

## Практическое занятие 3

### Поиск элемента, удовлетворяющего условию.

**Задание:** обсудить 2) способы поиска элемента, (не) удовлетворяющего условию, для решения вопроса: «ВСЕ ЛИ» или «ХОТЯ БЫ ОДИН» элемент одномерного массива удовлетворяет условию. 3) порядок решения задачи поиска экстремума с двумя условиями из лабораторных работ №6 и №10.

#### 3.1 Поиск элемента, удовлетворяющего заданному условию, и его номера

**Пример задачи:** Найти значение и номер первого отрицательного элемента массива.

**Замечание.** При поиске элемента, удовлетворяющего заданному условию, следует иметь в виду, что такого элемента может не оказаться, поэтому в уточненной формулировке задачи появляется проверка наличия такого элемента:

**Уточненная ПЗ:** Дан одномерный массив  $A$  из  $n$  элементов, найти значение и номер первого отрицательного элемента массива. Если не найдется ни одного отрицательного элемента, вывести сообщение «В массиве нет отрицательных элементов».

#### *Возможные формулировки задачи поиска элемента, удовлетворяющего условию*

**Задача 1.** Проверить, *есть ли* среди заданной совокупности элементов элемент (*хотя бы один*), удовлетворяющий условию  $C1$ ; если есть, то найти его значение и номер первого такого элемента.

**Вх. данные:** совокупность элементов

**Вых. данные:**

– результат проверки;	}	только в случае положительного результата проверки
– значение элемента;		
– его номер		

**Задача 2.** Проверить, *все ли* элементы заданной совокупности элементов удовлетворяют условию  $C2$ ; если нет, то найти значение и номер первого элемента, не удовлетворяющего условию.

**Вх. данные:** совокупность элементов

**Вых. данные:**

– результат проверки;	}	только в случае отрицательного результата проверки
– значение элемента;		
– его номер		

По существу, это две **взаимобратимые** формулировки одной и той же задачи, где  $C2 = \neg C1$  ( $C2$  – отрицание  $C1$ ).

Например, 1) проверить, **есть ли** в заданной совокупности элементов **хотя бы один** отрицательный элемент (т.е.  $C1$  – «элемент отрицателен»), если есть, то найти его значение и номер;

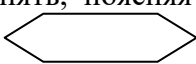
то же, что и: 2) проверить, **все ли** элементы заданной совокупности элементов **неотрицательны** (т.е.  $C2 = \neg C1$  – «элемент неотрицателен»), если нет, то найти значение и номер первого отрицательного.

Однако на практике целесообразно решать задачу так, как она поставлена, т.к. при изменении формулировки можно внести ошибку: например, если вместо слова «**неотрицательны**» написать «**положительны**», то нулевые значения элементов будут исключены из условия  $C2$ .

♦ В качестве заданной совокупности элементов будем рассматривать одномерный массив  $a(n)$ .

♦ Ответы «есть»/«нет» и «все»/«не все» при алгоритмизации наиболее точно отображаются логической переменной. Таким образом, *суть решения* сводится к поиску значения этой *логической переменной* как результата проверки. Но при *выводе* ответа в выходной форме более естественно использовать обычный язык.

Используем уже привычную простейшую схему нисходящего проектирования (выделение в первую очередь подзадач ввода-вывода входных данных; обработки; вывода результатов). Тогда формирование выходных данных, в том числе упомянутого логического результата проверки, будет отнесено к подзадаче обработки (собственно поиска), а вывод, в том числе текста ответа на вопрос задачи в зависимости от результата проверки, реализуется в подзадаче вывода.

**Рекомендации к объяснению:** Блок-схему уместить на доску и постепенно изменять, поясняя (стр.2-7). Оставить до и в конце тела цикла ДЛЯ место для преобразования одного блока  в три. Учитывайте, что студентам доступна полная версия в этом файле, и, возможно, распечатка есть «на руках».

♦ **Идея метода проверки наличия элемента, удовлетворяющего условию.**

Решим сначала более простую задачу.

**Задача 0.** Задан одномерный целочисленный массив  $A$  из  $N$  элементов. Проверить, есть ли среди его элементов отрицательные. Вывести соответствующее сообщение «Есть» или «Нет».

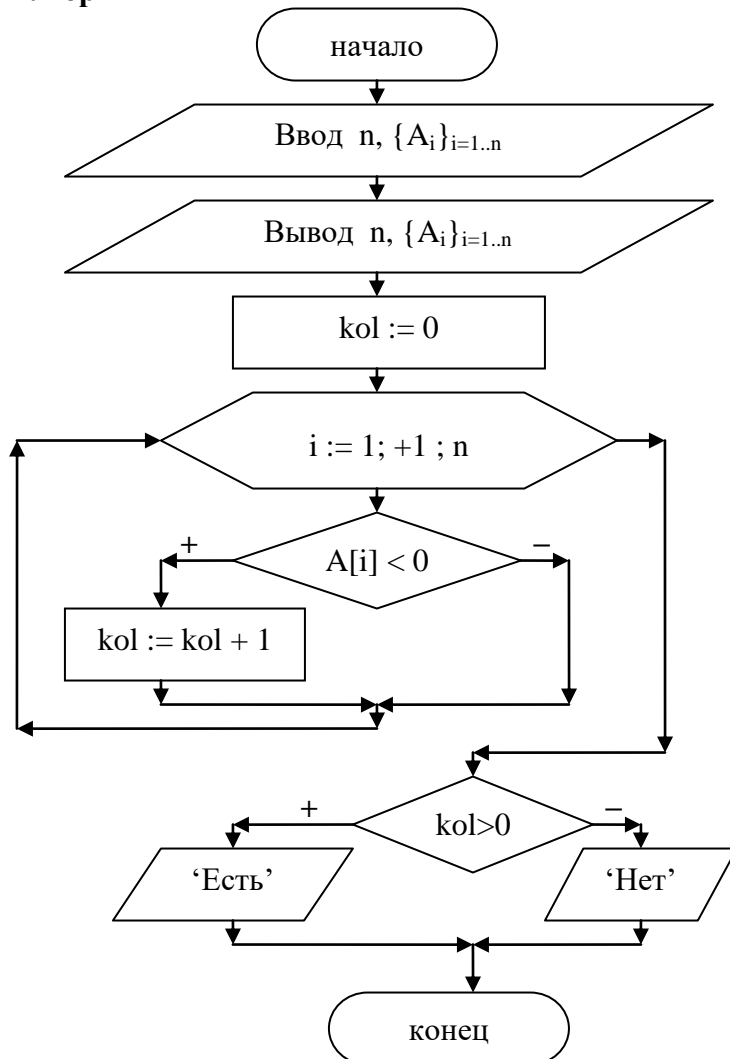
**Таблица данных**

Класс	Имя	Описание (смысл), диапазон, точность	Тип	Структура	Формат
Входные данные	$A$	заданный массив, $ A_i  < 1000$	цел	одномерный массив (20)	+XXX (:4)
	$n$	число элементов массива $a$ , $0 < n \leq 20$	цел	простая переменная	XX (:2)
Выходные данные	$y$	Результат проверки. $y = \text{True}$ (Истина), если <i>есть</i> отрицательные элементы, и $\text{False}$ (Ложь), если их <i>нет</i> .	лог	простая переменная	Выводится «Есть» или «Нет».
				Переменная будет введена при втором усовершенствовании алгоритма	
Промежуточные	$i$	индекс текущего элемента, $0 < i \leq 21$	цел	простая переменная	--
	$kol$	Количество отриц. элементов, $0 < kol \leq 20$	цел	простая переменная	--

**Метод** (не самый рациональный, но уже знакомый: через поиск количества)

Просматриваем все элементы массива, начиная с первого, и считаем количество отрицательных элементов. Если количество больше нуля, значит отрицательные элементы есть.

**Алгоритм**



**Программный код**

```

Program VseOtr;
{$APPTYPE CONSOLE}

```

```

const
  nmax = 20;
var
  A: array [1..20] of integer;
  n, i, kol: byte;

```

```

Begin
  AssignFile(input, 'VseOtr_dat.txt'); Reset(input);
  AssignFile(output, 'VseOtr_res.txt'); Rewrite(output);

```

```

  Readln(n);
  For i:=1 to n do read(A[i]); readln;
  Writeln('Задан массив из ', n:2, ' элементов:');
  For i:=1 to n do Write(A[i]:4); writeln;

```

```

  kol:= 0;
  For i:=1 to n do
    If A[i]<0 then kol:= kol+1;

```

```

  Writeln('Отрицательные элементы:');
  If kol>0 then writeln('Есть')
  else writeln('Нет');

```

```

  CloseFile(Input); CloseFile(Output);
End.

```

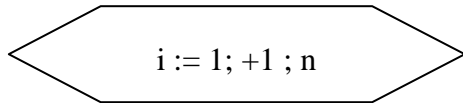
**Метод** (усовершенствование 1)

Зачем нам знать *точное* количество? Достаточно знать, что оно *отлично от нуля*, например 1, то есть «*есть хотя бы 1*». Изменим три блока

блок  $kol := kol + 1$  на  $kol := 1$

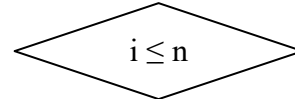
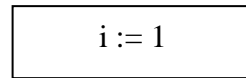
и блок  $kol > 0$  на  $kol \neq 0$

и вспомним, что означает блок

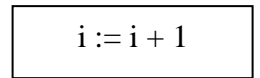
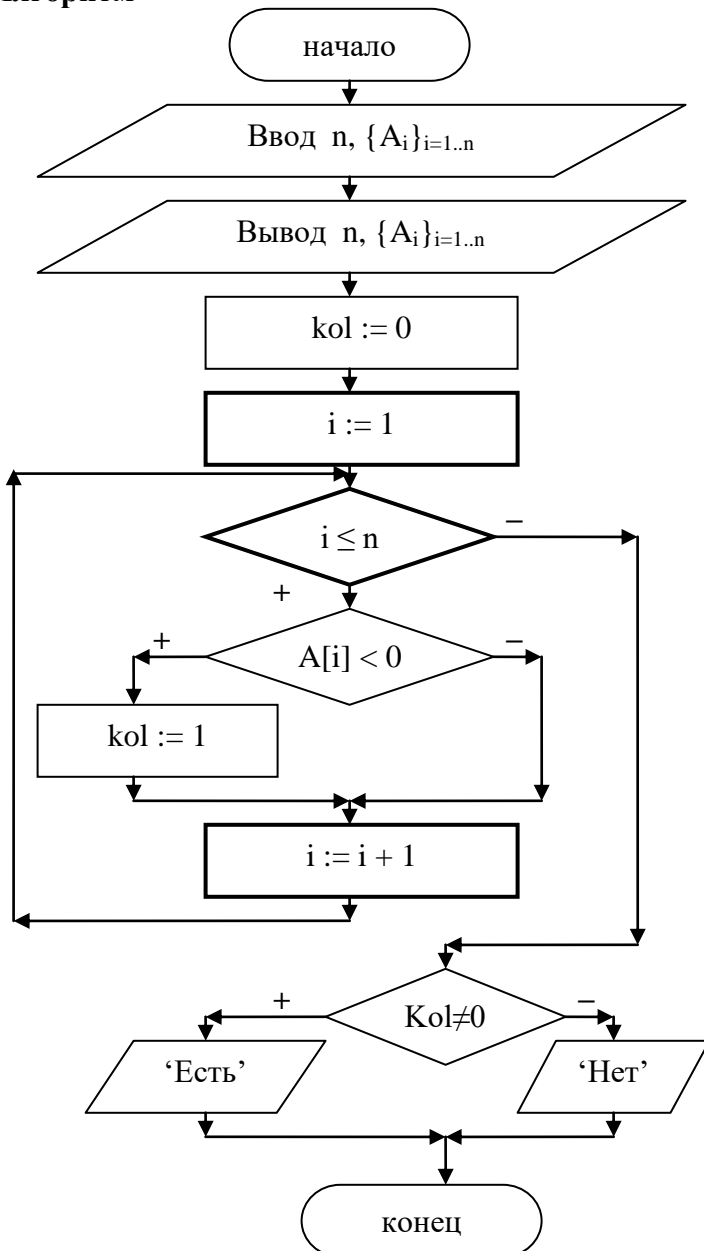


и заменим его на три базовых блока:

и закодирует его с помощью цикла ПОКА (*while*)



и

**Алгоритм****Программный код**

```
Program VseOtr;
{$APPTYPE CONSOLE}
```

```
const
  nmax = 20;
var
  A: array [1..20] of integer;
  n, i, kol: byte;
```

```
Begin
```

```
  AssignFile(input, 'VseOtr_dat.txt'); Reset(input);
  AssignFile(output, 'VseOtr_res.txt'); Rewrite(output);
```

```
  Readln(n);
  For i:=1 to n do read(A[i]); readln;
  Writeln('Задан массив из ', n, ' элементов:');
  For i:=1 to n do Write(A[i]:4); writeln;
```

```
  kol:= 0;
  i:=1;
  while i<=n do
  begin
    If A[i]<0 then kol:= 1;
    Inc(i);
  end;
```

```
  Writeln('Отрицательные элементы:');
  If kol<>0 then writeln('Есть')
  else writeln('Нет');
```

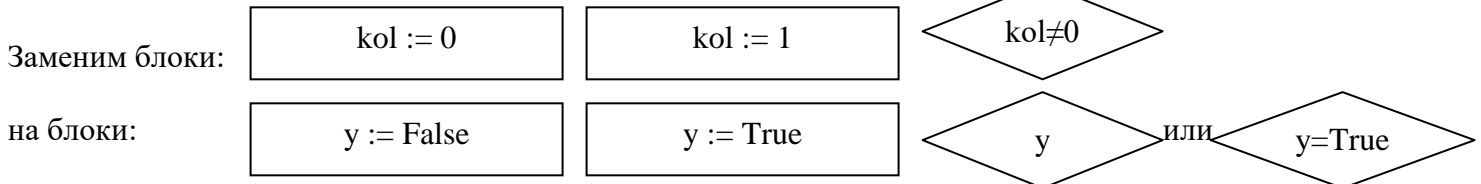
```
  CloseFile(Input); CloseFile(Output);
End.
```

### Метод (усовершенствование 2)

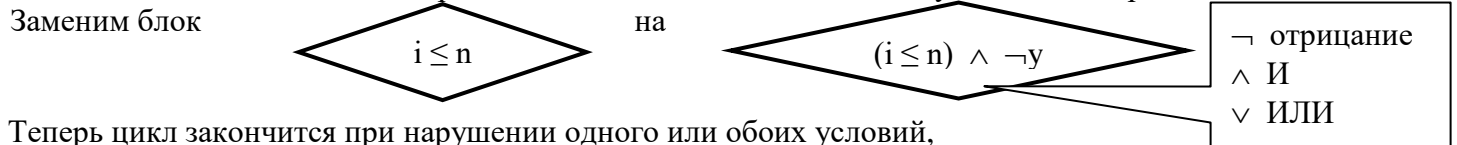
Заметим, что количество, это уже не количество, а флажок, имеющий два значения 0 и 1. Переменные, имеющие только два значения, обычно делают логическими: уберем переменную *kol* и введем простую логическую переменную *y* – результат проверки:

$y = \text{True}$  (Истина), если *есть* отрицательные элементы, и  $y = \text{False}$  (Ложь), если их *нет*.

Переменная является выходной.

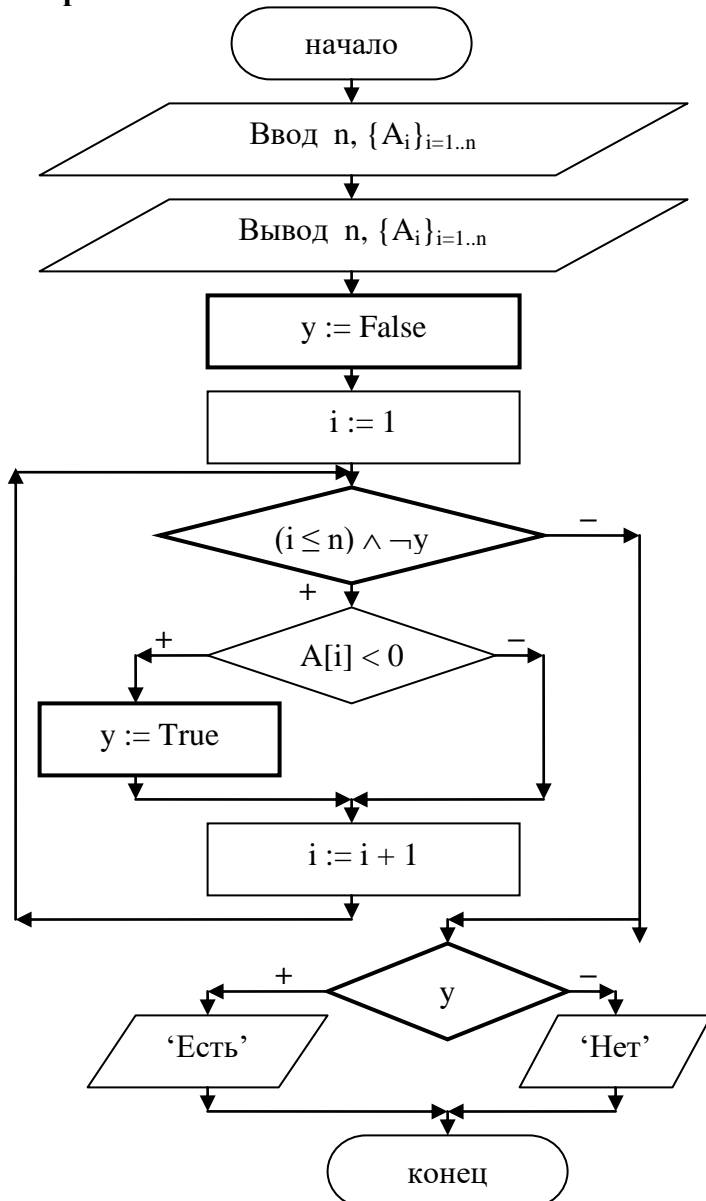


Далее, заметим, что после того, как мы найдем первый отрицательный элемент, можно дальше не искать, т.е. цикл должен выполняться не просто пока  $i \leq n$ , а еще и пока мы не узнали, что отриц.элемент есть.



Теперь цикл закончится при нарушении одного или обоих условий, проверяя не все  $n$  элементов, а столько, сколько необходимо для установления результата.

### Алгоритм



### Программный код

```

Program VseOtr;
{$APPTYPE CONSOLE}

```

```

const
  nmax = 20;
var
  A: array [1..20] of integer;
  n, i: byte;   y: boolean;

```

```

Begin
  AssignFile(input, 'VseOtr_dat.txt'); Reset(input);
  AssignFile(output, 'VseOtr_res.txt'); Rewrite(output);

```

```

  Readln(n);
  For i:=1 to n do read(A[i]); readln;
  Writeln('Задан массив из ', n, ' элементов:');
  For i:=1 to n do Write(A[i]:4); writeln;

```

```

  y:= False;
  i:=1;
  while (i<=n) and not y do
  begin
    If A[i]<0 then y:=True;
    Inc(i);
  end;

```

```

  Writeln('Отрицательные элементы:');
  If y then writeln('Есть')
  else writeln('Нет');

```

```

  CloseFile(Input); CloseFile(Output);
End.

```

## Вернемся к двум формулировкам задачи

### Задача 1.

**Условие.** Проверить, **есть ли** в одномерном целочисленном массиве  $A(n)$  элемент (**хотя бы один**), удовлетворяющий условию  $C1$ . Если есть, то найти *значение и номер* первого такого элемента.

**Таблица данных** как в задаче 0, но без  $kol$ , и добавлены три выход. переменные  $y1$  (вместо  $y$ ),  $k1$  и  $ak1$ :

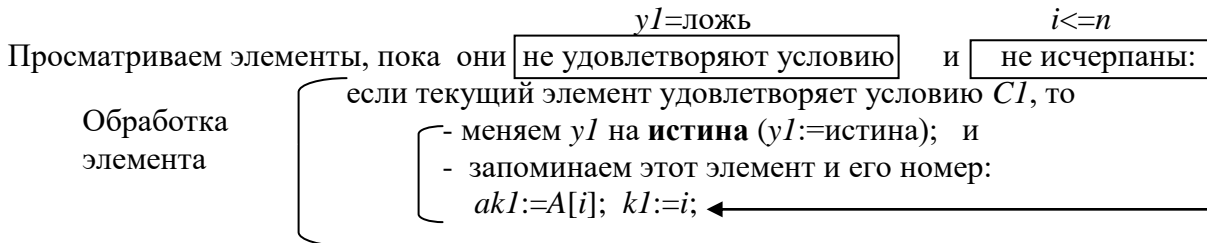
Класс	Имя	Описание (смысл), диапазон, точность	Тип	Структура	Формат
Входные данные	$A$		<тип>		
	$n$		цел		
Выходные данные	$y1$	Результат проверки. <i>True</i> (Истина), если <i>есть</i> элемент, удовлетворяющий условию $C1$ , и <i>False</i> (Ложь), если <i>нет</i> .	лог	простая переменная	«Есть» или «Нет»
	$k1$	номер искомого элемента	цел	прост.пер.	
	$ak1$	значение искомого элемента	<тип>	прост.пер.	
Промежуточные	$i$		цел		--

### **Метод**

(отличается от усовершенствованного метода к задаче 0 запоминанием номера и значения)

Используем  $y1$  следующим образом: пусть  $y1$  сохраняет значение "ложь", пока не найден нужный элемент, и меняет значение на "истина", как только элемент найден.

Тогда до начала поиска следует положить  $y1$ =**ложь** (элемент пока не найден).



конец просмотра;

### Задача 2)

**Условие.** Проверить, **все ли** элементы заданного одномерного массива  $A(n)$  удовлетворяют условию  $C2$ . Если нет, то найти *значение и номер* первого элемента, не удовлетворяющего условию.

**Таблица данных** как в задаче 0), но без  $kol$ , и добавлены три выход. переменные  $y2$  (вместо  $y$ ),  $k2$  и  $ak2$ .

Класс	Имя	Описание (смысл), диапазон, точность	Тип	Структура	Формат
Входные данные	$A$		<тип>		
	$n$		цел		
Выходные данные	$y2$	Результат проверки. <i>True</i> (Истина), если <b>все</b> элементы удовлетворяют условию, и <i>False</i> (Ложь), если хотя бы один элемент не удовл. условию $C2$ .	лог	простая переменная	
	$k2$	номер первого элемента, не удовлетворяющего условию	цел	прост.пер.	
	$ak2$	значение первого элемента, не удовлетворяющего условию	<тип>	прост.пер.	
Промежуточные	$i$		цел		--

**Метод** (аналогичен методу к задаче 1, но условие противоположно  $C2 = \neg C1$  и  $y2 = \neg y1$ )

Используем  $y2$  следующим образом: пусть  $y2$  сохраняет значение "истина", пока элементы удовлетворяют условию, и меняет значение на "ложь", как только условие нарушается (найдено противоречие).

Тогда до начала проверки следует положить  $y2 = \text{истина}$  (условие пока не нарушено).

Просматриваем элементы, пока они  $y2 = \text{истина}$  и  $i \leq n$   
удовлетворяют условию и не исчерпаны:

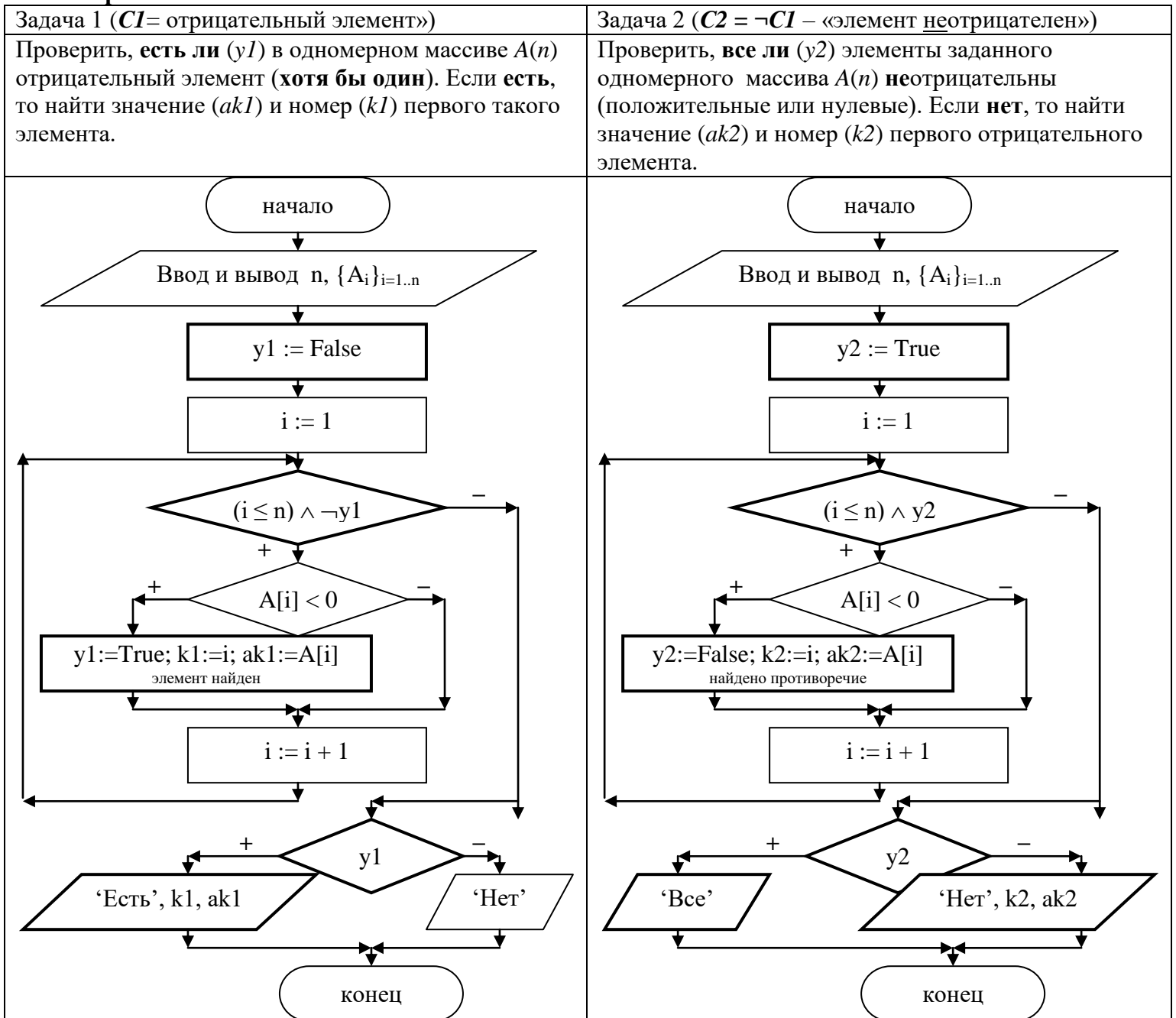
Обработка  
элемента

если текущий элемент не удовлетворяет условию  $C2$ , то

- меняем  $y2$  на **ложь** ( $y2 := \text{ложь}$ ); и
- запоминаем номер и значение:  
 $k2 := i$ ;  $ak2 := A[i]$ ;

конец просмотра;

### Алгоритмы



◆ **Указания и замечания:**

- Кодирование цикла ПОКА на *Delphi* – см. файл *Кодирование-алгоритмов.pdf*.
- В логических выражениях вместо  $y = \text{истина}$  (сравнение) записывается  $y$ , вместо  $y = \text{ложь}$  –  $\neg y$  (не  $y$ , в *Delphi* –  $\text{not } y$ )
- Логический тип возьмите *boolean*, константы *истина* и *ложь* закодируйте как *true* и *false*, отрицание( $\neg$ ) как *not*, И( $\wedge$ ) как *and*, ИЛИ( $\vee$ ) как *or*.

• Условие ( $C1, C2$ ) может быть наложено не только на один элемент, но и на пару рядом стоящих элементов. Например,

- 1) Проверить, что элементы массива упорядочены по убыванию, т.е.  
все  $(n-1)$  пары элементов упорядочены по убыванию:  $A_i > A_{i+1}, i=1..(n-1)$  (задача типа 2)
- 2) Проверить, что элементы массива упорядочены по возрастанию, т.е.  
все  $(n-1)$  пары элементов упорядочены по убыванию:  $A_i < A_{i+1}, i=1..(n-1)$  (задача типа 2)
- 3) Проверить, что элементы массива образуют арифметическую прогрессию, т.е.  
для каждой пары элементов должно выполняется условие:  $A_{i+1} = A_i + x, i=1..(n-1)$  (задача типа 2)  
где  $x = A_2 - A_1$
- 4) Проверить, что элементы массива образуют геометрическую прогрессию, т.е.  
для каждой пары элементов должно выполняется условие:  $A_{i+1} = A_i * x, i=1..(n-1)$  (задача типа 2)  
где  $x = A_2 / A_1$

Для сравнения: в задаче 2 для каждого элемента должно выполняется условие:  $A_i \geq 0, i=1..n$

### 3.2 Задача Cond2: "Поиск экстремума с двумя условиями"

#### 1. Общая схема решения задачи Cond2.

##### Условие.

Найти номер элемента с максимальным значением из всех элементов массива  $A(n)$ , удовлетворяющих условию  $c1$  и расположенных до последнего элемента, удовлетворяющего условию  $c2$ .




##### Уточненная ПЗ:

Найти номер ( $k_{\max}$ ) первого элемента с максимальным значением из всех элементов одномерного целочисленного массива  $A(n)$ , удовлетворяющих условию  $c1$  и расположенных до (включительно) последнего элемента, удовлетворяющего условию  $c2$ .

Если в массиве  $A$  не найдется ни одного элемента, удовлетворяющего условию  $c2$ , то сообщить об этом и выполнить поиск до конца массива.

Если будет невозможен поиск максимума, то вывести поясняющие сообщения о причинах, например: «В массиве  $A$  нет ни одного элемента, удовлетворяющего условию  $c1$ », : «Все элементы, удовлетворяющие условию  $c1$ , расположены правее всех элементов, удовлетворяющих условию  $c2$ », «В массиве  $A$  нет ни одного элемента, удовлетворяющего условию  $c1$ , и ни одного элемента, удовлетворяющего условию  $c2$ ».

##### Иллюстрация:

пусть  элементы, удовлетворяющие условию  $c1$ ,  $k1$  – номер первого элемента удовл. условию  $c1$   
 элементы, удовлетворяющие условию  $c2$ ;  $k2$  – номер последнего элемента удовл. условию  $c2$   
 элементы, удовлетворяющие условиям  $c1$  и  $c2$ ;

возможные варианты расположения элементов:

- 1а)   $k1 \leq k2$ : есть область поиска, можно найти максимум
- 1б)   $k1 = k2$ : частный случай, когда область поиска – один элемент, и сразу известен ответ
- 2)   $k1 > k2$ : все элементы, удовл.  $c1$ , лежат правее всех элементов, удовл.  $c2$ ; искать нигде
- 3)  нет элемента, удовл.  $c2$ ; договоримся искать до конца, т.е. сведем к случаю  $k2 = n$
- 4)  нет элемента, удовл.  $c1$ ; искать нигде
- 5)  нет ни элемента, удовл.  $c1$ , ни элемента, удовл.  $c2$ ; искать нигде

Схематическое описание ситуаций:

№ сит.	Элемент, удовл. $c1$	Элемент, удовл. $c2$	Что делаем
1	есть	есть, как надо ( $k1 \leq k2$ )	ищем максимум
2	есть	есть, но ( $k1 > k2$ )	искать нигде
3	есть	нет	уточняем, что делать; здесь – ищем до конца
4	нет	есть	искать нигде
5	нет	нет	искать нигде



#### 4. Таблица данных

Класс	Имя	Описание (смысл), диапазон, точность	Тип	Структура	Формат
Входные данные	$A$	Исходный массив, *	цел	одномерный массив (20)	
	$n$	Количество элементов в массиве $A$ , $0 < n \leq 20$	цел	простая переменная	
Выходные данные	$k_{max}$	искомый номер $0 < k_{max} \leq 20$	цел	простая переменная	
Промежу- точные	$k1$	номер первого элемента, удовл. $c1$	цел		--
	$k2$	номер последнего элемента, удовл. $c2$	цел		--
	$p1$	{ истина, если есть элемент, удовл. $c1$ ложь, в противном случае	лог		--
	$p2$	{ истина, если есть элемент, удовл. $c2$ ложь, в противном случае	лог		--
					--
					--

\*Доделать таблицу самим: диапазоны, форматы, дополнительные переменные

#### 5. Форма ввода

Данные вводить из текстового файла. *Cond2\_dat<№>.txt* (№ – номер теста)

Форма ввода элементарна, поэтому образцы можно не отмечать.  
Программировать ввод-вывод в точном соответствии с входной и выходной формами!

```
<n>
<A[1]>
<A[2]>
. . . .
<A[n]>
```

#### 6. Форма вывода (Данные вводить в текстовый файл. *Cond2\_res<№>.txt* (№ – номер теста))

обр1	Поиск по двум условиям	
обр2	Исходный массив $A$ из $\langle n \rangle$ элементов:	
	$\langle A[1] \rangle$	
Обр3	. . . .	
	$\langle A[n] \rangle$	
Обр4	Номер первого элемента с максимальным значением из элементов, удовлетворяющих условию $c1$ , равен $\langle k_{max} \rangle$	снт. 1
Обр5	Все элементы, удовл. $c1$ , правее всех элементов, удовл. $c2$ . Искать нигде	снт. 2
Обр6	Элемента, удовл. $c2$ , нет. При поиске до конца искомый номер равен $\langle k_{max} \rangle$	снт. 3
Обр7	Элемента, удовл. $c1$ , нет, хотя есть элемент, удовл. $c2$ Искать нигде	снт. 4
Обр8	Нет ни элемента, удовл. $c1$ , ни элемента, удовл. $c2$ . Искать нигде	снт. 5

## 6. Аномалии

не рассматриваем

## 7. Функциональные тесты.

Предусмотреть тесты для каждой из возможных пяти ситуаций расположения элементов.

Желателен также тест при  $k1=k2$  (ситуация 1б), если такое возможно.

*И не забудьте про диапазоны исходных данных и результатов!*

## 8. Метод

Соотношениями значений переменных  $k1, p1, k2, p2$  описываются все перечисленные ситуации (см таблицу ситуаций). Для начала надо найти их значения:

Подзадача А0.1. Проверить, есть ли ( $p1$ ) в массиве  $A(n)$  хотя бы один элемент, удовлетворяющий условию  $c1$ , и если есть, найти номер  $k1$  первого из них.

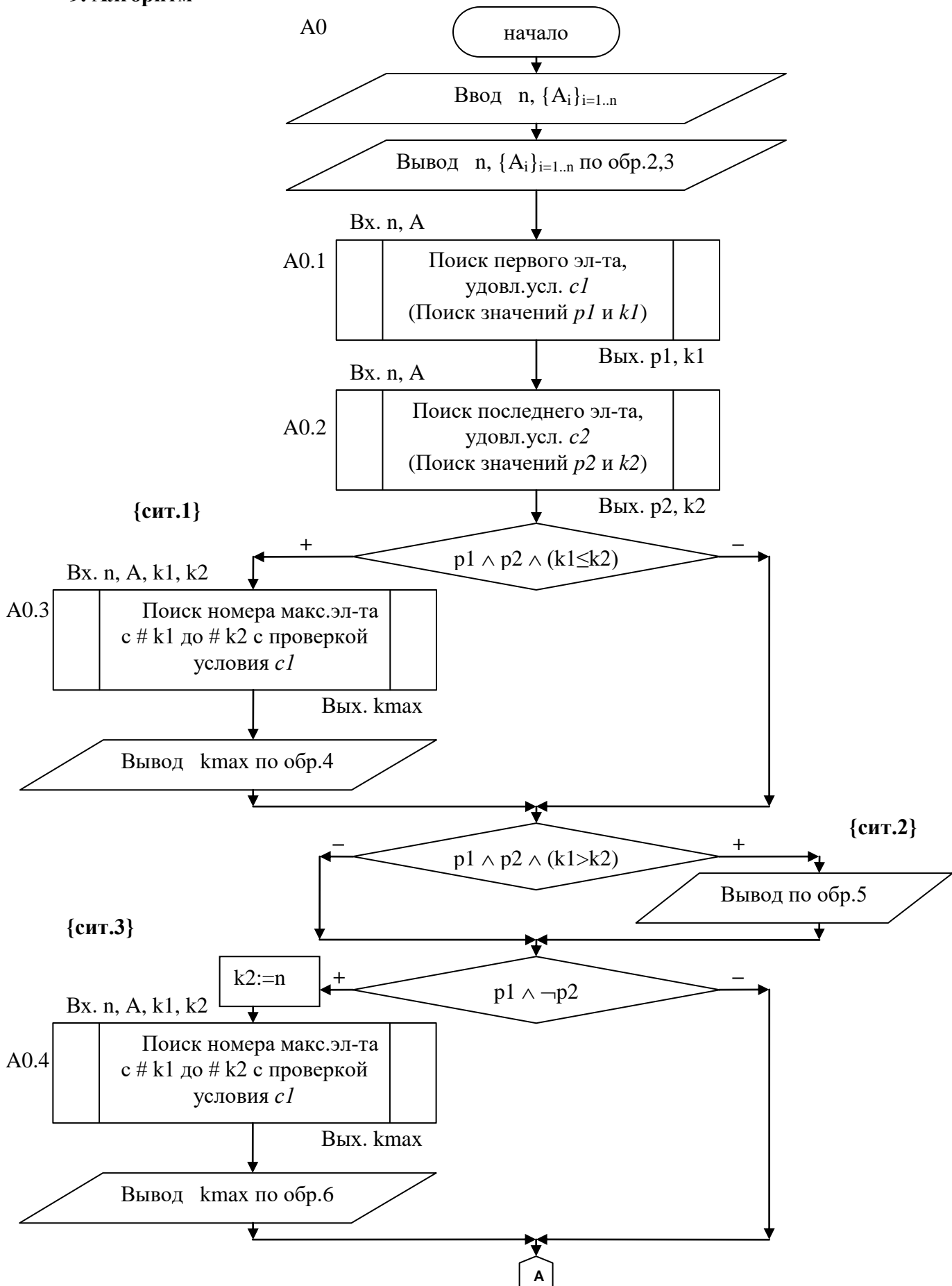
Подзадача А0.2. Проверить, есть ли ( $p2$ ) в массиве  $A(n)$  хотя бы один элемент, удовлетворяющий условию  $c2$ , и если есть, найти номер  $k2$  последнего из них.

Это задачи типа 1 из первой части файла.

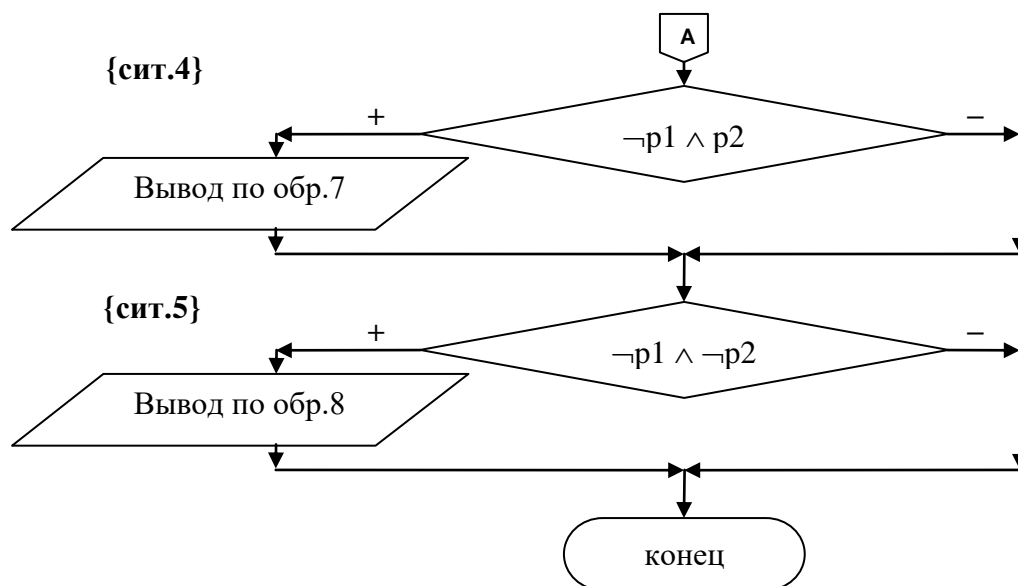
Затем надо выяснить, на какую ситуацию указывают значения переменных  $k1, p1, k2, p2$

№ сит.	$p1$	$p2$	$k1$ и $k2$	Что делаем
1	истина	истина	$k1 \leq k2$	Подзадача А0.3. Найти номер первого максимального элемента: модификация задачи <i>Extremum</i> : - поиск не с 1, а с $k1$ , - не до конца, а до $k2$ ; - плюс проверка условия $c1$ - начальное значение $A[k1]$  Выводим результат по обр.4
2	истина	истина	$k1 > k2$	Искать негде: выводим сообщение по обр.5
3	истина	ложь		Уточняем в условии, что делать: здесь – ищем до конца:  Подзадача А0.4. Найти номер первого максимального элемента: модификация задачи <i>Extremum</i> : - поиск не с 1, а с $k1$ , - до $n$ ; - плюс проверка условия $c1$ - начальное значение $A[k1]$  Выводим результат по обр.6
4	ложь	истина		искать негде: выводим сообщение по обр.7
5	ложь	ложь		искать негде: выводим сообщение по обр.8

## 9. Алгоритм



## 9. Алгоритм (продолжение)



**Замечание.** Видно, что задачи A0.3 и A0.4 схожи по постановке и могут быть сведены одна к другой присваиванием ( $k2:=n$ ). Тем не менее, каждая из них должна быть представлена в программе, поэтому они имеют разные имена. Очевидно, для их решения будет использован один и тот же алгоритм, который будет записан в программном коде дважды. Далее такие «одинаковые» подзадачи мы будем оформлять в виде *процедур*, и тогда *описание процедуры* будет представлять собой обобщенный алгоритм решения, а эти две подзадачи будут решаться двумя вызовами одной процедуры, т.е. A0.3 и A0.4 будут соответствовать отдельным вызовам одной и той же процедуры.

Блок-схемы для подзадач A0.1, A0.2, A0.3 начертить на отдельных листах. Для задачи A0.4 отдельная блок-схема не нужна, если она совпадает с A0.3. Спецификацию сохранить до защиты этой же задачи с процедурами (лаб.10).