

§ 8

Структурированные типы данных. Массивы

Мы повторили основные приёмы работы с простыми типами данных. Из элементов простых типов можно образовывать составные типы данных (структуры данных). Примером таких структур являются массивы.

Массив — это поименованная совокупность однотипных элементов, упорядоченных по индексам, определяющим положение элемента в массиве.

Размерность массива — это количество индексов, необходимое для однозначного доступа к элементу массива. Массивы с одним индексом называют **одномерными**, с двумя — **двумерными** и т. д.

Все переменные, входящие в массив, имеют одно и то же имя — имя массива, а различаются они по индексу — номеру (месту) в массиве.



8.1. Общие сведения об одномерных массивах

Массив в языке Pascal — это набор однотипных данных, причём количество этих данных фиксировано и определяется при описании массива.

Описание массива на языке Pascal выглядит так:

```
var <имя массива>: array [<границы индекса>] of <тип компонент>
```

Здесь:

- **array** и **of** — служебные слова («массив» и «из»);
- <границы индекса> — описание индексации компонент (элементов) массива;
- <тип компонент> — тип величин, составляющих массив.

Например:

- **var day: array [1..365] of integer** — 365 целочисленных элементов пронумерованы от 1 до 365;
- **var tem: array [1..12] of real** — 12 вещественных элементов пронумерованы от 1 до 12;
- **var ocenka: array [2..5] of integer** — 4 целочисленных элемента пронумерованы от 2 до 5;
- **const n = 10; var slovo: array [1..n] of string** — n строковых величин пронумерованы от 1 до n.

В языке Python нет такой структуры данных, как «массив»; для хранения группы однотипных объектов используют списки — объекты типа `list`. Далее, работая со списками, мы будем использовать слово «массив».

Нумерация элементов массива в Python всегда начинается с нуля.

Перед использованием в программе на языке Python массив необходимо создать, в противном случае обращение к несуществующему элементу вызовет ошибку и аварийное завершение программы.

Создать массив можно перечислением элементов через запятую в квадратных скобках, например так: `a = [1, 2, -3, 5, 7]`. С помощью записи `a = [0]*5` будет создан массив из пяти элементов, каждый из которых равен 0.

Вспомним основные приёмы работы с массивами.

Пример 1. Имеются сведения о количестве ежедневных осадков в течение июня в некотором регионе. Требуется найти среднее количество осадков и вывести таблицу, в которой для каждого дня месяца указать количество осадков в этот день и его отклонение от среднемесячного значения.



Для решения этой задачи данные о количестве ежедневных осадков в течение месяца будут просмотрены дважды:

- 1) при поиске среднего значения;
- 2) при расчёте отклонения.

Для решения задачи нам понадобится массив из 30 вещественных чисел. Назовём его `osad`. В программе будет два цикла. В первом цикле мы введём значения элементов массива и сразу же подсчитаем их сумму — по завершении цикла мы получим сумму осадков, выпавших в течение месяца. Во втором цикле мы выведем строки таблицы и вычислим значения отклонений.

При работе с элементами массива воспользуемся переменной `osad[i]`; значение индекса `i` при этом будет изменяться от 1 до 30 с шагом 1 при записи программы на Pascal и от 0 до 29 с шагом 1 при записи программы на Python. Для вычисления среднего значения задействуем вещественную переменную `sred`, присвоив ей начальное значение 0 и последовательно накапливая в ней сумму осадков, выпавших в течение месяца. Разделив по завершении цикла полученное значение на 30, вычислим требуемое среднемесячное количество осадков и присвоим результат этой же переменной.

Программа на языке Pascal

```
program osadki;
var osad: array [1..30] of real;
    sred: real; i: integer;
begin
    sred := 0;
    writeln('Введите количество осадков по дням');
    for i := 1 to 30 do
        begin
            write(i, ' июня: ');
            readln(osad[i]);
            sred := sred + osad[i]
        end;
    sred := sred / 30;
    writeln('День Количество осадков Отклонение');
    for i := 1 to 30 do
        writeln(i:3, osad[i]:15:3, osad[i] - sred:12:2)
    end.
```

Программа на языке Python

```
osad=[0]*30
sred = 0
print('Введите количество осадков по дням')
for i in range (30):
    print(i+1, 'июня:')
    osad[i] = float(input())
    sred += osad[i]
sred /= 30
print('День Количество осадков Отклонение')
for i in range (30):
    print ("{0:3d}{1:15.3f}{2:12.2f}".format(i+1,
        osad[i],osad[i]-sred))
```



Найдите в Интернете информацию о количестве ежедневных осадков, выпавших в течение месяца, в вашем регионе. Используя эти данные, выполните программу в среде программирования Pascal.

Выполните аналогичные расчёты с помощью электронных таблиц.

К типовым задачам обработки одномерных массивов, решаемым в процессе их однократного просмотра, относятся:

- задачи поиска элементов с заданными свойствами, в том числе максимумов и минимумов;
- проверка соответствия элементов массива некоторому условию (подсчёт количества или суммы элементов, удовлетворяющих некоторому условию; проверка соответствия всех элементов массива некоторому условию; проверка массива на упорядоченность и др.);
- задачи на удаление и вставку элементов массива;
- задачи на перестановку всех элементов массива в обратном порядке и т. д.

8.2. Задачи поиска элемента с заданными свойствами

Очень часто в реальной жизни нам приходится сталкиваться с задачей поиска информации в большом массиве данных. Например, поиск нужного слова в словаре, поиск времени отправления нужного поезда в расписании, поиск нужного товара в интернет-магазине и т. д.

В программировании поиск — одна из наиболее часто встречающихся задач невычислительного характера.

В алгоритмах поиска существует два возможных варианта окончания их работы: поиск может оказаться удачным — заданный элемент найден в массиве и определено его месторасположение, либо поиск может оказаться неудачным — необходимого элемента в рассматриваемом массиве нет.

Рассмотрим несколько типовых задач поиска, первое знакомство с которыми у вас состоялось ещё в основной школе.

Пример 3. Последовательный поиск в неупорядоченном массиве

Имеется массив a , состоящий из n элементов; требуется найти элемент массива, равный p .

Алгоритм последовательного поиска в неупорядоченном массиве может быть следующим.

1. Установить $i = 1$ или $i = 0$ в зависимости от используемого языка программирования.
2. Если $a[i] = p$, алгоритм завершил работу успешно.
3. Увеличить i на 1.
4. Если $i \leq n$ ($i < n$), то перейти к шагу 2. В противном случае алгоритм завершил работу безуспешно.

Программа на языке Pascal

```
const n = 10;
var a: array [1..n] of integer; i, p: integer;
begin
  writeln('Ввод значений элементов массива:');
  for i := 1 to n do
    read(a[i]);
  write('Ввод p: ');
  readln(p);
  i := 1;
  while (i <= n) and (a[i] <> p) do i := i + 1;
  if i = n + 1
  then writeln('Искомго элемента в массиве нет')
  else writeln('Искомый элемент a[' , i, ' ] = ' , a[i])
end.
```

Программа на языке Python

```

n = 10
a = [0] * n
print('Ввод значений элементов массива:')
for i in range (n):
    a[i]=int(input())
print('Ввод p: ')
p = int(input())
i = 0
while i < n and a[i] != p:
    i += 1
if i == n:
    print('Искомое элемента в массиве нет')
else:
    print('Искомый элемент a[' , i , ' ] =', a[i])

```



Внимательно рассмотрите условие продолжения цикла. В каком случае выполнение цикла продолжается? В каких случаях осуществляется выход из цикла?



Запустите программу на выполнение в среде программирования Pascal (Python).

Как иначе можно решить эту задачу, например, с использованием цикла `for`? Напишите соответствующую программу.



Оценим сложность рассмотренного алгоритма последовательного поиска, непосредственно зависящую от числа сравнений с искомым элементом. В худшем случае искомый элемент окажется на последнем месте или не будет найден вообще. В таком случае необходимо будет проделать n сравнений, т. е. сложность алгоритма будет равна $O(n)$.



Пример 4. Поиск максимумов и минимумов

Имеется массив a , состоящий из n элементов; требуется найти значение наибольшего (наименьшего) элемента массива.

Алгоритм поиска значения наибольшего (максимального) элемента в неупорядоченном массиве может быть следующим.

1. Установить значение текущего максимума равным первому исследуемому элементу: $\max := a[1]$ ($\max = a[0]$).
2. Установить начальное значение счётчика исследуемых элементов: $i := 2$ ($i = 1$).

3. Если исследованы ещё не все элементы: $i \leq n$ ($i < n$), то перейти к шагу 4, иначе алгоритм окончен (максимальный элемент равен \max).
4. Если рассматриваемый элемент больше, чем текущий максимум ($a[i] > \max$), то \max присвоить значение $a[i]$.
5. Перейти к следующему элементу (увеличить i на единицу).
6. Перейти к шагу 3.

Программа на языке Pascal

```
const n = 10;
var a: array [1..n] of integer;
    i, max: integer;
begin
  writeln('Ввод значений элементов массива:');
  for i := 1 to n do
    read(a[i]);
  max := a[1];
  i := 2;
  while (i <= n) do
    begin
      if a[i] > max then max := a[i];
      i := i + 1
    end;
  writeln('Max=', max)
end.
```

Программа на языке Python

```
n = 10
a = [0] * n
print('Ввод значений элементов массива:')
for i in range (n):
    a[i] = int(input())
max = a[0]
i = 1
while i < n:
    if a[i] > max:
        max = a[i]
    i += 1
print('Max =', max)
```



Запустите программу на выполнение в среде программирования Pascal (Python).



Как иначе можно решить эту задачу, например, с использованием цикла `for`? Напишите соответствующую программу.

Преобразуйте программу так, чтобы с её помощью можно было находить минимальный элемент массива.

Какие изменения надо внести в программу для поиска индекса максимального (минимального) элемента массива?



Самостоятельно оцените сложность рассмотренного алгоритма.

8.3. Проверка соответствия элементов массива некоторому условию



Пример 5. Подсчёт количества элементов, удовлетворяющих некоторому условию

Зачастую бывает важно выяснить, сколько элементов, обладающих определённым свойством, содержится в массиве.

Для решения этой задачи следует:

- 1) присвоить нулевое значение переменной, введённой для подсчёта количества элементов, удовлетворяющих заданному условию: `k := 0` ($k = 0$);
- 2) организовать просмотр всех элементов массива: если просматриваемый элемент удовлетворяет заданному условию, значение переменной `k` увеличивать на 1.

Фрагмент программы подсчёта количества элементов массива, например больших некоторого числа `p`, имеет следующий вид:

| Программа на языке Pascal | Программа на языке Python |
|---|---|
| <pre>k := 0; for i := 1 to n do if a[i] > p then k := k + 1;</pre> | <pre>k = 0 for i in range (n): if a[i] > p: k += 1</pre> |



Запишите полный текст программы и выполните её на компьютере для рассматриваемого в примере 8 массива `a`, состоящего из семи элементов, и числа `p = 15`.

Как модифицировать программу, чтобы можно было вычислить сумму элементов массива, больших некоторого числа `p`?

Пример 6. Проверка соответствия всех элементов массива некоторому условию

Для того чтобы установить факт соответствия всех элементов массива некоторому условию, достаточно:

- 1) подсчитать количество элементов массива, соответствующих заданному условию;
- 2) сравнить найденное количество с общим числом элементов массива и вывести соответствующий результат.

Самостоятельно разработайте программу, позволяющую определить, все ли элементы массива являются двузначными числами. Выполните её на компьютере для рассматриваемого в примере 8 массива a , состоящего из семи элементов.

Пример 7. Проверка массива на упорядоченность

Рассмотрим алгоритм, позволяющий определить, упорядочены ли элементы массива a , состоящего из n элементов, по неубыванию (т. е. каждый элемент массива с первого по предпоследний не больше последующего).

Самый простой путь решения этой задачи — проверить, есть ли в массиве такие пары элементов, что $a[i] > a[i+1]$. Если подобные пары элементов есть, то массив не упорядочен по неубыванию, а если таких пар нет, то упорядочен.

В программе будем использовать логическую переменную $flag$:

- если $flag = true$, то массив упорядочен;
- если $flag = false$, то массив неупорядочен.

Ниже представлен фрагмент программы, реализующей этот алгоритм:

Программа на языке Pascal

```
flag := true;
for i := 1 to n - 1 do
  if a[i] > a[i+1] then flag := false;
```

Программа на языке Python

```
flag = True
for i in range(n-1):
  if a[i] > a[i+1]: flag = False
```

Запишите полный текст программы и выполните её на компьютере для рассматриваемого в примере 8 массива a , состоящего из семи элементов.

Как можно решить эту же задачу путём подсчёта количества пар элементов массива, таких что $a[i] > a[i+1]$?



8.4. Удаление и вставка элементов массива



Пример 8. Удаление из массива элемента, занимающего k -е место

Имеется одномерный целочисленный массив из семи элементов:

| | | | | | | | |
|-------------------|----|----|---|---|---|----|----|
| i (Pascal) | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| a[i] | 10 | 12 | 5 | 8 | 4 | 15 | 20 |
| i (Python) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Удалим из массива элемент, занимающий 4-е место, а все элементы, расположенные справа от него, сдвинем на одну позицию влево. Получим следующий целочисленный массив из шести элементов:

| | | | | | | |
|-------------------|----|----|---|---|----|----|
| i (Pascal) | 1 | 2 | 3 | 4 | 5 | 6 |
| a[i] | 10 | 12 | 5 | 4 | 15 | 20 |
| i (Python) | 0 | 1 | 2 | 3 | 4 | 5 |

При удалении из массива любого из элементов размерность массива уменьшается на 1.

Мы видим, что первые три элемента не изменились. На место четвёртого элемента переместился пятый элемент, на место пятого элемента переместился шестой элемент и т. д.

В общем случае фрагмент программы удаления из массива элемента, занимающего k -е место, с последующим сдвигом всех расположенных справа от него элементов на одну позицию влево имеет вид:

| Программа на языке Pascal | Программа на языке Python |
|---|--|
| <pre>for i := k to n-1 do a[i] := a[i+1];</pre> | <pre>for i in range(k-1, n-1): a[i] = a[i+1]</pre> |



Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива a .

Пример 9. Вставка в массив элемента на k-е место

Будем работать с тем же массивом из семи элементов, как в примере 8. Но теперь наша задача будет состоять в том, чтобы вставить в массив на четвёртое место новый элемент, имеющий значение 11.

Получим следующий целочисленный массив из восьми элементов:

| | | | | | | | | |
|-------------------|----|----|---|----|---|---|----|----|
| i (Pascal) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| a[i] | 10 | 12 | 5 | 11 | 8 | 4 | 15 | 20 |
| i (Python) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

При вставке в массив ещё одного элемента размерность массива увеличивается на 1. Это надо учесть при описании массива в Pascal (создании в Python).

Итак, элемент, занимающий 4-е место, теперь равен 11. Элементу, занимающему 5-е место, следует присвоить то значение, которое ранее было у 4-го элемента, 6-му элементу — то значение, которое ранее было у 5-го элемента, и т. д. В общем случае элементу $a[k + 1]$ следует присвоить то значение, которое было у $a[k]$.

Подумайте, что получится в результате выполнения следующих групп операторов присваивания:

- 1) $a[4] := 11; a[5] := a[4]; a[6] := a[5]; a[7] := a[6]; a[8] := a[7];$
- 2) $a[8] := a[7]; a[7] := a[6]; a[6] := a[5]; a[5] := a[4]; a[4] := 11;$

Фрагмент программы вставки в массив a элемента на k -е место и сдвигом k -го, $(k+1)$ -го, ..., $(n-1)$ -го элементов на одну позицию вправо имеет вид:

| Программа на языке Pascal | Программа на языке Python |
|--|---|
| <pre>for i := n downto k+1 do a[i] := a[i-1]; a[k] := <значение элемента>;</pre> | <pre>for i in range(n-1, k): a[i] = a[i-1] a[k] = <значение элемента></pre> |

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива a . Помните, что при описании массива надо учесть размерность массива, получающегося в результате работы программы.



8.5. Перестановка всех элементов массива в обратном порядке



Пример 10. Перестановка в обратном порядке всех элементов массива a , состоящего из n элементов, сводится к тому, что меняются местами первый и последний элементы, второй и предпоследний элементы и т. д.

Перестановка нашего массива из семи элементов (см. пример 8) даст такой результат:

| | | | | | | | |
|--------------|----|----|---|---|---|----|----|
| i (Pascal) | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $a[i]$ | 20 | 15 | 4 | 8 | 5 | 12 | 10 |
| i (Python) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

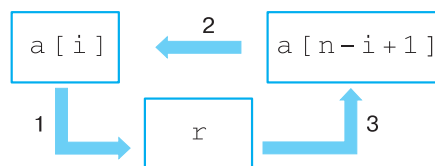
В общем случае меняются местами элементы $a[i]$ и $a[n-i+1]$.

Вспомним, как можно произвести обмен значений между двумя переменными. Выполнение операторов:

```
a[1]:=a[n]; a[n]:=a[1];
```

к желаемому результату не приводит. Самый простой вариант — использование вспомогательной переменной:

```
r := a[i];
a[i] := a[n-i+1];
a[n-i+1] := r;
```



Выясним, сколько всего операций обмена следует произвести.

Если произведена перестановка, например, первого и последнего элементов, то одновременно произведена и перестановка последнего и первого элементов. Таким образом, если число элементов массива чётное, то достаточно произвести $n/2$ операций обмена.

Но что происходит, если массив содержит нечётное число элементов? Например, в нашем массиве из семи элементов выполнялось три обмена, а четвёртый элемент, занимающий центральную позицию, оставался на своём месте. В общем случае число операций обмена при перестановке в обратном порядке всех n элементов массива определяется как $n \text{ div } 2$ ($n // 2$).

В общем случае фрагмент программы по перестановке в обратном порядке всех элементов массива a имеет вид:

| Программа на языке Pascal | Программа на языке Python |
|--|--|
| <pre>for i := 1 to n div 2 do begin r := a[i]; a[i] := a[n-i+1]; a[n-i+1] := r end</pre> | <pre>for i in range(n//2 + 1): r = a[i] a[i] = a[n-i-1] a[n-i-1] = r</pre> |

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива a , состоящего из семи и из шести элементов.



8.6. Сортировка массива

Сортировка — один из наиболее распространённых процессов современной обработки данных.

Сортировка — это распределение элементов массива в соответствии с определёнными правилами.



Под сортировкой (упорядочением) массива понимают перераспределение значений его элементов в некотором определённом порядке.

Порядок, при котором в массиве первый элемент имеет самое маленькое значение, а значение каждого следующего элемента не меньше значения предыдущего элемента, называют **неубывающим**.

Порядок, при котором в массиве первый элемент имеет самое большое значение, а значение каждого следующего элемента не больше значения предыдущего элемента, называют **невозрастающим**.

Цель сортировки — ускорить последующий поиск элементов, так как нужный элемент легче искать в упорядоченном массиве.

Рассмотрим и проанализируем несколько алгоритмов сортировки для решения следующей задачи. Дан одномерный массив целых чисел. Требуется отсортировать его так, чтобы все элементы были расположены в порядке неубывания: $a[i] \leq a[i + 1]$.

Обменная сортировка методом «пузырька»

Свое название алгоритм получил благодаря следующей ассоциации: если сортировать этим алгоритмом массив по неубыванию, то максимальный элемент «тонет», а «лёгкие» элементы поднимаются на одну позицию к началу массива на каждом шаге алгоритма.

Пусть n — количество элементов в неупорядоченном массиве.

1. Поместим на место n -го элемента ($a[n]$) наибольший элемент массива. Для этого:
 - 1) присвоим i значение индекса 1-го элемента массива;
 - 2) пока не обработана последняя пара элементов, т. е. $(n-1)$ -й и n -й элементы:
 - сравниваем i -й и $(i+1)$ -й элементы массива;
 - если $a[i] > a[i+1]$ (элементы расположены не по порядку), то меняем элементы местами;
 - переходим к следующей паре элементов, сдвинувшись на один элемент вправо.
2. Повторяем пункт 1, каждый раз уменьшая размерность неупорядоченного массива на 1, до тех пор, пока не будет обработан массив из одной пары элементов (таким образом, на k -м просмотре будут сравниваться первые $(n-k)$ элементов со своими соседями справа).



Пример 11. Есть массив: 5 4 3 2 1. На примере этого массива подсчитаем количество элементарных действий в вычислительном процессе алгоритма сортировки методом «пузырька»:

- 1-я итерация: 4 3 2 1 5 (4 сравнения, 4 обмена);
- 2-я итерация: 3 2 1 4 5 (3 сравнения, 3 обмена);
- 3-я итерация: 2 1 3 4 5 (2 сравнения, 2 обмена);
- 4-я итерация: 1 2 3 4 5 (1 сравнение, 1 обмен).

Алгоритм закончил работу. Было сделано 10 сравнений и 10 обменов ($4 + 3 + 2 + 1$).

Этот алгоритм легко запоминается, но на практике он используется достаточно редко из-за квадратичной сложности, означающей, что в общем случае количество выполненных сравнений и обменов сопоставимо с n^2 , где n — количество элементов массива.



Попытайтесь самостоятельно запрограммировать алгоритм сортировки методом «пузырька».

Сортировка выбором

Сортировка выбором (в порядке неубывания) осуществляется следующим образом:

- 1) в массиве выбирается минимальный элемент;
- 2) минимальный и первый элементы меняются местами (первый элемент считается отсортированным);
- 3) в неотсортированной части массива снова выбирается минимальный элемент и меняется местами с первым неотсортированным элементом массива;
- 4) действия, описанные в пункте 3, повторяются с неотсортированными элементами массива до тех пор, пока не останется один неотсортированный элемент (его значение будет максимальным).

Пример 12. Есть массив: 5 4 3 2 1.

1-я итерация: 1 4 3 2 5 (4 сравнения, 1 обмен);

2-я итерация: 1 2 3 4 5 (3 сравнения, 1 обмен);

3-я итерация: 1 2 3 4 5 (2 сравнения, 0 обменов);

4-я итерация: 1 2 3 4 5 (1 сравнение, 0 обменов).

В общем случае алгоритм сортировки выбором имеет квадратичную сложность относительно операций сравнения и линейную сложность относительно операций обменов. Этот алгоритм целесообразно применять, когда операция обмена над элементами массива особенно трудоёмка (например, если элементом массива является запись с большим числом полей).

Приведём фрагмент программы, реализующей описанный выше алгоритм:

Программа на языке Pascal

```
for i := 1 to n - 1 do
begin
  imin := i;
  for j := i + 1 to n do
    if a[j] < a[imin] then imin := j;
  per := a[i];
  a[i] := a[imin];
  a[imin] := per;
end;
```

Программа на языке Python

```
for i in range(n-1):
    imin = i
    for j in range(i+1, n):
        if a[j] < a[imin]: imin = j
    per = a[i]
    a[i] = a[imin]
    a[imin] = per
```

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива.

САМОЕ ГЛАВНОЕ

Из элементов простых типов в языке Pascal можно образовывать составные типы данных (структуры данных). Примером таких структур являются одномерные массивы.

Массив в языке Pascal — это набор однотипных данных, причём количество этих данных фиксировано и определяется при описании массива. Все переменные, входящие в массив, имеют одно и то же имя — имя массива, а различаются они по индексу — номеру (месту) в массиве.

Перед использованием в программе массив должен быть описан (в Pascal) или создан (в Python), т. е. должно быть указано имя массива, количество элементов массива и их тип. Это необходимо для того, чтобы выделить в памяти под массив блок ячеек нужного типа.

Чаще всего массив обрабатывается в цикле **for**. Но при работе с массивами можно использовать и другие циклы.

К типовым задачам обработки одномерных массивов, решаемым в процессе их однократного просмотра, относятся:

- задачи поиска элемента с заданными свойствами, в том числе максимумов и минимумов;
- проверка соответствия элементов массива некоторому условию (подсчёт количества или суммы элементов, удовлетворяющих некоторому условию; проверка соответствия всех элементов массива некоторому условию; проверка массива на упорядоченность и др.);
- задачи на удаление и вставку элементов массива;
- задачи на перестановку всех элементов массива в обратном порядке и т. д.

Сортировка — один из наиболее распространённых процессов современной обработки данных. Под сортировкой (упорядочением) массива понимают перераспределение значений его элементов в некотором определённом порядке.

Вопросы и задания

1. Приведите примеры задач поиска информации в больших массивах данных.
2. Почему важно уметь решать задачи, связанные с обработкой массивов путём однократного просмотра массива?
3. Программист написал программу суммирования элементов массива, но допустил в ней ошибку.

Программа на языке Pascal

```
program summa;
const n = 10;
var a: array [1..n] of integer; s, i: integer;
begin
  s := 0;
  for i := 1 to n do
    begin
      readln(a[i]);
      s := s + i
    end;
  writeln('s=', s)
end.
```

Программа на языке Python

```
n = 10
a = [0] * n
s = 0
for i in range(n):
    a[i] = int(input())
    s += i
print('s =', s)
```

- a) Что получится в результате выполнения этой программы, если в качестве элементов массива ввести числа: 1, -2, 3, -4, 5, -6, 7, -8, 9, -10?

- б) Придумайте пример такого массива, обработка которого с помощью этой программы приводила бы к правильному результату.
 - в) Найдите ошибку, допущенную программистом.
4. Программист написал программу нахождения произведения элементов массива, но допустил в ней ошибку.

Программа на языке Pascal

```
program proizv;  
const n = 10;  
var a: array [1..n] of integer; p, i: integer;  
begin  
  p := 0;  
  for i := 1 to n do  
    begin  
      readln(a[i]);  
      p := p * a[i]  
    end;  
  writeln('p=', p)  
end.
```

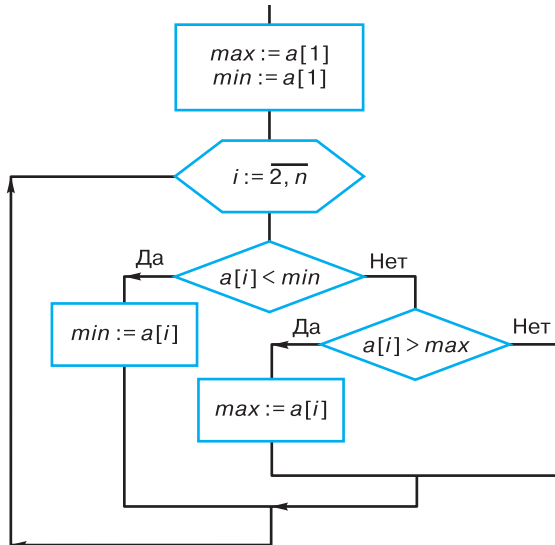
Программа на языке Python

```
n = 10  
a = [0] * n  
p = 0  
for i in range(n):  
    a[i] = int(input())  
    p *= a[i]  
print('p =', p)
```

- а) Что получится в результате выполнения этой программы, если в качестве элементов массива ввести числа: 1, -2, 3, -4, 5, -6, 7, -8, 9, -10?
- б) Придумайте пример такого массива, обработка которого с помощью этой программы приводила бы к правильному результату.
- в) Найдите ошибку, допущенную программистом.



5. На блок-схеме представлен алгоритм одновременного поиска максимального и минимального значений элементов массива:



Реализуйте этот алгоритм на языке программирования и выполните программу для массива из задания 6.

6. Имеется ли разница между операциями вставки в массив элемента на место с индексом k и замены значения элемента массива с индексом k ? Обоснуйте свой ответ.
7. Имеется одномерный целочисленный массив из семи элементов:

| | | | | | | |
|----|----|---|---|---|----|----|
| 10 | 12 | 5 | 8 | 4 | 15 | 20 |
|----|----|---|---|---|----|----|

Каким будет результат преобразования массива по следующему алгоритму, записанному на двух языках программирования?

| Программа на языке Pascal | Программа на языке Python |
|---|--|
| <pre> for i := 1 to n div 2 do begin r := a[i]; a[i] := a[n-i+1]; a[n-i+1] := r end; </pre> | <pre> for i in range(n//2): r = a[i] a[i] = a[n-i-1] a[n-i-1] = r </pre> |

8. Дана программа на двух языках программирования:

Программа на языке Pascal

```
const n = 5;
const a: array[1..n] of integer = (1,2,6,4,6);
var i, max1, max2: integer;
begin
  max1 := a[1];
  max2 := a[2];
  for i := 2 to n do
    if a[i] > max1
    then begin max2 := max1; max1 := a[i]; end
    else if a[i] > max2 then max2 := a[i];
  writeln('max1=', max1, ', max2=', max2);
end.
```

Программа на языке Python

```
n = 5
a = [1,2,6,4,6]
max1 = a[0]
max2 = a[1]
for i in range(1, n):
    if a[i] > max1:
        max2, max1 = max1, a[i]
    else:
        if a[i] > max2: max2 = a[i]
print('max1=', max1, ', max2=', max2, sep='')
```

Что получится в результате выполнения этой программы?
Какую задачу решает эта программа?



9. Дано натуральное десятичное число $n \leq 32\,000$. Напишите программу, в которой:
- 1) из цифр данного числа формируется одномерный целочисленный массив;
 - 2) определяются наибольшая и наименьшая цифры данного числа;
 - 3) находятся сумма и произведение цифр, образующих данное число.
10. Требуется упорядочить по весу в порядке неубывания n непрозрачных банок с чаем, имея в своём распоряжении только чашечные весы без гирь. Опишите возможный алгоритм решения этой задачи.

