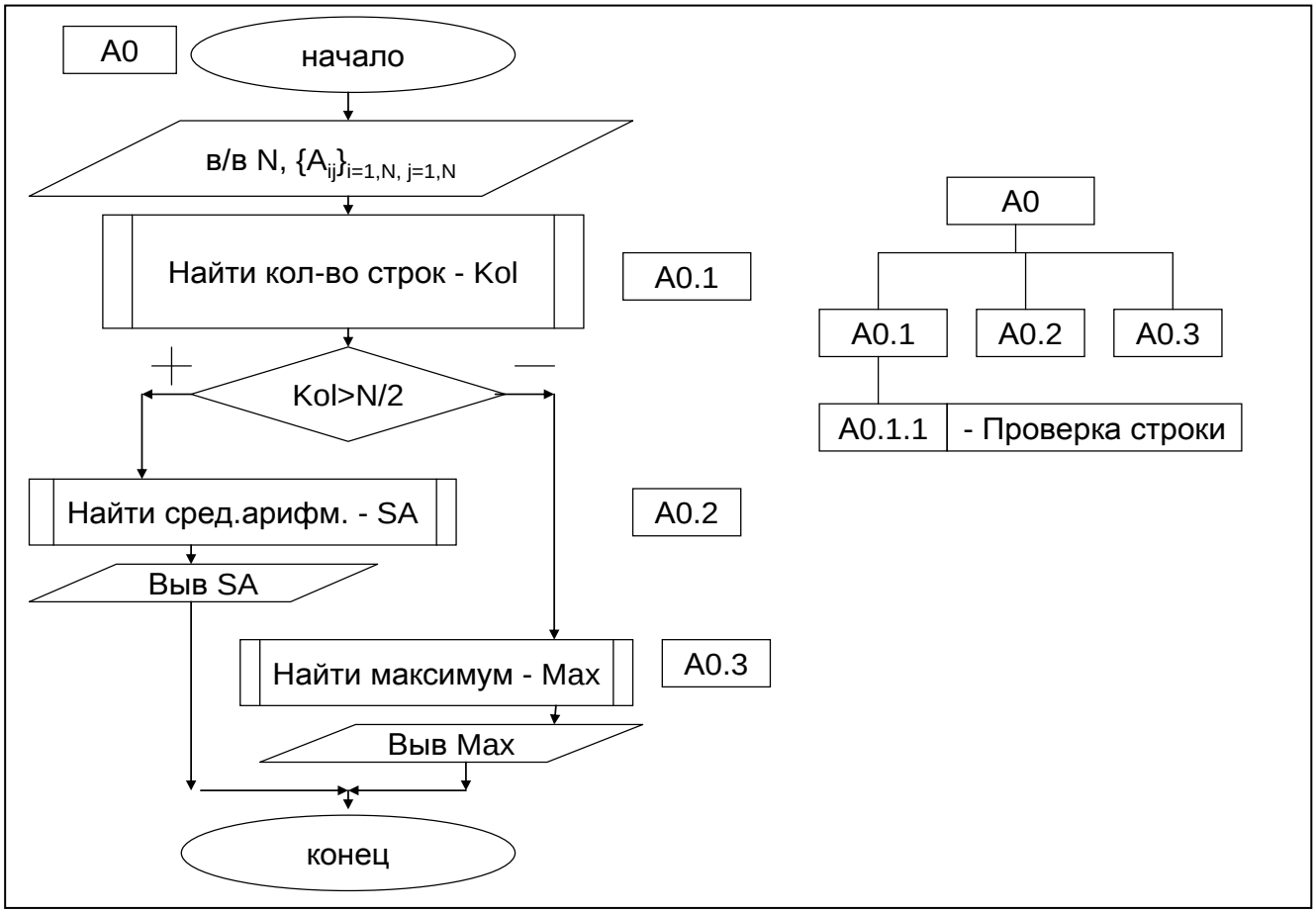
Лекция 13. Процедуры в языке Delphi (продолжение)

Рассмотрим пример разработки программы с процедурами общего вида.

Условие задачи:

Дана квадратная матрица *A* размером *N*x*N*.
Если количество (*Kol*) строк, в которых находятся только положительные элементы, больше *N*/2,
Найти *SA* – среднее арифметическое значений всех положительных элементов,
иначе
Найти *Max* – максимальное значение среди элементов, лежащих выше главной диагонали матрицы и на ней.

Выделим подзадачи:



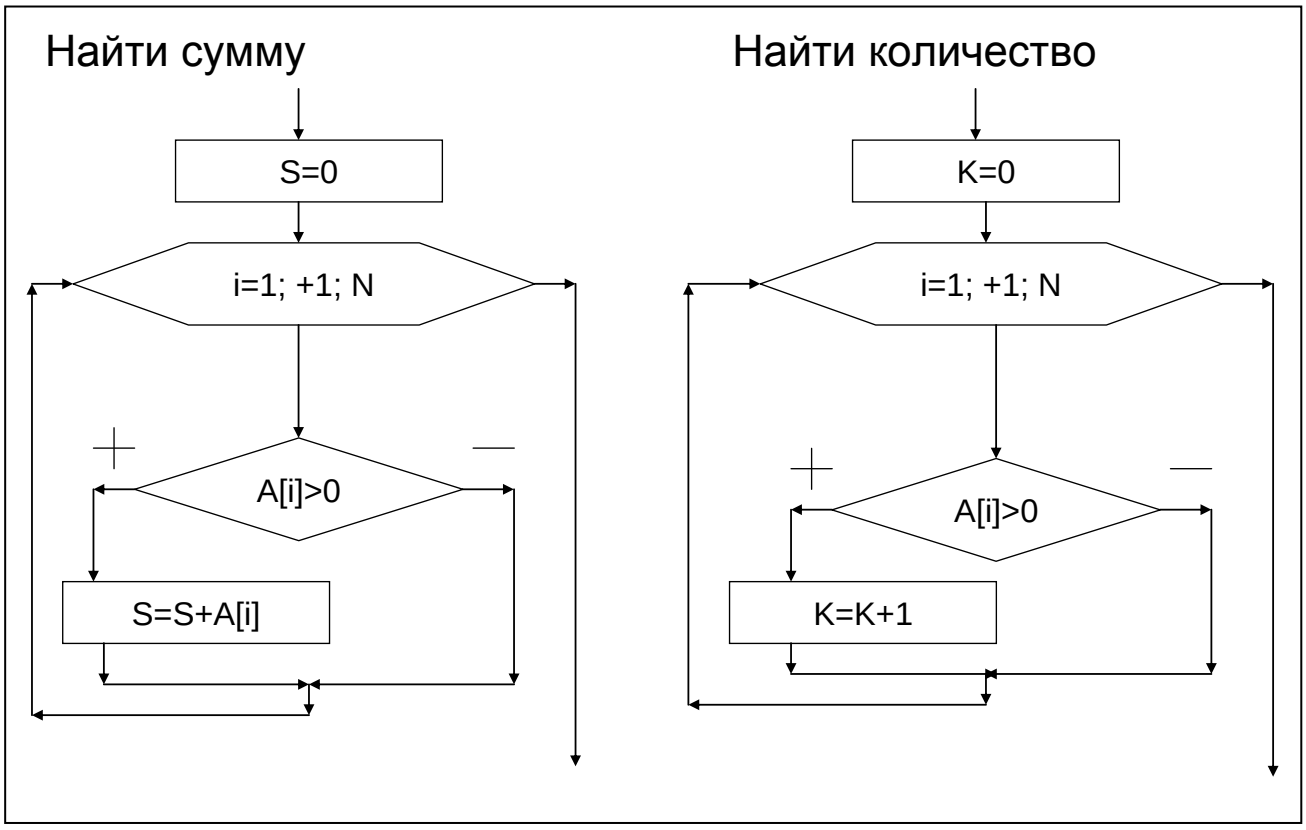
Случаев, когда не возможно найти среднее положительных элементов, или нет элементов среди которых надо найти максимум не предусматриваем, поскольку среднее ищется только при наличии положительных элементов в большинстве строк матрицы, а при поиске максимума возможно отсутствие только элементов выше главной диагонали, но ни как не на ней, даже при размере матрицы 1x1.

Рассмотрим сначала вторую подзадачу:

Дана квадратная матрица *A* размером *N*x*N*.
Если количество строк, в которых находятся только положительные элементы, больше *N*/2,
Найти SA – среднее арифметическое значений всех положительных элементов,
иначе –
Найти максимальное значение среди элементов, лежащих выше главной диагонали матрицы и на ней.

Чтобы найти среднее арифметическое, надо сначала найти сумму и количество. Зная алгоритмы поиска суммы и количества с условием для одномерного массива (*Базовые-алгоритмы.pdf*), создадим аналогичные алгоритмы для матрицы:

Для одномерного массива *A* из *N* элементов алгоритмы поиска суммы и количества положительных элементов выглядят так:

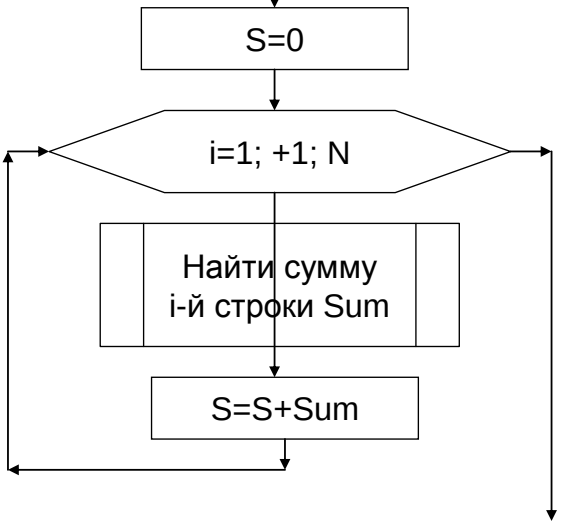


Для двухмерного массива *A* из *N* строк и *N* столбцов сумму можно найти как сумму (*S*) *N* сумм (*Sum*) положительных элементов строк из *N* элементов:

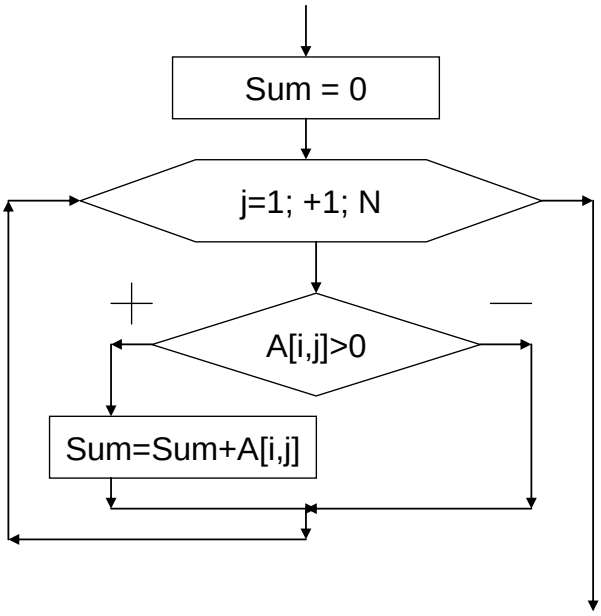
$$S = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\Big|_{Aij>0} = \sum_{i=1}^{N} Sum_i$$

$$Sum_i = \sum_{j=1}^{N} A_{i,j}\Big|_{Aij>0}$$

На примере суммы:



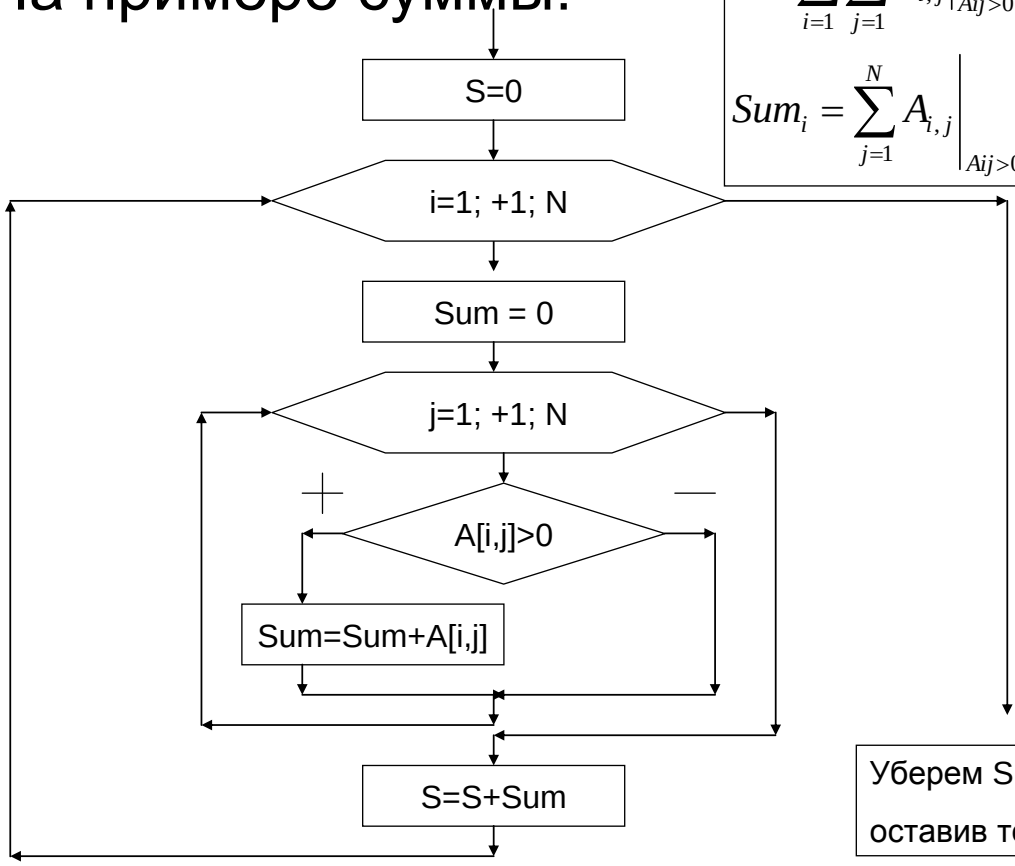
Сумма строки i



На примере суммы:

$$S = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\Big|_{Aij>0} = \sum_{i=1}^{N} Sum_i$$

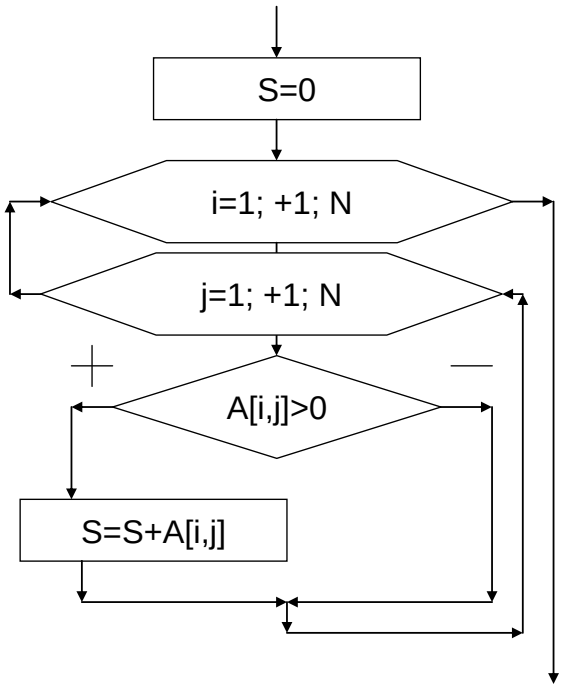
$$Sum_i = \sum_{j=1}^{N} A_{i,j}\Big|_{Aij>0}$$



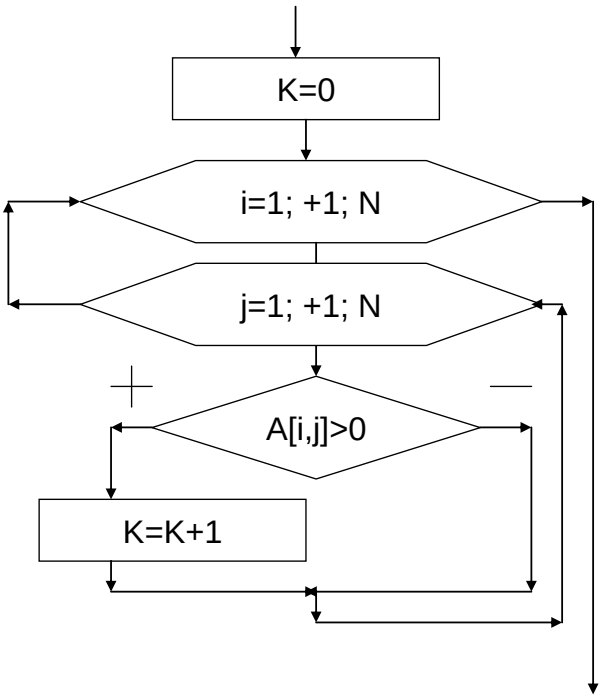
Уберем Sum, оставив только S

A0.2 - Найти сред арифм Aij>0

Найти сумму - S

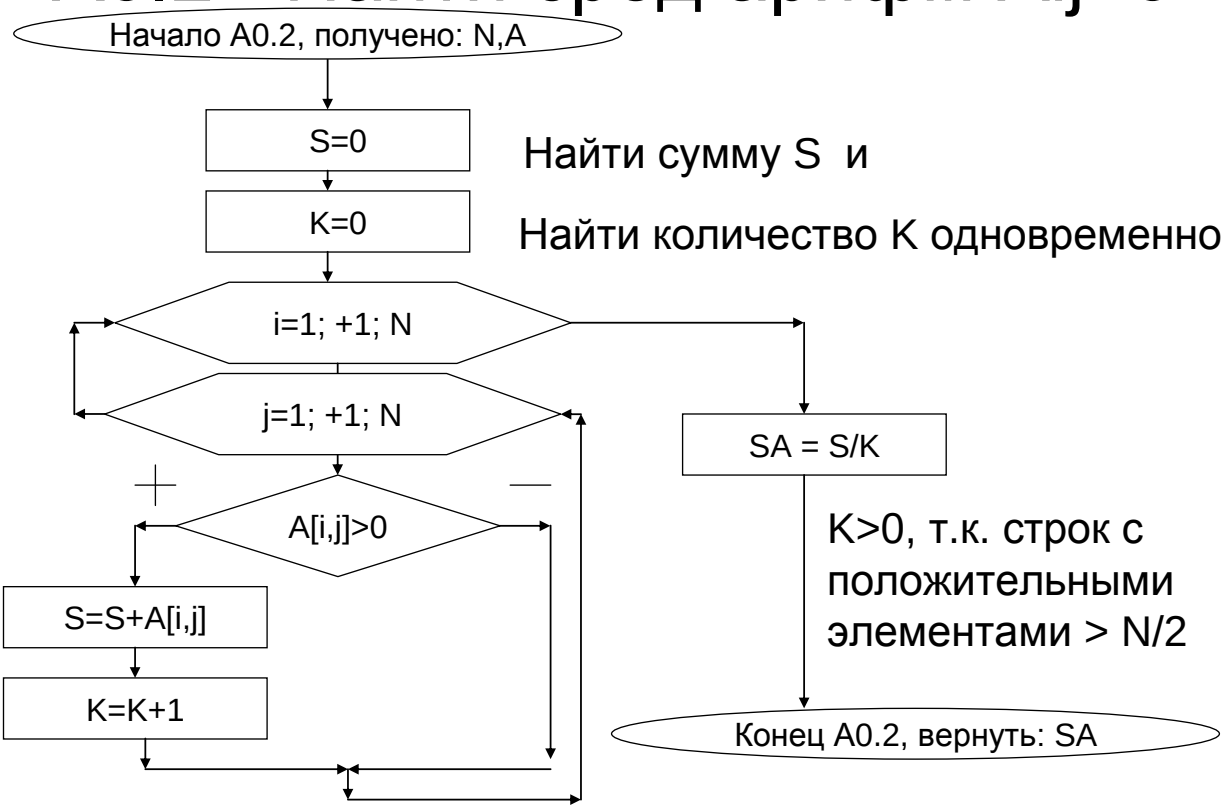


Найти количество - K



Т.о. для матрицы – удваиваем циклы

A0.2 - Найти сред арифм Aij>0



Найти сумму S и

Найти количество K одновременно

K>0, т.к. строк с положительными элементами > N/2

Теперь надо закодировать полученный алгоритм:

A0.2 - Найти сред.арифм Aij>0

```
Procedure FindSA( _______параметры_____________);
Var   _________локальные переменные_________

Begin
  S:=0;   K:=0;
  for i:=1 to N do
    for j:=1 to N do
      if A[i,j]>0 then
      begin
        S:=S+A[i,j]; K:=K+1;
      end;
  SA:=S/K;
End;
```

	вх	пром	вых	
S	-	+	-	лок
K	-	+	-	лок
i	-	+	-	лок
N	+	-	-	const
j	-	+	-	лок
A	+	-	-	const
SA	-	+	+	out

Выпишем в столбик **все переменные**, упоминающиеся в коде, и напротив каждой отмечаем, что она:

- **вх**одная, если ее значение должно быть известно до начала поиска среднего арифметического;
- **пром**ежуточная, если ей присваивается значение в этом коде;
- **вых**одная, если ее значение было изменено/найдено здесь и должно быть передано вызывающей программе.

Теперь

1. переменные, используемые только для промежуточных вычислений, описываем как ***локальные*** перед блоком оператором процедуры в разделе описания переменных *var*;
2. переменные, значения которых являются только входными описываем как ***параметры-константы*** (***const***) или ***параметры-значения***;
3. переменные, значения которых являются входными и промежуточными описываем как ***параметры-значения***;
4. переменные, значения которых являются входными и промежуточными и выходными описываем как ***параметры-переменные*** **(*var*)**;
5. переменные, значения которых не являются входными, но являются промежуточными и выходными описываем как ***параметры-переменные*** **(*out*** или ***var*)**.

Кратко это можно описать в виде следующей таблицы:

вх	**пром**	**вых**	**Описывается как**
+	–	–	параметр-константа (с ключевым словом *const*) или как параметр-значение или как параметр-переменная (с ключевым словом *var*)
+	+	–	параметр-значение (по умолчанию, без ключ.слова)
+	+	+	параметр-переменная (с ключевым словом *var*)
–	+	+	параметр-переменная (с ключевым словом *out*) или параметр-переменная (с ключевым словом *var*)
–	+	–	локальная переменная, не является параметром

Таким образом, **параметрами** становятся только входные и/или выходные переменные, остальные надо описывать как **локальные переменные**. **Все переменные**, упоминающиеся в коде процедуры, должны быть описаны либо как локальные переменные, либо как ее параметры. **Следуя этим правилам, вы создадите процедуры достаточно автономные и легко переносимые в ближайшем будущем в отдельные файлы-модули.**

Для данной процедуры потребовалось создать новый тип для пользовательского типа массив:

```
Const Nmax = 10;
Type  Matr = array [1..Nmax, 1..Nmax] of real;

Procedure FindSA(const N: byte; const A: Matr; out SA: real);
Var   i,j,K: byte; S: real;  //локальные
Begin
   S:=0;  K:=0;
   for i:=1 to N do
      for j:=1 to N do
         if A[i,j]>0 then
         begin
            S:=S+A[i,j]; K:=K+1;
         end;
   SA:=S/K;
End;
```

Заголовок можно сделать и таким:

```
Procedure FindSA( N: byte; var A: Matr; var SA: real);
```

При этом будет создана локальная копия *N* (1 байт), а остальные два параметра описаны как параметры-переменные и будут переданы по ссылке, без создания их локальной копии, что в случае с *A* позволит избежать создания ее копии (800 байт), а в случаи с *SA* позволит «передать результат через параметр», изменяя напрямую фактический параметр.

А такой, например, заголовок будет неправильным:

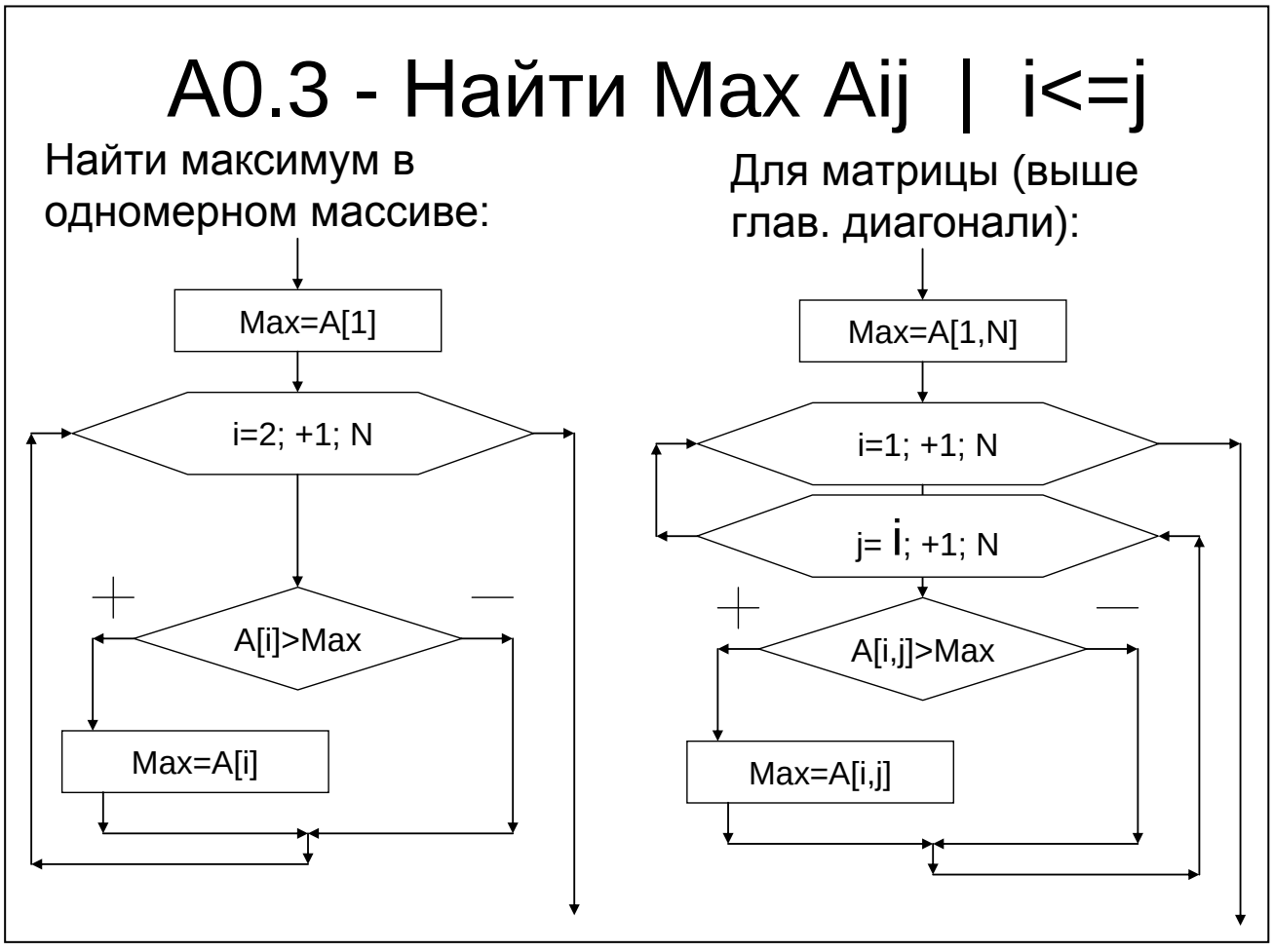
```
Procedure FindSA( N: byte; A: Matr; SA: real);
```

Поскольку, мало того, что тратится лишняя память (10x10x8 байт = 800 байт) на создание ненужной в данном случае копии массива *A*, так еще и результат вычисления среднего арифметического не сохранится в памяти после завершения работы процедуры, так как последний параметр *SA* указан как параметр-значение, а не как параметр-переменная, и результат будет сохранен только лишь в локальной копии *SA*, которая будет уничтожена при выходе из процедуры.

Рассмотрим следующую подзадачу:

Дана квадратная матрица *A* размером *N*x*N*.
Если количество (*Kol*) строк, в которых находятся только положительные элементы, больше *N*/2,
Найти SA – среднее арифметическое значений всех положительных элементов,
иначе –
Найти MAX – максимальное значение среди элементов, лежащих выше главной диагонали матрицы и на ней.

По аналогии с поиском максимума в одномерном массиве выполняется и поиск в матрице с перебором всех элементов из области поиска. Начальное значение также выбирается среди значений элементов, лежащих выше главной диагонали или на ней.



Закодируем алгоритм и выпишем все используемые переменные:

A0.3 - Найти Max Aij | i<=j

```
Const Nmax = 10;
Type  Matr = array [1..Nmax, 1..Nmax] of real;

Procedure FindMax( _______параметры_____________);
Var   _________локальные переменные_________
Begin
   Max:=A[1,N];
   for i:=1 to N do
      for j:=i to N do
         if A[i,j]>Max then
            Max:=A[i,j];
End;
```

	вх	пром	вых	
Max	-	+	+	out
i	-	+	-	лок
N	+	-	-	const
j	-	+	-	лок
A	+	-	-	const/ var

Используя ранее описанный тип, опишем по тем же правилам заголовок и локальные переменные:

```
Procedure FindMax( N: byte; var A: Matr; out Max: real);
Var
    i,j: byte;
Begin
   Max:=A[1,N];
   for i:=1 to N do
      for j:=i to N do
         if A[i,j]>Max then
            Max:=A[i,j];
End;
```

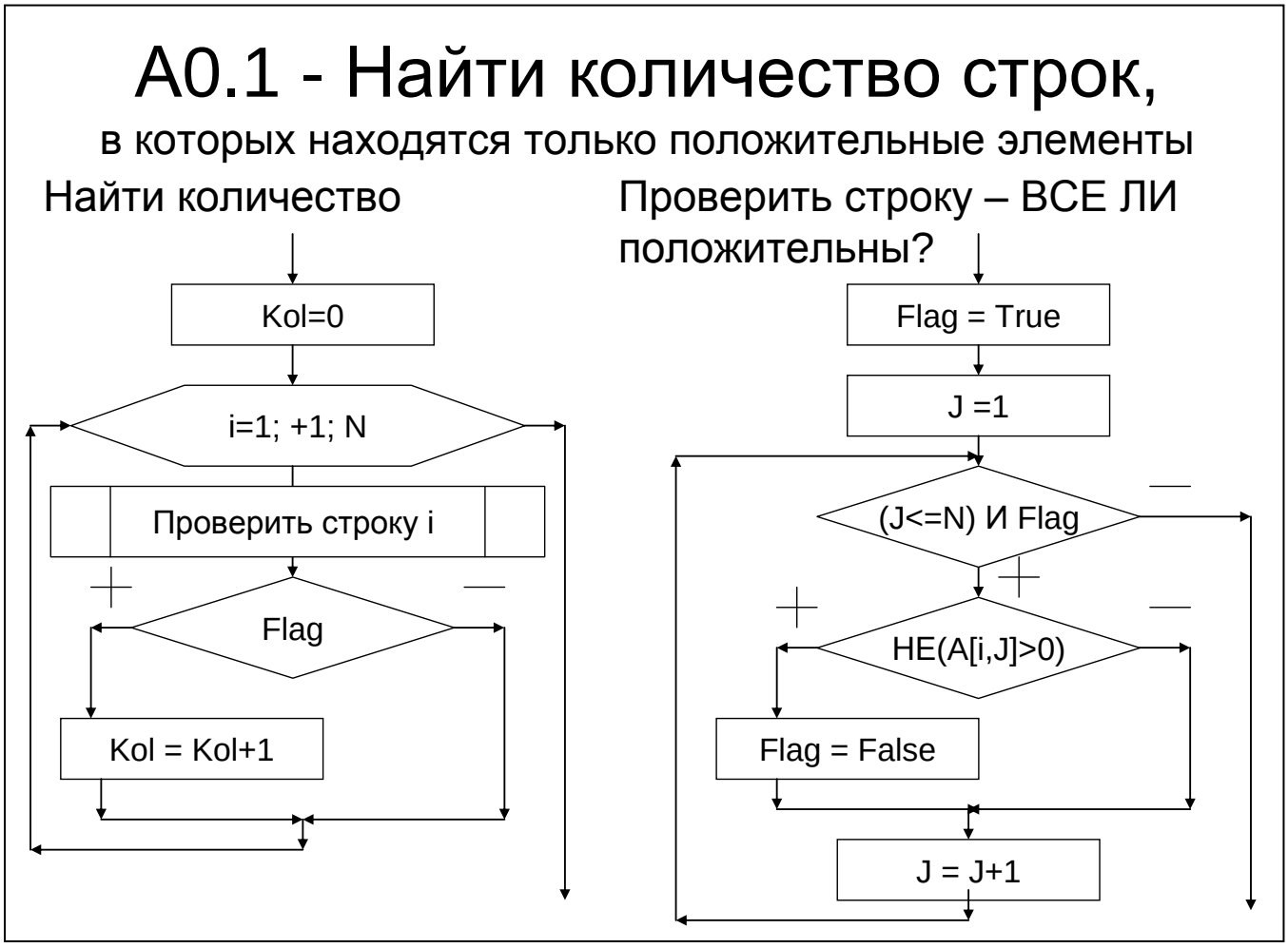
Заголовок также может быть, например, таким:

```
Procedure FindMax(const N: byte; const A: Matr; var Max: real);
```

Наконец, рассмотрим самую сложную подзадачу поиска *Kol*:

Дана квадратная матрица *A* размером *N*x*N*. Если **количество (*Kol*) строк, в которых находятся только положительные элементы**, больше *N*/2, Найти SA – среднее арифметическое значений всех положительных элементов, иначе – Найти MAX – максимальное значение среди элементов, лежащих выше главной диагонали матрицы и на ней.

Выделим из этой подзадачи одну вспомогательную задачу для проверки ***i***-ой строки, результатом которой будет значение True (Истина), если все элементы ***i***-ой строки положительны, и False (Ложь) в противном случае:



Используя ранее описанные константу и тип, закодируем алгоритмы:

A0.1.1 – Проверить строку i

```
Const Nmax = 10;
Type  Matr = array [1..Nmax, 1..Nmax] of real;
Procedure Check_i(     const N, i: byte; var A: Matr;
                       out Flag: boolean);
Var    j: byte;
Begin
   Flag:=TRUE;    j:=1;
   while (j<=N) AND Flag do
   begin
      if NOT (A[i,j]>0) then
         Flag:= FALSE;
      inc(j);
   end;
End;
```

	вх	пром	вых	
Flag	-	+	+	out
i	+	-	-	const
N	+	-	-	const
j	-	+	-	лок
A	+	-	-	const/ var

A0.1 – Найти количество строк

```
Procedure FindKol(     const N: byte; var A: Matr;
                       out Kol: byte);
Var    i: byte; Flag: boolean;
Begin
   Kol:=0;
   For i:=1 to N do
   begin
      Check_i( N,i, A, Flag);
      if Flag then
         inc(Kol);
   end;
End;
```

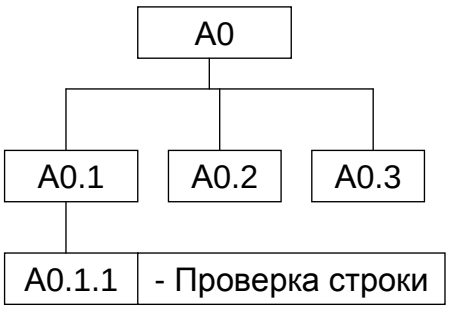
	вх	пром	вых	
Kol	-	+	+	out
Flag	-	+	-	лок
i	-	+	-	лок
N	+	-	-	const
A	+	-	-	const/ var

Соберем всю программу воедино, для краткости опуская разделы операторов в процедурах, добавим также не рассмотренную ранее процедуру *VvodVyvodNA* ввода и вывода исходных данных (*N* и *A*):

```
Program A0;
{$APPTYPE CONSOLE}

Const Nmax = 10;
Type  Matr = array [1..Nmax, 1..Nmax] of real;

Procedure Check_i( const N,i: byte; var A: Matr; out Flag: boolean);
Begin {..} End;
Procedure FindKol( const N: byte; var A: Matr; out Kol: byte);
Begin {..} End;

Procedure FindMax( N: byte; var A: Matr; out Max: real);
Begin {..} End;

Procedure FindSA( N: byte; var A: Matr; out SA: real);
Begin {..} End;

Procedure VvodVyvodNA( out N: byte; out A: Matr);
Begin {..} End;

Var
  N: Byte;
  A: Matr;
  Kol: byte;
  SA, Max: real;

Begin
   VvodVyvodNA( N, A);
   FindKol( N, A, Kol);
   if Kol > N/2 then
    begin
       FindSA( N, A, SA);
       writeln('Kol> N/2 and Sa=', SA:7:3);
    end
   else
    begin
       FindMax( N, A, Max);
       writeln('Kol<=N/2 and Max=',Max:5:1);
    end;

   writeln('Press ENTER'); readln;
End.
```

Важно отметить также, что процедуры *Check_i* и *FindKol*, могут быть описаны только в этом порядке (сначала *Chek_i*, затем вызывающая ее *FindKol*), либо первая из них может быть вложена во вторую (локальное описание):

A0.1 и A0.1.1 – Вложенность




```
Program A0;
{$APPTYPE CONSOLE}

Const Nmax = 10;
Type  Matr = array [1..Nmax, 1..Nmax] of real;


Procedure FindKol( const N: byte; var A: Matr; out Kol: byte);

   Procedure Check_i( const N,i: byte; var A: Matr; out Flag: boolean);
   Var …
   Begin…End;
Var …
Begin… Check_i …End;
```