МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №12**

по дисциплине: Основы программирования

тема: «Динамические переменные»

Выполнил: ст. группы ПВ-201

Машуров Дмитрий Русланович

Проверил:

Притчин Иван Сергеевич

Белгород 2020 г.

**Лабораторная работа №12**

**«Динамические переменные»**

**Цель работы:** получение навыков работы с указателями и динамическими переменными структурированных типов.

**Задания для подготовки к работе:**

1. Изучите ссылочный тип и его использование для создания динамических переменных и работы с ними.
2. Рассмотрите возможные способы хранения матриц в динамически распределяемой области памяти. Изобразите схемы хранения для каждого случая.
3. Разработайте алгоритм и составьте программы для решения задачи соответствующего варианта для четырех случаев, матрицы следует разместить в "куче" при выполнении следующих условий: a) число строк и число столбцов − константы; b) число строк − константа, а число столбцов − исходное данное; c) число строк − исходное данное, число столбцов − константа; d) число строк и число столбцов − исходные данные.
4. Ввод, вывод и обработку матриц опишите отдельными подпрограммами. Для случаев a) − d), где возможно, используйте одни и те же подпрограммы.
5. В блок-схемах обработки матриц не используйте операции разыменования, а обращайтесь к элементам матрицы как к элементам двумерного массива.
6. Опишите блок-схему алгоритма решения задачи с использованием блоков «предопределенный процесс».
7. Закодируйте алгоритм.
8. Подберите наборы тестовых данных с обоснованием их выбора.

**Задания к работе:**

1. Наберите программу, отладьте ее и протестируйте.
2. Выполните анализ ошибок, выявленных при отладке программы

**Задание варианта №17:**

Дана матрица. Упорядочить ее строки по убыванию первых элементов строк, если это возможно.

**Выполнение работы:**

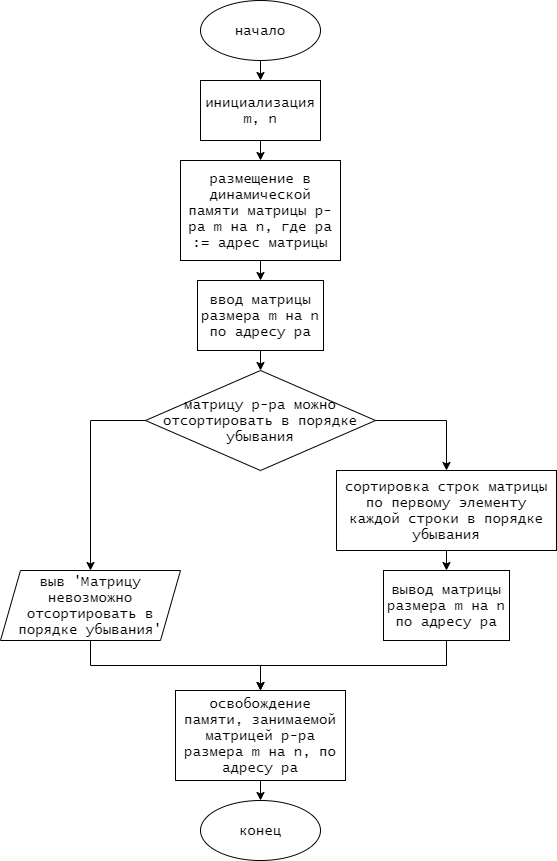
1. **Выделение подзадач**

Выделим следующие подзадачи:

* Размещение в динамической памяти матрицы размера MxN, pa – адрес матрицы
* Ввод матрицы MxN по адресу pa
* Вывод матрицы MxN по адресу pa
* Проверка на то, можно ли отсортировать строки матрицы по убыванию первых элементов строк
* Упорядочивание строк матрицы по первому элементу каждой строки в порядке убывания
* Освобождение памяти занимаемой матрицей размера MxN по адресу pa

Далее описание алгоритма приводится в блок-схеме с укрупнёнными блоками в терминах выделенных подзадач

1. **Блок-схема алгоритма в укрупнённых блоках:**

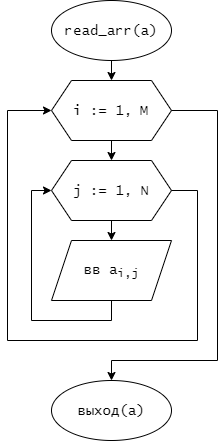
****

1. **Описание подпрограмм:**

Спецификация процедуры read\_arr

1. Заголовок: procedure read\_arr(var a: t\_matr)
2. Назначение: ввод матрицы a размера MxN
3. Входные параметры: нет
4. Выходные параметры: a

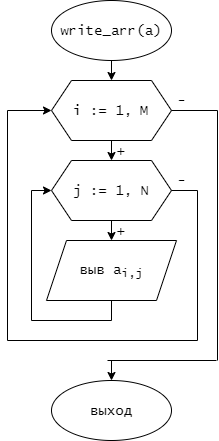
Блок-схема:



Спецификация процедуры write\_arr

1. Заголовок: procedure write\_arr(a: t\_matr)
2. Назначение: вывод матрицы a размера MxN
3. Входные параметры: a
4. Выходные параметры: нет

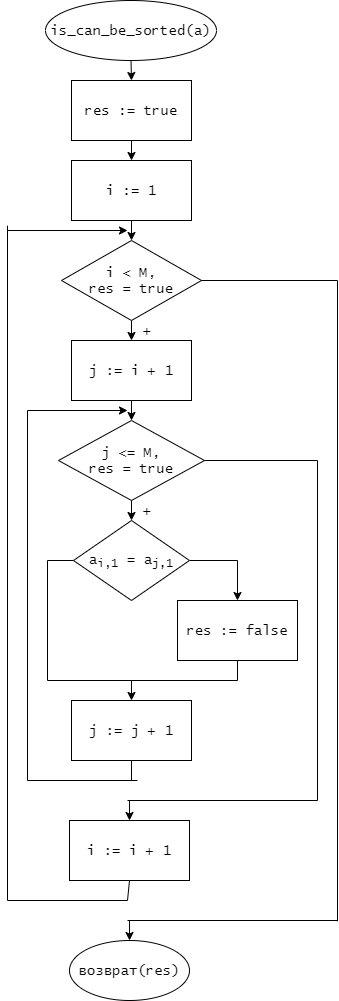
Блок-схема:



Спецификация функции is\_can\_be\_sorted

1. Заголовок: function is\_can\_be\_sorted(a: t\_matr): boolean
2. Назначение: возвращает значение «истина», если матрицу a размера MxN можно отсортировать в порядке убывания по первым элементам строк, иначе – «ложь». Применяется перед использованием процедуры sort
3. Входные параметры: a
4. Выходные параметры: нет

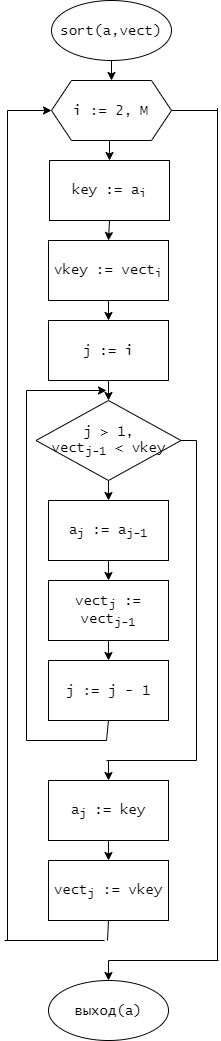
Блок-схема:



Спецификация процедуры sort

1. Заголовок: procedure sort(var a: t\_matr; vect: t\_vect)
2. Назначение: сортировка матрицы a размера MxN по указанному вектору vect размера M в порядке убывания
3. Входные параметры: a, vect
4. Выходные параметры: a

Блок-схема:



1. **Различные случаи алгоритма**
2. **Число строк и число столбцов – константы**
3. Описание структур данных:

const M = 4;

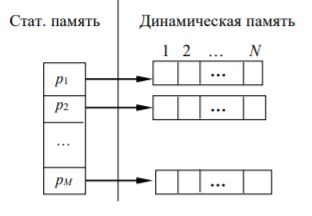
N = 4;

type t\_row = array[1..N] of integer;

t\_p\_row = ^t\_row;

t\_matr = array[1..M] of t\_p\_row;

1. Схема размещения в ДП:



1. Подпрограмма размещения матрицы в ДП:

procedure create\_matr(var a: t\_matr);

var i: byte;

begin

for i := 1 to M do

New(a[i]);

end;

1. Обращение к строке и к элементу матрицы:

Поскольку матрица – массив указателей на строки:

Обращение к строке матрицы: a[i]^

Обращение к элементу матрицы: a[i]^[j]

1. Заголовок процедуры ввода матрицы:

procedure read\_matr(a: t\_matr)

1. Подпрограмма освобождения памяти:

procedure del\_matr(var a: t\_matr);

var i: byte;

begin

for i := 1 to M do

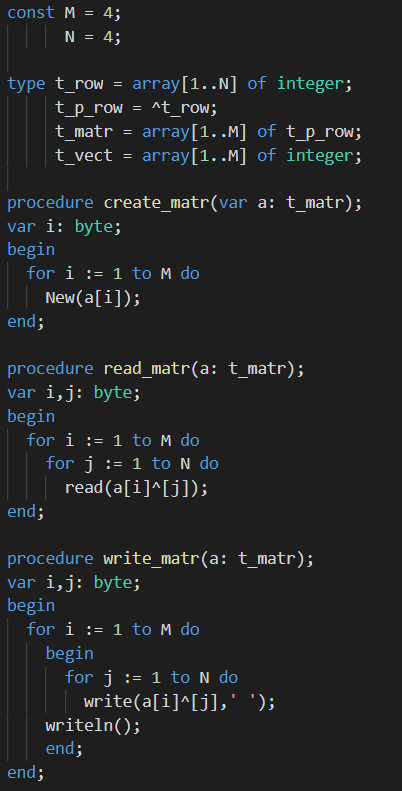
Dispose(a[i]);

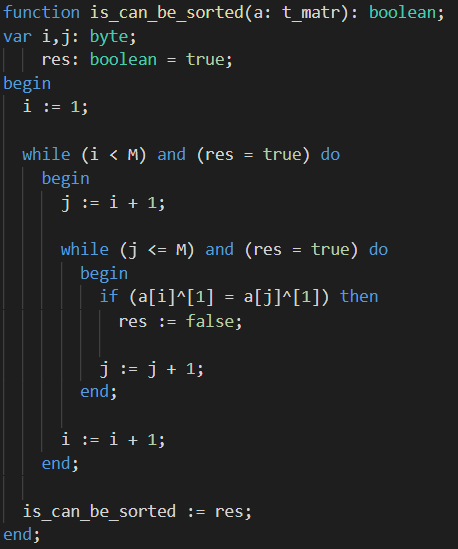
end;

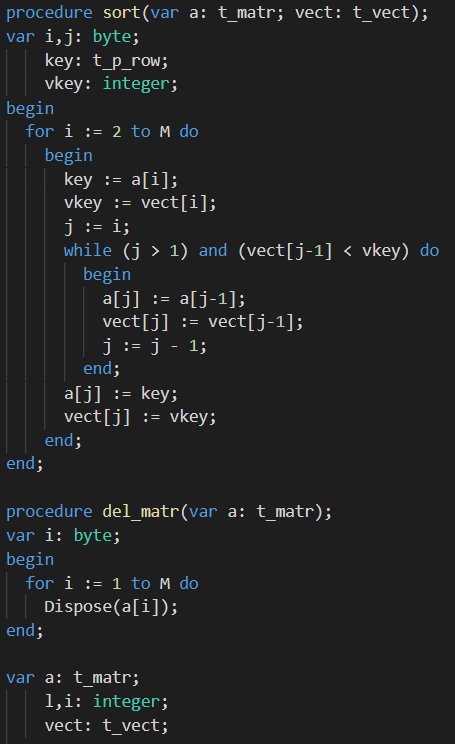
1. Тестовые данные:

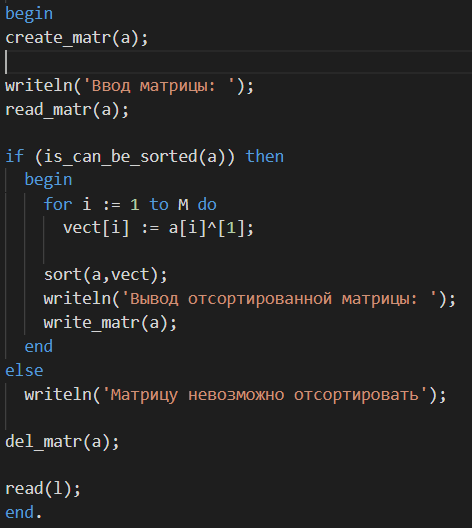
|  |  |  |
| --- | --- | --- |
| № | Вход | Выход |
| 1 |  |  |
| 2 |  | Матрицу невозможно отсортировать |

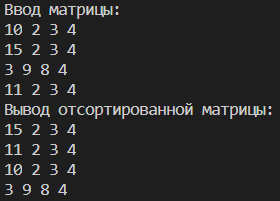
1. Скриншоты программы:

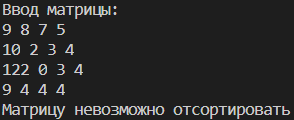












1. **Число строк - константа, число столбцов – исходное данное**
2. Описание структур данных

const MAX = 20000;

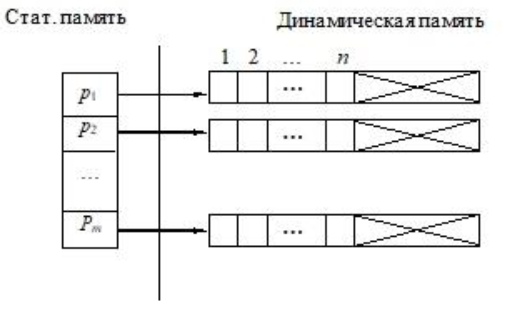
M = 4;

type t\_max\_row = array[1..MAX div SizeOf(integer)] of integer;

t\_p\_max\_row = ^t\_max\_row;

t\_matr = array[1..M] of t\_p\_max\_row;

1. Схема размещения матрицы в ДП:



1. Подпрограмма размещения матрицы в ДП:

procedure create\_matr(var a: t\_matr; n: byte);

var i: byte;

begin

for i := 1 to M do

GetMem(a[i],n\*SizeOf(integer));

end;

1. Обращение к строке и элементу матрицы:

Поскольку матрица – массив указателей на строки:

Обращение к строке матрицы: a[i]^

Обращение к элементу матрицы: a[i]^[j]

1. Заголовок программы вводы матрицы:

procedure read\_matr(a: t\_matr; n: byte);

1. Подпрограмма освобождения памяти:

procedure del\_matr(var a: t\_matr; n: byte);

var i: byte;

begin

for i := 1 to M do

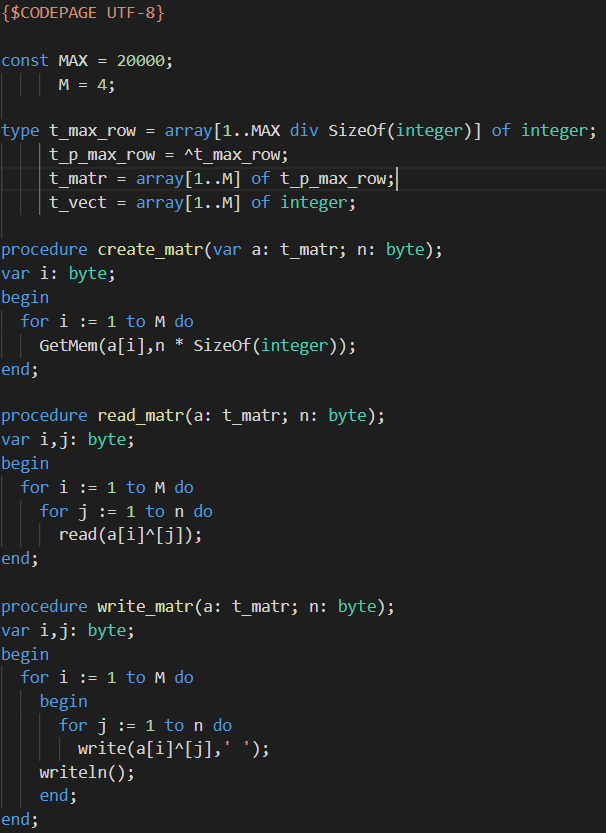
FreeMem(a[i],n\*SizeOf(integer));

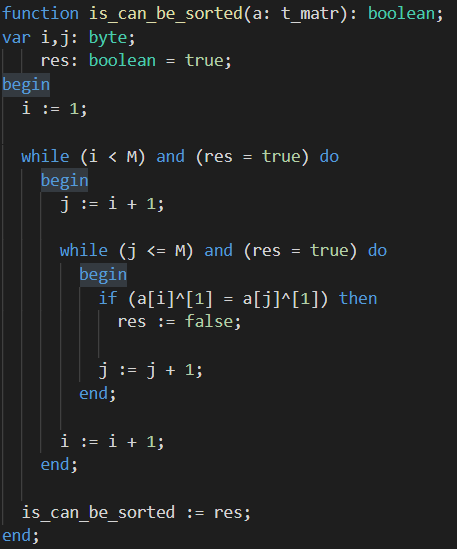
end;

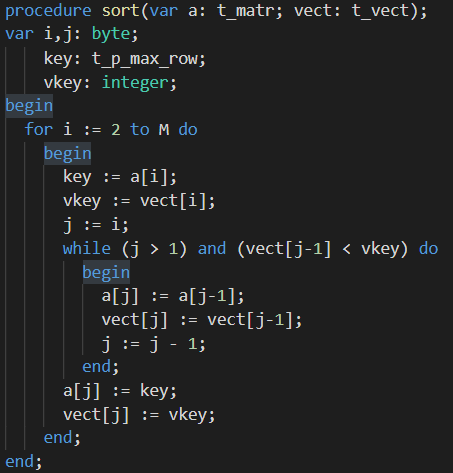
1. Тестовые данные:

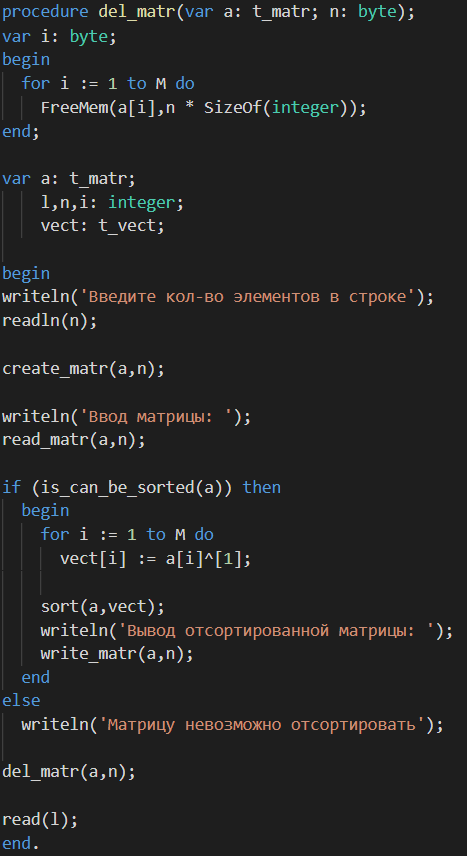
|  |  |  |
| --- | --- | --- |
| № | Вход | Выход |
| 1 |  |  |
| 2 |  | Матрицу невозможно отсортировать |

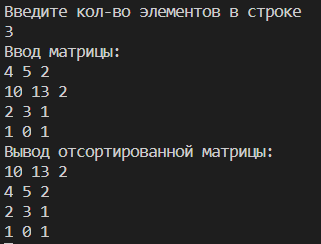
1. Скриншоты:

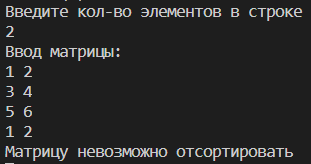












1. **Число строк – исходное данное, число столбцов – константа:**
2. Описание структур данных:

const MAX = 20000;

N = 4;

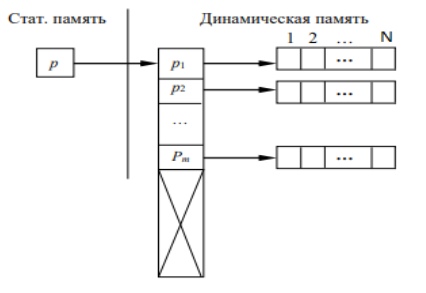
type t\_row = array[1..N] of integer;

t\_p\_row = ^t\_row;

t\_max\_matr = array[1..MAX div SizeOf(pointer)] of t\_p\_row

t\_p\_max\_matr = ^t\_max\_matr;

1. Схема размещения матрицы в ДП:



1. Подпрограмма размещения матрицы в ДП:

procedure create\_matr(var pa: t\_p\_max\_matr; m: byte);

var i: byte;

begin

GetMem(pa,m\*SizeOf(pointer));

for i := 1 to m do

New(pa^[i]);

end;

1. Обращение к строке и к элементу матрицы:

Поскольку матрица – указатель на массив указателей:

pa^[i]^ - обращение к строке

pa^[i]^[j] – обращение к элементу

1. Заголовок процедуры ввода матрицы:

procedure read\_matr(pa: t\_p\_max\_matr; m: byte);

1. Подпрограмма освобождения памяти:

procedure del\_matr(var pa: t\_p\_max\_matr; m: byte);

var i: byte;

begin

for i := 1 to m do

Dispose(pa^[i]);

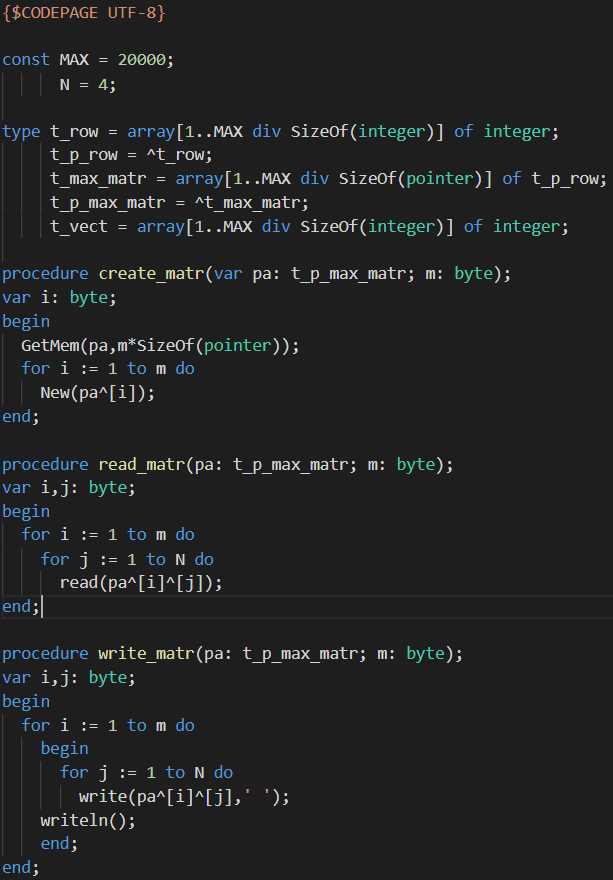
FreeMem(pa,m\*SizeOf(pointer));

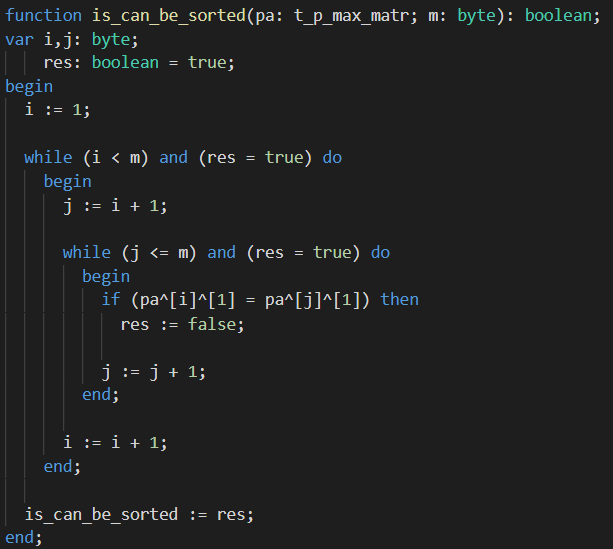
end;

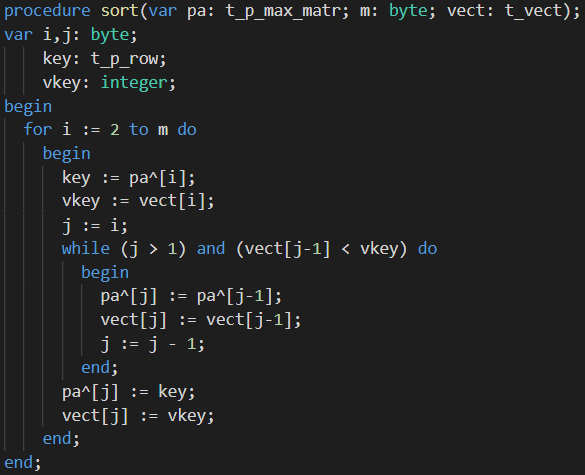
1. Тестовые данные:

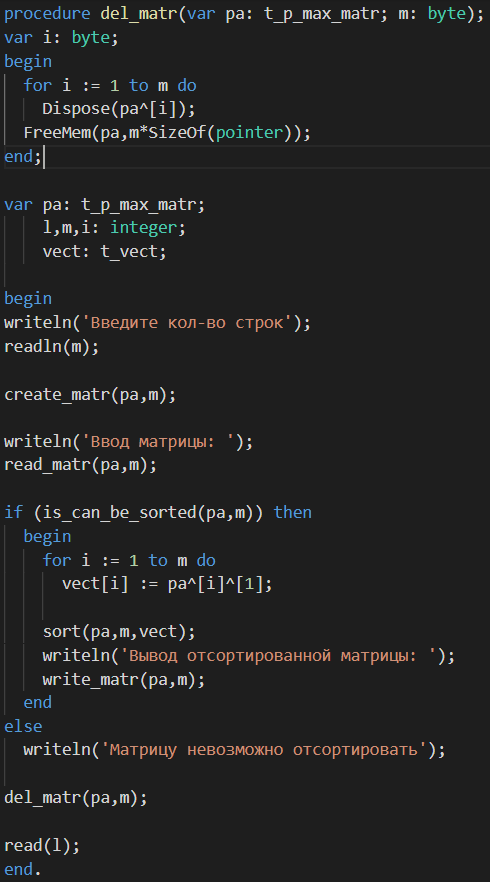
|  |  |  |
| --- | --- | --- |
| № | Вход | Выход |
| 1 |  |  |
| 2 |  | Матрицу невозможно отсортировать |

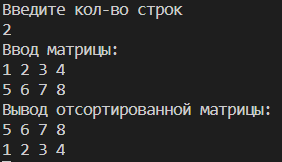
1. Скриншоты:

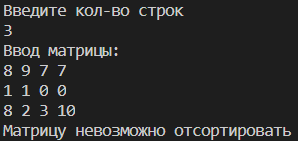












1. **Число строк и число столбцов – исходные данные:**
2. Описание структур данных:

const MAX = 20000;

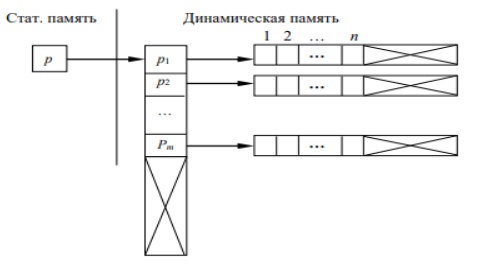
type t\_max\_row = array[1..MAX div SizeOf(integer)] of integer;

t\_p\_max\_row = ^t\_max\_row;

t\_max\_matr = array[1..MAX div SizeOf(pointer)] of t\_p\_max\_row;

t\_p\_max\_matr = ^t\_max\_matr;

1. Схема размещения матрицы в ДП:



1. Подпрограмма размещения матрицы в ДП:

procedure create\_matr(var pa: t\_p\_max\_matr; m,n: byte);

var i: byte;

begin

GetMem(pa,m\*SizeOf(pointer));

for i := 1 to m do

GetMem(pa^[i],n\*SizeOf(integer));

end;

1. Обращение к строке и к элементу матрицы:

Поскольку матрица – указатель на массив указателей:

pa^[i]^ - обращение к строке

pa^[i]^[j] – обращение к элементу

1. Заголовок процедуры ввода матрицы:

procedure read\_matr(pa: t\_p\_max\_matr; m,n: byte);

1. Подпрограмма освобождения памяти:

procedure del\_matr(var pa: t\_p\_max\_matr; m,n: byte);

var i: byte;

begin

for i := 1 to m do

FreeMem(pa^[i],n\*SizeOf(integer));

FreeMem(pa,m\*SizeOf(pointer));

end;

1. Тестовые данные:

|  |  |  |
| --- | --- | --- |
| № | Вход | Выход |
| 1 |  |  |
| 2 |  | Матрицу невозможно отсортировать |

1. Скриншоты:

