Q.2 a. Explain the scope of variables with an example. (6)

Answer:

In C we can define a variable in the block. The blocks are marked using { and }. braces. The blocks can be defined using the for statement. The scope of the variable is in the block in which it is declared, meaning that you can use that variable anywhere in the block. Even if we some block is declared in that block, we can use that variable when the variable is refered. in the block and if it can be resolved using two definitions, then the nearest definition has more precedence. So the variable is interpreted according to the nearest definition. Even if the Two definitions define two different data types for variables, they are accepted.

```
Program
#include <stdio.h>
main()
{.          \\Block 1
   int    i=10;  \\A
   {        \\Block 2.
     int  i=0;
     for (i=0; i<2; i++).
     {
            Printf ("value of i is %d\n", i);
     } // End of block 2
   }
   Printf ("the value of i is %d\n", i);
} \\ End of block 1.
```

Explanation 3M Program 3M.

b. Briefly explain dynamic memory allocation. (6)

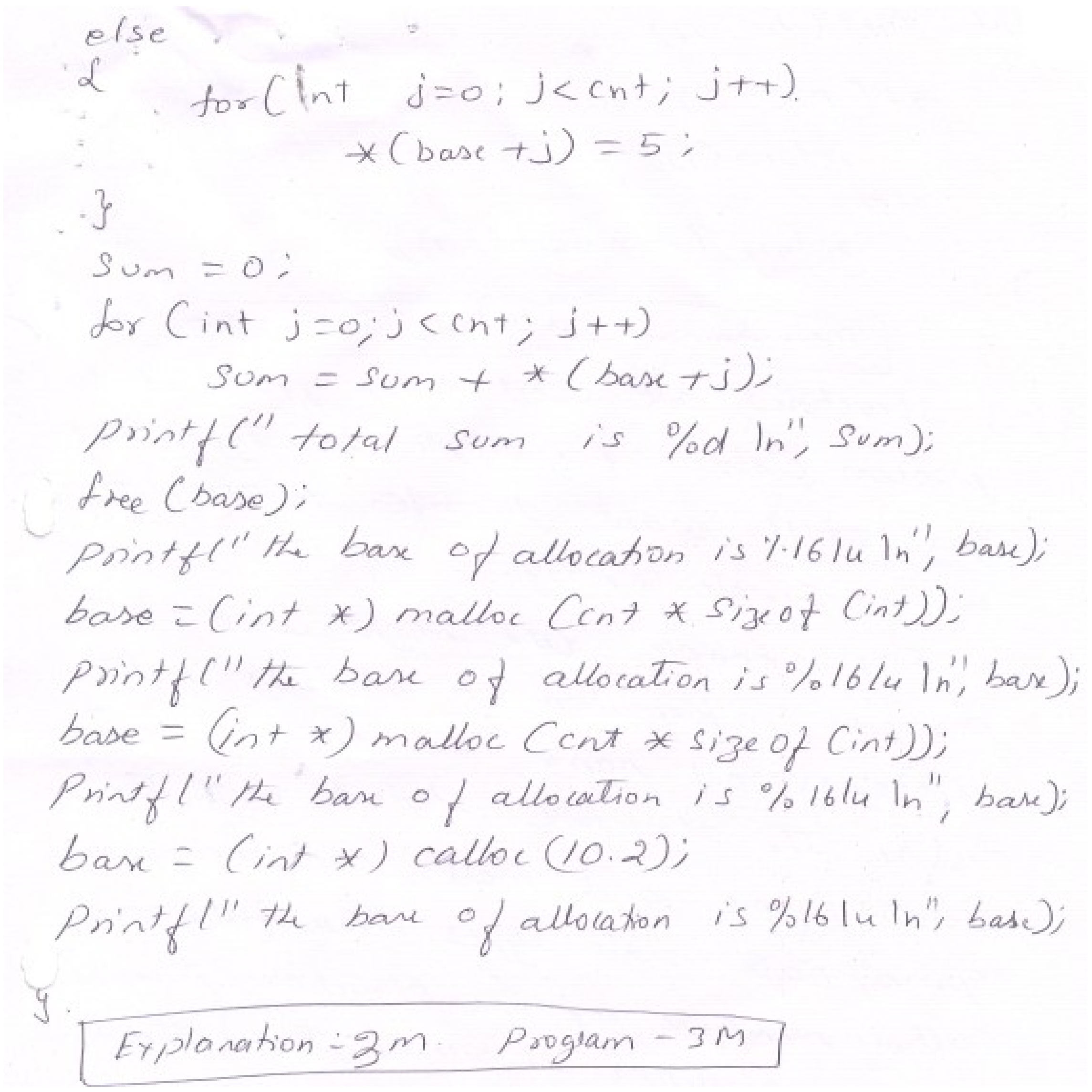
Answer:

Some times we may want to process the data but we don't Know what is the size of the data.

C has a facility of dynamic memory allocations. Using this we can allocate the memory for storage. The allocation is done at runtime. when our work is over, we can deallocate the memory. The allocation of memory is done using three functions: malloc, relloc & calloc. The functions return the pointers to void, so it can be typecast to any data type; thus making the functions generic. These functions take the input as the size of memory requirement.

<u>Example</u>

```
#include <stdio.h>
#include <malloc.h>
main ()
{
      int *base;   \\A
      int i, cnt=0, sum=0;
      Printf("how many integers you have to store\n");
      Scanf ("%d", &cnt);   \\B.
      base = (int *) malloc (cnt * size of (int)); \\C.
      Printf ("the base of allocation is %16lu
                  \n", base);  \\D
   if(!base) \\E
      Printf ("unable to allocate size \n");
```



```
else
{    for(int  j=0; j<cnt; j++)
            *(base+j) = 5;
}
Sum = 0;
for (int j=0;j<cnt; j++)
      sum = sum + *(base+j);
printf(" total  sum  is %d \n", sum);
free (base);
printf(" the base of allocation is %16lu \n", base);
base = (int *) malloc (cnt * sizeof (int));
printf(" the base of allocation is %16lu \n", base);
base = (int *) malloc (cnt * sizeof (int));
Printf(" the base of allocation is %16lu \n", base);
base = (int *) calloc (10.2);
Printf(" the base of allocation is %16lu \n", base);
}
```

Explanation - 3m.   Program - 3M

c. Write a recursive program to find the sum of all even numbers from 1 to n. (4)

Q.3 a. When do you use a structure? Define a structure data type called time_struct containing three members, integer hour, integer minute and integer second. Develop a programme that would assign values to the individual members and display the time in the following from: 16:40:51 (8)

Answer:

structures are used when you want to process data of multiple data types but you still want to refer to the data as a single entity.

For example. we might want to process information on students in the categories of name and marks. We can declare the structure 'student' with the fields "name" and marks and assign them appropriate data types. These fields are called members. of the structure. A member of the structure is referred to in the form of structurename. membername.

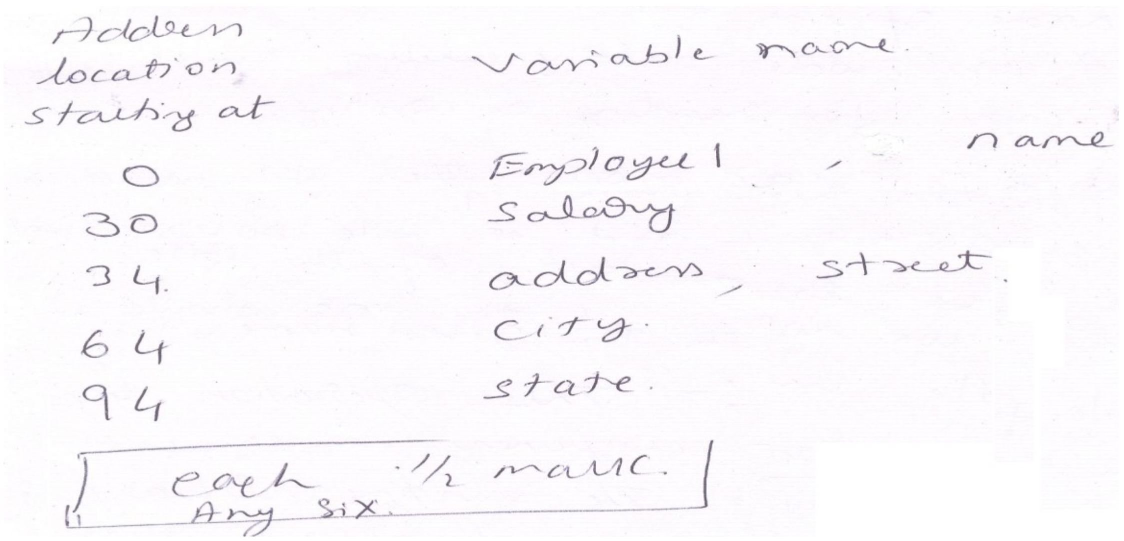
```
struct student
{
        char   name[30];
        float  marks;
} student1, student2;
main()
{
    struct student student3;
    char   s1[30];
    float  f;
    Scanf ("%s", name);
    Scanf ("%f", &f);
    student1. name   = s1;
    student2. marks  = f;
    Printf (" Name is  %s \n", student1. name);
    printf (" marks are %f \n", student2. marks);
}.
```

[Defination- 1m , Example with explanation - 5M]

b. Give the details of memory allocation to the following structure: (3)

```
Struct address
{
Street char [30];
City char [30];
State char [30];
}
Struct employee
{
name char [30];
salary float;
struct address adr 1;
}
Struct Empolyee Employee 1;
```

Answer:



Address location starting at	Variable name.
0	Employee1 , name
30	salary
34.	address, street.
64	city.
94	state.

each ½ mark.
Any six.

c. List out the different types of files and explain major operation that can be performed on them. (5)

Answer:

A file is a data object whose lifetime may be greater than the lifetime of a program responsible for creating it, because it is created on secondary storage device.

Types of files.

Sequential file.

~~D~~Direct - access file or indexed sequential file.

major operations on files are.

Open operation: when a file is to be used, it is first required to be opened. The open operation requires two operands

(1) name of the file

(2) Access mode telling whether the file is to be opened for reading or writing.

If access is read mode then file must exist.

If access is write mode. if the file exists, that file is emptied and the file position pointer is set to the starting of the file.

If file does not exist then OS should create the file with the given name.

Read operation :- This operation transfers the current file component to the designated program variable. The run time library of C provides a function fgetc(fp), where fp is a file descriptor, for fscanf(). fscanf() is similar to scanf(). except that fp extra parameter is required.

Write operation:- This operation transfers the contents of the designated program variable to the new components created at the current position. The runtime library of C provides a function fputc(c, fp), where fp is file descriptor, and c is a character to be written in the file fprintf().

close operation:- This operation notifies the operating system that the file can be detached from the program and that it can deallocate the internal storage used for the file. The file generally gets closed implicitly when the program terminates without explicit action by the programmer.

Types —1m.	
open operation 3m	read operation—1M.
write operation 1m.	close operation—1M.

Q.4 a. Write a program in C to implement bubble sort n integer numbers. (8)

Answer:

```
#include <stdio.h>
#define MAX 10
void swap (int *x, int *y)
{
      int temp;
      temp = *x;
      *x = *y;
      *y = temp;
}.
void bsort (int list[ ], int n)
{     int i, j;
for(i=0; i<(n-1); i++)
  for(j=0; j<(n-(i+1)); j++)
        if (list[j] > list[j+1])
            swap(&list[j], &list[j+1]);
}
void readlist(int list[], int n)
{
    int i;
    printf("Enter the elements \n");
    for(i=0; i<n; i++)
        scanf("%d", &list[i]);
}
void Printlist(int list[], int n)
{
    int i;
    Printf(" The elements of the list are: \n");
    for(i=0; i<n; i++)
        printf("%d\t", list[i]);
}
void main()
{
    int list[MAX], n;
    Printf("Enter The number of elements in
           The list max =10 \n");
    scanf(" %d", &n);
    readlist(list, n);
    printf(" The list before sorting is: \n");
    Printlist(list, n);
    bsort(list, n);
    Printf(" The list after sorting is: \n");
    printlist(list, n);
}
              [Complete Program - 8M]
```

b. Write a C program to implement binary search using single dimensional array. (8)

Answer:

```
#include <stdio.h>
#define MAX 10
void bsearch (int list[], int n, int element)
{
    int l,u,m, flag =0;
    l = 0; u = n-1;
    while (l <= u)
    {
        m = (l+u)/2;
        if (list[m] = = element)
        {
            printf(" The element whose value is %d is
                  present at position %d inlist\n",
                  element, m);
            flag =1;
            break;
        }
        else
               if (list[m] < element)
                      l =m+1;
               else
                      u =m-1;
    }
    if (flag = = 0)
        printf(" The element whose value is %d
               is not present in the list \n",
               element);
    }.
void readlist (int list[], int n)
{
    int i;
    printf(" Enter the elements \n");
    for (i=0; i<n; i++)
        scanf(" %d", & list [i]);
}.
```

```
void printlist(int list[], int n)
{   int i;
    printf("The elements of the list are: \n");
    for (i=0; i<n; i++)
        printf("%d\t", list[i]);
}
void main()
{
    int list[MAX], n, element;
    printf("Enter the number of elements in the
           list max =10\n");
    scanf("%d", &n);
    readlist(list, n);
    printf("\n The list before sorting is: \n");
    printf("\n old list \n");
    printlist(list, n.);
    printf("\n Enter the element to be Searched \n");
    Scanf("%d", &element);
    bSearch(list, n, element);
}.
```

[complete program – 8M]

Q.5 a. What is stack? Give array implementation of stack. (8)

Answer:

A stack is simply a list of elements with insertions and deletions permitted at one end.– called the stack top. It is possible to remove elements from a stack in reverse order from the insertion of elements into ~~the~~ stack. The stack is called LIFO.

Push & POP are the operations that are provided for insertion of an element into

The stack and the removal of an element from the stack,

C program to illustrate allay implementation of stack.

```
#include <stdio.h>
#define MAX 10 /*maximum size of stack 10*/
#include <stdlib.h>
void push (int stack[], int *top, int value)
{       if (*top < MAX)
        {  *top = *top +1;
           stack[*top] = value;
        }
        else
        {  printf("The stack is full can not push
                a value \n");
           exit(0);
        }
}
void pop (int stack[], int *top, int *value)
{       if (*top >= 0)
        {  *value = stack [*top];
           *top = *top -1;
        }
        else
        {  printf("The stack is empty can not pop
                a value \n");
           exit(0);
        }
}
```

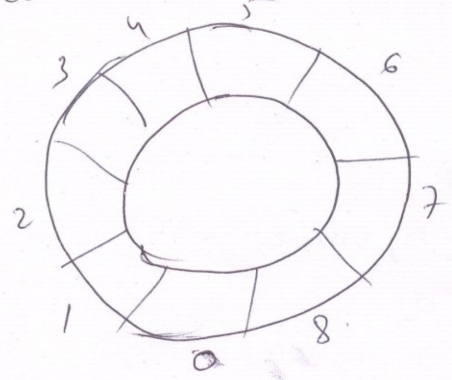
```
Void main()
{
    int stack[MAX];
    int top = -1;
    int n, value;
    do
    {
        do
        {   printf("Enter the element to be pushed\n");
            Scanf("%d", &value);
            Push(stack, &top, value);
            Printf("Enter 1 to continue\n");
            scanf("%d", &n);
        } while(n == 1);
        printf("Enter 1 to pop an element\n");
        Scanf(" %d ", &n);
        while(n == 1)
        {
            pop(stack, &top, &value);
            printf("The value poped is %d\n", value);
            printf("Enter 1 to pop an element\n");
            Scanf("%d", &n);
        }
        printf("Enter 1 to continue\n");
        Scanf(" %d ", &n);
    } while(n == 1);
}.
```

Explanation/difination-2M Program — 6M

b. Briefly explain a circular queue and write the C implementation of a circular queue using arrays. (8)

Answer:

In the regular queue insertion function gives a queue-full signal even if a considerable portion is free. This happens because the queue has a tendency to move to the right unless the "front" catches up with the 'rear' and both are reset to '0' again. To overcome this problem, the elements of the array are required to shift one position left whenever a deletion is made. But this will make the deletion process inefficient. Therefore, an efficient way of overcoming this problem is to consider the array to be circular.



C implementation.

```
#include <stdio.h>
#define MAX 10    /* maximum size of queue */
#include <stdlib.h>
Void insert (int queue[], int *rear, int front,
                int value)
{   *rear = (*rear + 1) % MAX;
```

```
  if (*rear == front)
  {
     printf("The queue is full can not insert a
              value \n");
     exit(0);
  }
  queue[*rear] = value;
}.
void  delete (int queue[], int *front, int rear,
              int  *value)
{
     if(*front == rear)
     {  printf("The queue is empty cannot delete a
              value \n");
        exit(0);
     }
     *front = (*front +1) %MAX;
     *value = queue[*front];
}
void main()
{
     int  queue[MAX];
     int  front, rear; n, value;
     front =0; rear =0;
     do{
        do{  printf("Enter the element to be inserted \n");
           Scanf("%d" , &value);
           insert (queue , &rear, front, value);
           printf("Enter 1 to continue \n");
           Scanf("%d", &n);
        }while (n == 1);
printf("Enter 1 to delete an element \n");
scanf("%d", &n);
while (n = =1)
{
    delete (queue, &front, rear, & value);
    printf(" The value deleted is %d \n", value);
    printf(" Enter 1 to delete an element \n");
    scanf(" %d", &n);
}
 printf(" Enter 1 to continue \n");
 scanf(" %d", &n);
} while (n = =1);
}.
```

Explanation – 3M Program – 5M

Q.6 a. Write a C program to delete a specific node from a singly linked list. (10)

Answer:

To delete a specific node in a singly linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node *delet (struct node *, int);
int length (struct node *);
struct node
{
    int data;
    struct node *link;
 };
 struct node *insert (struct node *p, int n)
 {  struct node *temp;
    if (p = = NULL)
    {
        p= (struct node *) malloc (size of (struct node));
        if (p = NULL)
        {   printf("Error\n");
            exit(0);
        }
        p -> data = n;
 P -> link = NULL;
}
else
{
    temp = P;
    while (temp -> link != NULL)
      temp = temp -> link;
     temp => link = (struct node *) malloc (size of (struct node));
    if(temp -> link == NULL)
    {
          printf(" Error \n");
          exit(0);

    }
     temp = temp -> link;
    temp => data = n;
   temp -> link = NULL;
  }
  return (P);

}.
void. Printlist (struct node *p)
{   printf(" The data values in the list are\n");
    while (P != NULL)
    {     printf("%d", P -> data);
          P = p -> link;
    }
 }
 void main()
 {     int n, x;
       Struct node *start = NULL;
       printf(" Enter the nodes to be created\n");
       Scanf("%d", &n);
```

```
start = delet (start, n);
printf(" The list after deletion is \n");
printlist (start);
}
struct node *delet( struct node *p, int node_no)
{   struct node  *prev, *curr;
    int i;
    if (p = = NULL)
    {   printf(" There is no node to be deleted \n");
    }
    else
    {       if( node_no > length (p))
            {   printf(" Error \n");
            }
            else
            {       prev = NULL;
                    curr = p;
                    i = 1;
                    while ( i < node_no)
                    {   prev = curr;
                        curr = curr -> link;
                        i = i+1;
                    }
                    if( prev = = NULL)
                    {   p = curr -> link;
                        free (curr);
                    }
                    else
                    {   prev -> link = curr -> link;
                        free (curr);
                    }
            }
    }
    return (p);
}
```

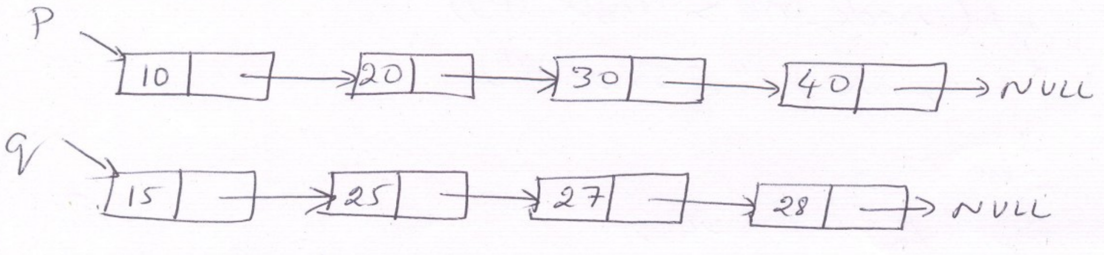
```
int length (struct node *p)
{
    int count =0;
    while (p != NULL)
    {
        count ++;
        p = p-> link;
    }
    return (count);
}
```

Complete program - 10M

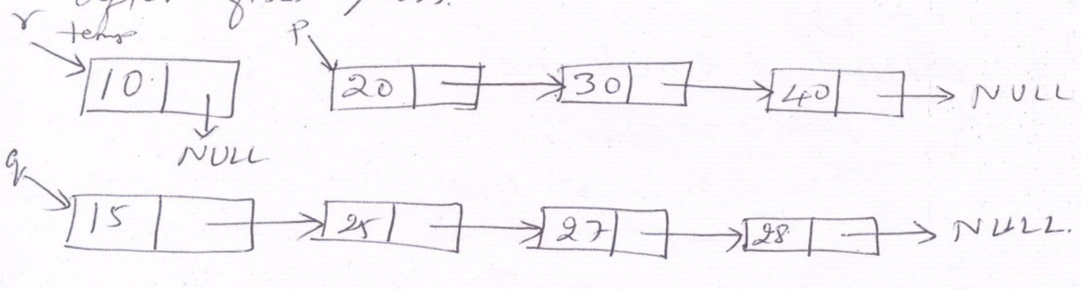
b. Explain how to merge two sorted singly linked lists. (6)

Answer:

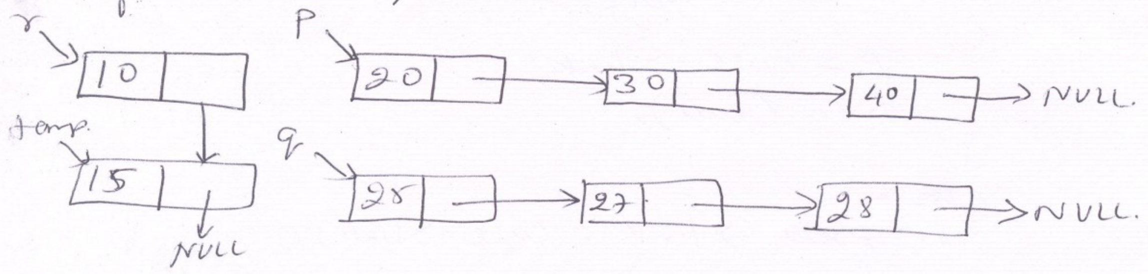
Consider the two sorted lists before merging



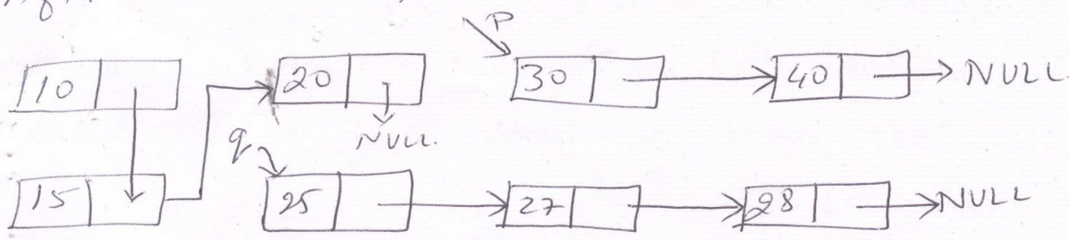
after first pass.



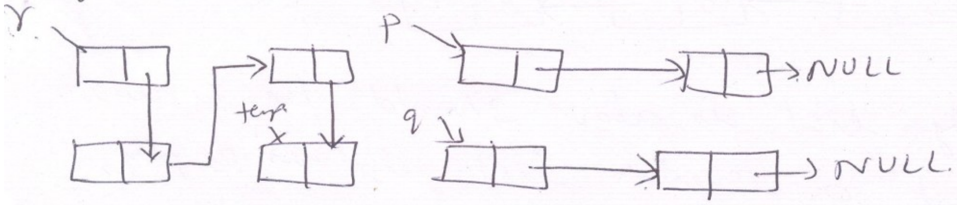
After second pass.



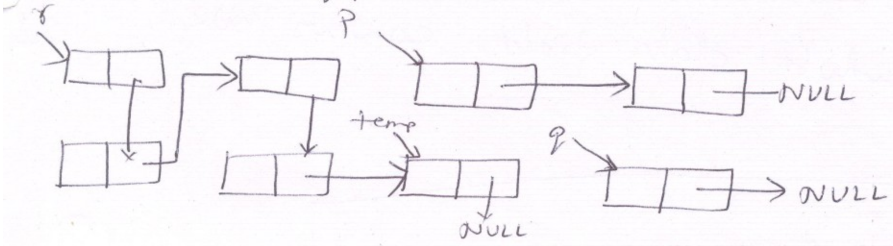
After the third pass



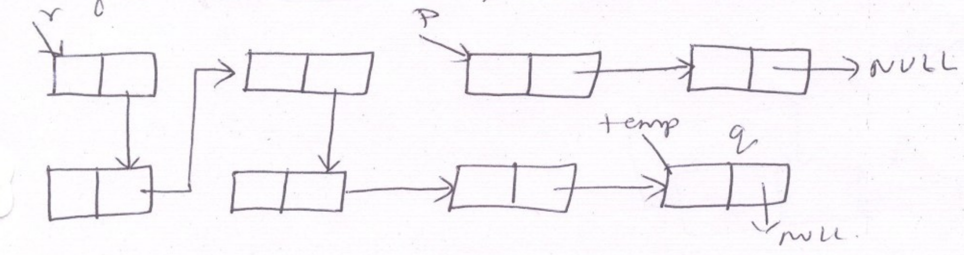
After the fourth pass.



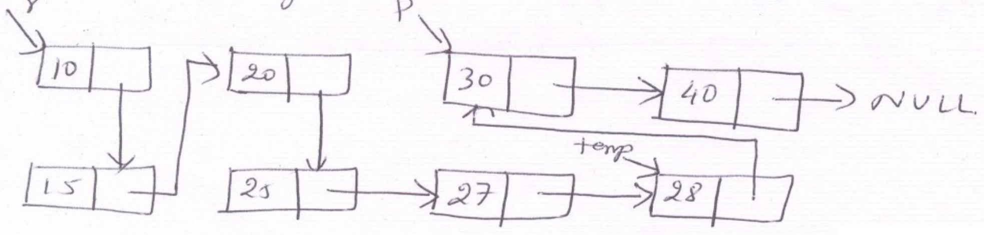
After the fifth pass



After the sixth pass



Final merged list.



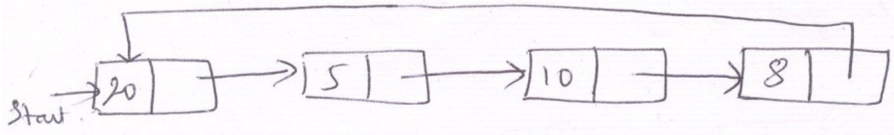
Each pass 1 mark

Q.7 a. With an example explain circular linked list. (4)

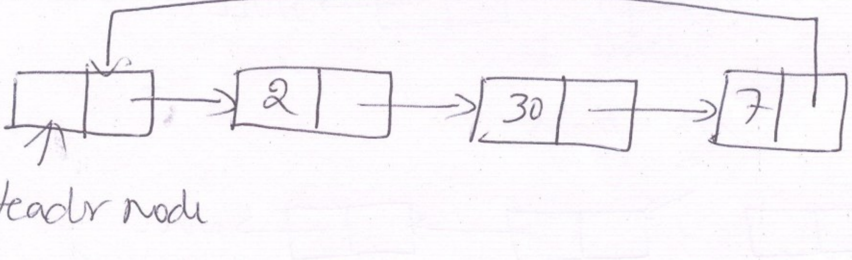
Answer:

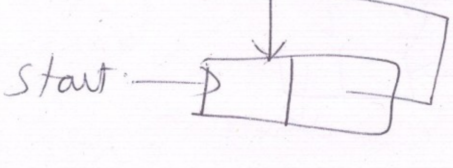
Circular Linked list

A circular list is a list in which the link field of the last node is made to point to the start/first node of the list.



The empty list is also should be circular. To represent a circular list that is empty, it is required to use a header node or a head node whose data field contents are irrelevant.





Defination 1M Explanation with example 3M.

b. Briefly explain the following: (12)
(i) Doubly linked list
(ii) How to insert a node in to a doubly linked list?

Answer: (i) Doubly linked list

Doubly linked list.

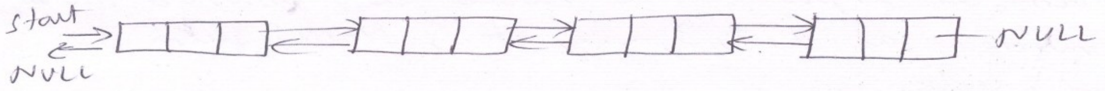
The problems of singly linked list can be overcome by adding one more link to each node which points to the previous node. when Such link is added to every node of a list, the corresponding linked list is called a doubly linked list. The two links are called left link & right link.

Left link of a node points to the previous node,

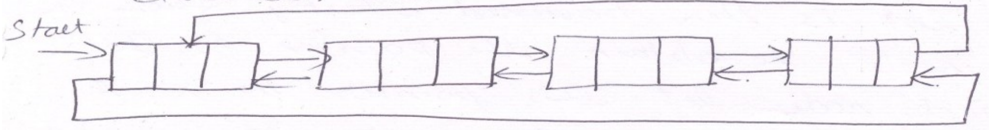
Right link points to the next node.

Doubly linked list can also be a chain or may be circular with or with out header node.

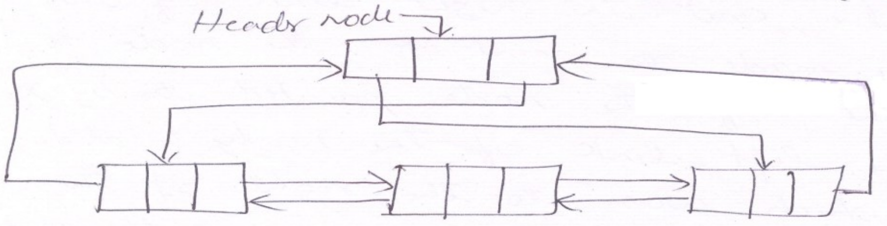
Chain.



circular list.



circular list with header node.



Doubly linked list is defined using a structure.

```
Struct dnode
{
    int data;
    struct dnode *left, *right;
};
```

(ii) How to insert a node in to a doubly linked list?

The following steps need to be followed to insert a node in a doubly linked list.

1. To insert a new node in a doubly linked chain, it is required to obtain a pointer to the node in the existing list after which a new node is to be inserted.

2. To obtain this pointer, the node number after which the new node is to be inserted is given as input. The nodes are assumed to be numbered as 1, 2, 3, ..., etc., starting from the first node.

3. The list is then traversed starting from the start node to obtain the pointer to the specified node. let this pointer be X. A new node is then created with the required data value, and the right link of this node is made to point to the node to the right of the node pointed to by X. And the left link of the newly created node which was to the right of the node pointed to by X is made to point to the newly created node. The right link of the node pointed to by X is made to point to the newly created node.

(i) Explanation 2M Pictorial representation 3M.
Structure - 1M.
(ii) each step 2m 2X3 = 6M.

Q.8 a. Define the following: **(4)**

(i) Tree
(ii) Degree of a node
(iii) Degree of a tree
(iv) Level of a node

Answer:

(a). Tree : A tree is a set of one or more nodes T such that:
(i) There is a specially designated node called root
(ii) The remaining nodes are partitioned into n. disjointed set of nodes T_1, T_2, ... T_n. each of which is a tree.

Degree of a Node: The degree of a node of a tree is the number. of subtrees having this node as a root.
(or)
The degree is the number of descendants of a node.

Degree of a Tree: The degree of a tree is defined as the maximum of degree of the nodes of the tree, that is, degree of tree = max (degree (node i) for I = 1 to n).

Level of a Node:
Level of a node is defined by taking the level of the root node as 1, and incrementing it by 1 as we move from the root towards the subtrees.

each defination 1M 1×4=4M

b. **Find the Preorder, Inorder and Postoder traversal sequences for the following trees: (6)**

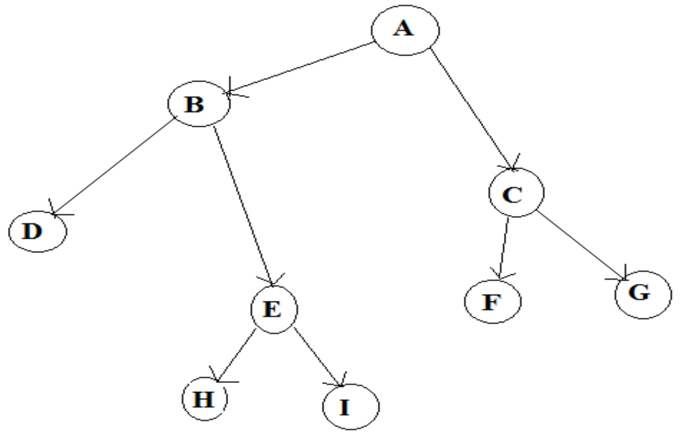


Fig.1

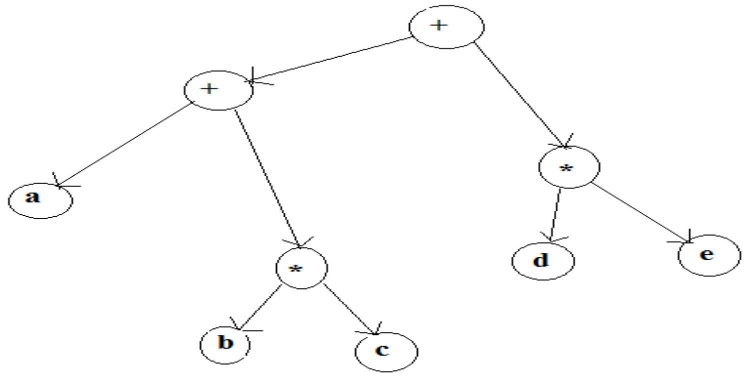


Fig.2

Answer:

(i) Inorder: DBHEIAFCG

Preorder: ABDEHICFG.

Postorder: DHIEBFGCA. $\textcircled{03}$

(ii) Inorder: a + b * c + d * e.

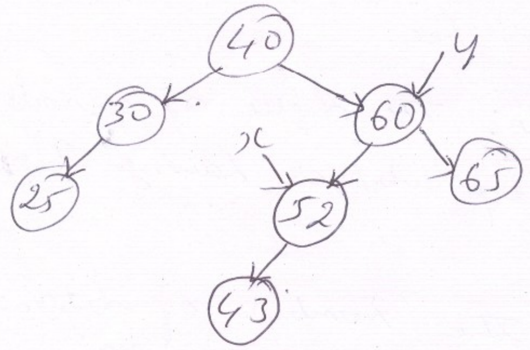
postorder: abc* + de* +.

pre order: + + a * bc *de. $\textcircled{03}$

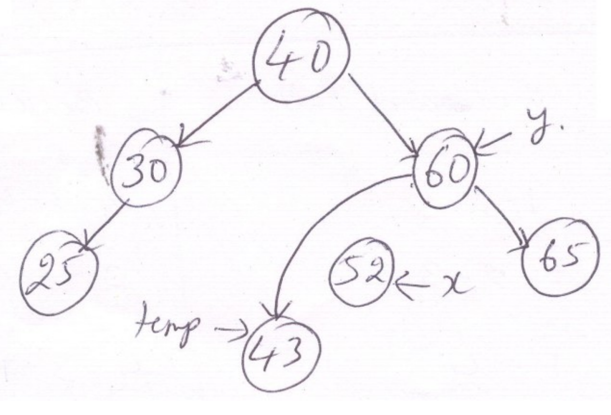
c. Explain the following operations with respect to a Binary Search Tree. (6)
(i) Deletion of a node with 2 children.
(ii) Deletion of a node with 1 child.

Answer:

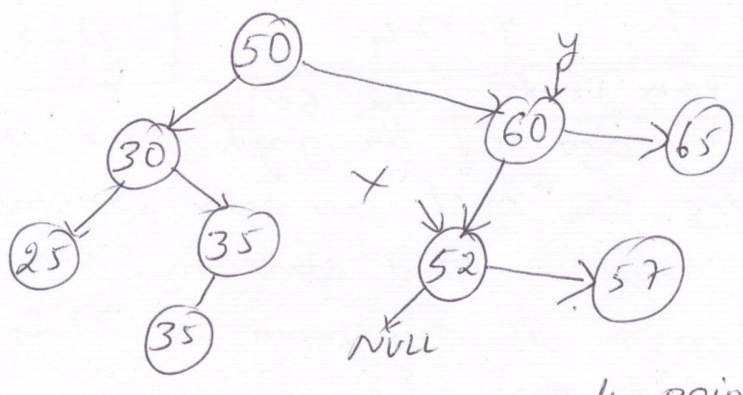
(i) dilition of node with 2 childrem.
consider the binary search tree.



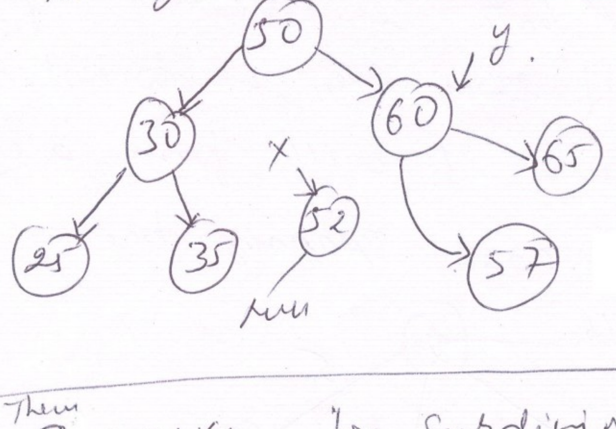
To delete a node pointed by x. y be a pointer to the node that is the root of the node pointed to by x. we store the pointer to the left child of the node pointed to by x in a temporary pointer temp. we then make the left child of the node pointed to by y the left child of the node pointed to by x. we then traverse the tree with the root as the node pointed to by temp to get its right leaf. and make the right child of this right leaf the right child of the node pointed to by x.



(ii) Deletion of a node with one child.
consider the binary search tree.



If we want to delete a node pointed to by x, we can do that by letting y be a pointer to the node that is the root of the node pointed to by x. make the left child of the node pointed to by y the right child of the node pointed to by x, and dispose of the node pointed to by x.



(i) ~~2~~ Three marks for subdivision (i)
(ii) ~~2~~ Three marks for subdivision (ii)

Q.9 a. For the following Fig.3 find the forward and backward path between every pair of vertices. Is the digraph strongly connected? (4)

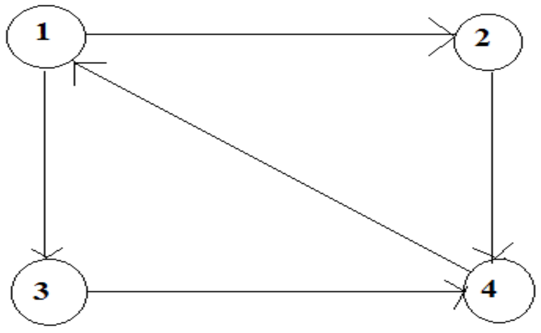
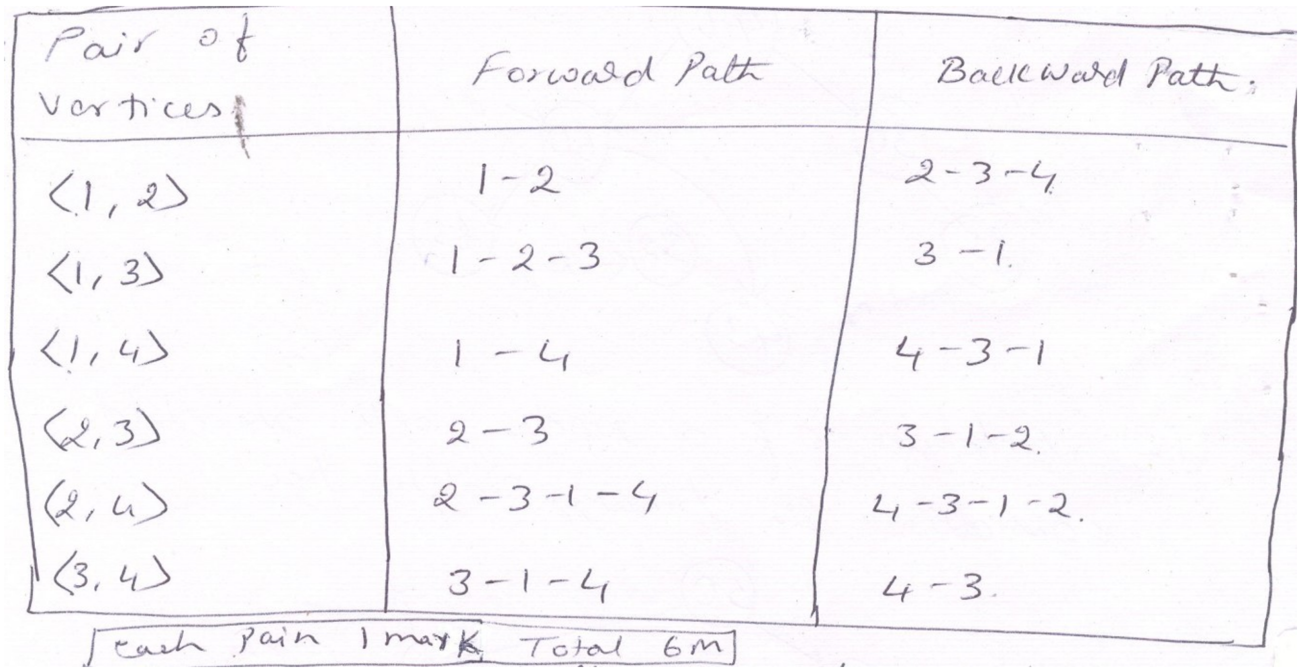


Fig.3

Answer:



Pair of Vertices	Forward Path	Backward Path
⟨1, 2⟩	1-2	2-3-4
⟨1, 3⟩	1-2-3	3-1.
⟨1, 4⟩	1-4	4-3-1
⟨2, 3⟩	2-3	3-1-2.
⟨2, 4⟩	2-3-1-4	4-3-1-2.
⟨3, 4⟩	3-1-4	4-3.

each pair 1 mark Total 6M

b. What is minimum-cost spanning tree of a graph? Compute the minimum cost spanning tree of the following graph using prim's Algorithm. (8)

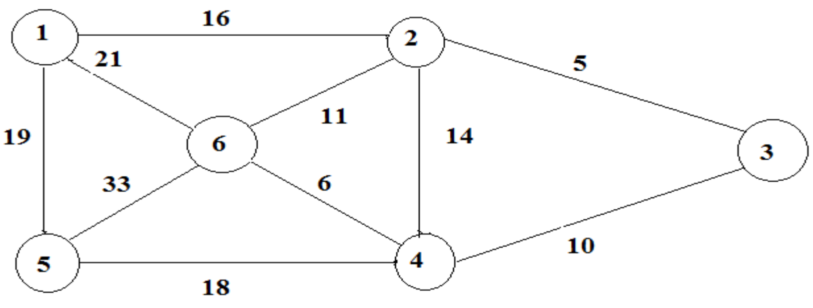
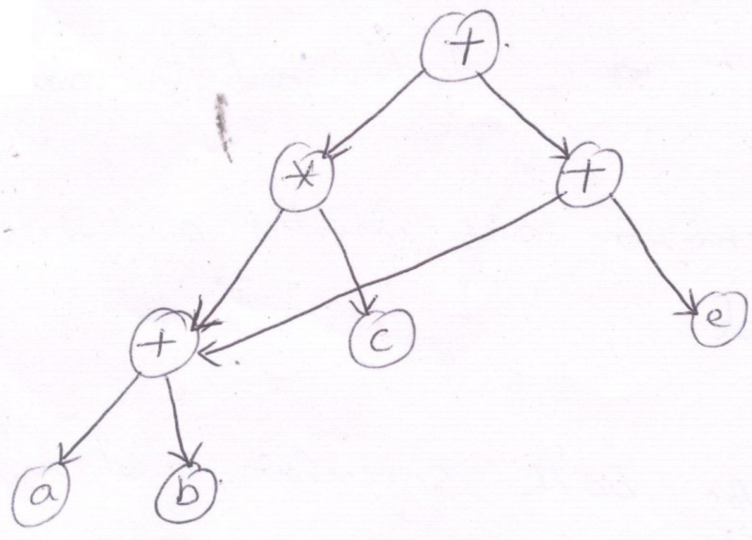


Fig.4

c. Draw the DAG representation for the following expression. (4)
(a+b) * c + ((a+b)+c)

Answer:



Complete DAG – 4M

<u>TEXT BOOK</u>

C & Data Structures, P.S. Deshpande and O.G. Kakde, Dreamtech Press, 2007