Задание 1

Чтобы перевести длину из дюймов в метрическую систему, нужно умножить значение в дюймах на коэффициент перевода 2,54, чтобы получить значение в сантиметрах. Затем, для получения метров и миллиметров, можно использовать следующие соотношения: 1 метр = 100 сантиметров и 1 сантиметр = 10 миллиметров.

Давайте рассчитаем значение для 21 дюйма:

21 дюйм \* 2,54 см/дюйм = 53,34 см.

Теперь переведем сантиметры в метры и миллиметры:

53,34 см = 0,5334 метра (поскольку 1 метр = 100 сантиметров)

0,5334 метра = 53,34 см = 533,4 миллиметра.

Таким образом, 21 дюйм равны 0 метров, 53 сантиметров и 533,4 миллиметра.

#include <stdio.h>  
#include <windows.h>  
  
void numberTranslation(float inches) {  
 float centimeters, meters, millimeters;  
  
 centimeters = inches \* 2.54;  
 meters = centimeters / 100;  
 millimeters = centimeters \* 10;  
  
 int metersInt = (int) meters;  
 int centimetersInt = (int) centimeters % 100;  
 float millimetersFloat = millimeters - (int) centimeters \* 10;  
  
 printf("Длина в метрической системе: %d м %d см %.2f мм\n",  
 metersInt, centimetersInt, millimetersFloat);  
}  
  
void test1() {  
 float inches = 1;  
 printf("Test1\n");  
 numberTranslation(inches);  
}  
  
void test2() {  
 float inches = 10;  
 printf("Test2\n");  
 numberTranslation(inches);  
}  
  
void test3() {  
 float inches = 100.3;  
 printf("Test3\n");  
 numberTranslation(inches);  
}  
  
void test() {  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
 /\*ручной ввод  
 float inches;  
 printf("Ручной ввод\n");  
 printf("Введите длину в дюймах: ");  
 scanf("%f", &inches);  
 numberTranslation(inches);  
 \*/  
 return 0;  
}

Задание 2

Треугольник задан координатами своих вершин на плоскости: . Определить, является он прямо-, остро- или тупоугольным.

Для определения типа треугольника по его вершинам, можно использовать геометрическую формулу, известную как "теорема косинусов". Согласно этой формуле, квадрат длины одной из сторон треугольника равен сумме квадратов длин двух других сторон, умноженной на два произведения этих сторон и косинуса угла между ними.

Пусть стороны треугольника определены следующим образом:

a = длина стороны BC (сторона против вершины A)

b = длина стороны AC (сторона против вершины B)

c = длина стороны AB (сторона против вершины C)

Также введем следующие обозначения:

A = длина стороны BC (сторона против угла A)

B = длина стороны AC (сторона против угла B)

C = длина стороны AB (сторона против угла C)

Тогда формула теоремы косинусов будет выглядеть следующим образом:

a^2 = b^2 + c^2 - 2bc \* cos(A)

b^2 = a^2 + c^2 - 2ac \* cos(B)

c^2 = a^2 + b^2 - 2ab \* cos(C)

Теперь можно определить тип треугольника на основе значений cos(A), cos(B) и cos(C):

- Если все значения cos(A), cos(B) и cos(C) положительны, то треугольник является остроугольным.

- Если хотя бы одно из значений cos(A), cos(B) или cos(C) отрицательно, то треугольник является тупоугольным.

- Если хотя бы одно из значений cos(A), cos(B) или cos(C) равно нулю, то треугольник является прямоугольным.

Таким образом, для определения типа треугольника необходимо вычислить значения cos(A), cos(B) и cos(C) и проверить их знаки.

P.S. Для вычисления косинусов углов можно использовать формулу косинусов:

cos(A) = (b^2 + c^2 - a^2) / (2bc)

cos(B) = (a^2 + c^2 - b^2) / (2ac)

cos(C) = (a^2 + b^2 - c^2) / (2ab)

#include <stdio.h>  
#include <math.h>  
#include <windows.h>  
  
// Функция для вычисления квадрата числа  
double square(double x) {  
 return x \* x;  
}  
  
// Функция для вычисления косинуса угла  
double cosine(double a, double b, double c) {  
 return (square(b) + square(c) - square(a)) / (2 \* b \* c);  
}  
  
// Функция для определения типа треугольника  
void determineTriangleType(double x1, double y1, double x2, double y2, double x3, double y3) {  
 // Вычисляем длины сторон треугольника  
 double a = sqrt(square(x2 - x3) + square(y2 - y3)); // Длина стороны BC  
 double b = sqrt(square(x1 - x3) + square(y1 - y3)); // Длина стороны AC  
 double c = sqrt(square(x1 - x2) + square(y1 - y2)); // Длина стороны AB  
  
 // Вычисляем значения косинусов углов  
 double cosA = cosine(a, b, c);  
 double cosB = cosine(b, a, c);  
 double cosC = cosine(c, a, b);  
  
 // Проверяем знаки косинусов для определения типа треугольника  
 if (cosA > 0 && cosB > 0 && cosC > 0) {  
 printf("Треугольник остроугольный.\n");  
 } else if (cosA < 0 || cosB < 0 || cosC < 0) {  
 printf("Треугольник тупоугольный.\n");  
 } else {  
 printf("Треугольник прямоугольный.\n");  
 }  
}  
  
void test1(){  
 double x1 = 1;  
 double y1 = 1;  
 double x2 = 1;  
 double y2 = 1;  
 double x3 = 1;  
 double y3 = 1;  
 printf("Test1\n");  
 determineTriangleType(x1, y1, x2, y2, x3, y3);  
}  
  
void test2(){  
 double x1 = 3;  
 double y1 = 2;  
 double x2 = 1;  
 double y2 = 3;  
 double x3 = 3;  
 double y3 = 7;  
 printf("Test2\n");  
 determineTriangleType(x1, y1, x2, y2, x3, y3);  
}  
  
void test3(){  
 double x1 = 6;  
 double y1 = 7;  
 double x2 = 3;  
 double y2 = 1;  
 double x3 = 2;  
 double y3 = 6;  
 printf("Test3\n");  
 determineTriangleType(x1, y1, x2, y2, x3, y3);  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
 /\*ручной ввод  
 double x1, y1, x2, y2, x3, y3;  
 printf("Ручной ввод\n");  
 printf("Введите координаты вершин треугольника A(x1, y1), B(x2, y2), C(x3, y3):\n");  
 printf("x1 = ");  
 scanf("%lf", &x1);  
 printf("y1 = ");  
 scanf("%lf", &y1);  
 printf("x2 = ");  
 scanf("%lf", &x2);  
 printf("y2 = ");  
 scanf("%lf", &y2);  
 printf("x3 = ");  
 scanf("%lf", &x3);  
 printf("y3 = ");  
 scanf("%lf", &y3);  
  
 determineTriangleType(x1, y1, x2, y2, x3, y3);  
\*/  
 return 0;  
}

Задание 3

Для заданных a и b найти все точки с целочисленными координатами, находящиеся внутри эллипса

#include <stdio.h>  
#include <windows.h>  
  
void findPointsInEllipse(int a, int b) {  
 int x\_min = -a;  
 int x\_max = a;  
 int y\_min = -b;  
 int y\_max = b;  
  
 for (int x = x\_min; x <= x\_max; x++) {  
 for (int y = y\_min; y <= y\_max; y++) {  
 if ((x \* x) / (a \* a) + (y \* y) / (b \* b) <= 1) {  
 printf("(%d, %d)\n", x, y);  
 }  
 }  
 }  
}  
  
void test1(){  
 int a = 3;  
 int b = 2;  
 printf("Test1\n");  
 findPointsInEllipse(a, b);  
}  
  
void test2(){  
 int a = 5;  
 int b = 2;  
 printf("Test2\n");  
 findPointsInEllipse(a, b);  
}  
  
void test3(){  
 int a = 3;  
 int b = 7;  
 printf("Test3\n");  
 findPointsInEllipse(a, b);  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
 /\* ручной ввод  
 printf("Ручной ввод\n");  
  
 int a;  
 printf("Введите значение a: ");  
 scanf("%d", &a);  
  
 int b;  
 printf("Введите значение b: ");  
 scanf("%d", &b);  
  
 findPointsInEllipse(a, b);  
\*/  
 return 0;  
}

Задание 4

От каждого из заданных чисел , , ..., отнять их среднее арифметическое. Результаты разместить на месте исходных данных.

Для решения данной задачи вам потребуется выполнить следующие шаги:

1. Вычислите среднее арифметическое заданных чисел x\_1, x\_2, ..., x\_m. Для этого найдите сумму всех чисел и разделите её на количество чисел m.

Среднее арифметическое = (x\_1 + x\_2 + ... + x\_m) / m

2. Отнимите полученное среднее арифметическое от каждого из исходных чисел x\_1, x\_2, ..., x\_m.

x\_1 = x\_1 - среднее арифметическое

x\_2 = x\_2 - среднее арифметическое

...

x\_m = x\_m - среднее арифметическое

3. Результаты вычитания разместите на месте исходных данных.

Например, если у вас есть числа x = [1, 2, 3, 4, 5], то:

Среднее арифметическое = (1 + 2 + 3 + 4 + 5) / 5 = 15 / 5 = 3

Вычитаем среднее арифметическое из каждого числа:

x\_1 = 1 - 3 = -2

x\_2 = 2 - 3 = -1

x\_3 = 3 - 3 = 0

x\_4 = 4 - 3 = 1

x\_5 = 5 - 3 = 2

Таким образом, результатом будет новый список чисел: x = [-2, -1, 0, 1, 2].

#include <stdio.h>  
#include <windows.h>  
  
void subtractMean(int arr[], int size) {  
 int sum = 0;  
 int i;  
  
 // Вычисляем сумму всех элементов  
 for (i = 0; i < size; i++) {  
 sum += arr[i];  
 }  
  
 // Вычисляем среднее арифметическое  
 int mean = sum / size;  
  
 // Вычитаем среднее арифметическое из каждого элемента  
 for (i = 0; i < size; i++) {  
 arr[i] -= mean;  
 }  
}  
  
void test1(){  
 int arr[] = {1, 2, 3, 4, 5};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 int i;  
  
 printf("Test1\n");  
 printf("Исходный массив: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 subtractMean(arr, size);  
  
 printf("Результат: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
}  
  
void test2(){  
 int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 int i;  
  
 printf("Test2\n");  
 printf("Исходный массив: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 subtractMean(arr, size);  
  
 printf("Результат: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
}  
  
void test3(){  
 int arr[] = {1, 5, 6, 9, 25};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 int i;  
  
 printf("Test3\n");  
 printf("Исходный массив: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 subtractMean(arr, size);  
  
 printf("Результат: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
/\*  
 printf("Ручной ввод\n");  
 int arr[] = {1, 2, 3, 4, 5};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 int i;  
  
 printf("Исходный массив: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 subtractMean(arr, size);  
  
 printf("Результат: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
\*/  
 return 0;  
}

Задание 5

Для двух заданных матриц A(n, n) и B(n, n) проверить, можно ли получить вторую из первой применением конечного числа (не более четырёх) операций транспонирования относительно главной и побочной диагоналей.

Для проверки, можно ли получить матрицу B из матрицы A путем применения конечного числа операций транспонирования относительно главной и побочной диагоналей, следует выполнить следующие шаги:

1. Проверить, совпадают ли размеры матриц A и B. Если размеры матриц не совпадают (то есть n не равно n), то невозможно получить матрицу B из матрицы A путем операций транспонирования.

2. Проверить, совпадают ли матрицы A и B. Если матрицы A и B уже совпадают, то не требуется никаких операций транспонирования для преобразования A в B.

3. Выполнить все возможные комбинации операций транспонирования относительно главной и побочной диагоналей над матрицей A и проверить, совпадает ли полученная матрица с матрицей B.

4. Если в результате применения операций транспонирования удалось получить матрицу B, то можно сказать, что можно получить матрицу B из матрицы A с помощью конечного числа операций транспонирования. В противном случае, невозможно получить матрицу B из матрицы A путем операций транспонирования.

Обратите внимание, что проверка всех возможных комбинаций операций транспонирования может быть вычислительно затратной для больших значений n. Поэтому ограничение "не более четырех" операций транспонирования может помочь сократить количество проверок в некоторых случаях.

#include <stdio.h>  
#include <windows.h>  
  
#define N 3 // Размерность матрицы  
  
// Функция для сравнения двух матриц  
int compareMatrices(int A[N][N], int B[N][N]) {  
 int i, j;  
 for (i = 0; i < N; i++) {  
 for (j = 0; j < N; j++) {  
 if (A[i][j] != B[i][j]) {  
 return 0; // Матрицы не идентичны  
 }  
 }  
 }  
 return 1; // Матрицы идентичны  
}  
  
// Функция для выполнения операции транспонирования матрицы относительно главной диагонали  
void transposeMainDiagonal(int matrix[N][N]) {  
 int i, j, temp;  
 for (i = 0; i < N; i++) {  
 for (j = i + 1; j < N; j++) {  
 temp = matrix[i][j];  
 matrix[i][j] = matrix[j][i];  
 matrix[j][i] = temp;  
 }  
 }  
}  
  
// Функция для выполнения операции транспонирования матрицы относительно побочной диагонали  
void transposeSecondaryDiagonal(int matrix[N][N]) {  
 int i, j, temp;  
 for (i = 0; i < N; i++) {  
 for (j = 0; j < N - i - 1; j++) {  
 temp = matrix[i][j];  
 matrix[i][j] = matrix[N - j - 1][N - i - 1];  
 matrix[N - j - 1][N - i - 1] = temp;  
 }  
 }  
}  
  
// Функция для проверки возможности получения матрицы B из матрицы A  
int canObtainMatrixB(int A[N][N], int B[N][N]) {  
 // Шаг 1: Проверка идентичности матриц  
 if (compareMatrices(A, B)) {  
 return 1; // Матрицы идентичны  
 }  
  
 // Шаг 2: Проверка возможности получения с помощью одной операции транспонирования относительно главной диагонали  
 transposeMainDiagonal(A);  
 if (compareMatrices(A, B)) {  
 return 1; // Матрицы идентичны после одной операции транспонирования относительно главной диагонали  
 }  
  
 // Шаг 3: Проверка возможности получения с помощью одной операции транспонирования относительно побочной диагонали  
 transposeMainDiagonal(A);  
 transposeSecondaryDiagonal(A);  
 if (compareMatrices(A, B)) {  
 return 1; // Матрицы идентичны после одной операции транспонирования относительно побочной диагонали  
 }  
  
 // Шаг 4: Проверка возможности получения с помощью двух операций транспонирования относительно главной и/или побочной диагонали  
 transposeSecondaryDiagonal(A);  
 transposeMainDiagonal(A);  
 if (compareMatrices(A, B)) {  
 return 1; // Матрицы идентичны после двух операций транспонирования  
 }  
  
 return 0; // Невозможно получить матрицу B из матрицы A с помощью четырех или меньшего числа операций транспонирования  
}  
  
void test1(){  
 int A[N][N] = {{1, 2, 3},  
 {4, 5, 6},  
 {7, 8, 9}};  
 int B[N][N] = {{1, 4, 7},  
 {2, 5, 8},  
 {3, 6, 9}};  
  
 printf("Test1\n");  
   
 if (canObtainMatrixB(A, B)) {  
 printf("Можно получить матрицу B из матрицы A.\n");  
 } else {  
 printf("Невозможно получить матрицу B из матрицы A.\n");  
 }  
}  
  
void test2(){  
 int A[N][N] = {{1, 4, 9},  
 {7, 2, 5},  
 {6, 8, 3}};  
 int B[N][N] = {{1, 7, 6},  
 {4, 2, 8},  
 {9, 5, 3}};  
  
 printf("Test2\n");  
   
 if (canObtainMatrixB(A, B)) {  
 printf("Можно получить матрицу B из матрицы A.\n");  
 } else {  
 printf("Невозможно получить матрицу B из матрицы A.\n");  
 }  
}  
  
void test3(){  
 int A[N][N] = {{9, 4, 1},  
 {5, 2, 7},  
 {3, 8, 6}};  
 int B[N][N] = {{9, 5, 3},  
 {4, 2, 8},  
 {1, 7, 6}};  
  
 printf("Test3\n");  
   
 if (canObtainMatrixB(A, B)) {  
 printf("Можно получить матрицу B из матрицы A.\n");  
 } else {  
 printf("Невозможно получить матрицу B из матрицы A.\n");  
 }  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
/\* ручной ввод  
 int A[N][N] = {{1, 2, 3},  
 {4, 5, 6},  
 {7, 8, 9}};  
 int B[N][N] = {{1, 4, 7},  
 {2, 5, 8},  
 {3, 6, 9}};  
   
 printf("Ручной ввод\n");  
   
 if (canObtainMatrixB(A, B)) {  
 printf("Можно получить матрицу B из матрицы A.\n");  
 } else {  
 printf("Невозможно получить матрицу B из матрицы A.\n");  
 }  
\*/  
 return 0;  
}

Задание 6

Задано n линейных функций: , , ..., . Найти минимум «верхней огибающей» этих функций, то есть кусочно-линейной функции .

#include <stdio.h>  
#include <float.h>  
#include <windows.h>  
  
struct LinearFunction {  
 double slope;  
 double intercept;  
};  
  
struct LinearFunction findUpperEnvelope(struct LinearFunction \*functions, int n) {  
 struct LinearFunction upperEnvelope;  
 upperEnvelope.slope = -DBL\_MAX;  
 upperEnvelope.intercept = -DBL\_MAX;  
  
 for (int i = 0; i < n; i++) {  
 struct LinearFunction currentFunction = functions[i];  
 if (currentFunction.slope > upperEnvelope.slope) {  
 upperEnvelope.slope = currentFunction.slope;  
 upperEnvelope.intercept = currentFunction.intercept;  
 }  
 }  
 return upperEnvelope;  
}  
  
double evaluateUpperEnvelope(struct LinearFunction \*functions, int n, double x) {  
 struct LinearFunction upperEnvelope = findUpperEnvelope(functions, n);  
 return upperEnvelope.slope \* x + upperEnvelope.intercept;  
}  
  
void test1(){  
 struct LinearFunction functions[] = {{2, 1},  
 {-1, 4},  
 {0.5, -2}};  
 int numFunctions = sizeof(functions) / sizeof(functions[0]);  
  
 struct LinearFunction upperEnvelope = findUpperEnvelope(functions, numFunctions);  
  
 printf("Test1\n");  
 printf("Верхняя огибающая: y = %.2fx + %.2f\n",  
 upperEnvelope.slope, upperEnvelope.intercept);  
  
 double x = 2.5;  
 double y = evaluateUpperEnvelope(functions, numFunctions, x);  
 printf("y(%.2f) = %.2f\n", x, y);  
}  
  
void test2(){  
 struct LinearFunction functions[] = {{5, 1},  
 {-9, 1},  
 {0.5, -4}};  
 int numFunctions = sizeof(functions) / sizeof(functions[0]);  
  
 struct LinearFunction upperEnvelope = findUpperEnvelope(functions, numFunctions);  
  
 printf("Test2\n");  
 printf("Верхняя огибающая: y = %.2fx + %.2f\n",  
 upperEnvelope.slope, upperEnvelope.intercept);  
  
 double x = 0.5;  
 double y = evaluateUpperEnvelope(functions, numFunctions, x);  
 printf("y(%.2f) = %.2f\n", x, y);  
}  
  
void test3(){  
 struct LinearFunction functions[] = {{2, 7},  
 {-1, 4},  
 {7.5, 10}};  
 int numFunctions = sizeof(functions) / sizeof(functions[0]);  
  
 struct LinearFunction upperEnvelope = findUpperEnvelope(functions, numFunctions);  
  
 printf("Test3\n");  
 printf("Верхняя огибающая: y = %.2fx + %.2f\n",  
 upperEnvelope.slope, upperEnvelope.intercept);  
  
 double x = 2.5;  
 double y = evaluateUpperEnvelope(functions, numFunctions, x);  
 printf("y(%.2f) = %.2f\n", x, y);  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
/\* ручной ввод  
 // Пример входных линейных функций (a\_i, b\_i)  
 struct LinearFunction functions[] = {{2, 1},  
 {-1, 4},  
 {0.5, -2}};  
 int numFunctions = sizeof(functions) / sizeof(functions[0]);  
  
 struct LinearFunction upperEnvelope = findUpperEnvelope(functions, numFunctions);  
  
 printf("Ручной ввод\n");  
 printf("Верхняя огибающая: y = %.2fx + %.2f\n",  
 upperEnvelope.slope, upperEnvelope.intercept);  
  
 double x = 2.5;  
 double y = evaluateUpperEnvelope(functions, numFunctions, x);  
 printf("y(%.2f) = %.2f\n", x, y);  
\*/  
 return 0;  
}

Задание 7

Число делится на 11, если разность между суммой цифр, стоящих на нечётных местах, и суммой цифр, стоящих на чётных местах, кратна 11. Проверить этот признак для всех натуральных чисел, не превосходящих заданного , и вывести числа, кратные 11.

#include <stdio.h>  
#include <windows.h>  
  
int is\_divisible\_by\_11(int number) {  
 int digits[10];  
 int odd\_sum = 0, even\_sum = 0, i = 0;  
  
 while (number > 0) {  
 digits[i] = number % 10;  
 number /= 10;  
 i++;  
 }  
  
 for (int j = 0; j < i; j++) {  
 if (j % 2 == 0) {  
 odd\_sum += digits[j];  
 } else {  
 even\_sum += digits[j];  
 }  
 }  
  
 if ((odd\_sum - even\_sum) % 11 == 0) {  
 return 1;  
 } else {  
 return 0;  
 }  
}  
  
void check\_divisibility\_by\_11(int limit) {  
 for (int number = 1; number <= limit; number++) {  
 if (is\_divisible\_by\_11(number)) {  
 printf("%d\n", number);  
 }  
 }  
}  
  
void test1(){  
 printf("Test1\n");  
 int limit = 10;  
 check\_divisibility\_by\_11(limit);  
}  
  
void test2(){  
 printf("Test2\n");  
 int limit = 23;  
 check\_divisibility\_by\_11(limit);  
}  
  
void test3(){  
 printf("Test3\n");  
 int limit = 100;  
 check\_divisibility\_by\_11(limit);  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
  
 printf("Ручной ввод\n");  
 printf("Введите лимит проверки кратности 11: ");  
  
 int limit;  
 scanf("%d", &limit);  
 check\_divisibility\_by\_11(limit);  
  
 return 0;  
}

Задание 8

Задано множество точек в трёхмерном пространстве. Найти его выпуклую оболочку, то есть множество плоскостей — граней многогранника наименьшего объёма, содержащего все заданные точки.

Замечание. Каждая грань проходит как минимум через некоторые 3 точки заданного множества.

Алгоритм Грэхема для нахождения выпуклой оболочки в трехмерном пространстве можно описать следующим образом:

Найти точку p с наименьшей координатой z. Если таких точек несколько, выбрать с наименьшей координатой y. Если таких точек тоже несколько, выбрать с наименьшей координатой x.

Отсортировать все остальные точки по углу, образованному с осью x и начальной точкой p. Можно использовать арктангенс для расчета углов.

Создать пустой стек и поместить в него точку p.

Для каждой точки q, начиная со второй, в отсортированном списке:

Пока в стеке есть больше двух точек и последние три точки в стеке (вершины выпуклой оболочки) образуют правый поворот, удалять последнюю точку из стека.

Добавить точку q в стек.

Вернуть точки из стека как результат.

Этот алгоритм работает со сложностью O(n log n), где n - количество точек в заданном множестве.

Применение этого алгоритма позволит найти грани многогранника наименьшего объёма, содержащего все заданные точки в трехмерном пространстве.

#include <stdio.h>  
#include <stdlib.h>  
#include <windows.h>  
#include <math.h>  
  
// Структура для представления точки в трехмерном пространстве  
typedef struct {  
 double x, y, z;  
} Point3D;  
  
// Структура для представления стека точек  
typedef struct {  
 Point3D\* points;  
 int top;  
} Stack;  
  
// Функция для создания пустого стека  
Stack\* createStack(int capacity) {  
 Stack\* stack = (Stack\*)malloc(sizeof(Stack));  
 stack->points = (Point3D\*)malloc(capacity \* sizeof(Point3D));  
 stack->top = -1;  
 return stack;  
}  
  
// Функция для добавления точки в стек  
void push(Stack\* stack, Point3D point) {  
 stack->points[++stack->top] = point;  
}  
  
// Функция для удаления верхней точки из стека  
void pop(Stack\* stack) {  
 stack->top--;  
}  
  
// Функция для получения верхней точки стека  
Point3D top(Stack\* stack) {  
 return stack->points[stack->top];  
}  
  
// Функция для проверки, образует ли тройка точек правый поворот  
int isRightTurn(Point3D p, Point3D q, Point3D r) {  
 double crossProduct =  
 (q.x - p.x) \* (r.y - p.y) \* (r.z - p.z) +  
 (q.y - p.y) \* (r.z - p.z) \* (r.x - p.x) +  
 (q.z - p.z) \* (r.x - p.x) \* (r.y - p.y) -  
 (q.z - p.z) \* (r.y - p.y) \* (r.x - p.x) -  
 (q.y - p.y) \* (r.x - p.x) \* (r.z - p.z) -  
 (q.x - p.x) \* (r.z - p.z) \* (r.y - p.y);  
  
 return crossProduct > 0; // Если crossProduct > 0, это правый поворот  
}  
  
// Функция для сортировки точек по углу  
int comparePoints(const void\* a, const void\* b) {  
 Point3D p = \*(Point3D\*)a;  
 Point3D q = \*(Point3D\*)b;  
  
 double angle1 = atan2(p.y, p.x);  
 double angle2 = atan2(q.y, q.x);  
  
 if (angle1 < angle2)  
 return -1;  
 if (angle1 > angle2)  
 return 1;  
 return 0;  
}  
  
// Функция для нахождения выпуклой оболочки в трехмерном пространстве  
void findConvexHull(Point3D points[], int n) {  
 // Находим точку p с наименьшей координатой z. Если таких точек несколько,  
 // выбираем с наименьшей координатой y. Если таких точек тоже несколько,  
 // выбираем с наименьшей координатой x.  
 int min = 0;  
 for (int i = 1; i < n; i++) {  
 if (points[i].z < points[min].z ||  
 (points[i].z == points[min].z && points[i].y < points[min].y) ||  
 (points[i].z == points[min].z && points[i].y == points[min].y && points[i].x < points[min].x)) {  
 min = i;  
 }  
 }  
  
 // Меняем местами точку с наименьшей координатой z и первую точку  
 Point3D temp = points[0];  
 points[0] = points[min];  
 points[min] = temp;  
  
 // Сортируем остальные точки по углу  
 qsort(&points[1], n - 1, sizeof(Point3D), comparePoints);  
  
 // Создаем стек и добавляем первые две точки  
 Stack\* stack = createStack(n);  
 push(stack, points[0]);  
 push(stack, points[1]);  
  
 // Добавляем остальные точки  
 for (int i = 2; i < n; i++) {  
 while (stack->top > 0 && isRightTurn(stack->points[stack->top - 1], top(stack), points[i])) {  
 pop(stack);  
 }  
 push(stack, points[i]);  
 }  
  
 // Выводим точки выпуклой оболочки  
 printf("Выпуклая оболочка:\n");  
 for (int i = 0; i <= stack->top; i++) {  
 printf("(%.2lf, %.2lf, %.2lf)\n", stack->points[i].x, stack->points[i].y, stack->points[i].z);  
 }  
  
 // Освобождаем память  
 free(stack->points);  
 free(stack);  
}  
  
void test1(){  
 int n = 3;  
  
 Point3D\* points = (Point3D\*)malloc(n \* sizeof(Point3D));  
  
 printf("Test1\n");  
 printf("Точек: %d\n", n);  
  
 points[0].x = 2, points[0].y = 5, points[0].z = 7,  
 points[1].x = 4, points[1].y = 5, points[1].z = 7,  
 points[2].x = 2, points[2].y = 4, points[2].z = 7;  
  
 findConvexHull(points, n);  
  
 free(points);  
}  
  
void test2(){  
 int n = 3;  
  
 Point3D\* points = (Point3D\*)malloc(n \* sizeof(Point3D));  
  
 printf("Test2\n");  
 printf("Точек: %d\n", n);  
  
 points[0].x = 7, points[0].y = 5, points[0].z = 7,  
 points[1].x = 2, points[1].y = 9, points[1].z = 6,  
 points[2].x = 2, points[2].y = 4, points[2].z = 7;  
  
 findConvexHull(points, n);  
  
 free(points);  
}  
  
void test3(){  
 int n = 3;  
  
 Point3D\* points = (Point3D\*)malloc(n \* sizeof(Point3D));  
  
 printf("Test3\n");  
 printf("Точек: %d\n", n);  
  
 points[0].x = 1, points[0].y = 2, points[0].z = 20,  
 points[1].x = 6, points[1].y = 7, points[1].z = 8,  
 points[2].x = 11, points[2].y = 2, points[2].z = 3;  
  
 findConvexHull(points, n);  
  
 free(points);  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
/\* ручной ввод  
 int n;  
 printf("Введите количество точек: ");  
 scanf("%d", &n);  
  
 Point3D\* points = (Point3D\*)malloc(n \* sizeof(Point3D));  
  
 printf("Введите координаты точек:\n");  
 for (int i = 0; i < n; i++) {  
 printf("Точка %d: ", i + 1);  
 scanf("%lf %lf %lf", &points[i].x, &points[i].y, &points[i].z);  
 }  
  
 findConvexHull(points, n);  
  
 free(points);  
\*/  
 return 0;  
}

Задание 9

Найти произведение матриц A(m, n) и B(n, k ): C(m, k) = A×B . Матрицы A, B и C описать как одномерные массивы, используя приведённые индексы.

#include <stdio.h>  
#include <windows.h>  
  
void matrixMultiplication(int \*A, int \*B, int \*C, int m, int n, int k) {  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < k; j++) {  
 int sum = 0;  
 for (int p = 0; p < n; p++) {  
 sum += A[i \* n + p] \* B[p \* k + j];  
 }  
 C[i \* k + j] = sum;  
 }  
 }  
}  
  
void test1(){  
 int A[] = {1, 2,  
 3, 4,  
 5, 6}; // Матрица A размером (2, 3)  
 int B[] = {7, 8, 9,  
 10, 11, 12}; // Матрица B размером (3, 2)  
 int m = 2, n = 3, k = 2; // Размерности матриц A, B и C  
 int C[m \* k]; // Массив C для хранения результата  
  
 printf("\nTest1\n");  
  
 matrixMultiplication(A, B, C, m, n, k);  
  
 printf("Результат произведения матриц:\n");  
 for (int i = 0; i < m; i++) {  
 printf("{ ");  
 for (int j = 0; j < k; j++) {  
 printf("%d ", C[i \* k + j]);  
 }  
 printf("}");  
 printf("\n");  
 }  
}  
  
void test2(){  
 int A[] = {1, 2, 3,  
 4, 5, 6,  
 7, 8, 9}; // Матрица A размером (3, 3)  
 int B[] = {7, 8, 9,  
 10, 11, 12,  
 13, 14, 15}; // Матрица B размером (3, 3)  
 int m = 3, n = 3, k = 3; // Размерности матриц A, B и C  
 int C[m \* k]; // Массив C для хранения результата  
  
 printf("\nTest2\n");  
  
 matrixMultiplication(A, B, C, m, n, k);  
  
 printf("Результат произведения матриц:\n");  
 for (int i = 0; i < m; i++) {  
 printf("{ ");  
 for (int j = 0; j < k; j++) {  
 printf("%d ", C[i \* k + j]);  
 }  
 printf("}");  
 printf("\n");  
 }  
}  
  
void test3(){  
 int A[] = {1, 2, 3, 4,  
 5, 6, 7, 8,  
 9, 10, 11, 12,  
 13, 14, 15, 16}; // Матрица A размером (4, 4)  
 int B[] = {7, 8, 9, 10,  
 11, 12, 13 , 14,  
 15, 16, 17, 18,  
 19, 20, 21, 22}; // Матрица B размером (4, 4)  
 int m = 4, n = 4, k = 4; // Размерности матриц A, B и C  
 int C[m \* k]; // Массив C для хранения результата  
  
 printf("\nTest3\n");  
  
 matrixMultiplication(A, B, C, m, n, k);  
  
 printf("Результат произведения матриц:\n");  
 for (int i = 0; i < m; i++) {  
 printf("{ ");  
 for (int j = 0; j < k; j++) {  
 printf("%d ", C[i \* k + j]);  
 }  
 printf("}");  
 printf("\n");  
 }  
}  
  
void test(){  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
 /\* ручной ввод  
 int A[] = {1, 2, 3, 4, 5, 6}; // Матрица A размером (2, 3)  
 int B[] = {7, 8, 9, 10, 11, 12}; // Матрица B размером (3, 2)  
 int m = 2, n = 3, k = 2; // Размерности матриц A, B и C  
 int C[m \* k]; // Массив C для хранения результата  
  
 matrixMultiplication(A, B, C, m, n, k);  
  
 printf("Результат произведения матриц:\n");  
 for (int i = 0; i < m; i++) {  
 printf("{ ");  
 for (int j = 0; j < k; j++) {  
 printf("%d ", C[i \* k + j]);  
 }  
 printf("}");  
 printf("\n");  
 }  
\*/  
 return 0;  
}

Задание 10

Дан текст программы на каком-либо алгоритмическом языке и словарь зарезервированных слов этого языка (в английской транскрипции). Преобразовать текст, записав все зарезервированные слова прописными буквами, а остальные конструкции (имена и так далее) — строчными. Русские буквы (имена, литералы) не заменять.

#include <stdio.h>  
#include <string.h>  
#include <windows.h>  
  
int isChar(char ch) {  
 return ('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z');  
}  
  
  
void checkWord(char \*strWord, char \*strSource) {  
  
 if (strcmp(strWord, "float") == 0 || strcmp(strWord, "int") == 0 || strcmp(strWord, "double") == 0 ||  
 strcmp(strWord, "char") == 0 || strcmp(strWord, "main") == 0 || strcmp(strWord, "return") == 0)  
 for (int i = 0; i < strlen(strWord); i++) {  
  
 strSource[i] = strSource[i] - 32;  
  
  
 }  
}  
  
void test1() {  
 printf("\nTest1\n");  
  
 char buf[100] = "int main() {\n\t int x = 5;\n\t return 0;\n}";  
 printf("Ввод: \"int main() {\\n\\t int x = 5;\\n\\t return 0;\\n}\"");  
 strlwr(buf);  
  
 int i, isWord, iW;  
 char strWord[100], chCurrent, \*ptrWord;  
  
 isWord = 0;  
 iW = 0;  
 for (i = 0; i < strlen(buf); i++) {  
 chCurrent = buf[i];  
 if (isChar(chCurrent)) {  
 if (!isWord) {  
 isWord = 1;  
 ptrWord = (buf + i);  
 }  
 strWord[iW] = chCurrent;  
 iW++;  
 } else {  
 if (isWord) {  
 isWord = 0;  
 strWord[iW] = 0;  
 iW = 0;  
 checkWord(strWord, ptrWord);  
 } else {  
 }  
 }  
 }  
 printf("\nВывод: %s\n", buf);  
}  
  
void test2() {  
 printf("\nTest2\n");  
  
 char buf[100] = "char numOne,\nfloat sugar = 2.3,\nreturn 1";  
 printf("Ввод: char numOne,\nfloat sugar = 2.3,\nreturn 1");  
 strlwr(buf);  
  
 int i, isWord, iW;  
 char strWord[100], chCurrent, \*ptrWord;  
  
 isWord = 0;  
 iW = 0;  
 for (i = 0; i < strlen(buf); i++) {  
 chCurrent = buf[i];  
 if (isChar(chCurrent)) {  
 if (!isWord) {  
 isWord = 1;  
 ptrWord = (buf + i);  
 }  
 strWord[iW] = chCurrent;  
 iW++;  
 } else {  
 if (isWord) {  
 isWord = 0;  
 strWord[iW] = 0;  
 iW = 0;  
 checkWord(strWord, ptrWord);  
 } else {  
 }  
 }  
 }  
 printf("\nВывод: %s\n", buf);  
}  
  
void test3() {  
 printf("\nTest3\n");  
  
 char buf[100] = "int main()\n\tfloat x = 5.7;\n\treturn 0;\n";  
 printf("Ввод: int main()\n\tfloat x = 5.7;\n\treturn 0;\n");  
 strlwr(buf);  
  
 int i, isWord, iW;  
 char strWord[100], chCurrent, \*ptrWord;  
  
 isWord = 0;  
 iW = 0;  
 for (i = 0; i < strlen(buf); i++) {  
 chCurrent = buf[i];  
 if (isChar(chCurrent)) {  
 if (!isWord) {  
 isWord = 1;  
 ptrWord = (buf + i);  
 }  
 strWord[iW] = chCurrent;  
 iW++;  
 } else {  
 if (isWord) {  
 isWord = 0;  
 strWord[iW] = 0;  
 iW = 0;  
 checkWord(strWord, ptrWord);  
 } else {  
 }  
 }  
 }  
 printf("\nВывод: %s\n", buf);  
}  
  
void test() {  
 test1();  
 test2();  
 test3();  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 test();  
/\* ручной ввод  
 char buf[100];  
 printf("\nРучной ввод\n);  
 printf("Enter text: ");  
 fgets(buf, 100, stdin);  
 strlwr(buf);  
  
 int i, isWord, iW;  
 char strWord[100], chCurrent, \*ptrWord;  
  
 isWord = 0;  
 iW = 0;  
 for (i = 0; i < strlen(buf); i++) {  
 chCurrent = buf[i];  
 if (isChar(chCurrent)) {  
 if (!isWord) {  
 isWord = 1;  
 ptrWord = (buf + i);  
 }  
 strWord[iW] = chCurrent;  
 iW++;  
 } else {  
 if (isWord) {  
 isWord = 0;  
 strWord[iW] = 0;  
 iW = 0;  
 checkWord(strWord, ptrWord);  
 } else {  
 }  
 }  
 }  
 printf("Result text: %s\n", buf);  
 \*/  
 return 0;  
}

Задание 11

Построить прямую 3x + 2y – 4 = 0 в диапазоне x ∈ [-1;3] с шагом ∆ = 0,25.

Задание 12

Построить верхнюю часть эллипса 0,1 ≤ x ≤ 5,1 с шагом ∆ = 0,25.

Задание 13

Найдите точку равновесия в заданном диапазоне с заданным шагом.

Задание 14

Построить плоскость, проходящую через точки , при -1 ≤ x ≤ 4 с шагом ∆ = 0,5 и -1 ≤ y ≤ 3 с шагом ∆ = 1.