How can you restore a redirected standard stream?

Answer:

The preceding example showed how you can redirect a standard stream from within your program. But what if later in your program you wanted to restore the standard

stream to its *original* state? By using the standard C library functions named dup() and fdopen(), you can restore a standard stream such as stdout to its original state. The dup() function duplicates a file handle. You can use the dup() function to save the file handle
corresponding to the stdout standard stream. The fdopen() function opens a stream that has been
duplicated with the dup() function.

Q28) What is the difference between text and binary modes?

Answer:

Streams can be classified into two types: *text* streams and *binary* streams. Text streams are interpreted, with a maximum length of 255 characters. With text streams, carriage return/line feed combinations are translated to the newline \n character and vice versa. Binary streams are uninterpreted and are treated one byte at a time with no translation of characters. Typically, a text stream would be used for reading and writing standard text files, printing output to the screen or printer, or receiving input from the keyboard.

A binary text stream would typically be used for reading and writing binary files such as graphics or word processing documents, reading mouse input, or reading and writing to the modem.

29) How do you determine whether to use a stream function or a low-level function?

Answer:

Stream functions such as fread() and fwrite() are buffered and are more efficient when reading and writing text or binary data to files. You generally gain better performance by using stream functions rather than their unbuffered low-level counterparts such as read() and write().

In multi-user environments, however, when files are typically shared and portions of files are continuously being locked, read from, written to, and unlocked, the stream functions do not perform as well as the low-level functions. This is because it is hard to buffer a shared file whose contents are constantly changing. Generally, you should always use buffered stream functions when accessing nonshared files, and you should always use the low-level functions when accessing shared files

.

Q30) How can I open a file so that other programs can update it at the same time?

Answer:

Your C compiler library contains a *low-level* file function called sopen() that can be used to open a file in shared mode. Beginning with DOS 3.0, files could be opened in shared mode by loading a special program named SHARE.EXE. Shared mode, as the name implies, allows a file to be shared with other programs as well as your own. Using this function, you can allow other programs that are running to update the same file you are updating.

The sopen() function takes four parameters: a pointer to the filename you want to open, the operational
mode you want to open the file in, the file sharing mode to use, and, if you are creating a file, the mode to create the file in. The second parameter of the sopen() function, usually referred to as the “operation flag”parameter, can have the following values assigned to it:

Constant Description

O_APPEND Appends all writes to the end of the file

O_BINARY Opens the file in binary (untranslated) mode
O_CREAT If the file does not exist, it is created
O_EXCL If the O_CREAT flag is used and the file exists, returns an error
O_RDONLY Opens the file in read-only mode
O_RDWR Opens the file for reading and writing
O_TEXT Opens the file in text (translated) mode
O_TRUNC Opens an existing file and writes over its contents
O_WRONLY Opens the file in write-only mode
The third parameter of the sopen() function, usually referred to as the "sharing flag," can have the following values assigned to it:
Constant Description
SH_COMPAT No other program can access the file
SH_DENYRW No other program can read from or write to the file
SH_DENYWR No other program can write to the file
SH_DENYRD No other program can read from the file
SH_DENYNO Any program can read from or write to the file
If the sopen() function is successful, it returns a non-negative number that is the file's handle. If an error
occurs, –1 is returned, and the global variable errno is set to one of the following values:
Constant Description
ENOENT File or path not found
EMFILE No more file handles are available
EACCES Permission denied to access file
EINVACC Invalid access code
Constant Description

Q31) How can I make sure that my program is the only one accessing a file?
Answer:
By using the sopen() function you can open a file in shared mode and explicitly deny reading and writing permissions to any other program but yours. This task is accomplished by using the SH_DENYWR shared flag to denote that your program is going to deny any writing or reading attempts by other programs.
For example, the following snippet of code shows a file being opened in shared mode, denying
access to all other files:

```
/* Note that the sopen() function is not ANSI compliant... */
fileHandle = sopen("C:\\DATA\\SETUP.DAT", O_RDWR, SH_DENYWR);
```

By issuing this statement, all other programs are denied access to the SETUP.DAT file. If another program were to try to open SETUP.DAT for reading or writing, it would receive an EACCES error code, denoting that access is denied to the file.

Q32) What is Preprocessor?
Answer:
The preprocessor is used to modify your program according to the *preprocessor directives* in your source code. Preprocessor directives (such as #define) give the preprocessor specific instructions on how to modify your source code. The preprocessor reads in all of your include files and the source code you are compiling and creates a preprocessed version of your source code. This preprocessed version has all of its macros and constant symbols replaced by their corresponding code and value

assignments. If your source code contains any conditional preprocessor directives (such as #if), the preprocessor evaluates the condition and modifies your source code accordingly.
The preprocessor contains many features that are powerful to use, such as creating macros, performing conditional compilation, inserting predefined environment variables into your code, and turning compiler features on and off. For the professional programmer, in-depth knowledge of the features of the preprocessor can be one of the keys to creating fast, efficient programs.

Q33) What is a macro, and how do you use it?
Answer:
A macro is a preprocessor directive that provides a mechanism for token replacement in your source code. Macros are created by using the #define statement.
Here is an example of a macro: Macros can also utilize special operators such as the *stringizing* operator (#) and the *concatenation* operator (##).The stringizing operator can be used to convert macro parameters to quoted strings, as in the following example:
#define DEBUG_VALUE(v) printf(#v " is equal to %d.\n", v)
In your program, you can check the value of a variable by invoking the DEBUG_VALUE macro:
...
int x = 20;
DEBUG_VALUE(x);
... The preceding code prints "x is equal to 20." on-screen. This example shows that the stringizing operator used with macros can be a very handy debugging tool.

Q34) What will the preprocessor do for a program?
Answer:
The C preprocessor is used to modify your program according to the *preprocessor directives* in your source code. A preprocessor directive is a statement (such as #define) that gives the preprocessor specific instructions on how to modify your source code. The preprocessor is invoked as the first part of your compiler program's compilation step. It is usually hidden from the programmer because it is run automatically by the compiler.
The preprocessor reads in all of your include files and the source code you are compiling and creates a
preprocessed version of your source code. This preprocessed version has all of its macros and constant
symbols replaced by their corresponding code and value assignments. If your source code contains any
conditional preprocessor directives (such as #if), the preprocessor evaluates the condition and modifies your source code accordingly.

Q35) How can you avoid including a header more than once?
Answer:
One easy technique to avoid multiple inclusions of the same header is to use the #ifndef and #define
preprocessor directives. When you create a header for your program, you can #define a symbolic name that is unique to that header. You can use the conditional preprocessor directive named #ifndef to check whether that symbolic name has

already been assigned. If it is assigned, you should not include the header, because it has already been preprocessed. If it is not defined, you should define it to avoid any further inclusions of the header. The following header illustrates this technique:

```
#ifndef _FILENAME_H
#define _FILENAME_H
#define VER_NUM “1.00.00”
#define REL_DATE “08/01/94”
#if _ _WINDOWS_ _
#define OS_VER “WINDOWS”
#else
#define OS_VER “DOS”
#endif
#endif
```

When the preprocessor encounters this header, it first checks to see whether _FILENAME_H has been defined. If it hasn’t been defined, the header has not been included yet, and the _FILENAME_H symbolic name is defined. Then, the rest of the header is parsed until the last #endif is encountered, signaling the end of the conditional #ifndef _FILENAME_H statement. Substitute the actual name of the header file for “FILENAME” in the preceding example to make it applicable for your programs.

Q36) Can a file other than a .h file be included with *#include*?

Answer:

The preprocessor will include whatever file you specify in your #include statement. Therefore, if you have the line

```
#include <macros.inc>
```

in your program, the file macros.inc will be included in your precompiled program. It is, however, unusual programming practice to put any file that does not have a .h or .hpp extension in an #include statement. You should always put a .h extension on any of your C files you are going to include. This method makes it easier for you and others to identify which files are being used for preprocessing purposes. For instance, someone modifying or debugging your program might not know to look at the macros.inc file for macro definitions. That person might try in vain by searching all files with .h extensions and come up empty. If your file had been named macros.h, the search would have included the macros.h file, and the searcher would have been able to see what macros you defined in it.

Q37) What is the benefit of using *#define* to declare a constant?

Answer:

Using the #define method of declaring a constant enables you to declare a constant in one place and use it throughout your program. This helps make your programs more maintainable, because you need to maintain only the #define statement and not several instances of individual constants throughout your program.

For instance, if your program used the value of pi (approximately 3.14159) several times, you might want to declare a constant for pi as follows:

```
#define PI 3.14159
```

Using the #define method of declaring a constant is probably the most familiar way of declaring constants to traditional C programmers. Besides being the most common method of declaring constants, it also takes up the least memory. Constants defined in this manner are simply placed directly into your source code, with no variable space

allocated in memory. Unfortunately, this is one reason why most debuggers cannot inspect constants created using the #define method.

Q38) What is the benefit of using an *enum* rather than a *#define* constant?
Answer:
The use of an *enumeration constant* (enum) has many advantages over using the traditional *symbolic constant*
style of #define. These advantages include a lower maintenance requirement, improved program readability,
and better debugging capability.

1) The first advantage is that enumerated constants are generated automatically by the compiler. Conversely, symbolic constants must be manually assigned values by the programmer.
For instance, if you had an enumerated constant type for error codes that could occur in your program, your
enum definition could look something like this:

```
enum Error_Code
{
OUT_OF_MEMORY,
INSUFFICIENT_DISK_SPACE,
LOGIC_ERROR,
FILE_NOT_FOUND
};
```

In the preceding example, OUT_OF_MEMORY is automatically assigned the value of 0 (zero) by the compiler
because it appears first in the definition. The compiler then continues to automatically assign numbers to
the enumerated constants, making INSUFFICIENT_DISK_SPACE equal to 1, LOGIC_ERROR equal to 2, and FILE_NOT_FOUND equal to 3, so on.
If you were to approach the same example by using symbolic constants, your code would look something like this:

```
#define OUT_OF_MEMORY 0
#define INSUFFICIENT_DISK_SPACE 1
#define LOGIC_ERROR 2
#define FILE_NOT_FOUND 3
```

values by the programmer. Each of the two methods arrives at the same result: four constants assigned numeric values to represent error codes. Consider the maintenance required, however, if you were to add two constants to represent the error codes DRIVE_NOT_READY and CORRUPT_FILE. Using the enumeration constant method, you simply would put these two constants anywhere in the enum definition. The compiler would generate two *unique* values for these constants. Using the symbolic constant method, you would have to *manually* assign two new numbers to these constants. Additionally, you would want to ensure that the numbers you assign to these constants are unique.
2) Another advantage of using the enumeration constant method is that your programs are more readable and
thus can be understood better by others who might have to update your program later.
3) A third advantage to using enumeration constants is that some symbolic debuggers can print the value of an

enumeration constant. Conversely, most symbolic debuggers cannot print the value of a symbolic constant.
This can be an enormous help in debugging your program, because if your program is stopped at a line that
uses an enum, you can simply inspect that constant and instantly know its value. On the other hand, because
most debuggers cannot print #define values, you would most likely have to search for that value by manually
looking it up in a header file.

Q39) How are portions of a program disabled in demo versions?
Answer:
If you are distributing a demo version of your program, the preprocessor can be used to enable or disable
portions of your program. The following portion of code shows how this task is accomplished, using the
preprocessor directives #if and #endif:

```
int save_document(char* doc_name)
{
#if DEMO_VERSION
printf("Sorry! You can't save documents using the DEMO version of this
program!\n");
return(0);
#endif
...
}
```

Q40) Is it better to use a macro or a function?
Answer:
The answer depends on the situation you are writing code for. Macros have the distinct advantage of being more efficient (and faster) than functions, because their corresponding code is inserted directly into your source code at the point where the macro is called. There is no overhead involved in using a macro like there is in placing a call to a function. However, macros are generally small and cannot handle large, complex coding constructs. A function is more suited for this type of situation. Additionally, macros are expanded inline, which means that the code is replicated for each occurrence of a macro. Your code therefore could be somewhat larger when you use macros than if you were to use functions.
Thus, the choice between using a macro and using a function is one of deciding between the tradeoff of faster
program speed versus smaller program size. Generally, you should use macros to replace small, repeatable code sections, and you should use functions for larger coding tasks that might require several lines of code.

Q41) What is the difference between *#include <file>* and *#include "file"*?
Answer:
When writing your C program, you can include files in two ways. The first way is to surround the file you
want to include with the angled brackets < and >. This method of inclusion tells the preprocessor to look for

the file in the predefined default location. This predefined default location is often an INCLUDE environment
variable that denotes the path to your include files. For instance, given the INCLUDE variable
INCLUDE=C:\COMPILER\INCLUDE;S:\SOURCE\HEADERS;
using the #include <file> version of file inclusion, the compiler first checks the C:\COMPILER\INCLUDE
directory for the specified file. If the file is not found there, the compiler then checks the
S:\SOURCE\HEADERS directory. If the file is still not found, the preprocessor checks the current directory.

The second way to include files is to surround the file you want to include with double quotation marks. This
method of inclusion tells the preprocessor to look for the file in the current directory first, then look for it in the predefined locations you have set up. Using the #include "file" version of file inclusion and applying
it to the preceding example, the preprocessor first checks the current directory for the specified file. If the
file is not found in the current directory, the C:\COMPILER\INCLUDE directory is searched. If the file
is still not found, the preprocessor checks the S:\SOURCE\HEADERS directory.
The #include <file> method of file inclusion is often used to include *standard* headers such as stdio.h or
stdlib.h. This is because these headers are rarely (if ever) modified, and they should always be read from your
compiler's standard include file directory.
The #include "file" method of file inclusion is often used to include *nonstandard* header files that you have created for use in your program. This is because these headers are often modified in the current directory, and you will want the preprocessor to use your newly modified version of the header rather than the older, unmodified version.

Q42) Can you define which header file to include at compile time?
Answer:
Yes. This can be done by using the #if, #else, and #endif preprocessor directives. For example, certain
compilers use different names for header files. One such case is between Borland C++, which uses the header
file alloc.h, and Microsoft C++, which uses the header file malloc.h. Both of these headers serve the same
purpose, and each contains roughly the same definitions. If, however, you are writing a program that is to
support Borland C++ and Microsoft C++, you must define which header to include at compile time. The
following example shows how this can be done:

```
#ifdef _ _BORLANDC_ _
#include <alloc.h>
#else
#include <malloc.h>
```

```
#endif
```

Q43) Can include files be nested?

Answer:

Yes. Include files can be nested any number of times. As long as you use precautionary measures , you can avoid including the same file twice. In the past, nesting header files was seen as bad programming practice, because it complicates the dependency tracking function of the MAKE program and thus slows down compilation. Many of today's popular compilers make up for this difficulty by implementing a concept called *precompiled headers,* in which all headers and associated dependencies are stored in a precompiled state.

Many programmers like to create a custom header file that has #include statements for every header needed
for each module. This is perfectly acceptable and can help avoid potential problems relating to #include files,
such as accidentally omitting an #include file in a module.

Q44) How many levels deep can include files be nested?

Answer:

Even though there is no limit to the number of levels of nested include files you can have, your compiler might
run out of stack space while trying to include an inordinately high number of files. This number varies
according to your hardware configuration and possibly your compiler.

Q45) How can type-insensitive macros be created?

Answer:

A type-insensitive macro is a macro that performs the same basic operation on different data types. This task can be accomplished by using the concatenation operator to create a call to a type-sensitive function based on the parameter passed to the macro. The following program provides an example:

```
#include <stdio.h>
#define SORT(data_type) sort_ ## data_type
void sort_int(int** i);
void sort_long(long** l);
void sort_float(float** f);
void sort_string(char** s);
void main(void);
void main(void)
{
int** ip;
long** lp;
float** fp;
char** cp;
...
sort(int)(ip);
sort(long)(lp);
sort(float)(fp);
```

```
sort(char)(cp);
...
}
```

This program contains four functions to sort four different data types: int, long, float, and string (notice
that only the function prototypes are included for brevity). A macro named SORT was created to take the data
type passed to the macro and combine it with the sort_ string to form a valid function call that is appropriate
for the data type being sorted. Thus, the string

```
sort(int)(ip);
```

translates into

```
sort_int(ip);
```

after being run through the preprocessor.

Q46) What are the standard predefined macros?

Answer:

The ANSI C standard defines six predefined macros for use in the C language:

Macro Name Purpose

_ _LINE_ _ Inserts the current source code line number in your code.
_ _FILE_ _ Inserts the current source code filename in your code.
_ _DATE_ _ Inserts the current date of compilation in your code.
_ _TIME_ _ Inserts the current time of compilation in your code.
_ _STDC_ _ Is set to 1 if you are enforcing strict ANSI C conformity.
_ _cplusplus Is defined if you are compiling a C++ program.

Q47) What is a pragma?

Answer:

The #pragma preprocessor directive allows each compiler to implement compiler-specific features that can
be turned on and off with the #pragma statement. For instance, your compiler might support a feature called
loop optimization. This feature can be invoked as a command-line option or as a #pragma directive. To
implement this option using the #pragma directive, you would put the following line into your code:

```
#pragma loop_opt(on)
```

Conversely, you can turn off loop optimization by inserting the following line into your code:

```
#pragma loop_opt(off)
```

Q48) What is *#line* used for?

Answer:

The #line preprocessor directive is used to reset the values of the _ _LINE_ _ and _ _FILE_ _ symbols,
respectively. This directive is commonly used in fourth-generation languages that generate C language source
files.

Q49) How do you override a defined macro?

Answer:
You can use the #undef preprocessor directive to undefine (override) a previously defined macro.

Q50) How can you check to see whether a symbol is defined?
Answer:
You can use the #ifdef and #ifndef preprocessor directives to check whether a symbol has been defined
(#ifdef) or whether it has not been defined (#ifndef).

Q51) What is the difference between a string copy (*strcpy*) and a memory copy (*memcpy*)? When should each be used?
Answer:
The strcpy() function is designed to work exclusively with strings. It copies each byte of the source string to the destination string and stops when the terminating *null character* (\0) has been moved. On the other hand, the memcpy() function is designed to work with any type of data. Because not all data ends with a null character, you must provide the memcpy() function with the number of bytes you want to copy from the source to the destination.

Q52) How can I convert a number to a string?
Answer:
The standard C library provides several functions for converting numbers of all formats (integers, longs, floats, and so on) to strings and vice versa

The following functions can be used to convert integers to strings:
Function Name Purpose
itoa() Converts an integer value to a string.
ltoa() Converts a long integer value to a string.
ultoa() Converts an unsigned long integer value to a string.

The following functions can be used to convert floating-point values to strings:
Function Name Purpose
ecvt() Converts a double-precision floating-point value to a string without an embedded decimal point.
fcvt() Same as ecvt(), but forces the precision to a specified number of digits.
gcvt() Converts a double-precision floating-point value to a string with an embedded decimal point.

Q53) How can I convert a string to a number?
Answer:
The standard C library provides several functions for converting strings to numbers of all formats (integers, longs, floats, and so on) and vice versa.

The following functions can be used to convert strings to numbers:
Function Name Purpose
atof() Converts a string to a double-precision floating-point value.
atoi() Converts a string to an integer.
atol() Converts a string to a long integer.

strtod() Converts a string to a double-precision floating-point value and reports any "leftover" numbers that could not be converted.
strtol() Converts a string to a long integer and reports any "leftover" numbers that could not be converted.
strtoul() Converts a string to an unsigned long integer and reports any "leftover" numbers that could not be converted.

Q54) How do you print only part of a string?
Answer:
/* Use printf() to print the first 11 characters of source_str. */
printf("First 11 characters: '%11.11s'\n", source_str);

Q55) What is indirection?
Answer:
If you declare a variable, its name is a direct reference to its value. If you have a pointer to a variable, or any other object in memory, you have an indirect reference to its value.

Q56) How many levels of pointers can you have?
Answer:
The answer depends on what you mean by "levels of pointers." If you mean "How many levels of indirection
can you have in a single declaration?" the answer is "At least 12."
int i = 0;
int *ip01 = & i;
int **ip02 = & ip01;
int ***ip03 = & ip02;
int ****ip04 = & ip03;
int *****ip05 = & ip04;
int ******ip06 = & ip05;
int *******ip07 = & ip06;
int ********ip08 = & ip07;
int *********ip09 = & ip08;
int **********ip10 = & ip09;
int ***********ip11 = & ip10;
int ************ip12 = & ip11;
************ip12 = 1; /* i = 1 */
The ANSI C standard says all compilers must handle at least 12 levels. Your compiler might support more.

Q57) What is a null pointer?
Answer:
There are times when it's necessary to have a pointer that doesn't point to anything. The macro NULL, defined in <stddef.h>, has a value that's guaranteed to be different from any valid pointer. NULL is a literal zero, possibly cast to void* or char*. Some people, notably C++ programmers, prefer to use 0 rather than NULL.

The null pointer is used in three ways:
1) To stop indirection in a recursive data structure
2) As an error value

3) As a sentinel value

Q58) What is a *void* pointer?
Answer:
A void pointer is a C convention for "a raw address." The compiler has no idea what type of object a void
Pointer "really points to." If you write
int *ip;
ip points to an int. If you write
void *p;
p doesn't point to a void!
In C and C++, any time you need a void pointer, you can use another pointer type. For example, if you have
a char*, you can pass it to a function that expects a void*. You don't even need to cast it. In C (but not
in C++), you can use a void* any time you need any kind of pointer, without casting. (In C++, you need to
cast it).

A void pointer is used for working with raw memory or for passing a pointer to an unspecified type.
Some C code operates on raw memory. When C was first invented, character pointers (char *) were used
for that. Then people started getting confused about when a character pointer was a string, when it was a
character array, and when it was raw memory.

Q59) Is *NULL* always defined as 0?
Answer:
NULL is defined as either 0 or (void*)0. These values are almost identical; either a literal zero or a void pointer
is converted automatically to any kind of pointer, as necessary, whenever a pointer is needed (although the
compiler can't always tell when a pointer is needed).

Q60) What does it mean when a pointer is used in an *if* statement?
Answer:
Any time a pointer is used as a condition, it means "Is this a non-null pointer?" A pointer can be used in an
if, while, for, or do/while statement, or in a conditional expression.

Q61) Can you add pointers together? Why would you?
Answer:
No, you can't add pointers together. If you live at 1332 Lakeview Drive, and your neighbor lives at 1364
Lakeview, what's 1332+1364? It's a number, but it doesn't mean anything. If you try to perform this type
of calculation with pointers in a C program, your compiler will complain.

The only time the addition of pointers might come up is if you try to add a pointer and the difference of two
pointers.

Q62) How do you use a pointer to a function?
Answer:
The hardest part about using a pointer-to-function is declaring it.
Consider an example. You want to create a pointer, pf, that points to the strcmp() function.
The strcmp() function is declared in this way:
int strcmp(const char *, const char *)
To set up pf to point to the strcmp() function, you want a declaration that looks just like the strcmp() function's declaration, but that has *pf rather than strcmp:
int (*pf)(const char *, const char *);
After you've gotten the declaration of pf, you can #include <string.h> and assign the address of strcmp() to pf: pf = strcmp;

Q63) When would you use a pointer to a function?
Answer:
Pointers to functions are interesting when you pass them to other functions. A function that takes function pointers says, in effect, "Part of what I do can be customized. Give me a pointer to a function, and I'll call it when that part of the job needs to be done. That function can do its part for me." This is known as a "callback." It's used a lot in graphical user interface libraries, in which the style of a display is built into the library but the contents of the display are part of the application.
As a simpler example, say you have an array of character pointers (char*s), and you want to sort it by the value
of the strings the character pointers point to. The standard qsort() function uses function pointers to perform that task. (For more on sorting, see Chapter III, "Sorting and Searching Data.") qsort() takes four arguments,

- a pointer to the beginning of the array,
- the number of elements in the array,
- the size of each array element, and
- a comparison function, and returns an int.

Q64) Why should we assign NULL to the elements (pointer) after freeing them?
Answer:
This is paranoia based on long experience. After a pointer has been freed, you can no longer use the pointed-to data. The pointer is said to "dangle"; it doesn't point at anything useful. If you "NULL out" or "zero out" a pointer immediately after freeing it, your program can no longer get in trouble by using that pointer. True, you might go indirect on the null pointer instead, but that's something your debugger might be able to help you with immediately. Also, there still might be copies of the pointer that refer to the memory that has been deallocated; that's the nature of C. Zeroing out pointers after freeing them won't solve all problems;

Q65) Is it better to use *malloc()* or *calloc()*?
Answer:

Both the malloc() and the calloc() functions are used to allocate dynamic memory. Each operates slightly different from the other. malloc() takes a size and returns a pointer to a chunk of memory at least that big:
void *malloc(size_t size);
calloc() takes a number of elements, and the size of each, and returns a pointer to a chunk of memory at least big enough to hold them all:
void *calloc(size_t numElements, size_t sizeOfElement);
There's one major difference and one minor difference between the two functions. The major difference is that malloc() doesn't initialize the allocated memory. The first time malloc() gives you a particular chunk of memory, the memory might be full of zeros. If memory has been allocated, freed, and reallocated, it probably has whatever junk was left in it. That means, unfortunately, that a program might run in simple cases (when memory is never reallocated) but break when used harder (and when memory is reused). calloc() fills the allocated memory with all zero bits. That means that anything there you're going to use as a char or an int of any length, signed or unsigned, is guaranteed to be zero. Anything you're going to use as a pointer is set to all zero bits. That's usually a null pointer, but it's not guaranteed.Anything you're going to use as a float or double is set to all zero bits; that's a floating-point zero on some types of machines, but not on all.
The minor difference between the two is that calloc() returns an array of objects; malloc() returns one object. Some people use calloc() to make clear that they want an array.

Q66) What is the difference between *far* and *near*?
Answer:
As described at the beginning of this chapter, some compilers for PC compatibles use two types of pointers.
near pointers are 16 bits long and can address a 64KB range. far pointers are 32 bits long and can address
a 1MB range.
near pointers operate within a 64KB segment. There's one segment for function addresses and one segment
for data. far pointers have a 16-bit base (the segment address) and a 16-bit offset. The base is multiplied by 16, so a far pointer is effectively 20 bits long. Before you compile your code, you must tell the compiler which memory model to use. If you use a smallcode memory model, near pointers are used by default for function addresses. That means that all the functions need to fit in one 64KB segment. With a large-code model, the default is to use far function addresses. You'll get near pointers with a small data model, and far pointers with a large data model. These are just the defaults; you can declare variables and functions as explicitly near or far.
far pointers are a little slower. Whenever one is used, the code or data segment register needs to be swapped out. far pointers also have odd semantics for arithmetic and comparison. For example, the two far pointers in the preceding example point to the same address, but they would compare as different! If your program fits in a small-data, small-code memory model, your life will be easier.

Q67) When should a *far* pointer be used?
Answer:
Sometimes you can get away with using a small memory model in most of a given program. There might be just a few things that don't fit in your small data and code

segments. When that happens, you can use explicit far pointers and function declarations to get at the rest of memory. A far function can be outside the 64KB segment most functions are shoehorned into for a small-code model. (Often, libraries are declared explicitly far, so they'll work no matter what code model the program uses.)
A far pointer can refer to information outside the 64KB data segment. Typically, such pointers are used with
farmalloc() and such, to manage a heap separate from where all the rest of the data lives. If you use a small-data, large-code model, you should explicitly make your function pointers far.

Q68) What is the stack?
Answer:
The stack is where all the functions' local (auto) variables are created. The stack also contains some
information used to call and return from functions.
A "stack trace" is a list of which functions have been called, based on this information. When you start using
a debugger, one of the first things you should learn is how to get a stack trace.
The stack is very inflexible about allocating memory; everything must be deallocated in exactly the reverse
order it was allocated in. For implementing function calls, that is all that's needed. Allocating memory off
the stack is extremely efficient. One of the reasons C compilers generate such good code is their heavy use
of a simple stack.
There used to be a C function that any programmer could use for allocating memory off the stack. The
memory was automatically deallocated when the calling function returned. This was a dangerous function
to call; it's not available anymore.

Q69) Can the size of an array be declared at runtime?
Answer:
No. In an array declaration, the size must be known at compile time. You can't specify a size that's known
only at runtime. For example, if i is a variable, you can't write code like this:
char array[i]; /* not valid C */
Some languages provide this latitude. C doesn't. If it did, the stack would be more complicated, function calls would be more expensive, and programs would run a lot slower. If you know that you have an array but you won't know until runtime how big it will be, declare a pointer to it and use malloc() or calloc() to allocate the array from the heap.

Q70) What is the heap?
Answer:
The heap is where malloc(), calloc(), and realloc() get memory.
Getting memory from the heap is much slower than getting it from the stack. On the other hand, the heap

is much more flexible than the stack. Memory can be allocated at any time and deallocated in any order. Such
memory isn't deallocated automatically; you have to call free().
Recursive data structures are almost always implemented with memory from the heap. Strings often come
from there too, especially strings that could be very long at runtime. If you can keep data in a local variable (and allocate it from the stack), your code will run faster than if you put the data on the heap. Sometimes you can use a better algorithm if you use the heap—faster, or more robust, or more flexible. It's a tradeoff.
If memory is allocated from the heap, it's available until the program ends. That's great if you remember to
deallocate it when you're done. If you forget, it's a problem. A "memory leak" is some allocated memory that's
no longer needed but isn't deallocated. If you have a memory leak inside a loop, you can use up all the memory
on the heap and not be able to get any more. (When that happens, the allocation functions return a null
pointer.) In some environments, if a program doesn't deallocate everything it allocated, memory stays
unavailable even after the program ends.

Q71) What is the difference between *NULL* and *NUL*?
Answer:
NULL is a macro defined in <stddef.h> for the null pointer.
NUL is the name of the first character in the ASCII character set. It corresponds to a zero value. There's no
standard macro NUL in C, but some people like to define it.
The digit 0 corresponds to a value of 80, decimal. Don't confuse the digit 0 with the value of '\0' (NUL)!
NULL can be defined as ((void*)0), NUL as '\0'.

Q72) What is a "null pointer assignment" error? What are bus errors, memory faults, and core dumps?
Answer:
These are all serious errors, symptoms of a wild pointer or subscript.
Null pointer assignment is a message you might get when an MS-DOS program finishes executing. Some
such programs can arrange for a small amount of memory to be available "where the NULL pointer points to"
(so to speak). If the program tries to write to that area, it will overwrite the data put there by the compiler.
When the program is done, code generated by the compiler examines that area. If that data has been changed,
the compiler-generated code complains with null pointer assignment.
This message carries only enough information to get you worried. There's no way to tell, just from a null
pointer assignment message, what part of your program is responsible for the error. Some debuggers, and
some compilers, can give you more help in finding the problem.

Bus error: core dumped and Memory fault: core dumped are messages you might see from a program running under UNIX. They're more programmer friendly. Both mean that a pointer or an array subscript was wildly out of bounds. You can get these messages on a read or on a write. They aren't restricted to null pointer problems.
The core dumped part of the message is telling you about a file, called core, that has just been written in your
current directory. This is a dump of everything on the stack and in the heap at the time the program was
running. With the help of a debugger, you can use the core dump to find where the bad pointer was used.
That might not tell you why the pointer was bad, but it's a step in the right direction. If you don't have write
permission in the current directory, you won't get a core file, or the core dumped message.

Q73) How can you determine the size of an allocated portion of memory?
Answer:
You can't, really. free() can , but there's no way for your program to know the trick free() uses. Even if you disassemble the library and discover the trick, there's no guarantee the trick won't change with the next release of the compiler.

Q74) Can math operations be performed on a *void* pointer?
Answer:
No. Pointer addition and subtraction are based on advancing the pointer by a number of elements. By definition, if you have a void pointer, you don't know what it's pointing to, so you don't know the size of what it's pointing to. If you want pointer arithmetic to work on raw addresses, use character pointers.

Q75) How do you print an address?
Answer:
The safest way is to use printf() (or fprintf() or sprintf()) with the %P specification. That prints a void
pointer (void*). Different compilers might print a pointer with different formats. Your compiler will pick
a format that's right for your environment.
If you have some other kind of pointer (not a void*) and you want to be very safe, cast the pointer to a void*:
printf("%P\n", (void*) buffer);

76) Why should I prototype a function?
Answer:
A function prototype tells the compiler what kind of arguments a function is looking to receive and what
kind of return value a function is going to give back. This approach helps the compiler ensure that calls to
a function are made correctly and that no erroneous type conversions are taking place.

77) What is a static function?
Answer:

A *static* function is a function whose *scope* is limited to the current source file. Scope refers to the visibility
of a function or variable. If the function or variable is visible outside of the current source file, it is said to
have global, or external, scope. If the function or variable is not visible outside of the current source file, it
is said to have local, or static, scope.

78) Is it possible to execute code even after the program exits the *main()* function?

Answer:

The standard C library provides a function named atexit() that can be used to perform "cleanup" operations when your program terminates. You can set up a set of functions you want to perform automatically when your program exits by passing function pointers to the atexit() function.

79) Is using *exit()* the same as using *return*?

Answer:

No. The exit() function is used to exit your program and return control to the operating system. The return
statement is used to return from a function and return control to the calling function. If you issue a return
from the main() function, you are essentially returning control to the calling function, which is the operating
system. In this case, the return statement and exit() function are similar.

80) Can the *sizeof* operator be used to tell the size of an array passed to a function?

Answer:

No. There's no way to tell, at runtime, how many elements are in an array parameter just by looking at the
array parameter itself. Remember, passing an array to a function is exactly the same as passing a pointer to
the first element.

81) Is it better to use a pointer to navigate an array of values,or is it better to use a subscripted array name?

Answer:

It's easier for a C compiler to generate good code for pointers than for subscripts.

82) What is the difference between a string and an array?

Answer:

An array is an array of anything. A string is a specific kind of an array with a well-known convention to
determine its length.
There are two kinds of programming languages: those in which a string is just an array of characters, and those
in which it's a special type. In C, a string is just an array of characters (type char), with one wrinkle: a C string
always ends with a NUL character. The "value" of an array is the same as the address of (or a pointer to) the
first element; so, frequently, a C string and a pointer to char are used to mean the same thing.

An array can be any length. If it's passed to a function, there's no way the function can tell how long the array
is supposed to be, unless some convention is used. The convention for strings is NUL termination; the last
character is an ASCII NUL ('\0') character.

83) what is a modulus operator? What are the restrictions of a modulus operator?
Ans) A Modulus operator gives the remainder value. The result of x%y is obtained by (x-(x/y)*y). This operator is applied only to integral operands and cannot be applied to float or double.

84) why n++ executes faster than n+1?
Ans) The expression n++ requires a single machine instruction such as INR to carry out the increment operation whereas, n+1 requires more instructions to carry out this operation.

85) Write the equivalent expression for x%8?
Ans) x&7

86) Write expressions to swap two integers without using a temporary variable?
Ans)

```
a=a+b;
b=a-b;
a=a-b;

a=a^b;
b=b^a;
a=a^b;
```

87) Which expression always return true? Which always return false?
Ans)
expression if (a=0) always return false
expression if (a=1) always return true

88) What is storage class and what are storage variable ?
Ans)A storage class is an attribute that changes the behavior of a variable. It controls the lifetime, scope and linkage.
There are five types of storage classes
1)auto 2)static 3)extern 4)register 5)typedef

89) What are the advantages of auto variables?
Ans) 1)The same auto variable name can be used in different blocks
2)There is no side effect by changing the values in the blocks
3)The memory is economically used
4)Auto variables have inherent protection because of local scope

90) Diffenentiate between an internal static and external static variable?
Ans) An internal static variable is declared inside a block with static storage class whereas an external static variable is declared outside all the blocks in a file.An

internal static variable has persistent storage,block scope and no linkage.An external static variable has permanent storage,file scope and internal linkage.

91) What are advantages and disadvantages of external storage class?

Ans)

Advantages of external storage class

1)Persistent storage of a variable retains the latest value

2)The value is globally available

Disadvantages of external storage class

1)The storage for an external variable exists even when the variable is not needed

2)The side effect may produce surprising output

3)Modification of the program is difficult

4)Generality of a program is affected

92) Differentiate between an external variable definition and external variable declaration

S.No	External Variable Definition	External Variable Declaration
1	It creates variables	It refers to the variable already defined
2	It allocates memory	It does not allocates memory
3	The keyword extern is not used	The keyword extern is used
4	It can be initialized	It can not be initialized
5	It appears only once	It can be declared in many places
6	It must be outside all the blocks	It can appear wherever a declaration is allowed

93) Differentiate between a linker and linkage?

Ans) A linker converts an object code into an executable code by linking together the necessary build in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

94) What are the characteristics of arrays in C?

Ans) 1) An array holds elements that have the same data type

2) Array elements are stored in subsequent memory locations

3) Two-dimentional array elements are stored row by row in subsequent memory locations.

4) Array name represents the address of the starting element

5) Array size should be mentioned in the declaration. Array size must be a constant expression and not a variable.

95) When does the compiler not implicitly generate the address of the first element of an array?

Ans) Whenever an array name appears in an expression such as

- array as an operand of the sizeof operator
- array as an operand of & operator
- array as a string literal initializer for a character array

Then the compiler does not implicitly generate the address of the address of the first element of an array

96) What is modular programming?

Ans) If a program is large, it is subdivided into a number of smaller programs that are called modules or subprograms. If a complex problem is solved using more modules, this approach is known as modular programming.

97) What is a function and built-in function?

Ans) A large program is subdivided into a number of smaller programs or subprograms. Each subprogram specifies one or more actions to be performed for a large program.such subprograms are functions.

The function supports only static and extern storage classes.By default, function assumes extern storage class.functions have global scope. Only register or auto storage class is allowed in the function parameters.

Built-in functions that predefined and supplied along with the compiler are known as built-in functions.They are also known as library functions.

98) What is an argument ? differentiate between formal arguments and actual arguments?

Ans) An argument is an entity used to pass the data from calling funtion to the called funtion. Formal arguments are the arguments available in the funtion definition.They are preceded by their own data types.Actual arguments are available in the function call.

99) What is the purpose of main( ) function?

Ans) The function main( ) invokes other functions within it.It is the first function to be called when the program starts execution.

- It is the starting function
- It returns an int value to the environment that called the program
- Recursive call is allowed for main( ) also.
- It is a user-defined function
- Program execution ends when the closing brace of the function main( ) is reached.
- It has two arguments 1)argument count and 2) argument vector (represents strings passed).
- Any user-defined name can also be used as parameters for main( ) instead of argc and argv

100) What are the advantages of the functions?

Ans)

- Debugging is easier
- It is easier to understand the logic involved in the program
- Testing is easier
- Recursive call is possible
- Irrelevant details in the user point of view are hidden in functions
- Functions are helpful in generalizing the program

101) what is a method?

Ans) a way of doing something, especially a systematic way; implies an orderly logical arrangement (usually in steps)

102) What is a pointer value and address?

Ans) A pointer value is a data object that refers to a memory location. Each memory locaion is numbered in the memory.The number attached to a memory location is called the address of the location.

103) What is a pointer variable?

Ans) A pointer variable is a variable that may contain the address of another variable or any valid address in the memory.

104) Are pointers integers?

Ans) No, pointers are not integers.A pointer is an address.It is merely a positive number and not an integer.

105) How are pointer variables initialized?

Ans) Pointer variable are initialized by one of the following two ways

- Static memory allocation
- Dynamic memory allocation

106)What is static memory allocation and dynamic memory allocation?

Ans)

Static memory allocation: The compiler allocates the required memory space for a declared variable.By using the address of operator,the reserved address is obtained and this address may be assigned to a pointer variable.Since most of the declared variable have static memory,this way of assigning pointer value to a pointer variable is known as static memory allocation. memory is assigned during compilation time.

Dynamic memory allocation: It uses functions such as malloc( ) or calloc( ) to get memory dynamically.If these functions are used to get memory dynamically and the values returned by these functions are assingned to pointer variables, such assignments are known as dynamic memory allocation.memory is assined during run time.

107)What is the purpose of realloc( )?

Ans) the function realloc(ptr,n) uses two arguments.the first argument ptr is a pointer to a block of memory for which the size is to be altered.The second argument n specifies the new size.The size may be increased or decreased.If n is greater than the old size and if sufficient space is not available subsequent to the old region, the function realloc( ) may create a new region and all the old data are moved to the new region.

108)Diffence arrays and pointers?
Ans)

- Pointers are used to manipulate data using the address. Pointers use * operator to access the data pointed to by them
- Arrays use subscripted variables to access and manipulate data.Array variables can be equivalently written using pointer expression.

106) Differentiate between an array of pointers and a pointer to an array?
Ans)

SNo	Array of pointers	Pointer to an array
1	Declaration is data_type *array_name[size];	Declaration is Data_type (*array_name)[size];
2	Size represents the row size	size represents the column size
3	The space for columns may be dynamically allocated	The space for rows may be dynamically allocated

107) How is pointer to function declared?
Ans) data_type (*function_name)(arg1,arg2,……,argN);
E.g. double(*pf)(int x,floaty)

108) How is function returning pointer is declared?
Ans) data_type *function_name(arg1,arg2,…….,argN)
double *f (double,double)

109) Declare an array of N pointers to a function returning pointer to a funtion returning a pointer to a character.
Ans) char *(*(*x[n]( ))( ));

110) How to cast a null pointer in a function call?
Ans) Only 3 ways are given below:

- #define NULL ((void *) 0)
- It automatically converts 0 to the respective pointer type
- #define NULLL(type) (type*) 0
- It passes the null pointer type needed in the function call
- Explicitly type cast the null pointer in the respective function call

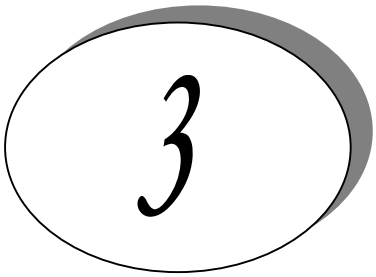
111) Differentiate between memcpy( ) and memmove( )?
Ans)

	memcpy( )	memmove( )
1	memcpy(void *s,const void *t,int n)	memmove(void *s,cons void *t,int n)
2	It copies the first n characters from t to s, and return s	It copies the first n characters from t to s, and return s.It works even if the objects s and t overlap
3	It is not safer to use when there is over lap	It is safer to use because of its guaranteed behavior even if there is an overlap
4	It is more efficiently implemented	It is preferable compared to memcpy

112) Difference between memchr( ) and memset( )
Ans)

	memchr( )	memset( )
1	memchr(void *s,int c,int n)	memset(void *s,int c,int n)
2	It return pointer to first occurrence of character c in s or NULL if c is present among first n characters	It places character c into first n characters of s and return s

C++ Aptitude and OOPS



Note : All the programs are tested under Turbo C++ 3.0, 4.5 and Microsoft VC++ 6.0 compilers.

It is assumed that,

- Programs run under Windows environment,
- The underlying machine is an x86 based system,
- Program is compiled using Turbo C/C++ compiler.

The program output may depend on the information based on this assumptions (for example sizeof(int) == 2 may be assumed).

1)
```
class Sample
{
public:
      int *ptr;
      Sample(int i)
      {
      ptr = new int(i);
      }
      ~Sample()
      {
      delete ptr;
      }
      void PrintVal()
      {
      cout << "The value is " << *ptr;
      }
          };
void SomeFunc(Sample x)
{
cout << "Say i am in someFunc " << endl;
}
int main()
{
Sample s1= 10;
SomeFunc(s1);
s1.PrintVal();
}
```

Answer:

Say i am in someFunc
Null pointer assignment(Run-time error)

Explanation:

As the object is passed by value to SomeFunc the destructor of the object is called when the control returns from the function. So when PrintVal is called it meets up with ptr that has been freed.The solution is to pass the Sample object by *reference* to SomeFunc:

```
void SomeFunc(Sample &x)
{
cout << "Say i am in someFunc " << endl;
}
```

because when we pass objects by refernece that object is not destroyed. while returning from the function.

1) Which is the parameter that is added to every non-static member function when it is called?

Answer:

'this' pointer

3)
```
class base
    {
    public:
    int bval;
    base(){ bval=0;}
    };

class deri:public base
    {
    public:
    int dval;
    deri(){ dval=1;}
    };
void SomeFunc(base *arr,int size)
{
for(int i=0; i<size; i++,arr++)
    cout<<arr->bval;
cout<<endl;
}

int main()
{
base BaseArr[5];
SomeFunc(BaseArr,5);
deri DeriArr[5];
SomeFunc(DeriArr,5);
}
```

Answer:

00000
01010

Explanation:

The function SomeFunc expects two arguments.The first one is a pointer to an array of base class objects and the second one is the sizeof the array.The first call of someFunc calls it with an array of bae objects, so it works correctly and prints the bval of all the objects. When Somefunc is called the second time the argument passed is the pointeer to an array of derived class objects and not the array of base class objects. But that is what the function expects to be sent. So the derived class pointer is promoted to base class pointer and the address is sent to the function. SomeFunc() knows nothing about this and just treats the pointer as an array of base class objects. So when arr++ is met, the size of base class object is taken into consideration and is incremented by sizeof(int) bytes for bval (the deri class objects have bval and dval as members and so is of size >= sizeof(int)+sizeof(int)).

4)
```
class base
    {
    public:
        void baseFun(){ cout<<"from base"<<endl;}
    };
class deri:public base
    {
    public:
        void baseFun(){ cout<< "from derived"<<endl;}
    };
void SomeFunc(base *baseObj)
{
    baseObj->baseFun();
}
int main()
{
base baseObject;
SomeFunc(&baseObject);
deri deriObject;
SomeFunc(&deriObject);
}
```

Answer:

from base
from base

Explanation:

As we have seen in the previous case, SomeFunc expects a pointer to a base class. Since a pointer to a derived class object is passed, it treats the argument only as a base class pointer and the corresponding base function is called.

5)
```
class base
    {
    public:
        virtual void baseFun(){ cout<<"from base"<<endl;}
    };
class deri:public base
    {
    public:
```

```
            void baseFun(){ cout<< "from derived"<<endl;}
    };
void SomeFunc(base *baseObj)
{
     baseObj->baseFun();
}
int main()
{
base baseObject;
SomeFunc(&baseObject);
deri deriObject;
SomeFunc(&deriObject);
}
```

Answer:

from base
from derived

Explanation:

Remember that baseFunc is a virtual function. That means that it supports run-time polymorphism. So the function corresponding to the derived class object is called.

```
6) void main()
{
    int a, *pa, &ra;
    pa = &a;
    ra = a;
    cout <<"a="<<a <<"*pa="<<*pa <<"ra"<<ra ;
}
/*
```

Answer :

Compiler Error: 'ra',reference must be initialized

Explanation :

Pointers are different from references. One of the main differences is that the pointers can be both initialized and assigned, whereas references can only be initialized. So this code issues an error.

```
*/

const int size = 5;
void print(int *ptr)
{
    cout<<ptr[0];
}

void print(int ptr[size])
{
    cout<<ptr[0];
}
```

```
void main()
{
	int a[size] = {1,2,3,4,5};
	int *b = new int(size);
	print(a);
	print(b);
}
/*
```
Answer:
	Compiler Error : function 'void print(int *)' already has a body
Explanation:
	Arrays cannot be passed to functions, only pointers (for arrays, base addresses) can be passed. So the arguments int *ptr and int prt[size] have no difference as function arguments. In other words, both the functoins have the same signature and so cannot be overloaded.
```
*/

class some{
public:
	~some()
	{
		cout<<"some's destructor"<<endl;
	}
};

void main()
{
	some s;
	s.~some();
}
/*
```
Answer:
	some's destructor
	some's destructor
Explanation:
	Destructors can be called explicitly. Here 's.~some()' explicitly calls the destructor of 's'. When main() returns, destructor of s is called again, hence the result.
```
*/

#include <iostream.h>

class fig2d
{
	int dim1;
	int dim2;

public:
	fig2d() { dim1=5; dim2=6;}
```

```
    virtual void operator<<(ostream & rhs);
};

void fig2d::operator<<(ostream &rhs)
{
    rhs <<this->dim1<<" "<<this->dim2<<" ";
}

/*class fig3d : public fig2d
{
    int dim3;
public:
    fig3d() { dim3=7;}
    virtual void operator<<(ostream &rhs);
};
void fig3d::operator<<(ostream &rhs)
{
    fig2d::operator <<(rhs);
    rhs<<this->dim3;
}
*/

void main()
{
    fig2d obj1;
//  fig3d obj2;

    obj1 << cout;
//  obj2 << cout;
}
/*
```

Answer :

5 6

Explanation:

In this program, the << operator is overloaded with ostream as argument. This enables the 'cout' to be present at the right-hand-side. Normally, 'cout' is implemented as global function, but it doesn't mean that 'cout' is not possible to be overloaded as member function.

Overloading << as virtual member function becomes handy when the class in which it is overloaded is inherited, and this becomes available to be overrided. This is as opposed to global friend functions, where friend's are not inherited.

```
*/

class opOverload{
public:
    bool operator==(opOverload temp);
};
```

```
bool opOverload::operator==(opOverload temp){
    if(*this  == temp ){
            cout<<"The both are same objects\n";
            return true;
    }
    else{
            cout<<"The both are different\n";
            return false;
    }
}

void main(){
    opOverload a1, a2;
    a1= =a2;
}
```

Answer :
Runtime Error: Stack Overflow

Explanation :
Just like normal functions, operator functions can be called recursively. This program just illustrates that point, by calling the operator == function recursively, leading to an infinite loop.

```
class complex{
    double re;
    double im;
public:
    complex() : re(1),im(0.5) {}
    bool operator==(complex &rhs);
    operator int(){}
};

bool complex::operator == (complex &rhs){
    if((this->re == rhs.re) && (this->im == rhs.im))
            return true;
    else
            return false;
}

int main(){
    complex  c1;
    cout<<  c1;
}
```

Answer : Garbage value

Explanation:

The programmer wishes to print the complex object using output re-direction operator,which he has not defined for his lass.But the compiler instead of giving an error sees the conversion function and converts the user defined object to standard object and prints some garbage value.

```
class complex{
    double re;
    double im;
public:
    complex() : re(0),im(0) {}
    complex(double n) { re=n,im=n;};
    complex(int m,int n) { re=m,im=n;}
    void print() { cout<<re; cout<<im;}
};

void main(){
    complex c3;
    double i=5;
    c3 = i;
    c3.print();
}
```

Answer:

5,5

Explanation:

Though no operator= function taking complex, double is defined, the double on the rhs is converted into a temporary object using the single argument constructor taking double and assigned to the lvalue.

```
void main()
{
    int a, *pa, &ra;
    pa = &a;
    ra = a;
    cout <<"a="<<a <<"*pa="<<*pa <<"ra"<<ra ;
}
```

Answer :

Compiler Error: 'ra',reference must be initialized

Explanation :

Pointers are different from references. One of the main differences is that the pointers can be both initialized and assigned, whereas references can only be initialized. So this code issues an error.

Try it Yourself

1) Determine the output of the 'C++' Codelet.

```
class base
{
public :
        out()
        {
                cout<<"base ";
        }
};
class deri{
public : out()
{
cout<<"deri ";
}
};
void main()
{       deri dp[3];
        base *bp = (base*)dp;
        for (int i=0; i<3;i++)
        (bp++)->out();
}
```

2) Justify the use of virtual constructors and destructors in C++.

3) Each C++ object possesses the 4 member fns,(which can be declared by the programmer explicitly or by the implementation if they are not available). What are those 4 functions?

4) What is wrong with this class declaration?

```
class something
{
        char *str;
        public:
          something(){
          st = new char[10]; }
         ~something()
          {
                delete str;
          }
 };
```

5) Inheritance is also known as -------- relationship. Containership as _____ relationship.

6) When is it necessary to use member-wise initialization list (also known as header initialization list) in C++?

7) Which is the only operator in C++ which can be overloaded but NOT inherited.

8) Is there anything wrong with this C++ class declaration?

```
class temp
{
 int value1;
 mutable int value2;
 public :
        void fun(int val)
        const{
        ((temp*) this)->value1 = 10;
        value2 = 10;
        }
};
```

1. What is a modifier?

Answer:

A modifier, also called a modifying function is a member function that changes the value of at least one data member. In other words, an operation that modifies the state of an object. Modifiers are also known as 'mutators'.

2. What is an accessor?

Answer:

An accessor is a class operation that does not modify the state of an object. The accessor functions need to be declared as *const* operations

3. Differentiate between a template class and class template.

Answer:

Template class:

A generic definition or a parameterized class not instantiated until the client provides the needed information. It's jargon for plain templates.

Class template:

A class template specifies how individual classes can be constructed much like the way a class specifies how individual objects can be constructed. It's jargon for plain classes.

4. When does a name clash occur?

Answer:

A name clash occurs when a name is defined in more than one place. For example., two different class libraries could give two different classes the same name. If you try to use many class libraries at the same time, there is a fair chance that you will be unable to compile or link the program because of name clashes.

5. Define namespace.

Answer:

It is a feature in c++ to minimize name collisions in the global name space. This namespace keyword assigns a distinct name to a library that allows other libraries to use the same identifier names without creating any name collisions. Furthermore, the compiler uses the namespace signature for differentiating the definitions.

6. What is the use of 'using' declaration.

Answer:

A using declaration makes it possible to use a name from a namespace without the scope operator.

7. What is an Iterator class?

Answer:

A class that is used to traverse through the objects maintained by a container class. There are five categories of iterators:

input iterators,
output iterators,
forward iterators,
bidirectional iterators,
random access.

An iterator is an entity that gives access to the contents of a container object without violating encapsulation constraints. Access to the contents is granted on a one-at-a-time basis in order. The order can be storage order (as in lists and queues) or some arbitrary order (as in array indices) or according to some ordering relation (as in an ordered binary tree). The iterator is a construct, which provides an interface that, when called, yields either the next element in the container, or some value denoting the fact that there are no more elements to examine. Iterators hide the details of access to and update of the elements of a container class.

The simplest and safest iterators are those that permit read-only access to the contents of a container class. The following code fragment shows how an iterator might appear in code:

```
cont_iter:=new cont_iterator();
x:=cont_iter.next();
while x/=none do
    ...
    s(x);
    ...
    x:=cont_iter.next();
end;
```

In this example, cont_iter is the name of the iterator. It is created on the first line by instantiation of cont_iterator class, an iterator class defined to iterate over some container class, cont. Succesive elements from the container are carried to x. The loop terminates when x is bound to some empty value. (Here, none)In the middle of the loop, there is s(x) an operation on x, the current element from the container. The next element of the container is obtained at the bottom of the loop.

9. List out some of the OODBMS available.

Answer:

- GEMSTONE/OPAL of Gemstone systems.
- ONTOS of Ontos.
- Objectivity of Objectivity inc.
- Versant of Versant object technology.
- Object store of Object Design.
- ARDENT of ARDENT software.
- POET of POET software.

10. List out some of the object-oriented methodologies.

Answer:

- Object Oriented Development (OOD) (Booch 1991,1994).
- Object Oriented Analysis and Design (OOA/D) (Coad and Yourdon 1991).
- Object Modelling Techniques (OMT) (Rumbaugh 1991).
- Object Oriented Software Engineering (Objectory) (Jacobson 1992).
- Object Oriented Analysis (OOA) (Shlaer and Mellor 1992).
- The Fusion Method (Coleman 1991).

11. What is an incomplete type?

Answer:

Incomplete types refers to pointers in which there is non availability of the implementation of the referenced location or it points to some location whose value is not available for modification.

Example:

```
int *i=0x400  // i points to address 400
*i=0;        //set the value of memory location pointed by i.
```

Incomplete types are otherwise called uninitialized pointers.

12. What is a dangling pointer?

Answer:

A dangling pointer arises when you use the address of an object after its lifetime is over.

This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed.

13. Differentiate between the message and method.

Answer:

Message	***Method***
Objects communicate by sending messages to each other.	Provides response to a message.
A message is sent to invoke a method.	It is an implementation of an operation.

14. What is an adaptor class or Wrapper class?

Answer:

A class that has no functionality of its own. Its member functions hide the use of a third party software component or an object with the non-compatible interface or a non- object- oriented implementation.

15. What is a Null object?

Answer:

It is an object of some class whose purpose is to indicate that a real object of that class does not exist. One common use for a null object is a return value from a member function that is supposed to return an object with some specified properties but cannot find such an object.

16. What is class invariant?

Answer:

A class invariant is a condition that defines all valid states for an object. It is a logical condition to ensure the correct working of a class. Class invariants must hold when an object is created, and they must be preserved under all operations of the class. In particular all class invariants are both preconditions and post-conditions for all operations or member functions of the class.

17. What do you mean by Stack unwinding?

Answer:

It is a process during exception handling when the destructor is called for all local objects between the place where the exception was thrown and where it is caught.

18. Define precondition and post-condition to a member function.

Answer:

Precondition:

A precondition is a condition that must be true on entry to a member function. A class is used correctly if preconditions are never false. An operation is not responsible for doing anything sensible if its precondition fails to hold.

For example, the interface invariants of *stack class* say nothing about pushing yet another element on a stack that is already full. We say that *isful()* is a precondition of the *push* operation.

Post-condition:

A post-condition is a condition that must be true on exit from a member function if the precondition was valid on entry to that function. A class is implemented correctly if post-conditions are never false.

For example, after pushing an element on the stack, we know that *isempty()* must necessarily hold. This is a post-condition of the *push* operation.

19. What are the conditions that have to be met for a condition to be an invariant of the class?

Answer:

- The condition should hold at the end of every constructor.
- The condition should hold at the end of every mutator(non-const) operation.

20. What are proxy objects?

Answer:

Objects that stand for other objects are called proxy objects or surrogates.

Example:

```
template<class T>
class Array2D
{
    public:
        class Array1D
        {
            public:
          T& operator[] (int index);
          const T& operator[] (int index) const;
          ...
        };
        Array1D operator[] (int index);
        const Array1D operator[] (int index) const;
        ...
};
```

The following then becomes legal:

```
Array2D<float>data(10,20);
.....
cout<<data[3][6];    // fine
```

Here data[3] yields an Array1D object and the operator [] invocation on that object yields the float in position(3,6) of the original two dimensional array. Clients

of the Array2D class need not be aware of the presence of the Array1D class. Objects of this latter class stand for one-dimensional array objects that, conceptually, do not exist for clients of Array2D. Such clients program as if they were using real, live, two-dimensional arrays. Each Array1D object stands for a one-dimensional array that is absent from a conceptual model used by the clients of Array2D. In the above example, Array1D is a proxy class. Its instances stand for one-dimensional arrays that, conceptually, do not exist.

21. Name some pure object oriented languages.

Answer:

- Smalltalk,
- Java,
- Eiffel,
- Sather.

22. Name the operators that cannot be overloaded.

Answer:

sizeof . .* .-> :: ?:

23. What is a node class?

Answer:

A node class is a class that,

- relies on the base class for services and implementation,
- provides a wider interface to te users than its base class,
- relies primarily on virtual functions in its public interface
- depends on all its direct and indirect base class
- can be understood only in the context of the base class
- can be used as base for further derivation
- can be used to create objects.

A node class is a class that has added new services or functionality beyond the services inherited from its base class.

24. What is an orthogonal base class?

Answer:

If two base classes have no overlapping methods or data they are said to be independent of, or orthogonal to each other. Orthogonal in the sense means that two classes operate in different dimensions and do not interfere with each other in any way. The same derived class may inherit such classes with no difficulty.

25. What is a container class? What are the types of container classes?

Answer:

A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behavior and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

26. What is a protocol class?

Answer:

An abstract class is a protocol class if:

- it neither contains nor inherits from classes that contain member data, non-virtual functions, or private (or protected) members of any kind.
- it has a non-inline virtual destructor defined with an empty implementation,
- all member functions other than the destructor including inherited functions, are declared pure virtual functions and left undefined.

27. What is a mixin class?

Answer:

A class that provides some but not all of the implementation for a virtual base class is often called mixin. Derivation done just for the purpose of redefining the virtual functions in the base classes is often called mixin inheritance. Mixin classes typically don't share common bases.

28. What is a concrete class?

Answer:

A concrete class is used to define a useful object that can be instantiated as an automatic variable on the program stack. The implementation of a concrete class is defined. The concrete class is not intended to be a base class and no attempt to minimize dependency on other classes in the implementation or behavior of the class.

29.What is the handle class?

Answer:

A handle is a class that maintains a pointer to an object that is programmatically accessible through the public interface of the handle class.

Explanation:

In case of abstract classes, unless one manipulates the objects of these classes through pointers and references, the benefits of the virtual functions are lost. User code may become dependent on details of implementation classes because an abstract type cannot be allocated statistically or on the stack without its size being known. Using pointers or references implies that the burden of memory management falls on the user. Another limitation of abstract class object is of fixed size. Classes however are used to represent concepts that require varying amounts of storage to implement them.

A popular technique for dealing with these issues is to separate what is used as a single object in two parts: a handle providing the user interface and a representation holding all or most of the object's state. The connection between the handle and the representation is typically a pointer in the handle. Often, handles have a bit more data than the simple representation pointer, but not much more. Hence the layout of the handle is typically stable, even when the representation changes and also that handles are small enough to move around relatively freely so that the user needn’t use the pointers and the references.

30. What is an action class?

Answer:

The simplest and most obvious way to specify an action in C++ is to write a function. However, if the action has to be delayed, has to be transmitted 'elsewhere' before being performed, requires its own data, has to be combined with other actions,

etc then it often becomes attractive to provide the action in the form of a class that can execute the desired action and provide other services as well. Manipulators used with iostreams is an obvious example.

Explanation:

A common form of action class is a simple class containing just one virtual function.

```
class Action
{
    public:
        virtual int do_it( int )=0;
        virtual ~Action( );
}
```

Given this, we can write code say a member that can store actions for later execution without using pointers to functions, without knowing anything about the objects involved, and without even knowing the name of the operation it invokes. For example:

```
class write_file : public Action
{
    File& f;
    public:
        int do_it(int)
        {
            return fwrite( ).suceed( );
        }
};
class error_message: public Action
{
    response_box db(message.cstr( ),"Continue","Cancel","Retry");
    switch (db.getresponse( ))
    {
        case 0: return 0;
        case 1: abort();
        case 2: current_operation.redo( );return 1;
    }
};
```

A user of the Action class will be completely isolated from any knowledge of derived classes such as write_file and error_message.

31. When can you tell that a memory leak will occur?

Answer:

A memory leak occurs when a program loses the ability to free a block of dynamically allocated memory.

32.What is a parameterized type?

Answer:

A template is a parameterized construct or type containing generic code that can use or manipulate any type. It is called parameterized because an actual type is a parameter of the code body. Polymorphism may be achieved through parameterized types. This type of polymorphism is called parameteric polymorphism. Parameteric

polymorphism is the mechanism by which the same code is used on different types passed as parameters.

33. Differentiate between a deep copy and a shallow copy?

Answer:

Deep copy involves using the contents of one object to create another instance of the same class. In a deep copy, the two objects may contain ht same information but the target object will have its own buffers and resources. the destruction of either object will not affect the remaining object. The overloaded assignment operator would create a deep copy of objects.

Shallow copy involves copying the contents of one object into another instance of the same class thus creating a mirror image. Owing to straight copying of references and pointers, the two objects will share the same externally contained contents of the other object to be unpredictable.

Explanation:

Using a copy constructor we simply copy the data values member by member. This method of copying is called shallow copy. If the object is a simple class, comprised of built in types and no pointers this would be acceptable. This function would use the values and the objects and its behavior would not be altered with a shallow copy, only the addresses of pointers that are members are copied and not the value the address is pointing to. The data values of the object would then be inadvertently altered by the function. When the function goes out of scope, the copy of the object with all its data is popped off the stack.

If the object has any pointers a deep copy needs to be executed. With the deep copy of an object, memory is allocated for the object in free store and the elements pointed to are copied. A deep copy is used for objects that are returned from a function.

34. What is an opaque pointer?

Answer:

A pointer is said to be opaque if the definition of the type to which it points to is not included in the current translation unit. A translation unit is the result of merging an implementation file with all its headers and header files.

35. What is a smart pointer?

Answer:

A smart pointer is an object that acts, looks and feels like a normal pointer but offers more functionality. In C++, smart pointers are implemented as *template classes* that encapsulate a pointer and override standard pointer operators. They have a number of advantages over regular pointers. They are guaranteed to be initialized as either null pointers or pointers to a heap object. Indirection through a null pointer is checked. No delete is ever necessary. Objects are automatically freed when the last pointer to them has gone away. One significant problem with these smart pointers is that unlike regular pointers, they don't respect inheritance. Smart pointers are unattractive for polymorphic code. Given below is an example for the implementation of smart pointers.

Example:

```
template <class X>
class smart_pointer
{
        public:
            smart_pointer();                    // makes a null pointer
            smart_pointer(const X& x)           // makes pointer to copy of x

            X& operator *( );
            const X& operator*( ) const;
            X* operator->() const;

            smart_pointer(const smart_pointer <X> &);
            const smart_pointer <X> & operator =(const smart_pointer<X>&);
            ~smart_pointer();
        private:
            //...
};
```

This class implement a smart pointer to an object of type X. The object itself is located on the heap. Here is how to use it:

```
smart_pointer <employee> p= employee("Harris",1333);
```

Like other overloaded operators, p will behave like a regular pointer,

```
cout<<*p;
p->raise_salary(0.5);
```

36. What is reflexive association?

Answer:

The 'is-a' is called a reflexive association because the reflexive association permits classes to bear the is-a association not only with their super-classes but also with themselves. It differs from a 'specializes-from' as 'specializes-from' is usually used to describe the association between a super-class and a sub-class. For example:

Printer is-a printer.

37. What is slicing?

Answer: Slicing means that the data added by a subclass are discarded when an object of the subclass is passed or returned by value or from a function expecting a base class object.

Explanation:

Consider the following class declaration:

```
class base
{
      ...
      base& operator =(const base&);
      base (const base&);
}
void fun( )
{
     base e=m;
     e=m;
}
```

As base copy functions don't know anything about the derived only the base part of the derived is copied. This is commonly referred to as slicing. One reason to pass objects of classes in a hierarchy is to avoid slicing. Other reasons are to preserve polymorphic behavior and to gain efficiency.

38. What is name mangling?

Answer:

Name mangling is the process through which your c++ compilers give each function in your program a unique name. In C++, all programs have at-least a few functions with the same name. Name mangling is a concession to the fact that linker always insists on all function names being unique.

Example:

In general, member names are made unique by concatenating the name of the member with that of the class e.g. given the declaration:

```
class Bar
 {
      public:
         int ival;
         ...
 };
```

ival becomes something like:

```
 // a possible member name mangling
ival__3Bar
```

Consider this derivation:

```
class Foo : public Bar
{
    public:
       int ival;
       ...
}
```

The internal representation of a Foo object is the concatenation of its base and derived class members.

```
 // Pseudo C++ code
// Internal representation of Foo
class Foo
{
   public:
      int ival__3Bar;
      int ival__3Foo;
      ...
};
```

Unambiguous access of either ival members is achieved through name mangling. Member functions, because they can be overloaded, require an extensive mangling to provide each with a unique name. Here the compiler generates the same name for the two overloaded instances(Their argument lists make their instances unique).

39. What are proxy objects?

Answer:

Objects that points to other objects are called proxy objects or surrogates. Its an object that provides the same interface as its server object but does not have any functionality. During a method invocation, it routes data to the true server object and sends back the return value to the object.

40. Differentiate between declaration and definition in C++.

Answer:

A declaration introduces a name into the program; a definition provides a unique description of an entity (e.g. type, instance, and function). Declarations can be repeated in a given scope, it introduces a name in a given scope. There must be exactly one definition of every object, function or class used in a C++ program.

A declaration is a definition unless:

- it declares a function without specifying its body,
- it contains an extern specifier and no initializer or function body,
- it is the declaration of a static class data member without a class definition,
- it is a class name definition,
- it is a typedef declaration.

A definition is a declaration unless:

- it defines a static class data member,
- it defines a non-inline member function.

41. What is cloning?

Answer:

An object can carry out copying in two ways i.e. it can set itself to be a copy of another object, or it can return a copy of itself. The latter process is called cloning.

42. Describe the main characteristics of static functions.

Answer:

The main characteristics of static functions include,

- It is without the a this pointer,
- It can't directly access the non-static members of its class
- It can't be declared const, volatile or virtual.
- It doesn't need to be invoked through an object of its class, although for convenience, it may.

43. Will the inline function be compiled as the inline function always? Justify.

Answer:

An inline function is a request and not a command. Hence it won't be compiled as an inline function always.

Explanation:

Inline-expansion could fail if the inline function contains loops, the address of an inline function is used, or an inline function is called in a complex expression. The rules for inlining are compiler dependent.

44. Define a way other than using the keyword inline to make a function inline.

Answer:

The function must be defined inside the class.

45. How can a '::' operator be used as unary operator?

Answer:

The scope operator can be used to refer to members of the global namespace. Because the global namespace doesn't have a name, the notation :: member-name refers to a member of the global namespace. This can be useful for referring to members of global namespace whose names have been hidden by names declared in nested local scope. Unless we specify to the compiler in which namespace to search for a declaration, the compiler simple searches the current scope, and any scopes in which the current scope is nested, to find the declaration for the name.

46. What is placement new?

Answer:

When you want to call a constructor directly, you use the placement new. Sometimes you have some raw memory that's already been allocated, and you need to construct an object in the memory you have. Operator new's special version placement new allows you to do it.

```
class Widget
{
    public :
        Widget(int widgetsize);
        ...
        Widget* Construct_widget_int_buffer(void *buffer,int widgetsize)
        {
            return new(buffer) Widget(widgetsize);
        }
};
```

This function returns a pointer to a Widget object that's constructed within the buffer passed to the function. Such a function might be useful for applications using shared memory or memory-mapped I/O, because objects in such applications must be placed at specific addresses or in memory allocated by special routines.

C++ FAQ

1) What is a class?

Ans) Class is a user-defined data type in C++. It can be created to solve a particular kind of problem. After creation the user need not know the specifics of the working of a class.

2) What is an object?

Ans) Object is a software bundle of variables and related methods. Objects have state and behavior.

2) What is public, protected, private?

Ans)

- Public, protected and private are three access specifiers in C++.
- Public data members and member functions are accessible outside the class.
- Protected data members and member functions are only available to derived classes.
- Private data members and member functions can't be accessed outside the class.

However there is an exception can be using friend classes.

4) What is the difference between class and structure?

Ans)

Structure: Initially (in C) a structure was used to bundle different type of data types together to perform a particular functionality. But C++ extended the structure to contain functions also. The major difference is that all declarations inside a structure are by default public.

Class: Class is a successor of Structure. By default all the members inside the class are private.

5) What is friend function?

Ans) As the name suggests, the function acts as a friend to a class. As a friend of a class, it can access its private and protected members. A friend function is not a member of the class. But it must be listed in the class definition.

6) What are virtual functions?

Ans) A virtual function allows derived classes to replace the implementation provided by the base class. The compiler makes sure the replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer. This allows algorithms in the base class to be replaced in the derived class, even if users don't know about the derived class.

7) What is a scope resolution operator?

A scope resolution operator (**::**), can be used to define the member functions of a class outside the class.

8) What do you mean by inheritance?
Ans) Inheritance is the process of creating new classes, called derived classes, from existing classes or base classes. The derived class inherits all the capabilities of the base class, but can add embellishments and refinements of its own.

9) What is polymorphism? Explain with an example?
Ans) "Poly" means "many" and "morph" means "form". Polymorphism is the ability of an object (or reference) to assume (be replaced by) or become many different forms of object.
Example: function overloading, function overriding, virtual functions.
Another example can be a plus '+' sign, used for adding two integers or for using it to concatenate two strings.

10) What is the difference between an object and a class?
Ans) Classes and objects are separate but related concepts. Every object belongs to a class and every class contains one or more related objects.

- A Class is static. All of the attributes of a class are fixed before, during, and after the execution of a program. The attributes of a class don't change.
- The class to which an object belongs is also (usually) static. If a particular object belongs to a certain class at the time that it is created then it almost certainly will still belong to that class right up until the time that it is destroyed.
- An Object on the other hand has a limited lifespan. Objects are created and eventually destroyed. Also during that lifetime, the attributes of the object may undergo significant change.

11) What is encapsulation?
Ans) Packaging an object's variables within its methods is called encapsulation.

12) What is abstraction?
Ans) Abstraction is of the process of hiding unwanted details from the user.

13) What do you mean by binding of data and functions?
Ans) Encapsulation.

14) Difference between realloc() and free()?
Ans) The free subroutine frees a block of memory previously allocated by the malloc subroutine. Undefined results occur if the Pointer parameter is not a valid pointer. If the Pointer parameter is a null value, no action will occur.
The realloc subroutine changes the size of the block of memory pointed to by the Pointer parameter to the number of bytes specified by the Size parameter and returns a new pointer to the block. The pointer specified by the Pointer parameter must have been created with the malloc, calloc, or realloc subroutines and not been deallocated with the free or realloc subroutines. Undefined results occur if the Pointer parameter is not a valid pointer

16) What is function overloading and operator overloading?
Ans)
Function overloading: C++ enables several functions of the same name to be defined, as long as these functions have different sets of parameters (at least as far as their types are concerned). This capability is called function overloading. When an overloaded function is called, the C++ compiler selects the proper function by examining the number, types and order of the arguments in the call. Function overloading is commonly used to create several functions of the same name that perform similar tasks but on different data types.

Operator overloading allows existing C++ operators to be redefined so that they work on objects of user-defined classes. Overloaded operators are syntactic sugar for equivalent function calls. They form a pleasant facade that doesn't add anything fundamental to the language (but they can improve understandability and reduce maintenance costs).

17) What is virtual class and friend class?
Ans) ***Friend classes*** are used when two or more classes are designed to work together and need access to each other's implementation in ways that the rest of the world shouldn't be allowed to have. In other words, they help keep private things private. For instance, it may be desirable for class DatabaseCursor to have more privilege to the internals of class Database than main() has.

18) What do you mean by inline function?
Ans) The idea behind ***inline*** functions is to insert the code of a called function at the point where the function is called. If done carefully, this can improve the application's performance in exchange for increased compile time and possibly (but not always) an increase in the size of the generated binary executables.

19) What is a template?
Ans) Templates allow to create generic functions that admit any data type as parameters and return value without having to overload the function with all the possible data types. Until certain point they fulfill the functionality of a macro. Its prototype is any of the two following ones:

template <class *indetifier*> *function_declaration*; **template <typename** *indetifier*> *function_declaration*;

the only difference between both prototypes is the use of keyword **class** or **typename**, its use is indistinct since both expressions have exactly the same meaning and behave exactly the same way.

20) What is RTTI?
Ans) Runtime type identification (RTTI) lets you find the dynamic type of an object when you have only a pointer or a reference to the base type. RTTI is the official way in standard C++ to discover the type of an object and to convert the type of a pointer or reference (that is, dynamic typing). The need came from practical experience with C++. RTTI replaces many homegrown versions with a solid, consistent approach.

21) What is namespace?

Ans) Namespaces allow us to group a set of global classes, objects and/or functions under a name. To say it somehow, they serve to split the global scope in sub-scopes known as *namespaces*.

The form to use *namespaces* is:

namespace *identifier* **{** *namespace-body* **}**

Where *identifier* is any valid identifier and *namespace-body* is the set of classes, objects and functions that are included within the *namespace*. For example:

```
namespace general
{
  int a, b;
}
```

In this case, **a** and **b** are normal variables integrated within the **general** *namespace*. In order to access to these variables from outside the namespace we have to use the scope operator **::**. For example, to access the previous variables we would have to put:

general::a general::b

The functionality of *namespaces* is specially useful in case that there is a possibility that a global object or function can have the same name than another one, causing a redefinition error.

22) What do you mean by pure virtual functions?

Ans) A pure virtual member function is a member function that the base class forces derived classes to provide. Normally these member functions have no implementation. Pure virtual functions are equated to zero.

```
class Shape {
public:
  virtual void draw() = 0;
};
```

23) What is virtual constructors/destructors?

Ans) ***Virtual destructors:*** If an object (with a non-virtual destructor) is destroyed explicitly by applying the delete operator to a base-class pointer to the object, the base-class destructor function (matching the pointer type) is called on the object.

There is a simple solution to this problem – declare a virtual base-class destructor. This makes all derived-class destructors virtual even though they don't have the same name as the base-class destructor. Now, if the object in the hierarchy is destroyed explicitly by applying the delete operator to a base-class pointer to a derived-class object, the destructor for the appropriate class is called.

Virtual constructor: Constructors cannot be virtual. Declaring a constructor as a virtual function is a syntax error.

Does c++ support multilevel and multiple inheritance?

Yes.

What are the advantages of inheritance?

- It permits code reusability.
- Reusability saves time in program development.

• It encourages the reuse of proven and debugged high-quality software, thus reducing problem after a system becomes functional.

What is the difference between declaration and definition?
The declaration tells the compiler that at some later point we plan to present the definition of this declaration.
E.g.: void stars () //function declaration

The definition contains the actual implementation.
E.g.: void stars () // declarator

```
{
 for(int j=10; j>=0; j--) //function body
  cout<<"*";
 cout<<endl;
}
```

OOAD

1. What do you mean by analysis and design?

Ans)

Analysis:

Basically, it is the process of determining what needs to be done before how it should be done. In order to accomplish this, the developer refers the existing systems and documents. So, simply it is an art of discovery.

Design:

It is the process of adopting/choosing the one among the many, which best accomplishes the users needs. So, simply, it is compromising mechanism.

2. What are the steps involved in designing?

Ans) Before getting into the design the designer should go through the SRS prepared by the System Analyst.

The main tasks of design are Architectural Design and Detailed Design.

In Architectural Design we find what are the main modules in the problem domain.

In Detailed Design we find what should be done within each module.

3. What are the main underlying concepts of object orientation?

*Ans)O*bjects, messages, class, inheritance and polymorphism are the main concepts of object orientation.

4. What do u meant by "SBI" of an object?

Ans) SBI stands for State, Behavior and Identity. Since every object has the above three.

- *State:* It is just a value to the attribute of an object at a particular time.
- *Behaviour*: It describes the actions and their reactions of that object.
    - *Identity*: An object has an identity that characterizes its own existence. The identity makes it possible to distinguish any object in an unambiguous way, and independently from its state.

5. Differentiate persistent & non-persistent objects?

Ans) Persistent refers to an object's ability to transcend time or space. A persistent object stores/saves its state in a permanent storage system with out losing the information represented by the object.

A non-persistent object is said to be transient or ephemeral. By default objects are considered as non-persistent.

6. What do you meant by active and passive objects?

*Ans)*Active objects are one which instigate an interaction which owns a thread and they are responsible for handling control to other objects. In simple words it can be referred as *client*.

Passive objects are one, which passively waits for the message to be processed. It waits for another object that requires its services. In simple words it can be referred as

server.

Diagram:

client server
(Active) (Passive)

7. What is meant by software development method?

Ans) Software development method describes how to model and build software systems in a reliable and reproducible way. To put it simple, methods that are used to represent ones' thinking using graphical notations.

8. What are models and meta models?

Ans)Model:

It is a complete description of something (i.e. system).

Meta model:

It describes the model elements, syntax and semantics of the notation that allows their manipulation.

9. What do you meant by static and dynamic modeling?

Ans) Static modeling is used to specify structure of the objects that exist in the problem domain. These are expressed using *class*, *object* and *USECASE diagrams*.
But Dynamic modeling refers representing the object interactions during runtime. It is represented by *sequence*, *activity*, *collaboration* and *statechart diagrams*.

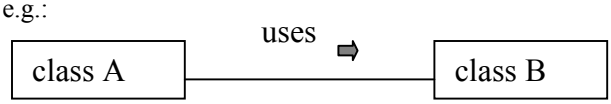
10. How to represent the interaction between the modeling elements?

Ans) Model element is just a notation to represent (Graphically) the entities that exist in the problem domain. e.g. for modeling element is class notation, object notation etc.

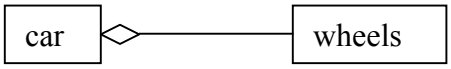
Relationships are used to represent the interaction between the modeling elements.

The following are the Relationships.

- *Association:* Its' just a semantic connection two classes.
- e.g.:



- *Aggregation:* Its' the relationship between two classes which are related in the fashion that *master and slave*. The master takes full rights than the slave. Since the slave works under the master. It is represented as line with diamond in the master area.
- ex:
- car contains wheels, etc.
- car



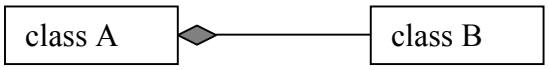
- *Containment:* This relationship is applied when the part contained with in the whole part, dies when the whole part dies.
- It is represented as darked diamond at the whole part.

example:

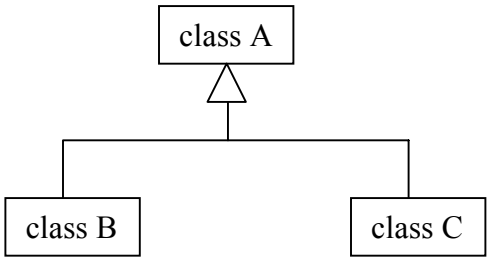
```
class A{
//some code
};

class B
{
        A aa; // an object of class A;
        // some code for class B;
};
```

In the above example we see that an object of class A is instantiated with in the class B. so the object class A dies when the object class B dies.we can represnt it in diagram like this.



- *Generalization:* This relationship used when we want represents a class, which captures the common states of objects of different classes. It is represented as arrow line pointed at the class, which has captured the common states.



- *Dependency:* It is the relationship between dependent and independent classes. Any change in the independent class will affect the states of the dependent class.
  - DIAGRAM:
    - class A class B

11. Why generalization is very strong?

Ans) Even though Generalization satisfies Structural, Interface, Behaviour properties. It is mathematically very strong, as it is Antisymmetric and Transitive.

Antisymmetric: employee is a person, but not all persons are employees. Mathematically all As' are B, but all Bs' not A.

Transitive: A=>B, B=>c then A=>c.

A. Salesman.
B. Employee.
C. Person.

Note: All the other relationships satisfy all the properties like Structural properties, Interface properties, Behaviour properties.

12. Differentiate Aggregation and containment?

Ans) Aggregation is the relationship between the whole and a part. We can add/subtract some properties in the part (slave) side. It won't affect the whole part. Best example is Car, which contains the wheels and some extra parts. Even though the parts are not there we can call it as car.

But, in the case of containment the whole part is affected when the part within that got affected. The human body is an apt example for this relationship. When the whole body dies the parts (heart etc) are died.

13. Can link and Association applied interchangeably?

Ans) No, You cannot apply the link and Association interchangeably. Since link is used represent the relationship between the two objects.

But Association is used represent the relationship between the two classes.

link :: student:Abhilash course:MCA

Association:: student course

14. what is meant by "method-wars"?

Ans) ***B***efore 1994 there were different methodologies like Rumbaugh, Booch, Jacobson, Meyer etc who followed their own notations to model the systems. The developers were in a dilemma to choose the method which best accomplishes their needs. This particular span was called as "method-wars"

15. Whether unified method and unified modeling language are same or different?

Ans) Unified method is convergence of the Rumbaugh and Booch.

Unified modeling lang. is the fusion of Rumbaugh, Booch and Jacobson as well as Betrand Meyer (whose contribution is "sequence diagram"). Its' the superset of all the methodologies.

16. Who were the three famous amigos and what was their contribution to the object community?

Ans) The Three amigos namely,

- *James Rumbaugh (OMT):* A veteran in analysis who came up with an idea about the objects and their Relationships (in particular Associations).
- *Grady Booch:* A veteran in design who came up with an idea about partitioning of systems into subsystems.
- *Ivar Jacobson (Objectory):* The father of USECASES, who described about the user and system interaction.

17. Differentiate the class representation of Booch, Rumbaugh and UML?

Ans) If you look at the class representaiton of Rumbaugh and UML, It is some what similar and both are very easy to draw.

Representation: OMT UML.

Diagram:

Booch: In this method classes are represented as "Clouds" which are not very easy to draw as for as the developer's view is concern.

Diagram:

18. What is an USECASE? Why it is needed?

Ans) A Use Case is a description of a set of sequence of actions that a system performs that yields an observable result of value to a particular action.

In SSAD process <=> In OOAD USECASE. It is represented elliptically.

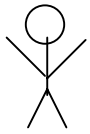
Representation:

19. Who is an Actor?

An Actor is someone or something that must interact with the system.In addition to that an Actor initiates the process(that is USECASE).

It is represented as a stickman like this.

Diagram:



20. What is guard condition?

Ans) Guard condition is one, which acts as a firewall. The access from a particular object can be made only when the particular condition is met.

For Example,

customer check customer number ATM.

Here the object on the customer accesses the ATM facility only when the guard condition is met.

21. Differentiate the following notations?

I: :obj1 :obj2

II: :obj1 :obj2

Ans) In the above representation I, obj1 sends message to obj2. But in the case of II the data is transferred from obj1 to obj2.

22. USECASE is an implementation independent notation. How will the designer give the implementation details of a particular USECASE to the programmer?

This can be accomplished by specifying the relationship called "refinement" which Talks about the two different abstraction of the same thing.

Or example,

calculate pay calculate

class1 class2 class3

23. Suppose a class acts an Actor in the problem domain, how to represent it in the static model?

Ans) In this scenario you can use “stereotype”. Since stereotype is just a string that gives extra semantic to the particular entity/model element.
It is given with in the << >>.

```
class A
<< Actor>>
attributes

methods.
```

24. Why does the function arguments are called as "signatures"?

*Ans)*The arguments distinguish functions with the same name (functional polymorphism). The name alone does not necessarily identify a unique function. However, the name and its arguments (signatures) will uniquely identify a function.

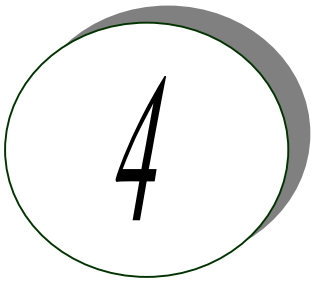
In real life we see suppose, in class there are two guys with same name, but they can be easily identified by their signatures. The same concept is applied here.

ex:

```
class person
{
    public:
        char getsex();
        void setsex(char);
        void setsex(int);
};
```

In the above example we see that there is a function setsex() with same name but with different signature.

Quantitative Aptitude



Exercise 1
Solve the following and check with the answers given at the end.

1. It was calculated that 75 men could complete a piece of work in 20 days. When work was scheduled to commence, it was found necessary to send 25 men to another project. How much longer will it take to complete the work?

2. A student divided a number by 2/3 when he required to multiply by 3/2. Calculate the percentage of error in his result.

3. A dishonest shopkeeper professes to sell pulses at the cost price, but he uses a false weight of 950gm. for a kg. His gain is …%.

4. A software engineer has the capability of thinking 100 lines of code in five minutes and can type 100 lines of code in 10 minutes. He takes a break for five minutes after every ten minutes. How many lines of codes will he complete typing after an hour?

5. A man was engaged on a job for 30 days on the condition that he would get a wage of Rs. 10 for the day he works, but he have to pay a fine of Rs. 2 for each day of his absence. If he gets Rs. 216 at the end, he was absent for work for ... days.

6. A contractor agreeing to finish a work in 150 days, employed 75 men each working 8 hours daily. After 90 days, only 2/7 of the work was completed. Increasing the number of men by _____ each working now for 10 hours daily, the work can be completed in time.

7. what is a percent of b divided by b percent of a?
   (a) a (b) b (c) 1 (d) 10
   (d) 100

8. A man bought a horse and a cart. If he sold the horse at 10 % loss and the cart at 20 % gain, he would not lose anything; but if he sold the horse at 5% loss and the cart at 5% gain, he would lose Rs. 10 in the bargain. The amount paid by him was Rs.-_____ for the horse and Rs._____ for the cart.

9. A tennis marker is trying to put together a team of four players for a tennis tournament out of seven available. males - a, b and c; females – m, n, o and p. All players are of equal ability and there must be at least two males in the team. For a

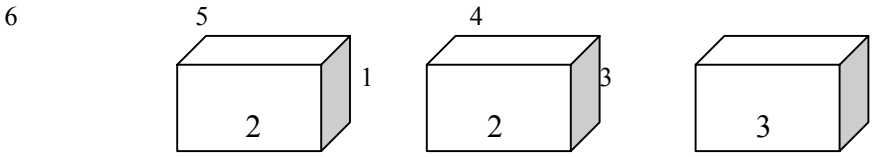
team of four, all players must be able to play with each other under the following restrictions:

b should not play with m,
c should not play with p, and
a should not play with o.

Which of the following statements must be false?

1. b and p cannot be selected together
2. c and o cannot be selected together
3. c and n cannot be selected together.

10-12. The following figure depicts three views of a cube. Based on this, answer questions 10-12.



10. The number on the face opposite to the face carrying 1 is _____ .

11. The number on the faces adjacent to the face marked 5 are _____ .

12. Which of the following pairs does not correctly give the numbers on the opposite faces.

(1) 6,5 (2) 4,1 (3) 1,3 (4) 4,2

13. Five farmers have 7, 9, 11, 13 & 14 apple trees, respectively in their orchards. Last year, each of them discovered that every tree in their own orchard bore exactly the same number of apples. Further, if the third farmer gives one apple to the first, and the fifth gives three to each of the second and the fourth, they would all have exactly the same number of apples. What were the yields per tree in the orchards of the third and fourth farmers?

14. Five boys were climbing a hill. J was following H. R was just ahead of G. K was between G & H. They were climbing up in a column. Who was the second?

15-18 John is undecided which of the four novels to buy. He is considering a spy
thriller, a Murder mystery, a Gothic romance and a science fiction novel. The books are written by Rothko, Gorky, Burchfield and Hopper, not necessary in that order, and published by Heron, Piegon, Blueja and sparrow, not necessary in that order.

(1) The book by Rothko is published by Sparrow.
(2) The Spy thriller is published by Heron.
(3) The science fiction novel is by Burchfield and is not published by Blueja.
(4)The Gothic romance is by Hopper.

15. Pigeon publishes _____.

16. The novel by Gorky _____.

17. John purchases books by the authors whose names come first and third in alphabetical order. He does not buy the books _____.

18. On the basis of the first paragraph and statement (2), (3) and (4) only, it is possible to deduce that
1. Rothko wrote the murder mystery or the spy thriller
2. Sparrow published the murder mystery or the spy thriller
3. The book by Burchfield is published by Sparrow.

19. If a light flashes every 6 seconds, how many times will it flash in ¾ of an hour?

20. If point P is on line segment AB, then which of the following is always true?
(1) AP = PB (2) AP > PB (3) PB > AP (4) AB > AP (5) AB > AP + PB

21. All men are vertebrates. Some mammals are vertebrates. Which of the following conclusions drawn from the above statement is correct.
All men are mammals
All mammals are men
Some vertebrates are mammals.
None

22. Which of the following statements drawn from the given statements are correct?
Given:
All watches sold in that shop are of high standard. Some of the HMT watches are sold in that shop.
a) All watches of high standard were manufactured by HMT.
b) Some of the HMT watches are of high standard.
c) None of the HMT watches is of high standard.
d) Some of the HMT watches of high standard are sold in that shop.

23-27.
1. Ashland is north of East Liverpool and west of Coshocton.
2. Bowling green is north of Ashland and west of Fredericktown.
3. Dover is south and east of Ashland.
4. East Liverpool is north of Fredericktown and east of Dover.
5. Fredericktown is north of Dover and west of Ashland.
6. Coshocton is south of Fredericktown and west of Dover.

23. Which of the towns mentioned is furthest of the north – west
(a) Ashland (b) Bowling green (c) Coshocton
(d) East Liverpool (e) Fredericktown

24. Which of the following must be both north and east of Fredericktown?
(a) Ashland (b) Coshocton (c) East Liverpool
I a only II b only III c only IV a & b V a & c

25. Which of the following towns must be situated both south and west of at least one other town?

A. Ashland only
B. Ashland and Fredericktown
C. Dover and Fredericktown
D. Dover, Coshocton and Fredericktown
E. Coshocton, Dover and East Liverpool.

26. Which of the following statements, if true, would make the information in the numbered statements more specific?
(a) Coshocton is north of Dover.
(b) East Liverpool is north of Dover
(c) Ashland is east of Bowling green.
(d) Coshocton is east of Fredericktown
(e) Bowling green is north of Fredericktown

27. Which of the numbered statements gives information that can be deduced from one or more of the other statements?
(A) 1 (B) 2 (C) 3 (D) 4 (E) 6

28. Eight friends Harsha, Fakis, Balaji, Eswar, Dhinesh, Chandra, Geetha, and Ahmed are sitting in a circle facing the center. Balaji is sitting between Geetha and Dhinesh. Harsha is third to the left of Balaji and second to the right of Ahmed. Chandra is sitting between Ahmed and Geetha and Balaji and Eshwar are not sitting opposite to each other. Who is third to the left of Dhinesh?

29. If every alternative letter starting from B of the English alphabet is written in small letter, rest all are written in capital letters, how the month " September" be written.
(1) SeptEMbEr (2) SEpTeMBEr (3) SeptembeR
(4) SepteMber (5) None of the above.

30. The length of the side of a square is represented by x+2. The length of the side of an equilateral triangle is 2x. If the square and the equilateral triangle have equal perimeter, then the value of x is _____.

31. It takes Mr. Karthik y hours to complete typing a manuscript. After 2 hours, he was called away. What fractional part of the assignment was left incomplete?

32. Which of the following is larger than 3/5?
(1) ½ (2) 39/50 (3) 7/25 (4) 3/10 (5) 59/100

33. The number that does not have a reciprocal is _____.

34. There are 3 persons Sudhir, Arvind, and Gauri. Sudhir lent cars to Arvind and Gauri as many as they had already. After some time Arvind gave as many cars to Sudhir and Gauri as many as they have. After sometime Gauri did the same thing. At the end of this transaction each one of them had 24. Find the cars each originally had.

35. A man bought a horse and a cart. If he sold the horse at 10 % loss and the cart at 20 % gain, he would not lose anything; but if he sold the horse at 5% loss and the cart

at 5% gain, he would lose Rs. 10 in the bargain. The amount paid by him was Rs.-_____ for the horse and Rs._____ for the cart.

Answers:

1. ***Answer:***
30 days.
Explanation:
Before:
One day work = 1 / 20
One man's one day work = 1 / (20 * 75)
Now:
No. Of workers = 50
One day work = 50 * 1 / (20 * 75)

The total no. of days required to complete the work = (75 * 20) / 50 = 30

2. ***Answer:***
0 %
Explanation:
Since 3x / 2 = x / (2 / 3)

3. ***Answer:***
5.3 %
Explanation:
He sells 950 grams of pulses and gains 50 grams.
If he sells 100 grams of pulses then he will gain (50 / 950) *100 = 5.26

4. ***Answer:***
250 lines of codes

5. ***Answer:***
7 days
Explanation:
The equation portraying the given problem is:
10 * x – 2 * (30 – x) = 216 where x is the number of working days.
Solving this we get x = 23
Number of days he was absent was 7 (30-23) days.

6. ***Answer:***
150 men.
Explanation:
One day's work = 2 / (7 * 90)
One hour's work = 2 / (7 * 90 * 8)
One man's work = 2 / (7 * 90 * 8 * 75)

The remaining work (5/7) has to be completed within 60 days, because the total number of days allotted for the project is 150 days.

So we get the equation

(2 * 10 * x * 60) / (7 * 90 * 8 * 75)= 5/7 where x is the number of men working after the 90^{th} day.

We get x = 225

Since we have 75 men already, it is enough to add only 150 men.

7. ***Answer:***

(c) 1

Explanation:

a percent of b : (a/100) * b

b percent of a : (b/100) * a

a percent of b divided by b percent of a : ((a / 100)*b) / (b/100) * a)) = 1

8. ***Answer:***

Cost price of horse = Rs. 400 & the cost price of cart = 200.

Explanation:-

Let x be the cost price of the horse and y be the cost price of the cart.

In the first sale there is no loss or profit. (i.e.) The loss obtained is equal to the gain.

Therefore (10/100) * x = (20/100) * y

X = 2 * y ----------(1)

In the second sale, he lost Rs. 10. (i.e.) The loss is greater than the profit by Rs. 10.

Therefore (5 / 100) * x = (5 / 100) * y + 10 -------(2)

Substituting (1) in (2) we get

(10 / 100) * y = (5 / 100) * y + 10

(5 / 100) * y = 10

y = 200

From (1) 2 * 200 = **x = 400**

9. ***Answer:***

3.

Explanation:

Since inclusion of any male player will reject a female from the team. Since there should be four member in the team and only three males are available, the girl, n should included in the team always irrespective of others selection.

10. ***Answer:***

5

11. ***Answer:***

1,2,3 & 4

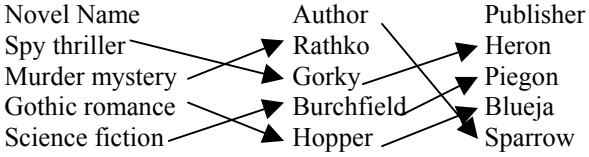
12. ***Answer:***

B

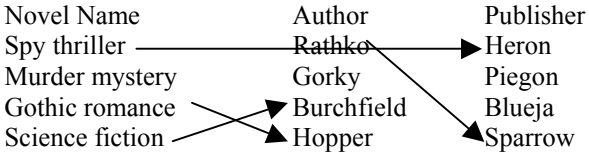
13. ***Answer:***
11 & 9 apples per tree.
Explanation:
Let a, b, c, d & e be the total number of apples bored per year in A, B, C, D & E ‘s orchard. Given that a + 1 = b + 3 = c – 1 = d + 3 = e – 6
But the question is to find the number of apples bored per tree in C and D ‘s orchard. If is enough to consider c – 1 = d + 3.
Since the number of trees in C’s orchard is 11 and that of D’s orchard is 13. Let x and y be the number of apples bored per tree in C & d ‘s orchard respectively.
Therefore 11 x – 1 = 13 y + 3
By trial and error method, we get the value for x and y as 11 and 9

14. ***Answer:***
G.
Explanation:
The order in which they are climbing is R – G – K – H – J

15 – 18
Answer:



Explanation:
Given



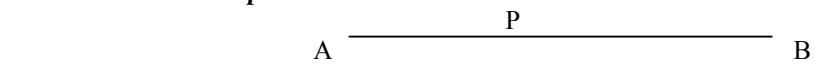
Since Blueja doesn’t publish the novel by Burchfield and Heron publishes the novel spy thriller, Piegon publishes the novel by Burchfield.
Since Hopper writes Gothic romance and Heron publishes the novel spy thriller, Blueja publishes the novel by Hopper.
Since Heron publishes the novel spy thriller and Heron publishes the novel by Gorky, Gorky writes Spy thriller and Rathko writes Murder mystery.

19. ***Answer:***
451 times.
Explanation:
There are 60 minutes in an hour.
In ¾ of an hour there are (60 * ¾) minutes = 45 minutes.
In ¾ of an hour there are (60 * 45) seconds = 2700 seconds.
Light flashed for every 6 seconds.
In 2700 seconds 2700/6 = 450 times.

The count start after the first flash, the light will flashes 451 times in ¾ of an hour.

20. ***Answer:***
(4)

Explanation:

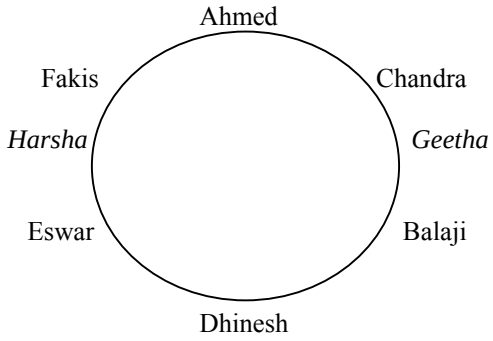


Since p is a point on the line segment AB, AB > AP

21. ***Answer:*** (c)

22. ***Answer:*** (b) & (d).

23 - 27.***Answer:***



28. ***Answer:*** Fakis
Explanation:

29. ***Answer:***
(5).
Explanation:

Since every alternative letter starting from B of the English alphabet is written in small letter, the letters written in small letter are b, d, f...

In the first two answers the letter E is written in both small & capital letters, so they are not the correct answers. But in third and fourth answers the letter is written in small letter instead capital letter, so they are not the answers**.**

30. ***Answer:***
x = 4
Explanation:
Since the side of the square is x + 2, its perimeter = 4 (x + 2) = 4x + 8
Since the side of the equilateral triangle is 2x, its perimeter = 3 * 2x = 6x
Also, the perimeters of both are equal.

(i.e.) 4x + 8 = 6x
(i.e.) 2x = 8 ➔ x = 4.

31. ***Answer:***
(y – 2) / y.
Explanation:

To type a manuscript karthik took y hours.
Therefore his speed in typing = 1/y.

He was called away after 2 hours of typing.
Therefore the work completed = 1/y * 2.
Therefore the remaining work to be completed = 1 – 2/y.
(i.e.) work to be completed = (y-2)/y

32. ***Answer:***
(2)

33. ***Answer:***
1
Explanation:
One is the only number exists without reciprocal because the reciprocal of one is one itself.

34. ***Answer:***
Sudhir had 39 cars, Arvind had 21 cars and Gauri had 12 cars.
Explanation:

	Sudhir	Arvind	Gauri
Finally	24	24	24
Before Gauri's transaction	12	12	48
Before Arvind's transaction	6	42	24
Before Sudhir' s transaction	39	21	12

35. ***Answer:***
Cost price of horse: Rs. 400 &
Cost price of cart: Rs. 200
Explanation:
Let x be the cost of horse & y be the cost of the cart.
10 % of loss in selling horse = 20 % of gain in selling the cart
Therefore (10 / 100) * x = (20 * 100) * y
➔ x = 2y ----------(1)
5 % of loss in selling the horse is 10 more than the 5 % gain in selling the cart.
Therefore (5 / 100) * x - 10 = (5 / 100) * y
➔ 5x - 1000 = 5y
Substituting (1)
10y - 1000 = 5y
5y = 1000
y = 200
x = 400 from (1)

Exercise 2.1
For the following, find the next term in the series

1. 6, 24, 60,120, 210

a) 336 b) 366 c) 330 d) 660

Answer : a) 336

Explanation : The series is 1.2.3, 2.3.4, 3.4.5, 4.5.6, 5.6.7, ... ('.' means product)

2. 1, 5, 13, 25

Answer : 41
Explanation : The series is of the form 0^2+1^2, 1^2+2^2,...

3. 0, 5, 8, 17

Answer : 24
Explanation : 1^2-1, 2^2+1, 3^2-1, 4^2+1, 5^2-1

4. 1, 8, 9, 64, 25 (Hint : Every successive terms are related)

Answer : 216
Explanation : 1^2, 2^3, 3^2, 4^3, 5^2, 6^3

5. 8,24,12,36,18,54

Answer : 27

6. 71,76,69,74,67,72
Answer : 67

7. 5,9,16,29,54
Answer : 103
Explanation : 5*2-1=9; 9*2-2=16; 16*2-3=29; 29*2-4=54; 54*2-5=103

8. 1,2,4,10,16,40,64 (Successive terms are related)
Answer : 200
Explanation : The series is powers of 2 (2^0,2^1,..).
All digits are less than 8. Every second number is in octal number system.
128 should follow 64. 128 base 10 = 200 base 8.

Exercise 2.2

Find the odd man out.

1. 3,5,7,12,13,17,19
Answer : 12
Explanation : All but 12 are odd numbers

2. 2,5,10,17,26,37,50,64
Answer : 64
Explanation : 2+3=5; 5+5=10; 10+7=17; 17+9=26; 26+11=37; 37+13=50; 50+15=65;

3. 105,85,60,30,0,-45,-90
Answer : 0

Explanation : 105-20=85; 85-25=60; 60-30=30; 30-35=-5; -5-40=-45; -45-45=-90;

Exercise 3
Solve the following.

1. What is the number of zeros at the end of the product of the numbers from 1 to 100?

Answer : 127

2. A fast typist can type some matter in 2 hours and a slow typist can type the same in 3 hours. If both type combinely, in how much time will they finish?

Answer : 1 hr 12 min

Explanation : The fast typist's work done in 1 hr = 1/2
The slow typist's work done in 1 hr = 1/3
If they work combinely, work done in 1 hr = 1/2+1/3 = 5/6
So, the work will be completed in 6/5 hours. i.e., 1+1/5 hours = 1hr 12 min

3. Gavaskar's average in his first 50 innings was 50. After the 51st innings, his average was 51. How many runs did he score in his 51st innings. (supposing that he lost his wicket in his 51st innings)

Answer : 101

Explanation :Total score after 50 innings = 50*50 = 2500
Total score after 51 innings = 51*51 = 2601
So, runs made in the 51st innings = 2601-2500 = 101
If he had not lost his wicket in his 51st innings, he would have scored an unbeaten 50 in his 51st innings.

4. Out of 80 coins, one is counterfeit. What is the minimum number of weighings needed to find out the counterfeit coin?

Answer : 4

5. What can you conclude from the statement : All green are blue, all blue are red. ?

(i) some blue are green
(ii) some red are green
(iii) some green are not red
(iv) all red are blue
(a) i or ii but not both
(b) i & ii only
(c) iii or iv but not both
(d) iii & iv

Answer : (b)

6. A rectangular plate with length 8 inches, breadth 11 inches and thickness 2 inches is available. What is the length of the circular rod with diameter 8 inches and equal to the volume of the rectangular plate?

Answer : 3.5 inches

Explanation : Volume of the circular rod (cylinder) = Volume of the rectangular plate
(22/7)*4*4*h = 8*11*2

h = 7/2 = 3.5

7. What is the sum of all numbers between 100 and 1000 which are divisible by 14 ?
Answer : 35392
Explanation : The number closest to 100 which is greater than 100 and divisible by 14 is 112, which is the first term of the series which has to be summed.
The number closest to 1000 which is less than 1000 and divisible by 14 is 994, which is the last term of the series.
112 + 126 + ... + 994 = 14(8+9+ ... + 71) = 35392

8. If s(a) denotes square root of a, find the value of s(12+s(12+s(12+ ..... upto infinity.
Answer : 4
Explanation : Let x = s(12+s(12+s(12+.....
We can write x = s(12+x). i.e., x^2 = 12 + x. Solving this quadratic equation, we get x = -3 or x=4. Sum cannot be -ve and hence sum = 4.

9. A cylindrical container has a radius of eight inches with a height of three inches. Compute how many inches should be added to either the radius or height to give the same increase in volume?
Answer : 16/3 inches
Explanation : Let x be the amount of increase. The volume will increase by the same amount if the radius increased or the height is increased.
So, the effect on increasing height is equal to the effect on increasing the radius.
i.e., (22/7)*8*8*(3+x) = (22/7)*(8+x)*(8+x)*3
Solving the quadratic equation we get the x = 0 or 16/3. The possible increase would be by 16/3 inches.

10. With just six weights and a balance scale, you can weigh any unit number of kgs from 1 to 364. What could be the six weights?
Answer : 1, 3, 9, 27, 81, 243 (All powers of 3)

11. Diophantus passed one sixth of his life in childhood, one twelfth in youth, and one seventh more as a bachelor; five years after his marriage a son was born who died four years before his father at half his final age. How old is Diophantus?
Answer : 84 years
Explanation : x/6 + x/12 + x/7 + 5 + x/2 + 4 = x

12 . If time at this moment is 9 P.M., what will be the time 23999999992 hours later?
Answer : 1 P.M.
Explanation : 24 billion hours later, it would be 9 P.M. and 8 hours before that it would be 1 P.M.

13. How big will an angle of one and a half degree look through a glass that magnifies things three times?
Answer : 1 1/2 degrees
Explanation : The magnifying glass cannot increase the magnitude of an angle.

14. Divide 45 into four parts such that when 2 is added to the first part, 2 is subtracted from the second part, 2 is multiplied by the third part and the fourth part is divided by two, all result in the same number.
Answer: 8, 12, 5, 20
Explanation: a + b + c + d =45; a+2 = b-2 = 2c = d/2; a=b-4; c = (b-2)/2; d = 2(b-2); b-4 + b + (b-2)/2 + 2(b-2) = 45;

15. I drove 60 km at 30 kmph and then an additional 60 km at 50 kmph. Compute my average speed over my 120 km.
Answer : 37 1/2
Explanation : Time reqd for the first 60 km = 120 min.; Time reqd for the second 60 km = 72 min.; Total time reqd = 192 min
Avg speed = (60*120)/192 = 37 1/2

Questions 16 and 17 are based on the following :
Five executives of European Corporation hold a Conference in Rome
Mr. A converses in Spanish & Italian
Mr. B, a spaniard, knows English also
Mr. C knows English and belongs to Italy
Mr. D converses in French and Spanish
Mr. E , a native of Italy knows French

16. Which of the following can act as interpreter if Mr. C & Mr. D wish to converse
a) only Mr. A b) Only Mr. B c) Mr. A & Mr. Bd) Any of the other three

Answer : d) Any of the other three.
Explanation : From the data given, we can infer the following.
A knows Spanish, Italian
B knows Spanish, English
C knows Italian, English
D knows Spanish, French
E knows Italian, French
To act as an interpreter between C and D, a person has to know one of the combinations Italian&Spanish, Italian&French, English&Spanish, English&French
A, B, and E know atleast one of the combinations.

17. If a 6th executive is brought in, to be understood by maximum number of original five he should be fluent in
a) English & French b) Italian & Spanish c) English & French
d) French & Italian
Answer : b) Italian & Spanish
Explanation : No of executives who know
i) English is 2
ii) Spanish is 3
iii) Italian is 3
iv) French is 2
Italian & Spanish are spoken by the maximum no of executives. So, if the 6th executive is fluent in Italian & Spanish, he can communicate with all the original five because everybody knows either Spanish or Italian.

18. What is the sum of the first 25 natural odd numbers?
Answer : 625
Explanation : The sum of the first n natural odd nos is square(n).
1+3 = 4 = square(2) 1+3+5 = 9 = square(3)

19. The sum of any seven consecutive numbers is divisible by
a) 2 b) 7 c) 3 d) 11

Exercise 3
Try the following.

1. There are seventy clerks working in a company, of which 30 are females. Also, 30 clerks are married; 24 clerks are above 25 years of age; 19 married clerks are above 25 years, of which 7 are males; 12 males are above 25 years of age; and 15 males are married. How many bachelor girls are there and how many of these are above 25?

2. A man sailed off from the North Pole. After covering 2,000 miles in one direction he turned West, sailed 2,000 miles, turned North and sailed ahead another 2,000 miles till he met his friend. How far was he from the North Pole and in what direction?

3. Here is a series of comments on the ages of three persons J, R, S by themselves.
S : The difference between R's age and mine is three years.
J : R is the youngest.
R : Either I am 24 years old or J 25 or S 26.
J : All are above 24 years of age.
S : I am the eldest if and only if R is not the youngest.
R : S is elder to me.
J : I am the eldest.
R : S is not 27 years old.
S : The sum of my age and J's is two more than twice R's age.
One of the three had been telling a lie throughout whereas others had spoken the truth. Determine the ages of S,J,R.

4. In a group of five people, what is the probability of finding two persons with the same month of birth?

5. A father and his son go out for a 'walk-and-run' every morning around a track formed by an equilateral triangle. The father's walking speed is 2 mph and his running speed is 5 mph. The son's walking and running speeds are twice that of his father. Both start together from one apex of the triangle, the son going clockwise and the father anti-clockwise. Initially the father runs and the son walks for a certain period of time. Thereafter, as soon as the father starts walking, the son starts running. Both complete the course in 45 minutes. For how long does the father run? Where do the two cross each other?

6. The Director of Medical Services was on his annual visit to the ENT Hospital. While going through the out patients' records he came across the following data for a particular day : " Ear consultations 45; Nose 50; Throat 70; Ear and Nose 30; Nose and Throat 20; Ear and Throat 30; Ear, Nose and Throat 10; Total patients 100." Then he came to the conclusion that the records were bogus. Was he right?

7. Amongst Ram, Sham and Gobind are a doctor, a lawyer and a police officer. They are married to Radha, Gita and Sita (not in order). Each of the wives have a profession. Gobind's wife is an artist. Ram is not married to Gita. The lawyer's wife is a teacher. Radha is married to the police officer. Sita is an expert cook. Who's who?

8. What should come next?
1, 2, 4, 10, 16, 40, 64,

Questions 9-12 are based on the following :

Three adults – Roberto, Sarah and Vicky – will be traveling in a van with five children – Freddy, Hillary, Jonathan, Lupe, and Marta. The van has a driver's seat and one passenger seat in the front, and two benches behind the front seats, one beach behind the other. Each bench has room for exactly three people. Everyone must sit in a seat or on a bench, and seating is subject to the following restrictions: An adult must sit on each bench.

Either Roberto or Sarah must sit in the driver's seat.

Jonathan must sit immediately beside Marta.

9. Of the following, who can sit in the front passenger seat ?
(a) Jonathan (b) Lupe (c) Roberto (d) Sarah (e) Vicky

10. Which of the following groups of three can sit together on a bench?
(a) Freddy, Jonathan and Marta (b) Freddy, Jonathan and Vicky
(c) Freddy, Sarah and Vicky (d) Hillary, Lupe and Sarah
(e) Lupe, Marta and Roberto

11. If Freddy sits immediately beside Vicky, which of the following cannot be true ?
a. Jonathan sits immediately beside Sarah
b. Lupe sits immediately beside Vicky
c. Hillary sits in the front passenger seat
d. Freddy sits on the same bench as Hillary
e. Hillary sits on the same bench as Roberto

12. If Sarah sits on a bench that is behind where Jonathan is sitting, which of the following must be true ?
a. Hillary sits in a seat or on a bench that is in front of where Marta is sitting
b. Lupe sits in a seat or on a bench that is in front of where Freddy is sitting
c. Freddy sits on the same bench as Hillary
d. Lupe sits on the same bench as Sarah
e. Marta sits on the same bench as Vicky

13. Make six squares of the same size using twelve match-sticks. (Hint : You will need an adhesive to arrange the required figure)

14. A farmer has two rectangular fields. The larger field has twice the length and 4 times the width of the smaller field. If the smaller field has area K, then the are of the larger field is greater than the area of the smaller field by what amount?
(a) 6K (b) 8K (c) 12K (d) 7K

15. Nine equal circles are enclosed in a square whose area is 36sq units. Find the area of each circle.

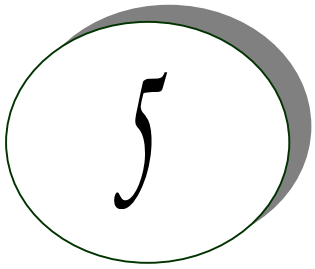
16. There are 9 cards. Arrange them in a 3*3 matrix. Cards are of 4 colors. They are red, yellow, blue, green. Conditions for arrangement: one red card must be in first row or second row. 2 green cards should be in 3rd column. Yellow cards must be in the 3 corners only. Two blue cards must be in the 2nd row. At least one green card in each row.

17. Is z less than w? z and w are real numbers.
(I) $z^2 = 25$
(II) w = 9
To answer the question,
a) Either I or II is sufficient
b) Both I and II are sufficient but neither of them is alone sufficient
c) I & II are sufficient
d) Both are not sufficient

18. A speaks truth 70% of the time; B speaks truth 80% of the time. What is the probability that both are contradicting each other?

19. In a family 7 children don't eat spinach, 6 don't eat carrot, 5 don't eat beans, 4 don't eat spinach & carrots, 3 don't eat carrot & beans, 2 don't eat beans & spinach. One doesn't eat all 3. Find the no. of children.

20. Anna, Bena, Catherina and Diana are at their monthly business meeting. Their occupations are author, biologist, chemist and doctor, but not necessarily in that order. Diana just told the neighbour, who is a biologist that Catherina was on her way with doughnuts. Anna is sitting across from the doctor and next to the chemist. The doctor was thinking that Bena was a good name for parent's to choose, but didn't say anything. What is each person's occupation?



UNIX Concepts

SECTION - I
FILE MANAGEMENT IN UNIX

1. How are devices represented in UNIX?

Ans) All devices are represented by files called *special files* that are located in/dev directory. Thus, device files and other files are named and accessed in the same way. A 'regular file' is just an ordinary data file in the disk. A 'block special file' represents a device with characteristics similar to a disk (data transfer in terms of blocks). A 'character special file' represents a device with characteristics similar to a keyboard (data transfer is by stream of bits in sequential order).

2. What is 'inode'?

Ans) All UNIX files have its description stored in a structure called 'inode'. The inode contains info about the file-size, its location, time of last access, time of last modification, permission and so on. Directories are also represented as files and have an associated inode. In addition to descriptions about the file, the inode contains pointers to the data blocks of the file. If the file is large, inode has indirect pointer to a block of pointers to additional data blocks (this further aggregates for larger files). A block is typically 8k.

Inode consists of the following fields:

- File owner identifier
- File type
- File access permissions
- File access times
- Number of links
- File size
- Location of the file data

3. Brief about the directory representation in UNIX?

Ans) A Unix directory is a file containing a correspondence between filenames and inodes. A directory is a special file that the kernel maintains. Only kernel modifies directories, but processes can read directories. The contents of a directory are a list of filename and inode number pairs. When new directories are created, kernel makes two entries named '.' (refers to the directory itself) and '..' (refers to parent directory).
System call for creating directory is mkdir (pathname, mode).

4. What are the Unix system calls for I/O?

Ans)

- open(pathname,flag,mode) - open file
- creat(pathname,mode) - create file
- close(filedes) - close an open file
- read(filedes,buffer,bytes) - read data from an open file
- write(filedes,buffer,bytes) - write data to an open file
- lseek(filedes,offset,from) - position an open file
- dup(filedes) - duplicate an existing file descriptor
- dup2(oldfd,newfd) - duplicate to a desired file descriptor
- fcntl(filedes,cmd,arg) - change properties of an open file
- ioctl(filedes,request,arg) - change the behaviour of an open file

The difference between fcntl anf ioctl is that the former is intended for any open file, while the latter is for device-specific operations.

5. How do you change File Access Permissions?

Ans) Every file has following attributes:

- owner's user ID (16 bit integer)
- owner's group ID (16 bit integer)
- File access mode word

'r w x -r w x- r w x'
(user permission-group permission-others permission)
r-read, w-write, x-execute
To change the access mode, we use chmod(filename,mode).
Example 1:
To change mode of myfile to 'rw-rw-r--' (ie. read, write permission for user - read,write permission for group - only read permission for others) we give the args as:
chmod(myfile,0664) .
Each operation is represented by discrete values
'r' is 4
'w' is 2
'x' is 1
Therefore, for 'rw' the value is 6(4+2).
Example 2:
To change mode of myfile to 'rwxr--r--' we give the args as:
chmod(myfile,0744).

6. What are links and symbolic links in UNIX file system?

Ans) A link is a second name (not a file) for a file. Links can be used to assign more than one name to a file, but cannot be used to assign a directory more than one name or link filenames on different computers.
Symbolic link 'is' a file that only contains the name of another file.Operation on the symbolic link is directed to the file pointed by the it.Both the limitations of links are eliminated in symbolic links.
Commands for linking files are:

Link	ln filename1 filename2
Symbolic link	ln -s filename1 filename2

7. What is a FIFO?

Ans) FIFO are otherwise called as 'named pipes'. FIFO (first-in-first-out) is a special file which is said to be data transient. Once data is read from named pipe, it cannot be read again. Also, data can be read only in the order written. It is used in interprocess communication where a process writes to one end of the pipe (producer) and the other reads from the other end (consumer).

8. How do you create special files like named pipes and device files?

Ans) The system call mknod creates special files in the following sequence.

1. kernel assigns new inode,
2. sets the file type to indicate that the file is a pipe, directory or special file,
3. If it is a device file, it makes the other entries like major, minor device numbers.

For example:

If the device is a disk, major device number refers to the disk controller and minor device number is the disk.

9. Discuss the mount and unmount system calls

Ans) The privileged mount system call is used to attach a file system to a directory of another file system; the unmount system call detaches a file system. When you mount another file system on to your directory, you are essentially splicing one directory tree onto a branch in another directory tree. The first argument to mount call is the mount point, that is , a directory in the current file naming system. The second argument is the file system to mount to that point. When you insert a cdrom to your unix system's drive, the file system in the cdrom automatically mounts to /dev/cdrom in your system.

10. How does the inode map to data block of a file?

Ans) Inode has 13 block addresses. The first 10 are direct block addresses of the first 10 data blocks in the file. The 11th address points to a one-level index block. The 12th address points to a two-level (double in-direction) index block. The 13th address points to a three-level(triple in-direction)index block. This provides a very large maximum file size with efficient access to large files, but also small files are accessed directly in one disk read.

11. What is a shell?

Ans) A shell is an interactive user interface to an operating system services that allows an user to enter commands as character strings or through a graphical user interface. The shell converts them to system calls to the OS or forks off a process to execute the command. System call results and other information from the OS are presented to the user through an interactive interface. Commonly used shells are sh,csh,ks etc.

SECTION - II
PROCESS MODEL and IPC

1. Brief about the initial process sequence while the system boots up. ?

Ans) While booting, special process called the 'swapper' or 'scheduler' is created with Process-ID 0. The swapper manages memory allocation for processes and influences CPU allocation. The swapper inturn creates 3 children:

- the process dispatcher,
- vhand and
- dbflush

with IDs 1,2 and 3 respectively.
This is done by executing the file /etc/init. Process dispatcher gives birth to the shell. Unix keeps track of all the processes in an internal data structure called the Process Table (listing command is ps -el).

2. What are various IDs associated with a process?

Ans) Unix identifies each process with a unique integer called ProcessID. The process that executes the request for creation of a process is called the 'parent process' whose PID is 'Parent Process ID'. Every process is associated with a particular user called the 'owner' who has privileges over the process. The identification for the user is 'UserID'. Owner is the user who executes the process. Process also has 'Effective User ID' which determines the access privileges for accessing resources like files.
getpid() -process id
getppid() -parent process id
getuid() -user id
geteuid() -effective user id

3. Explain fork() system cal?

Ans)The `fork()' used to create a new process from an existing process. The new process is called the child process, and the existing process is called the parent. We can tell which is which by checking the return value from `fork()'. The parent gets the child's pid returned to him, but the child gets 0 returned to him.

4. Predict the output of the following program code

```
main()
{
    fork();
printf("Hello World!");
}
```

Answer:
Hello World!Hello World!
Explanation:
The fork creates a child that is a duplicate of the parent process. The child begins from the fork().All the statements after the call to fork() will be executed twice.(once by the parent process and other by child). The statement before fork() is executed only by the parent process.

5. Predict the output of the following program code

```
main()
{
fork(); fork(); fork();
printf("Hello World!");
}
```

Answer:

"Hello World" will be printed 8 times.

Explanation:

2^n times where n is the number of calls to fork()

6. List the system calls used for process management:

System calls	**Description**
fork()	To create a new process
exec()	To execute a new program in a process
wait()	To wait until a created process completes its execution
exit()	To exit from a process execution
getpid()	To get a process identifier of the current process
getppid()	To get parent process identifier
nice()	To bias the existing priority of a process
brk()	To increase/decrease the data segment size of a process

7. How can you get/set an environment variable from a program?

Ans) Getting the value of an environment variable is done by using `getenv()'.
Setting the value of an environment variable is done by using `putenv()'.

8. How can a parent and child process communicate?

Ans) A parent and child can communicate through any of the normal inter-process communication schemes (pipes, sockets, message queues, shared memory), but also have some special ways to communicate that take advantage of their relationship as a parent and child. One of the most obvious is that the parent can get the exit status of the child.

9. What is a zombie?

Ans) When a program forks and the child finishes before the parent, the kernel still keeps some of its information about the child in case the parent might need it - for example, the parent may need to check the child's exit status. To be able to get this information, the parent calls `wait()'; In the interval between the child terminating and the parent calling `wait()', the child is said to be a `zombie' (If you do `ps', the child will have a `Z' in its status field to indicate this.)

10. What are the process states in Unix?

Ans) As a process executes it changes state according to its circumstances. Unix processes have the following states:
Running : The process is either running or it is ready to run .
Waiting : The process is waiting for an event or for a resource.
Stopped : The process has been stopped, usually by receiving a signal.
Zombie : The process is dead but have not been removed from the process table.

11. What Happens when you execute a program?

Ans) When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system. Each process has process context, which is everything that is unique about the state of the program you are currently running. Every time you execute a program the UNIX system does a fork, which performs a series of operations to create a process context and then execute your program in that context. The steps include the following:

- Allocate a slot in the process table, a list of currently running programs kept by UNIX.
- Assign a unique process identifier (PID) to the process.
- iCopy the context of the parent, the process that requested the spawning of the new process.
- Return the new PID to the parent process. This enables the parent process to examine or control the process directly.

After the fork is complete, UNIX runs your program.

12. What Happens when you execute a command?

Ans) When you enter 'ls' command to look at the contents of your current working directory, UNIX does a series of things to create an environment for ls and the run it: The shell has UNIX perform a fork. This creates a new process that the shell will use to run the ls program. The shell has UNIX perform an exec of the ls program. This replaces the shell program and data with the program and data for ls and then starts running that new program. The ls program is loaded into the new process context, replacing the text and data of the shell. The ls program performs its task, listing the contents of the current directory.

13. What is a Daemon?

Ans) A daemon is a process that detaches itself from the terminal and runs, disconnected, in the background, waiting for requests and responding to them. It can also be defined as the background process that does not belong to a terminal session. Many system functions are commonly performed by daemons, including the sendmail daemon, which handles mail, and the NNTP daemon, which handles USENET news. Many other daemons may exist. Some of the most common daemons are:

- init: Takes over the basic running of the system when the kernel has finished the boot process.
- inetd: Responsible for starting network services that do not have their own stand-alone daemons. For example, inetd usually takes care of incoming rlogin, telnet, and ftp connections.
- cron: Responsible for running repetitive tasks on a regular schedule.

14. What is 'ps' command for?

Ans) The ps command prints the process status for some or all of the running processes. The information given are the process identification number (PID),the amount of time that the process has taken to execute so far etc.

15. How would you kill a process?

Ans) The kill command takes the PID as one argument; this identifies which process to terminate. The PID of a process can be got using 'ps' command.

16. What is an advantage of executing a process in background?
Ans) The most common reason to put a process in the background is to allow you to do something else interactively without waiting for the process to complete. At the end of the command you add the special background symbol, &. This symbol tells your shell to execute the given command in the background.
Example: cp *.* ../backup& (cp is for copy)

17. How do you execute one program from within another?
Ans) The system calls used for low-level process creation are execlp() and execvp(). The execlp call overlays the existing program with the new one , runs that and exits. The original program gets back control only when an error occurs.
execlp(path,file_name,arguments..); //last argument must be NULL
A variant of execlp called execvp is used when the number of arguments is not known in advance.
execvp(path,argument_array); //argument array should be terminated by NULL

18. What is IPC? What are the various schemes available?
Ans) The term IPC (Inter-Process Communication) describes various ways by which different process running on some operating system communicate between each other. Various schemes available are as follows:
Pipes:
One-way communication scheme through which different process can communicate. The problem is that the two processes should have a common ancestor (parent-child relationship). However this problem was fixed with the introduction of named-pipes (FIFO).

Message Queues :
Message queues can be used between related and unrelated processes running on a machine.

Shared Memory:
This is the fastest of all IPC schemes. The memory to be shared is mapped into the address space of the processes (that are sharing). The speed achieved is attributed to the fact that there is no kernel involvement. But this scheme needs synchronization.

Various forms of synchronisation are mutexes, condition-variables, read-write locks,record-locks, and semaphores.

SECTION - III
MEMORY MANAGEMENT

1. What is the difference between Swapping and Paging?
Ans)
Swapping:
Whole process is moved from the swap device to the main memory for execution. Process size must be less than or equal to the available main memory. It is easier to implementation and overhead to the system. Swapping systems does not handle the memory more flexibly as compared to the paging systems.

Paging:
Only the required memory pages are moved to main memory from the swap device for execution. Process size does not matter. Gives the concept of the virtual memory. It provides greater flexibility in mapping the virtual address space into the physical memory of the machine. Allows more number of processes to fit in the main memory simultaneously. Allows the greater process size than the available physical memory. Demand paging systems handle the memory more flexibly.

2. What is major difference between the Historic Unix and the new BSD release of Unix System V in terms of Memory Management?
Ans) Historic Unix uses Swapping – entire process is transferred to the main memory from the swap device, whereas the Unix System V uses Demand Paging – only the part of the process is moved to the main memory. Historic Unix uses one Swap Device and Unix System V allow multiple Swap Devices.

3. What is the main goal of the Memory Management?
Ans)
- It decides which process should reside in the main memory,
- Manages the parts of the virtual address space of a process which is non-core resident,
- Monitors the available main memory and periodically write the processes into the swap device to provide more processes fit in the main memory simultaneously.

4. What is a Map?
Ans) A Map is an Array, which contains the addresses of the free space in the swap device that are allocatable resources, and the number of the resource units available there.

Address	Units
1	10,000

This allows First-Fit allocation of contiguous blocks of a resource. Initially the Map contains one entry – address (block offset from the starting of the swap area) and the total number of resources.

Kernel treats each unit of Map as a group of disk blocks. On the allocation and freeing of the resources Kernel updates the Map for accurate information.

5. What scheme does the Kernel in Unix System V follow while choosing a swap device among the multiple swap devices?
Ans) Kernel follows Round Robin scheme choosing a swap device among the multiple swap devices in Unix System V.

6. What is a Region?
Ans) A Region is a continuous area of a process's address space (such as text, data and stack). The kernel in a 'Region Table' that is local to the process maintains region. Regions are sharable among the process.

7. What are the events done by the Kernel after a process is being swapped out from the main memory?

Ans) When Kernel swaps the process out of the primary memory, it performs the following:

- Kernel decrements the Reference Count of each region of the process. If the reference count becomes zero, swaps the region out of the main memory,
- Kernel allocates the space for the swapping process in the swap device,
- Kernel locks the other swapping process while the current swapping operation is going on,
- The Kernel saves the swap address of the region in the region table.

8. Is the Process before and after the swap are the same? Give reason.

Ans) Process before swapping is residing in the primary memory in its original form. The regions (text, data and stack) may not be occupied fully by the process, there may be few empty slots in any of the regions and while swapping Kernel do not bother about the empty slots while swapping the process out.

After swapping the process resides in the swap (secondary memory) device. The regions swapped out will be present but only the occupied region slots but not the empty slots that were present before assigning.

While swapping the process once again into the main memory, the Kernel referring to the Process Memory Map, it assigns the main memory accordingly taking care of the empty slots in the regions.

9. What do you mean by u-area (user area) or u-block?

Ans) This contains the private data that is manipulated only by the Kernel. This is local to the Process, i.e. each process is allocated a u-area.

10. What are the entities that are swapped out of the main memory while swapping the process out of the main memory?

Ans) All memory space occupied by the process, process's u-area, and Kernel stack are swapped out, theoretically.

Practically, if the process's u-area contains the Address Translation Tables for the process then Kernel implementations do not swap the u-area.

11. What is Fork swap?

Ans) fork() is a system call to create a child process. When the parent process calls fork() system call, the child process is created and if there is short of memory then the child process is sent to the read-to-run state in the swap device, and return to the user state without swapping the parent process. When the memory will be available the child process will be swapped into the main memory.

12. What is Expansion swap?

Ans) At the time when any process requires more memory than it is currently allocated, the Kernel performs Expansion swap. To do this Kernel reserves enough space in the swap device. Then the address translation mapping is adjusted for the new virtual address space but the physical memory is not allocated. At last Kernel swaps the process into the assigned space in the swap device. Later when the Kernel swaps the process into the main memory this assigns memory according to the new address translation mapping.

13. How the Swapper works?
Ans)The swapper is the only process that swaps the processes. The Swapper operates only in the Kernel mode and it does not uses System calls instead it uses internal Kernel functions for swapping. It is the archetype of all kernel process.

14. What are the processes that are not bothered by the swapper? Give Reason.
Ans)
- Zombie process: They do not take any up physical memory.
- Processes locked in memories that are updating the region of the process.
- Kernel swaps only the sleeping processes rather than the 'ready-to-run' processes, as they have the higher probability of being scheduled than the Sleeping processes.

15. What are the requirements for a swapper to work?
Ans) The swapper works on the highest scheduling priority. Firstly it will look for any sleeping process, if not found then it will look for the ready-to-run process for swapping. But the major requirement for the swapper to work the ready-to-run process must be core-resident for at least 2 seconds before swapping out. And for swapping in the process must have been resided in the swap device for at least 2 seconds. If the requirement is not satisfied then the swapper will go into the wait state on that event and it is awaken once in a second by the Kernel.

16. What are the criteria for choosing a process for swapping into memory from the swap device?
Ans) The resident time of the processes in the swap device, the priority of the processes and the amount of time the processes had been swapped out.

17. What are the criteria for choosing a process for swapping out of the memory to the swap device?
Ans)
- The process's memory resident time,
- Priority of the process and
- The nice value.

18. What do you mean by nice value?
Ans) Nice value is the value that controls {increments or decrements} the priority of the process. This value that is returned by the nice () system call. The equation for using nice value is:
Priority = ("recent CPU usage"/constant) + (base- priority) + (nice value)
Only the administrator can supply the nice value. The nice () system call works for the running process only. Nice value of one process cannot affect the nice value of the other process.

19. What are conditions on which deadlock can occur while swapping the processes?
Ans)
- All processes in the main memory are asleep.
- All 'ready-to-run' processes are swapped out.
- There is no space in the swap device for the new incoming process that are swapped out of the main memory.

- There is no space in the main memory for the new incoming process.

20. What are conditions for a machine to support Demand Paging?
Ans)
- Memory architecture must based on Pages,
- The machine must support the 'restartable' instructions.

21. What is 'the principle of locality'?
Ans) It's the nature of the processes that they refer only to the small subset of the total data space of the process. i.e. the process frequently calls the same subroutines or executes the loop instructions.

22. What is the working set of a process?
Ans) The set of pages that are referred by the process in the last 'n', references, where 'n' is called the *window* of the working set of the process.

23. What is the window of the working set of a process?
Ans) The window of the working set of a process is the total number in which the process had referred the set of pages in the working set of the process.

24. What is called a page fault?
Ans) Page fault is referred to the situation when the process addresses a page in the working set of the process but the process fails to locate the page in the working set. And on a page fault the kernel updates the working set by reading the page from the secondary device.

25. What are data structures that are used for Demand Paging?
Ans) Kernel contains 4 data structures for Demand paging. They are,
- Page table entries,
- Disk block descriptors,
- Page frame data table (pfdata),
- Swap-use table.

26. What are the bits that support the demand paging?
Valid, Reference, Modify, Copy on write, Age. These bits are the part of the page table entry, which includes physical address of the page and protection bits.

Page Address	Age	Copy on write	Modify	Reference	Valid	Protection

27. How the Kernel handles the fork() system call in traditional Unix and in the System V Unix, while swapping?
Ans) Kernel in traditional Unix, makes the duplicate copy of the parent's address space and attaches it to the child's process, while swapping. Kernel in System V Unix, manipulates the region tables, page table, and pfdata table entries, by incrementing the reference count of the region table of shared regions.

28. Difference between the fork() and vfork() system call?

Ans)During the fork() system call the Kernel makes a copy of the parent process's address space and attaches it to the child process.

But the vfork() system call do not makes any copy of the parent's address space, so it is faster than the fork() system call. The child process as a result of the vfork() system call executes exec() system call. The child process from vfork() system call executes in the parent's address space (this can overwrite the parent's data and stack) which suspends the parent process until the child process exits.

29. What is BSS(Block Started by Symbol)?

Ans) A data representation at the machine level, that has initial values when a program starts and tells about how much space the kernel allocates for the un-initialized data. Kernel initializes it to zero at run-time.

30. What is Page-Stealer process?

Ans) This is the Kernel process that makes rooms for the incoming pages, by swapping the memory pages that are not the part of the working set of a process. Page-Stealer is created by the Kernel at the system initialization and invokes it throughout the lifetime of the system. Kernel locks a region when a process faults on a page in the region, so that page stealer cannot steal the page, which is being faulted in.

31. Name two paging states for a page in memory?

Ans) The two paging states are:

- The page is aging and is not yet eligible for swapping,
- The page is eligible for swapping but not yet eligible for reassignment to other virtual address space.

32. What are the phases of swapping a page from the memory?

Ans)

- Page stealer finds the page eligible for swapping and places the page number in the list of pages to be swapped.
- Kernel copies the page to a swap device when necessary and clears the *valid* bit in the page table entry, decrements the pfdata reference count, and places the pfdata table entry at the end of the free list if its reference count is 0.

33. What is page fault? Its types?

Ans)Page fault refers to the situation of not having a page in the main memory when any process references it.

There are two types of page fault :

- Validity fault,
- Protection fault.

34. In what way the Fault Handlers and the Interrupt handlers are different?

Ans) Fault handlers are also an interrupt handler with an exception that the interrupt handlers cannot sleep. Fault handlers sleep in the context of the process that caused the memory fault. The fault refers to the running process and no arbitrary processes are put to sleep.

35. What is validity fault?
Ans) If a process referring a page in the main memory whose valid bit is not set, it results in validity fault.

The valid bit is not set for those pages:

- that are outside the virtual address space of a process,
- that are the part of the virtual address space of the process but no physical address is assigned to it.

36. What does the swapping system do if it identifies the illegal page for swapping?
Ans) If the disk block descriptor does not contain any record of the faulted page, then this causes the attempted memory reference is invalid and the kernel sends a *"Segmentation violation"* signal to the offending process. This happens when the swapping system identifies any invalid memory reference.

37. What are states that the page can be in, after causing a page fault?
Ans)

- On a swap device and not in memory,
- On the free page list in the main memory,
- In an executable file,
- Marked "demand zero",
- Marked "demand fill".

38. *In what way the validity fault handler concludes?*
Ans)

- It sets the valid bit of the page by clearing the modify bit.
- It recalculates the process priority.

39. *At what mode the fault handler executes?*
Ans) At the Kernel Mode.

40. *What do you mean by the protection fault?*
Ans) Protection fault refers to the process accessing the pages, which do not have the access permission. A process also incur the protection fault when it attempts to write a page whose *copy on write* bit was set during the fork() system call.

41. *How the Kernel handles the copy on write bit of a page, when the bit is set?*
Ans) In situations like, where the copy on write bit of a page is set and that page is shared by more than one process, the Kernel allocates new page and copies the content to the new page and the other processes retain their references to the old page. After copying the Kernel updates the page table entry with the new page number. Then Kernel decrements the reference count of the old pfdata table entry.

In cases like, where the copy on write bit is set and no processes are sharing the page, the Kernel allows the physical page to be reused by the processes. By doing so, it clears the copy on write bit and disassociates the page from its disk copy (if one exists), because other process may share the disk copy. Then it removes the pfdata table entry from the page-queue as the new copy of the virtual page is not on the swap device. It decrements the swap-use count for the page and if count drops to 0, frees the swap space.

42. *For which kind of fault the page is checked first?*
Ans) The page is first checked for the validity fault, as soon as it is found that the page is invalid (valid bit is clear), the validity fault handler returns immediately, and the process incur the validity page fault. Kernel handles the validity fault and the process will incur the protection fault if any one is present.

43. *In what way the protection fault handler concludes?*
Ans) After finishing the execution of the fault handler, it sets the *modify* and *protection* bits and clears the *copy on write* bit. It recalculates the process-priority and checks for signals.

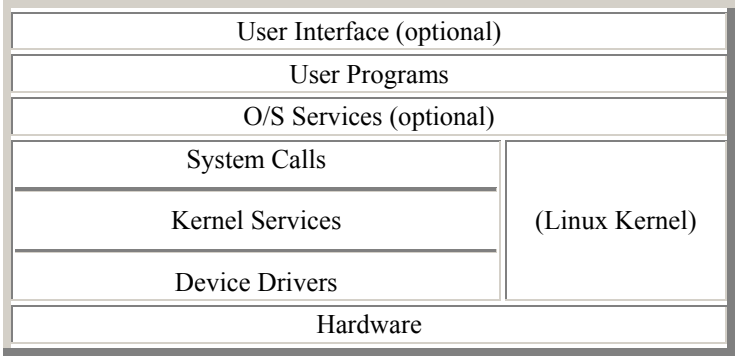
44. *How the Kernel handles both the page stealer and the fault handler?*
Ans) The page stealer and the fault handler thrash because of the shortage of the memory. If the sum of the working sets of all processes is greater that the physical memory then the fault handler will usually sleep because it cannot allocate pages for a process. This results in the reduction of the system throughput because Kernel spends too much time in overhead, rearranging the memory in the frantic pace.

45) Differece between the architecture of Linux and Windows?
Ans)
Linux architecture:
The Linux kernel is one layer in the architecture of the entire Linux system. It is conceptually composed of *five major subsystems:*

1. The process scheduler
2. The memory manager
3. The virtual file system
4. The network interface
5. The inter-process communication interface.

These subsystems interact with each other using function calls and shared data structures.
Another important extension to the Linux kernel is the addition of more supported hardware platforms.The architecture of the system supports portability by separating all hardware-specific code into distinct modules within each subsystem. In this way, a small group of developers can port the Linux kernel to new hardware by re-implementing only the machine-specific portions of the kernel.

Layers: FIGURE 1: Layers of an OS using the Linux Kernel



A strict Layered Architecture would only have calls from higher to the next lower layers, and then returns to the next higher level.

User Programs could institute pipes and filters, using pipe calls, then create descendant processes with the man2 fork System Call. The fork System Call would then use Kernel Services like the Memory Manager routines. Memory Manager routines could also make use of File System routines for swap files. This addresses again in Figure 2 below.

After the Kernel Services, the Device Drivers are the last interface between the Kernel Services and the Hardware.

The Linux Virtual File System allows sufficient flexibility that there does not have to be data storage to a disk at all. The Linux process file system allows computation of user I/O requests on demand, with memory or network interfaces instead of disks. Another example of an interrelationship is the proc System Call. The very early UNIX ps read the Kernel virtual memory directly and was a privileged process. Linux uses ps to simply parse and format the information from proc

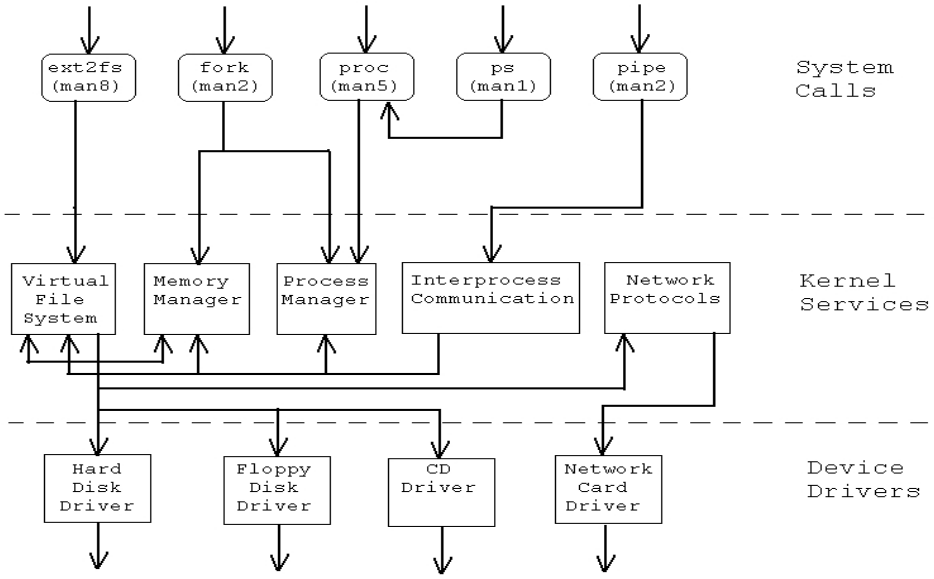


Figure 2. The Kernel Layers with a small sample of System Calls and a small sample of interfaces

Linux is not a Microkernel:

There are other patterns that would provide additional advantages but that were not used. Some aspects of the Microkernel pattern have been incorporated into modern OS. A very small amount of the Microkernel pattern is incorporated into Linux. The Linux Kernel can be customized to include only the packages that are wanted.

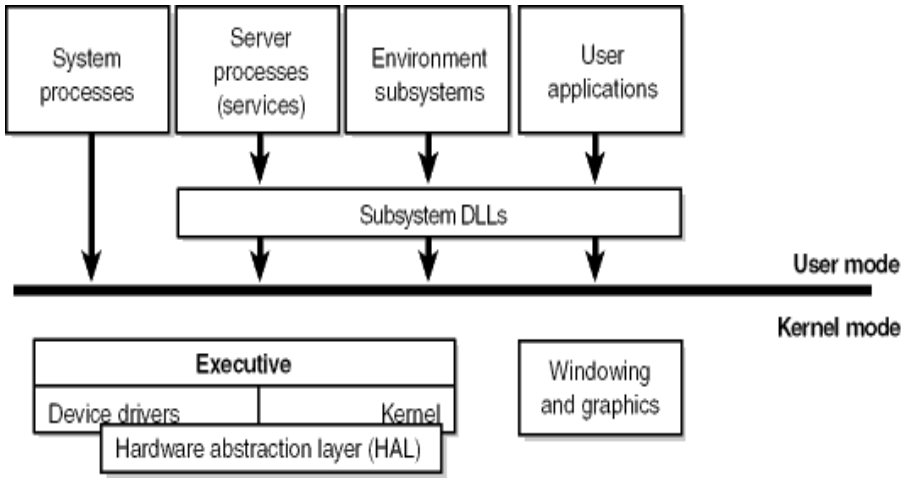
The Microkernel, in a Microkernel architecture, has been defined to include:

- Interprocess communications
- Low-level input and output capability,
- Limited scheduling and process management, and
- Basic memory management

A Microkernel would NOT include the device drivers and the file system management, yet the Linux kernel includes these.However there is a version of the Linux OS called MkLinux, that runs on top of a Mach Microkernel.
Obviously, Linux could have used a Microkernel Architecture, as evidenced by the fact that one exists. The fact that it does not, use a Microkernel, may or may not have been by intent.
Some of the bigger advantages of the Microkernel Architecture - portability, flexibility, extensibility and a small memory footprint - were not initial considerations.
The disadvantages of the Microkernel Architecture can be rather daunting - poor performance, especially when there is considerable message passing, and complexity of design and implementation

Windows NT architecture:

A simplified version of the Windows NT architecture is shown in figure



Notice the line dividing the user-mode and the kernel mode parts of the Windows NT operating system. The boxes above the line represent user-mode processes, and the components below the line are kernel-mode operating system services. User-mode threads execute in a protected process address space (although while they are executing in kernel mode, they have access to system space). Thus, system processes, server processes (services), the environment subsystems, and user applications each have their own private process address space

The four basic types of user processes are:

- Special *system support processes,* such as the logon process and the session manager, that are not Windows NT services (that is, not started by the service controller).
- *Server processes* that are Windows NT services, such as the Event Log and Schedule services. Many add-on server applications, such as Microsoft SQL Server and Microsoft Exchange Server, also include components that run as Windows NT services.
- *Environment subsystems,* which expose the native operating system services to user applications, thus providing an operating system *environment*, or personality. Windows NT ships with three environment subsystems: Win32, POSIX, and OS/2 1.2
- *User applications,* which can be one of five types: Win32, Windows 3.1, MS-DOS, POSIX, or OS/2 1.2.

Notice the "Subsystem DLLs" box below the "User applications" one. Under Windows NT, user applications do not call the native Windows NT operating system services directly; rather, they go through one or more *subsystem dynamic-link libraries* (*DLLs*). The role of the subsystem DLLs is to translate a documented function into the appropriate undocumented Windows NT system service calls. This translation might or might not involve sending a message to the environment subsystem process that is serving the user application.

The kernel mode of the operating system includes the following components

- The Windows NT *executive* contains the base operating system services, such as memory management, process and thread management, security, I/O, and interprocess communication
- The Windows NT *kernel* performs low-level operating system functions, such as thread scheduling, interrupt and exception dispatching, and multiprocessor synchronization. It also provides a set of routines and basic objects that the rest of the executive uses to implement higher-level constructs
- The *hardware abstraction layer* (*HAL*) is a layer of code that isolates the kernel, device drivers, and the rest of the Windows NT executive from platform-specific hardware differences.
- *Device drivers* include both file system and hardware device drivers that translate user I/O function calls into specific hardware device I/O requests
- The *windowing and graphics system* implements the graphical user interface (GUI) functions (better known as the Win32 USER and GDI functions), such as dealing with windows, controls, and drawing

User mode is the least-privileged mode supported by the NT . It does not had direct access to hardware and only restricted access to memory. For example, when programs such as Word and Lotus Notes execute in user mode, they are confined to sandboxes with well-defined restrictions. They don't have direct access to hardware devices, and they can't touch parts of memory that are not specifically assigned to them. Kernel mode is a privileged mode. Those parts of NT that execute in kernel mode, such as device drivers and subsystems such as the Virtual Memory Manager, have direct access to all hardware and memory.

WINDOWS NT & THE UNIX KERNEL

The OS architecture of most versions of UNIX is similar to that of Windows NT. The figure1 and figure2 below shows the UNIX and Windows NT architectures, respectively.

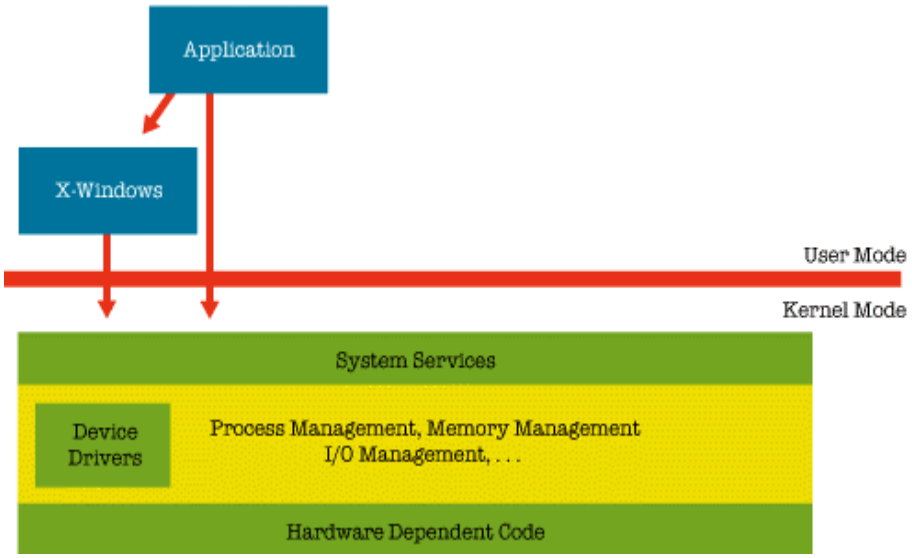


FIGURE 1

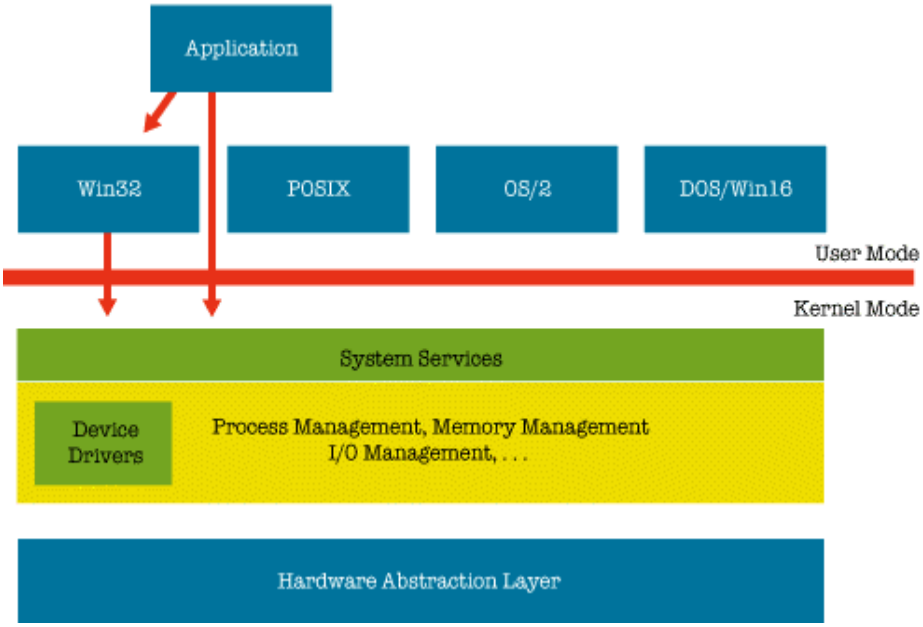
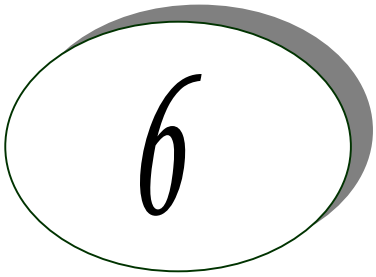


FIGURE 2

Both UNIX and Windows NT OS architectures have two modes of operation: user mode and kernel mode. Familiar applications such as word processors and database programs execute in user mode. User mode is *nonprivileged*, which means that the system restricts programs operating in user mode from directly accessing hardware or resources belonging to other programs. Most of the OS code executes in kernel mode. Kernel mode is *privileged,* which means that code running in kernel mode can access hardware and resources belonging to any application, with few limitations.

A major difference between UNIX's and Windows NT's architecture is that UNIX does not incorporate its windowing system--the subsystem that manages GUI resources for applications--into kernel mode, as does Windows NT. Instead, the UNIX windowing system is an add-on user-mode application that its developers wrote using publicly defined UNIX APIs; consequently, third-party products can replace UNIX's windowing system. However, the majority of the UNIX community has adopted MIT's X-Windows as a *de facto*, if not official, graphical interface standard. In earlier version of Windows NT , the windowing system was a user-mode implementation, but it was found that the performance of graphics-intensive applications improved when the windowing system operated in kernel mode. In later version of Windows NT the windowing system is also part of kernel and is implemented in kernel mode.

Another difference between the Windows NT and UNIX OS architectures is that UNIX applications can call kernel functions, or *system calls,* directly. In Windows NT, applications call APIs that the OS environment to which they are coded (DOS, Windows 3.x, OS/2, POSIX) exports. A kernel system-call interface provides APIs for managing processes, memory, and files. The Windows NT system-call interface, called the Native API, is hidden from programmers and largely undocumented. The API that UNIX applications write to is the UNIX system-call interface, whereas the API that the majority of Windows NT applications write to is the Win32 API, which translates many Win32 APIs to Native APIs.



RDBMS Concepts

1. What is database?

Ans) A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

2. What is DBMS and RDBMS?

Ans) DBMS is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of *defining*, *constructing* and *manipulating* the database for various applications.

Relational Database Management System(RDBMS) a type of database management system (DBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways.

An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table.

Almost all full-scale database systems are RDBMS's. Small database systems, however, use other designs that provide less flexibility in posing queries.

3. What is a Database system?

Ans) The database and DBMS software together is called as Database system.

4. Advantages of DBMS?

Ans)

- Redundancy is controlled.
- Unauthorised access is restricted.
- Providing multiple user interfaces.
- Enforcing integrity constraints.
- Providing backup and recovery.

5. Disadvantage in File Processing System?

Ans)

- Data redundancy & inconsistency.
- Difficult in accessing data.
- Data isolation.
- Data integrity.
- Concurrent access is not possible.
- Security Problems.

6. Describe the three levels of data abstraction?

Ans) The are three levels of abstraction:

- *Physical level:* The lowest level of abstraction describes how data are stored.
- *Logical level:* The next higher level of abstraction, describes what data are stored in database and what relationship among those data.
- *View level:* The highest level of abstraction describes only part of entire database.

7. Define the "integrity rules"?

Ans) There are two Integrity rules.

- *Entity Integrity:* States that "Primary key cannot have NULL value"
- *Referential Integrity:* States that "Foreign Key can be either a NULL value or should be Primary Key value of other relation".

8. What is extension and intension?

Ans)

Extension :-It is the number of tuples present in a table at any instance. This is time dependent.

Intension :- It is a constant value that gives the name, structure of table and the constraints laid on it.

9. What is System R? What are its two major subsystems?

Ans) System R was designed and developed over a period of 1974-79 at IBM San Jose Research Center. It is a prototype and its purpose was to demonstrate that it is possible to build a Relational System that can be used in a real life environment to solve real life problems, with performance at least comparable to that of existing system.

Its two subsystems are

- Research Storage
- System Relational Data System.

10. How is the data structure of System R different from the relational structure?

Ans) Unlike Relational systems in System R

- Domains are not supported
- Enforcement of candidate key uniqueness is optional
- Enforcement of entity integrity is optional
- Referential integrity is not enforced

11. What is Data Independence?
Ans) Data independence means that "the application is independent of the storage structure and access strategy of data". In other words, The ability to modify the schema definition in one level should not affect the schema definition in the next higher level.
Two types of Data Independence:
- Physical Data Independence: Modification in physical level should not affect the logical level.
- Logical Data Independence: Modification in logical level should affect the view level.

NOTE: Logical Data Independence is more difficult to achieve

12. What is a view? How it is related to data independence?
Ans) A view may be thought of as a virtual table, that is, a table that does not really exist in its own right but is instead derived from one or more underlying base table. In other words, there is no stored file that direct represents the view instead a definition of view is stored in data dictionary.
Growth and restructuring of base tables is not reflected in views. Thus the view can insulate users from the effects of restructuring and growth in the database. Hence accounts for logical data independence.

13. What is Data Model?
Ans) A collection of conceptual tools for describing data, data relationships, data semantics and constraints.

14. What is E-R model?
Ans) This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

15. What is Object Oriented model?
Ans) This model is based on collection of objects. An object contains values stored in instance variables with in the object. An object also contains bodies of code that operate on the object.These bodies of code are called methods.Objects that contain same types of values and the same methods are grouped together into classes.

16. What is an Entity?
Ans) It is a 'thing' in the real world with an independent existence.

17. What is an Entity type?
Ans) It is a collection (set) of entities that have same attributes.

18. What is an Entity set?
Ans) It is a collection of all entities of particular entity type in the database.

19. What is an Extension of entity type?
Ans) The collections of entities of a particular entity type are grouped together into an entity set.

20. What is Weak Entity set?

Ans) An entity set may not have sufficient attributes to form a primary key, and its primary key compromises of its partial key and primary key of its parent entity, then it is said to be Weak Entity set.

21. What is an attribute?
Ans) It is a particular property, which describes the entity.

22. What is a Relation Schema and a Relation?
Ans) A relation Schema denoted by R(A1, A2, …, An) is made up of the relation name R and the list of attributes Ai that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples (t1, t2, t3, ..., tn). Each tuple is an ordered list of n-values t=(v1,v2, ..., vn).

23. What is degree of a Relation?
Ans) It is the number of attribute of its relation schema.

24. What is Relationship?
Ans) It is an association among two or more entities.

25. What is Relationship set?
Ans) The collection (or set) of similar relationships.

26. What is Relationship type?
Ans) Relationship type defines a set of associations or a relationship set among a given set of entity types.

27. What is degree of Relationship type?
Ans) It is the number of entity type participating.

25. What is DDL (Data Definition Language)?
Ans) A data base schema is specifies by a set of definitions expressed by a special language called DDL.

26. What is VDL (View Definition Language)?
Ans) It specifies user views and their mappings to the conceptual schema.

27. What is SDL (Storage Definition Language)?
Ans) This language is to specify the internal schema. This language may specify the mapping between two schemas.

28. What is Data Storage - Definition Language ?
Ans) The storage structures and access methods used by database system are specified by a set of definition in a special type of DDL called data storage-definition language.

29. What is DML (Data Manipulation Language)?
Ans) This language that enable user to access or manipulate data as organised by appropriate data model.
- *Procedural DML or Low level:* DML requires a user to specify what data are needed and how to get those data.

- *Non-Procedural DML or High level:* DML requires a user to specify what data are needed without specifying how to get those data.

30. *What is Triggers?*
Ans) A *trigger* is a statement that is executed automatically by the system as a side effect of a modification to the database.
To design a trigger mechanism, we must:
- Specify the conditions under which the trigger is to be executed.
- Specify the actions to be taken when the trigger executes

31. What is DML Compiler?
Ans) It translates DML statements in a query language into low-level instruction that the query evaluation engine can understand.

32. What is Query evaluation engine?
Ans) It executes low-level instruction generated by compiler.

33. What is DDL Interpreter?
Ans) It interprets DDL statements and record them in tables containing metadata.

34. What is Record-at-a-time?
Ans) The Low level or Procedural DML can specify and retrieve each record from a set of records. This retrieve of a record is said to be Record-at-a-time.

35. What is Set-at-a-time or Set-oriented?
Ans) The High level or Non-procedural DML can specify and retrieve many records in a single DML statement. This retrieve of a record is said to be Set-at-a-time or Set-oriented.

36. What is Relational Algebra?
Ans) It is procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation.

37. What is Relational Calculus?
Ans) It is an applied predicate calculus specifically tailored for relational databases proposed by E.F. Codd. E.g. of languages based on it are DSL ALPHA, QUEL.

38. How does Tuple-oriented relational calculus differ from domain-oriented relational calculus
Ans) The tuple-oriented calculus uses a tuple variables i.e., variable whose only permitted values are tuples of that relation. E.g. QUEL
The domain-oriented calculus has domain variables i.e., variables that range over the underlying domains instead of over relation. E.g. ILL, DEDUCE.

39. What is normalization?
Ans) It is a process of analysing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties
- Minimizing redundancy
- Minimizing insertion, deletion and update anomalies.

40. What is Functional Dependency?
Ans) A Functional dependency is denoted by X⟶Y between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuple that can form a relation state r of R. The constraint is for any two tuples t1 and t2 in r if t1[X] = t2[X] then they have t1[Y] = t2[Y]. This means the value of X component of a tuple uniquely determines the value of component Y.

41. When is a functional dependency F said to be minimal?
Ans)
- Every dependency in F has a single attribute for its right hand side.
- We cannot replace any dependency X⟶A in F with a dependency Y⟶A where Y is a proper subset of X and still have a set of dependency that is equivalent to F.
- We cannot remove any dependency from F and still have set of dependency that is equivalent to F.

42. What is Multivalued dependency?
Ans) Multivalued dependency denoted by X⟶⟶Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation r of R: if two tuples t1 and t2 exist in r such that t1[X] = t2[X] then t3 and t4 should also exist in r with the following properties
- t3[X] = t4[X] = t1[X] = t2[X]
- t3[Y] = t1[Y] and t4[Y] = t2[Y]
- t3[Z] = t2[Z] and t4[Z] = t1[Z]

where [Z = (R-(X U Y))]

43. What is Lossless join property?
Ans) It guarantees that the spurious tuple generation does not occur with respect to relation schemas after decomposition.

44. What is 1 NF (Normal Form)?
Ans) The domain of attribute must include only atomic (simple, indivisible) values.

45. What is Fully Functional dependency?
Ans) It is based on concept of full functional dependency. A functional dependency X Y is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

46. What is 2NF?
Ans) A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

47. What is 3NF?
Ans) A relation schema R is in 3NF if it is in 2NF and for every FD X⟶A either of the following is true
- X is a Super-key of R.
- A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

48. What is BCNF (Boyce-Codd Normal Form)?
Ans) A relation schema R is in BCNF if it is in 3NF and satisfies an additional constraint that for every FD X A, X must be a candidate key.

49. What is 4NF?
Ans) A relation schema R is said to be in 4NF if for every Multivalued dependency X ⟶ Y that holds over R, one of following is true
- X is subset or equal to (or) XY = R.
- X is a super key.

50. What is 5NF?
Ans) A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ..., Rn} that holds R, one the following is true
- Ri = R for some i.
- The join dependency is implied by the set of FD, over R in which the left side is key of R.

51. What is Domain-Key Normal Form?
Ans) A relation is said to be in DKNF if all constraints and dependencies that should hold on the the constraint can be enforced by simply enforcing the domain constraint and key constraint on the relation.

52. What are partial, alternate,, artificial, compound and natural key?
Ans)
Partial Key:
It is a set of attributes that can uniquely identify weak entities and that are related to same owner entity. It is sometime called as Discriminator.
Alternate Key:
All Candidate Keys excluding the Primary Key are known as Alternate Keys.
Artificial Key:
If no obvious key, either stand alone or compound is available, then the last resort is to simply create a key, by assigning a unique number to each record or occurrence. Then this is known as developing an artificial key.
Compound Key:
If no single data element uniquely identifies occurrences within a construct, then combining multiple elements to create a unique identifier for the construct is known as creating a compound key.
Natural Key:
When one of the data elements stored within a construct is utilized as the primary key, then it is called the natural key.

53. What is indexing and what are the different kinds of indexing?
Ans) Indexing is a technique for determining how quickly specific data can be found.
Types:
- Binary search style indexing
- B-Tree indexing
- Inverted list indexing
- Memory resident table
- Table indexing

54. What is system catalog or catalog relation? How is better known as?

Ans) A RDBMS maintains a description of all the data that it contains, information about every relation and index that it contains. This information is stored in a collection of relations maintained by the system called metadata. It is also called data dictionary.

55. What is meant by query optimization?

Ans) The phase that identifies an efficient execution plan for evaluating a query that has the least estimated cost is referred to as query optimization.

56. What is join dependency and inclusion dependency?

Ans) Join Dependency:

A Join dependency is generalization of Multivalued dependency.A JD {R1, R2, ..., Rn} is said to hold over a relation R if R1, R2, R3, ..., Rn is a lossless-join decomposition of R . There is no set of sound and complete inference rules for JD.

Inclusion Dependency:

An Inclusion Dependency is a statement of the form that some columns of a relation are contained in other columns. A foreign key constraint is an example of inclusion dependency.

57. What is durability in DBMS?

Ans) Once the DBMS informs the user that a transaction has successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability.

58. What do you mean by atomicity and aggregation?

Ans)

Atomicity:

Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions. DBMS ensures this by undoing the actions of incomplete transactions.

Aggregation:

A concept which is used to model a relationship between a collection of entities and relationships. It is used when we need to express a relationship among relationships.

59. What is a Phantom Deadlock?

Ans) In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts.

60. What is a checkpoint and When does it occur?

Ans) A Checkpoint is like a snapshot of the DBMS state. By taking checkpoints, the DBMS can reduce the amount of work to be done during restart in the event of subsequent crashes.

61. What are the different phases of transaction?
Ans) Different phases are
- Analysis phase
- Redo Phase
- Undo phase

62. What do you mean by flat file database?
Ans) It is a database in which there are no programs or user access languages. It has no cross-file capabilities but is user-friendly and provides user-interface management.

63. What is "transparent DBMS"?
Ans) It is one, which keeps its Physical Structure hidden from user.

64. Brief theory of Network, Hierarchical schemas and their properties?
Ans) Network schema uses a graph data structure to organize records example for such a database management system is CTCG while a hierarchical schema uses a tree data structure example for such a system is IMS.

65. What is a query?
*Ans)*A query with respect to DBMS relates to user commands that are used to interact with a data base. The query language can be classified into data definition language and data manipulation language.

66. What do you mean by Correlated subquery?
Ans) Subqueries, or nested queries, are used to bring back a set of rows to be used by the parent query. Depending on how the subquery is written, it can be executed once for the parent query or it can be executed once for each row returned by the parent query. If the subquery is executed for each row of the parent, this is called a *correlated subquery*.
A correlated subquery can be easily identified if it contains any references to the parent subquery columns in its WHERE clause. Columns from the subquery cannot be referenced anywhere else in the parent query. The following example demonstrates a non-correlated subquery.
E.g. Select * From CUST Where '10/03/1990' IN (Select ODATE From ORDER Where CUST.CNUM = ORDER.CNUM)

67. What are the primitive operations common to all record management systems?
Ans) Addition, deletion and modification.

68. Name the buffer in which all the commands that are typed in are stored
Ans)'Edit' Buffer

69. What are the unary operations in Relational Algebra?
*Ans)*PROJECTION and SELECTION.

70. Are the resulting relations of PRODUCT and JOIN operation the same?
Ans) No.
PRODUCT: Concatenation of every row in one relation with every row in another.
JOIN: Concatenation of rows from one relation and related rows from another.

71. What is RDBMS KERNEL?

Ans) Two important pieces of RDBMS architecture are the kernel, which is the software, and the data dictionary, which consists of the system-level data structures used by the kernel to manage the database

You might think of an RDBMS as an operating system (or set of subsystems), designed specifically for controlling data access; its primary functions are storing, retrieving, and securing data. An RDBMS maintains its own list of authorized users and their associated privileges; manages memory caches and paging; controls locking for concurrent resource usage; dispatches and schedules user requests; and manages space usage within its table-space structures

.

72. Name the sub-systems of a RDBMS

*Ans)*I/O, Security, Language Processing, Process Control, Storage Management, Logging and Recovery, Distribution Control, Transaction Control, Memory Management, Lock Management

73. Which part of the RDBMS takes care of the data dictionary? How

Ans) Data dictionary is a set of tables and database objects that is stored in a special area of the database and maintained exclusively by the kernel.

74. What is the job of the information stored in data-dictionary?

Ans) The information in the data dictionary validates the existence of the objects, provides access to them, and maps the actual physical storage location.

75. Not only RDBMS takes care of locating data it also _____

Ans) determines an optimal access path to store or retrieve the data

76. How do you communicate with an RDBMS?

Ans) You communicate with an RDBMS using Structured Query Language (SQL)

77. Define SQL and state the differences between SQL and other conventional programming Languages

Ans) SQL is a nonprocedural language that is designed specifically for data access operations on normalized relational database structures. The primary difference between SQL and other conventional programming languages is that SQL statements specify what data operations should be performed rather than how to perform them.

78. Name the three major set of files on disk that compose a database in Oracle

Ans) There are three major sets of files on disk that compose a database. All the files are binary. These are

- Database files
- Control files
- Redo logs

The most important of these are the database files where the actual data resides. The control files and the redo logs support the functioning of the architecture itself.

All three sets of files must be present, open, and available to Oracle for any data on the database to be useable. Without these files, you cannot access the database, and

the database administrator might have to recover some or all of the database using a backup, if there is one.

79. What is an Oracle Instance?
Ans) The Oracle system processes, also known as Oracle background processes, provide functions for the user processes—functions that would otherwise be done by the user processes themselves
Oracle database-wide system memory is known as the SGA, the *system global area* or *shared global area*. The data and control structures in the SGA are shareable, and all the Oracle background processes and user processes can use them.
The combination of the SGA and the Oracle background processes is known as an *Oracle instance*

80. What are the four Oracle system processes that must always be up and running for the database to be useable
Ans) The four Oracle system processes that must always be up and running for the database to be useable include *DBWR* (Database Writer), *LGWR* (Log Writer), *SMON* (System Monitor), and *PMON* (Process Monitor).

81. What are database files, control files and log files. How many of these files should a database have at least? Why?
Database Files
The database files hold the actual data and are typically the largest in size. Depending on their sizes, the tables (and other objects) for all the user accounts can go in one database file—but that's not an ideal situation because it does not make the database structure very flexible for controlling access to storage for different users, putting the database on different disk drives, or backing up and restoring just part of the database. You must have at least one database file but usually, more than one files are used. In terms of accessing and using the data in the tables and other objects, the number (or location) of the files is immaterial.
The database files are fixed in size and never grow bigger than the size at which they were created
Control Files
The control files and redo logs support the rest of the architecture. Any database must have at least one control file, although you typically have more than one to guard against loss. The control file records the name of the database, the date and time it was created, the location of the database and redo logs, and the synchronization information to ensure that all three sets of files are always in step. Every time you add a new database or redo log file to the database, the information is recorded in the control files.
Redo Logs
Any database must have at least two redo logs. These are the journals for the database; the redo logs record all changes to the user objects or system objects. If any type of failure occurs, the changes recorded in the redo logs can be used to bring the database to a consistent state without losing any committed transactions. In the case of non-data loss failure, Oracle can apply the information in the redo logs automatically without intervention from the DBA.
The redo log files are fixed in size and never grow dynamically from the size at which they were created.

82. What is ROWID?
Ans) The ROWID is a unique database-wide physical address for every row on every table. Once assigned (when the row is first inserted into the database), it never changes until the row is deleted or the table is dropped.
The ROWID consists of the following three components, the combination of which uniquely identifies the physical storage location of the row.

- Oracle database file number, which contains the block with the rows
- Oracle block address, which contains the row
- The row within the block (because each block can hold many rows)

The ROWID is used internally in indexes as a quick means of retrieving rows with a particular key value. Application developers also use it in SQL statements as a quick way to access a row once they know the ROWID

83. What is Oracle Block? Can two Oracle Blocks have the same address?
Ans) Oracle "formats" the database files into a number of Oracle blocks when they are first created—making it easier for the RDBMS software to manage the files and easier to read data into the memory areas.
The block size should be a multiple of the operating system block size. Regardless of the block size, the entire block is not available for holding data; Oracle takes up some space to manage the contents of the block. This block header has a minimum size, but it can grow.
These Oracle blocks are the smallest unit of storage. Increasing the Oracle block size can improve performance, but it should be done only when the database is first created.
Each Oracle block is numbered sequentially for each database file starting at 1. Two blocks can have the same block address if they are in different database files.

84. What is database Trigger?
Ans) A database trigger is a PL/SQL block that can defined to automatically execute for insert, update, and delete statements against a table. The trigger can e defined to execute once for the entire statement or once for every row that is inserted, updated, or deleted. For any one table, there are twelve events for which you can define database triggers. A database trigger can call database procedures that are also written in PL/SQL.

85. Name two utilities that Oracle provides, which are use for backup and recovery.
Ans) Along with the RDBMS software, Oracle provides two utilities that you can use to back up and restore the database. These utilities are *Export* and *Import*.
The *Export utility* dumps the definitions and data for the specified part of the database to an operating system binary file. The *Import utility* reads the file produced by an export, recreates the definitions of objects, and inserts the data
If Export and Import are used as a means of backing up and recovering the database, all the changes made to the database cannot be recovered since the export was performed. The best you can do is recover the database to the time when the export was last performed.

86. What are stored-procedures? And what are the advantages of using them.

Ans) Stored procedures are database objects that perform a user defined operation. A stored procedure can have a set of compound SQL statements. A stored procedure executes the SQL commands and returns the result to the client. Stored procedures are used to reduce network traffic.

87. How are exceptions handled in PL/SQL? Give some of the internal exceptions' name

Ans) PL/SQL exception handling is a mechanism for dealing with run-time errors encountered during procedure execution. Use of this mechanism enables execution to continue if the error is not severe enough to cause procedure termination.

The exception handler must be defined within a subprogram specification. Errors cause the program to raise an exception with a transfer of control to the exception-handler block. After the exception handler executes, control returns to the block in which the handler was defined. If there are no more executable statements in the block, control returns to the caller.

User-Defined Exceptions

PL/SQL enables the user to define exception handlers in the declarations area of subprogram specifications. User accomplishes this by naming an exception as in the following example:

```
ot_failure  EXCEPTION;
```

In this case, the exception name is ot_failure. Code associated with this handler is written in the EXCEPTION specification area as follows:

```
EXCEPTION
    when OT_FAILURE then
      out_status_code := g_out_status_code;
      out_msg         := g_out_msg;
```

The following is an example of a subprogram exception:

```
EXCEPTION
    when NO_DATA_FOUND then
      g_out_status_code := 'FAIL';
      RAISE ot_failure;
```

Within this exception is the RAISE statement that transfers control back to the ot_failure exception handler. This technique of raising the exception is used to invoke all user-defined exceptions.

System-Defined Exceptions

Exceptions internal to PL/SQL are raised automatically upon error. NO_DATA_FOUND is a system-defined exception. Table below gives a complete list of internal exceptions.

PL/SQL internal exceptions.

Exception Name	*Oracle Error*
CURSOR_ALREADY_OPEN	ORA-06511
DUP_VAL_ON_INDEX	ORA-00001
INVALID_CURSOR	ORA-01001
INVALID_NUMBER	ORA-01722
LOGIN_DENIED	ORA-01017
NO_DATA_FOUND	ORA-01403
NOT_LOGGED_ON	ORA-01012

PROGRAM_ERROR	ORA-06501
STORAGE_ERROR	ORA-06500
TIMEOUT_ON_RESOURCE	ORA-00051
TOO_MANY_ROWS	ORA-01422
TRANSACTION_BACKED_OUT	ORA-00061
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476

In addition to this list of exceptions, there is a catch-all exception named *OTHERS* that traps all errors for which specific error handling has not been established.

88. Does PL/SQL support "overloading"? Explain?
Ans) The concept of *overloading* in PL/SQL relates to the idea that you can define procedures and functions with the same name. PL/SQL does not look only at the referenced name, however, to resolve a procedure or function call. The count and data types of formal parameters are also considered.
PL/SQL also attempts to resolve any procedure or function calls in locally defined packages before looking at globally defined packages or internal functions. To further ensure calling the proper procedure, you can use the dot notation. Prefacing a procedure or function name with the package name fully qualifies any procedure or function reference.

89. Tables derived from the ERD
a) Are totally unnormalised
b) Are always in 1NF
c) Can be further denormalised
d) May have multi-valued attributes

(b) Are always in 1NF

90. Spurious tuples may occur due to
i. Bad normalization
ii. Theta joins
iii. Updating tables from join
a) i & ii b) ii & iii
c) i & iii d) ii & iii

(a) i & iii because theta joins are joins made on keys that are not primary keys.

91. A B C is a set of attributes. The functional dependency is as follows
AB -> B
AC -> C
C -> B
a) is in 1NF
b) is in 2NF
c) is in 3NF
d) is in BCNF

(a) is in 1NF since $(AC)^+ = \{ A, B, C\}$ hence AC is the primary key. Since C $\rightarrow$ B is a FD given, where neither C is a Key nor B is a prime attribute, this it is not in 3NF.

Further B is not functionally dependent on key AC thus it is not in 2NF. Thus the given FDs is in 1NF.

92. In mapping of ERD to DFD
a) entities in ERD should correspond to an existing entity/store in DFD
b) entity in DFD is converted to attributes of an entity in ERD
c) relations in ERD has 1 to 1 correspondence to processes in DFD
d) relationships in ERD has 1 to 1 correspondence to flows in DFD

(a) entities in ERD should correspond to an existing entity/store in DFD

93. A dominant entity is the entity
a) on the N side in a 1 : N relationship
b) on the 1 side in a 1 : N relationship
c) on either side in a 1 : 1 relationship
d) nothing to do with 1 : 1 or 1 : N relationship

(b) on the 1 side in a 1 : N relationship

94. Select 'NORTH', CUSTOMER From CUST_DTLS Where REGION = 'N' Order By CUSTOMER Union Select 'EAST', CUSTOMER From CUST_DTLS Where REGION = 'E' Order By CUSTOMER
The above is
a) Not an error
b) Error - the string in single quotes 'NORTH' and 'SOUTH'
c) Error - the string should be in double quotes
d) Error - ORDER BY clause

(d) Error - the ORDER BY clause. Since ORDER BY clause cannot be used in UNIONS

95. What is Storage Manager?
Ans) It is a program module that provides the interface between the low-level data stored in database, application programs and queries submitted to the system.

96. What is Buffer Manager?
Ans) It is a program module, which is responsible for fetching data from disk storage into main memory and deciding what data to be cache in memory.

97. What is Transaction Manager?
Ans) It is a program module, which ensures that database, remains in a consistent state despite system failures and concurrent transaction execution proceeds without conflicting.

98. What is File Manager?
Ans) It is a program module, which manages the allocation of space on disk storage and data structure used to represent information stored on a disk.

99. What is Authorization and Integrity manager?
Ans) It is the program module, which tests for the satisfaction of integrity constraint and checks the authority of user to access data.

100. What are stand-alone procedures?
Ans) Procedures that are not part of a package are known as stand-alone because they independently defined. A good example of a stand-alone procedure is one written in a SQL*Forms application. These types of procedures are not available for reference from other Oracle tools. Another limitation of stand-alone procedures is that they are compiled at run time, which slows execution.

101. What are cursors give different types of cursors.
Ans) PL/SQL uses cursors for all database information accesses statements. The language supports the use two types of cursors

- *Implicit*
- *Explicit*

102. What is cold backup and hot backup (in case of Oracle)?
Ans)

- *Cold Backup:*

It is copying the three sets of files (database files, redo logs, and control file) when the instance is shut down. This is a straight file copy, usually from the disk directly to tape. You must shut down the instance to guarantee a consistent copy.
If a cold backup is performed, the only option available in the event of data file loss is restoring all the files from the latest backup. All work performed on the database since the last backup is lost.

- *Hot Backup:*

Some sites (such as worldwide airline reservations systems) cannot shut down the database while making a backup copy of the files. The cold backup is not an available option.
So different means of backing up database must be used — the hot backup. Issue a SQL command to indicate to Oracle, on a tablespace-by-tablespace basis, that the files of the tablespace are to backed up. The users can continue to make full use of the files, including making changes to the data. Once the user has indicated that he/she wants to back up the tablespace files, he/she can use the operating system to copy those files to the desired backup destination.
The database must be running in ARCHIVELOG mode for the hot backup option.
If a data loss failure does occur, the lost database files can be restored using the hot backup and the online and offline redo logs created since the backup was done. The database is restored to the most consistent state without any loss of committed transactions.

103. What are Armstrong rules? How do we say that they are complete and/or sound
The well-known inference rules for FDs

- Reflexive rule :

  If Y is subset or equal to X then $X \rightarrow Y$.

- Augmentation rule:

  If $X \rightarrow Y$ then $XZ \rightarrow YZ$.

- Transitive rule:

  If $\{X \rightarrow Y, Y \rightarrow Z\}$ then $X \rightarrow Z$.

- Decomposition rule :

  If $X \longrightarrow YZ$ then $X \longrightarrow Y$.

- Union or Additive rule:

  If $\{X \longrightarrow Y, X \longrightarrow Z\}$ then $X \longrightarrow YZ$.

- Pseudo Transitive rule :

  If $\{X \longrightarrow Y, WY \longrightarrow Z\}$ then $WX \longrightarrow Z$.

Of these the first three are known as Amstrong Rules. They are sound because it is enough if a set of FDs satisfy these three. They are called complete because using these three rules we can generate the rest all inference rules.

104.How can you find the minimal key of relational schema?

Ans) Minimal key is one which can identify each tuple of the given relation schema uniquely. For finding the minimal key it is required to find the closure that is the set of all attributes that are dependent on any given set of attributes under the given set of functional dependency.

Algo. I Determining X^+, closure for X, given set of FDs F

1. Set $X^+ = X$
2. Set Old $X^+ = X^+$
3. For each FD Y Z in F and if Y belongs to X^+ then add Z to X^+
4. Repeat steps 2 and 3 until Old $X^+ = X^+$

Algo.II Determining minimal K for relation schema R, given set of FDs F

1. Set K to R that is make K a set of all attributes in R
2. For each attribute A in K
   a) Compute $(K - A)^+$ with respect to F
   b) If $(K - A)^+ = R$ then set $K = (K - A)^+$

105.What do you understand by dependency preservation?

Ans) Given a relation R and a set of FDs F, dependency preservation states that the closure of the union of the projection of F on each decomposed relation Ri is equal to the closure of F. i.e.,

$((\Pi_{R1}(F)) \cup \ldots \cup (\Pi_{Rn}(F)))^+ = F^+$

if decomposition is not dependency preserving, then some dependency is lost in the decomposition.

106.What is meant by Proactive, Retroactive and Simultaneous Update.

107.

Ans)

Proactive Update:

The updates that are applied to database before it becomes effective in real world .

Retroactive Update:

The updates that are applied to database after it becomes effective in real world .

Simulatneous Update:

The updates that are applied to database at the same time when it becomes effective in real world .

108.What are the different types of JOIN operations?

Ans)

Equi Join: This is the most common type of join which involves only equality comparisions. The disadvantage in this type of join is that there

Importany Points

A database is a large, integrated collection of data.
A database management system is a software for efficient storage and retrieval of data. For simple and small databases files are ok but when once the database become large managing data and retrieving data becomes a complex issue.

Files vs. DBMS

- Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)
- Special code for different queries
- Must protect data from inconsistency due to multiple concurrent users
- Crash recovery
- Security and access control

Why use Databases:

- Data independence and efficient storage.
- Reduced application development time.
- Data integrity and security
- Concurrent access and crash recovery
- Uniform data administration.

Data models:

A data model is a collection of concepts for describing data
Ex: er diagrams, relational model.
A schema is a description of particular collection of data using the given data model.

The relational model of data is much widely used model today. In relational model Data is stored in relations (basically tables with rows and columns)

Levels of abstraction:

Conceptual Schema describes the stored data in terms of the data model of the DBMS.
Physical Schema specifies additional storage details. Essentially physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices.
External Schema which usually are also in terms of the data model of the DBMS, allow data access to be customized at the level of individual users or groups of users.

Data Independence:
Applications insulated from how data is structured and stored.
_ *Logical data independence*: Protection from changes in *logical* structure of data.
_ *Physical data independence*: Protection from changes in *physical* structure of data.
- *One of the most important benefits of using a DBMS!*

A transaction is any one execution of a user program in a DBMS. This is the basic unit of change as seen by the DBMS. Partial transactions are not allowed and group transactions is equivalent to some serial execution of all transactions.

Concurrency Control:
Concurrent execution of user programs is essential for good DBMS performance.
_ Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently.
_ Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
_ DBMS ensures such problems don't arise: users
can pretend they are using a single-user system.

Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes).
- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
- Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
- Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Atomicity:
DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.

The DBMS maintains a log of all writes to the database. A crucial property of the log is that each write action must be recorded in the log (on disk) before the corresponding change is reflected in the database itself. This property is called write ahead log(WAL).

Bringing the database to a consistent state after a sys crash can be a slow process, since the dbms must ensure that the effects of incomplete transactions are undone. The time required to recover from a crash can be reproduced by periodically forcing some info to disk; this periodic operation is called a check point.

Architecture of DBMS:

When a user issues a query the parsed query is presented to a query optimizer which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blue print for evaluating a query and is usually represented as a tree of relational operators.

The DBMS supports concurrency and crash recovery by carefully scheduling user requests and maintaining a log of all changes to the database.
DBMS components associated with concurrency control and recovery include
The transaction manager ensures transactions request and release locks according to a suitable locking protocol and schedules the execution of transactions.
The lock manager keeps track of requests for locks and grants locks on database objects when they become available.
The recovery manager is responsible for maintaining a log, and restoring the sys to a consistent state after a crash.

E-R diagrams

The entity relationship data model allows us to describe the data involved in real world in terms of objects and their relationships and is widely used in initial database design.

An entity is a real-world object distinguishable from other objects. Entity is described using attributes. An Entity set is a collection of similar entities.
All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)

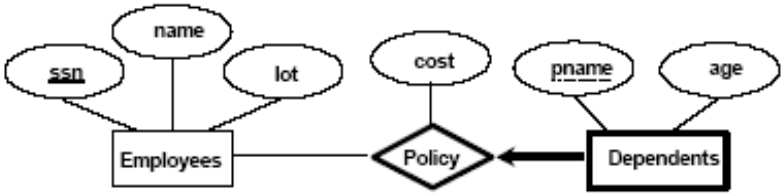
- Each entity set has a *key*.
- Each attribute has a *domain*.(set of permitted values)

A key is a minimal set of attributes whose values uniquely identify an entity in the set.
A relationship is an association among two or more entities.
Relationship Set: Collection of similar relationships.
A relationship can also have descriptive attributes.

Arrow directions pending:

Participation constraints:
The participation of the entity set Departments in the relationship set Manages is said to be total (since each department should have a manager) but the participation of the entity set Employees in Manages is partial since not every employee gets to manage a department.

Weak Entity:

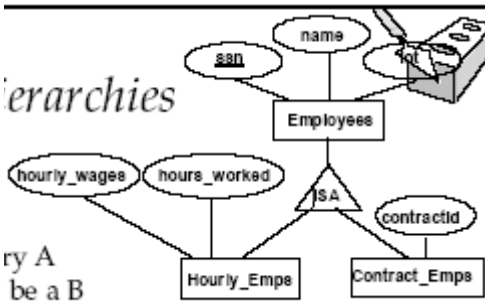


Suppose that employees can purchase policies to cover their dependants. We wish to record info about policies including who is covered by each policy. The attribute pname doesn't identify a dependent uniquely. it should be combined with ssn of employee

A weak entity can be identified uniquely only by considering some of its attributes in conjunction with the primary key of another entity which is called identifying owner.

The weak entity set must have a total participation in the identifying relationship.

Class Hierarchies:



Sometimes it is natural to classify the entities in an entity set into subclasses. For ex: employees are classified into hourly_emps and Contract_emps.

A class hierarchy can be viewed in one of the two ways:

Employees is specialized into subclasses Specialization is the process of identifying subsets of an entity set (superset) that share some distinguishing characteristic. First superset is defined, then subsets specific attributes and relationship sets are added.

Hours_Emps and Contract_Emps are generalized by Employees. Generalization consists of identifying some common characteristics of a collection of entity sets and creating a new entity set that contain entities possessing these common characteristics.

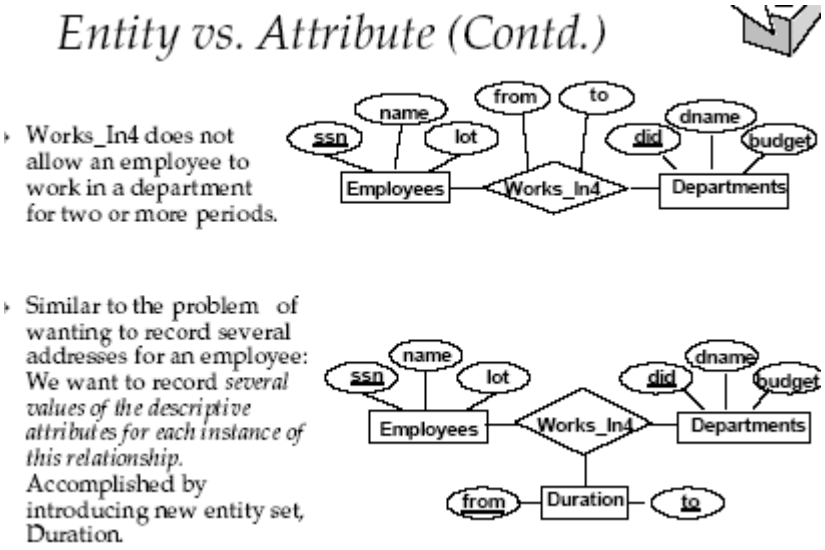
Aggregation: pending

Conceptual Database design with the ER Model:

Design choices:

- Should a concept be modeled as an entity or an attribute?
- Should a concept be modeled as an entity or a relationship?
- Identifying relationships: Binary or ternary?
- Aggregation?

Constraints in the ER Model:
A lot of data semantics can (and should) be captured.
But some constraints cannot be captured in ER diagrams.
Entity vs attribute
Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
_ Depends upon the use we want to make of address information, and the semantics of the data:
• If we have several addresses per employee, *address* must be an entity (since attributes cannot be setvalued).
• If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).



Cardinality = no of rows , degree = no of columns, all rows distinct

IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
_ ICs are specified when schema is defined.
_ ICs are checked when relations are modified.

A key Constraint is a statement that a certain minimal subset of fields of a relation Is a unique identifier for a tuple.
A set of fields that uniquely identifies a tuple according to a key constraint is called a Candidate key for the relation.
A super key is a set of fields that contains a key.
Out of all available candidate keys, a database designer can identify a primary key.

Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.
If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

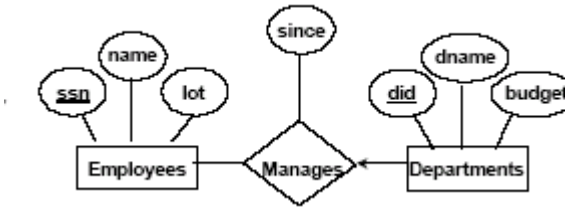
Enforcing integrity constraints:

The impact of domain, Primary key and Unique constraints is straightforward. If an insert, delete or update command causes a violation it is rejected.

Default is NO ACTION (*delete/update is rejected*)
_ CASCADE (also delete all tuples that refer to deleted tuple)
_ SET NULL / SET DEFAULT(sets foreign key value of referencing tuple)
eg: on delete cascade, on update no action.

ER to Relations:

An entity set is mapped to a relation. Each attribute of the entity set becomes an attribute of the table.
Each dept has at most one manager, according to the *key constraint* on Manages.
Map relationship to a table:



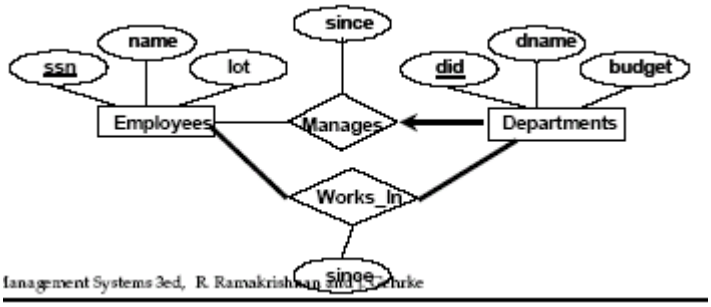

```
CREATE TABLE  Manages(
ssn CHAR(11),
did  INTEGER,
since  DATE,
PRIMARY KEY  (did),
FOREIGN KEY (ssn) REFERENCES Employees,
FOREIGN KEY (did) REFERENCES Departments)

CREATE TABLE  Dept_Mgr(
did  INTEGER,
dname CHAR(20),
budget  REAL,
ssn CHAR(11),
since  DATE,
PRIMARY KEY  (did),
```

FOREIGN KEY (ssn) REFERENCES Employee

_ Note that did(department id) is the key now!
_ Separate tables forEmployees and Departments.
_ Since each department has a unique manager, we could instead combine Manages and Departments.

Managing Participation Constraints:
If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
•Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



CREATE TABLE Dept_Mgr(
did INTEGER,
dname CHAR(20),
budget REAL,
ssn CHAR(11) NOT NULL,
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees,
ON DELETE NO ACTION)

handling weak entities:
Weak entity set and identifying relationship set are translated into a single table.
_ When the owner entity is deleted, all owned weak entities must also be deleted.
CREATE TABLE Dep_Policy (
pname CHAR(20),
age INTEGER,
cost REAL,
ssn CHAR(11) NOT NULL,
PRIMARY KEY (pname, ssn),
FOREIGN KEY (ssn) REFERENCES Employees,
ON DELETE CASCADE)

Translating ISA hierarchies:
General approach:
_ 3 relations: Employees, Hourly_Emps and Contract_Emps.

•*Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn)*; must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
•Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
_ Alternative: Just Hourly_Emps and Contract_Emps.
_ *Hourly_Emps*: *ssn*, *name, lot, hourly_wages, hours_worked.*
_ Each employee must be in one of these two subclasses.

Views:
A *view* is just a relation, but we store a *definition*, rather than a set of tuples.
Views can be dropped using the DROP VIEW command.

Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).

Relational Algebra:
Relational Algebra
_ Basic operations:
_ *Selection* (□) Selects a subset of rows from relation.
_ *Projection* (□) Deletes unwanted columns from relation.
_ *Cross-product* (x) Allows us to combine two relations.
_ *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
_ *Union* (U) Tuples in reln. 1 and in reln. 2.
_ Additional operations:
_ Intersection, *join*, division, renaming: Not essential, but (very!) useful.
_ Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Projection:
Deletes attributes that are not in *projection list*. Projection operator has to eliminate *duplicates*!

Selection:
Selects rows that satisfy *selection condition*.

Union Intersection and set difference
All of these operations take two input relations, which must be *union-compatible*:
_ Same number of fields.
_ `Corresponding' fields have the same type.

Cross Product:
_ Each row of S1 is paired with each row of R1.
_ *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.

Joins:
Conditional join is also called theta join.
$R \ X_{r.sid > s.sid} \ S$

Equi-Join : A special case of conditional join where condition contains only equality.
Like r.sid=s.sid
Division:
Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
If the set of *y* values (boats) associated with an *x* value
(sailor) in *A* contains all *y* values in *B*, the *x* value is in *A/B*.

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2
B

sno
s1
s2
s3
s4
A/B

Query Optimization example:
$\Pi_{sname}(o^{\cdot\cdot}(bid=103)r \text{ X } s)$ this can be optimized in the following way
$\Pi_{sname}((o^{\cdot\cdot}(bid=103)r) \text{ X } s)$

Disk storage:
1) disk space manager is responsible for disk space mgt.
2) fie manager which provides abstraction of a file of records to DBMS issues requests for read or write operations.
3) file mgt layer is responsible for arranging records in pages.
4) when a record is needed, the particular page must be brought into RAM. the page on which record lies is identified by file manager. sometimes it uses auxiliary datastructures to determine the page quickly. After identifying the required page, the file manager issues a request to the layer of DBMS code called the buffer manager. it fetches the requested page into a portion of main mem called buffer pool and tells the location of page to the file manager.

Why not store everything in main memory:
other reasons:
1. on 32 bit systems the no. of addresses can be referenced is limited to 2 power 32, but our data is not limited.
2. secondary and tertiary devices are nonvolatile.

Indexes from Navathe:

1) Additional auxiliary access structures called indexes, which are used to speed up the retrieval of records in response to certain search conditions.
2) The index structures provide secondary access paths, which provide alternative ways of accessing data without affecting the physical placement of records.
3) They enable efficient access to records based on the indexing fields that are used to construct the index.

4) Any field can be used to create an index and multiple indexes on different fields can be constructed on the same file.
5) The most prevalent types of indexes are based on ordered files(single level indexes) and tree data structures(multi level indexes, B+ trees).
6) ISAM(Indexed Sequential Access Method) is a popular indexing scheme is based on single-level index.
7) Different types of single level indexes – primary, secondary, and clustering.
8) B+ trees are the data structures commonly used in DBMSs to implement dynamically changing multilevel indexes.
9) Single level ordered indexes:
10) The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value. The values in the index are ordered so that we can do a binary search on the index.
11) As the index file is much smaller than actual data searching index file is reasonably efficient.
12) Multi level indexing creates index to the index itself.
13) A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field – primary key of the data file and the second field is a pointer to a disk block.
14) There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two field values.
15) The first record in each block of the data file is called the anchor or block record.
16) Indexes can also be characterized as dense or sparse. A dense index has an index entry for every search key value(and hence every record) in the data file.
17) A sparse index has index entries for only some of the search values.
18) A primary index is hence a non-dense (sparse) index , since it includes an entry for each disk block of the data file rather than for every search value.
19) If records of a file are physically ordered on a non-key field (often doesn't have distinct value for each record) that field is called clustering field.
20) We can create clustering index to speed up the retrieval of records that have the same value for the clustering field.
21) A clustering index is also an ordered file with 2 fields; the first is of the same type as the clustering field of the data file and the second is a block pointer.
22) There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record with that value for its clustering field.
23) Notice in clustered index record insertion and deletion still cause problems because the data records are physically ordered. To alleviate this problem of insertion, it is common to reserve a whole block for each value of the clustering field; all records with that value are placed in the block. This makes insertion and deletion relatively straight forward.
24) A clustering index is another example of a non-dense index, since it has an entry for every distinct value of the indexing field rather than for every record in the file.
25) Problems with primary and clustering indexes:
26) As with any ordered file, insertion and deletion becomes very expensive.
27) • We might also have to move records in the data file.
28) • When the data file is changed the anchor items might change.

29) A secondary index is also an ordered file with two fields. First is of the same data type as some non ordering field of the data file that is an indexing field.
30) In secondary index there is one index entry for each record in the data file, which contains the value of the secondary key and a pointer either to the block in which the record is stored or to the record itself. Hence secondary index is dense index.
31) Secondary index usually needs more storage space and longer search time than does a primary index, because of large no of entries.
32) the indexing schemes we discussed thus far involve an ordered index file. A binary search is applied to the index to locate pointers to a disk block or record.
33) A binary search requires approximately logb(to base2) block accesses for an index with b blocks.
34) Multi level indexing:

35) The idea behind multilevel index is to Reduce the part of the index we need to
36) search after each test.• The reduction will be the blocking factor.• The blocking factor is called the *fan-out*. Refer 168 page for diagram in navathe.

37) B and B+ trees are special cases of tree data structures.
Search trees:
• A search tree is a tree that is used to guide the search for a record given the value of one of the record's fields.
• A search tree of order *p* consists of nodes that have at most *p-1 search* values and *p* pointers.

B trees and B+ trees are pending

Query Optimization:

Main heuristic rules is to apply select and project operations before applying the join or other binary operations. This is because the size of the file resulting from a binary operation such as join is usually a multiplicative function of the sizes of the input files. The select and project operations reduce the size of a file and hence should be applied before a join or other binary operation.

Distributed Databases:
Data in a distributed database system is stored across several sites. Each site runs independently of the other sites.
2 properties are considered desirable;
1) Distributed data independence.(users should able to ask queries without specifying where the referenced relations are located)
2) Distributed transaction atomicity.(users should able to write transactions that access and update data at several sites just as they would write transactions over located data.)
If data is distributed but all servers run the same DBMS s/w then we have homogeneous distributed database system other wise heterogeneous distributed system(multidatabase system)
Heterogeneous systems require well accepted standards for gateway protocols. A gateway protocol is an API that exposes DBMS functionality to external applications. Eg: odbc,jdbc.

Distributed data management comes at a significant cost in terms of performance, software complexity and administration difficulty.

Distributed DBMS architectures:

1) client server (client sends query request and server responds with data.)
Disadv: doesn't allow a single query to span multiple servers.
2) collaborating server systems (a variation of client server. When a server receives a query that requires access to data at other servers, it generates appropriate subqueries to be executed by other servers and puts the results together to compute answers to the original query.)
3) middleware systems(is designed to allow a single query to span multiple servers,without requiring all database servers to be capable of managing such multi site execution strategies. We can think of this special server as a layer of software that coordinates the execution of queries and transactions across one or more database servers.)

Fragmentation:

Accessing a relation (stored at remote site) requires message passing costs, to reduce this, a single relation is partitioned or fragmented across several sites, with fragments stored at the sites where they are most often accessed, or replicated at each site.

In horizontal fragmentation each fragment consists of a subset of rows of the original relation. In vertical fragmentation each fragment consists of a subset of columns of the original relation.

Horizontal fragmentation:
Eg: employee tuples might be organized into fragments b;y city with city. By storing data fragments in the database site at the corresponding city – delhi data is most likely to be updated and queried from delhi.

Vertical fragmentation:
Eg: the tuples in a given vertical fragment are identified by a projection query.

The union of the horizontal fragments must be equal to the original relation.
Fragments are usually also required to be disjoint.
The collection of vertical fragments should be a lossless join decomposition.

Replication: means we store several copies of a relation or relation fragment.
Advantages:
Increased availability of data.
Faster query evaluation.

Non join queries in a distributed dbms:

In Distributed DBMS, even simple operations such a scanning a relation, selection and projection are affected by fragmentation and replication.

```
Select s.age
    From sailors s
    Where s.rating > 3 and s.rating < 7
```

If suppose that sailors relation is horizontally fragmented with all tuples having a rating <5 are stored in one place and remaining at some other place the dbms must evaluate the query at two locations and the result is the union of two results

And in the query if there is avg or count then union is not sufficient.

If suppose the sailors table is vertically fragmented with sid and rating field at one place and sname and age field (along with sid (for synchronization))at some other place. Then the dbms has to reconstruct the sailors relation by joining two fragments.

Joins in a Distributed dbms:

Joining relations located at different sites is very expensive.

Fetch as needed: technique used is nested joins
Ship to one site
Semijoins and bloom joins : technique used is sort-merge join

DBMS FAQ

1) Transaction Management

- A *transaction* is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

2) Entity Sets

- A database can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An entity is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays

3) Keys

- A *super key* of an entity set is a set of one or more attributes whose values uniquely determine each entity.
- A *candidate key* of an entity set is a minimal super key
  - *social-security* is candidate key of *customer*
  - *account-number* is candidate key of *account*
- Although several candidate keys may exist, one of the candidate keys is selected to be the *primary key*.
- The combination of primary keys of the participating entity sets forms a candidate key of a relationship set.
  - must consider the mapping cardinality and the semantics of the relationship set when selecting the *primary key*.
  - (*social security, account-number*) is the primary key of *depositor*)

4) Weak Entity Sets

- An entity set that does not have a primary key is referred to as a *weak entity* set.
- The existence of a weak entity set depends on the existence of a strong entity set, it must relate to the strong set via a one-to-many relationship set.
- The *discriminator (or partial key)* of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.

5) Determining Keys from E-R Sets

- Strong entity set. The primary key of the entity set becomes the primary key of the relation.
- Weak entity set. The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set.
- Relationship set. The union of the primary keys of the related entity sets becomes a super key of the relation.

For binary many-to-one relationship sets, the primary key of the "many" entity set becomes the relation's primary key.
For one-to-one relationship sets, the relation's primary key can be that of either entity set.

6) Relational Algebra

- Procedural language
- Six basic operators
- select
- project
- union
- set difference

- Cartesian product
- rename
- The operators take two or more relations as inputs and give a new relation as a result.

7) Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
- *null* signifies that the value is unknown or does not exist
- All comparisons involving *null* are false by definition.

8) Aggregate Functions

- Aggregation operator ς takes a collection of values and returns a single value as a result.
- avg: average value, min: minimum value, max: maximum value
  sum: sum of values, count: number of values
- G1, G2, …, Gn ς F1, A1, F2, A2, …, Fn, An (*E*)
- *E* is any relational-algebra expression
- *G*1, *G*2 …, *G*n is a list of attributes on which to group
- *Fi* is an aggregate function
- *Ai* is an attribute name

9) Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by
  - $\Pi_{CUSTOMER\text{-}NAME,\ LOAN\text{-}NUMBER}$ (*borrower* ⋈ *loan*)
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a *view*.

10) Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes, *null* signifies an unknown value or that a value does not exist.

- The result of any arithmetic expression involving *null* is *null.*

- Roughly speaking, all comparisons involving *null* return *false*. More precisely.
  - Any comparison with *null* returns *unknown*

- (*unknown* or *unknown) - unknown,*
  (true and *unknown) - unknown, (false* and *unknown) = false, (unknown* and *unknown) = unknown*

- Result of where clause predicate is treated as *false* if it evaluates to *unknown*

- "*P* is unknown" evaluates to true if predicate *P* evaluates to *unknown*

11) Domain Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

- Domain constraints are the most elementary form of integrity constraint.

- They test values inserted in the database, and test queries to ensure that the comparisons make sense.

E.g. not null, check, primary key

12) Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

  - Example: If "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".

Referential Integrity in SQL

- Primary and candidate keys and foreign keys can be specified as part of the SQL create table statement:

- The primary key clause of the create table statement includes a list of the attributes that comprise the primary key.

- The unique key clause of the create table statement includes a list of the attributes that comprise a candidate key.

- The foreign key clause of the create table statement includes both a list of the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.

create table *account (branch-name* char(15),*account-number* char(10) not null, *balance* integer,primary key (*account-number),* foreign key (*branch-name)* references *branch)*

13) Triggers

- A *trigger* is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
    - Specify the conditions under which the trigger is to be executed.
    - Specify the actions to be taken when the trigger executes.

14) Functional Dependencies

- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.

15) Indexing

- Indexing mechanisms used to speed up access to desired data.
- E.g., author catalog in library
- Search Key - attribute to set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
- Ordered indices: search keys are stored in sorted order
- Hash indices: search keys are distributed uniformly across "buckets" using a "hash function".

16) B+-Tree Index Files

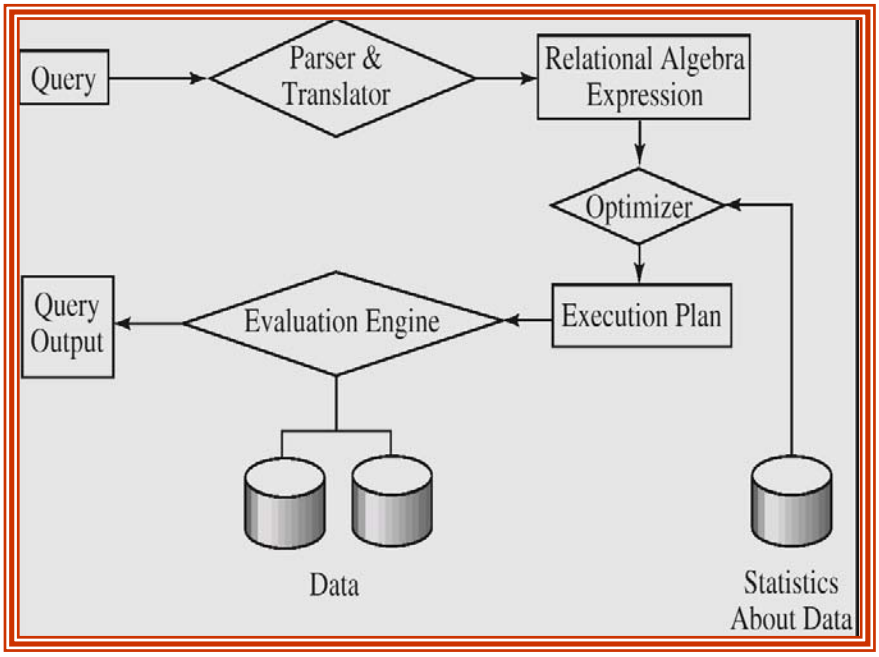
B+-tree indices are an alternative to indexed-sequential files.

- Disadvantage of indexed-sequential files: performance degrades as file grows, since many overflow blocks get created. Periodic reorganization of entire file is required.
- Advantage of B+-tree index files: automatically reorganizes itself with small, local, changes, in the face of insertions and deletions. Reorganization of entire file is not required to maintain performance.

- Disadvantage of B+-trees: extra insertion and deletion overhead, space overhead.
- Advantages of B+-trees outweigh disadvantages, and they are used extensively.
- A B+-tree is a rooted tree satisfying the following properties:
- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between $[n/2]$ and n children.
- A leaf node has between $[(n–1)/2]$ and $n–1$ values
- Special cases: if the root is not a leaf, it has at least 2 children. If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n–1)$ values.
- 17) Comparison of Ordered Indexing and Hashing
- Cost of periodic re-organization
- Relative frequency of insertions and deletions
- Is it desirable to optimize average access time at the expense of worst-case access time?
- Expected type of queries:
- Hashing is generally better at retrieving records having a specified value of the key.
- If range queries are common, ordered indices are to be preferred

18) Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Parsing and translation

1) Translate the query into its internal form. This is then translated into relational algebra.

2) Parser checks syntax, verifies relations

Evaluation

The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

Optimization :– finding the cheapest evaluation plan for a query.

- Given relational algebra expression may have many equivalent expressions
  E.g., $\sigma_{balance<2500}(\prod_{balance}(account))$ is equivalent to $\prod_{balance}(\sigma_{balance<2500}(account))$

- Any relational-algebra expression can be evaluated in many ways.
  Annotated expression specifying detailed evaluation strategy is called an evaluation-plan.
  E.g., can use an index on *balance* to find accounts with balance < 2500, or

can perform complete relation scan and discard accounts with balance ≥ 2500

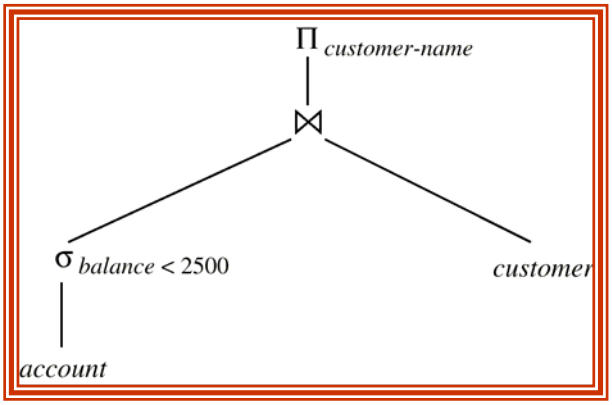
- Amongst all equivalent expressions, try to choose the one with cheapest possible evaluation plan. Cost DBMS catalog.

19) Measures of Query Cost

1) Many possible ways to estimate cost, for instance *disk accesses, CPU* time, or even *communication overhead* in a distributed or parallel system.
2) Typically disk access is the predominant cost, and is also relatively easy to estimate. Therefore *number of block transfers from disk* is used as a measure of the actual cost of evaluation. It is assumed that all transfers of blocks have the same cost.
3) Costs of algorithms depend on the size of the buffer in main memory, as having more memory reduces need for disk access. Thus memory size should be a parameter while estimating cost; often use worst case estimates.
4) We refer to the cost estimate of algorithm *A* as *EA, We* do not include cost to writing output to disk.

20) Evaluation of Expressions

1) Materialization: evaluate one operation at a time, starting at the lowest-level. Use intermediate results materialized into temporary relations to evaluate next-level operations.
2) E.g., in figure below, compute and store
then compute the store its join with *customer,* and finally compute the projections on *customer-name.*



3) Pipelining: evaluate several operations simultaneously, passing the results of one operation on to the next.
4) E.g., in expression in previous slide, don't store result of – instead, pass tuples directly to the join.. Similarly, don't store result of join, pass tuples directly to projection.

5) Much cheaper than materialization: no need to store a temporary relation to disk.
6) Pipelining may not always be possible – e.g., sort, hash-join.
7) For pipelining to be effective, use evaluation algorithms that generate output tuples even as tuples are received for inputs to the operation.
8) Pipelines can be executed in two ways: demand driven and producer driven.

21) ACID Properties

To preserve integrity of data, the database system must ensure:

1) Atomicity: Either all operations of the transaction are properly reflected in the database or none are.
2) Consistency: Execution of a transaction in isolation preserves the consistency of the database.
3) Isolation: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions. This is, for every pair of transactions *Ti* and *Tj,* it appears to *Ti* that either *Tj,* finished execution before *Ti* started, or *Tj* started execution after *Ti* finished.
4) Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

SQL

1. Which is the subset of SQL commands used to manipulate Oracle Database structures, including tables?
Ans) Data Definition Language (DDL)

2. What operator performs pattern matching?
Ans) LIKE operator

3. What operator tests column for the absence of data?
Ans) IS NULL operator

4. Which command executes the contents of a specified file?
Ans) START <filename> or @<filename>

5. What is the parameter substitution symbol used with INSERT INTO command?
Ans) &

6. Which command displays the SQL command in the SQL buffer, and then executes it?
Ans) RUN

7. What are the wildcards used for pattern matching?
Ans) _ for single character substitution and % for multi-character substitution

8. State true or false. EXISTS, SOME, ANY are operators in SQL.
Ans) True

9. State true or false. !=, <>, ^= all denote the same operation.
Ans) True

10. What are the privileges that can be granted on a table by a user to others?
Ans) Insert, update, delete, select, references, index, execute, alter, all

11. What command is used to get back the privileges offered by the GRANT command?
Ans) REVOKE

12. Which system tables contain information on privileges granted and privileges obtained?
Ans) USER_TAB_PRIVS_MADE, USER_TAB_PRIVS_RECD

13. Which system table contains information on constraints on all the tables created?
Ans) USER_CONSTRAINTS

14. TRUNCATE TABLE EMP;
DELETE FROM EMP;
Will the outputs of the above two commands differ?
Ans) Both will result in deleting all the rows in the table EMP.

15. What is the difference between TRUNCATE and DELETE commands?
Ans) TRUNCATE is a DDL command whereas DELETE is a DML command. Hence DELETE operation can be rolled back, but TRUNCATE operation cannot be rolled back. WHERE clause can be used with DELETE and not with TRUNCATE.

16. What command is used to create a table by copying the structure of another table?
Answer :
CREATE TABLE .. AS SELECT command
Explanation :
To copy only the structure, the WHERE clause of the SELECT command should contain a FALSE statement as in the following.
CREATE TABLE NEWTABLE AS SELECT * FROM EXISTINGTABLE WHERE 1=2;
If the WHERE condition is true, then all the rows or rows satisfying the condition will be copied to the new table.

17. What will be the output of the following query?
*SELECT REPLACE(TRANSLATE(LTRIM(RTRIM('!! ATHEN !!','!'), '!'), 'AN', '**'),'*','TROUBLE') FROM DUAL;*
TROUBLETHETROUBLE

18. What will be the output of the following query?
SELECT DECODE(TRANSLATE('A','1234567890','1111111111'), '1','YES', 'NO');
Answer :
NO
Explanation :
The query checks whether a given string is a numerical digit.

19. What does the following query do?
Ans) *SELECT SAL + NVL(COMM,0) FROM EMP;*
This displays the total salary of all employees. The null values in the commission column will be replaced by 0 and added to salary.

20. Which date function is used to find the difference between two dates?
Ans) MONTHS_BETWEEN

21. Why does the following command give a compilation error?
DROP TABLE &TABLE_NAME;
Variable names should start with an alphabet. Here the table name starts with an '&' symbol.

22. What is the advantage of specifying WITH GRANT OPTION in the GRANT command?
Ans) The privilege receiver can further grant the privileges he/she has obtained from the owner to any other user.

23. What is the use of the DROP option in the ALTER TABLE command?
Ans) It is used to drop constraints specified on the table.

24. What is the value of 'comm' and 'sal' after executing the following query if the initial value of 'sal' is 10000?
Ans)*UPDATE EMP SET SAL = SAL + 1000, COMM = SAL*0.1;*
sal = 11000, comm = 1000

25. What is the use of DESC in SQL?
Answer :
DESC has two purposes. It is used to describe a schema as well as to retrieve rows from table in descending order.
Explanation :
The query SELECT * FROM EMP ORDER BY ENAME DESC will display the output sorted on ENAME in descending order.

26. What is the use of CASCADE CONSTRAINTS?
Ans) When this clause is used with the DROP command, a parent table can be dropped even when a child table exists.

27. Which function is used to find the largest integer less than or equal to a specific value?
Ans) FLOOR

28. What is the output of the following query?
Ans) *SELECT TRUNC(1234.5678,-2) FROM DUAL;*
1200

SQL – QUERIES

I. SCHEMAS

Table 1 : ***STUDIES***

PNAME (VARCHAR), SPLACE (VARCHAR), COURSE (VARCHAR), CCOST (NUMBER)

Table 2 : ***SOFTWARE***

PNAME (VARCHAR), TITLE (VARCHAR), DEVIN (VARCHAR), SCOST (NUMBER), DCOST (NUMBER), SOLD (NUMBER)

Table 3 : ***PROGRAMMER***

PNAME (VARCHAR), DOB (DATE), DOJ (DATE), SEX (CHAR), PROF1 (VARCHAR), PROF2 (VARCHAR), SAL (NUMBER)

LEGEND :

PNAME – Programmer Name, SPLACE – Study Place, CCOST – Course Cost, DEVIN – Developed in, SCOST – Software Cost, DCOST – Development Cost, PROF1 – Proficiency 1

QUERIES :

1) *Find out the selling cost average for packages developed in Oracle.*
2) *Display the names, ages and experience of all programmers.*
3) *Display the names of those who have done the PGDCA course.*
4) *What is the highest number of copies sold by a package?*
5) *Display the names and date of birth of all programmers born in April.*
6) *Display the lowest course fee.*
7) *How many programmers have done the DCA course.*
8) *How much revenue has been earned through the sale of packages developed in C.*
9) *Display the details of software developed by Rakesh.*
10) *How many programmers studied at Pentafour.*
11) *Display the details of packages whose sales crossed the 5000 mark.*
12) *Find out the number of copies which should be sold in order to recover the development cost of each package.*
13) *Display the details of packages for which the development cost has been recovered.*
14) *What is the price of costliest software developed in VB?*
15) *How many packages were developed in Oracle ?*
16) *How many programmers studied at PRAGATHI?*
17) *How many programmers paid 10000 to 15000 for the course?*
18) *What is the average course fee?*
19) *Display the details of programmers knowing C.*
20) *How many programmers know either C or Pascal?*
21) *How many programmers don't know C and C++?*
22) *How old is the oldest male programmer?*
23) *What is the average age of female programmers?*
24) *Calculate the experience in years for each programmer and display along with their names in descending order.*
25) *Who are the programmers who celebrate their birthdays during the current month?*
26) *How many female programmers are there?*
27) *What are the languages known by the male programmers?*
28) *What is the average salary?*
29) *How many people draw 5000 to 7500?*
30) *Display the details of those who don't know C, C++ or Pascal.*
31) *Display the costliest package developed by each programmer.*

32) Produce the following output for all the male programmers
33) Programmer
34) Mr. Arvind – has 15 years of experience

KEYS:

1) SELECT AVG(SCOST) FROM SOFTWARE WHERE DEVIN = 'ORACLE';
2) SELECT PNAME,TRUNC(MONTHS_BETWEEN(SYSDATE,DOB)/12) "AGE", TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) "EXPERIENCE" FROM PROGRAMMER;
3) SELECT PNAME FROM STUDIES WHERE COURSE = 'PGDCA';
4) SELECT MAX(SOLD) FROM SOFTWARE;
5) SELECT PNAME, DOB FROM PROGRAMMER WHERE DOB LIKE '%APR%';
6) SELECT MIN(CCOST) FROM STUDIES;
7) SELECT COUNT(*) FROM STUDIES WHERE COURSE = 'DCA';
8) SELECT SUM(SCOST*SOLD-DCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = 'C';
9) SELECT * FROM SOFTWARE WHERE PNAME = 'RAKESH';
10) SELECT * FROM STUDIES WHERE SPLACE = 'PENTAFOUR';
11) SELECT * FROM SOFTWARE WHERE SCOST*SOLD-DCOST > 5000;
12) SELECT CEIL(DCOST/SCOST) FROM SOFTWARE;
13) SELECT * FROM SOFTWARE WHERE SCOST*SOLD >= DCOST;
14) SELECT MAX(SCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = 'VB';
15) SELECT COUNT(*) FROM SOFTWARE WHERE DEVIN = 'ORACLE';
16) SELECT COUNT(*) FROM STUDIES WHERE SPLACE = 'PRAGATHI';
17) SELECT COUNT(*) FROM STUDIES WHERE CCOST BETWEEN 10000 AND 15000;
18) SELECT AVG(CCOST) FROM STUDIES;
19) SELECT * FROM PROGRAMMER WHERE PROF1 = 'C' OR PROF2 = 'C';
20) SELECT * FROM PROGRAMMER WHERE PROF1 IN ('C','PASCAL') OR PROF2 IN ('C','PASCAL');
21) SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++') AND PROF2 NOT IN ('C','C++');
22) SELECT TRUNC(MAX(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'M';
23) SELECT TRUNC(AVG(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'F';
24) SELECT PNAME, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) FROM PROGRAMMER ORDER BY PNAME DESC;
25) SELECT PNAME FROM PROGRAMMER WHERE TO_CHAR(DOB,'MON') = TO_CHAR(SYSDATE,'MON');
26) SELECT COUNT(*) FROM PROGRAMMER WHERE SEX = 'F';
27) SELECT DISTINCT(PROF1) FROM PROGRAMMER WHERE SEX = 'M';
28) SELECT AVG(SAL) FROM PROGRAMMER;
29) SELECT COUNT(*) FROM PROGRAMMER WHERE SAL BETWEEN 5000 AND 7500;
30) SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++','PASCAL') AND PROF2 NOT IN ('C','C++','PASCAL');

31) SELECT PNAME,TITLE,SCOST FROM SOFTWARE WHERE SCOST IN (SELECT MAX(SCOST) FROM SOFTWARE GROUP BY PNAME);
32) 32.SELECT 'Mr.' || PNAME || ' - has ' || TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'M' UNION SELECT 'Ms.' || PNAME || ' - has ' || TRUNC (MONTHS_BETWEEN (SYSDATE,DOJ)/12) || ' years of experience' "Programmer" FROM PROGRAMMER WHERE SEX = 'F';

II . SCHEMA :

Table 1 : **DEPT**

DEPTNO (NOT NULL , NUMBER(2)), DNAME (VARCHAR2(14)),
LOC (VARCHAR2(13)

Table 2 : **EMP**

EMPNO (NOT NULL , NUMBER(4)), ENAME (VARCHAR2(10)),
JOB (VARCHAR2(9)), MGR (NUMBER(4)), HIREDATE (DATE),
SAL (NUMBER(7,2)), COMM (NUMBER(7,2)), DEPTNO (NUMBER(2))

MGR is the empno of the employee whom the employee reports to. DEPTNO is a foreign key.

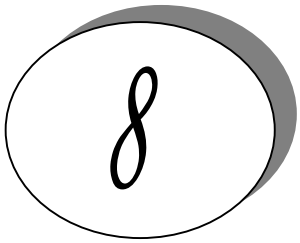
QUERIES

1. List all the employees who have at least one person reporting to them.
2. List the employee details if and only if more than 10 employees are present in department no 10.
3. List the name of the employees with their immediate higher authority.
4. List all the employees who do not manage any one.
5. List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20.
6. List the details of the employee earning more than the highest paid manager.
7. List the highest salary paid for each job.
8. Find the most recently hired employee in each department.
9. In which year did most people join the company? Display the year and the number of employees.
10. Which department has the highest annual remuneration bill?
11. Write a query to display a '' against the row of the most recently hired employee.*
12. Write a correlated sub-query to list out the employees who earn more than the average salary of their department.
13. Find the nth maximum salary.
14. Select the duplicate records (Records, which are inserted, that already exist) in the EMP table.

15. Write a query to list the length of service of the employees (of the form n years and m months).

KEYS:

```
1.  SELECT DISTINCT(A.ENAME) FROM EMP A, EMP B WHERE A.EMPNO
= B.MGR;   or  SELECT ENAME FROM EMP WHERE EMPNO IN (SELECT
MGR FROM EMP);
2.  SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM
EMP GROUP BY DEPTNO HAVING COUNT(EMPNO)>10 AND DEPTNO=10);
3.  SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO" FROM EMP
A, EMP B WHERE A.MGR=B.EMPNO;
4.  SELECT * FROM EMP WHERE EMPNO IN ( SELECT EMPNO FROM EMP
MINUS SELECT MGR FROM EMP);
5.  SELECT * FROM EMP WHERE SAL > ( SELECT MIN(SAL) FROM EMP
GROUP BY DEPTNO HAVING DEPTNO=20);
6.  SELECT * FROM EMP WHERE SAL > ( SELECT MAX(SAL) FROM EMP
GROUP BY JOB HAVING JOB = 'MANAGER' );
7.  SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB;
8.  SELECT * FROM EMP WHERE (DEPTNO, HIREDATE) IN (SELECT
DEPTNO, MAX(HIREDATE) FROM EMP GROUP BY DEPTNO);
9.  SELECT TO_CHAR(HIREDATE,'YYYY') "YEAR", COUNT(EMPNO) "NO.
OF EMPLOYEES" FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY')
HAVING COUNT(EMPNO) = (SELECT MAX(COUNT(EMPNO)) FROM EMP
GROUP BY TO_CHAR(HIREDATE,'YYYY'));
10. SELECT DEPTNO, LPAD(SUM(12*(SAL+NVL(COMM,0))),15)
"COMPENSATION" FROM EMP GROUP BY DEPTNO HAVING SUM(
12*(SAL+NVL(COMM,0))) = (SELECT MAX(SUM(12*(SAL+NVL(COMM,0))))
FROM EMP GROUP BY DEPTNO);
11. SELECT ENAME, HIREDATE, LPAD('*',8) "RECENTLY HIRED" FROM
EMP WHERE HIREDATE = (SELECT MAX(HIREDATE) FROM EMP) UNION
SELECT ENAME NAME, HIREDATE, LPAD(' ',15) "RECENTLY HIRED" FROM
EMP WHERE HIREDATE != (SELECT MAX(HIREDATE) FROM EMP);
12. SELECT ENAME,SAL FROM EMP E WHERE SAL > (SELECT AVG(SAL)
FROM EMP F WHERE E.DEPTNO = F.DEPTNO);
13. SELECT ENAME, SAL FROM EMP A WHERE &N = (SELECT COUNT
(DISTINCT(SAL)) FROM EMP B WHERE A.SAL<=B.SAL);
14. SELECT * FROM EMP A WHERE A.EMPNO IN (SELECT EMPNO FROM
EMP GROUP BY EMPNO HAVING COUNT(EMPNO)>1) AND A.ROWID!=MIN
(ROWID));
15. SELECT ENAME
"EMPLOYEE",TO_CHAR(TRUNC(MONTHS_BETWEEN(SYSDATE,HIREDAT
E)/12))||' YEARS '|| TO_CHAR(TRUNC(MOD(MONTHS_BETWEEN (SYSDATE,
HIREDATE),12)))||' MONTHS ' "LENGTH OF SERVICE" FROM EMP;
```



Computer Networks

---

1. What are the two types of transmission technology available?
Ans) (i) Broadcast and (ii) point-to-point

2. What is subnet?
Ans) A generic term for section of a large networks usually separated by a bridge or router.

3. Difference between the communication and transmission.
Ans) Transmission is a physical movement of information and concern issues like bit polarity, synchronisation, clock etc.
Communication means the meaning full exchange of information between two communication media.

4. What are the possible ways of data exchange?
Ans) (i) Simplex (ii) Half-duplex (iii) Full-duplex.

5. What is SAP?
Ans) Series of interface points that allow other computers to communicate with the other layers of network protocol stack.

6. What do you meant by "triple X" in Networks?
Ans) The function of PAD (Packet Assembler Disassembler) is described in a document known as X.3. The standard protocol has been defined between the terminal and the PAD, called X.28; another standard protocol exists between hte PAD and the network, called X.29. Together, these three recommendations are often called "triple X"

7. What is frame relay, in which layer it comes?
Ans) Frame relay is a packet switching technology. It will operate in the data link layer.

8. What is terminal emulation, in which layer it comes?
Ans) Telnet is also called as terminal emulation. It belongs to application layer.

9. What is Beaconing?
Ans) The process that allows a network to self-repair networks problems. The stations on the network notify the other stations on the ring when they are not receiving the transmissions. Beaconing is used in Token ring and FDDI networks.

10. What is redirector?
Ans) Redirector is software that intercepts file or prints I/O requests and translates them into network requests. This comes under presentation layer.

11. What is NETBIOS and NETBEUI?
Ans) NETBIOS is a programming interface that allows I/O requests to be sent to and received from a remote computer and it hides the networking hardware from applications.
Ans) NETBEUI is NetBIOS extended user interface. A transport protocol designed by microsoft and IBM for the use on small subnets.

12. What is RAID?
Ans) A method for providing fault tolerance by using multiple hard disk drives.

13. What is passive topology?
Ans) When the computers on the network simply listen and receive the signal, they are referred to as passive because they don't amplify the signal in any way. Example for passive topology - linear bus.

14. What is Brouter?
Ans) Hybrid devices that combine the features of both bridges and routers.

15. What is cladding?
Ans) A layer of a glass surrounding the center fiber of glass inside a fiber-optic cable.

16. What is point-to-point protocol
Ans) A communications protocol used to connect computers to remote networking services including Internet service providers.

17. How Gateway is different from Routers?
Ans) A gateway operates at the upper levels of the OSI model and translates information between two completely different network architectures or data formats

18. What is attenuation?
Ans) The degeneration of a signal over distance on a network cable is called attenuation.

19. What is MAC address?
Ans) The address for a device as it is identified at the Media Access Control (MAC) layer in the network architecture. MAC address is usually stored in ROM on the network adapter card and is unique.

20. Difference between bit rate and baud rate.
Ans) Bit rate is the number of bits transmitted during one second whereas baud rate refers to the number of signal units per second that are required to represent those bits.
baud rate = bit rate / N
where N is no-of-bits represented by each signal shift.

21. What is Bandwidth?

Ans) Every line has an upper limit and a lower limit on the frequency of signals it can carry. This limited range is called the bandwidth.

22. What are the types of Transmission media?

Ans) Signals are usually transmitted over some transmission media that are broadly classified in to two categories.

a)Guided Media:

These are those that provide a conduit from one device to another that include twisted-pair, coaxial cable and fiber-optic cable. A signal traveling along any of these media is directed and is contained by the physical limits of the medium. Twisted-pair and coaxial cable use metallic that accept and transport signals in the form of electrical current. Optical fiber is a glass or plastic cable that accepts and transports signals in the form of light.

b) Unguided Media:

This is the wireless media that transport electromagnetic waves without using a physical conductor. Signals are broadcast either through air. This is done through radio communication, satellite communication and cellular telephony.

23. What is Project 802?

Ans) It is a project started by IEEE to set standards to enable intercommunication between equipment from a variety of manufacturers. It is a way for specifying functions of the physical layer, the data link layer and to some extent the network layer to allow for interconnectivity of major LAN
protocols.

It consists of the following:

- 802.1 is an internetworking standard for compatibility of different LANs and MANs across protocols.
- 802.2 Logical link control (LLC) is the upper sublayer of the data link layer which is non-architecture-specific, that is remains the same for all IEEE-defined LANs.
- Media access control (MAC) is the lower sublayer of the data link layer that contains some distinct modules each carrying proprietary information specific to the LAN product being used. The modules are Ethernet LAN (802.3), Token ring LAN (802.4), Token bus LAN (802.5).
- 802.6 is distributed queue dual bus (DQDB) designed to be used in MANs.

24. What is Protocol Data Unit?

Ans) The data unit in the LLC level is called the protocol data unit (PDU). The PDU contains of four fields a destination service access point (DSAP), a source service access point (SSAP), a control field and an information field. DSAP, SSAP are addresses used by the LLC to identify the protocol stacks on the receiving and sending machines that are generating and using the data. The control field specifies whether the PDU frame is a information frame (I - frame) or a supervisory frame (S - frame) or a unnumbered frame (U - frame).

25. What are the different type of networking / internetworking devices?
Ans)
Repeater:
Also called a regenerator, it is an electronic device that operates only at physical layer. It receives the signal in the network before it becomes weak, regenerates the original bit pattern and puts the refreshed copy back in to the link.
Bridges:
These operate both in the physical and data link layers of LANs of same type. They divide a larger network in to smaller segments. They contain logic that allow them to keep the traffic for each segment separate and thus are repeaters that relay a frame only the side of the segment containing the intended recipent and control congestion.
Routers:
They relay packets among multiple interconnected networks (i.e. LANs of different type). They operate in the physical, data link and network layers. They contain software that enable them to determine which of the several possible paths is the best for a particular transmission.
Gateways:
They relay packets among networks that have different protocols (e.g. between a LAN and a WAN). They accept a packet formatted for one protocol and convert it to a packet formatted for another protocol before forwarding it. They operate in all seven layers of the OSI model.

26. What is ICMP?
Ans) ICMP is Internet Control Message Protocol, a network layer protocol of the TCP/IP suite used by hosts and gateways to send notification of datagram problems back to the sender. It uses the echo test / reply to test whether a destination is reachable and responding. It also handles both control and error messages.

27. What are the data units at different layers of the TCP / IP protocol suite?
Ans) The data unit created at the application layer is called a message, at the transport layer the data unit created is called either a segment or an user datagram, at the network layer the data unit created is called the datagram, at the data link layer the datagram is encapsulated in to a frame and finally transmitted as signals along the transmission media.

28. What is difference between ARP and RARP?
Ans) The address resolution protocol (ARP) is used to associate the 32 bit IP address with the 48 bit physical address, used by a host or a router to find the physical address of another host on its network by sending a ARP query packet that includes the IP address of the receiver.
The reverse address resolution protocol (RARP) allows a host to discover its Internet address when it knows only its physical address.

29. What is the minimum and maximum length of the header in the TCP segment and IP datagram?
Ans) The header should have a minimum length of 20 bytes and can have a maximum length of 60 bytes.

30. What is the range of addresses in the classes of internet addresses?

Ans)

Class A	0.0.0.0	-	127.255.255.255
Class B	128.0.0.0	-	191.255.255.255
Class C	192.0.0.0	-	223.255.255.255
Class D	224.0.0.0	-	239.255.255.255
Class E	240.0.0.0	-	247.255.255.255

31. What is the difference between TFTP and FTP application layer protocols?

Ans) The Trivial File Transfer Protocol (TFTP) allows a local host to obtain files from a remote host but does not provide reliability or security. It uses the fundamental packet delivery services offered by UDP.

The File Transfer Protocol (FTP) is the standard mechanism provided by TCP / IP for copying a file from one host to another. It uses the services offer by TCP and so is reliable and secure. It establishes two connections (virtual circuits) between the hosts, one for data transfer and another for control information.

32. What are major types of networks and explain?

Ans)

- Server-based network
- Peer-to-peer network

Peer-to-peer network, computers can act as both servers sharing resources and as clients using the resources.

Server-based networks provide centralized control of network resources and rely on server computers to provide security and network administration

33. What are the important topologies for networks?

Ans)

- *BUS topology:*

In this each computer is directly connected to primary network cable in a single line.

Advantages:

Inexpensive, easy to install, simple to understand, easy to extend.

- *STAR topology:*

In this all computers are connected using a central hub.

Advantages:

Can be inexpensive, easy to install and reconfigure and easy to trouble shoot physical problems.

- *RING topology:*

In this all computers are connected in loop.

Advantages:

All computers have equal access to network media, installation can be simple, and signal does not degrade as much as in other topologies because each computer regenerates it.

34. What is mesh network?

Ans) A network in which there are multiple network links between computers to provide multiple paths for data to travel.

35. What is difference between baseband and broadband transmission?
Ans) In a baseband transmission, the entire bandwidth of the cable is consumed by a single signal. In broadband transmission, signals are sent on multiple frequencies, allowing multiple signals to be sent simultaneously.

36. Explain 5-4-3 rule?
Ans) In a Ethernet network, between any two points on the network ,there can be no more than five network segments or four repeaters, and of those five segments only three of segments can be populated.

37. What MAU?
Ans) In token Ring , hub is called Multistation Access Unit(MAU).

38. What is the difference between routable and non- routable protocols?
Ans) Routable protocols can work with a router and can be used to build large networks. Non-Routable protocols are designed to work on small, local networks and cannot be used with a router

39. Why should you care about the OSI Reference Model?
Ans) It provides a framework for discussing network operations and design.

40. What is logical link control?
Ans) One of two sublayers of the data link layer of OSI reference model, as defined by the IEEE 802 standard. This sublayer is responsible for maintaining the link between computers when they are sending data across the physical network connection.

41. What is virtual channel?
Ans) Virtual channel is normally a connection from one source to one destination, although multicast connections are also permitted. The other name for virtual channel is virtual circuit.

42. What is virtual path?
Ans) Along any transmission path from a given source to a given destination, a group of virtual circuits can be grouped together into what is called path.

43. What is packet filter?
Ans) Packet filter is a standard router equipped with some extra functionality. The extra functionality allows every incoming or outgoing packet to be inspected. Packets meeting some criterion are forwarded normally. Those that fail the test are dropped.

44. What is traffic shaping?
Ans) One of the main causes of congestion is that traffic is often busy. If hosts could be made to transmit at a uniform rate, congestion would be less common. Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This is called traffic shaping.

45. What is multicast routing?
Ans) Sending a message to a group is called multicasting, and its routing algorithm is called multicast routing.

46. What is region?
Ans) When hierarchical routing is used, the routers are divided into what we will call regions, with each router knowing all the details about how to route packets to destinations within its own region, but knowing nothing about the internal structure of other regions.

47. What is silly window syndrome?
Ans) It is a problem that can ruin TCP performance. This problem occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads 1 byte at a time.

48. What are Digrams and Trigrams?
Ans) The most common two letter combinations are called as digrams. e.g. th, in, er, re and an. The most common three letter combinations are called as trigrams. e.g. the, ing, and, and ion.

49. Expand IDEA.
Ans) IDEA stands for International Data Encryption Algorithm.

50. What is wide-mouth frog?
Ans) Wide-mouth frog is the simplest known key distribution center (KDC) authentication protocol.

51. What is Mail Gateway?
Ans) It is a system that performs a protocol translation between different electronic mail delivery protocols.

52. What is IGP (Interior Gateway Protocol)?
Ans) It is any routing protocol used within an autonomous system.

53. What is EGP (Exterior Gateway Protocol)?
Ans) It is the protocol the routers in neighboring autonomous systems use to identify the set of networks that can be reached within or via each autonomous system.

54. What is autonomous system?
Ans) It is a collection of routers under the control of a single administrative authority and that uses a common Interior Gateway Protocol.

55. What is BGP (Border Gateway Protocol)?
Ans) It is a protocol used to advertise the set of networks that can be reached with in an autonomous system. BGP enables this information to be shared with the autonomous system. This is newer than EGP (Exterior Gateway Protocol).

56. What is Gateway-to-Gateway protocol?
Ans) It is a protocol formerly used to exchange routing information between Internet core routers.

57. What is NVT (Network Virtual Terminal)?
Ans) It is a set of rules defining a very simple virtual terminal interaction. The NVT is used in the start of a Telnet session.

58. What is a Multi-homed Host?
Ans) It is a host that has a multiple network interfaces and that requires multiple IP addresses is called as a Multi-homed Host.

59. What is Kerberos?
Ans) It is an authentication service developed at the Massachusetts Institute of Technology. Kerberos uses encryption to prevent intruders from discovering passwords and gaining unauthorized access to files.

60. What is OSPF?
Ans) It is an Internet routing protocol that scales well, can route traffic along multiple paths, and uses knowledge of an Internet's topology to make accurate routing decisions.

61. What is Proxy ARP?
Ans) It is using a router to answer ARP requests. This will be done when the originating host believes that a destination is local, when in fact is lies beyond router.

62. What is SLIP (Serial Line Interface Protocol)?
Ans) It is a very simple protocol used for transmission of IP datagrams across a serial line.

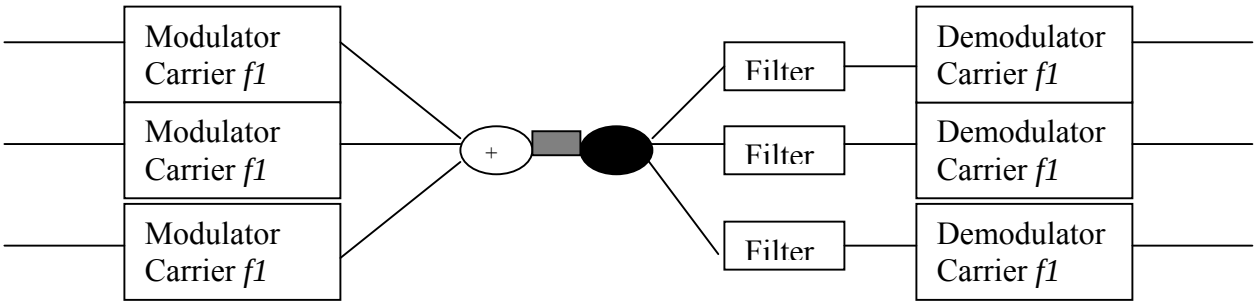
63. What is RIP (Routing Information Protocol)?
Ans) It is a simple protocol used to exchange information between the routers.

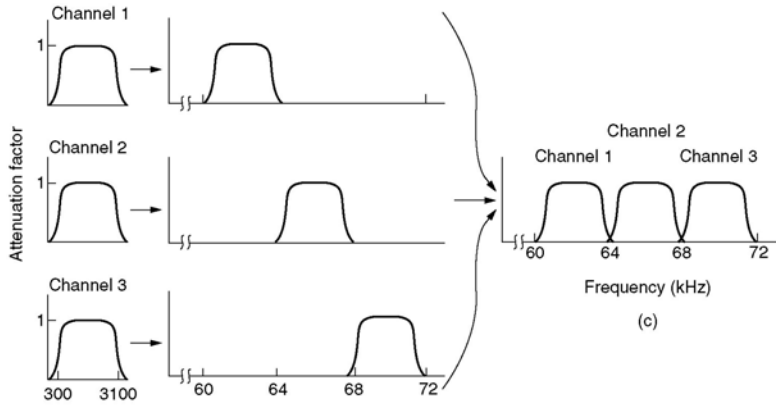
64. What is source route?
Ans) It is a sequence of IP addresses identifying the route a datagram must follow. A source route may optionally be included in an IP datagram header.

65. What are different Types of Multiplexing ?
Ans)
Frequency-Division Multiplexing(FDM): FDM is an analog technique that can be applied when the bandwidth of a link is greater than the combined bandwidth of the signals to be transmitted.In FDM, signals generated by each sending device modulate different carrier frequencies.These modulated signals are then combined into a single composite signal that can be transported by the link.

Carrier frequencies are separated by enough bandwidth to accommodate the modulated signal.These bandwidth ranges are the channels through which the various signals travel.Channels must be separted by strips of unused bandwith to prevent signals from overlapping.In addition, carrier frequency must not interfere with the original data frequencies. ***E.g.*** Cable Television.

Multiplexer
Demultiplexer

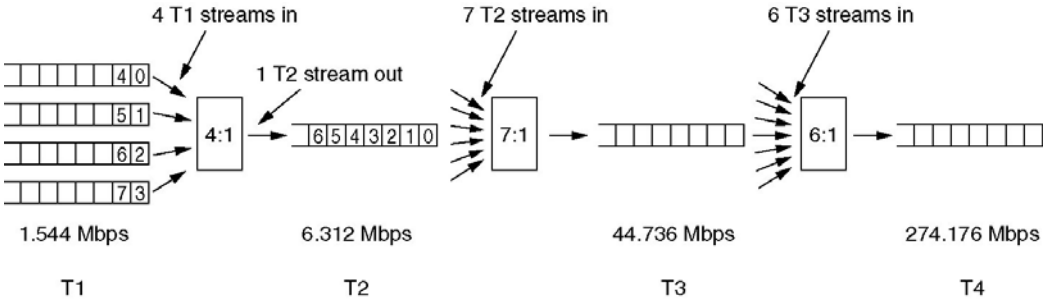




(a)
(c) T

cy.

Time-Division Multiplexing: TDM is a digital process that can be applied when the data rate capacity of the transmission medium is greater than the data rate required by the sending and receiving devices.In such a case,multiple transmissions can occupy a single link by subdividing them and interleaving the portions.



Multiplexing T1 streams into higher carriers.

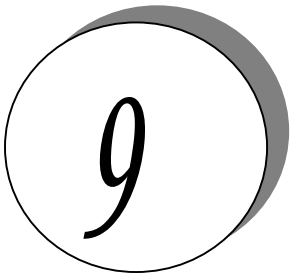
TDM can be implemented in two ways:

Synchrounous TDM: Multiplexer allocates exactly the same time slot to each device at all times, whether or not a device has anything to transmit. If a device is unable to transmit or does not have data to transmit, its time slot remains empty.
Time slots are grouped into frames. A frame consits of one complete cycle of time slots,including one or more slots dedicated to each sending device,plus framing bits.

Asynchrounous TDM: The number of time slots in an asynchrounous TDM frame is based on a statical analysis of the number of input lines that are likely to be transmitting at any given time. Rather than being preasssigned,each slot is available to any of the attached input lines that has data to send.

Inverse Multiplexing: Inverse multiplexing takes the data stream from one high-speed line and breaks it into a portions that can be send across several lower speed lines simultaneously, with no loss in the collective data rate.

Why do we need inverse multiplexing? Think of an organization that wants to send data,voice and video,each of which requires a different data rate.To send voice you may need a 64 Kbps link.To send data,it may need a 128 Kbps link.To accommodate all of these needs the organization has two options.It can lease a 1.544 Mbps channel from a common carrier or it can lease several separate channels of lower data rates.
If they choose the second option they have to use the Inverse Multiplexing.



Operating Systems

Following are a few basic questions that cover the essentials of OS:

1. Explain the concept of Reentrancy.
Ans) It is a useful, memory-saving technique for multiprogrammed timesharing systems. A *Reentrant Procedure* is one in which multiple users can share a single copy of a program during the same period. Reentrancy has 2 key aspects: The program code cannot modify itself, and the local data for each user process must be stored separately. Thus, the permanent part is the code, and the temporary part is the pointer back to the calling program and local variables used by that program. Each execution instance is called *activation*. It executes the code in the permanent part, but has its own copy of local variables/parameters. The temporary part associated with each activation is the *activation record*. Generally, the activation record is kept on the stack.
Note: A reentrant procedure *can* be interrupted and called by an interrupting program, and still execute correctly on returning to the procedure.

2. Explain Belady's Anomaly.
Ans) Also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process' virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

3. What is a binary semaphore? What is its use?
Ans) A binary semaphore is one, which takes only 0 and 1 as values. They are used to implement mutual exclusion and synchronize concurrent processes.

4. What is thrashing?
Ans) It is a phenomenon in virtual memory schemes when the processor spends most of its time swapping pages, rather than executing instructions. This is due to an inordinate number of page faults.

5. List the Coffman's conditions that lead to a deadlock.
Ans)
- Mutual Exclusion: Only one process may use a critical resource at a time.
- Hold & Wait: A process may be allocated some resources while waiting for others.
- No Pre-emption: No resource can be forcible removed from a process holding it.
- Circular Wait: A closed chain of processes exist such that each process holds at least one resource needed by another process in the chain.

6. What are short-, long- and medium-term scheduling?

Ans) Long term scheduler determines which programs are admitted to the system for processing. It controls the *degree of multiprogramming*. Once admitted, a job becomes a process.

Medium term scheduling is part of the swapping function. This relates to processes that are in a blocked or suspended state. They are swapped out of real-memory until they are ready to execute. The swapping-in decision is based on memory-management criteria.

Short term scheduler, also know as a *dispatcher* executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs. It may lead to interruption of one process by preemption.

7. What are turnaround time and response time?

Ans) Turnaround time is the interval between the submission of a job and its completion. Response time is the interval between submission of a request, and the first response to that request.

8. What are the typical elements of a process image?

Ans)

- User data: Modifiable part of user space. May include program data, user stack area, and programs that may be modified.
    - User program: The instructions to be executed.
    - System Stack: Each process has one or more LIFO stacks associated with it. Used to store parameters and calling addresses for procedure and system calls.
- Process control Block (PCB): Info needed by the OS to control processes.

9. What is the Translation Lookaside Buffer (TLB)?

Ans) In a cached system, the base addresses of the last few referenced pages is maintained in registers called the TLB that aids in faster lookup. TLB contains those page-table entries that have been most recently used. Normally, each virtual memory reference causes 2 physical memory accesses-- one to fetch appropriate page-table entry, and one to fetch the desired data. Using TLB in-between, this is reduced to just one physical memory access in cases of TLB-hit.

10. What is the resident set and working set of a process?

Ans) Resident set is that portion of the process image that is actually in real-memory at a particular instant. Working set is that subset of resident set that is actually needed for execution. (Relate this to the variable-window size method for swapping techniques.)

11. When is a system in safe state?

Ans) The set of dispatchable processes is in a safe state if there exists at least one temporal order in which all processes can be run to completion without resulting in a deadlock.

12. What is cycle stealing?

Ans) We encounter cycle stealing in the context of Direct Memory Access (DMA). Either the DMA controller can use the data bus when the CPU does not need it, or it

may force the CPU to temporarily suspend operation. The latter technique is called cycle stealing. Note that cycle stealing can be done only at specific break points in an instruction cycle.

13. What is meant by arm-stickiness?
Ans) If one or a few processes have a high access rate to data on one track of a storage disk, then they may monopolize the device by repeated requests to that track. This generally happens with most common device scheduling algorithms (LIFO, SSTF, C-SCAN, etc). High-density multisurface disks are more likely to be affected by this than low density ones.

14. What are the stipulations of C2 level security?
Ans) C2 level security provides for:
- Discretionary Access Control
- Identification and Authentication
- Auditing
- Resource reuse

15. What is busy waiting?
Ans) The repeated execution of a loop of code while waiting for an event to occur is called busy-waiting. The CPU is not engaged in any real productive activity during this period, and the process does not progress toward completion.

16. Explain the popular multiprocessor thread-scheduling strategies.
Ans)
- *Load Sharing:* Processes are not assigned to a particular processor. A global queue of threads is maintained. Each processor, when idle, selects a thread from this queue. Note that load *balancing* refers to a scheme where work is allocated to processors on a more permanent basis.
- *Gang Scheduling:* A set of related threads is scheduled to run on a set of processors at the same time, on a 1-to-1 basis. Closely related threads / processes may be scheduled this way to reduce synchronization blocking, and minimize process switching. Group scheduling predated this strategy.
- *Dedicated processor assignment:* Provides implicit scheduling defined by assignment of threads to processors. For the duration of program execution, each program is allocated a set of processors equal in number to the number of threads in the program. Processors are chosen from the available pool.
- *Dynamic scheduling:* The number of thread in a program can be altered during the course of execution.

17. When does the condition 'rendezvous' arise?
Ans) In message passing, it is the condition in which, both, the sender and receiver are blocked until the message is delivered.

18. What is a trap and trapdoor?
Ans) Trapdoor is a secret undocumented entry point into a program used to grant access without normal methods of access authentication. A trap is a software interrupt, usually the result of an error condition.

19. What are local and global page replacements?
Ans) Local replacement means that an incoming page is brought in only to the relevant process' address space. Global replacement policy allows any page frame from any process to be replaced. The latter is applicable to variable partitions model only.

20. Define latency, transfer and seek time with respect to disk I/O.
Ans) Seek time is the time required to move the disk arm to the required track. Rotational delay or latency is the time it takes for the beginning of the required sector to reach the head. Sum of seek time (if any) and latency is the access time. Time taken to actually transfer a span of data is transfer time.

21. Describe the Buddy system of memory allocation.
Ans) Free memory is maintained in linked lists, each of equal sized blocks. Any such block is of size 2^k. When some memory is required by a process, the block size of next higher order is chosen, and broken into two. Note that the two such pieces differ in address only in their kth bit. Such pieces are called buddies. When any used block is freed, the OS checks to see if its buddy is also free. If so, it is rejoined, and put into the original free-block linked-list.

22. What is time-stamping?
Ans) It is a technique proposed by Lamport, used to order events in a distributed system without the use of clocks. This scheme is intended to order events consisting of the transmission of messages. Each system 'i' in the network maintains a counter Ci. Every time a system transmits a message, it increments its counter by 1 and attaches the time-stamp Ti to the message. When a message is received, the receiving system 'j' sets its counter Cj to 1 more than the maximum of its current value and the incoming time-stamp Ti. At each site, the ordering of messages is determined by the following rules: For messages x from site i and y from site j, x precedes y if one of the following conditions holds....(a) if Ti<Tj or (b) if Ti=Tj and i<j.

23. How are the wait/signal operations for monitor different from those for semaphores?
Ans) If a process in a monitor signal and no task is waiting on the condition variable, the signal is lost. So this allows easier program design. Whereas in semaphores, every operation affects the value of the semaphore, so the wait and signal operations should be perfectly balanced in the program.

24. In the context of memory management, what are placement and replacement algorithms?
Ans) Placement algorithms determine where in available real-memory to load a program. Common methods are first-fit, next-fit, best-fit. Replacement algorithms are used when memory is full, and one process (or part of a process) needs to be swapped out to accommodate a new program. The replacement algorithm determines which are the partitions to be swapped out.

25. In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?
Ans) For load-time dynamic linking: Load module to be loaded is read into memory. Any reference to a target external module causes that module to be loaded and the references are updated to a relative address from the start base address of the application module.
With run-time dynamic loading: Some of the linking is postponed until actual reference during execution. Then the correct module is loaded and linked.

26. What are demand- and pre-paging?
Ans) With demand paging, a page is brought into memory only when a location on that page is actually referenced during execution. With pre-paging, pages other than the one demanded by a page fault are brought in. The selection of such pages is done based on common access patterns, especially for secondary memory devices.

27. Paging a memory management function, while multiprogramming a processor management function, are the two interdependent?
Ans) Yes.

28. What is page cannibalizing?
Ans) Page swapping or page replacements are called page cannibalizing.

29. What has triggered the need for multitasking in PCs?
Ans)
- Increased speed and memory capacity of microprocessors together with the support fir virtual memory and
- Growth of client server computing

30. What are the four layers that Windows NT have in order to achieve independence?
Ans)
- Hardware abstraction layer
- Kernel
- Subsystems
- System Services.

31. What is SMP?
Ans) To achieve maximum efficiency and reliability a mode of operation known as symmetric multiprocessing is used. In essence, with SMP any process or threads can be assigned to any processor.

32. What are the key object oriented concepts used by Windows NT?
Ans)
- Encapsulation
- Object class and instance

33. Is Windows NT a full blown object oriented operating system? Give reasons.
Ans) No Windows NT is not so, because its not implemented in object oriented language and the data structures reside within one executive component and are not represented as objects and it does not support object oriented capabilities .

34. What is a drawback of MVT?
Ans) It does not have the features like
- ability to support multiple processors
- virtual storage
- source level debugging

35. What is process spawning?
Ans) When the OS at the explicit request of another process creates a process, this action is called process spawning.

36. How many jobs can be run concurrently on MVT?
Ans) 15 jobs

37. List out some reasons for process termination.
Ans)
- Normal completion
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error
- Arithmetic error
- Time overrun
- I/O failure
- Invalid instruction
- Privileged instruction
- Data misuse
- Operator or OS intervention
- Parent termination.

38. What are the reasons for process suspension?
Ans)
- swapping
- interactive user request
- timing
- parent process request

39. What is process migration?
Ans) It is the transfer of sufficient amount of the state of process from one machine to the target machine

40. What is mutant?
Ans) In Windows NT a mutant provides kernel mode or user mode mutual exclusion with the notion of ownership.

41. What is an idle thread?
Ans) The special thread a dispatcher will execute when no ready thread is found.

42. What is FtDisk?
Ans) It is a fault tolerance disk driver for Windows NT.

43. What are the possible threads a thread can have?
Ans)
- Ready
- Standby
- Running
- Waiting
- Transition
- Terminated.

44. What are rings in Windows NT?
Ans) Windows NT uses protection mechanism called rings provides by the process to implement separation between the user mode and kernel mode.

45. What is Executive in Windows NT?
Ans) In Windows NT, executive refers to the operating system code that runs in kernel mode.

46. What are the sub-components of I/O manager in Windows NT?
Ans)
- Network redirector/ Server
- Cache manager.
- File systems
- Network driver
- Device driver

47. What are DDks? Name an operating system that includes this feature.
Ans) DDks are device driver kits, which are equivalent to SDKs for writing device drivers. Windows NT includes DDks.

48. What level of security does Windows NT meets?
Ans) C2 level security.

OS FAQ

1) What are the basic functions of an operating system?
Ans) Operating system controls and coordinates the use of the hardware among the various applications programs for various uses. Operating system acts as resource allocator and manager. Since there are many possibly conflicting requests for resources the operating system must decide which requests are allocated resources to operating the computer system efficiently and fairly. Also operating system is control program which controls the user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

2) Explain briefly about, processor, assembler, compiler, loader, linker and the functions executed by them.
Ans)
Processor:--A processor is the part a computer system that executes instructions .It is also called a CPU

Assembler: -- An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. Some people call these instructions assembler language and others use the term assembly language.

Compiler: --- A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C one line at a time using an editor. The file that is created contains what are called the source statements. The programmer then runs the appropriate language compiler, specifying the name of the file that contains the source statements.

Loader:--In a computer operating system, a loader is a component that locates a given program (which can be an application or, in some cases, part of the operating system itself) in offline storage (such as a hard disk), loads it into main storage (in a personal computer, it's called random access memory), and gives that program control of the compute

Linker: -- Linker performs the linking of libraries with the object code to make the object code into an executable machine code.

3) What is a Real-Time System?
Ans)A real time process is a process that must respond to the events within a certain time period. A real time operating system is an operating system that can run real time processes successfully

4) What is the difference between Hard and Soft real-time systems?
Ans) A hard real-time system guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the operating system to finish any request made of it.

A soft real time system where a critical real-time task gets priority over other tasks and retains that priority until it completes. As in hard real time systems kernel delays need to be bounded

5) What is the important aspect of a real-time system or Mission Critical Systems?
Ans)A real time operating system has well defined fixed time constraints. Process must be done within the defined constraints or the system will fail. An example is the operating system for a flight control computer or an advanced jet airplane. Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.Real-Time systems may be either *hard* or *soft* real-time.
Hard real-time:
-> Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
-> Conflicts with time-sharing systems, not supported by general-purpose operating systems.
Soft real-time:
->Limited utility in industrial control of robotics
->Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

6) What is hard disk and what is its purpose?
Ans) Hard disk is the secondary storage device, which holds the data in bulk, and it holds the data on the magnetic medium of the disk.Hard disks have a hard platter that holds the magnetic medium, the magnetic medium can be easily erased and rewritten, and a typical desktop machine will have a hard disk with a capacity of between 10 and 40 gigabytes. Data is stored onto the disk in the form of files.

7) What is virtual memory?
Ans) A virtual memory is hardware technique where the system appears to have more memory that it actually does. This is done by time-sharing, the physical memory and storage parts of the memory one disk when they are not actively being used.

8)What are the difference phases of software development or software life cycle?
Ans
Specification of the task
Design of algorithms
Implementation *(coding)*
Testing and debugging
Maintenance and evolution of the system
Obsolescence
9) What is cache memory?
Ans) Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory.

10) Differentiate between Complier and Interpreter?
Ans) An interpreter reads one instruction at a time and carries out the actions implied by that instruction. It does not perform any translation. But a compiler translates the entire instructions.

11) Describe different job scheduling in operating systems.
Ans) Scheduling is the activity of the deciding when process will receive the resources they request.
FCFS: --- FCSFS stands for First Come First Served. In FCFS the job that has been waiting the longest is served next.
Round Robin Scheduling: ---Round Robin scheduling is a scheduling method where each process gets a small quantity of time to run and then it is preempted and the next process gets to run. This is called time-sharing and gives the effect of all the processes running at the same time
Shortest Job First: -- The Shortest job First scheduling algorithm is a nonpreemptive scheduling algorithm that chooses the job that will execute the shortest amount of time.
Priority Scheduling: ---Priority scheduling is a scheduling method where at all times the highest priority process is assigned the resource.

12) What are different tasks of Lexical Analysis?
Ans) The purpose of the lexical analyzer is to partition the input text, delivering a sequence of comments and basic symbols. Comments are character sequences to be ignored, while basic symbols are character sequences that correspond to terminal symbols of the grammar defining the phrase structure of the input

13) Why paging is used?
Ans) Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory wherever the latter is available.

14) What are the main difference between Micro-Controller and Micro- Processor?
Ans) A microcontroller is by definition a is a computer on a chip. It includes all the necessary parts (including the memory) all in one IC. You just need to apply the power (and possibly clock signal) to that device and it starts executing the program programmed to it. A microcontroller generally has the main CPU core, ROM/EPROM/EEPROM/FLASH, RAM and some necessary functions (like timers and I/O controllers) all integrated into one chip. The original idea behind the microcontroller was to limit the capabilities of the CPU itself, allowing a complete computer (memory, I/O, interrupts, etc) to fit on the available silicon real estate. Microcontrollers are typically used where processing power isn't so important. More important are generally compact construction, small size, low power consumption and that those chips are cheap. For example controlling a microwave oven is easily accomplished with the smallest of microcontrollers. There is countless number of small electronic devices which are nowadays based on microcontroller. A modern home can include easily tens or hundreds of microcontrollers, as almost every modern device which has electronics have a microcontroller (or more than one) inside.
Microprocessor is generally just the CPU core itself, although nowadays it might have some accessory parts also integrated to the same chip

15) What is Context Switch?
Ans) Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch. Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers which must be copied, the existed of special instructions(such as a single instruction to load or store all registers).

16) What is an Operating System?
Ans) A program that acts as an intermediary between a user of a computer and the computer hardware.
Operating system goals:
Execute user programs and make solving user problems easier.
Make the computer system convenient to use.

17) What is DRAM? In which form does it store data?
Ans) DRAM is the Hershey's chocolate of readable/writable memory: it's not the best, but it's cheap, does the job, and is available almost everywhere you look. DRAM data resides in a cell made of a capacitor and a transistor. The capacitor tends to lose data unless it's recharged every couple of milliseconds, and this recharging tends to slow down the performance of DRAM compared to speedier RAM types.

18) Parallel Systems?
Ans) Multiprocessor systems with more than one CPU in close communication.
of parallel system:
->Increased *throughput*
->Economical
->Increased reliability
->graceful degradation
->fail-soft systems
If two processes which shares same system memory and system clock in a distributed system it is called parallel systems

19) What is the state of the processor, when a process is waiting for some event to occur?
Ans) Waiting state

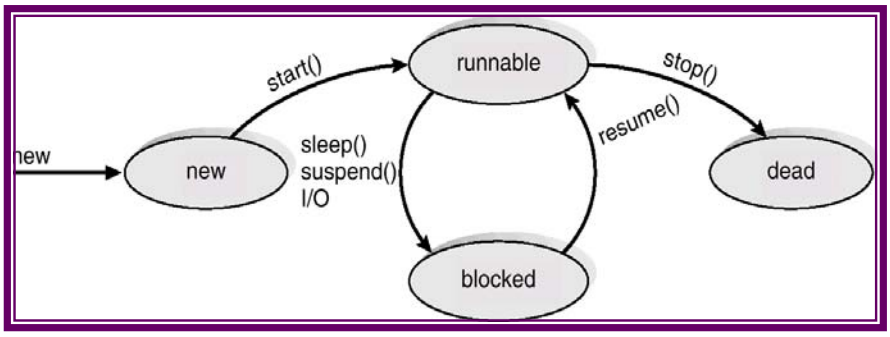
20) Distributed Systems?
Ans) Distribute the computation among several physical processors.
Loosely coupled system – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines
Advantages of distributed systems:
->Resources Sharing
->Computation speed up – load sharing
->Reliability
->Communications

21) Common Functions of Interrupts?
Ans)
->Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
->Interrupt architecture must save the address of the interrupted instruction.
->Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
->A trap is a software-generated interrupt caused either by an error or a user request.
->An operating system is interrupt driven.

22) Difference between Primary storage and secondary storage?
Ans)
Main memory: – only large storage media that the CPU can access directly.
Secondary storage: – extension of main memory that provides large nonvolatile storage capacity.

23) While running DOS on a PC, which command would be used to duplicate the entire diskette?
Ans) diskcopy

24) **Java Thread States**



25) What is CPU Scheduler?
Ans)
->Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
->CPU scheduling decisions may take place when a process:
1.Switches from running to waiting state.
2.Switches from running to ready state.
3.Switches from waiting to ready.
4.Terminates.
->Scheduling under 1 and 4 is *nonpreemptive*.
->All other scheduling is *preemptive.*

26) What do you mean by deadlock?
Ans) Deadlock is a situation where a group of processes are all blocked and none of them can become unblocked until one of the other becomes unblocked.
The simplest deadlock is two processes each of which is waiting for a message from the other.

27) What is Dispatcher?
Ans)
->Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
1) Switching context
2) Switching to user mode
3) Jumping to the proper location in the user program to restart that program

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

28) What is Throughput, Turnaround time, waiting time and Response time?
Ans)
1) Throughput – number of processes that complete their execution per time unit
2) Turnaround time – amount of time to execute a particular process
3) Waiting time – amount of time a process has been waiting in the ready queue
4) Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

29) Explain the difference between microkernel and macro kernel?
Ans)
Micro-Kernel: A micro-kernel is a minimal operating system that performs only the essential functions of an operating system. All other operating system functions are performed by system processes.

Monolithic: A monolithic operating system is one where all operating system code is in a single executable image and all operating system code runs in system mode.

30) Give an example of microkernel
Ans) Amoeba, WinNT, Minix

31) What is multi tasking, multi programming, multi threading?
Ans)
Multi programming: Multiprogramming is the technique of running several programs at a time using timesharing.
It allows a computer to do several things at the same time. Multiprogramming creates logical parallelism.

The concept of multiprogramming is that the operating system keeps several jobs in memory simultaneously. The operating system selects a job from the job pool and starts executing a job, when that job needs to wait for any i/o operations the CPU is switched to another job. So the main idea here is that the CPU is never idle.

Multi tasking: Multitasking is the logical extension of multiprogramming .The concept of multitasking is quite similar to multiprogramming but difference is that the switching between jobs occurs so frequently that the users can interact with each program while it is running. This concept is also known as time-sharing systems. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of time-shared system.

Multi threading: An application typically is implemented as a separate process with several threads of control. In some situations a single application may be required to perform several similar tasks for example a web server accepts client requests for web pages, images, sound, and so forth. A busy web server may have several of clients concurrently accessing it. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time. The amount of time that a client might have to wait for its request to be serviced could be enormous.

So it is efficient to have one process that contains multiple threads to serve the same purpose. This approach would multithread the web-server process, the server would create a separate thread that would listen for client requests when a request was made rather than creating another process it would create another thread to service the request.
So to get the advantages like responsiveness, Resource sharing economy and utilization of multiprocessor architectures multithreading concept can be used

32) Differentiate between RAM and ROM?
Ans)
Semiconductor memories are of two types: RAM (random access memory) and ROM (read only memory).
RAM is a read/write memory. Information can be written into and read from a RAM. It is volatile memory.
It stores information so long as power supply is on.

ROM is permanent type memory. Its contents are not lost when power supply goes off. the user cannot write into a ROM.Its contents are decided by the manufacturer and written at the time of manufacture.
Programmable ROMs are also available. They are called PROMs.

33) What resources are used when a thread created? How do they differ from those when a process is created?
Ans) When a thread is created the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.
Where as if a new process creation is very heavyweight because it always requires new address space to be created and even if they share the memory then the inter process communication is expensive when compared to the communication between the threads.

34) Describe the actions taken by thread library to context switch between user level threads?

Ans)

The thread library function performs the following actions to context switch between user level threads

a) Copy all live registers to Thread control Block (TCB)

b) Restore the state of the thread to run next i.e (copy the values of live registers from (TCB) to registers)

c) Move to the next thread to execute

35) Give a non-computer example of preemptive and non-preemptive scheduling.

Ans) Consider any system where people use some kind of resources and compete for them. The non-computer examples for preemptive scheduling the traffic on the single lane road if there is emergency or there is an ambulance on the road the other vehicles give path to the vehicles that are in need. The example for preemptive scheduling is people standing in queue for tickets.

36) Compare Linux credit based algorithm with other scheduling algorithms?

Ans) For the conventional time –shared processes, Linux uses a prioritized, credit-based algorithm. Each process possesses a certain number of scheduling credits; when a new task must be chosen to run, the process with most credits is selected. Every time that a timer interrupt occurs, the currently running process loses one credit; when its credits reaches zero, it is suspended and another process is chosen.

If no runnable processes have any credits, then Linux performs a recrediting operation, adding credits to every process in the system (rather than just to the runnable ones), according to the following rule:

Credits = credits/2 + priority

The above scheduling class is used for time-shared process and the in Linux for the real-time scheduling is simpler it uses scheduling classes: first come, first served (FCFS), and round-robin (RR) .In both cases, each process has a priority in addition to its scheduling class. In time-sharing scheduling, however, processes of different priorities can still compete with one another to some extent; in real-time scheduling, the scheduler always runs the process with the highest priority. Among processes of equal priority, it runs the process that has been waiting longest. The only difference between FCFS and RR scheduling is that FCFS processes continue to run until they either exit or block, whereas a round-robin process will be preempted after a while and will be moved to the end of the scheduling queue, so round-robin processes of equal priority will automatically time share among themselves.

Linux's real-time scheduling is soft-real time rather than hard-real time. The scheduler offers strict guarantees about the relative priorities of real-time processes, but the kernel does not offer any guarantees about how quickly a real-time process will be scheduled once that process becomes runnable.

Thus the Linux uses different scheduling classes for time-shared and real-time processes.

37) What is starvation and aging?

Ans)

Starvation: Starvation is a resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

Aging: Aging is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the request's priority as time passes and must ensure that a request will eventually be the highest priority request (after it has waited long enough)

38) Different types of Real-Time Scheduling?

Ans)

Hard real-time systems – required to complete a critical task within a guaranteed amount of time.

Soft real-time computing – requires that critical processes receive priority over less fortunate ones.

39) Condition for deadlock occurrence?

Ans)

Deadlock can arise if four conditions hold simultaneously.

Mutual exclusion: only one process at a time can use a resource.

Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes.

No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular wait: there exists a set {*P*0, *P*1, …, *P*0} of waiting processes such that *P*0 is waiting for a resource that is held by *P*1, *P*1 is waiting for a resource that is held by *P*2, …, *Pn*–1 is waiting for a resource that is held by *P*n, and *P*0 is waiting for a resource that is held by *P*0.

40) What are the Methods for Handling Deadlocks?

Ans)

->Ensure that the system will *never* enter a deadlock state.

->Allow the system to enter a deadlock state and then recover.

->Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including
UNIX.

41) What is a Safe State and its' use in deadlock avoidance?

Ans)

When a process requests an available resource, system must decide if immediate allocation leaves the system in a
s*afe state*

->System is in safe state if there exists a safe sequence of all processes.

->Sequence <*P*1, *P*2… *Pn*> is safe if for each *P*i, the resources that *Pi* can still request can be satisfied by
currently available resources + resources held by all the *Pj*, with *j*<*I*.
If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished.

When *Pj* is finished, *P*i can obtain needed resources, execute, return allocated resources, and terminate.

When *Pi* terminates, *Pi*+1 can obtain its needed resources, and so on.

->Deadlock Avoidance ⇒ ensure that a system will never enter an unsafe state.

42) Deadlock Detection-Algorithm Usage?

Ans)

->When, and how often, to invoke depends on:

How often a deadlock is likely to occur?

How many processes will need to be rolled back?

->If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be

able to tell which of the many deadlocked processes "caused" the deadlock.

43) Recovery from Deadlock?

Ans)

Process Termination:

->Abort all deadlocked processes.

->Abort one process at a time until the deadlock cycle is eliminated.

->In which order should we choose to abort?

1) Priority of the process.
2) How long process has computed, and how much longer to completion.
3) Resources the process has used.
4) Resources process needs to complete.
5) How many processes will need to be terminated?
6) Is process interactive or batch?

Resource Preemption:

->Selecting a victim – minimize cost.

->Rollback – return to some safe state, restart process for that state.

->Starvation – same process may always be picked as victim, include number of rollback in cost factor.

44) Difference between Logical and Physical Address Space?

Ans)

->The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.

Logical address – generated by the CPU; also referred to as *virtual address*.

Physical address – address seen by the memory unit.

->Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

45) **Binding of Instructions and Data to Memory?**

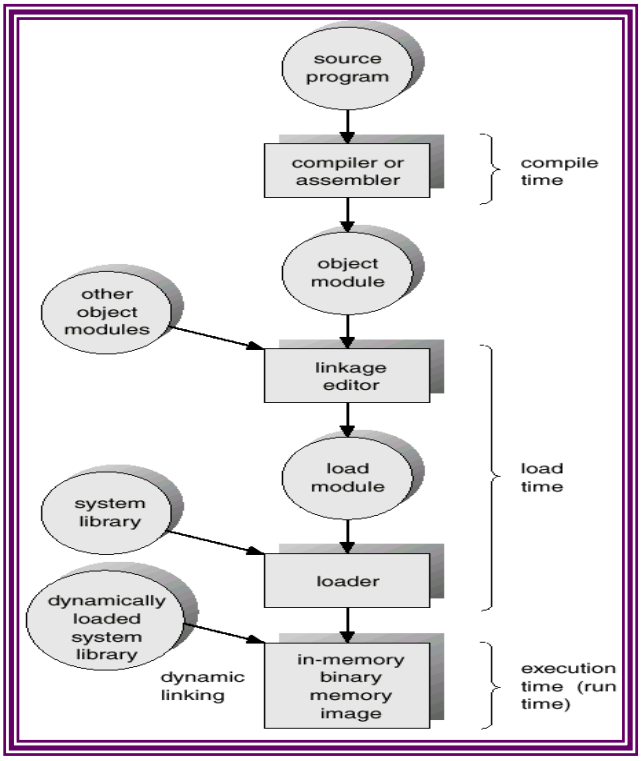
Ans)

Address binding of instructions and data to memory addresses can happen at three different stages

Compile time: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.

Load time: Must generate *relocatable* code if memory location is not known at compile time.
Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).

Multistep Processing of a User Program



46) What is Memory-Management Unit (MMU)?
Ans)
->Hardware device that maps virtual to physical address.
In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
->The user program deals with *logical* addresses; it never sees the *real* physical addresses.

47) What are Dynamic Loading, Dynamic Linking and Overlays?
Ans)
Dynamic Loading:
->Routine is not loaded until it is called
->Better memory-space utilization; unused routine is never loaded.
->Useful when large amounts of code are needed to handle infrequently occurring cases.
->No special support from the operating system is required implemented through program design.
Dynamic Linking:
->Linking postponed until execution time.
->Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
->Stub replaces itself with the address of the routine, and executes the routine.
->Operating system needed to check if routine is in processes' memory address.
->Dynamic linking is particularly useful for libraries.
Overlays:
->Keep in memory only those instructions and data that are needed at any given time.
->Needed when process is larger than amount of memory allocated to it.
->Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

48) what are the different Dynamic Storage-Allocation methods?
Ans)
How to satisfy a request of size *n* from a list of free holes?
First-fit: Allocate the *first* hole that is big enough.
Best-fit: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. It produces the smallest leftover hole.
Worst-fit: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

First-fit and best-fit are better than worst-fit in terms of speed and storage utilization.

49) What is fragmentation? Different types of fragmentation?
Ans) Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request.

External Fragmentation: External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used. If too much external fragmentation occurs, the amount of usable memory is drastically reduced.
Total memory space exists to satisfy a request, but it is not contiguous

Internal Fragmentation: Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks.
Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Reduce external fragmentation by compaction
->Shuffle memory contents to place all free memory together in one large block.
->Compaction is possible *only* if relocation is dynamic, and is done at execution time.

50) Define Demand Paging, Page fault interrupt, and Trashing?
Ans)
Demand Paging: Demand paging is the paging policy that a page is not read into memory until it is requested, that is, until there is a page fault on the page.

Page fault interrupt: A page fault interrupt occurs when a memory reference is made to a page that is not in memory.
The present bit in the page table entry will be found to be off by the virtual memory hardware and it will signal an interrupt.

Trashing: The problem of many page faults occurring in a short time, called "page thrashing,"

51) Explain Segmentation with paging?
Ans) Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment.This give some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

52) Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?
Ans) A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

53) What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?
Ans) Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

54) Why are page sizes always powers of 2?
Ans) Recall that paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

55) On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?
Ans) An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data—they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.