#include <stdio.h>

// 交换两个整数

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// 选择排序

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex!= i) {

swap(&arr[i], &arr[minIndex]);

}

}

}

int main() {

int arr[] = { 5, 4, 3, 2, 1 }; // 这里可以替换成你想要排序的数组

int n = sizeof(arr) / sizeof(arr[0]);

selectionSort(arr, n);

// 输出排序后的数组

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

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

}

return 0;

}

#include <stdio.h>

// 交换两个整数

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// 冒泡排序

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

swap(&arr[j], &arr[j + 1]);

}

}

}

}

int main() {

int arr[] = { 5, 4, 3, 2, 1 }; // 可替换为你要排序的数组

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

// 输出排序后的数组

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

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

}

return 0;

}

~

strcpy(temp, A[row1]);

strcpy(A[row1], A[row2]);

strcpy(A[row2], temp);

}

// 对二维字符数组进行排序

void sortCharArray(char A[][6], int rows) {

for (int i = 0; i < rows - 1; i++) {

for (int j = i + 1; j < rows; j++) {

if (rowAsciiSum(A[i], 6) > rowAsciiSum(A[j], 6)) {

swapRows(A, i, j);

} else if (rowAsciiSum(A[i], 6) == rowAsciiSum(A[j], 6)) {

for (int k = 0; k < 6; k++) {

if (A[i][k] > A[j][k]) {

swapRows(A, i, j);

break;

} else if (A[i][k] < A[j][k]) {

break;

}

}

}

}

}

}

int main() {

char A[][6] = {

"abcde",

"fghij",

"klmno"

};

int rows = sizeof(A) / sizeof(A[0]);

sortCharArray(A, rows);

// 输出排序后的二维字符数组

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

printf("%s\n", A[i]);

}

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#define N 8

// 检查当前位置是否可以放置皇后

int isSafe(int board[N][N], int row, int col) {

int i, j;

// 检查同一列

for (i = 0; i < row; i++) {

if (board[i][col] == 1) {

return 0;

}

}

// 检查左上方对角线

for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (board[i][j] == 1) {

return 0;

}

}

// 检查右上方对角线

for (i = row, j = col; i >= 0 && j < N; i--, j++) {

if (board[i][j] == 1) {

return 0;

}

}

return 1;

}

// 解决八皇后问题的递归函数

void solveNQueens(int board[N][N], int row) {

if (row == N) {

// 找到一个解决方案，打印棋盘

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

for (int j = 0; j < N; j++) {

if (board[i][j] == 1) {

printf("Q ");

} else {

printf(". ");

}

}

printf("\n");

}

printf("\n");

return;

}

for (int col = 0; col < N; col++) {

if (isSafe(board, row, col)) {

board[row][col] = 1;

solveNQueens(board, row + 1);

board[row][col] = 0;

}

}

}

// 主函数调用解决八皇后问题的函数

int main() {

int board[N][N] = {0};

solveNQueens(board, 0);

return 0;

}

#include <stdio.h>

int main() {

int A[2][3] = {{1, 2, 3}, {4, 5, 6}};

// 尝试通过修改元素值并查看其他位置元素是否变化来判断存放方式

// 先修改按行存放可能相邻的元素

A[0][0] = 100;

if (A[0][1]!= 2) {

printf("可能是按列存放\n");

} else {

// 再修改按列存放可能相邻的元素

A[0][0] = 1;

A[1][0] = 100;

if (A[0][0]!= 1) {

printf("可能是按行存放\n");

} else {

printf("无法准确判断，可能需要更多复杂测试\n");

}

}

return 0;

}

#include <stdio.h>

// 计算二维数组元素地址并返回元素值

int two\_dim\_array\_access(int \*x, int i, int j, int m, int n, int C) {

return \*(x + m \* C + n);

}

int main() {

int arr[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

int \*ptr = (int \*)arr; // 将二维数组视为一维数组来获取首地址

int i = 1; // 示例行索引

int j = 2; // 示例列索引

int m = sizeof(arr) / sizeof(arr[0]);

int n = sizeof(arr[0]) / sizeof(int);

int C = sizeof(arr[0]);

int value = two\_dim\_array\_access(ptr, i, j, m, n, C);

printf("二维数组中第 %d 行第 %d 列的元素值为: %d\n", i, j, value);

return 0;

}

#include <stdio.h>

// 矩阵相乘函数

void matrix\_multi(double A[3][4], double B[4][5], double C[3][5]) {

// 遍历矩阵A的行

for (int i = 0; i < 3; i++) {

// 遍历矩阵B的列

for (int j = 0; j < 5; j++) {

C[i][j] = 0;

// 计算矩阵A的第i行与矩阵B的第j列的点积

for (int k = 0; k < 4; k++) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

}

int main() {

double A[3][4] = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

double B[4][5] = {

{1, 2, 3, 4, 5},

{6, 7, 8, 9, 10},

{11, 12, 13, 14, 15},

{16, 17, 18, 19, 20}

};

double C[3][5] = {0};

matrix\_multi(A, B, C);

// 输出结果矩阵C

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 5; j++) {

printf("%.2lf ", C[i][j]);

}

printf("\n");

}

return 0;

}