NAME= WANIA HASEEB

NUMBER GUESSING GAME

CODE

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

int lower, upper, x, guess, count = 0, flag = 0;

int total\_chances;

// Taking Inputs

printf("Enter Lower bound: ");

scanf("%d", &lower);

// Taking Inputs

printf("Enter Upper bound: ");

scanf("%d", &upper);

// Seed the random number generator

srand(time(0));

// Generating random number between the lower and upper

x = (rand() % (upper - lower + 1)) + lower;

total\_chances

= (int)ceil(log(upper - lower + 1) / log(2));

printf("\n\tYou've only %d chances to guess the "

"integer!\n\n",

total\_chances);

// for calculation of minimum number of guesses depends

// upon range

while (count < total\_chances) {

count++;

// Taking guessing number as input

printf("Guess a number: ");

scanf("%d", &guess);

// Condition testing

if (x == guess) {

printf(

"Congratulations you did it in %d try!\n",

count);

// Once guessed, loop will break

flag = 1;

break;

}

else if (x > guess) {

printf("You guessed too small!\n");

}

else if (x < guess) {

printf("You guessed too high!\n");

}

}

// If Guessing is more than required guesses, shows this

// output.

if (!flag) {

printf("\nThe number is %d\n", x);

printf("\tBetter Luck Next time!\n");

}

return 0;

}

TASK 2

SODUKO SOLVER

// C++ Program to solve Sudoku problem

#include <iostream>

#include <vector>

using namespace std;

// Function to heck if it is safe to place num at mat[row][col]

bool isSafe(vector<vector<int>> &mat, int i, int j, int num,

vector<int> &row, vector<int> &col, vector<int> &box) {

if( (row[i] & (1 << num)) || (col[j] & (1 << num)) ||

(box[i / 3 \* 3 + j / 3] & (1 << num)) )

return false;

return true;

}

bool sudokuSolverRec(vector<vector<int>> &mat, int i, int j,

vector<int> &row, vector<int> &col, vector<int> &box) {

int n = mat.size();

// base case: Reached nth column of last row

if (i == n - 1 && j == n)

return true;

// If reached last column of the row go to next row

if (j == n) {

i++;

j = 0;

}

// If cell is already occupied then move forward

if (mat[i][j] != 0)

return sudokuSolverRec(mat, i, j + 1, row, col, box);

for (int num = 1; num <= n; num++) {

// If it is safe to place num at current position

if (isSafe(mat, i, j, num, row, col, box)) {

mat[i][j] = num;

// Update masks for the corresponding row, column and box

row[i] |= (1 << num);

col[j] |= (1 << num);

box[i / 3 \* 3 + j / 3] |= (1 << num);

if (sudokuSolverRec(mat, i, j + 1, row, col, box))

return true;

// Unmask the number num in the corresponding row, column and box masks

mat[i][j] = 0;

row[i] &= ~(1 << num);

col[j] &= ~(1 << num);

box[i / 3 \* 3 + j / 3] &= ~(1 << num);

}

}

TASK 3

AI CHATBOT

// C++ Program to solve Sudoku problem

#include <iostream>

#include <vector>

using namespace std;

// Function to heck if it is safe to place num at mat[row][col]

bool isSafe(vector<vector<int>> &mat, int i, int j, int num,

vector<int> &row, vector<int> &col, vector<int> &box) {

if( (row[i] & (1 << num)) || (col[j] & (1 << num)) ||

(box[i / 3 \* 3 + j / 3] & (1 << num)) )

return false;

return true;

}

bool sudokuSolverRec(vector<vector<int>> &mat, int i, int j,

vector<int> &row, vector<int> &col, vector<int> &box) {

int n = mat.size();

// base case: Reached nth column of last row

if (i == n - 1 && j == n)

return true;

// If reached last column of the row go to next row

if (j == n) {

i++;

j = 0;

}

// If cell is already occupied then move forward

if (mat[i][j] != 0)

return sudokuSolverRec(mat, i, j + 1, row, col, box);

for (int num = 1; num <= n; num++) {

// If it is safe to place num at current position

if (isSafe(mat, i, j, num, row, col, box)) {

mat[i][j] = num;

// Update masks for the corresponding row, column and box

row[i] |= (1 << num);

col[j] |= (1 << num);

box[i / 3 \* 3 + j / 3] |= (1 << num);

if (sudokuSolverRec(mat, i, j + 1, row, col, box))

return true;

// Unmask the number num in the corresponding row, column and box masks

mat[i][j] = 0;

row[i] &= ~(1 << num);

col[j] &= ~(1 << num);

box[i / 3 \* 3 + j / 3] &= ~(1 << num);

}

}