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:

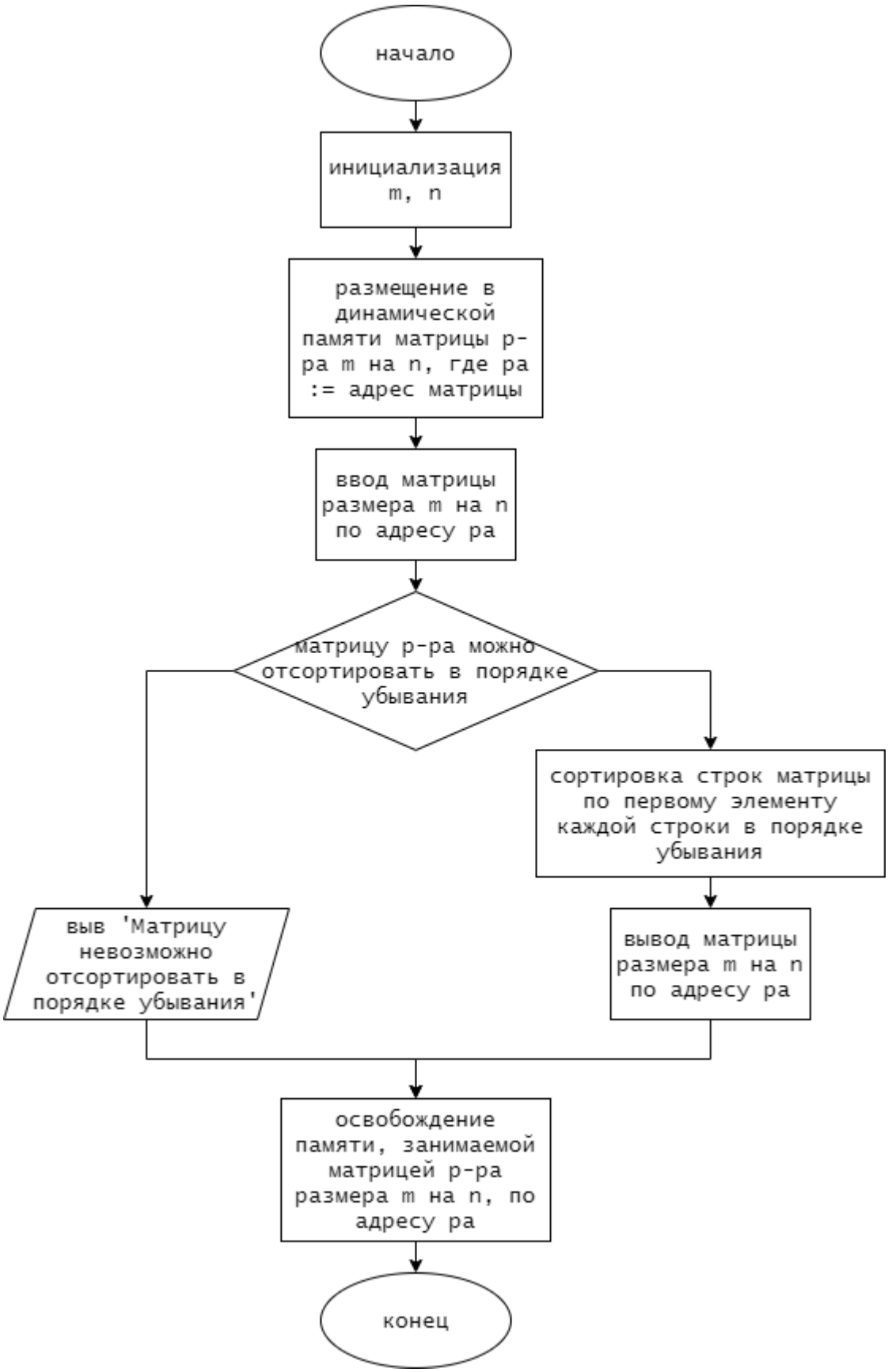
Дана матрица. Упорядочить ее строки по убыванию первых элементов строк, если это возможно.

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

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

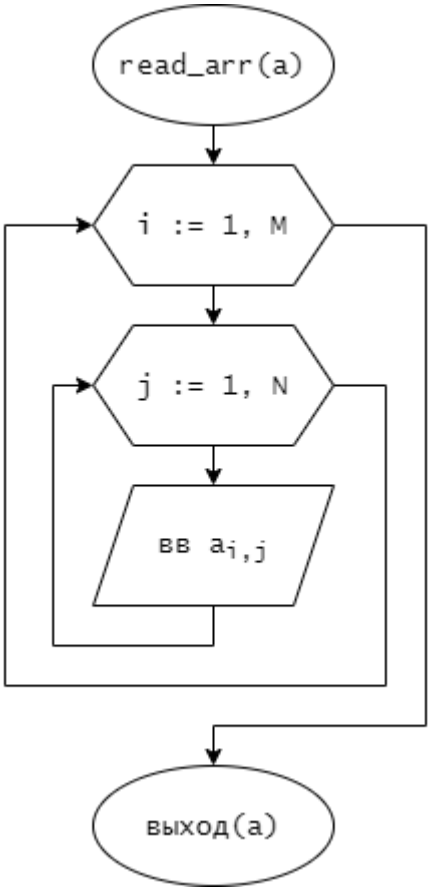

начало
инициализация m, n
размещение в динамической памяти матрицы р-ра m на n, где ра := адрес матрицы
ввод матрицы размера m на n по адресу ра
матрицу р-ра можно отсортировать в порядке убывания
сортировка строк матрицы по первому элементу каждой строки в порядке убывания
выв 'Матрицу невозможно отсортировать в порядке убывания'
вывод матрицы размера m на n по адресу ра
освобождение памяти, занимаемой матрицей р-ра размера m на n, по адресу ра
конец

3. **Описание подпрограмм:**

   Спецификация процедуры `read_arr`

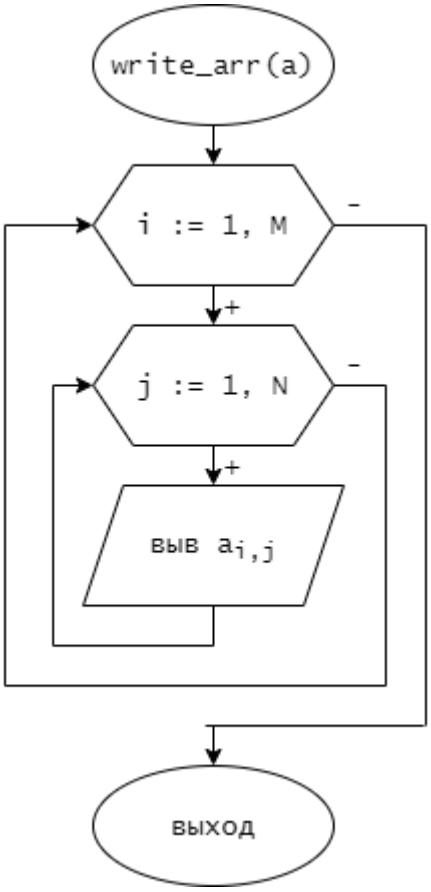
   1) Заголовок: `procedure read_arr(var a: t_matr)`
   2) Назначение: ввод матрицы `a` размера `MxN`
   3) Входные параметры: `нет`
   4) Выходные параметры: `a`

      Блок-схема:



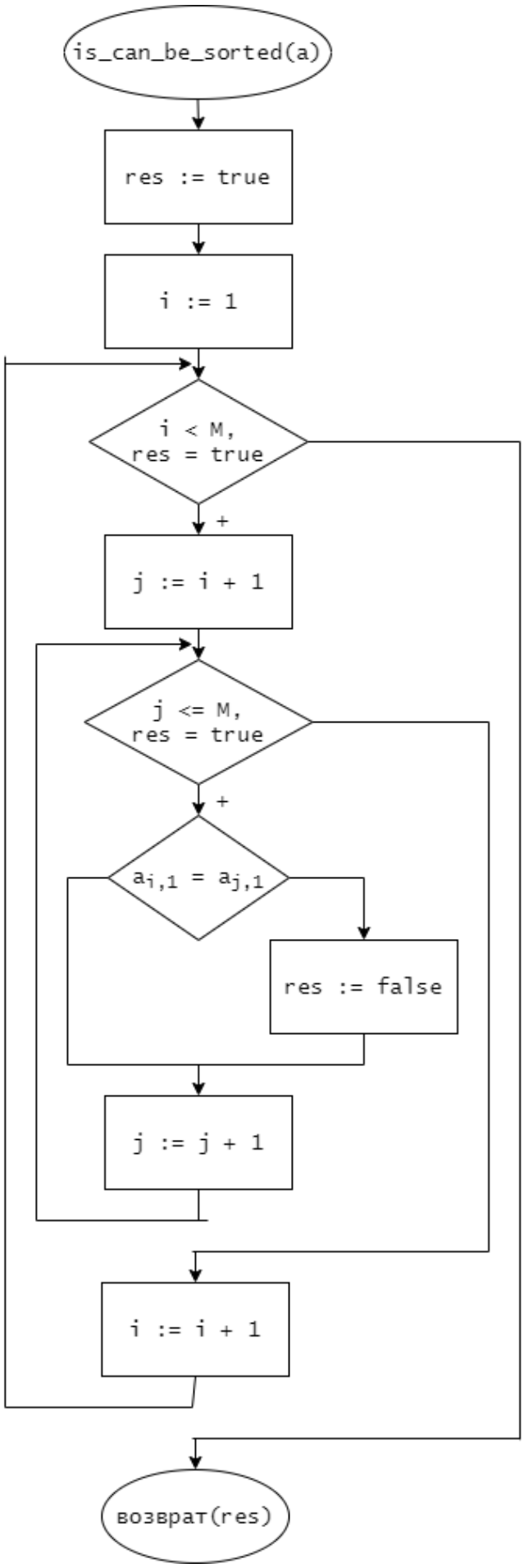
Спецификация процедуры `write_arr`

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:



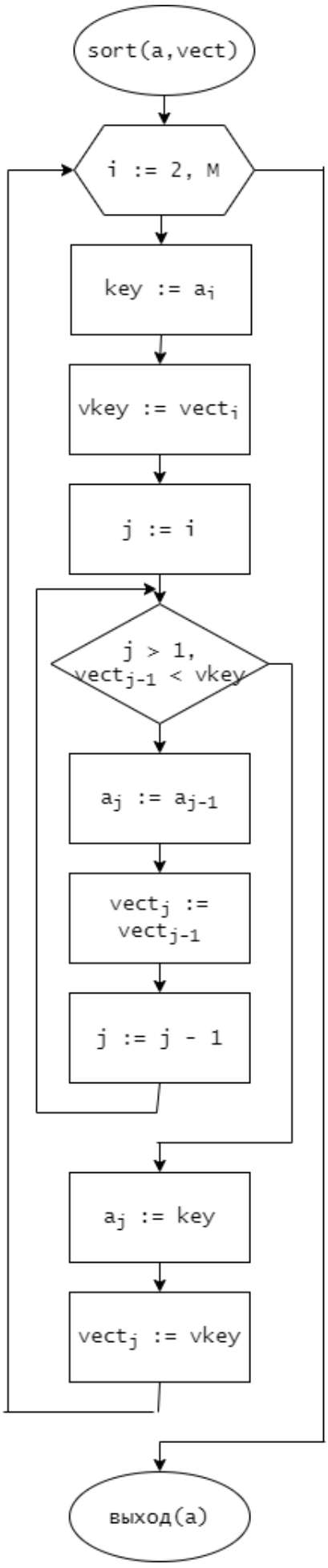
Спецификация функции `is_can_be_sorted`

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:


is_can_be_sorted(a)
res := true
i := 1
i < M,
res = true
+
j := i + 1
j <= M,
res = true
+
$a_{i,1} = a_{j,1}$
res := false
j := j + 1
i := i + 1
возврат(res)

Спецификация процедуры `sort`

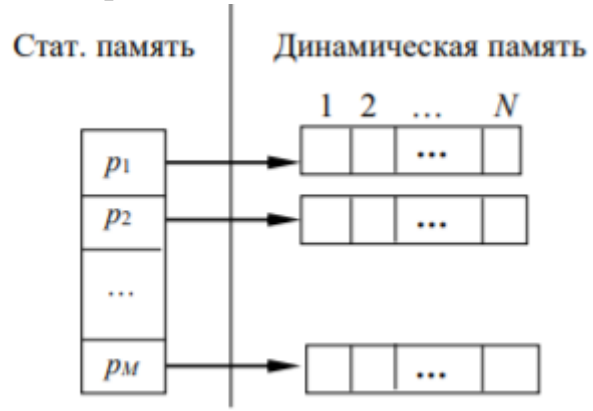
1) Заголовок: `procedure sort(var a: t_matr; vect: t_vect)`
2) Назначение: сортировка матрицы `a` размера `MxN` по указанному вектору `vect` размера `M` в порядке убывания
3) Входные параметры: `a`, `vect`
4) Выходные параметры: `a`
   Блок-схема:


sort(a,vect)
i := 2, M
key := a_i
vkey := $vect_i$
j := i
j > 1,
$vect_{j-1}$ < vkey
a_j := a_{j-1}
$vect_j$:= $vect_{j-1}$
j := j - 1
a_j := key
$vect_j$:= vkey
выход(a)

4. **Различные случаи алгоритма**
    1) **Число строк и число столбцов – константы**
        a) 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;
```

        b) Схема размещения в ДП:



        c) 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;
```

        d) 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]`
        e) Заголовок процедуры ввода матрицы:

```
procedure read_matr(a: t_matr)
```

        f) Подпрограмма освобождения памяти:

```
procedure del_matr(var a: t_matr);
var i: byte;
begin
  for i := 1 to M do
    Dispose(a[i]);
end;
```

g) Тестовые данные:

№	Вход	Выход
1	10 2 3 4 15 2 3 4 3 9 8 4 11 2 3 4	15 2 3 4 11 2 3 4 10 2 3 4 3 9 8 4
2	9 8 7 5 10 2 3 4 122 0 3 4 9 4 4 4	Матрицу невозможно отсортировать

h) Скриншоты программы:

```
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;
     t_vect = array[1..M] of integer;

procedure create_matr(var a: t_matr);
var i: byte;
begin
  for i := 1 to M do
    New(a[i]);
end;

procedure read_matr(a: t_matr);
var i,j: byte;
begin
  for i := 1 to M do
    for j := 1 to N do
      read(a[i]^[j]);
end;

procedure write_matr(a: t_matr);
var i,j: byte;
begin
  for i := 1 to M do
    begin
      for j := 1 to N do
        write(a[i]^[j],' ');
    writeln();
    end;
end;
```

```
function is_can_be_sorted(a: t_matr): boolean;
var i,j: byte;
    res: boolean = true;
begin
  i := 1;

  while (i < M) and (res = true) do
    begin
      j := i + 1;

      while (j <= M) and (res = true) do
        begin
          if (a[i]^[1] = a[j]^[1]) then
            res := false;

          j := j + 1;
        end;

      i := i + 1;
    end;

  is_can_be_sorted := res;
end;
```

```
procedure sort(var a: t_matr; vect: t_vect);
var i,j: byte;
    key: t_p_row;
    vkey: integer;
begin
  for i := 2 to M do
    begin
      key := a[i];
      vkey := vect[i];
      j := i;
      while (j > 1) and (vect[j-1] < vkey) do
        begin
          a[j] := a[j-1];
          vect[j] := vect[j-1];
          j := j - 1;
        end;
      a[j] := key;
      vect[j] := vkey;
    end;
end;

procedure del_matr(var a: t_matr);
var i: byte;
begin
  for i := 1 to M do
    Dispose(a[i]);
end;

var a: t_matr;
    l,i: integer;
    vect: t_vect;
```

```
begin
create_matr(a);

writeln('Ввод матрицы: ');
read_matr(a);

if (is_can_be_sorted(a)) then
  begin
    for i := 1 to M do
      vect[i] := a[i]^[1];

    sort(a,vect);
    writeln('Вывод отсортированной матрицы: ');
    write_matr(a);
  end
else
  writeln('Матрицу невозможно отсортировать');

del_matr(a);

read(l);
end.
```

```
Ввод матрицы:
10 2 3 4
15 2 3 4
3 9 8 4
11 2 3 4
Вывод отсортированной матрицы:
15 2 3 4
11 2 3 4
10 2 3 4
3 9 8 4
```

```
Ввод матрицы:
9 8 7 5
10 2 3 4
122 0 3 4
9 4 4 4
Матрицу невозможно отсортировать
```

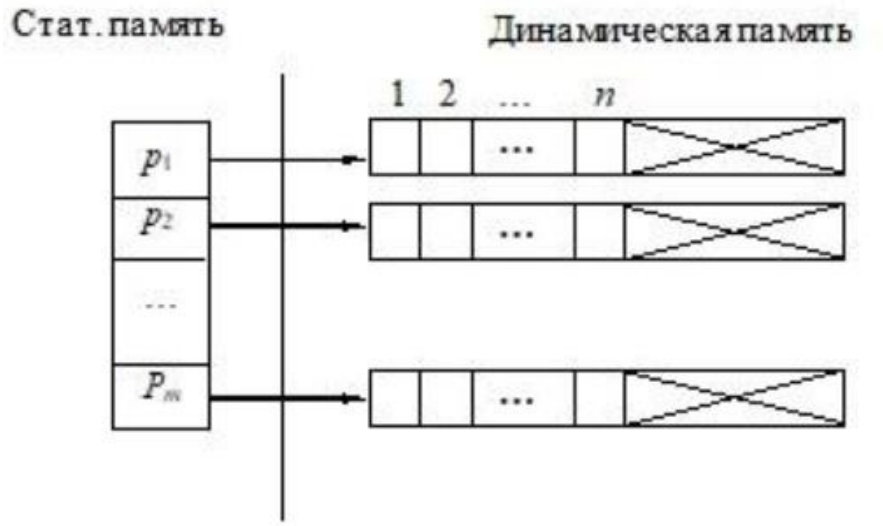
2) **Число строк - константа, число столбцов – исходное данное**

a) 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;
```

b) Схема размещения матрицы в ДП:



c) 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;
```

d) 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]`

e) Заголовок программы вводы матрицы:

```
procedure read_matr(a: t_matr; n: byte);
```

f) 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;
```

g) Тестовые данные:

№	Вход	Выход
1	$n = 3$ $4 \quad 5 \quad 2$ $10 \quad 13 \quad 2$ $2 \quad 3 \quad 1$ $1 \quad 0 \quad 1$	$10 \quad 13 \quad 2$ $4 \quad 5 \quad 2$ $2 \quad 3 \quad 1$ $1 \quad 0 \quad 1$
2	$n = 2$ $1 \quad 2$ $3 \quad 4$ $5 \quad 6$ $1 \quad 2$	Матрицу невозможно отсортировать

h) Скриншоты:

```
{$CODEPAGE UTF-8}

const MAX = 20000;
      M = 4;

type t_max_row = array[1..MAX div SizeOf(integer)] of in
     t_p_max_row = ^t_max_row;
     t_matr = array[1..M] of t_p_max_row;
     t_vect = array[1..M] of integer;

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;

procedure read_matr(a: t_matr; n: byte);
var i,j: byte;
begin
  for i := 1 to M do
    for j := 1 to n do
      read(a[i]^[j]);
end;

procedure write_matr(a: t_matr; n: byte);
var i,j: byte;
begin
  for i := 1 to M do
    begin
      for j := 1 to n do
        write(a[i]^[j],' ');
    writeln();
    end;
end;
```

```
function is_can_be_sorted(a: t_matr): boolean;
var i,j: byte;
    res: boolean = true;
begin
  i := 1;

  while (i < M) and (res = true) do
    begin
      j := i + 1;

      while (j <= M) and (res = true) do
        begin
          if (a[i]^[1] = a[j]^[1]) then
            res := false;

          j := j + 1;
        end;

      i := i + 1;
    end;

  is_can_be_sorted := res;
end;
```

```
procedure sort(var a: t_matr; vect: t_vect);
var i,j: byte;
    key: t_p_max_row;
    vkey: integer;
begin
  for i := 2 to M do
    begin
      key := a[i];
      vkey := vect[i];
      j := i;
      while (j > 1) and (vect[j-1] < vkey) do
        begin
          a[j] := a[j-1];
          vect[j] := vect[j-1];
          j := j - 1;
        end;
      a[j] := key;
      vect[j] := vkey;
    end;
end;
```

```
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;

var a: t_matr;
    l,n,i: integer;
    vect: t_vect;

begin
writeln('Введите кол-во элементов в строке');
readln(n);

create_matr(a,n);

writeln('Ввод матрицы: ');
read_matr(a,n);

if (is_can_be_sorted(a)) then
  begin
    for i := 1 to M do
      vect[i] := a[i]^[1];

    sort(a,vect);
    writeln('Вывод отсортированной матрицы: ');
    write_matr(a,n);
  end
else
  writeln('Матрицу невозможно отсортировать');

del_matr(a,n);

read(l);
end.
```

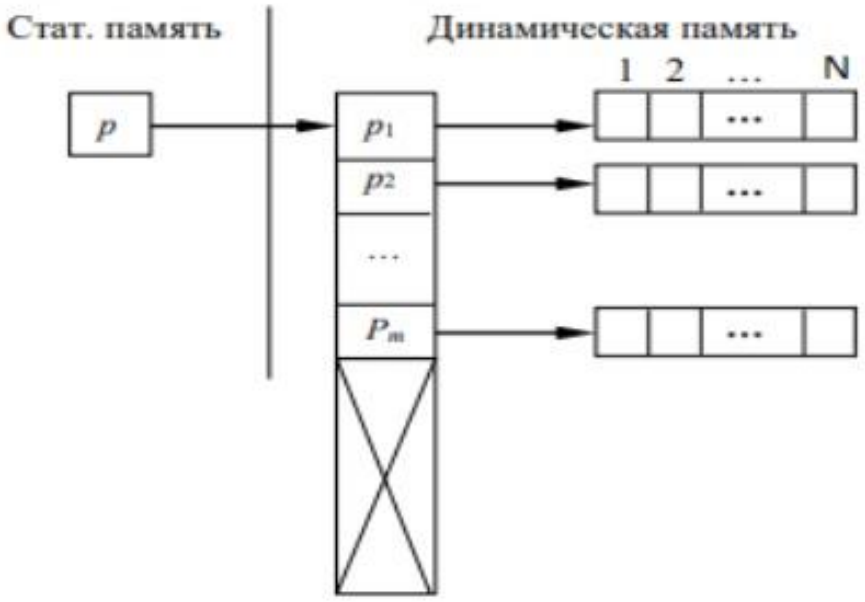
```
Введите кол-во элементов в строке
3
Ввод матрицы:
4 5 2
10 13 2
2 3 1
1 0 1
Вывод отсортированной матрицы:
10 13 2
4 5 2
2 3 1
1 0 1
```

```
Введите кол-во элементов в строке
2
Ввод матрицы:
1 2
3 4
5 6
1 2
Матрицу невозможно отсортировать
```

3) **Число строк – исходное данное, число столбцов – константа:**
   a) 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;
```

   b) Схема размещения матрицы в ДП:



c) 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;
```

d) Обращение к строке и к элементу матрицы:
Поскольку матрица – указатель на массив указателей:
`pa^[i]^` - обращение к строке
`pa^[i]^[j]` – обращение к элементу

e) Заголовок процедуры ввода матрицы:

```
procedure read_matr(pa: t_p_max_matr; m:
byte);
```

f) 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;
```

g) Тестовые данные:

№	Вход	Выход
1	$m = 2$ 1 2 3 4 5 6 7 8	5 6 7 8 1 2 3 4
2	$m = 3$ 8 9 7 7 1 1 0 0 8 2 3 10	Матрицу невозможно отсортировать

h) Скриншоты:

```
{$CODEPAGE UTF-8}

const MAX = 20000;
      N = 4;

type t_row = array[1..MAX div SizeOf(integer)] of intege
     t_p_row = ^t_row;
     t_max_matr = array[1..MAX div SizeOf(pointer)] of t
     t_p_max_matr = ^t_max_matr;
     t_vect = array[1..MAX div SizeOf(integer)] of integ

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;

procedure read_matr(pa: t_p_max_matr; m: byte);
var i,j: byte;
begin
  for i := 1 to m do
    for j := 1 to N do
      read(pa^[i]^[j]);
end;

procedure write_matr(pa: t_p_max_matr; m: byte);
var i,j: byte;
begin
  for i := 1 to m do
    begin
      for j := 1 to N do
        write(pa^[i]^[j],' ');
    writeln();
    end;
end;
```

```
function is_can_be_sorted(pa: t_p_max_matr; m: byte): b
var i,j: byte;
    res: boolean = true;
begin
  i := 1;

  while (i < m) and (res = true) do
    begin
      j := i + 1;

      while (j <= m) and (res = true) do
        begin
          if (pa^[i]^[1] = pa^[j]^[1]) then
            res := false;

          j := j + 1;
        end;

      i := i + 1;
    end;

  is_can_be_sorted := res;
end;
```

```
procedure sort(var pa: t_p_max_matr; m: byte; vect: t_v
var i,j: byte;
    key: t_p_row;
    vkey: integer;
begin
  for i := 2 to m do
    begin
      key := pa^[i];
      vkey := vect[i];
      j := i;
      while (j > 1) and (vect[j-1] < vkey) do
        begin
          pa^[j] := pa^[j-1];
          vect[j] := vect[j-1];
          j := j - 1;
        end;
      pa^[j] := key;
      vect[j] := vkey;
    end;
end;
```

```
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;

var pa: t_p_max_matr;
    l,m,i: integer;
    vect: t_vect;

begin
writeln('Введите кол-во строк');
readln(m);

create_matr(pa,m);

writeln('Ввод матрицы: ');
read_matr(pa,m);

if (is_can_be_sorted(pa,m)) then
  begin
    for i := 1 to m do
      vect[i] := pa^[i]^[1];

    sort(pa,m,vect);
    writeln('Вывод отсортированной матрицы: ');
    write_matr(pa,m);
  end
else
  writeln('Матрицу невозможно отсортировать');

del_matr(pa,m);

read(l);
end.
```

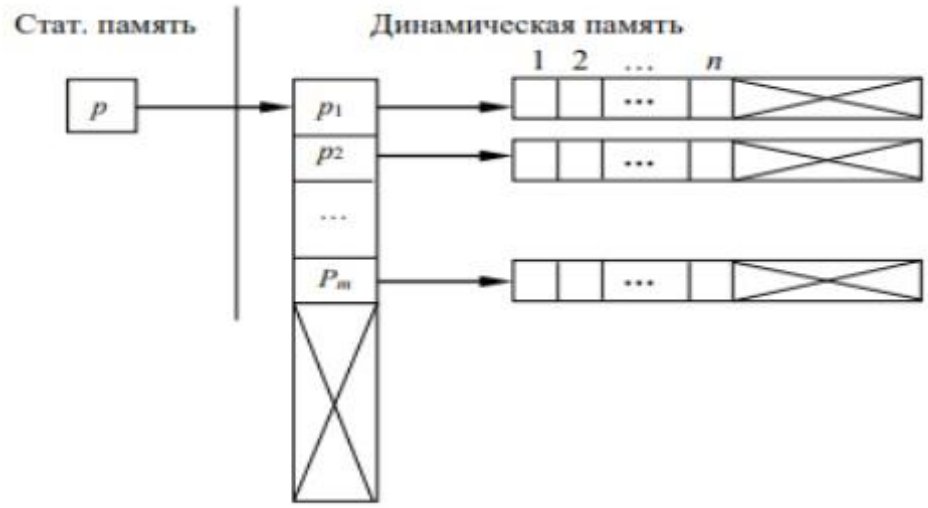
```
Введите кол-во строк
2
Ввод матрицы:
1 2 3 4
5 6 7 8
Вывод отсортированной матрицы:
5 6 7 8
1 2 3 4
```

```
Введите кол-во строк
3
Ввод матрицы:
8 9 7 7
1 1 0 0
8 2 3 10
Матрицу невозможно отсортировать
```

4) **Число строк и число столбцов – исходные данные:**
   a) 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;
```

   b) Схема размещения матрицы в ДП:



   c) 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;
```

d) Обращение к строке и к элементу матрицы:
Поскольку матрица – указатель на массив указателей:
`pa^[i]^` - обращение к строке
`pa^[i]^[j]` – обращение к элементу

e) Заголовок процедуры ввода матрицы:
```
procedure read_matr(pa: t_p_max_matr; m,n:
byte);
```

f) 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;
```

g) Тестовые данные:

№	Вход	Выход
1	$m = 3, n = 2$ 4 5 10 9 1 0	10 9 4 5 1 0
2	$m = 5, n = 3$ 4 5 6 13 20 3 100 2 3 4 6 7 9 9 9	Матрицу невозможно отсортировать

h) Скриншоты:

```
{$CODEPAGE UTF-8}

const MAX = 20000;

type t_max_row = array[1..MAX div SizeOf(integer)] of inte
     t_p_max_row = ^t_max_row;
     t_max_matr = array[1..MAX div SizeOf(pointer)] of t_p
     t_p_max_matr = ^t_max_matr;
     t_vect = array[1..MAX div SizeOf(integer)] of integer

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;

procedure read_matr(pa: t_p_max_matr; m,n: byte);
var i,j: byte;
begin
  for i := 1 to m do
    for j := 1 to n do
      read(pa^[i]^[j]);
end;

procedure write_matr(pa: t_p_max_matr; m,n: byte);
var i,j: byte;
begin
  for i := 1 to m do
    begin
      for j := 1 to n do
        write(pa^[i]^[j],' ');
    writeln();
    end;
end;
```

```
function is_can_be_sorted(pa: t_p_max_matr; m: byte): bo
var i,j: byte;
    res: boolean = true;
begin
  i := 1;

  while (i < m) and (res = true) do
    begin
      j := i + 1;

      while (j <= m) and (res = true) do
        begin
          if (pa^[i]^[1] = pa^[j]^[1]) then
            res := false;

          j := j + 1;
        end;

      i := i + 1;
    end;

  is_can_be_sorted := res;
end;
```

```
procedure sort(var pa: t_p_max_matr; m: byte; vect: t_ve
var i,j: byte;
    key: t_p_max_row;
    vkey: integer;
begin
  for i := 2 to m do
    begin
      key := pa^[i];
      vkey := vect[i];
      j := i;
      while (j > 1) and (vect[j-1] < vkey) do
        begin
          pa^[j] := pa^[j-1];
          vect[j] := vect[j-1];
          j := j - 1;
        end;
      pa^[j] := key;
      vect[j] := vkey;
    end;
end;
```

```
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;

var pa: t_p_max_matr;
    l,m,n,i: integer;
    vect: t_vect;

begin
writeln('Введите кол-во строк');
readln(m);

writeln('Введите кол-во элементов в строке');
readln(n);

create_matr(pa,m,n);

writeln('Ввод матрицы: ');
read_matr(pa,m,n);

if (is_can_be_sorted(pa,m)) then
  begin
    for i := 1 to m do
      vect[i] := pa^[i]^[1];

    sort(pa,m,vect);
    writeln('Вывод отсортированной матрицы: ');
    write_matr(pa,m,n);
  end
else
  writeln('Матрицу невозможно отсортировать');

del_matr(pa,m,n);
```

```
Введите кол-во строк
3
Введите кол-во элементов в строке
2
Ввод матрицы:
4 5
10 9
1 0
Вывод отсортированной матрицы:
10 9
4 5
1 0
```

```
Введите кол-во строк
5
Введите кол-во элементов в строке
3
Ввод матрицы:
4 5 6
13 20 3
100 2 3
4 6 7
9 9 9
Матрицу невозможно отсортировать
```