Question 1. There is a colony of 8 cells arranged in a straight line where each day every cell competes with its adjacent cells (neighbour). Each day, for each cell, if its neighbours are both active and both inactive, the cell becomes inactive the next day, otherwise it becomes active the next day.

Assumptions: The two cells on the ends have single adjacent cell, so the other adjacent cell can be assumed to be always inactive.

Even after updating the cell state. Consider its previous state for updating the state of other cells. Update the cell information of all cells simultaneously.

Write a function cell Compete which takes one 8 element array of integer's cells representing the current state of 8 cells and one integer days representing the number of days to simulate.

An integer value of 1 represents an active cell and value of 0 represents an inactive cell.

Existing Program

```
int* cellCompete(int* cells,int days)
{/
/write your code here
}
//function signature ends
```

Test Cases

TESTCASES 1:

INPUT:

[1,0,0,0,0,1,0,0],1

EXPECTED RETURN VALUE:

[0,1,0,0,1,0,1,0]

TESTCASE 2:

INPUT:

[1,1,1,0,1,1,1,1,],2

EXPECTED RETURN VALUE:

[0,0,0,0,0,1,1,0]

Solution:

```
#include<iostream>
using namespace std;
 int cellCompete(int *cells ,int day){
//write your code here
 for (int i=0;i<day;i++){
 cells[-1]=0; //assumptions
cells[8]=0;//assumptions
int u[8]; //another array to copy value
for (int i=-1;i<9;i++){
u[i]=cells[i];
}
for(int j=0;j<8;j++){
if(u[j-1]==u[j+1]){ //comparing the value of the neighbouring cells of u[]
cells[j]=0; //changing value of cell according to condition
}
else
cells[j]=1;
} }
for (int i=0;i<8;i++){
cout<<cells[i];
}
return 0;}
int main(){ //main function
int days,cells[]={1,0,0,0,0,1,0,0}; //array to pass through function
```

```
int *cellsptr=cells; //creating array values to pointer
cout<<"enter days:"; //for days
cin>>days;
cout<<"n[1,0,0,0,0,1,0,0]n";
cellCompete(cellsptr, days); //passing to function
return 0;
}
```

Question 2.

Mooshak the mouse has been placed in a maze. There is a huge chunk of cheese somewhere in the maze. The maze is represented as a two-dimensional array of integers, where 0 represents walls.1 represents paths where Mooshak can move and 9 represents the huge chunk of cheese. Mooshak starts in the top left corner at 0.
Write a method is Path of class Maze Path to determine if Mooshak can reach the huge chunk of cheese. The input to is Path consists of a two dimensional array and for the maze matrix. The method should return 1 if there is a path from Mooshak to the cheese. And 0 if not Mooshak is not allowed to leave the maze or climb on walls.

EX: 8 by 8(8*8) matrix maze where Mooshak can get the cheese.

```
1 0 1 1 1 0 0 1
1 0 0 0 1 1 1 1
1 0 0 0 0 0 0 0
1 0 1 0 9 0 1 1
1 1 1 0 1 0 0 1
1 0 1 0 1 1 0 1
1 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1
```

Test Cases:
Case 1:
Input:[[1,1,1,][9,1,1],[0,1,0]]
Expected return value :1

Explanation:
The piece of cheese is placed at(1,0) on the grid Mooshak can move from (0,0) to (1,0) to reach it or can move from (0,0) to (0,1) to (1,1) to (1,0)
Test case 2:
Input: [[0,0,0],[9,1,1],[0,1,1]]
Expected return value: 0

Explanation:
Mooshak cannot move anywhere as there exists a wall right on (0,0)

Existing Program

```
/*include header files needed by your program
some library functionality may be restricted
define any function needed
function signature begins, this function is required*/
Int isPath(int **grid,int m,int n)
{/*
write your code here
*/}
/function signature ends
```

Solution:

```
if (x,y outside maze) return false
if (x,y is goal) return true
if (x,y not open) return false
mark x,y as part of solution path
if (FIND-PATH(North of x,y) == true) return true
if (FIND-PATH(East of x,y) == true) return true
if (FIND-PATH(South of x,y) == true) return true
if (FIND-PATH(West of x,y) == true) return true
unmark x,y as part of solution path
return false*/
```

```
public boolean findPath(int x, int y){
// check x,y are outside maze.
if(x < 0 || x >= mazelength || y < 0 || y >= mazelength ){
return false;
}
if(maze[x][y] == 9) {
return true;
}
if(maze[x][y] != 1) return false;
//mark x, y as part of the solution path
if(maze[x][y] == 1){
maze[x][y] = 3;
}
// move North
if( findPath(x-1,y)){
return true;
}
//move East
if( findPath(x,y+1)) return true;
//move South
if( findPath(x+1,y)) return true;
//move West
if( findPath(x,y-1)) return true;
// unMark x,y as part of the solution.
maze[x][y] = 0;
return false;
}
public void printSolution(){
System.out.println("Final Solution ::::::: ");
for(int i=0;i<mazelength;i++){
for(int j=0;j<mazelength;j++){
System.out.print(" "+maze[i][j]+" ");
}
System.out.println();
}
}
public static void main(String args[]){
```

```
RatMazeProblem ratMazeProblem = new RatMazeProblem();
System.out.println(" is Path exist ::: "+ratMazeProblem.findPath(0,0));
ratMazeProblem.printSolution();
}
}
```

Question 3. The least recently used (LRU) cache algorithm exists the element from the cache(when it's full) that was least recently used. After an element is requested from the cache, it should be added to the cache (if not already there) and considered the most recently used element in the cache.

Initially, the cache is empty. The input to the function LruCountMiss shall consist of an integer max_cache_size, an array pages and its length Len
The function should return an integer for the number of cache misses using the LRU cache algorithm.
Assume that the array pages always have pages numbered from 1 to 50.
TEST CASES:

TEST CASE1:
INPUT:
3,[7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0],16
EXPECTED RETURN VALUE:
11

TESTCASE 2:
INPUT:
2,[2,3,1,3,2,1,4,3,2],9
EXPECTED RETURN VALUE:
8

EXPLANATION:
The following page numbers are missed one after the other 2,3,1,2,1,4,3,2.This results in 8 page misses.

Existing Program

```
int lruCountMiss(int max_cache_size, int *pages,int len)
{/
/write tour code
}
```

Solved Program

```
#include<iostream.h>
using namespace std;
int lruCountMiss(int max_cache_size, int *pages,int len)
{
int miss=0,cache[max_cache_size]; /*variable miss and cache declared*/
for (int i=0;i&lt;max_cache_size;i++){
/*this is for clearing value of cache i.e. to empty cache. I used any value here*/
cache[i]=0x87787;
}
for (int i=0;i&lt;len;i++){ /*for loop for all the value from list one by one*/
for(int j=0;j&lt;max_cache_size;j++){
/*loop to check the cache value, if it matches the value*/
if (pages[i]==cache[j]){
for (int k=i; k&lt;max_cache_size;k++){
/*if the value is already present in the cache then shifting values from the present value.*/
 cache[i]=cache[i+1];
}
cache[max_cache_size-1]=pages[i]; /*updating the last value of cache*/
break;
}
else if(j==(max_cache_size-1)){
for (int l=0; l&lt;max_cache_size;l++){
/*if the value is not present in the cache then shifting values from starting.*/
cache[l]=cache[l+1];
}
cache[max_cache_size-1]=pages[i];
/*updating the last value of cache*/
 miss++;
}
}}
```

```
return miss;}/*returning the Miss*/
int main(){ /*main function*/
cout&lt;&lt; "We are checking two cases.\n"&lt;&lt;"Case 1:t"&lt;&lt;"Array values are
[7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0] nand cache size= 3; and total values to check are 16: n We got the
miss. ";
int days,pages[]={7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0};
/*array to pass through function*/
int *cellsptr=pages;
/*creating array values to pointer*/
cout&lt;&lt;"Miss =t"&lt;&lt;lruCountMiss(3,cellsptr,16);//passing to function
cout&lt;&lt;"nnCase 2:t"&lt;&lt;"Array values are [2,3,1,3,2,1,4,3,2] nand cache size= 2; and total
values to check are 9: n We got the miss. ";
int pages2[]={2,3,1,3,2,1,4,3,2};
int *cellsptr2=pages2;
/*creating array values to pointer*/
cout&lt;&lt;"Miss =t"&lt;&lt;lruCountMiss(2,cellsptr2,9);//passing to function
return 0;
}
```

Ques. 4 Write a function to insert an integer into a circular linked _list whose elements are sorted in ascending order 9smallest to largest). The input to the function insert Sorted List is a pointer start to some node in the circular list and an integer n between 0 and 100. Return a pointer to the newly inserted node...

The structure to follow for a node of the circular linked list is

```
Struct CNode ;
Typedef struct CNode cnode;
Struct CNode
{I
nt value;
Cnode* next;
};C
node* insertSortedList (cnode* start,int n
{/
/WRITE YOUR CODE HERE
}/
```

/FUNCTION SIGNATURE ENDS

Test Case 1:

Input:
[3>4>6>1>2>^],5
Expected Return Value:
[5>6>1>2>3>4>^]
Test Case 2:

Input:
[1>2>3>4>5>^],0
Expected Return Value:
[0>1>2>3>4>5>^]
Given the maximum size of the cache and a list of integers (to request from the cache), calculate the number of cache misses using the LRU cache algorithm. A cache miss occur when the requested integer does not exist in the cache.

Solution

```
Cnode* insertSortedList (cnode* start,int n)
{
   int count=1,i,count1=0;

   Cnode* temp=start;
   Cnode* temp1=start;
   Cnode* temp2=start;
   Cnode* newHead=NULL;

   // One principle with circular linked list is that ->next is never NULL
   while(temp!=NULL)
   {
      temp=temp->next;
      count ++;
   }
```

```
   int arr[count]; // you can declare a dynamic array this way, toy need to use apointer and malloc
   // One principle with circular linked list is that there is no need to copy it to insert a new
element somewhere
   arr[0]=n;
   while(temp2!=NULL)
   {
      for(j=1;j<count;j++)
      {
         arr[j]=temp->data;
         temp2=temp2->next;
      }
   }

   // wrong logic here
   while(n>=temp2->value)
   {
      temp2=temp2->next;
   }

   newHead=temp2;

   for(i=0;i<count;i++)
   {
      Cnode* newNode=new Cnode();

      newNode->data=arr[i];
      newNode->next=NULL;
   }
   return newNode;
}
```

Or

```
Cnode* insertSortedList (cnode* start,int n)
{
int count=1,i,count1=0;
```

```
Cnode* temp=start;
Cnode* temp1=start;
Cnode* temp2=start;
Cnode* newHead=NULL;

while(temp!=NULL)
{
temp=temp->next;
count ++;
}

int arr[count];

while(temp2!=NULL)
{
for(j=0;j<count;j++)
{
arr[j]=temp->data;
temp2=temp2->next;
}
}


while(n>=temp2->value)
{
temp2=temp2->next;
}

newHead=temp2;

for(i=0;i<count;i++)
{
Cnode* newNode=new Cnode();

newNode->data=arr[i];
newNode->next=NULL;
```

```
}

}
```

Question 5. Removal of vowel from string

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

int check_vowel(char);

int main()
{
  char s[100], t[100];
  int i, j = 0;

  printf("Enter a string to delete vowels\n");
  gets(s);

  for(i = 0; s[i] != '\0'; i++) {
    if(check_vowel(s[i]) == 0) {       //not a vowel
      t[j] = s[i];
      j++;
    }
  }

  t[j] = '\0';

  strcpy(s, t);    //We are changing initial string

  printf("String after deleting vowels: %s\n", s);

  return 0;
}


int check_vowel(char c)
```

```
{
  switch(c) {
    case 'a':
    case 'A':
    case 'e':
    case 'E':
    case 'i':
    case 'I':
    case 'o':
    case 'O':
    case 'u':
    case 'U':
      return 1;
    default:
      return 0;
  }
}
```

Question 6. GCD of two numbers.

```
#include <stdio.h>

// Recursive function to return gcd of a and b
int gcd(int a, int b)
{
    // Everything divides 0
    if (a == 0 || b == 0)
       return 0;

    // base case
    if (a == b)
        return a;

    // a is greater
    if (a > b)
        return gcd(a-b, b);
```

```
    return gcd(a, b-a);
}

// Driver program to test above function
int main()
{
    int a = 98, b = 56;
    printf("GCD of %d and %d is %d ", a, b, gcd(a, b));
    return 0;
}
```

Or

```
class Test
{
    // Recursive function to return gcd of a and b
    static int gcd(int a, int b)
    {
        // Everything divides 0
        if (a == 0 || b == 0)
            return 0;

        // base case
        if (a == b)
            return a;

        // a is greater
        if (a > b)
            return gcd(a-b, b);
        return gcd(a, b-a);
    }

    // Driver method
    public static void main(String[] args)
    {
```

```
    int a = 98, b = 56;
    System.out.println("GCD of " + a +" and " + b + " is " + gcd(a, b));
  }
}
```

Question 7. Eliminate repeated elements in Array.

Solution -

```
#include<stdio.h>

int main() {
  int arr[20], i, j, k, size;

  printf("\nEnter array size : ");
  scanf("%d", &size);

  printf("\nAccept Numbers : ");
  for (i = 0; i < size; i++)
    scanf("%d", &arr[i]);

  printf("\nArray with Unique list  : ");
  for (i = 0; i < size; i++) {
    for (j = i + 1; j < size;) {
      if (arr[j] == arr[i]) {
        for (k = j; k < size; k++) {
          arr[k] = arr[k + 1];
        }
        size--;
      } else
        j++;
    }
  }

  for (i = 0; i < size; i++) {
```

```
    printf("%d ", arr[i]);
  }

  return (0);
}
```

OutPut -

```
Enter array size : 5
Accept Numbers : 1 3 4 5 3
Array with Unique list  : 1 3 4 5
```

C program to print reverse Pyramid star pattern

Example

Input

Input rows: 5
Output

```
*********
 *******
  *****
   ***
    *
```

Logic to print reverse pyramid star pattern

```
*********
 *******
  *****
   ***
    *
```

The above pattern has `N` (in this case 5) rows. Each row has exactly `N * 2 - (i * 2 - 1)` stars. In addition the pattern consists of leading spaces. For each row it contain `i - 1` leading space (where `i` is current row number).
Step by step descriptive logic to print reverse pyramid star pattern.

1. Input number of rows to print from user. Store it in a variable say `rows`.
2. To iterate through rows, run an outer loop from 1 to `rows`. The loop structure should look like `for(i=1; i<=rows; i++)`.
3. To print spaces, run an inner loop from 1 to `i - 1`. The loop structure should look like `for(j=1; j<i; j++)`. Inside this loop print single space.
4. To print stars, run another inner loop from 1 to `rows * 2 - (i * 2 - 1)`. The loop structure should look like `for(j=1; j<= (rows*2 - (i*2-1)); j++)`. Inside this loop print star.
5. After printing all stars for each row, move to next line i.e. print new line.

```
/**
 * C program to print reverse pyramid star pattern
 */

#include <stdio.h>

int main()
{
    int i, j, rows;

    /* Input rows to print from user */
    printf("Enter number of rows : ");
    scanf("%d", &rows);

    for(i=1; i<=rows; i++)
    {
        /* Print leading spaces */
        for(j=1; j<i; j++)
        {
            printf(" ");
        }
```

```
        /* Print stars */
        for(j=1; j<=(rows*2 -(2*i-1)); j++)
        {
            printf("*");
        }

        /* Move to next line */
        printf("\n");
    }

    return 0;
}
```

C program to print half diamond star pattern

Example
Input

Output

```
*
**
***
****
*****
****
***
**
*
```

Logic to print half diamond star pattern

```
*
**
```

```
***
****
*****
****
***
**
*
```

The above pattern consist of `N * 2 - 1`rows. For each row columns are in increasing order till `N`th row. After `N`th row columns are printed in descending order.
Step by step descriptive logic to print half diamond star pattern.

1. Input number of columns to print from user. Store it in a variable say `N`.
2. Declare a variable as loop counter for each column, say `columns = 1`.
3. To iterate through rows, run an outer loop from 1 to `N * 2 - 1`. The loop structure should look like `for(i=1; i<N*2; i++)`.
4. To iterate through columns, run an inner loop from 1 to `columns`. The loop structure should look like `for(j=1; j<=columns; j++)`. Inside this loop print star.
5. After printing all columns of a row, move to next line.
6. After inner loop check `if(i <= N)` then increment `columns` otherwise decrement by 1.

Program to print half diamond star pattern

```
/**
 * C program to print half diamond star pattern series.
 */

#include<stdio.h>

int main()
{
    int i, j, N, columns;

    /* Input number of columns from user */
    printf("Enter number of columns:");
    scanf("%d",&N);

    columns=1;
```

```
    for(i=1;i<N*2;i++)
    {
        for(j=1; j<=columns; j++)
        {
            printf("*");
        }

        if(i < N)
        {
            /* Increment number of columns per row for upper part */
            columns++;
        }
        else
        {
            /* Decrement number of columns per row for lower part */
            columns--;
        }

        /* Move to next line */
        printf("\n");
    }

    return 0;
}
```

C program to print square or rectangle star pattern

Example
Input

Output

```
*****
*****
*****
*****
*****
```

Logic to print square star pattern

```
*****
*****
*****
*****
*****
```

Have a close look to the pattern for a minute so that you can think a little basic things about the pattern.

The pattern is a matrix of `N` rows and columns containing stars(asterisks). Here, you need to iterate through `N` rows, and for each row iterate for `N` columns.

Step by step descriptive logic to print the square number pattern.

1. Input number of rows from user. Store it in some variable say `N`.
2. To iterate through rows, run an outer loop from 1 to `N`. The loop structure should be similar to `for(i=1; i<=N; i++)`.
3. To iterate through columns, run an inner loop from 1 to `N`. Define a loop inside above loop with structure `for(j=1; j<=N; j++)`.
4. Inside inner loop print `*`.
5. After printing all columns of a row move to next line i.e. print a new line.

Let us implement the given pattern in C program.

Program to print square star pattern

```
/**
 * C program to print square star pattern
 */

#include <stdio.h>

int main()
{
    int i, j, N;

    /* Input number of rows from user */
```

```
    printf("Enter number of rows: ");
    scanf("%d", &N);

    /* Iterate through N rows */
    for(i=1; i<=N; i++)
    {
        /* Iterate over columns */
        for(j=1; j<=N; j++)
        {
            /* Print star for each column */
            printf("*");
        }

        /* Move to the next line/row */
        printf("\n");
    }

    return 0;
}
```

Logic to print rectangle star pattern

```
**********
**********
**********
**********
**********
```

Step by step descriptive logic to print rectangle star pattern.

1. Input number of rows and columns from user. Store it in a variable say `rows` and `columns`.
2. To iterate through rows, run an outer loop from 1 to `rows`. Define a loop with structure `for(i=1; i<=rows; i++)`.
3. To iterate through columns, run an inner loop from 1 to `columns`. Define a loop with structure `for(j=1; j<=columns; j++)`.
4. Inside inner loop print star `*`.
5. After printing all columns of a row. Move to next line i.e. print a new line.

Program to print rectangle star pattern

```
/**
 * C program to print rectangle star pattern
 */

#include <stdio.h>

int main()
{
    int i, j, rows, columns;

    /* Input rows and columns from user */
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &columns);

    /* Iterate through each row */
    for(i=1; i<=rows; i++)
    {
        /* Iterate through each column */
        for(j=1; j<=columns; j++)
        {
            /* For each column print star */
            printf("*");
        }

        /* Move to the next line/row */
        printf("\n");
    }

    return 0;
}
```

C program to print mirrored right triangle star pattern

Example
Input

Output

```
   *
   **
  ***
 ****
*****
```

Logic to print mirrored right triangle star pattern

```
   *
   **
  ***
 ****
*****
```

The above pattern is similar to right triangle star pattern with leading spaces.
Leading spaces in the above pattern are in decreasing order i.e. first row contains `n-1` space, second contains `n-2`space and so on. Point your mouse cursor over the pattern to count total spaces per row.
Step by step descriptive logic to print mirrored right triangle star pattern.

1. Input number of rows to print from user. Store it in some variable say `rows`.
2. To iterate through rows run an inner loop from 1 to `rows`. The loop structure should look like `for(i=1; i<=rows; i++)`.
3. To print spaces, run an inner loop from `i` to `rows` with loop structure `for(j=i;j<=rows;j++)`. Inside this loop print single space.

4. To print stars run another inner loop from 1 to i, with loop structure `for(j=1; j<=i; j++)`.
5. After printing all columns of a row, move to next line i.e. print new line.

Program to print mirrored right triangle star pattern

```
/**
 * C program to print mirrored right triangle star pattern series
 */

#include <stdio.h>

int main()
{
    int i, j, rows;

    /* Input rows from user */
    printf("Enter number of rows: ");
    scanf("%d", &rows);

    /* Iterate through rows */
    for(i=1; i<=rows; i++)
    {
        /* Print spaces in decreasing order of row */
        for(j=i; j<rows; j++)
        {
            printf(" ");
        }

        /* Print star in increasing order or row */
        for(j=1; j<=i; j++)
        {
            printf("*");
        }

        /* Move to next line */
        printf("\n");
```

```
    }

    return 0;
}
```

C program to print Equilateral triangle (Pyramid) star pattern

Example
Input

Output

```
    *
   ***
  *****
 *******
*********
```

Logic to print pyramid star pattern

```
    *
   ***
  *****
 *******
*********
```

Before you read further have a close look at the above pattern. The pattern consists of `N` (for this case 5) rows. Each row contain exactly `2 * N - 1` stars. In addition to stars the pattern has leading spaces. Each row contain `N - i` spaces (where `i` is current row number). To count total spaces per row point your mouse over the above pattern.
Step by step descriptive logic to print Pyramid star pattern.

1. Input number of rows to print from user. Store it in a variable say `rows`.

2. To iterate through rows, run an outer loop from 1 to rows. The loop structure should look like for(i=1; i<=rows; i++).
3. To print spaces, run an inner loop from i to rows - 1. The loop structure should look like for(j=i; j<rows; j++). Inside this loop print single space.
4. **Note:** Iterating from 1 to N - i or i to rows - 1 both are equal.
5. To print star, run another inner loop from 1 to 2 * i - 1. The loop structure should look like for(j=1; j<=(2*i - 1); j++). Inside this loop print star.
6. After printing stars for current row, move to next line i.e. print new line.

Program to print pyramid star pattern series

```
/**
 * C program to print equilateral triangle or pyramid star pattern
 */

#include <stdio.h>

int main()
{
    int i, j, rows;

    /* Input number of rows to print */
    printf("Enter number of rows : ");
    scanf("%d", &rows);

    /* Iterate through rows */
    for(i=1; i<=rows; i++)
    {
        /* Print leading spaces */
        for(j=i; j<rows; j++)
        {
            printf(" ");
        }

        /* Print star */
        for(j=1; j<=(2*i-1); j++)
        {
            printf("*");
```

```
        }

        /* Move to next line */
        printf("\n");
    }

    return 0;
}
```

C program to print hollow inverted pyramid star pattern

Example
Input

Output

```
*********
 *       *
  *     *
   *   *
    * *
     *
```

Logic to print hollow inverted pyramid star pattern

```
*********
 *   *
  * *
   *
```

The above pattern is similar to inverted pyramid star pattern except spaces are printed in center of the pyramid.
Step by step descriptive logic to print hollow pyramid star pattern.

1. Input number of rows to print from user. Store it in a variable say rows.
2. To iterate through rows, run an outer loop from 1 to rows. The loop structure should look like for(i=1; i<=rows; i++).
3. To print leading spaces, run an inner loop from 1 to i - 1. The loop structure should look like for(j=1; j<i; j++). Inside this loop print single space.
4. To print stars, run another inner loop from 1 to rows * 2 - (i * 2 - 1). The loop structure should look like for(j=1; j<=(rows*2-(i*2-1)); j++).
5. Inside this loop print star for ith and last column and for first row otherwise print space.
6. After printing all columns of a row, move to next line i.e. print new line.

Program to print hollow inverted pyramid star pattern

```
/**
 * C program to print hollow inverted pyramid star pattern
 */

#include <stdio.h>

int main()
{
    int i, j, rows;

    /* Input rows to print from user */
    printf("Enter number of rows: ");
    scanf("%d", &rows);

    /* Iterate through rows */
    for(i=1; i<=rows; i++)
    {
        /* Print leading spaces */
        for(j=1; j<i; j++)
```

```
        {
            printf(" ");
        }

        /* Print hollow pyramid */
        for(j=1; j<=(rows*2 - (2*i-1)); j++)
        {
            /*
             * Print star for first row(i==1),
             * ith column (j==1) and for
             * last column (rows*2-(2*i-1))
             */
            if(i==1 || j==1 || j==(rows*2 - (2*i - 1)))
            {
                printf("*");
            }
            else
            {
                printf(" ");
            }
        }

        /* Move to next line */
        printf("\n");
    }

    return 0;
}
```

GCD of more than two (or array) numbers

Given an array of numbers, find GCD of the array elements. In a previous post we find GCD of two number.

Examples:

```
Input  : arr[] = {1, 2, 3}
Output : 1

Input  : arr[] = {2, 4, 6, 8}
```

Output : 2

he GCD of three or more numbers equals the product of the prime factors common to all the numbers, but it can also be calculated by repeatedly taking the GCDs of pairs of numbers.

```
gcd(a, b, c) = gcd(a, gcd(b, c))
             = gcd(gcd(a, b), c)
             = gcd(gcd(a, c), b)
```

For an array of elements, we do following.

```
result = arr[0]
For i = 1 to n-1
  result = GCD(result, arr[i])
```

Below is C++ implementation of above idea.

```
// C++ program to find GCD of two or
// more numbers
#include<bits/stdc++.h>
using namespace std;

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b%a, a);
}

// Function to find gcd of array of
// numbers
int findGCD(int arr[], int n)
{
    int result = arr[0];
    for (int i=1; i<n; i++)
        result = gcd(arr[i], result);
```

```
    return result;
}

// Driven code
int main()
{
    int arr[] = {2, 4, 6, 8, 16};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << findGCD(arr, n) << endl;
    return 0;
}
```

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) –> (**1 4** 2 5 8)

(1 **4** **2** 5 8) –> (1 **2** **4** 5 8), Swap since 4 > 2

(1 2 **4** **5** 8) –> (1 2 **4** **5** 8)

(1 2 4 **5** **8**) –> (1 2 4 **5** **8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1** **2** 4 5 8) –> (**1** **2** 4 5 8)

(1 **2** **4** 5 8) –> (1 **2** **4** 5 8)

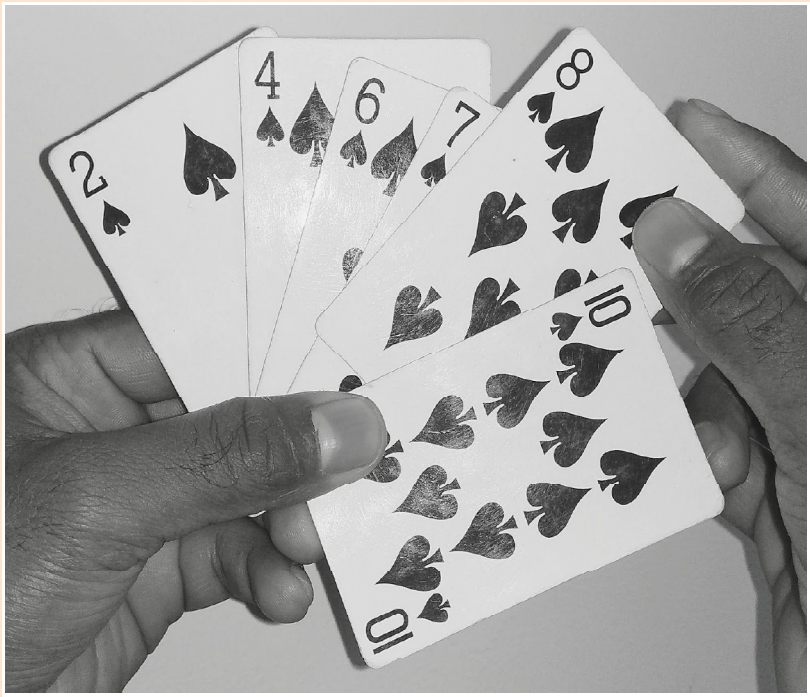
(1 2 **4** **5** 8) –> (1 2 **4** **5** 8)

(1 2 4 **5** **8**) –> (1 2 4 **5** **8**)

```
// C program for implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Insertion Sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.



Algorithm

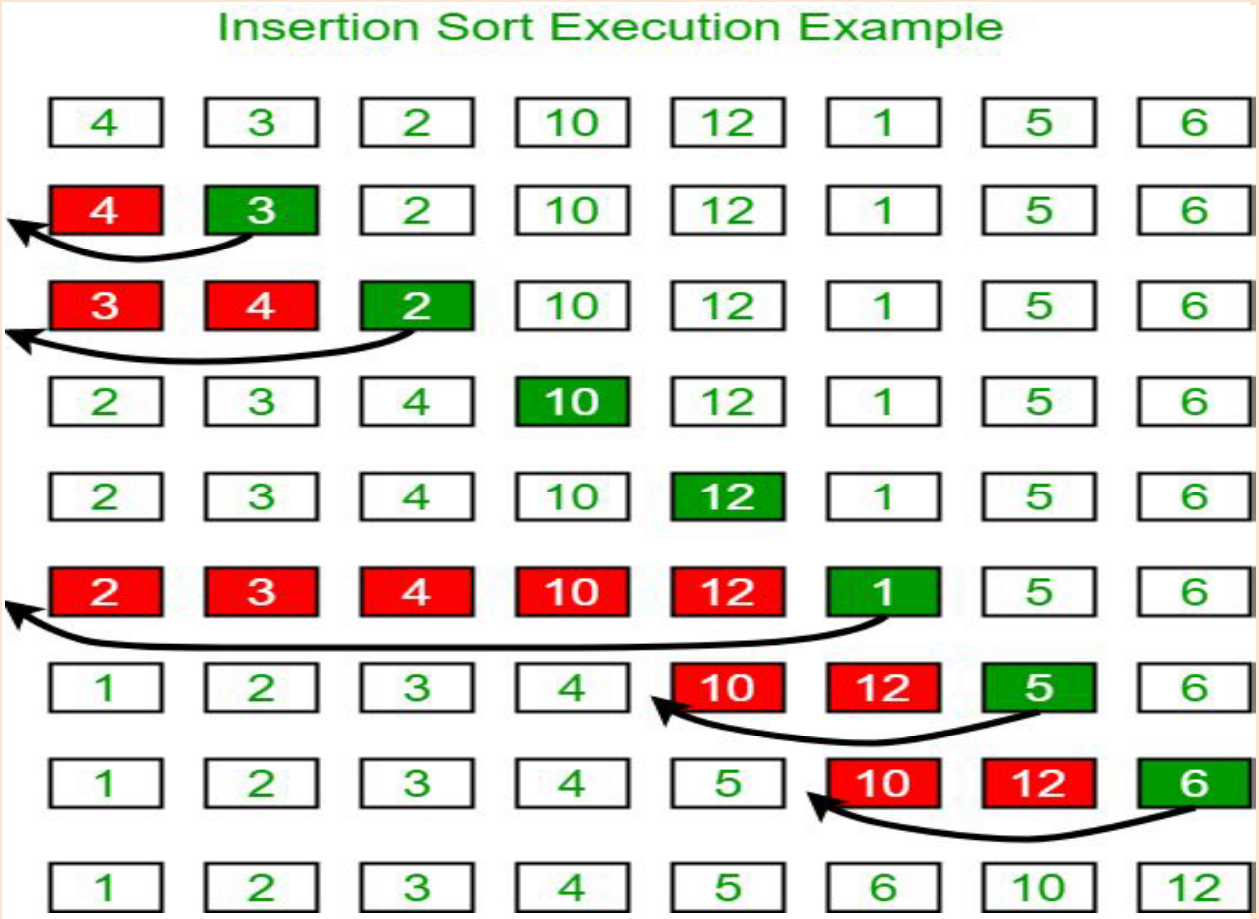
// Sort an arr[] of size n

insertionSort(arr, n)

Loop from i = 1 to n-1.

……a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]

Example:



Another Example:

12, 11, 13, 5, 6

Let us loop for **i** = 1 (second element of the array) to 5 (Size of input array)

i = 1. Since 11 is smaller than 12, move 12 and insert 11 before 12

11, 12, 13, 5, 6

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13

11, 12, 13, 5, 6

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

5, 11, 12, 13, 6

i = 4. 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.

5, 6, 11, 12, 13

```
// C program for insertion sort
#include <stdio.h>
#include <math.h>

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
       key = arr[i];
       j = i-1;

       /* Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position */
       while (j >= 0 && arr[j] > key)
       {
           arr[j+1] = arr[j];
           j = j-1;
       }
       arr[j+1] = key;
   }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
   int i;
   for (i=0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}



/* Driver program to test insertion sort */
int main()
```

```
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.

2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:

arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 **12** 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 **22** 25 64

// Find the minimum element in arr[3...4]

// and place it at beginning of arr[3...4]

11 12 22 **25** 64

```
// C program for implementation of selection sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
```

```
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Find whether array is a palindrome or not

Using loop

Loop over the array and compare the element from left end of the array with the corresponding element from the right end of the array. At any point, if the elements do not match, terminate the loop. If the whole loop completed, then the array is palindrome.

```
public boolean isPalindrome(int[] array) {
        int length = array.length;
        for (int index = 0; index < array.length; index++) {
                // get the element from the start
                int start = array[index];
                // get corresponding element from end
                int end = array[--length];
                // check if elements till the middle have been compared
                if (length < index) {
                        return true;
                }
                // if start element is not the same as end element, the array is not
                // palindrome
                if (start != end) {
                        return false;
                }
        }
        // if the control reaches here, means all the elements were same
return true;
}
```

Using Recursion

This approach also works same as the above but difference is that it does not use any loop to traverse the array. Instead it uses recursion, which means the method calls itself till all the elements are compared.

This method also compares element at the left end of the array with the corresponding element at the right end. It accepts the array to be checked, left(or start) index and right(or end) index as arguments. If elements at both index are same, then it calls itself with start index incremented and end index decreased by 1 till start index becomes greater than the end index. If the elements at any iteration are not equal, then the method returns and recursion is terminated.

```
public boolean isPalindrome(int[] array, int startIndex, int endIndex) {
    // if array is empty or has 1 element, it is palindrome
    if (array.length == 0 || array.length == 1)
        return true;
    // check if start index is greater than end index, which means
    // whole array has been checked
    if (startIndex >= endIndex)
        return true;
    // check if element from left is not equal to element from last,
    // then array is not palindrome
    if (array[startIndex] != array[endIndex])
        return false;
    // check for palindrome with start index incremented and end index
    // decreased by 1
    return isPalindrome(array, startIndex + 1, endIndex - 1);
}
```

2.4

Given an array of integers, write a function that returns true if there is a triplet (a, b, c) that satisfies $a^2 + b^2 = c^2$.

Example:

Input: arr[] = {3, 1, 4, 6, 5}
Output: True
There is a Pythagorean triplet (3, 4, 5).

Input: arr[] = {10, 4, 6, 12, 5}
Output: False
There is no Pythagorean triplet.

```
/ A C++ program that returns true if there is a Pythagorean
// Triplet in a given aray.
#include <iostream>
using namespace std;

// Returns true if there is Pythagorean triplet in ar[0..n-1]
bool isTriplet(int ar[], int n)
{
    for (int i=0; i<n; i++)
    {
       for (int j=i+1; j<n; j++)
       {
          for (int k=j+1; k<n; k++)
          {
            // Calculate square of array elements
            int x = ar[i]*ar[i], y = ar[j]*ar[j], z = ar[k]*ar[k];

            if (x == y + z || y == x + z || z == x + y)
                 return true;
          }
       }
    }

    // If we reach here, no triplet found
    return false;
}

/* Driver program to test above function */
int main()
{
    int ar[] = {3, 1, 4, 6, 5};
    int ar_size = sizeof(ar)/sizeof(ar[0]);
    isTriplet(ar, ar_size)? cout << "Yes": cout << "No";
```

```
    return 0;
}
```

Question. Write a function that accepts a sentence as a parameter, and returns the same with each of its words reversed. The returned sentence should have 1 blank space between each pair of words. Example:
Parameter: “jack and jill went up a hill”
Return Value: “kcaj dna llij tnew pu a llih”

```
#include<stdio.h>
#include<string.h>
char *strrev1(char *st);
void main()
{
char st1[30];
char *pt1;
printf("enter the sentence");
gets(st1);
pt1=strrev1(st1);
puts(pt1);
}
char *strrev1(char *st)
{
int i;
char *pt=st;
for(i=0;st[i]!='\0';i++);
st[i]=' ';
st[i+1]='\0';
i=0;
for(;st[i]!='\0';i++)
{
for(;st[i]!=' ';i++);
st[i]='\0';
strrev(pt);
pt=st+i+1;
st[i]=' ';
}
return(st);
}
```

Apart from these you must also practice these programs these were asked in the exams earlier.

14. Pascal triangle program

15. Print following pattern

Input:4

1222

2333

3444

4555

Input:5

12222

23333

34444

45555

56666

Click here to see the program.

16. Display array dissimilar numbers

Given two arrays and need print (A-B)U(B-A)

17. Mirror image of a matrix.

18. To delete a specified a letter from a given word

19. Input is any odd number

a. If n=3

1 2 3 10 11 12

4 5 89

6 7

If n=5

1 2 3 4 5 26 27 28 29 30

6 7 8 9 22 23 24 25

10 11 12 19 20 21

13 14 17 18

15 16

20. Input string : bhanu

Output: bhanu, hanub, anubh, nubha, ubhan

21. Prime numbers upto n numbers

22. ((()) If the no of open and closing braces match ok else print -1 (matching Parenthesis)

23.

1

2 2

3 3 3

4 4 4 4

4 4 4 4

3 3 3

2 2

1

24. Two arrays given and combine them and sorting order.

25. Frequency sorting algorithm

26. Input is- series of numbers(take an array to store)

Ex: test case1:- 16734515888801

Output should be the:-

number which had appeared highest no.of times- how many times it occurred in sequence.

8-4

1-3

5-2

27. tring reversal with out using temporary array

28. Removal of vowels from string with out temporary array

29. String palindrome or not

30. Alternate sorted array

31. merge sort using dynamic memory allocation.

32. multiplication of two matrices and print the result in transverse order

33. Given an array of n elements and we need to find the least number repetition count

a. Example 3 4 4 5 3

i. Least number is 3 and repeated two times so output: 2

34. Pattern

1

2 3

6 5 4

7 8 9 10

35. factorial and hcf programs

36. Swapping the array index and array values

37. Input a number let's say 3 ,.. we need to make a matrix

1 2 3

4 5 6

7 8 9

And its transpose. multiply the matrix with it's transpose and return it

38. Given a number and a digit. We need to find out how many times digit occurred in the number

Example: 12134 and digit is 1

As 1 present 2 times so we need to print 2

39. Pattern program

1 2 3 4

9 10 11 12

13 14 15 16

5 6 7 8

40. Balanced parenthesis. If string is balanced then print number of balanced parenthesis otherwise -1

41. Pattern program

1

2*2

3*3*3

4*4*4*4

4*4*4*4

3*3*3

2*2

1

42. Pattern program

1*2*3*4

9*10*11*12

13*14*15*16

5*6*7*8

43. Cell compete

There is a colony of 8 cells arranged in a straight line where each day every cell competes with its adjacent cells(neighbour).

Each day, for each cell, if its neighbours are both active or both inactive, the cell becomes inactive the next day,. otherwise itbecomes active the next day.

Assumptions:

The two cells on the ends have single adjacent cell, so the other adjacent cell can be assumsed to be always inactive.

Even after updating the cell state. consider its pervious state for updating the state of other cells. Update the cell informationof allcells simultaneously.

Write a fuction cellCompete which takes takes one 8 element array of integers cells representing the current state of 8 cells and one integer days representing te number of days to simulate.

An integer value of 1 represents an active cell and value of 0 represents an inactive cell.

program:

int* cellCompete(int* cells,int days)

{

//write your code here

}

//function signature ends

TESTCASES 1:

INPUT:

[1,0,0,0,0,1,0,0],1

EXPECTED RETURN VALUE:

[0,1,0,0,1,0,1,0]

TESTCASE 2:

INPUT:

[1,1,1,0,1,1,1,1,],2

EXPECTED RETURN VALUE:

[0,0,0,0,0,1,1,0]

Click here to see the program

44. Pattern n=4 and s=3

3

4 4

5 5 5

6 6 6 6

6 6 6 6

5 5 5

4 4

3

45. Pattern

1111112

3222222

3333334

5444444

5555556

7666666

46. Combine two arrays then sort

47. Pattern

1

3*2

4*5*6

10*9*8*7

48. Grey character program

The first question was to test whether an input string of opening and closing parentheses was balanced or not. If yes , return the no. of parentheses. If no, return -1.

The second question was to reverse the second half of an input linked list.

If the input linked list contained odd number of elements , consider the middlemost element too in the second half.

Write a program to print the following program:-

For N=3

3333
4414
5525
6636
3333

Q1. Find the prime numbers in a given range 1 to n all separated by comma.
Eg. n = 10
Output : 2,3,5,7

Q2. Pattern
1
2*2
3*3*3
3*3*3
2*2
1

2nd question I have got in is:-
For given integer N print 2*N lines

1
2*3
4*5*6
7*8*9*10
7*8*9*10
4*5*6
2*3
1