Big Job Hunter

First Edition

By

Niraj Agarwal

Dedicated to

- *To my parents*
- *To my Alma Mater*
- *To all industrious workers*

Acknowledgement

Even though the book cover bears only my name, essentially it reflects the efforts of many of my coworkers with whom I worked. My colleagues have contributed greatly to the quality of this book. I thank all of them from the core of my heart.

I would like to thank all of my college friends who spend their valuable time in reviewing the book and providing me valuable suggestions.

Special thanks should go to my brother, Suraj for his encouragement and immense help during the writing of this book.

- *Niraj Agarwal*

Preface

Dear Reader,

I have compiled quite a few good interview questions with appropriate explanantion, algorithm, and code.

This book is very much useful for the students of Engineering and Technology and will assist them in their preparation for campus interviews.

It all began during my Engineering days when I was preparing for my campus placements. Once I had a critical mass of written material, I thought of compiling it in the form of a book.

Please go through the complete book to have a sound understanding of how to attack various problems and please provide your valuable suggestions at niraj.ece05@gmail.com

Happy Reading !

- *Niraj Agarwal*

Interview Questions

Question 1> There are two sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above two arrays (i.e. array of length 2n). Full points are given for the *Solution* bearing efficiency O (log n) and partial points are given for the *Solution* bearing efficiency O (n).

Solution> Use the strategy where we will compare the medians of two arrays. This method works by first getting medians of the two sorted arrays and then comparing them. Let ar1 and ar2 be the input arrays.

- Calculate the medians m1 and m2 of the input arrays ar1[] and ar2[] respectively.
- If m1 and m2 both are equal then we are done and return m1 (or m2)
- If m1 is greater than m2, then median is present in one of the below two sub arrays.
    - From first element of ar1 to m1
    - From m2 to last element of ar2
- If m2 is greater than m1, then median is present in one of the below two sub arrays.
    - From m1 to last element of ar1
    - From first element of ar2 to m2
- Repeat the above process until size of both the sub arrays becomes 2.
- If size of the two arrays is two then use below formula to get the median.

  Median = {max (ar1 [0], ar2 [0]) + min (ar1 [1], ar2 [1])} / 2

Example:

ar1 [] = {1, 12, 15, 26, 38}

ar2 [] = {2, 13, 17, 30, 45}

For above two arrays m1 = 15 and m2 = 17

For the above two arrays, m1 is smaller than m2. So median is present in one of the following two sub arrays - [15, 26, 38] and [2, 13, 17]

Let us repeat the process for above two sub arrays:

m1 = 26 m2 = 13.

m1 is greater than m2. So the sub arrays become [15, 26] and [13, 17]

Now size is 2, so median = (max (ar1 [0], ar2 [0]) + min (ar1 [1], ar2 [1]))/2

= (max (15, 13) + min (26, 17))/2

= (15 + 17)/2

= 16

Implementation:

```
#include<stdio.h>
int max(int, int); /* to get maximum of two integers */
int min(int, int); /* to get minimum of two integeres */
int median(int [], int); /* to get median of a single array */
 /* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
```

```
  Both ar1 [] and ar2 [] are sorted arrays
Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
 int m1; /* For median of ar1 */
 int m2; /* For median of ar2 */
 /* return -1  for invalid input */
 if(n <= 0)
   return -1;
 if(n == 1)
   return (ar1[0] + ar2[0])/2;
 if (n == 2)
   return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;
 m1 = median(ar1, n); /* get the median of the first array */
 m2 = median(ar2, n); /* get the median of the second array */
 /* If medians are equal then return either m1 or m2 */
 if(m1 == m2)
   return m1;
  /* if m1 < m2 then median must exist in ar1[m1 ....] and ar2[.... m2] */
 if (m1 < m2)
   return getMedian(ar1 + n/2, ar2, n - n/2);
  /* if m1 > m2 then median must exist in ar1[.... m1] and ar2[m2 ...] */
 return getMedian(ar2 + n/2, ar1, n - n/2);
}
 /* Driver program to test above function */
int main()
{
 int ar1[] = {1, 12, 15, 26, 38};
 int ar2[] = {2, 13, 17, 30, 45};
 printf("%d ", getMedian(ar1, ar2, 5));
 return 0;
}
 /* Utility functions */
int max(int x, int y)
{
   return x > y? x : y;
}
int min(int x, int y)
{
   return x > y? y : x;
}
```

```
/* Function to get median of a single array */
int median(int arr[], int n)
{
  if(n%2 == 0)
    return (arr[n/2] + arr[n/2-1])/2;
  else
    return arr[n/2];
}
```

Time Complexity: O (logn)

Algorithmic Paradigm: Divide and Conquer

Question 2> Write a program to find the intersection and union of two linked lists.

Solution>

Intersection of two lists:

Initialize result list to NULL. Traverse first list and for each element that is also present in the second list, insert that element into the result.

Union of two lists:

Initialize result list as NULL. Traverse first list and push all of its elements in result. Now start traversing the second list and if an element of the second list is already present in result then do not insert it otherwise insert it.

```
#include<stdio.h>
/* Link list node */
struct node
{
    int data;
    struct node* next;
};
/* Function to insert a node in a linked list*/
void push(struct node** head_ref, int new_data)
{
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}
/* Function to print a linked list*/
void printList(struct node *node)
```

```
{
  while(node != NULL)
  {
    printf("%d ", node->data);
    node = node->next;
  }
}
/* Function returns 1 if data is present in linked list
  else return 0 */
int isPresent(struct node *head, int data)
{
 struct node *t = head;
 while(t != NULL)
 {
  if(t->data == data)
    return 1;
  t = t->next;
 }
 return 0;
}
/* Function to get union of two linked lists head1 and head2 */
struct node *getUnion(struct node *head1, struct node *head2)
{
 struct node *result = NULL;
 struct node *t1 = head1, *t2 = head2;
 while(t1 != NULL)
 {
  push(&result, t1->data);
  t1 = t1->next;
 }
 while(t2 != NULL)
 {
  if(!isPresent(result, t2->data))
    push(&result, t2->data);
  t2 = t2->next;
 }
 return result;
}
/* Function to get intersection of two linked lists head1 and head2 */
struct node *getIntersection(struct node *head1, struct node *head2)
{
```

```
  struct node *result = NULL;
  struct node *t1 = head1;
  while(t1 != NULL)
  {
    if(isPresent(head2, t1->data))
       push(&result, t1->data);
    t1 = t1->next;
  }
  return result;
}
/* Drier program to test above function*/
int main()
{
    struct node* head1 = NULL;
    struct node* head2 = NULL;
    struct node* intersecn = NULL;
    struct node* unin = NULL;
    /*create a linked lits 10->15->5->20 */
    push(&head1, 20);
    push(&head1, 4);
    push(&head1, 15);
    push(&head1, 10);
    /*create a linked lits 8->4->2->10 */
    push(&head2, 10);
    push(&head2, 2);
    push(&head2, 4);
    push(&head2, 8);
    intersecn = getIntersection(head1, head2);
    unin = getUnion(head1, head2);
    printf("\n First list is \n");
    printList(head1);
    printf("\n Second list is \n");
    printList(head2);
    printf("\n Intersection list is \n");
    printList(intersecn);
    printf("\n Union list is \n");
    printList(unin);
    getchar();
}
```

Question 3> Write a function that takes three integers corresponding to the lengths of sides and returns what kind of triangle can be made out of those three sides.

Solution>
1) Let the three integers be a, b and c
2) If not (a+b > c and a+c > b and b+c > a) then input is not valid and triangle can not be formed
3) if a = b = c then return equilateral
4) If (a = b or a = c or b = c) then return isosceles
5) Arrange variables such that a > b > c and if sqr(a) = srq(b) + sqr(c) then return right angle triangle
6) Else scalene triangle

```
#include<stdio.h>
int main()
{
int a,b,c;
printf("Enter sides");
scanf("%d %d %d",&a, &b, &c);
if !( (a+b)>c || (b+c)>a || (c+a)>b )
 {
    printf("Invalid Input");
        return 0;
 }
if(a==b==c)
 { printf("Equilateral"); return 0; }
if(a==b || b==c || c==a)
 { printf("Isosceles"); return 0; }
// Arrange  a,b,c in ascending order so that a>b>c
if(a*a == b*b + c*c)
 { printf("Right angled triangle); return 0; }
else
 printf("Scalene Trinagle");
 return 0;
} // end of main
```

Question 4> There are 25 horses and only five racing tracks in a race course. How do you find the second coming horse out of all the 25 horses, provided there is no stop clock in minimum number of races ?

Solution> Divide the set of 25 horses into 5 non-overlapping sets of 5 horses each. Have a race each for all the horses in each set. This makes it a total of 5 races, one for each set. Now, have a race for the winners of each of

the previous 5 races. This makes it a total of 6 races. Observe the position of each horse in the 6th race and correspondingly number the sets i.e. the set of the winner of 6th race will be said to be set no. 1 while the set of the loser of the 6th race will be said to be set number 5. Now, possible candidates for the first three positions exclude the following:

1. All horse from set 4 or set 5.
2. All horse except the winner from set 3.
3. All horse except the winner and the 2nd position from set 2.
4. All horse except the winner, 2nd position and 3rd position from set 1.

So now we have 6 candidates for top 3 positions. However, we know that the winner of set 1 is the fastest horse in the whole group of 25 sets.
So now we have 5 candidates for the second and third position. Have a race of these 5 horses and this will solve our problem in just 7 races.

Question 5> A randomly filled array of 2N numbers (N odd and N even) is given . Arrange the array in such a way that odd numbers are at even indexes and even numbers are at odd indexes.

Solution> There are number of possible *Solution*s for it but you need to give the best possible *Solution*.
Best possible is O(n) time and O(1) space complexity

Algorithm:

1. Keep two pointers one at 0th index (even) and other at 1st index(odd) of the array.
2. move both pointer towards other end by 2 position at a time.
3. Keep checking if first pointer encounters an index where even number is at even place. If that is the case, then stop first pointer there and if 2nd pointer encounters an odd number at an odd index then stop 2nd pointer; Otherwise keep moving both of them until one of them reaches to the other end.
4. If both have stopped at 3rd step then swap numbers at the index pointed by two pointers and go to step 2.
5. END

Since we require only one pass over the array hence it is O(n) algorithm without using any extra space.

```
#include<stdio.h>
int main()
{
int a[20],p,q,t,n,i;
printf("Enter n \n ");
scanf("%d", &n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
p=0;
```

```
q=1;
while (p<n && q<n)


      while(a[p]%2!=0)
      p=p+2;
      while(a[q]%2==0)
      q=q+2;
      t=a[p];
      a[p]=a[q];
      a[q]=t;
}
printf("Output\n");
for(i=0;i<n;i++)
printf("%d  ",a[i]);
return 0;
}
```

Question 6> Predict output of the following code snippet

```
#include<stdio.h>
void f(int n);
int main()
{
f(5);
return 0;
}
void f(int n)
{
      if(n>0)
      {
         f(--n);
         printf("%d ",n);
         f(--n);
      }
}
```

Solution> 0 1 2 0 3 0 1 4 0 1 2 0

Question 7> Implement the following function:- FindSortedArrayRotation, which takes as its input an array of unique integers that has been first sorted in ascending order, then rotated by an unknown amount X where 0 <= X <= (arrayLength - 1). An array rotation by amount X moves every element array[i] to array[(i + X) % arrayLength]. The function FindSortedArrayRotation discovers and returns X by examining the array. Consider performance, memory utilization and code clarity and elegance of the *Solution* when implementing the function.

Solution>

```
#include<stdio.h>
#include<stdlib.h>
int FindSortedArrayRotationIterative(int *array, int length)
{
    int start = 0;
    int end = length - 1;
    if(array[start] < array [end])
      return start;
    while(start != end)
    {
    int middle = (start + end)/2;
    if(array[start] < array[middle])
      start = middle;
    else
      end = middle;
    }
    return (start + 1);
}
  int main()
  {
    int array[] = { 4,6,8,10,2};
    int y = FindSortedArrayRotationIterative(array, 5);
    printf("%d\n", y);
  }
```

Question 8> How to detect a loop Linked Lists ? Write appropriate code.

Solution> One way to detect the cycle is to use two pointers - one that advances one step at a time and one that advances two steps at a time. If either pointer runs into a NULL node then we can say that the list does not have a cycle. As the faster pointer steps ahead two nodes at a time it will eventually overtake the slower pointer. We can use this fact to detect the loop and break out of the infinite loop. This technique is known as Floyd's cycle

finding algorithm or the Tortoise and Hare algorithm. It has linear run time complexity and constant space complexity.

Code :

```
#include <stdio.h>
struct node
{
    int data;
    node* next;
};
void push(node** head, int data)
{
  struct node* newNode = (struct node *) malloc sizeof(struct node);
  newNode->data = data;
  newNode->next = *head;
  *head = newNode;
}
bool detectCycle(node* head)
{
    node* fast = head;
    node* slow = head;
    do {
            if (fast->next == NULL || fast->next->next == NULL)
            return false;
            fast = fast->next->next;
            slow = slow->next;
      } while(fast != slow);
     return true;
}
 int main(){
    node* head= NULL;
    for(int i=7; i>=1;i--){
    push(&head, i);
    }
    /* Manually creating a loop */
    node* last = head;
    while(last->next != NULL){
    last = last->next;
    }
    last->next = head;
    bool hasCycle = detectCycle(head);
```

```
printf("hasCycle= %d\n",hasCycle);
}
```

To find the size of the loop :
Now at the point where loop is detected start moving fast pointer one node at a time and begin a counter while keeping the slow ponter at the same position. When they meet again this counter will return the size of loop.

To find the starting point of the loop :
Reset both pointers to the start of the linked list. Increment one pointer by the size of the loop and then start the second pointer. Now increment both pointers one step at a time and when they meet again, it'll be the starting point of the loop.

eg 2 -> 5 -> 10 -> 12 -> 14 -> 16 -> 20 --- // 20 points back to 12. So loop size = 4
*----------

Now make pointers P1 and P2 point to 2. Incremet P2 by loop size so that P2 now points to 14. P1 is currently pointing to 2. Now start incrementing both P1 and P2 by 1 unit until they meet. Whereever P1 and P2 meet, it is the starting point of the loop.

Write a Program to remove loop from linked list.
1. Find out how many nodes are there in the the loop, say k. Then find starting point of the loop .
2. Move a pointer till its next node is not the loop's starting node. It's the loop ending node.
3. Set the next node of the loop ending node to NULL to remove the loop.
4. Print the list.

Question 9> Prepare all possible sets from a given array with minimum time complexity.
S1 = 1,2,3,4,5
sets = {}, {1}, {2}, {3}, {4} ... {1,2}, {1,3}, ... {1,2,3,4,5}

Solution>

```
#include<stdio.h>
int main ()
{
    char set [] = {'a','b'};
    int setSize = 2; /*As per above array*/
    int totalSubset = 1<<2;
```

```
int i,j,k;
printf ("{{}");
for (i = 1; i < totalSubset; i++)
{
    printf("{");
    k=i;
    j = 0;
    while(k)
    {
        if (k&1)
        {
            printf ("%c", set[j]);
        }
        j++;
        k = k>>1;
    }
    printf ("}");
        }
printf ("}\n");
        return 0;
}
```

Question 10> Given the value of two nodes in a binary search tree, find the lowest (nearest) common ancestor. You may assume that both values already exist in the tree.

Solution>

Algorithm :

Examine the current node(i.e root node). If value1 and value2 are both less than the current node then examine the left child. If value1 and value2 are both greater than the current node then examine the right child, Otherwise the current node is the lowest common ancestor.

```
Node findLowestCommonAncestor(Node root, int value1, int value2)
{
 while( root!=null)
 {
  int value = root.getValue();
  if (value > value1 && value > value2)
```

```
   root = root.getLeft();
  else if (value < value1 && value < value2)
   root = root.getright();
  else
   return root;
 }
}
```

Question 11> Write a program to perform Binary Search opeeration in 2D array

Solution>

```
#include<stdio.h>
int BinarySearch(int a[],int col,int val)
{
  int min = 0, max = col-1, mid;
  mid=(min+max)/2;
  for(;max>=min;)
  {
    if(a[mid]==val)
      return mid;
    else if(a[mid]<val)
      min = mid + 1;
    else
      max = mid - 1;
    mid = (min + max) / 2;
  }
  return -1; //Element not found
}
int main()
{
int i, j, val, re = -1, flag = 0;
int a[2][4] = { {-1,4,5,6},{2,5,7,9} };
printf("\nEnter the number you want to find\n");
scanf("%d",&val);
for(i=0;i<2;i++)
{
    re = BinarySearch(&a[i][0],4,val);
    if(re != -1)
         { flag = 1; break; }
```

```
}
if(flag)
 {
   printf("\n Val is found at row = %d  and col = %d ",(i+1),(re+1));
 }
else
 {
  printf("\n Number is not found");
 }
return 0;
}
```

Question 12> Give an oneline C expression to test whether a number is a power of 2?

Solution>

```
#include<stdio.h>
int main()
{
int x;
printf(" Enter x ");
scanf("%d",&x);
if (!(x & (x-1)))  // false if x is a power of 2
  printf(" \n Power of 2 \n");
else
  printf("\n Not power of 2 \n ");
return 0;
}
```

Question 13> Give a fastest way to multiply a number by 7.

Solution>

Multiply by 8 (left shift by 3 bits) and then subtract the number. (x << 3) - x

---

Question 14> Write a code for extracting unique elements from a sorted list of array.

e.g. Input Array = (1, 1, 3, 3, 3, 5, 5, 5, 9, 9, 9, 9)

Output Array = (1, 3, 5, 9)

Solution>

```
#include<stdio.h>
int getarray(int p[100], int n)
{
  int c, i = 1;
  for (c = 1; c < n;)
    if (p[c] != p[i-1])
    {
      p[i] = p[c];
      c++;
      i++;
    }
    else
     { c++; }
  return i;
}
int main()
{
  int a[100], n, i, j;
  printf("Enter value of n");
  scanf("%d",&n);
  printf("\nEnter elements of the array\n");
  for(i = 0 ; i < n ; i++)
    scanf("%d",&a[i]);
  j = getarray(a, n);
  printf("New array is\n");
  for(i = 0 ; i < j ; i++)
  printf("\t%d", a[i]);
  return 0;
}
```

---

*Question*n 15> Give a method to count the number of ones in a 16 bit number.

Solution>

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

```
 int n;
 printf("Enter n : ");
 scanf("%d",&n);
 int count = iterativecount(n);
 printf("\n Answer = %d ",count);
 return 0;
}
int iterativecount(int n)
{
 int count = 0;
 while(n)
 {
  count += n & 1 ;
  n >>= 1;
 }
 return count;
}
```

Question 16> Write a recursive code to reverse a linked list.

Solution>

```
void RecursiveReverse(struct node** headRef)
{
 struct node* first;
 struct node* rest;
 if (*headRef == NULL) return; // Base case : Empty list
 first = *headRef;   // suppose first = {1, 2, 3}
 rest = first->next; // rest = {2, 3}
 if (rest == NULL) return;
 RecursiveReverse(&rest);   // Recursively reverse the smaller {2, 3} case
 first->next->next = first; // put the first element on the end of the list
 first->next = NULL;
 *headRef = rest;
}
```

Question 17> Write a code for printing a 2 dimensional matrix in spiral form.

Input Array

---
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20

Output : 1, 2, 3, 4, 8, 12, 16, 20, 19, 18, 17, 13, 9, 5, 6, 7, 11, 15, 14, 10

Solution>
Code :

```
#include <stdio.h>
void print_layer_top_right(int a[][4], int x1, int y1, int x2, int y2);
void print_layer_bottom_left(int a[][4], int x1, int y1, int x2, int y2);
int main(void)
{
     int a[5][4] = {
                  {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12},
                  {13, 14, 15, 16},
                  {17, 18, 19, 20}
                 };

     print_layer_top_right(a,0,0,3,4);
}
// prints the top and right shells of the matrix
void print_layer_top_right(int a[][4], int x1, int y1, int x2, int y2)
{
    int i = 0, j = 0;
    // print the row
    for(i = x1; i <= x2; i++)
    {
      printf("%d,", a[y1][i]);
    }
    // print the column
    for(j = y1 + 1; j <= y2; j++)
    {
      printf("%d,", a[j][x2]);
    }
```

```
    // see if we have more cells left
    if(x2-x1 > 0)
    {
       // 'recursively' call the function to print the bottom-left layer
       print_layer_bottom_left(a, x1, y1 + 1, x2-1, y2);
    }
}
// prints the bottom and left shells of the matrix
void print_layer_bottom_left(int a[][4], int x1, int y1, int x2, int y2)
{
     int i = 0, j = 0;
     // print the row of the matrix in reverse
     for(i = x2; i >= x1; i--)
     {
       printf("%d,", a[y2][i]);
     }
     // print the last column of the matrix in reverse
     for(j = y2 - 1; j >= y1; j--)
     {
       printf("%d,", a[j][x1]);
     }
     if(x2-x1 > 0)
     {
       // 'recursively' call the function to print the top-right layer
       print_layer_top_right(a, x1+1, y1, x2, y2-1);
     }
}
```

Question 18> Write a function to count the number of leaf nodes in a binary tree

Solution>

```
unsigned int getLeafCount(struct node* node)
{
  if(node == NULL)
    return 0;
  if(node->left == NULL && node->right==NULL)
    return 1;
  else
    return getLeafCount(node->left)+ getLeafCount(node->right);
```

```
}
```

Question 19> An array of characters that forms a sentence of words is given. Give an efficient algorithm to reverse the order of the words(not characters) in it.

Solution>

```
#include<stdio.h>
int main()
{
   char str[]="Rahul is good boy";      // Output should be boy good is Rahul
   int flag = 1; int i;
   char temp[50]="";
   int n = 0;
   while(str[n]!='\0') n++;
  for(i=n-1 ;i>=0 ; i=i-1)
  {
    if(str[i]==' ')
    {
       int j = 0;
       int m = i+1;
       while((str[m]!=' ') && (m<n))
       {
         temp[j]=str[m];
         j++;
         m++;
       }
       if(flag)
       {
                    temp[j]='\0'; j=0;
                          printf("%s ",temp);
       }
   } // end of if
 } // end of for loop

 // Now print last word
 for(i=0; str[i]!=' ';i++)
 temp[i]=str[i];
 temp[i]='\0';
 printf("%s",temp);
```

```
 return 0;
}
```

Question 20> Imagine you are standing in front of a mirror, facing it. Raise your left hand. Raise your right hand. Look at your reflection. When you raise your left hand your reflection raises what appears to be his right hand. But when you tilt your head up, your reflection does the sameand does not appear to tilt his/her head down. Why is it that the mirror appears to reverse left and right, but not up and down?

Solution> Left and right depend on the direction you face. Up never changes when you change direction.

Question 21> Write a function to wap the valur of two variable using XOR

Solution>

```
void xorSwap (int *x, int *y)
{
   if (x != y)
      {
     *x ^= *y;
     *y ^= *x;
     *x ^= *y;
   }
}
```

Question 22> What does following code do , if we pass head pointer of a link list.

```
void list( struct node * head)
{
 if(head)
 {
   list(head->next);
   printf("%d",head->value);
 }
}
```

Solution> The above code snippet prints link list in reverse order.

Question 23> Given a randomly filled array of N numbers and a number X . Print all pair inside array such that they sum up to X.

Solution>
Algorithm :
---
1. Sort the array using any of the many sorting algorithms available.
2. Keep two pointers one at the start of the array and other at the end of the array.
3. Now check : if sum of the value pointed by those pointer is equal to X then print the pair and move both pointer towards each other else if sum of the value is less then X then move first pointer towards end of the array by one step else if sum of the value is more than X then move 2nd pointer towards start of the array by one step.
4. Go to step 3 till first pointer index value is less than the 2nd pointer index value.
5. END

Step 2-4 takes only O(n). As we are having only one pass over the array but step 1 takes nlogn time due of sorting [without using any extra space], so total complexity is O(nlogn) time and O(1) space.

Question 24> Find triplets in an integer array A[] which satisfy following condition :
a[i]^2 + a[j]^2 = a[k]^2

Solution>
1. Let the inpur array is A[] = 2 1 9 4 8 7 6 3 5
2. Sort(A) the input array
3. Finally A[] = 1 2 3 4 5 6 7 8 9
4. // Make an array of squares to avoid computing them again and again

```
for (i = 0; i < n; i++)
{
  Sq[i] = A[i] * A[i];
}
Sq[] = 1 4 9 16 25 49 64 81
n = length;
for (i = n-1; i >= 0; i--)
{
   res = false;
```

/* Search for 2 numbers in Sq array from 0 to i-1 which adds to Sq[i] like we have discussed in previous *Question*. This will give a search result in O(n) time. Make res as true if we are able to find any triplet. */

```
   if (res)
```

```
    {
      print(triplet);
    }
  }
```

Question 25> Find 4th smallest element in a Binary Search Tree.

Solution>

```
struct NODE
{
  struct NODE *left;
  int value;
  struct NODE *right;
}

int inorder(struct NODE *curr)
{
static int i=0;
if(curr->left != NULL)
  inorder(curr->left);
printf("%d", curr->value);
i++;
if(i == 4)    // 4th smallest
return curr->value;
if(curr->right != NULL)
  inorder(curr->right);
}
```

Question 26> Two coordinate points p(x0,y0)and q(x1,y1) are given and write an algorithm to draw a straight line using teh two coordinates.

Solution>

```
function line(x0, x1, y0, y1)
   int deltax = x1 - x0
   int deltay = y1 - y0
   real error = 0
   real deltaerr := abs (deltay / deltax)    // Assume deltax != 0 (line is not vertical)
   int y = y0
   for x from x0 to x1
```

```
  {
    plot(x,y) // plot(x,y) plots a point
    error = error + deltaerr
    if abs(error) >= 0.5 then  // abs returns absolute value
        {
          y = y + 1;
          error = error - 1.0
        }
  }
```

The above algorithm is tracking a small error value between -0.5 and +0.5 : The vertical distance between the rounded and the exact y values for the current x. Each time x is increased, the error is increased by the slope. If it exceeds 0.5, y is increased by 1 and the error is decremented by 1.0. The above algorithm is called Bresenham's line algorithm.

Question 27> Give the binary equivalance of -5

Solution>

```
#include<stdio.h>
void showbits(int n);
int main()
{
      int n;
      printf("Enter an integer for which you want to see the binary equivalent");
      scanf("%d",&n);
      showbits(n);
      return 0;
}
void showbits(int n)
{
      int i,bit,mask,count=0;
      for(i=15;i>=0;i--)        // Assuming size of the integer is 16 bits
      {
        mask=1<<i;
        bit = n & mask;
        if(bit == mask)
                    printf("1");
        else
                    printf("0");
        count ++;
```

```
        if(count%4==0)
        printf("  ");
       }
}
```

Question 28> Write an algorithm to reverse a doubly link list.

Solution>

```
public void ReverseLinkedList (LinkedList linkedList)
{
LinkedListNode firstNode;  // This node will be the first node in the swapping
LinkedListNode secondNode; // This node will be the second node in the swapping
int numberOfRun;
 // This variable will be used to determine the number of time swapping is required
LinkedListNode tail = linkedList.Head;
while (tail.Next != null)
tail = tail.Next;
firstNode = linkedList.Head;
secondNode = tail;
numberOfRun = linkedList.Length / 2;
// ------------------------------------------------------------
// Swap node's objects
// ------------------------------------------------------------
object tempObject; // This will be used in the swapping of 2 objects
for (int i=0; i < numberOfRun; i++)
{
// Swap the objects by using a 3rd temporary variable
tempObject = firstNode.Item;
firstNode.Item = secondNode.Item;
secondNode.Item = tempObject;
firstNode = firstNode.Next;
// Hop to the next node from the beginning and to the previous node from the end
secondNode = secondNode.Previous;
}
}
```

Question 29> Predict the output of the following code snippet.

```
#include<stdio.h>
void crazy(int,int,int);
int main()
{
  crazy(3,4,5);
  return 0;
}

void crazy(int n, int a, int b)
{
  if(n == 0) return;
  crazy(n-1,b+n,a);
  printf("%d%d%d",n,a,b);
  crazy(n-1,b,a+n);
}
```

Solution>

168284141034519525717 7

Question 30> What are Data breakpoints? Give two scenarios where we make the use of this ?

Solution>

A data breakpoint is an intentional stopping of a program, put in place for debugging purposes. More generally, a breakpoint is a means of acquiring knowledge about a program during its execution. During the interruption, the programmer inspects the test environment (general purpose registers, memory, logs, files, etc.) to find out whether the program is functioning as expected. When a breakpoint is hit, stack trace of each thread may be used to see the chain of function calls that led to the paused instruction.

* A function breakpoint causes the program to break when execution reaches a specified location within a specified function.
* A file breakpoint causes the program to break when execution reaches a specified location within a specified file.
* An address breakpoint causes the program to break when execution reaches a specified memory address.
* A data breakpoint causes the program to break when the value of a variable changes. You can set a data breakpoint on a global variable or a local variable in the top-most scope of a function.

Use a data bp when you know some data is getting changed and you want to know where in the instruction stream that is happening.

Question 31> Write the function for changing a number from hexadecimal to integer or implemene

Solution>

the function htoi()

```
#include <stdio.h>
#include <limits.h>
#include <ctype.h>
unsigned int htoi(char s[])
{
   unsigned int val = 0;
   int x = 0;
   if(s[x] == '0' && (s[x+1]=='x' || s[x+1]=='X')) x+=2;
   while(s[x]!='\0')
   {
     if(s[x] >= '0' && s[x] <='9')    { val = val * 16 + s[x] - '0';     }
     else if(s[x]>='A' && s[x] <='F') { val = val * 16 + s[x] - 'A' + 10; }
     else if(s[x]>='a' && s[x] <='f') { val = val * 16 + s[x] - 'a' + 10; }
     else return 0;
        x++;
   }
   return val;
}
void main(int argc, char *argv[])
{
   char hexalpha[] = "0xB";
   if(htoi(hexalpha)==0)
        printf("\nHex string overflow or not a hex character.\n");
   else
        printf("%u\n", htoi(hexalpha));
}
```

Question 32> Write a function so that memory is assigned to a 2 dimensional array using malloc function.

Solution>

```
#include<stdio.h>
#include<stdlib.h>
int** matrix;
```

```
int numRows = 3;int numCols=3;
int x, y,i;
int main()
{
matrix = (int **) malloc(numRows * sizeof(int *));
if(matrix == NULL)
{
  free(matrix);
  printf("Memory allocation failed for matrix[]\n");
  exit(-1);
}
for(x = 0; x < numRows; x++)
{
matrix[x] = (int *) malloc(numCols * sizeof(int));
if(matrix[x] == NULL)
 {
  free(matrix[x]);
  printf("Memory allocation failed while allocating for matrix[x][]\n");
  exit(-1);
 }
}
/* Initialize matrix to be filled with 0's. */
for(x = 0; x < numRows; x++)
   for(y = 0; y < numCols; y++)
      matrix[x][y] = 0;
/* Print matrix */
for(x = 0; x < numRows; x++)
{
  printf("\n");
  for(y = 0; y < numCols; y++)
  printf("%d\t", matrix[x][y]);
}
/* Deallocate memory of 2D array. */
for(x = 0; x < numRows; x++)
   free(matrix[x]);
   free(matrix);
return 0;
}
```

Question 33> Write a program to count the number of bits that needs to be altered while swaping values a and b.

Solution>

```
#include <stdio.h>
#include <stdlib.h>
unsigned bit_count(unsigned n);
void main()
{
   unsigned int a, b, mask;
   a = 10;
   b = 0;
   mask = a ^ b;
   printf("a is %u\n b is %u \n", a, b);
   printf("%u bits would need to be flipped in each object \n", bit_count(mask));
   printf("%u ^ %u is %u\n", a, mask, a ^ mask);
   printf("%u ^ %u is %u\n", b, mask, b ^ mask);
}
unsigned bit_count(unsigned int n)
{
   unsigned int count;
   for (count = 0; n != 0; n &= n - 1)
      {
        ++count;
      }
   return count;
}
```

Output

```
------
a is 10
b is 0
2 bits would need to be flipped in each object  // 10 = 1010 and 0 = 0000. So two bits are req to flip the numbers
10 ^ 10 is 0
0 ^ 10 is  10
```

Question 34> Write a function to sort a link list

Solution>

```
Sort( Node *Head)
```

```
{
   node* first,second,temp;
   first = Head;
   while(first!=null)
      {
    second=first->NEXT;
    while(second!=null)
       {
     if(first->value < second->value)
                {
       temp = new node();
       temp->value = first->value;
       first->value = second->value;
       second->value = temp->value;
       delete temp;
      }
      second = second->NEXT;
    }
    first = first->NEXT;
   }
}
```

Question 35> Write a code to check if a binary tree confoms to the properties of a binary search tree.

Solution>

```
bool IsBST(node *tree, int lower_bound, int upper_bound)
{
 if (tree == NULL) return true;                                    // An empty subtree is OK.
 if (tree->value <= lower_bound) return false;   // Value in the root is too low.
 if (tree->value >= upper_bound) return false;   // Value in the root is too high.
 // Values at the left subtree should be strictly lower than tree->value
 if (!IsBST(tree->left, lower_bound, tree->value)) return false;
 // Values at the right subtree should be strictly greater than tree->value
 if (!IsBST(tree->right, tree->value, upper_bound)) return false;
 return true;
}
```

Question 36> There are two ropes. Each one can burn in exactly one hour. They are not necessarily of the same length or width as each other. They are also not of uniform width (may be wider in middle than in the end), thus burning half of the rope is not necessarily 1/2 hour. By burning the ropes, how do you measure exactly 45 minutes worth of time?

Solution>

A rope has two ends. Setting fire to both ends of a rope simultaneously burns the rope twice as fast, hence it's burning time is 30 minutes. First lit rope 1 at both the ends and one end of rope 2. After 30 minutes rope 1 is completely burnt and rope 2 still has 30 minutes worth of burning time left. Lit the other end of rope 2. This will burn the remaining rope in 15 minutes, totaling the burning time to 45 minutes.

Question 37> Write a program to find the number of zeros in n factorial

Solution>

```
include<stdio.h>
int main()
{
 int n, res = 0;
 printf("Enter the valur of n\n");
 scanf("%d", &n);
 while(n>0)
 {
       n = n/5 ;
      res += n;
 }
 printf("\nNumber of eros is %d", res);
 return 0;
}
```

Question 38> Write a code to check if a string is palindrome or not ?

Solution>

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int isPalindrome( char *s );
int main()
{
  int i = 0;
  int ch;
```

```
  char s[100];
  while ((ch = getchar()) != '\n') {
    if (isalpha(ch)) {
      s[i] = ch;
      i++;
    }
  }
  if( isPalindrome(s) == 1)
  {
   printf("Yes, is a palindrome.\n");
  }
  else
  {
   printf("No, not a palindrome.\n");
  }
  return 0;
 }
int isPalindrome( char *s )
{
  int i = strlen(s)-1;
  int j = 0;
  while (j<=i) {
   if(s[j] != s[i])
       {
     return 0;
   }
   i--;
   j++;
  }
  return 1;
}
```

Question 39> Write a code to find the Kth largest element in a Binary Search Tree.

Solution>

```
#include<stdio.h>
#include<stdlib.h>
struct tree
{
```

```
        int data, size;
        struct tree *left, *right;
};
struct tree* insert(struct tree* t, int p)
{
   if(t == NULL)
   {
        struct tree *tmp=(struct tree *)malloc(sizeof(struct tree));
    tmp->data = p;
    tmp->left = tmp->right = NULL;
    tmp->size = 1;
    return tmp;
   }
   else if( p <= t->data )
   {
    t->size += 1;
    t->left = insert(t->left, p);
   }
   else
   {
    t->size += 1;
    t->right = insert(t->right,p);
   }
   return t;
 }
struct tree* findkthsmallest(struct tree* t, int k)
   {
        int left_size = 0;
        if(t->left != NULL)
                left_size = t->left->size;
        if(left_size + 1 == k)
                return t;
        else if(k <= left_size)
                return findkthsmallest(t->left,k);
        else
                return findkthsmallest(t->right, k-left_size - 1);
   }
struct tree* findkthlargest(struct tree* t,int k)
{
        int right_size = 0;
        if(t->right != NULL) right_size = t->right->size;
```

```
    if(right_size + 1 == k) return t;
    else if(k <= right_size) return findkthlargest(t->right,k);
    else return findkthlargest(t->left, k-right_size - 1);
}
 int main()
   {
    struct tree *p, *q, *r;
    p = (struct tree *)malloc(sizeof(struct tree));
    int n, i, k;
    for(i = 0 ; i < 5 ; i++)
    {
                printf ("\n Enter element");
                scanf("%d",&n);
                p = insert(p,n);
               }
    printf("\n Enter k ");
    scanf("%d",&k);     // kth largest element
    q = findkthlargest(p, k);
    printf("\n Answer = %d\n",q->data);
    r = findkthsmallest(p, k+1);
    printf("\n Answer = %d\n", r->data);
    return 0;
    }
```

Question 40> Write a program to reverse each word in a sentence.

Solution>

```
#include<stdio.h>
#include<string.h>
char * xstrrev(char *s);
int main()
{
   char str[]="Rahul is bad";  // Output should be luhaR si dab
   char temp[50]; char *t1;
   int i; int j = 0;
   int n=strlen(str);
   printf(" \n Len = %d \n",n);
  for(i=0;i<=n ; i++)
  {
       if(str[i]!=' ' &&  str[i]!='\0')
      {
```

```
            temp[j]=str[i]; j++;
        }
         if(str[i]==' ')
        { temp[j]='\0';  t1=xstrrev(temp);
          printf("%s ",t1);
         j=0;
        }
        if(str[i]=='\0')
        {
         temp[j]='\0'; t1=xstrrev(temp);
         printf("%s ",t1);
        }
   }
  return 0;
}
char * xstrrev(char *s)
{
 int i; int len = strlen(s); int j; char t;
 for(i=0,j=len-1;i<(len/2);i++,j--)
{
  if(s[i]!='\0')
   {  t=s[i]; s[i]=s[j]; s[j]=t; }
}
return s;
}
```

Question 41> Write a program to check if a linked list is palindrome or not.

Solution>

```
#include<stdio.h>
#include<stdlib.h>
#define FALSE 0
#define TRUE 1
typedef struct node_tag { int data; struct node_tag *next; }node_type;
void print_link_list(node_type *root);
node_type *create_link_list(int n);
int check_palindrome(node_type *root);
node_type *hroot;
main()
{
```

```
	int n;
	int ret_val=FALSE;
	node_type *head;
	printf("Give the number of node\n");
	scanf("%d",&n);
	head = create_link_list(n);;
	print_link_list(head);
	hroot = head;
	ret_val = check_palindrome(head);
	if(ret_val == 1)
		{
		  printf("\nret_val=%d\t PALINDROME",ret_val);
		}
	else
		{
		  printf("\nret_val=%d\t NOT PALINDROME",ret_val);
		}
}
int check_palindrome(node_type *x)
{
	int ret_val;
	if(x)
	{
	  ret_val=check_palindrome(x->next);
	  if(x->data != hroot->data)
			return FALSE;
	  hroot = hroot->next;
	  return ret_val;
	}
	return TRUE;
}
void print_link_list(node_type *root)
{
	node_type *temp=root;
	while(temp)
	{
		printf("%d-->", temp->data);
		temp = temp->next;
	}
	printf("\n");
}
```

```
node_type *create_link_list(int n)
{
      int i, a[10];
      node_type *head,*temp,*x;
      head = NULL;
      printf("Give the elements of Linked list\n");
      for(i=0 ; i<n ; i++)
      { scanf("%d",&a[i]); }
      for(i=0;i<n;i++)
      {
            temp=(node_type *)malloc(sizeof(node_type));
            temp->data=a[i];
            temp->next=NULL;
                                    if(i==0)
                                    {
              head = temp;
              x = temp;
            }
            else
            {
              x->next = temp;
              x = x->next;
            }
      }
      return head;
}
```

Question 42> Write a program to find out whether stack addresses will be increasing or decreasing in memory? This *Question* can be rephrased as write a program to check if the stack is growing upward or downward.

Solution>

```
/* To show if the stack is increasing or decreasing */
#include<stdio.h>
int main()
{
 int a,b;
 printf("%u",&a);
 printf("\t%u\n",&b);
 printf("%s",((&a - &b) > 0 ? "increasing" : "decreasing"));
```

```
 return 0;
}
```
In gcc compiler output : 2972454252 2972454248
increasing

Question 43> How can four employees calculate the average of their salaries without knowing each other's salary?

Solution>
Lets name these employees A, B, C and D
1. A chooses any random value and whispers it to B privately
2. B chooses any random value and whispers it to C privately
3. C chooses any random value and whispers it to D privately
4. D chooses any random value and whispers to to A privately
So each employee has two numbers now - incoming and outgoing. Each employee should get own salary, then add the incoming and subtract the outgoing and report the result. Then they should add all reports and divide by four.

Question 44> There is a bank which give the 100% rate of interest(annualy). You have 1 dollar today with you and you deposit that in the bank. After how much time would you become the richest man.

Solution>
pow(2,n) = R ; R is amount of money richest person is having and n is the no. of yrs in which he will become the richest.
So, n = log(R) // Log is base 2

Question 45> In a country in which people only want boys, every family continues to have children until they have a boy. If they have a girl, they go for another child. If they have a boy, they stop. What is the proportion of boys to girls in the country?

Solution>
Out of x couples in the village, x/2 would have boy and x/2 would have girl. Therefore the x/2 would go for another child and there would be x/4 boy and x/4 girl. The process goes on. Mathematically you can write as:
no of boys : x/(2) + x/(2^2) + x/(2^3) .....
no of girls: x/(2) + x/(2^2) + x/(2^3) .....

So the ratio will always be one .

Question 46> A chinese emperor had to choose a new adviser amongst 3 sages, all of them equally wise. He placed a problem to them: "To choose one of you, you'll play a simple and fair game: In this sack there are 3 white balls and 2 black balls. Each of you will be blindfolded and will pick one ball and place it on your head. After that, the blindfolds will be removed and each one in turn will try to guess the colour of the ball upon his head, by observation of the other picked balls. However, beware. You may pass your turn in guessing, but if you state a colour and fail, you're disqualified. This way I'll learn which one is the most intelligent amongst you" The sages talked briefly to each other and promptly refused: "Dear lord, it's of no use, since the game is not fair. The last one of us to guess in the first round will know the answer." and the sages promptly demonstrated this to the emperor, who was so amazed by their wits that he appointed all 3 has his advisers.

Solution>
Case 1. If they have two black balls, the third will immediately know that he is having a white.

Case 2. They have picked one black and two white.
Person1 sees one black and one white, he wont be sure and he will pass. Second one would come to know that the first has seen either both white or one white and one black. If he sees the third has black, he will come to know that he has white, but if he sees white, he won't be sure of his color. And he will pass. Now the third one will know that the second has passed, means he has not seen black on me. So he will be sure that he has white.

Case 3. They picked all white.
The first will not be sure and will pass. passing again means the first has seen either two white or one black. The second again can see two whites, so will not be sure if he has black or white. If he has seen any black, he would have sure he has white one. Again third one will be sure that he has not black, else the second would have known his color. So he will come to know his color .

Question 47> Write a generic Swap Macro in C.

Solution>
```
#include <stdio.h>
#define SWAP(x, y) do { typeof(x) temp = x; x = y; y = temp; } while (0)
/* typeof is an operator provided by several programming languages which determines the data type of a given variable. */
int main(void)
{
int a = 10000, b = 2;
```

```
float x = 99999.0f, y = 2.0f;
int *pa = &a;
int *pb = &b;
printf("BEFORE:\n");
printf("a = %d, b = %d\n", a, b);
printf("x = %f, y = %f\n", x, y);
printf("pa = %p, pb = %p\n", pa, pb);
SWAP(a, b); // swap ints
SWAP(x, y); // swap floats
SWAP(pa, pb); // swap pointers
printf("AFTER:\n");
printf("a = %d, b = %d\n", a, b);
printf("x = %f, y = %f\n", x, y);
printf("pa = %p, pb = %p\n", pa, pb);
return 0;
}
```

Output :

```
---------
BEFORE:
a = 10000, b = 2
x = 99999.000000, y = 2.000000
pa = 0022FF74, pb = 0022FF70
AFTER:
a = 2, b = 10000
x = 2.000000, y = 99999.000000
pa = 0022FF70, pb = 0022FF74
```

Question 48> There are 8 cricket teams say t1, t2, t3 ... t8 and each team plays two matches against each team. These matches are known as league matches. Now after league matches, top 4 teams(top scorer) will enter semi finals. Some data is -

A) Each win gives one point to winning team.

B) There is no draw in any match.

C) At any point if scores of any two teams are equal then winner is decided by an automatic machine and is out of your control.

Now find -

i) Minimum number of matches to win so that a team can qualify for Semi finals.

ii) Maximum Number of matches won by a team when it canNOT qualify for semifinals.

Solution>

Let A, B, C, D, E, F, G, H are 8 teams. Total number of matches that will be played = 8C2 * 2 = 56. Hence total points available = 56

Mimum number of matches to win to qualify = 4

Teams Wins Losses

A 14 0

B 12 2

C 10 4

Now pts left : 56 - 36 = 20. Let us divide it equally among rest of the teams.

D 4 10

E 4 10

F 4 10

G 4 10

H 4 10

Maximum number of matches won by team when not qualify = 10

Last 3 teams play amongst themselves = 3C2 *2 = 6 matches. Now points left = 56 - 6 = 50. Divide it equally among top 5 teams.

Teams Win Lost

A 10 4

B 10 4

C 10 4

D 10 4

E 10 4

F 4 10

G 2 12

H 0 14

Question 49> Write Test cases to test Coffee vending machine.

Solution>

The first thing to check in this case is whether the vending machine is functional or not.

1> An item will be dispensed if you click the dispense button with sufficient amount of cash.

2> If you click the dispense button with more money than required for buying the item, then the item will be given to you along with the excess money.

3> If you click the dispense button without inserting cash at all, then no item will be given and you will be suggested to insert the required amount.

4> If you enter very large amount of items, greater than what the holding capacity of the vending machine is, then you will be told on the screen that the number of items you entered must be reduced.
5> If there are no items in the vending machine and you still put in money, then you will not get any items and the money will be returned back to you.
6> If invalid coin is inserted M/c should not work
7> Insert coin -> press coffee button -> while coffee is pouring down -> press other flavour button -> check whether you get the same flavour selected first or the flavour selected second.
8> Insert coin -> press coffee button -> in the middle switch off the machine -> check whether the coffee comes or not.

Question 50> A function prototype
char * replace(char *str, char *find, char *replace)
Code it and specifications are - str is input string, find is a given pattern and you need to find this pattern and if it exists replace it with another
string (replace).

for example -
If Input string is "aabcdef" and find pattern is "bcd" and replace string is "xxxx" the output should be "aaxxxxef".

Solution>
Working Code

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 100
int i,j;
int is_match_found(char *src,char *input,int l);
void replace_string(char *dst,char *replace,int j);
main()
{
    int flag;
    char src[MAX],dst[MAX],input[10],replace[10];
    printf("Enter your Source string \n");
    gets(src);
    printf("Enter the string to be replaced \n");
    gets(input);
    printf("Enter what you want to put \n");
```

```
        gets(replace);
    j=0;
    i=0;
    while(i<strlen(src))
    {
        flag=is_match_found(src,input,i);
        if(flag)
        {
            replace_string(dst,replace,j);
            i=i+strlen(input);
            j=j+strlen(replace);
        }
        else
        {
            dst[j]=src[i];
            j++;
            i++;
        }
    }
    printf("Final string is %s", dst);
}
int is_match_found(char *src,char *input,int l)
{
    int flag=1;
    int k,m;
    for(k=l,m=0;((k<strlen(src)&&m<strlen(input)));k++,m++)
    {
        if(src[k]!=input[m])
        {
            flag=0;
            break;
        }

    }
    return flag;
}
void replace_string(char *dst,char *replace,int j)
{
    int k=0;
    while(k<=strlen(replace))
    {
```

```
        dst[j]=replace[k];
        j++;
        k++;
    }
}
```

Question 51> Given a binary tree, print out all of its root-to-leaf paths .

Solution>

```
#include<stdio.h>
int main()
{
  printAllPaths(root);
}
void printAllPaths(struct Tree *root)
{
       int path[100];
       pathFinder(root,path,0);
}
void printThisPath(int path[], int n)
{
       for(int i = 0; i < n; i++)
         printf("%d ",path[i]);
       printf("\n);
}
void pathFinder(struct Tree *node, int path[], int pathLength)
{
       if(node == NULL) return;
       path[pathLength++] = node->data;
       /* Leaf node is the end of a path.
          So, let's print the path */
       if(node->left == NULL && node->right == NULL)
                printThisPath(path, pathLength);
       else {
                pathFinder(node->left, path, pathLength);
                pathFinder(node->right, path, pathLength);
       }
}
```

Question 52> Given a point and a polygon, check if the point is inside or outside the polygon.

Solution> The algorithm that is discussed below is called Ray-Casting algorithm.
A pseudocode can be simply:

```
Draw a straight line from test point P
count = 0
for each side in polygon:
{
  if ray_intersects_segment(P,side) then  // P represents the point being tested
    count = count + 1
}
if is_odd(count) then
  return inside
else
  return outside
```

Where the function ray_intersects_segment return true if the horizontal ray starting from the point P intersects the side (segment), false otherwise.
An intuitive explanation of why it works is that every time we cross a border, we change "country" (inside-outside, or outside-inside), but the last "country" we land on is surely outside (since the inside of the polygon is finite, while the ray continues towards infinity). So, if we crossed an odd number of borders we were surely inside, otherwise we were outside;

One simple way of finding whether the point is inside or outside a polygon is to test how many times a ray starting from the point intersects the edges of the polygon. If the point in *Question* is not on the boundary of the polygon, the number of intersections is an even number if the point is outside, and it is odd if inside.

If implemented on a computer with high precision, the results may be incorrect if the point lies very close to that boundary, because of rounding errors. Hence one should introduce a numerical tolerance "e" and test whether P lies within e of L, in which case the algorithm should stop and report "P lies very close to the boundary."

Question 53> You have been given sand time clocks of 4 and 7 minutes. How to measure 9 minutes?

Solution>
What we can do is:
Start with both sand clocks i.e. 7 and 4.

at t = 4 sand clock(4) will be empty and sand clock(7) will be remaining with 3 units. Revert sand clock(4).
at t = 7 sand clock(7) will be empty and sand clock(4) will be remaining with 1 unit. Revert sand clock(7).
at t = 8 sand clock(4) will be empty and sand clock(7) will be remaining with 6 units. Revert sand clock(7) to measure 1 unit.
It will measure 9 minutes

Question 54> Design a Thread safe Array Based queue with a fixed size. The behavior should be in such a way that if multiple threads are accessing the same queue and if the queue is full the threads will wait for their turn and if some other threads removes elements from queue, the waiting threads will get a chance to add elements. [This Ques is identical to Producer - Consumer Problem having multiple consumers and producers]

In computer science, producer-consumer problem (also known as the bounded-buffer problem) is an example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The *Solution* for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The *Solution* can be reached by using semaphores. An inadequate *Solution* could result in a deadlock where both processes are waiting to be awakened. [The problem can also be generalized to have multiple producers and consumers and this was asked by Adobe].

Solution>
We use two semaphores, fillCount and emptyCount, to solve the problem. fillCount is incremented and emptyCount decremented when a new item has been put into the buffer. If the producer tries to decrement emptyCount while its value is zero, the producer is put to sleep. The next time an item is consumed, emptyCount is incremented and the producer wakes up. We also use 1 binary

semaphore called mutex.
semaphore mutex = 1;
semaphore fillCount = 0;
semaphore emptyCount = BUFFER_SIZE;

```
producer()
{
   while (true) {
```

```
      item = produceItem();
      down(emptyCount);
         down(mutex);
            putItemIntoBuffer(item);
         up(mutex);
      up(fillCount);
   }
}
consumer()
{
   while (true) {
      down(fillCount);
         down(mutex);
            item = removeItemFromBuffer();
         up(mutex);
      up(emptyCount);
      consumeItem(item);
   }
}
```

Question 55> There is an array having 1 to 10 numbers randomly placed. But two numbers are missing from the list. What are those two numbers? You are not allowed to use any hashing or any sorting techniques.

Solution>

```
int main()
{
int a[] = {1,2,3,4,5,6,7,9};
int b[11];
int i;
for(i=1;i<=10;i++) { b[i] = FALSE; }
for(i=0;i<8;i++)   { b[a[i]] = TRUE; }
for(i=1;i<11;i++)
      {
                  if(arr1[i]!=TRUE)
                        printf("\n%d ", i);
               }
return 0;
}
```

Question 57> Count the number of Set bits in a given number in O(1).

Solution>

```
#include<stdio.h>
int main()
{
 int x = 15;
 int z;
 z = BitCount(x);
 printf("\nAnswer = %d\n",z);
 return 0;
}
int BitCount(int u)
{
     int uCount = u;
     do
     {
          u = u>>1;
          uCount -= u;
     } while(u);
    return uCount;
}
```

Explanation:

Take a 32 bit number n;

n = a31 * (2^31) + a30 * (2^30) +.....+ ak * (2^k) +....+ a1 * (2^1) + a0 * (2^0);

Here a0 through a31 are the values of bits (0 or 1) in a 32 bit number. Since the problem at hand is to count the number of set bits in the number, simply summing up these co-efficients would yeild the *Solution*. (a0 + a1 +..+ a31).

Take the original number n and store in the count variable.

count=n;

Shift the orignal number 1 bit to the right and subtract from the orignal.

count = n - (n >>1);

Now Shift the original number 2 bits to the right and subtract from count;

count = n - (n>>1) - (n>>2);

Keep doing this until you reach the end.

count = n - (n>>1) - (n>>2) - ... -(n>>31);

Let analyze and see what count holds now.

n = a31 * (2^31) + a30 * (2^30) +.....+.....+....+ a1 * (2^1) + a0 * (2^0);

n >> 1 = a31 * (2^30) + a30 * (2^29) +.....+.....+....+ a1 * (2^0)

n >> 2 = a31 * (2^29) + a30 * (2^28) +.....+.....+....+ a2 * (2^0)

..

..

n >> 31 = a31;

count = n - (n>>1) - (n>>2) - ... -(n>>31) = a31 + a30 +..+ a0;

Question 57> Two sorted arrays are given. Without using additional memory, merge these two arrays (second array is having enough space for merging).

Solution>

```
int A[5] = {1,3,8,9};
int B[9] = {2,4,5,6,7,0,0,0,0}; // last four as place holders for the merge operation
int idx = 8; // start from the end of the larger array;
int idx_a = 3, idx_b = 4; // we also need the indices of the largest elements in both arrays
while (idx_a >= 0) { // done when A has been traversed
if (idx_b < 0 || A[idx_a] > B[idx_b])
{
  B[idx] = A[idx_a];
  idx_a--;
}
else
{
B[idx] = B[idx_b];
idx_b--;
}
                                        idx--;
                                        }
```

Question 58> Find if a number is divisible my 3, without using % , / , *. You can use atoi()?

Solution>

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

```
int fnSum(char *);
char ch;
int main()
{
char cNumber[100];
int nsum=0,i=0;
printf("Enter the number\n");
scanf("%s",&cNumber);

nsum = fnSum(cNumber);
if(nsum == 3 || nsum == 6||nsum == 9 )
printf("Given number is divisible by 3\n");
else
printf("Given number is not divisible by 3\n");
return 0;
}
int fnSum(char *cNumber)
{
int nsum=0; int i=0;
while (*(cNumber+i) != '\0')
{
ch=(*(cNumber+i));
nsum +=atoi(&ch);
i++;
}
if(nsum >= 10)
{
sprintf(cNumber,"%d",nsum);
nsum = fnSum(cNumber);
}
return nsum;
}
```

Question 59> How do we find the nth element from the right of a singly linked list? Would reversing the link list and traversing n elements be the only possible *Solution*?

Solution>

Keep first pointer to the first node of the list and second pointer at the (n+1)th node from the beginning of the list. Traverse both simultaneously till second becomes NULL. At that time, first pointer would point to the nth node from the right.

eg in case of 1 -> 2 -> 3 -> 4 , if i want to get 4th node from right (Ans is 1), make second pointer point to (n+1)th node from the beginning {1 is the 1st node, 2 is the 2nd node, 3 is the 3rd node, 4 is the 4th node and NULL is the 5th node} ie second will point to NULL. As second is pointing to NULL , so no extra moves and first already points to required 4th node from right.

Question 60> Difference between Macro and Inline Functions \\?

Solution>
i) Inline functions are similar to macros because they both are expanded at compile time, but the macros are expanded by the preprocessor, while inline functions are parsed by the compiler.
ii)Expressions passed as arguments to inline functions are evaluated once. In some cases, expressions passed as arguments to macros can be evaluated more than once. One must be careful with macros, because they can have the arguments evaluated more than once. Here is an example:

Code:

```
#include<stdio.h>
#define max(a,b) (a>b?a:b)
int main()
{
  int a = 0;
  int b = 1;
  int c = max(a++, b++);
  printf("a = %d  b = %d \n",a,b);
  return 0;
}
```

The intention is to have the program print 1 and 2, but because the macro is expanded to:
Code: int c = a++ > b++ ? a++ : b++;
b will be incremented twice, and the program prints 1 and 3.
Inline : It is a function provided by C++. It is declared by using the keyword inline before the function protoype.
Macro : It is a a preprocessor directive provided by C. It is declared by using the preprocessor directive #define before the actual statement.
Inline functions may or may not be expanded by the compiler. It is the compiler's decision whether to expand the inline function or not.
Macros are always expanded.
Inline functions pass arguments just like regular functions do. If the argument is an expression such as 4.5 +7.5 then the function passes the value of the expression 12 in this case.

Expressions passed into macros are not always evaluated before entering the macro body.
Thus #define square(x) x*x
:
:
b=square(4.5+7.5)
This will be replaced by : b=4.5+7.5*4.5+7.5

Question 61> Write a program to illustrate Merge Sort.

Solution>

```
#include<stdio.h>
void merge(int a[], int low, int high, int mid);
void mergesort(int a[], int low, int high)
{
int mid;
if(low < high)
 {
  mid=(low+high)/2;
  mergesort(a,low,mid);
  mergesort(a,mid+1,high);
  merge(a,low,high,mid);
 }
}
void merge(int a[], int low, int high, int mid)
{
int i, j, k, c[50];
i=low;
j=mid+1;
k=low;
while((i<=mid)&&(j<=high))
 {
   if(a[i]<a[j])
   {
     c[k]=a[i];
     k++;
     i++;
   }
   else
   {
```

```
    c[k]=a[j];
    k++;
    j++;
   }
 }

while(i<=mid)
 {
  c[k]=a[i];
  k++;
  i++;
 }
while(j<=high)
 {
  c[k]=a[j];
  k++;
  j++;
 }
for(i=low;i<k;i++)     // copying the result into original array
{ a[i]=c[i]; }
} // end of func
int main()
{
 int arr[20];
 int n,i;
 printf("Enter number of data:");
 scanf("%d",&n);
 printf("enter the data: ");
 for(i=0;i<n;i++)
 { scanf("%d",&arr[i]); }
 mergesort(arr,0,n-1);
 printf("\n");
 for(i=0;i<n;i++)
 {  printf("%d ",arr[i]); }
 return 0;
}
```

Question 62> Write a program to illustrate Heap Sort.

Solution>

```
#include<stdio.h>
void hs(int a[],int n);
void make_max_heap(int a[],int low,int high);
void build_max_heap(int a[],int n);
int main()
{
    int a[15]={7,10,25,17,23,27,16,19,37,42,4,33,1,5,11};
    int i,n=15; // n is no. of elements
    hs(a,n);
    for(i=0;i<15;i++)
    printf(" %d  ",a[i]);
    return 0;
}
    void hs(int a[],int n)
    {
        build_max_heap(a,n);  int x,i;
        for(i=n-1;i>0;i--)
        {
                    x=a[0];
                    a[0]=a[i];
                    a[i]=x;
                    make_max_heap(a,0,i-1);
        }
    }
    void build_max_heap(int a[],int n)
    {   int i;
        for(i=n/2-1;i>=0;i--)
        make_max_heap(a,i,n-1); // n-1 is index of last element
    }
    void make_max_heap(int a[],int low,int high)
    {
        int childpos = 2*low+1;
        while(childpos<=high)
        {
                    if((childpos<high) && (a[childpos]<a[childpos+1]))
                       childpos++;
                    if(a[low]>a[childpos]) break;
                    else {    int x=a[low];
                            a[low]=a[childpos];
                            a[childpos]=x;
```

```
                    low=childpos;
                    childpos=2*low+1;
                }
        }
    }
```

Question 63> There is a 2X4 matrix. In this you are supposed to arrange the numbers from 1-8, so that no consecutive numbers are adjacent(vertically, horizontally and diagonally) to each other.

Solution>
It is possible to do if one keeps on trying it randomly but it can be done with an intelligent approach too. What would that be?
1,2,3,4,5,6,7,8

Step:1) Keep no. in group of two alternate numbers. {1,3} {2,4} {5,7} {6,8}
Step:2) Now fill the center 4 block with two group containing first and last number.
(1,3) {6,8}
- 1 8 -
- 3 6 -
Step:3) Now fill with other group.
5 1 8 4
7 3 6 2
for 16 numbers : (1,2,3,4,5,6 ..... 15,16)
(1,3) (2,4) (5,7) (6,8) (9,11) (10,12) (13,15) (14,16)
- - - 1 14 - - -
- - - 3 16 - - -
- - 13 1 14 2 - -
- - 15 3 16 4 - -
- 5 13 1 14 2 12 -
- 7 15 3 16 4 10 -
9 5 13 1 14 2 12 6
11 7 15 3 16 4 10 8

Question 64> There is a circle enclosed in a square,such that it is touching all the four sides of the square. In the top left space between square and the circle, there is a rectangle with length 14 and breadth 7, such that top left corner of the rect is the top-left corner of square and bottom right corner lies on the circumference of the circle. What is the radius of the circle .

Solution>
Lets assume the radius of circle is 'r' and centre of circle is the origin.
So the coordinate of top right vertex of the enclosing square will be (r,r).
now the co-ordinate of the rectangle falling on the circle will be
(r-7, r-14) or (r-14, r-7).
Since this point is falling on the circle it has to follow the circle equation i.e
x^2 + y^2 = r^2
putting the values of x and y
(r-7)^2 + (r-14)^2 = r^2
=> r^2 - 42r + 245 = 0
=> (r-7)*(r-35) = 0
so r = 7 or r = 35
now r cannot be 7
so r = 35

Question 65> A frog has to cross a river. There are n rocks in the river, using which the frog can leap across the river. On its way across the river the frog can chose to skip a rock, but it cannot skip two consecutive rocks because that would be two far a distance for the frog to hop, also the frog would not skip the first rock and the last rock. E.g. if there are 3 rocks, 1,2,3 and 4, there could be three following routes it could take:
1,2,3,4
1,2,3,4
1,3,4
1,2,4
Write a recursive algorithm, that takes a number of rocks' and prints all the feasible paths. Ofcourse there can be other arguments too. (Recent Ques of March)

Solution>
Code [Backtracking Approach]

```
#include<stdio.h>
void PrintFrogPath(int a[], int length, int aCount,  int path[], int bCount)
{   int i;
      if(aCount == length && path[bCount-1] == a[aCount-1])
      {
            for(i = 0 ; i < bCount ; i++)
                  printf("%d ",path[i]);
               printf("\n");
            return;
      }
```

```
        else if(aCount < length)
        {
                for(i = 1; i<=2 ; i++)   // either 1 jump or 2 jumps are allowed. So i varies from 1 and 2.
                {
                        path[bCount] = a[aCount];
                        PrintFrogPath(a, length, aCount+i, path, bCount+1);
                }
        }
}
int main()
{
        int c[] = {1,2,3,4};
        int d[] = {0,0,0,0};
        PrintFrogPath(c,4,0,d,0);    // 4 represents no. of elements in c[].
        return 0;
}
```

Similar *Question* Could be : Given n stairs. How many number of ways can you climb if you climb either 1 or 2 stairs at a time?

Question 66> Consider there is an array with duplicates and you are given two numbers as input and you have to return the minimum distance between the two in the array with minimum omplexity.

Solution>

```
void minDistance (int* pArray, int size, int num1, int num2, int* firstIndex, int* secIndex)
{
        int curDst=0, minDst=0x7FFFFFFF, index1=0, index2=0;  // 7FFFFFFF is equi to 2147483647
        int moveFirst=1, moveSec = 1;
        while ((index1<size) && (index2<size))
  {
     while ((index1 < size) && (moveFirst))
    {
                if (pArray [index1] == num1)
                {
                        index1++;
                        break;
                }
                index1++;
        }
```

```
        while ((index2 < size) && (moveSec))
    {
      if (pArray [index2] == num2)
      {
        index2++;
                    break;
      }
         index2++;
    }
        if (index1 > index2)
    {
                moveFirst = 0;
                moveSec = 1;
                curDst = index1 - index2;
                if (curDst < minDst)
                {
                  minDst = curDst;
                  *firstIndex = index2-1;
                  *secIndex = index1-1;
                }
        }
       else
   {
          moveFirst = 1;
          moveSec = 0;
          curDst = index2 - index1;
          if (curDst < minDst)
     {
                minDst = curDst;
                *firstIndex = index1-1;
                *secIndex = index2-1;
          }
    } // end of else
   } // end of top while
} // end of func
int main ()
{
int array [] = {2,2,4,1,8,3,2,5,6,7,5,6,6,4,7,5};
int index1=0, index2=0;
minDistance (array, 16, 1, 6, &index1, &index2);
```

```
printf ("Index1 = %d, Index2 = %d\n", index1, index2);
printf ("Minimum distance b|w given num = %d\n", index2-index1);
return 0;
}
for minDistance (array, 16, 1, 6, &index1, &index2);
output :
------
Index1 = 3, Index2 = 8
Minimum distance b|w given num = 5
for minDistance (array, 16, 6, 1, &index1, &index2);
output :
------
Index1 = 3, Index2 = 8
Minimum distance b|w given num = 5
```

Question 67> Suggest algorithm to solve Readers Writers Problem. [Hint: Try using Semaphores for process synchronization]

Solution>

Semaphores can be used to restrict access to the database under certain conditions. In this example, semaphores are used to prevent any writing processes from changing information in the database while other processes are reading from the database.

```
semaphore mutex = 1;            // Controls access to the reader_count variable
semaphore db = 1;               // Controls access to the database
int reader_count;               // The number of reading processes accessing the database
Reader()
{
down(&mutex);                   // gain access to reader_count
reader_count = reader_count + 1;  // increment the reader_count
if (reader_count == 1)                              // if this is the first process to read the database
 down(&db);                  // a down on db is executed to prevent access to the database by a writing process
 up(&mutex);                 // allow other processes to access reader_count
 read_db();                  // read the database
 down(&mutex);               // gain access to reader_count
reader_count = reader_count - 1;  // decrement reader_count
if (reader_count == 0)                          // if there are no more processes reading from the database
 up(&db);                    // allow writing process to access the data
 up(&mutex);                 // allow other processes to access
}
```

```
Writer()
{
    create_data();          // create data to enter into database (non-critical)
    down(&db);              // gain access to the database
    write_db();             // write information to the database
    up(&db);                // release exclusive access to the database
}
```

Question 68> Explain briefly about Red Black tree and its properties.

Solution>

A red-black tree is a type of self-balancing binary search tree.

	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

A red-black tree is a binary search tree where each node has a color attribute, the value of which is either red or black. In addition to the ordinary requirements imposed on binary search trees, the following requirements apply to red-black trees:

1. A node is either red or black.
2. The root is black.
3. All leaves are black.
4. Both children of every red node are black.
5. Every simple path from a given node to any of its descendant leaves contains the same number of black nodes.

A red-black tree with n internal nodes (non leaf nodes) has height O(log n). A red-black tree is similar in structure to a B-tree of order 4, where each node can contain between 1 to 3 values and (accordingly) between 2 to 4 child pointers. Red-black trees offer worst-case guarantees for insertion time, deletion time, and search time. AVL trees are more rigidly balanced than red-black trees, leading to slower insertion and removal but faster retrieval.

Question 69> Explain briefly about splay trees and their usages in real world.

Solution>

A splay tree is a self-adjusting binary search tree with the additional property that recently accessed elements are quick to access again. Splaying the tree for a certain element rearranges the tree so that the element is placed at the root of the tree. One way to do this is to first perform a standard binary tree search for the

element in *Question*, and then use tree rotations in a specific fashion to bring the element to the top. Frequently accessed nodes will move nearer to the root where they can be accessed more quickly. When a node x is accessed, a splay operation is performed on x to move it to the root. To perform a splay operation we carry out a sequence of splay steps, each of which moves x closer to the root.

It is particularly useful for implementing caches and garbage collection algorithms. The garbage collector, or just collector, attempts to reclaim garbage (ie memory occupied by objects that are no longer in use by the program)

In computer engineering, a cache is a component that stores data so that future requests for that data can be served faster. The data that is stored within a cache might be values that have been computed earlier or copies of original values that are stored elsewhere. If requested data is contained in the cache (cache hit), this request can be served by simply reading the cache, which is comparatively faster. Otherwise (cache miss), the data has to be recomputed or fetched from its original storage location, which is comparatively slower. Hence, the more requests can be served from the cache the faster the overall system performance is.

Insertion : To insert a node x into a splay tree, we first insert it as with a normal binary search tree. Then we splay the new node x to the top of the tree.

Access : When a node x is accessed, a splay operation is performed on x to move it to the root. Each particular step depends on three factors:

* Whether x is the left or right child of its parent node, p,
* whether p is the root or not, and if not
* whether p is the left or right child of its parent, g (the grandparent of x)

It is important to remember to set gg (the great-grandparent of x) to now point to x after any splay operation. If gg is null, then x obviously is now the root. A tree rotation moves one node up in the tree and one node down. It is used to change the shape of the tree, and in particular to decrease its height by moving smaller subtrees down and larger subtrees up, resulting in improved performance of many tree operations.

Question 70> Eplain programatically how would levearge the library function sprintf().

Solution>

```
#include <stdio.h>
    int main()
    {
    char buff[50], string[100];
    int ret;
    printf("What is your favorite color ??\n");
    scanf("%s",string);
    ret = sprintf(buff, "Your favorite is %s !! Mine is too\n", string); // (Doubt: What does ret stores ??)
    printf("%s", buff);
    return 0;
    }
```

O/p:
What is your favorite color ??
Green
Your favorite is Green !! Mine is too

Question 71> Solve the famous Maximum Sub Array Problem.

Solution>
1. This code doesn't works if the array contains only -ve numbers.

This sol is Kadane's Algorithm is an O(n) algorithm for finding the maximum contiguous subsequence in a one-dimensional sequence.
We are given an array of positive and negative numbers . We need to find the sub array with the maximum possible sum.
eg; -1 3 -2 9 -6 5 // indexing starts from 0
Here the maximum sub array is from index 2 to 4 {3, -2, 9} with sum 10. Here is the O(n) time complexity *Solution* to the problem.
DEFINE:
x : the final start index of the maximum sub-array // Finally it will contain 1
y : the final end index of the maximum sub-array // Finally it will contain 3
x1 : the temporary start index of the maximum sub-array // It will change before it settles finally at 1
y1 : the temporary end index of the maximum sub-array // It will change before it settles finally at 3
tsum : the temporary maximum sum
sum : the final max sum
x1 = 0, sum = -INFINITY, tsum = 0.
for(y1=0;y1<n;y1++) // run y1 from the start to the end of the array
{
tsum = tsum + A[y1]
if (tsum > sum)
{ sum = tsum;
x = x1;
y = y1;
}

if (tsum < 0)
{
tsum = 0;
x1 = y1 + 1;
}
}

At each step we add the latest array value to tsum and check if this temporary sum is greater than the “Final Sum”. If yes, then we set the final sum to this value and also set x and y to x1 and y1 respectively.

Working Code

```
#include<stdio.h>
int main()
{
   int x1 = 0, y1, sum = -9999, tsum = 0, x, y;
   int a[] = { -1, -2, 4, -1, -1 };
   for(y1=0;y1<5;y1++)     // run y1 from the start to the end of the array
    {
      tsum = tsum + a[y1];
      if (tsum > sum)
          {
             sum = tsum;
             x = x1;
                 y = y1;
       }
          if (tsum < 0)
          {
           tsum = 0;
           x1 = y1 + 1;
      }
    }
  printf("\n Max Sum : %d  IndexL  %d   IndexU  %d", sum, x, y);
  return 0;
}
```

2. This code works even for array of -ve numbers. But does not prints the indexes of the array

---

```
#include<stdio.h>
int main()
{
   int a[] = { 0, -2, -4, -3, -5 };
       int result = a[0];
       int curSum = a[0]; int i;
       for(i=1; i < 5; i++ )
        {
                    if (curSum > 0) curSum += a[i] ;
                    else  curSum = a[i] ;
       if (curSum > result) result = curSum;
        }
```

```
   printf("MaxSum   :%d\n", result);
   return 0;
}
```

Question 72> Write code to find the middle of a linked list.

Solution>

```
void midList(struct node* head)
{
   struct node *fast, *slow;
   int count = 0;
      slow = fast = head;
      while(fast->next && fast->next->next)
      {
                fast=fast->next->next;
                slow=slow->next;
                count++;
      }
      if(count%2 == 0) // odd no of elements
   {
       printf("\n Middle value : %f",slow->d);
       return;
      }
      else        // even no of elements
   {
       printf("\n Middle value : %f",(slow->d + slow->next->d)/2);
       return;
      }
}
```

Question 73> An array having 2N elements is given. Print it in a given format w/o using any extra memory.
eg if 2N = 8 ; I/P Array : 1, 2, 3, 4, 5, 6, 7, 8
O/P Array : 1, 5, 2, 6, 3, 7, 4, 8

Solution>

```
1, 2, 3,  4, 5 , 6, 7, 8
          |----|  // rotate it
```

```
                1, 2, 3,  5, 4 , 6, 7, 8
          |----|  |----| // rotate it
          1, 2, 5,  3, 6 , 4, 7, 8
                  |----| |----| |----| // rotate it
                1, 5, 2, 6, 3, 7, 4, 8   // Final Output
```

Code

```
#include<stdio.h>
int main()
{
int a[8]={1,2,3,4,5,6,7,8};
int i, j, k, tmp;
int n, t;
static int count = 0;
printf("\nEnter no of elements ");
scanf("%d", &n);
printf("\nEnter Elements\n");
for(i=0;i<n;i++)
scanf("%d  ",&a[i]);
printf("\nOriginal Array\n");
for(i=0;i<n;i++)
printf("%d  ",a[i]);
for(i=n/2-1;i>=1;i--)   // 1 less than half the no. of elements
 {
   count ++;
   t = count;
   j = i;
  while(t>=1)         // in 1st iteration, 1 swap is req, in 2nd iteration, 2 swaps are req and so on ...
  {
       k=a[j];
       a[j]=a[j+1];
       a[j+1]=k;
       j=j+2;
       t--;
  }
}
printf("\nResult\n");
for(i=0;i<n;i++)
printf("%d  ",a[i]);
return 0;
}
```

Question 74> Given a binary tree. Convert it into a doubly linked list. The list must be as if the tree is traversed in a zig-zag order from top to botton (left to right in one level and right to level in the next). Write an algorithm for the same.

Solution>

```
Use 2 stacks
stack s1,s2
s1.push(root)
{
       while(s1!=null)
       {
    s = s1.pop()
    Add s to doubly linked list
        s2.push(s.left)
        s2.push(s.right)
       }
       while(s2!=null)
       {
        s = s2.pop()
        Add s to doubly linked list
        s1.push(s.right)
        s1.push(s.left)
       }
}
```

Question 75> Write a C program to reverse a doubly linked list?

Solution>

```
#include<stdio.h>
#include<ctype.h>
typedef struct node
{
 int value;
 struct node *next;
 struct node *prev;
}nodeptr;
nodeptr *head, *tail;
```

```
  void insert(int value);
  void print_list();
  void reverse();
int main()
{
  head=NULL;
  tail=NULL;
  insert(1);
  insert(2);
  insert(3);
  insert(4);
  insert(5);
  print_list();
  reverse();
  print_list();
  return(0);
}
  void insert(int value)
  {
   nodeptr *temp, *cur;
   temp = (nodeptr*) malloc(sizeof(nodeptr));
   temp->next=NULL;
   temp->prev=NULL;
   if(head == NULL)
   {
     printf("\nAdding a head pointer\n");
     head=temp;
     tail=temp;
     temp->value = value;
   }
  else
   {
     for(cur = head; cur->next!= NULL;cur = cur->next); // empty loop
     cur->next=temp;
     temp->prev=cur;
     temp->value=value;
     tail=temp;
   }
  }
  void print_list()
  {
```

```
  nodeptr*temp;
  for(temp=head;temp!=NULL;temp=temp->next)
    printf("%d\t",temp->value);
}
void reverse()
{
nodeptr *cur, *temp, *save_next;
if(head == tail)
     return;
if(head == NULL || tail == NULL)
     return;
for(cur=head;cur!=NULL;)
 {
  temp = cur->next;
  save_next = cur->next;
  cur->next = cur->prev;
  cur->prev = temp;
  cur = save_next;
 }
temp=head;
head=tail;
tail=temp;
}
```

Question 76> Given only a pointer to a node to be deleted in a singly linked list, how can we delete that node?

Solution>

The *Solution* to this is to copy the data from the next node into this node and then delete that next node.

```
// Let's say we have input pointer to del_node
void delete_node()
{
nodeptr *temp;
temp = del_node->next;
del_node->value = temp->value;
del_node->next = temp->next;
free(temp);
}
```

NOTE: This will not work if the node to be deleted is the last node. In such a case, Mark it as a dummy node.

Question 77> Suggest an approach to add two numbers. Each number is represented as a linked list.

L1 : 9 -> 1 -> 8 -> 2 -> NULL

L2 : 5 -> 1 -> 7 -> NULL

Their data's should be added i.e. 9182 + 517 = 9699. This should be in a linked list format.

Res : 9 -> 6 -> 9 -> 9 -> NULL

Solution>

Take a variable say NUM and travese through the list and add each node's Info to NUM

NUM = 0 ;

For (NODE != NULL)

NUM = (NUM *10) + NUM ;

Then add the two numbers obtained (as previously mentioned). Then the result(RES) can be achieved by splitting the RES(TEMP = RES % 10 and RES = RES / 10), and inserting each digit in a new linked list. Now reverse the linked list.

Question 78> Check if a linked list is palindrome (This ques also finds middle of a linked list)

Solution>

```
/* Program to check if a linked list is palindrome */
#include<stdio.h>
#include<stdlib.h>
struct node
{
   char data;
   struct node* next;
};
void reverse(struct node**);
int compareLists(struct node*, struct node *);
/* Function to check if given linked list is
  palindrome or not */
int isPalindrome(struct node *head)
{
  struct node *slow_ptr = head;
  struct node *fast_ptr = head;
  struct node *second_half;
  struct node *prev_of_slow_ptr = head;
  char res;
```

```
  if(head!=NULL)
  {
     /* Get the middle of the list. Move slow_ptr by 1
       and fast_ptrr by 2 */
     while((fast_ptr->next)!=NULL &&
           (fast_ptr->next->next)!=NULL)
     {
       fast_ptr = fast_ptr->next->next;
       /*We need previous of the slow_ptr for
        linked lists  with odd elements */
       prev_of_slow_ptr = slow_ptr;
       slow_ptr = slow_ptr->next;
     }
     /* Case where we have even no of elements */
     if(fast_ptr->next != NULL)
     {
      second_half = slow_ptr->next;
      reverse(&second_half);
      slow_ptr->next = NULL;
      res = compareLists(head, second_half);
      /*construct the original list back*/
      reverse(&second_half);
      slow_ptr->next = second_half;
     }
     /* Case where we have odd no. of elements. Neither first
        nor second list should have the middle element */
     else
     {
       second_half = slow_ptr->next;
       prev_of_slow_ptr->next = NULL;
       reverse(&second_half);
       res = compareLists(head, second_half);
      /*construct the original list back*/
      reverse(&second_half);
      prev_of_slow_ptr->next = slow_ptr;
      slow_ptr->next = second_half;
     }
     return res;
  }
}
void reverse(struct node** headRef)
```

```
{
  struct node* first;
  struct node* rest;
  if (*headRef == NULL) return; // empty list base case
  first = *headRef; // suppose first = {1, 2, 3}
  rest = first->next; // rest = {2, 3}
  if (rest == NULL) return;
  reverse(&rest); // Recursively reverse the smaller {2, 3} case
  first->next->next = first; // put the first elem on the end of the list
  first->next = NULL; // (tricky step -- make a drawing)
  *headRef = rest; // fix the head pointer
}
int compareLists(struct node* head1, struct node *head2)
{
   struct node* temp1 = head1;
 struct node* temp2 = head2;
   while(temp1 && temp2)
   {
     if(temp1->data == temp2->data)
     {
       temp1 = temp1->next;
       temp2 = temp2->next;
     }
     else return 0;
   }
   /* Both are empty reurn 1*/
   if(temp1 == NULL && temp2 == NULL)
     return 1;
   /* Will reach here when one is NULL
    and other is not */
   return 0;
}
void append ( struct node **, char ) ;
void display ( struct node * ) ;
int main()
{
   struct node *p, *q ; q=NULL;
       p = NULL ;  /* empty linked list */
       append ( &p, 'R' ) ;
       append ( &p, 'A' ) ;
       append ( &p, 'D' ) ;
```

```
        append ( &p, 'A' ) ;
        append ( &p, 'R' ) ;
        system ( "cls" ) ;
        display ( p ) ;
    printf("\n\n");
    int result = isPalindrome(p);
    if(result==1) printf("\nPalindrome\n");
    else printf("\nNot Palindrome\n");
        return 0;
}
/* adds a node at the end of a linked list */
void append ( struct node **q, char ch )
{
        struct node *temp, *r ;
        if ( *q == NULL )  /* if the list is empty, create first node */
        {
                temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
                temp -> data = ch ;
                temp -> next = NULL ;
                *q = temp ;
        }
        else
        {
                temp = *q ;
                /* go to last node */
                while ( temp -> next != NULL )
                        temp = temp -> next ;
                /* add node at the end */
                r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
                r -> data = ch ;
                r -> next = NULL ;
                temp -> next = r ;
        }
}
/* displays the contents of the linked list */
void display ( struct node *q )
{
        /* traverse the entire linked list */
        while ( q != NULL )
        {
                printf ( "%c ", q -> data ) ;
```

```
                q = q -> next ;
        }
        printf ( "\n" ) ;
}
```

Question 79> The diameter of a tree T is the largest of the following quantities :

* the diameter of T's left subtree
* the diameter of T's right subtree
* the longest path between leaves that goes through the root of T

Write a code to find the diameter.

Solution>

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int val;
   struct node* left;
   struct node* right;
};
struct node* newNode(int data);
int height(struct node* node)
{
  if (node==NULL)
    return 0;
  else
  {
    int lHeight = height(node->left);
    int rHeight = height(node->right);
    return (lHeight > rHeight)? (lHeight+1): (rHeight+1);
  }
}
struct node* newNode(int data)
{
 struct node* node = (struct node*)malloc(sizeof(struct node));
 node->val   = data;
 node->left  = NULL;
 node->right = NULL;
 return(node);
```

```
}
int max(int a,int b)
{
   return (a>b?a:b);
}
int diameter(struct node* t)
{
   if (t == NULL) return 0;
   int lheight = height(t->left);
   int rheight = height(t->right);
   int ldiameter = diameter(t->left);
   int rdiameter = diameter(t->right);
   return max(lheight + rheight + 1,max(ldiameter,rdiameter));
}
int main()
{
 int n;
 struct node *root   = newNode(40);
 root->left         = newNode(30);
 root->right        = newNode(65);
 root->left->left    = newNode(22);
 root->left->right   = newNode(38);
 root->right->left   = newNode(78);
 n=diameter(root);
 printf("\nDiameter is  %d \n", n);
 return 0;
}
```

Question 80> Preorder traversal of a binary tree is given. Form the appropriate tree. Tree has the property that all non-leaf nodes have value 'N' and all leaf have value 'L' and each node has either 2 or 0 children.

Solution>

Explanation

For each subtree, count of L will be one more than count of N.

If the tree has only root node then it will be L. If it has N L L then count of L is 2 and of N is 1.

So you can use this thing to find each subtree.

If the given preorder is : N N N L L L N L L

So the root is N

Now call recursively for left subtree where count of L becomes one more than count of N for first time.
And the rest part for right subtree.

```
Node fun(String[] arr)
{
Node n = new Node();
n.value = currentValue; // suppose for starting it is N.
// calculate str1 and str2
n->left = fun(str1); // str1 will be N N L L L
n ->right = fun(str2); //str2 will be N L L
return n;
}
```

Working Code :

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

struct tree { char data;
            struct tree *left;
            struct tree *right;
           };
struct tree* buildNLTree(char preorder[]);
void tree_preorder(struct tree* root)
{
    if(root!=NULL)
    {
                printf("%c  ",root->data);
                tree_preorder(root->left);
                tree_preorder(root->right);
    }
}
int main()
{
   char preorder[] = {'N','N','L','L','L','\0'};
   struct tree *root = NULL;
   root = buildNLTree(preorder);
   tree_preorder(root);
   return 0;
}
struct tree* buildNLTree(char preorder[])
{
int length_p = strlen(preorder);  // find length ie p
```

```
if(length_p<1 )
{
//Wrong input
return NULL;
}
if(length_p==1)
{
//Stop condition for the recursion;
struct tree* node=(struct tree*)malloc(sizeof(struct tree));
node->left=NULL;
node->right=NULL;
node->data=preorder[0];
return node;
}
int n_count=0,l_count=0;
int pos_right_sub_tree;
char newpreorder_left[5] = {'\0','\0','\0','\0','\0'};
char newpreorder_right[5]= {'\0','\0','\0','\0','\0'};
struct tree* node=(struct tree*)malloc(sizeof(struct tree));
node->data=preorder[0];  int i;
for(i=1;i<length_p;i++)
{
  if(preorder[i]=='N')
  n_count++;
  else
  l_count++;
  newpreorder_left[i-1]=preorder[i];
  if(l_count>n_count)
 {
  pos_right_sub_tree=i+1;
  newpreorder_left[i]='\0';
  break;
 }
}
int k=0;  n_count=0; l_count=0;
for(i=pos_right_sub_tree;i<length_p;i++)
{
  if(preorder[i]=='N')
  n_count++;
  else
  l_count++;
```

```
  newpreorder_right[k]=preorder[i]; k++;
  if(l_count>n_count)
 {
  newpreorder_right[k]='\0';
  break;
 }
}
node->left=buildNLTree(newpreorder_left);
node->right=buildNLTree(newpreorder_right);
return node;
}
```

Question 81> Write an algorithm to obtain a mirror of binary tree.

Solution>

```
/*
 Change a tree so that the roles of the
 left and right pointers are swapped at every node.
 So the tree...
      4
     / \
    2   5
   / \
  1   3
 is changed to...
      4
     / \
    5   2
       / \
      3   1
*/
void mirror(struct node* node) {
 if (node==NULL)
 {
   return;
 }
 else
 {
   struct node* temp;
   mirror(node->left);
```

```
   mirror(node->right);
   // swap the pointers in this node
   temp = node->left;
   node->left = node->right;
   node->right = temp;
  }
}
```

Question 82> Write an algorithm to perform Level order traversal in a tree.

Solution>

We won't use recursion because we don't want to traverse one level at a time.

```
// Level order traversal.
void levelOrderTraversal(TNODE *root)
{
  TNODE *queue[100] = {(TNODE *)0}; // Important to initialize!
  int size = 0;
  int queue_pointer = 0;
  while(root)
  {
     printf("[%d] ", root->data);
     if(root->left)
     {
       queue[size++] = root->left;
     }
     if(root->right)
     {
       queue[size++] = root->right;
     }
     root = queue[queue_pointer++];
  }
}
```

Question 83> For a given binary tree, Write a function to find the number of nodes with only 1 child.

Solution>

```
   int getMeThis(struct node *ptr){
   int value = 0;
   if(ptr != NULL)
```

```
    {
   if(ptr->left && !ptr->right)
     value = 1 + getMeThis(ptr->left);
   else if(ptr->right && !ptr->left)
     value = 1 + getMeThis(ptr->right);
   else
     value = getMeThis(ptr->left) + getMeThis(ptr->right);
  }
  return (value);
}
```

Quetion 84> Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path exists.

Solution>

Strategy: Subtract the node value from the sum when recurring down, and check to see if the sum is 0 when you run out of tree.

```
int hasPathSum(struct node* node, int sum)
{
 // return true if we run out of tree and sum == 0
    if (node == NULL)
    {
        return(sum == 0);
       }
   else
  {
     int subSum = sum - node -> data;
    return( hasPathSum(node->left, subSum) || hasPathSum(node->right, subSum) );
  }
}
```

Question 85> Write a code to perform ternary search in an array.

Solution>

```
#include<stdio.h>
void  tsearch(int *a, int i, int j, int k);
int main()
{
```

```
 int a[30],n,i,k;
 printf("\nEnter n: ");
 scanf("%d",&n);
 printf("\nEnter nos in ascending order: ");
 for(i=0;i<n;i++)
    scanf("%d",&a[i]);
printf("Enter no to search: ");
scanf("%d",&k);
tsearch(a,0,n-1,k);
}
void tsearch(int *a, int i, int j, int k)
{
 int m1, m2;
 m1 = (i+j)/3;
 m2 = 2*(i+j)/3;
while(i<=j)
{
   if(k==a[m1])
   {
    printf("\nno found at %d",m1);
    return;
   }
   else  if(k==a[m2])
   {
    printf("\nno found at %d",m2);
    return;
   }
   if(k < a[m1])
   return(tsearch(a,i,m1-1,k));
   if(k > a[m2])
     return(tsearch(a,m2+1,j,k));
   else
     return(tsearch(a,m1+1,m2-1,k));
 }
   printf("\n Element not found \n");
   return;
}
```

Question 86> Wite a program to create a singleton class in C.

Solution>

```
#include <stdlib.h>
struct node
{
   int a;
   int b;
};
struct node* getObject()
{
   static struct node *instance = NULL;
   if(instance == NULL)
   {
      instance = (struct node *)malloc(sizeof(struct node));
      instance->a = 1;
      instance->b = 2;
   }
       return instance;
};
```

A class whose number of instances that can be instantiated is limited to one is called a singleton class. Thus, at any given time only one instance can exist, not more. It is used in implementing Logger classes for logging purposes of an application where there is usually one logger instance required.

Question 87> Suggest an algorithm to implement a queue using two stacks.

Solution>

Take two stacks, let's call them inbox and outbox.

Queue:

- Push the new element onto inbox

Dequeue:

- If outbox is empty, refill it by popping each element from inbox and pushing it onto outbox
- Pop and return the top element from outbox

```
public class Queue<E>
{
   private Stack<E> inbox = new Stack<E>();
   private Stack<E> outbox = new Stack<E>();
   public void queue(E item) {
      inbox.push(item);
   }
```

```
    public E dequeue() {
        if (outbox.isEmpty()) {
            while (!inbox.isEmpty()) {
                outbox.push(inbox.pop());
            }
        }
        return outbox.pop();
    }
}
```

Question 88> Write a program to find the character that is having highest frequency in a given word.

Sample Input:

```
Enter Your number of str
2
Enter Your string No : 1
niraj
The max freq char is n
The max freq char is i
The max freq char is a
The max freq char is j
The max freq char is r
Enter Your string No : 2
aaaarul
The max freq char is a
```

Solution>

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define MAX 20
int length;
typedef struct
{
  int count;
  char letter;
}Hash_Table;
int hash_function(char s);
void find_max(Hash_Table HT[]);
int main()
```

```
{
    int i,index,n;
    Hash_Table HT[MAX];
    char str[MAX];
    printf("Enter Your number of string \n");
    scanf("%d",&n);
    for(int m = 0 ; m < n ; m++)
    {
            printf("Enter Your string No : %d \n",m+1);
            scanf("%s",&str);
            i=0;
            length=strlen(str);
             for(int k=0;k<length;k++)
             {
                   HT[k].count=0;
                   HT[k].letter=' ';
             }
             while(str[i]!='\0')
             {
               index=hash_function(str[i]);
                 if(HT[index].letter==' ')
                    {
                       HT[index].letter=str[i];
                       HT[index].count++;
                    }
                    else if(HT[index].letter==str[i])
                    {
                         HT[index].count++;
                    }
                    else
                    {
                         while((HT[index].letter!=str[i])&&(HT[index].letter!=' '))
                         {
                              index=(index+1)%length;
                         }
                         HT[index].letter=str[i];
                         HT[index].count++;
                    }
                     i++;
             }
            find_max(HT);
```

```
    }
    getch();
}
void find_max(Hash_Table HT[])
{
    int big=0;
    for(int i=0;i<length;i++)
    {
          if(HT[i].count>big)
          big=HT[i].count;
    }
   for(int i=0;i<length;i++)
   {
        if(HT[i].count==big)
         printf("The max freq char is %c \n",HT[i].letter);
   }
}
int hash_function(char c)
{
  int i=c;
  return(i%length);
}
```

Question 89> Given a binary tree. It might be full or some nodes might be missing. Every node has a extra "next" pointer. Make the "next" pointer point to the next cousin/sibling node. Last node's "next" pointer should point to null.

Eg. Given Binary Tree -

```
          1
     2          3
  4    5     6    7
```

1's next = null
2's next = 3
3's next = null
4's next = 5
5's next = 6
6's next = 7
7's next = null

Solution>

```
Q.Enqueue(root), currQsize=1, newQsize=0, currNode=prevNode=NULL;
while(Q is not empty)
{
	while(currQsize)
	{
		currNode=Q.Dqueue();
		if(prevNode)
		{
			prevNode->next=currNode;
		}
		/*enque the childrens of next level and make count of number of node at next level*/
		if(currNode->left)
		{
			Q.Enqueue(currNode->left);
			newQsize++;
		}
		if(currNode->right)
		{
			Q.Enqueue(currNode->right);
			newQsize++;
		}
		currQsize--;
		prevNode=currNode;
	}
	currQsize=newQsize; /*number of node at next level*/
	newQsize=0;
	currNode->next=NULL; /*last node sibling is null*/
	prevNode=NULL;
}
```

Question 90> Find an element in a partially rotated sorted array.

Solution>

```
#include<stdio.h>

int find_pos(int * arr,int start,int end,int item)
{
```

```
   if(start>end) return -1;
   int mid=(start+end)/2;
   if(arr[mid]==item) return mid;
   else
   {
    if(arr[mid+1]<=arr[end])
    {
      if(item>=arr[mid+1] && item<= arr[end])
      {
        return  find_pos(arr,mid+1,end,item);
      }
      else
      {
        return  find_pos(arr,start,mid-1,item);
      }
      }
    else
    {
      if(item>=arr[start] && item<= arr[mid-1])
      {
        return find_pos(arr,start,mid-1,item);
      }
      else
      {
        return find_pos(arr,mid+1,end,item);
      }
       }
     }
   }
int main()
{
int arr[]={2,3,4,5,6,1};
int len=sizeof(arr)/sizeof(arr[0]);
int pos=find_pos(arr,0,len-1,6);
printf("\n%d ",pos);
return 0;
}
```

Question 91> How will you find the point of intersection of two linked list(Y shape)

Solution>

```
void Intersection(struct node* head1, struct node *head2)
{
	// find the length of each linked list
	int len1=Count(head1), len2=Count(head2);
	int diff = (len1>len2)?(len1-len2):len2-len1;
	int remainingLength=0;
	if(len1-len2>0)
	{
	  remainingLength=len1-diff;
	  for(int i=1;i<=diff;i++)
	  head1=head1->next;
	}
		else
	{
	remainingLength=len2-diff;
	for(int i=1;i<=diff;i++)
	 head2=head2->next;
	}
		for(int i=0;i<remainingLength;i++)
	{
	  if(head1==head2)
	  {
	    printf("\nPoint of intersection is at %d ",head1->data);
	    break;
	  }
	  head1=head1->next;
	  head2=head2->next;
	}
}
```

Question 92> Find size of a data type without using sizeof() operator(Implement sizeof() operator).

Solution>

```
#include<stdio.h>
#define SIZEOF(var) (unsigned int)(&var+1) - (unsigned int)(&var)
int main()
{
    char x;
    printf("The size of x is %d\n",SIZEOF(x));
    return 0;
```

}

Question 93> Explain how structure padding happens in C.

Solution>
On dev Compiler
---
int : 4 bytes
char : 1 "
short : 2 "
long : 4 "
float : 4 "
double : 8 "
pointer : 4 "
char variables can appear at any byte boundary. Short variables must be 2 byte aligned, they can appear at any even byte boundary. This means that 0x10004567 is not a valid location for a short variable but 0x10004566 is. long variables must be 8 byte aligned, they can only appear at byte boundaries that are a multiple of 8 bytes. This means that 0x10004566 is not a valid location for a long variable but 0x10004568 is.
Structure padding occurs because the members of the structure must appear at the correect byte boundary, to achieve this the compiler puts in padding bytes (or bits if bit fields are in use) so that the structure members appear in the correct location. Additionally the size of the structure must be such that in an array of the structures all the structures are correctly aligned in memory so there may be padding bytes at the end of the structure too.
struct example
{
char c1;
short s1;
char c2;
long l1;
char c3;
}
c1 can appear at any byte boundary, however s1 must appear at a 2 byte boundary so there is 1 padding byte between c1 and s1. c2 can then appear in the available memory location, however l1 must be at a 4 byte boundary so there are 3 padding bytes between c2 and l1. c3 then appear in the available memory location, however because the structure contains a long member the structure must be 4 byte aligned and must be a multiple of 4 bytes in size. Therefore there are 3 padding bytes at the end of the structure. It would appear in memory in this order :
c1
padding byte

s1 byte 1
s1 byte 2
c2
padding byte
padding byte
padding byte
l1 byte 1
l1 byte 2
l1 byte 3
l1 byte 4
c3
padding byte
padding byte
padding byte
The structure would be 16 bytes long.
re-written like this
struct example
{
long l1;
short s1;
char c1;
char c2;
char c3;
}
Then l1 appears at the correct byte alignment, s1 will be correctly aligned so no need for padding between l1 and s1. c1, c2, c3 can appear at any location. The structure must be a multiple of 4 bytes in size since it contains a long so 3 padding bytes appear after c3 are added.
It appears in memory in the order
l1 byte 1
l1 byte 2
l1 byte 3
l1 byte 4
s1 byte 1
s1 byte 2
c1
c2
c3
padding byte
padding byte
padding byte
and is only 12 bytes long.

Note : Structure padding is platform and compiler dependent.

eg 1:

```
struct  MyStructA
{
   char a;
   char b;
   int  c;
};
struct MyStructB {
   char a;
   int  c;
   char b;
};
```

Results compiling with dev : In MyStructA we have (1 byte + 1 byte + 2 bytes of padding + 4 bytes = 8bytes). In MyStructB we have (1 byte + 3 bytes of padding + 4 bytes + 1 byte + 3 bytes of padding = 12 bytes)

This example illustrates a good rule of thumb. Generally, it is good to group structure fields of the same type together to minimize the extra padding and thus space.

The simple answer is compilers pad structures to optimize data transfers. This is an hardware architecture issue. Most modern CPUs perform best when fundemental types, like 'int' or 'float', are algined on memory boundaries . Many architectures don't allow misaligned access or if they allow, they do inccur a performance penalty.

When a compiler processes a structure declaration it will add extra bytes between fields to meet alignment needs.

Byte padding will be done by compiler to reduce number of machine cycles needed to read the value. If the integers are stored in the address which is divisible by its size (say 4) then CPU can read whole 4 bytes in a single cycle other wise cpu cycles required will be more; hence byte padding will be done.

Question 94> You are given a Doubly Link List with one pointer of each node pointing to the next node just like in a singly linked list. The second pointer however can point to any node in the list and not just the previous node. Now write a algorithm in O(n) time to duplicate this list. Let us call the second pointer as arbit pointer as it can point to any arbitrary node in the linked list.

Solution>

1> Create the copy of 1st node and insert it between 1st & 2nd node ; Create the copy of 2nd node and insert it between 2nd & 3rd node ; Continue in this fashion and add a copy of last node after last (i.e Nth) node.

2> Now copy the arbitrary link in this fashion

```
Original->next->arbitrary = Original->arbitrary->next;
traverse two nodes
```

3> Now restore the Original and copy linked lists in this fashion in a single loop.

```
Original->next = Original->next->next;
```

```
  copy->next = copy->next->next;
```
4> Make sure that last element of original->next is NULL.
Time Complexity: O(n)
Space Complexity: O(1)

Question 95> Write an efficient program to check if two strings are anagrams . eg orchestra = carthorse

Solution>

```
/*
 This function takes two ASCII strings in C syntax (null-terminated character arrays) and determines whether they are anagrams or not.
 The function contains a histogram that stores the frequency of each character it encounters in the first string. For each character of the second string, it decrements the corresponding entry in the histogram. If before every decrement, the value of the histogram bucket reaches zero, then the two strings are not anagrams, as there is some character that has more occurrences in the second string than in the first string.
 If either of the two strings contains a non-alphabetic character, the anagram property is considered to be violated.

 Remarks:
 It is an efficient implementation because:
 It is in-place, the original strings are not copied .
 Returns: 1 if the strings are anagrams, 0 otherwise.
*/
#include<stdio.h>
int is_anagram (char w1[], char w2[])
{
	unsigned int i, sz;
	/* The histogram */
	int freqtbl[26];
	/* Sanity check */
	if ((sz = strlen(w1)) != strlen(w2))
		return 0;
	/* Initialize the histogram */
	for(i=0;i<26;i++)
	freqtbl[i]=0;
	/* Read the first string, incrementing the corresponding histogram entry */
	for (i = 0; i < sz; i++)
  {
		if (w1[i] >= 'A' && w1[i] <= 'Z')
```

```
                        freqtbl[w1[i]-'A']++;
                else if (w1[i] >= 'a' && w1[i] <= 'z')
            freqtbl[w1[i]-'a']++;
                else
                        return 0;
        }
        /* Read the second string, decrementing the corresponding histogram entry */
        for (i = 0; i < sz; i++)
    {
                if (w2[i] >= 'A' && w2[i] <= 'Z')
        {
                        if (freqtbl[w2[i]-'A'] == 0)
                                return 0;
                        freqtbl[w2[i]-'A']--;
                }
        else if (w2[i] >= 'a' && w2[i] <= 'z')
        {
                        if (freqtbl[w2[i]-'a'] == 0)
                                return 0;
                        freqtbl[w2[i]-'a']--;
                }
        else
        {
                        return 0;
                }
        }
        return 1;
}
int main()
{
    int res;
    char s1[] ="NirajKumar";
    char s2[] ="KumarNiraj";
    res = is_anagram(s1,s2);
    printf("\nResult : %d " ,res);
    return 0;
}
```

Question 96> Write a program to perform Inorder Tree Traversal without recursion and without stack (Morris Traversal).

Solution>

Using Morris Traversal, we can traverse the tree without using stack and recursion. The idea of Morris Traversal is based on Threaded Binary Tree. In this traversal, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore original tree.

```
#include<stdio.h>
struct tNode
{
  int data;
  struct tNode* left;
  struct tNode* right;
};
/* Function to traverse binary tree without recursion and
  without stack */
void MorrisTraversal(struct tNode *root)
{
 struct tNode *current,*pre;
 if(root == NULL)
   return;
 current = root;
 while(current != NULL)
 {
  if(current->left == NULL)
  {
   printf(" %d ", current->data);
   current = current->right;
  }
  else
  {
   /* Find the inorder predecessor of current */
   pre = current->left;
   while(pre->right != NULL && pre->right != current)
     pre = pre->right;
   /* Make current as right child of its inorder predecessor */
   if(pre->right == NULL)
   {
     pre->right = current;
     current = current->left;
   }
```

```
    /* Revert the changes made in if part to restore the original
      tree i.e., fix the right child of predecssor */
    else
    {
      pre->right = NULL;
      printf(" %d ",current->data);
      current = current->right;
    } /* End of if condition pre->right == NULL */
   }
  } // End of while
} // end of func
struct tNode* newtNode(int data)
{
  struct tNode* tNode = (struct tNode*)malloc(sizeof(struct tNode));
  tNode->data = data;
  tNode->left = NULL;
  tNode->right = NULL;
  return(tNode);
}
int main()
{
  /* Constructed binary tree is
          1
        /   \
       2     3
     /  \
    4    5
                      */
  struct tNode *root = newtNode(1);
  root->left        = newtNode(2);
  root->right       = newtNode(3);
  root->left->left   = newtNode(4);
  root->left->right  = newtNode(5);
  MorrisTraversal(root);
  getchar();
  return 0;
}
```

Question 97> What is the time complexity of the following code?

```
int fib(int n)
{
  if(n==0||n==1)
  {
    return 1;
  }
return fib(n-1)+fib(n-2);
}
```

Solution> fib(n) = fib(n-1) + fib(n-2)

Tn = Tn-1 + Tn-2 // n , n-1 , n-2 are subscripts

x^n - x^(n-1) - x^(n-2) = 0

x^(n-2) (x^2 - x - 1) = 0

x^(n-2) is not equal to zero , so (x^2 - x - 1) = 0

solve x1 = (1+sq root 5)/2 , x2 = (1-sq root 5)/2

so, *Solution* is C1(X1^n) + C2(X2^n) ; find C1 and C2 from boundary condition of fibonacci series.

Question 98> Write a function that returns true if a binary tree is a mirror image of itself.

Solution>

Let us write the code to check if a tree is a mirror image of another tree and then we can just pass the same root as two different trees and process them.

```
int isMirror(struct node *root1, struct node *root2)
{
   // If both roots (in this case, the same one) are null, then return true.
      if(root1==null && root2==null)
                return 1;
   // If only one of them is NULL , then return 0
      // (You can delete this case for our case since we will be passing the same root)
      else if(root1==null || root2==null)
                return 0;
   // If the value of the root in not equal, return false
      else if(root1.value != root2.value)
                return 0;
   // Now call the function on left and right recursively
      return isMirror(root1.left,root2.right) && isMirrorImage(root1.right,root2.left);
}
```

Question 99> int x = 2 is given. Using only this value how will you obtain 37?

Solution>

```
x = 2;
int one = x/x;
int five = x << one + one;
int thirtyseven = one << five + five;
```

Question 100> Explain the differences between Typedef and #define

Solution>

A typedef is just a new name for an already existing type. defines are handled by the preprocessor while typedefs are handled by the C compiler itself.

You can extend a macro typename with other type specifiers, but not a typedef typename. That is,

```
#include<stdio.h>
#define myint int
unsigned myint i;     /* Works Fine */
typedef int yourint;
unsigned yourint j;   /* Illegal    */
```

A typedef provides the type for every declarator in a declaration.

```
#include<stdio.h>
#define double_ptr1 double *
int main()
{
  double_ptr1 a, b; // After macro expansion, the second line effectively becomes: double * a, b;
  // a is a pointer-to-a-double while b is a double variable.
  typedef double* double_ptr2;
  double_ptr2 c, d;
  printf("\n%d",sizeof(a));
  printf("\n%d",sizeof(b));
  printf("\n%d",sizeof(c));
  printf("\n%d",sizeof(d));
  return 0;
}
```

Output :

```
4
8
4
4
```

Question 101> Explain Big Endian vs Little Endian.

Solution>
Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first. For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001). In a little-endian system, it would be stored as 524F (52 at address 1000, 4F at 1001).

Question 102> Explain Volatile Keyword in C.

Solution>

```
int * x;
...
void func( )
{
 * x = 1;
 /* OTHER INSTRUCTIONS */
 * x = 2;
}
...
func( );
```

If the comment section marked "OTHER INSTRUCTIONS" in above example never used the value pointed by "x", then the first assignment of * x = 1 is probably not required and could be removed. Yet a compiler which did not perform optimization would not recognize this fact and * x = 1 command would still end up being included in the executable.
End result of these optimizations is more efficient executables and a smaller required memory footprint.
Consider what happens if the memory pointed by variable "x" was a control line for an external device. Also imagine that sending a value of 1 to that memory told our ficticious device to begin some operation and that sending a value of 2 told it to stop. If the compiler did optimization and removed the * x = 1 instruction then the external device would never receive a start signal. *Solution* is to use the "volatile" type qualifier in the definition of the variable in *Question*.
When the keyword volatile is used in the type definition it is giving an indication to the compiler on how it should handle the variable. Primarily it is telling the compiler that the value of the variable may change at any

time as a result of actions external to the program. Once the compiler knows this it will also know that no operations involving the variable should be optimized out of the code no matter how extraneous they may appear to the optimization algorithm.

Question 103> Explain the concept of Virtual Func

Solution>
In object-oriented programming, a virtual function or virtual method is a function or method whose behaviour can be overridden within an inheriting class by a function with the same signature.

In OOP when a derived class inherits a base class, an object of the derived class may be referred to (or cast) as either being the base class type or the derived class type. If there are base class methods overridden by the derived class, the method call behaviour is ambiguous.
The distinction between virtual and non-virtual resolves this ambiguity. If the function in *Question* is designated virtual in the base class then the derived class' function would be called (if it exists). If it is not virtual, the base class' function would be called.
eg; class Animal {

```
public:
   virtual void eat() {
      std::cout << "I eat like a generic animal.\n";
   }
   virtual ~Animal() {
   }
};
class Wolf : public Animal {
public:
   void eat() {
      std::cout << "I eat like a wolf!\n";
   }
};
class Fish : public Animal
{
public:
   void eat() {
      std::cout << "I eat like a fish!\n";
   }
};
class GoldFish : public Fish {
public:
```

```
    void eat() {
        std::cout << "I eat like a goldfish!\n";
    }
};
class OtherAnimal : public Animal {
};
```

A pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived class. Classes containing pure virtual methods are termed "abstract" ; they cannot be instantiated directly. A subclass of an abstract class can only be instantiated directly if all inherited pure virtual methods have been implemented by that class or a parent class. Pure virtual methods typically have a declaration (signature) and no definition (implementation).

Question 104> Explain the concept of Operator Overloading

Solution>

An Example of Operator Overloading

```
Complex a(1.2, 1.3);     //this class is used to represent complex numbers
Complex b(2.1, 3);       //notice the construction taking 2 parameters for the real and imaginary part
Complex c = a+b;         //for this to work the addition operator must be overloaded
```

The addition without having overloaded operator + could look like this:

```
Complex c = a.Add(b);
```

This piece of code is not as readable as the first example

```
class Complex
{
public:
        Complex(double re,double im)
          { real = re; imag = im; }
        Complex operator+(const Complex& other);
        Complex operator=(const Complex& other);
private:
        double real;
        double imag;
};
Complex Complex::operator+(const Complex&  other)
{
    double result_real = real + other.real;
    double result_imaginary = imag + other.imag;
    return Complex( result_real, result_imaginary );
}
```

Question 105> You stand before a 100-story building with two eggs. Using only these two eggs, you must figure out the highest floor from which you can drop an egg such that the egg won't break when it hits the ground.

Solution>
Let us say 16 is my answer. That I need 16 drops to find out the answer. Let us see whether we can find out the height in 16 drops. First we drop from height 16, and if it breaks we try all floors from 1 to 15.
Let us take the case with 16 as the answer
16 if breaks at 16 checks from 1 to 15 in 15 drops
16 + 14 + 1 = 31 if breaks at 31 checks from 17 to 30 in 14 drops
31 + 13 + 1 = 45 .....
45 + 12 + 1 = 58
58 + 11 + 1 = 70
70 + 10 + 1 = 81
81 + 09 + 1 = 91
91 + 08 + 1 = 100
Now we can see that we could have done it in either 15 or 14 drops only but how can we find the optimal one.
From the above table we can see that the optimal one will be needing 0 linear trials in the last step.
So we could write it as
(1+p) + (1+(p-1))+ (1+(p-2)) + ...+ (1+0) >= 100.
Let 1+p = q which is the answer we are looking for
q (q+1)/2 >=100
Solving for 100 you get q = 14.
So the answer is: 14

Question 106> You are given a function. When it is called, it returns 0 with 60% probability, and 1 with 40% probability. Now using this function, write a new function that returns 0 with 50% probability and 1 with 50% probability.

Solution>
f1 = 0 with 60% probability.
= 1 with 40% probability.
If we call f1() two times and say that :
0,1 = 0 (60% * 40%)
1,0 = 1 (40% * 60%)
(0,0) and (1,1) = Continue
int function f2()

```
{
 int ret = -1;
 while(ret == -1)
 {
       int res1 = f1();
       int res2 = f1();
       if(res1 == 0 && res2 == 1) ret = 0;
       else if(res1 == 1 && res2 == 0) ret = 1;
 }
return ret
}
```

It gives 24% probability to return 1, 24% to return 0 and 52% probability to recurse.

Question 107> WAP to find whether a m/c is little endian or big endian.

Solution>

```
#include <stdio.h>
int main()
{
   unsigned int n = 1;
   char *p;
   p = (char*)&n;
   if (*p == 1)
     printf("Little Endian\n");
   else if (*(p + sizeof(int) - 1) == 1)
     printf("Big Endian\n");
   return 0;
}
```

The program stores an unsigned integer 1, assuming that either the first or last byte of that integer will be 1, and all the other bytes will be 0, then checks to see which byte was set to 1.

Each byte-order system has its advantages. Little-endian machines let you read the lowest-byte first, without reading the others. You can check whether a number is odd or even (last bit is 0) very easily, which is cool if you're into that kind of thing. Big-endian systems store data in memory the same way we humans think about data (left-to-right), which makes low-level debugging easier.

Question 108> Suggest algorithms for rotating a two dimensional matrix.

Solution>
O(n^2) time and O(1) space algorithm (without any workarounds and hanky-panky stuff!)
Rotate by +90:
 Transpose
 Reverse each row
Rotate by -90:
 Transpose
 Reverse each column
Rotate by +180:
 Method 1: Rotate by +90 twice
 Method 2: Reverse each row and then reverse each column
Rotate by -180:
 Method 1: Rotate by -90 twice
 Method 2: Reverse each column and then reverse each row
 Method 3: Reverse by +180 as they are same

Question 109> Explain the concept of smart pointers

Solution> Smart pointer is the same as a normal pointer, but it provides automatic memory management. It avoids issues like dangling pointers, memory leaks.
We need a reference count variable that is incremented when we add a new reference to the object and decremented when we remove a reference.

```
template <class T> class SmartPointer {
 T *ref;
 unsigned *ref_count;
 public:
  SmartPointer(T * ptr) {
    ref = ptr;
    ref_count = (unsigned*)malloc(sizeof(unsigned));
    *ref_count = 1;
  }
  SmartPointer(SmartPointer<T> & sptr) {
    ref = sptr.ref;
    ref_count = sptr.ref_count;
    ++(*ref_count);
   }
  ~SmartPointer() {
     remove(); // Remove one reference to object.
  }
  T getValue() {
```

```
  return *ref;
 }
 void remove() {
  --(*ref_count);
  if (*ref_count == 0) {
   delete ref;
   free(ref_count);
   ref = NULL;
   ref_count = NULL;
  }
 } // end of remove()
}
```

Question 110> You are given a 2D array of characters and a character pattern. WAP to find if pattern is present in 2D array. Pattern can be in any way (all 8 neighbors to be considered) but you can't use same character twice while matching. Return 1 if match is found, 0 if not.

Solution>

```
# grid m * n is a 2D array of characters
# visited is a m * n bool array
# say from a given position (x,y) if you want to move to one of the diagnals you just need to add (1,1) or (-1, -1) to (x y)
directions = [(-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (0, -1), (-1, -1), (1, -1)]
bool is_present(grid, x, y, str, index, visited)
   if len(str) == index: # we reached the end
         return true
   else:
        found =  false
        visited[x][y] = true
        for d in directions:
          dx, dy = d
                nextx = x + dx
       nexty = y + dy
       if is_valid(nextx, nexty, m, n) && not visited[nextx][nexty]: # m * n 2D grid
                found = found || is_present(grid, nextx, nexty, str, index + 1) # look for next char
        if not found:
          visited[x][y] = false
      return found
      # in the below code we reuse a char any no of times. so we don't need to keep track of the cell we visited.
```

```
      # so in the grid written below, the word 'amazon' is present but the previous code will return false.
      # a m p
      # z o n
bool is_present(grid, x, y, str, index)
  if len(str) == index: # we reached the end
       return true
  else:
       found =  false
       for d in directions:
         dx, dy = d
                nextx = x + dx
       nexty = y + dy
       if is_valid(nextx, nexty, m, n): # m * n grid
                found = found || is_present(grid, nextx, nexty, str, index + 1) # look for next char
      return found
      # just call this function for every index in the grid.
is_present(grid, 1, 1, str, 0)
```

Question 111> At a movie theater, the manager announces that a free ticket will be given to the first person in line whose birthday is the same as someone in line who has already bought a ticket. You have the option of getting in line at any time. Assuming that you don't know anyone else's birthday, and that birthdays are uniformly distributed throughout a 365 day year, what position in line gives you the best chance of being the first duplicate birthday?

Solution>

The probability p(n), of getting a free ticket when you are the nth person in line is:

(probability that none of the first n-1 people share a birthday) . (probability that you share a birthday with one of the first n-1 people)

So p(n) = [365/365 · 364/365 · 363/365 ·.. (365-((n-1)-1))/365] . [(n-1)/365]

We seek the least n such that p(n) > p(n+1), or p(n)/p(n+1) > 1

p(n)/p(n+1) = 365/(366-n) . (n-1)/n

Now, p(n)/p(n+1) > 1 implies 365n - 365 > 366n - n^2

so n^2 - n - 365 > 0

An approximate factorization of this quadratic equationgives: (n + 18.6)(n - 19.6) > 0

Rejecting the negative region, inequality is satisfied if n > 19.6. Therefore the position in line that gives you the best chance of being the first duplicate birthday is 20th.

Question 112> Two old friends, Jack and Bill, meet after a long time.

Three kids

Jack: Hey, how are you man?

Bill: Not bad, got married and I have three kids now.

Jack: That's awesome. How old are they?

Bill: The product of their ages is 72 and the sum of their ages is the same as your birth date.

Jack: Cool … But I still don't know.

Bill: My eldest kid just started taking piano lessons.

Jack: Oh now I get it.

How old are Bill's kids?

Solution> Let's break it down. The product of their ages is 72. So what are the possible choices?

2, 2, 18 – sum(2, 2, 18) = 22

2, 4, 9 – sum(2, 4, 9) = 15

2, 6, 6 – sum(2, 6, 6) = 14

2, 3, 12 – sum(2, 3, 12) = 17

3, 4, 6 – sum(3, 4, 6) = 13

3, 3, 8 – sum(3, 3, 8) = 14

1, 8, 9 – sum(1, 8, 9) = 18

1, 3, 24 – sum(1, 3, 24) = 28

1, 4, 18 – sum(1, 4, 18) = 23

1, 2, 36 – sum(1, 2, 36) = 39

1, 6, 12 – sum(1, 6, 12) = 19

The sum of their ages is the same as your birth date. That could be anything from 1 to 31 but the fact that Jack was unable to find out the ages, it means there are two or more combinations with the same sum. From the choices above, only two of them are possible now.

2, 6, 6 – sum(2, 6, 6) = 14

3, 3, 8 – sum(3, 3, 8) = 14

Since the eldest kid is taking piano lessons, we can eliminate combination 1 since there are two eldest ones. The answer is 3, 3 and 8.

About the Author

Niraj Agarwal *worked as Software Development Engineer at Amazon Corporation, India. Prior to joining Amazon, he worked at Oracle Corporation (Server Technology) as Member of Technical Staff (Software Developer 2).*

He received his Bachelors in Engineering degree from Birla Institute of Technology Mesra in 2009. He was born and brought up in Ranchi, Jharkhand.

He can be contacted at niraj.ece05@gmail.com