**Inteview**

- Analytic Skill

- Puzzle solving

- Geometry

- Finding the distance between 2 lines given their vector equations.

- Finding the distance of a point from a plane in 3D.

- Techical Skill

- C/C++

- Pointer

- Null pointer and its use / void pointer

- how we make a function type safe.

- Dangling Pointer

- Diff b/w array and pointer

- Diff b/w reference & pointer

- Memory management

- what is static memory allocation & dynamic memory allocation

- What is calloc, alloc & realloc.

- What is the difference b/w calloc & alloc

- Static keyword

- What is static variable

- Use of static variable

- example

- File handling

- Logical Gate and use

- convert\_to\_binary

- Call by value & call by reference

- Recursion

- GCD

- Function pointer

- go to & jump statement

- OOP

- Design  
 - Problem solving

- Reverse a number in recursive & non-recursive way

- In a given matrix find 0 and replace the row & column by 0.

- Solve it using recursion and scan each corner.

- write a program which convert a int value to its binary representation.

- Recursive way also

- 16 is a number and write a program which calculate how many bits it need to store its binary represtation.

- GCD using recursion

- C++

- Why we use const in copy constructor.

- Why we are not using pointer instead of reference in copy constructor.

- virtual mechanism/vtable explain.

- Data Structure

- Write a program which swap alternative node in a double linked list.

- Write a program which implement a stack using two queue.

- How to check whether a linked list is circular one or not.

- Delete middle of linked list

- Decision making

- Refactoring a code segment

- Design a problem/project.

- Do you think, is it possible to make a ‘vector’ like container in c. If yes, how do you design it and what are the things you consider during implement of it.

- Design a class/interface which provide facility to draw any geometrical elements using openGL.

- Design a “Smart Pointer”.

* Basic C/C++
* Const
* Pointer & reference
* Basic C++
* Class
* Ctor
* Copy ctor
* Dtor
* Static member & function
* Type casting
* Virtual
* Free & delete
* Exception Handling
* Threading
* DLLs
* OOPA
* Design Pattern

#include <iostream>

/\*\*

\* The idea with my algorithm is to delay the writing of zeros

\* till all rows and cols can be processed. I do this using

\* recursion:

\* 1) Enter Recursive Function:

\* 2) Check the row and col of this "corner" for zeros and store the results in bools

\* 3) Send recursive function to the next corner

\* 4) When the recursive function returns, use the data we stored in step 2

\* to zero the the row and col conditionally

\*

\* The corners I talk about are just how I ensure I hit all the row's a cols,

\* I progress through the matrix from (0,0) to (1,1) to (2,2) and on to (n,n).

\*

\* For simplicities sake, I use ints instead of individual bits. But I never store

\* anything but 0 or 1 so it's still fair ;)

\*/

// ================================

// Using globals just to keep function

// call syntax as straight forward as possible

int n = 5;

int m[5][5] = {

{ 1, 0, 1, 1, 0 },

{ 0, 1, 1, 1, 0 },

{ 1, 1, 1, 1, 1 },

{ 1, 0, 1, 1, 1 },

{ 1, 1, 1, 1, 1 }

};

// ================================

// Just declaring the function prototypes

void processMatrix();

void processCorner( int cornerIndex );

bool checkRow( int rowIndex );

bool checkCol( int colIndex );

void zeroRow( int rowIndex );

void zeroCol( int colIndex );

void printMatrix();

// This function primes the pump

void processMatrix() {

processCorner( 0 );

}

// Step 1) This is the heart of my recursive algorithm

void processCorner( int cornerIndex ) {

// Step 2) Do the logic processing here and store the results

bool rowZero = checkRow( cornerIndex );

bool colZero = checkCol( cornerIndex );

// Step 3) Now progress through the matrix

int nextCorner = cornerIndex + 1;

if( nextCorner < n )

processCorner( nextCorner );

// Step 4) Finially apply the changes determined earlier

if( colZero )

zeroCol( cornerIndex );

if( rowZero )

zeroRow( cornerIndex );

}

// This function returns whether or not the row contains a zero

bool checkRow( int rowIndex ) {

bool zero = false;

for( int i=0; i<n && !zero; ++i ) {

if( m[ rowIndex ][ i ] == 0 )

zero = true;

}

return zero;

}

// This is just a helper function for zeroing a row

void zeroRow( int rowIndex ) {

for( int i=0; i<n; ++i ) {

m[ rowIndex ][ i ] = 0;

}

}

// This function returns whether or not the col contains a zero

bool checkCol( int colIndex ) {

bool zero = false;

for( int i=0; i<n && !zero; ++i ) {

if( m[ i ][ colIndex ] == 0 )

zero = true;

}

return zero;

}

// This is just a helper function for zeroing a col

void zeroCol( int colIndex ) {

for( int i=0; i<n; ++i ) {

m[ i ][ colIndex ] = 0;

}

}

// Just a helper function for printing our matrix to std::out

void printMatrix() {

std::cout << std::endl;

for( int y=0; y<n; ++y ) {

for( int x=0; x<n; ++x ) {

std::cout << m[y][x] << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}

// Execute!

int main() {

printMatrix();

processMatrix();

printMatrix();

}