Содержание

Предисловие

На Едином государственном экзамене по информатике и ИКТ задания, связанные с программированием, занимают важное место. Так, в демонстрационном варианте экзамена 2018 года их 8 при общем числе заданий 27. При этом высока весомость заданий (максимальный балл за выполнение заданий части 2 равен 3–4). Это говорит о том, что от умения решать задачи по программированию в значительной степени зависит успешность сдачи ЕГЭ в целом.

В то же время, как показывает опыт, такие задачи часто вызывают у школьников заметные трудности, особенно задачи части 2 экзамена. В большой степени это связано с недостаточным числом часов, отводимых на изучение программирования в школе.

Данная книга должна восполнить этот недостаток – помочь учащимся подготовиться к экзамену самостоятельно. В ней системно, подробно и доступно описана методика выполнения заданий по программированию, встречающихся на ЕГЭ.

Сначала в главе 1 рассмотрены все частные, вспомогательные задачи, умение решать которые позволит успешно выполнить задания экзамена по программированию, после чего в главах 2–7 описана методика выполнения заданий из ЕГЭ. В приложениях приведены другие материалы, связанные с заданиями по программированию на ЕГЭ.

При разработке программ (частных и из заданий ЕГЭ) использован школьный алгоритмический язык. Русский синтаксис этого языка и большое число комментариев сделают программы максимально понятными и легко переносимыми на любой другой язык программирования. Это означает, что книга может быть использована читателями, владеющими любым языком программирования. В большинстве случаев после разбора методики решения и программы на школьном алгоритмическом языке приводятся также соответствующие программы на популярном среди школьников языке Паскаль. Предлагаются задания для самостоятельной работы.

Обратим внимание на широкое использование в книге задач от разработчиков контрольно-измерительных материалов для ЕГЭ [11–13] – существует большая вероятность того, что подобные задачи будут включены в экзамен в будущем.

Кроме учащихся, готовящихся к сдаче экзамена самостоятельно, книгу могут использовать учителя и преподаватели информатики, а также студенты и учащиеся, изучающие программирование вне связи с ЕГЭ.

Глава 1

Вспомогательные задачи

В данной главе рассмотрена методика решения задач, предусмотренных Кодификатором элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2018 году единого государственного экзамена по информатике и ИКТ [7], которые используются в демонстрационных вариантах ЕГЭ по информатике последних лет, а также в книгах от разработчиков ЕГЭ [11–13].

1.1. Обработка натурального числа

1.1.1. Выделение цифр

Задача формулируется так: «Дано натуральное число `n`. Вывести его цифры в “столбик”».

Решение

Когда количество цифр в заданном числе известно, можно выделить любую его цифру. Но в данном случае это количество неизвестно. Поэтому подумаем над вопросом – какую цифру целого числа (см. схему ниже) определить можно?

первая	*вторая*	…	*предпоследняя*	*последняя*

цифры числа

Ответ – последнюю (последняя цифра любого натурального числа равна остатку от деления этого числа на 10 – убедитесь в этом!):

```
посл = mod(n, 10) |посл - последняя цифра числа n
```

где `mod` – функция школьного алгоритмического языка, возвращающая остаток от деления своего первого аргумента на второй (в других языках программирования для этого используется не функция, а специальная операция).

А остальные цифры? Как, например, определить предпоследнюю цифру? Ее можно найти только так: получить число без последней цифры исходного числа (разделив исходное нацело на 10) и для него определить последнюю цифру.

Аналогично можно получить и предпредпоследнюю, и остальные цифры.

Следовательно, для решения задачи в программе надо многократно выполнить следующие действия:

1) определить последнюю цифру числа;
2) вывести ее на экран;
3) получить число без последней цифры.

Соответствующий фрагмент программы:

```
нц пока n > 0:
    посл := mod(n, 10)
    вывод нс, посл
    n := div(n, 10)
кц
```

где div – функция школьного алгоритмического языка, возвращающая целую часть частного от деления первого аргумента на второй (в других языках для расчета также используется не функция, а специальная операция).

Язык Паскаль

```
while n > 0 do
  begin:
    posl := n mod 10;
    writeln(posl);
    n := n div 10
  end;
```

1.1.2. Определение суммы цифр числа

Для решения задачи в программе надо многократно выполнить следующие действия:

1) определить последнюю цифру числа;
2) учесть ее в найденной ранее сумме;
3) получить число без последней цифры.

С использованием переменной сум, накапливающей в себе сумму уже учтенных цифр, программа решения задачи оформляется следующим образом:

```
вывод нс, "Введите натуральное число "
ввод n
сум := 0 |Начальное значение суммы цифр
нц пока n > 0
  посл := mod(n, 10)
  сум := сум + посл
  n := div(n, 10)
кц
вывод нс, "Сумма цифр этого числа равна ", сум
```

Язык Паскаль

```
write('Введите натуральное число ');
readln(n);
sum := 0;
```

```
while n > 0 do
  begin
    posl := n mod 10;
    sum := sum + posl;
    n := n div 10
  end;
writeln('Сумма цифр этого числа равна ', sum);
```

Обратим внимание на важное обстоятельство. Хотя в языке программирования Паскаль и в ряде других языков переменной по умолчанию присваивается начальное значение, равное нулю, в программах решения заданий ЕГЭ оператор

```
sum := 0;
```

является обязательным, и его отсутствие, как показывает опыт, рассматривается проверяющими экзаменационные работы как ошибка.

1.1.3. Определение произведения цифр числа

Задача решается аналогично предыдущей. Отличия:

1) рассчитывается не сумма, а произведение цифр;
2) начальное значение искомой величины `произв` должно быть принято равным 1.

Программа:

```
вывод нс, "Введите натуральное число "
ввод n
произв := 1 |Начальное значение
нц пока n > 0
  посл := mod(n, 10)
  произв := произв * посл
  n := div(n, 10)
кц
вывод нс, "Произведение цифр этого числа равно ", произв
```

Язык Паскаль

```
write('Введите натуральное число ');
readln(n);
proizv := 1;
while n > 0 do
  begin
    posl := n mod 10;
    proizv := proizv * posl;
    n := n div 10
  end;
writeln('Произведение цифр этого числа равно ', proizv);
```

1.1.4. Определение количества цифр числа

Здесь следует использовать переменную-«счетчик» количества уже «обработанных» цифр:

```
вывод нс, "Введите натуральное число "
ввод n
кол := 0 |Начальное значение количества цифр
нц пока n > 0
  посл := mod(n, 10)
  кол := кол + 1
  n := div(n, 10)
кц
вывод нс, "Количество цифр этого числа равно ", кол
```

Обратим внимание на то, что последнюю цифру `посл` можно не определять.

Язык Паскаль

```
write('Введите натуральное число ');
readln(n);
kol := 0;
while n > 0 do
  begin
    kol := kol + 1;
    n := n div 10
  end;
writeln('Количество цифр этого числа равно ', kol);
```

Здесь также присваивание начального значения, равного нулю, является обязательным (см. *п. 1.1.2*).

1.1.5. Определение максимальной цифры числа

Алгоритм решения этой задачи аналогичен алгоритму действий человека, который определяет максимальную цифру в некотором числе:

324086312

– сначала он запоминает первую цифру, а затем рассматривает вторую. Если она больше того числа, которое помнил, то запоминает вторую цифру и переходит к следующей, третьей цифре, в противном случае просто переходит к следующей цифре и делает то же самое. В нашей программе единственное отличие – в том, что «просматривать» (выделять и сравнивать) цифры мы будем, начиная с последней.

В приведенной далее программе искомое максимальное значение хранит переменная макс.

```
вывод нс, "Введите натуральное число "
ввод n
посл := mod(n, 10) |Выделяем последнюю цифру
макс := посл       |Принимаем ее в качестве максимальной
n := div(n, 10)    |Отбрасываем последнюю цифру
|Рассматриваем остальные цифры
нц пока n > 0
  посл := mod(n, 10) |Выделяем
  если посл > макс   |Сравниваем
   то                |и при необходимости
     макс := посл    |меняем значение макс
  все
  n := div(n, 10)
кц
вывод нс, "Максимальная цифра заданного числа равна ", макс
```

В данном случае (см. *п. 1.4.1*) отдельно последнюю цифру можно не выделять, так как можно в качестве начального значения переменной макс принять 0:

```
вывод нс, "Введите натуральное число "
ввод n
макс := 0
|Рассматриваем все цифры
нц пока n > 0
  посл := mod(n, 10)
  если посл > макс
    то
      макс := посл
  все
  n := div(n, 10)
кц
вывод нс, "Максимальная цифра заданного числа равна ", макс
```

Язык Паскаль

```
write('Введите натуральное число ');
readln(n);
max := 0;
while n > 0 do
  begin
    posl := n mod 10;
    if posl > max then
      max := posl;
    n := n div 10
  end;
writeln('Максимальная цифра заданного числа равна ', max);
```

1.1.6. Определение минимальной цифры числа

Задача решается аналогично предыдущей (начальное значение искомой переменной `мин` можно принять равным 9).

1.2. Операции с элементами массива, отобранными по некоторому условию[1]

Для каждой рассмотренной задачи в данном разделе приведен фрагмент программы ее решения на школьном алгоритмическом языке и на языке Паскаль. Использованы следующие основные величины:

- `a` – имя массива;
- `n` – общее количество элементов массива (условно принято, что нумерация элементов массива начинается с 1).

Смысл остальных величин можно легко определить по их именам.

Принято, что элементы массива имеют целые значения.

1.2.1. Изменение элементов массива с заданными свойствами (удовлетворяющих некоторому условию)

Общая формулировка задач рассматриваемого типа: «Все элементы массива с заданными свойствами заменить на ...», где вместо многоточия указывается конкретное значение, на которое следует заменить указанные элементы, или правило, по которому надо провести замену.

Примеры задач

1. Дан массив целых чисел. Все четные элементы заменить числом 666.
2. Дан массив чисел, среди которых есть отрицательные. Все отрицательные элементы уменьшить на 10.
3. Дан массив целых чисел. Все элементы с нечетным индексом, оканчивающиеся цифрой 5, увеличить на 1.

---

[1] Такая формулировка приведена в Кодификаторе элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2018 году единого государственного экзамена по информатике и ИКТ [7].

Решение

```
нц для i от 1 до n
  |Если элемент обладает заданными свойствами
  если <условие>
    то
     |Меняем его значение
     a[i] := …
  все
кц
```

где *<условие>* – заданное условие. Это условие может определяться значением элемента массива a[i] или/и его индексом i (см., например, выше пример 3 задачи).

Язык Паскаль

```
for i := 1 to n do
  {Если элемент обладает заданными свойствами}
  if <условие>
    then {Меняем его значение}
      a[i] := …;
```

1.2.2. Нахождение суммы элементов массива с заданными свойствами (удовлетворяющих некоторому условию)

Примеры задач рассматриваемого типа

1. В массиве записана стоимость 30 предметов. Определить общую стоимость тех предметов, которые стоят менее 200 руб.
2. В массиве записаны 20 целых чисел. Определить сумму тех из них, которые оканчиваются нулем.
3. В массиве записаны данные о количестве осадков, выпавших за каждый день января. Определить общее количество осадков, выпавших второго, четвертого, шестого и т. д. числа этого месяца. В программу должны вводиться данные за каждый день месяца.

Решение

Обсудим сначала более простую задачу: «Найти сумму всех элементов массива».

С использованием переменной сум, накапливающей в себе сумму значений уже рассмотренных элементов массива, фрагмент программы решения задачи оформляется следующим образом:

```
сум := 0[1]
нц для i от 1 до n
  сум := сум + a[i]
кц
```

Вернемся теперь к «основной» задаче. Отличие задач данного типа от только что рассмотренной – в том, что добавлять значение элемента массива к уже рассчитанной ранее сумме следует только тогда, когда элемент обладает заданными свойствами:

```
сум := 0
нц для i от 1 до n
  |Если элемент обладает заданными свойствами
  если <условие>
    то
     |Учитываем его значение в сумме
     сум := сум + a[i]
  все
кц
```

В данном случае *<условие>* также может определяться значением элемента массива `a[i]` или/и его индексом `i`.

Однако так можно оформить программу только в случае, когда известно, что в массиве имеется хотя бы один элемент с заданными свойствами. Если допускается, что таких элементов может не быть, то фрагмент, связанный с выводом ответа, должен иметь вид:

```
если сум = 0
  то
    вывод нс, "Чисел, удовлетворяющих условию, в массиве нет"
  иначе
    вывод нс, "Сумма чисел, удовлетворяющих условию, равна ", сум
все
```

Примечание. Случай, когда элементы с заданными свойствами в массиве имеются, но их сумма равна нулю, не учитывается.

Язык Паскаль

```
sum := 0;
for i := 1 to n do
  {Если элемент обладает заданными свойствами}
  if <условие>
```

[1] Еще раз напомним, что присваивание переменной начального значения, равного нулю, в программах решения заданий ЕГЭ является обязательным.

```
    then {Учитываем его значение в сумме}
      sum := sum + a[i];
if sum = 0
  then write('Чисел, удовлетворяющих условию, в массиве нет'
  else write('Сумма чисел, удовлетворяющих условию, равна ', sum);
```

1.2.3. Нахождение количества элементов массива с заданными свойствами

Примеры задач рассматриваемого типа

1. В массиве хранятся 40 целых чисел. Найти количество четных чисел.
2. В массиве записан рост 22 юношей. Определить, сколько из них имеют рост менее 165 см.
3. В массиве записаны оценки по информатике каждого из 25 учеников класса. Определить количество пятерок.

Особенность задач данного типа – в том, что в случае, когда элемент обладает заданными свойствами (удовлетворяет некоторому условию), искомое количество увеличивается на 1:

```
кол := 0
нц для i от 1 до n
  |Если элемент обладает заданными свойствами
  если <условие>
    то
      |Учитываем его в искомом количестве
      кол := кол + 1
  все
кц
вывод нс, "Количество чисел, удовлетворяющих условию, равно ", кол
```

В данном случае условие в команде **если** (в условном операторе) определяется значением элемента массива `a[i]` или одновременно значениями `a[i]` и `i`. Количество элементов, зависящих только от значения индекса `i`, может быть найдено без использования оператора цикла (убедитесь в этом!).

Язык Паскаль

```
kol := 0;
for i := 1 to n do
  {Если элемент обладает заданными свойствами}
  if <условие>
    then {Учитываем его в искомом количестве}
      kol := kol + 1; {или inc(kol)}
write('Количество чисел, удовлетворяющих условию, равно ', kol);
```

1.2.4. Нахождение среднего арифметического значения элементов массива с заданными свойствами

Для нахождения искомого значения необходимо определить сумму элементов массива с заданными свойствами и их количество. Такие две задачи мы уже решили ранее. Здесь их можно объединить в одном операторе цикла:

```
сум := 0
кол := 0
нц для i от 1 до n
  |Если элемент обладает заданными свойствами
  если <условие>
    то
      |Учитываем его значение в сумме
      сум := сум + a[i]
      |и учитываем этот элемент в количестве
      кол:= кол + 1
  все
кц
|Подсчет результата
сред_арифм := сум/кол
```

Обратите внимание на то, что многократно определять значение сред_арифм в «теле» условного оператора (команды **если**) необходимости нет. Это можно сделать один раз после окончания оператора цикла. Однако может оказаться, что чисел, удовлетворяющих заданному условию, в массиве не окажется. В этом случае при расчете будет иметь место деление на ноль, что недопустимо. Правильное оформление:

```
...
|Подсчет и вывод результата
если кол > 0
  то
   средн_ариф := сум/кол
   вывод нс, "Среднее арифметическое: ", сред_арифм
  иначе
   вывод нс, "Чисел, удовлетворяющих условию, в массиве нет"
все
```

Язык Паскаль

```
sum := 0;
kol := 0;
for i := 1 to n do
  {Если элемент обладает заданными свойствами}
  if <условие> then
```

```
    begin
      {Учитываем его значение в сумме}
      sum := sum + a[i];
      {и учитываем этот элемент в количестве}
      kol := kol + 1
    end;
{Подсчет и вывод результата}
if kol > 0 then
  begin
    sred_arifm := sum/kol;
    write('Среднее арифметическое: ', sred_arifm:7:2)
  end
else
  write('Чисел, удовлетворяющих условию, в массиве нет');
```

Примечание. В ряде случаев в заданиях ЕГЭ указывается требование о разработке программы, эффективной с точки зрения используемой памяти. Для учета этого требования величину сред_арифм можно не использовать, а включить в команду **вывод** выражение для ее расчета:

```
если кол > 0
  то
    вывод нс, "Среднее арифметическое: ", сумма/кол
  иначе
    вывод нс, "Чисел, удовлетворяющих условию, в массиве нет"
все
```

1.2.5. Нахождение максимального количества подряд идущих элементов массива, обладающих заданными свойствами

Пример задачи: «Определить максимальное количество подряд идущих четных элементов в заданном целочисленном массиве».

В дальнейшем для краткости будем называть участок массива с указанными элементами «подмассивом». Значит, для решения обсуждаемой задачи надо определить максимальное из чисел – длин каждого подмассива (количество элементов в них). Это можно сделать по аналогии с определением максимальной цифры натурального числа (см. *п. 1.1.5*), только вместо цифр следует использовать длину каждого подмассива.

Используем в программе следующие основные величины:

- кол – число элементов в текущем подмассиве;
- макс_кол – искомое значение (максимальное количество подряд идущих элементов с заданными свойствами).

Можно рассуждать так. Если очередной элемент обладает заданными свойствами, то подмассив таких элементов продолжается или начался – увеличиваем его длину кол на 1, иначе – текущий подмассив закончился, и в этом случае:

1) сравниваем его длину с максимальной длиной уже рассмотренных ранее подмассивов макс_кол. Если длина текущего подмассива больше, то принимаем ее в качестве нового значения величины макс_кол;
2) имея в виду обработку следующего подмассива, обнуляем значение величины кол.

Фрагмент программы на основе сделанных рассуждений:

```
кол := 0
макс_кол := 0
нц для i от 1 до n
  если <условие>
   то
     |Подмассив продолжается или начался
     |Увеличиваем его длину на 1
     кол := кол + 1
   иначе |Встретился элемент, не обладающий заданными свойствами
     |Текущий подмассив закончился
     |Сравниваем его длину со значением макс_кол
     если кол > макс_кол
      то
       макс_кол := кол
     все
     кол := 0 /Новое значение
   все
кц
```

Однако при таких рассуждениях в случае, когда последний, n-й элемент массива также обладает заданными свойствами (например, четный), последний подмассив не будет учтен. Значит, нужно дополнительно учесть и его, сравнив длину этого подмассива с максимальной длиной:

```
|Проверяем длину последнего (возможного) подмассива
если кол > макс_кол
  то
    макс_кол := кол
все
|Выводим ответ
вывод нс, макс_кол
```

Обратим внимание на то, что после окончания текущего подмассива величине кол присваивается нулевое значение независимо от результата сравнения величин кол и макс_кол.

Язык Паскаль

```
kol := 0;
max_kol := 0;
for i := 1 to n do
  if <условие>
   then {Подмассив продолжается или начался.
         Увеличиваем его длину на 1}
     kol := kol + 1
   else {Встретился элемент, не обладающий
        заданными свойствами, - текущий подмассив закончился.
        Сравниваем его длину со значением max_kol}
      begin
       if kol > max_kol then max_kol := kol;
       kol := 0 {Новое значение}
      end;
{Проверяем длину последнего (возможного) подмассива}
if kol > max_kol
  then max_kol := kol;
{Выводим ответ}
write(max_kol);
```

Возможны также задачи рассмотренного типа, в которых начальное значение величины кол и ее значение после окончания некоторого подмассива принимаются равными 1. Это имеет место, когда первый элемент подмассива также учитывается в его длине.

Пример: «Определить максимальное количество подряд идущих совпадающих элементов целочисленного массива».

Ясно, что здесь первый из совпадающих элементов также должен быть учтен. Поэтому начальные значения переменных величин, хранящих длину текущего подмассива и максимальную длину подмассивов, равны 1 (если потом выяснится, что во всех парах находящихся рядом элементов массива нет одинаковых, то искомая величина будет равна 1):

```
кол_совп := 1
макс_совп := 1
нц для i от 2 до n
  если m[i] = m[i - 1]
    то
      кол_совп := кол_совп + 1
    иначе
      если кол_совп > макс_совп
        то
          макс_совп := кол_совп
      все
      кол_совп := 1 /Новое значение
  все
```

```
кц
если кол_совп > макс_совп
  то
    макс_совп := кол_совп
все
вывод макс_совп
```

где кол_совп – количество подряд идущих совпадающих элементов в текущем подмассиве; макс_совп – максимальное количество подряд идущих совпадающих элементов в подмассивах.

Язык Паскаль

```
kol_sovp := 1;
max_ sovp := 1;
for i := 2 to n do
  if m[i] = m[i - 1] then
    kol_sovp := kol_ sovp + 1
  else
    begin
      if kol_sovp > max_sovp
        then max_sovp := kol_sovp;
      kol_sovp := 1 {Новое значение}
    end;
if kol_sovp > max_sovp
  then max_sovp := kol_sovp;
write(max_sovp);
```

1.2.6. Нахождение максимальной суммы подряд идущих элементов массива, обладающих заданными свойствами

Пример задачи: «Определить максимальную сумму подряд идущих четных элементов в заданном целочисленном массиве».

Задача решается во многом аналогично предыдущей (при обработке подмассива рассчитывается сумма значений элементов, а по его окончании – сумма сравнивается с максимальной суммой, найденной ранее). Конечно, и здесь нужно при необходимости дополнительно проверить возможный последний подмассив:

```
сум := 0
макс_сум := 0
нц для i от 1 до n
  если <условие>
    то
      |Подмассив продолжается или начался
      |Учитываем текущий элемент в сумме
      сум := сум + m[i]
```

```
    иначе |Встретился элемент, не обладающий заданными свойствами
      |Текущий подмассив закончился
      |Сравниваем сумму его элементов со значением макс_сум
      если сум > макс_сум
        то
          макс_сум := сум
      все
      сум := 0 /Новое значение
  все
кц
|Проверяем сумму элементов последнего (возможного) подмассива
если сум > макс_сум
  то
    макс_сум := сум
все
вывод нс, макс_сум
```

Язык Паскаль

```
sum := 0;
max_sum := 0;
for i := 1 to n do
  if <условие>
   then {Подмассив продолжается или начался.
         Учитываем текущий элемент в сумме}
     sum := sum + m[i]
   else {Встретился элемент, не обладающий
         заданными свойствами, - текущий подмассив закончился}
       begin
         {Сравниваем сумму его элементов со значением max_sum}
         if sum > max_sum then max_sum := sum;
         summa := 0 {Новое значение}
       end;
{Проверяем сумму элементов последнего (возможного) подмассива}
if sum > max_sum then max_sum := sum;
{Выводим ответ}
write(max_sum);
```

Возможны также задачи обсуждаемого типа, в которых начальное значение величины сум и ее значение после окончания некоторого подмассива принимаются равными первому элементу массива (и, соответственно, текущему). Это имеет место, когда первый элемент подмассива также учитывается в его сумме.

Пример: «Определить сумму элементов наибольшей возрастающей последовательности подряд идущих элементов массива».

Обратим внимание на то, что здесь необходимо определить не максимальную длину[1] (количество элементов) подмассива и не

[1] Хотя эту величину также придется рассчитывать, но как вспомогательную.

максимальную сумму его элементов, а сумму элементов в подмассиве максимальной длины. Значит, нужно в ходе поиска такого подмассива рассчитывать и запоминать также сумму его элементов (в том числе и первого элемента подмассива).

В программе решения приведенного примера используем следующие основные величины:

- кол_возр – количество элементов возрастающей последовательности в текущем подмассиве;
- сумма_возр – сумма значений элементов в таком подмассиве;
- макс_кол – максимальное количество элементов в возрастающих последовательностях элементов;
- макс_сумма – максимальная сумма значений элементов в соответствующей последовательности.

Соответствующий фрагмент программы:

```
кол_возр := 1 |Учитываем первый
сумма_возр := m[1] |элемент
макс_кол := 1
нц для i от 2 до n
  если m[i] > m[i - 1]
    то
      кол_возр := кол_возр + 1
      сумма_возр := сумма_возр + m[i]
    иначе
      если кол > макс_кол
        то
          макс_кол := кол_возр
          макс_сумма := сумма_возр
      все
      |Новые значения
      сумма_возр := m[i]
      кол_возр := 1
  все
кц
если кол_возр > макс_кол
  то
    макс_сумма := сумма_возр
    |Здесь уточняем только значение макс_сумма
все
вывод нс, макс_сумма
```

Обратим внимание на то, что начальные значения величины кол_возр принимаются равными 1. Заметим также, что в решении, приведенном в [12], начальное значение величины макс_кол

принимается равным 0. Очевидно, авторы допускают возможность того, что в массиве не будет двух и более элементов, образующих возрастающую последовательность.

Язык Паскаль

```
kol_vozr := 1; {Учитываем первый}
summa_vozr := m[1]; {элемент}
max_kol := 1;
for i := 2 to n do
  if m[i] m[i] > m[i - 1] then
      begin
        kol_vozr := kol_vozr + 1
        summa_vozr := summa_vozr + m[i]
      end
   else
      begin
        if kol_vozr > max_kol then
          begin
            max_kol := kol_vozr;
            max_summa := summa_vozr
          end;
        {Новые значения}
        summa_vozr := m[i];
        kol_vozr := 1
      end;
if kol_vozr > max_kol
  then max_summa := summa_vozr; {Здесь уточняем только значение max_summa}
write(max_summa);
```

В заключение заметим, что задачи нахождения максимального/минимального значения среди элементов с заданными свойствами и номера такого значения рассматриваются в *разделе 1.4*.

1.3. Линейный поиск элемента

1.3.1. Проверка факта наличия в массиве элемента с заданным значением

Пример задачи: «Определить, есть ли в целочисленном массиве число 13».

Решение

Если перебрать все элементы массива, каждый сравнивать с числом 13 и в случае равенства выводить «Да, есть», в противном случае – выводить «Нет, такого числа нет»:

```
нц для i от 1 до n
  если a[i] = 13
    то
      вывод нс, "Да, число 13 есть"
    иначе
      вывод нс, "Нет, такого числа нет"
  все
кц
```

то на экран будет выведено несколько ответов (причем противоречащих друг другу). Ясно, что ответ (правильный) должен выводиться только один раз, и приведенное решение неприемлемо.

Еще одна типичная ошибка, встречающая при решении обсуждаемой задачи, – использование переменной (логического типа или принимающей значения 0 и 1), которая фиксирует факт равенства тринадцати для каждого проверяемого элемента, и вывод ответа в зависимости от значения этой переменной. В приведенном ниже фрагменте программы имя этой переменной – имеется.

```
нц для i от 1 до n
  если a[i] = 13
    то
      имеется := да |или имеется := 1
    иначе
      имеется := нет |или имеется := 0
  все
кц
если имеется |или имеется = да или имеется = 1
  то
    вывод нс, " Да, число 13 есть"
  иначе
    вывод нс, "Нет, такого числа нет"
все
```

Здесь ответ, выводимый на экран один раз, может быть ошибочным – он зависит от того, равен ли тринадцати последний элемент массива!

Правильный вариант:

```
имеется := нет |или имеется := 0
нц для i от 1 до n
  если a[i] = 13
    то
      имеется := да |или имеется := 1
  все
кц
если имеется |или имеется = да или имеется = 1
```

```
  то
    вывод нс, " Да, число 13 есть "
  иначе
    вывод нс, "Нет, такого числа"
все
```

Можно решить задачу и так:

1) подсчитать количество элементов массива, равных 13 (такая задача рассмотрена в *разделе 1.2*);
2) в зависимости от найденного количества вывести соответствующий ответ.

В обоих описанных вариантах решения массив просматривается полностью, в то время как искомое значение может оказаться в начале массива, и после нахождения числа 13 продолжать проверку остальных элементов массива нерационально. Желательно прекратить обработку массива после нахождения искомого элемента. В программах на языках программирования, в которых предусмотрена инструкция выхода из цикла (`break` или др.), для этого следует использовать эту инструкцию. Можно также использовать оператор цикла с условием. Например, для задачи, приведенной в начале раздела (определить, имеется ли в массиве четное число), соответствующий фрагмент должен быть оформлен в виде:

```
i := 1
нц пока i < n и не (a[i] <> 13) |Обратите внимание на условие
  i := i + 1
кц
```

В результате величина `i` не будет превышать значения `n`. Если значение элемента, на котором обработка массива остановилась (на последнем элементе или «досрочно»), равно 13, то это определяет один из вариантов ответа, в противном случае – второй вариант:

```
|Вывод результата
если a[i] = 13
  то
    вывод нс, "Да, число 13 есть"
  иначе
    вывод нс, "Нет, такого числа нет"
все
```

Обратим внимание на то, что при этом величина логического типа (или принимающая значение 0 или 1) не используется.