Гомельский государственный университет
имени Франциска Скорины

Массивы

Составил: Ассистент кафедры общей физики
Соколов С.И.

Гомель, 2015

Теперь мы приступаем к изучению массива - наиболее широко используемого структурированного типа данных, предназначенного для хранения нескольких однотипных элементов.

Массив – это последовательность однотипных данных, объединенная общим именем, элементы (компоненты) которой отличаются (идентифицируются) индексами.

Индекс элемента указывает место (номер) элемента в массиве.

Количество элементов массива фиксировано и определено в его описании. К элементам массива можно обращаться только по их номеру (индексу). Все компоненты массива являются одинаково доступными. Значения элементам массива присваиваются также как и другим переменным с учетом типа массива.

Массивы

Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом.

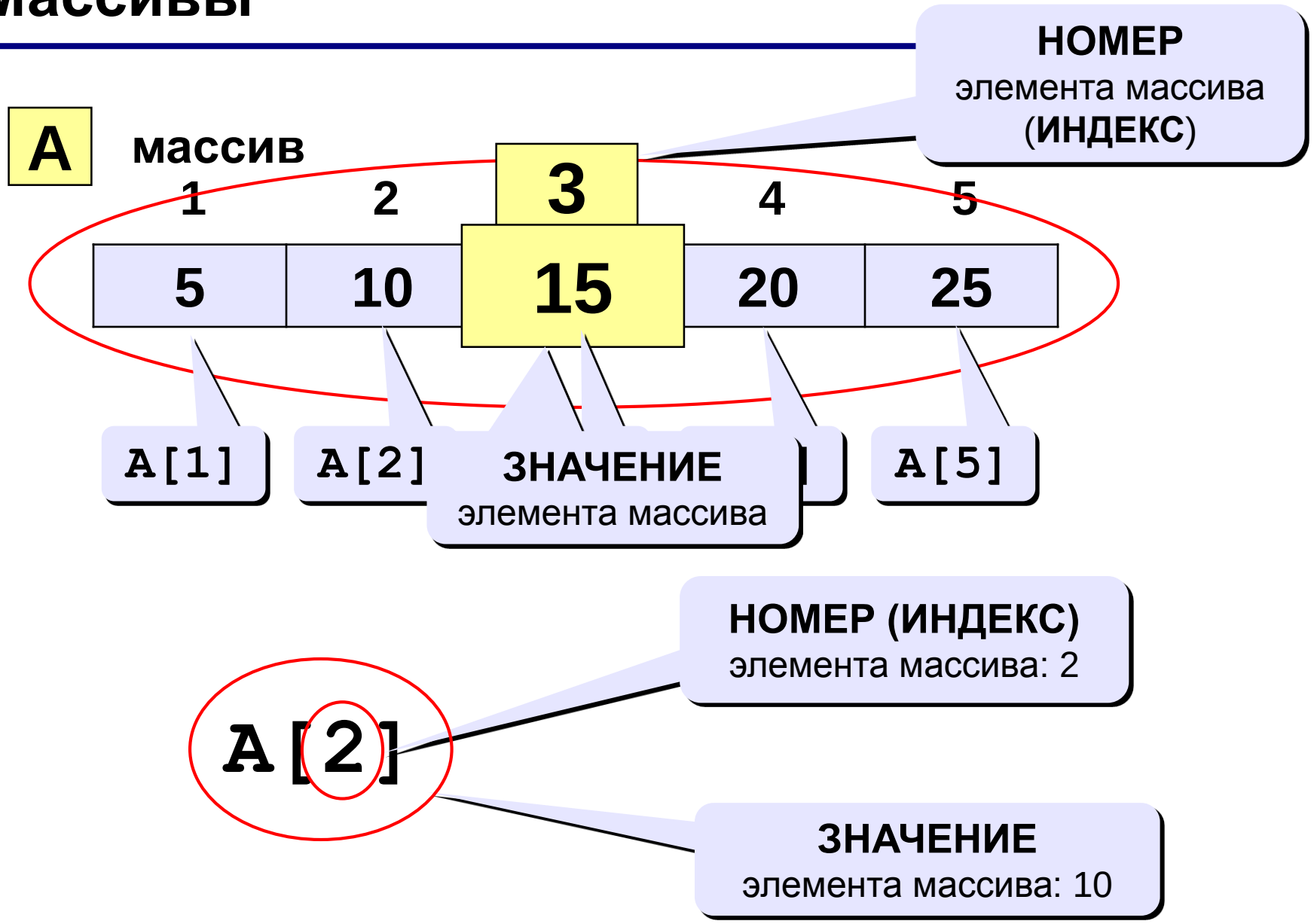
Особенности:

- все элементы имеют **один тип**
- весь массив имеет **одно имя**
- все элементы расположены в памяти **рядом**

Примеры:

- список студентов в группе
- квартиры в доме
- школы в городе
- данные о температуре воздуха за год

Массивы

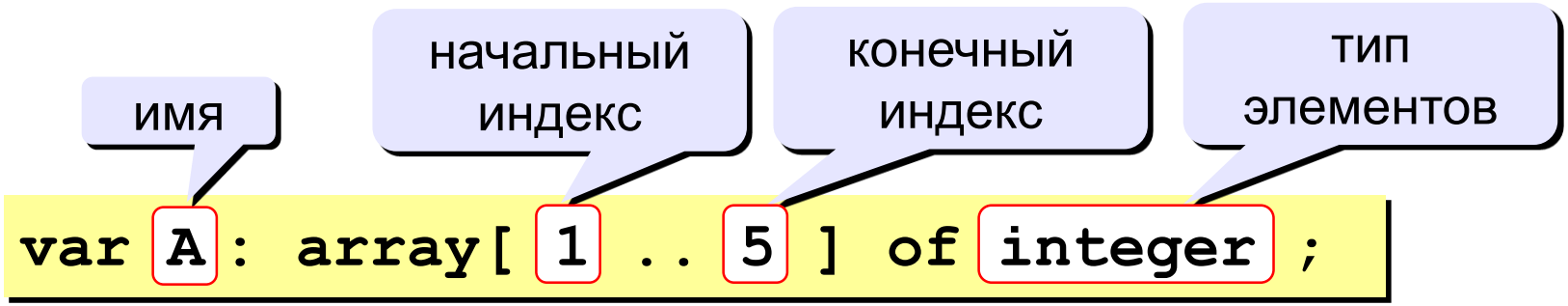

НОМЕР
элемента массива
(ИНДЕКС)
A
массив
1
2
3
4
5
5
10
15
20
25
A[1]
A[2]
ЗНАЧЕНИЕ
элемента массива
A[5]
НОМЕР (ИНДЕКС)
элемента массива: 2
A[2]
ЗНАЧЕНИЕ
элемента массива: 10

Объявление массивов

Зачем объявлять?

- определить **имя** массива
- определить **тип** массива
- определить **число элементов**
- выделить **место в памяти**

Массив целых чисел:



Размер через константу:

```
const N=5;
var A: array[1..N] of integer;
```

Объявление массивов

Массивы других типов:

```
var X, Y: array [1..10] of real;
    C: array [1..20] of char;
```

Другой диапазон индексов:

```
var Q: array [0..9] of real;
    C: array [-5..13] of char;
```

Индексы других типов:

```
var A: array ['A'..'Z'] of real;
    B: array [False..True] of integer;
...
   A['C'] := 3.14259*A['B'];
   B[False] := B[False] + 1;
```

Объявление массива

Для того чтобы задать массив, необходимо в разделе описания переменных (***var***) указать его размеры и тип его компонент.

Общий вид описания (одномерного) массива:

```
array [<тип_индексов>] of <тип_компонент>;
```

Чаще всего это трактуется так:

```
array [<левая_граница>..<правая_граница>]
                of <тип_компонент>;
```

Нумерация

Нумеровать компоненты массива можно не только целыми числами.

Любой порядковый тип данных (перечислимый, интервальный, символьный, логический, а также произвольный тип, созданный на их основе) имеет право выступать в роли нумератора.

Таким образом, допустимы следующие описания массивов:

```
type charrr = 'a','c'..'z';
      (- отсутствует символ "b")
var a1: array[charrr] of integer;
              - 25 компонент
a2: array [char] of integer;
              - 256 целых компонент
a3: array [shortint] of real;
              - 256 вещественных компонент
```

Нумерация

Общий размер массива не должен превосходить 65 520 байт. Следовательно, попытка задать массив

```
a4:array[integer] of byte;
```

не увенчается успехом, поскольку тип ***integer*** покрывает 65 535 различных элементов.

А про тип ***longint*** в данном случае лучше и вовсе не вспоминать.

Тип компонент

Тип компонент массива может быть любым:

```
var a4: array[10..20] of real;
```
- массив из компонент простого типа

```
a5: array[0..100] of record1;
```
- массив из записей

```
a6: array[-10..10] of ^string;
```
- массив из указателей на строки

```
a7: array[-1..1] of file;
```
- массив из имен файловых переменных

```
a8: array[1..100] of array[1..100] of char;
```
- двумерный массив (массив векторов)

Что неправильно?

```
var a: array[1..10] of integer;
...
  A[5] := 4.5;
```

```
var a: array['a'..'z'] of integer;
...
  A['b'] := 15;
```

```
var a: array [0..9] of integer;
...
  A[10] := 'X';
```

Массивы

Объявление:

```
const N = 5;
var a: array[1..N] of integer;
    i: integer;
```

Ввод с клавиатуры:

```
for i:=1 to N do begin
  write('a[', i, ']=');
  read ( a[i] );
end;
```

```
a[1] = 5
a[2] = 12
a[3] = 34
a[4] = 56
a[5] = 13
```

Почему write?

Поэлементные операции:

```
for i:=1 to N do a[i]:=a[i]*2;
```

Вывод на экран:

```
writeln('Массив A:');
for i:=1 to N do
  write(a[i]:4);
```

```
Массив A:
  10  24  68 112  26
```

Многомерные массивы

Для краткости и удобства ***многомерные массивы*** можно описывать и более простым способом:

```
var a9: array[1..10,1..20] of real;
      - двумерный массив 10 x 20
a10: array[boolean, -1..1, char, -10..10]
          of word;
      - четырехмерный массив 2 x 3 x 256 x 21
```

Общее ограничение на размер массива - не более 65 520 байт - сохраняется и для многомерных массивов.

Количество компонент многомерного массива вычисляется как произведение всех его "измерений".

Таким образом, в массиве ***a9*** содержится 200 компонент, а в массиве ***a10*** - 32 256 компонент.

Описание переменных размерностей

Если ваша программа должна обрабатывать матрицы переменных размерностей (***N*** по горизонтали и ***M*** по вертикали), то вы столкнетесь с проблемой изначального задания массива, ведь в разделе ***var*** не допускается использование переменных.

Следовательно, самый логичный, казалось бы, вариант

```
var  m, n: integer;
  a: array[1..m, 1..n] of real;
```

придется отбросить.

Описание переменных размерностей

Если на этапе написания программы ничего нельзя сказать о предполагаемом размере входных данных, то не остается ничего другого, как воспользоваться техникой динамически распределяемой памяти

Описание переменных размерностей

Предположим, однако, что известны максимальные границы, в которые могут попасть индексы обрабатываемого массива. Скажем, ***N*** и ***M*** заведомо не могут превосходить 100.

Тогда можно выделить место под наибольший возможный массив, а реально работать только с малой его частью:

```
const nnn=100;
var  a: array[1..nnn,1..nnn] of real;
  m, n: integer;
```

Обращение к компонентам массива

Массивы относятся к структурам прямого доступа. Это означает, что возможно напрямую (не перебирая предварительно все предшествующие компоненты) обратиться к любой интересующей нас компоненте массива.

Доступ к компонентам линейного массива осуществляется так:

```
<имя_массива>[<индекс_компоненты>]
```

а многомерного – так:

```
<имя_массива>[ <индекс 1>, ..., <индекс K>]
```

Обращение к компонентам массива

Массив состоит из элементов, имеющих порядковые номера, т.е. элементы массива упорядочены. Таким образом, если объекты одного типа обозначить именем, например "***A***", то элементы объекта будут ***A[1]***, ***A[2]*** и т.д. В квадратных скобках указан номер элемента.

Порядковый номер элемента массива, обычно не несет никакой информации о значении элемента, а показывает расположение элемента среди других.

Обращение к компонентам массива

Правила употребления ***индексов*** при обращении к компонентам массива:

1) Индекс компоненты может быть константой, переменной или выражением, куда входят операции и вызовы функций.
2) Тип каждого индекса должен быть совместим с типом, объявленным в описании массива именно для соответствующего "измерения"; менять индексы местами нельзя.
3) Количество индексов не должно превышать количество "измерений" массива. Попытка обратиться к линейному массиву как к многомерному обязательно вызовет ошибку. А вот обратная ситуация вполне возможна: например, если вы описали ***N***-мерный массив, то его можно воспринимать как линейный массив, состоящий из (***N***-1)-мерных массивов.

Описание переменных размерностей

Примеры использования компонент массива:

```
a1[1, 3]:= 0;
a1[i, 2]:= a1[i, 2]-1;
a2['z']:= a2['z']+1;
a3[-10]:= 2.5;
a3[i+j]:= a9[i, j];
a10[x>0, sgn(x), '!', abs(k*5)]:= 0;
```

Задание массива константой

Чтобы не вводить массивы вручную во время отладки программы, можно пользоваться не только файлами. Существует и более простой способ, когда входные данные задаются прямо в тексте программы при помощи типизированных констант.

Если массив линейный (вектор), то начальные значения для компонент этого вектора задаются через запятую, а сам вектор заключается в круглые скобки.

Задание массива константой

Многомерный массив также можно рассматривать как линейный, предполагая, что его компонентами служат другие массивы. Т.о., для системы вложенных векторов действует то же правило задания типизированной константы: каждый вектор ограничивается снаружи круглыми скобками.

Исключение составляют только массивы, компонентами которых являются величины типа ***char***. Такие массивы можно задавать проще: строкой символов.

Задание массива константой

Примеры задания массивов типизированными константами:

```
type mass = array[1..3, 1..2] of byte;
const a: array[-1..1] of byte = (0,0,0);
                    {линейный}
  b: mass = ((1, 2),(3, 4),(5, 6));
                    {двумерный}
  s: array[0..9] of char = '0123456789';
```

Поиск элемента в массиве

Постановка задачи

Пусть $A = \{a_1, a_2, ...\}$ – последовательность однотипных элементов и b – некоторый элемент, обладающий свойством P. **Найти место элемента** b в последовательности A.

Постановка задачи

Поскольку представление последовательности в памяти может быть осуществлено в виде массива, задачи могут быть уточнены как одна из следующих задач поиска элемента в массиве *A*:

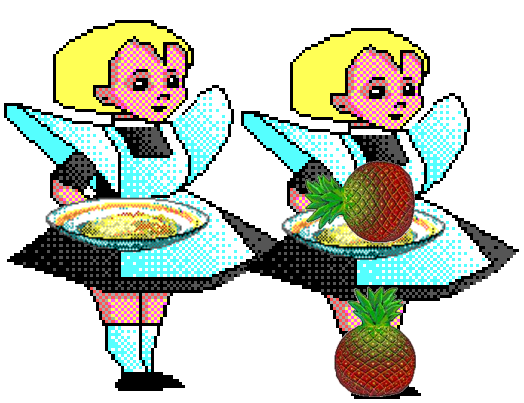
- найти максимальный (минимальный) элемент массива;
- найти заданный элемент массива;
- найти *k*–ый по величине элемент массива.

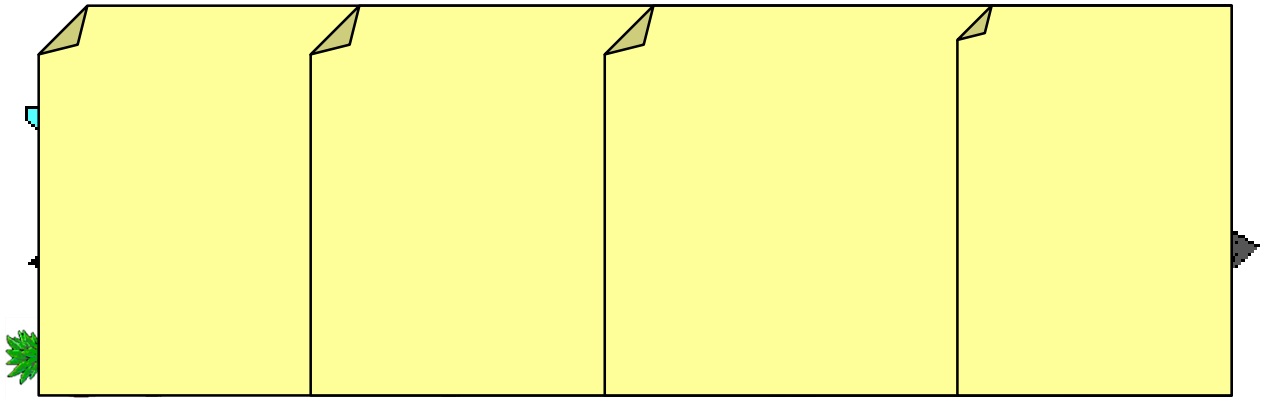
Наиболее простые и часто оптимальные алгоритмы основаны на последовательном просмотре массива ***A*** с проверкой свойства ***P*** на каждом элементе.

Максимальный элемент

Задача: найти в массиве максимальный элемент.

Алгоритм:





Псевдокод:

```
{ считаем, что первый элемент – максимальный }
for i:=2 to N do
  if a[i] > { максимального } then
    { запомнить новый максимальный элемент a[i] }
```

Почему цикл от i=2?

Максимальный элемент

Дополнение: как найти номер максимального элемента?

```
                { считаем, что первый – максимальный }
iMax := 1;
for i:=2 to N do      { проверяем все остальные }
  if a[i] > a[iMax] then { нашли новый максимальный }
  begin
                      { запомнить a[i] }
    iMax := i;        { запомнить i }
  end;
```

? Как упростить?

По номеру элемента `iMax` всегда можно найти его значение `a[iMax]`. Поэтому везде меняем max на `a[iMax]` и убираем переменную `max`.

Максимальный элемент

```
const N = 5;
var a: array [1..N] of integer;
    i, iMax: integer;
begin
  writeln('Исходный массив:');
  for i:=1 to N do begin
    a[i] := random(100) + 50;
    write(a[i]:4);
  end;
  iMax := 1; { считаем, что первый – максимальный }
  for i:=2 to N do      { проверяем все остальные }
    if a[i] > a[iMax] then { новый максимальный }
      iMax := i;              { запомнить i }
  writeln('Максимальный элемент a[', iMax, ']=', a[iMax]);
end.
```

случайные числа в интервале [50,150)

поиск максимального

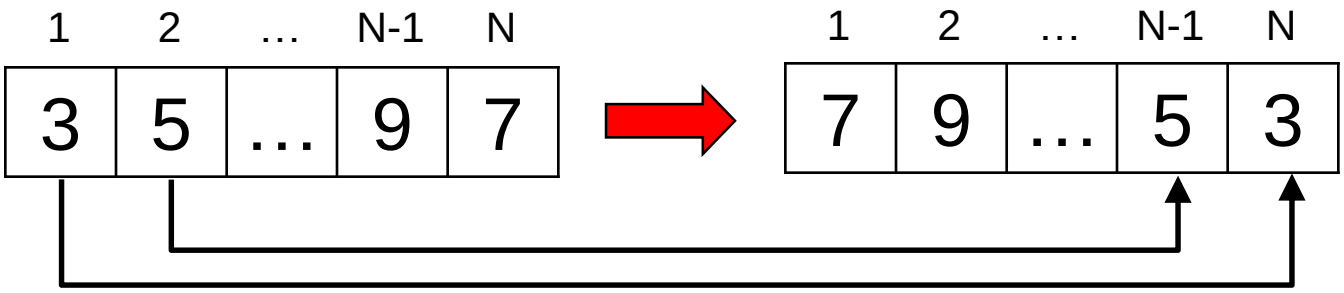
Поиск заданного элемента в массиве

```
Program Search_in_Array;
 Label   1;
 Const   n = 100;
 Var   A : Array[1..n]  of  Real;
    b : Real;   Flag : Boolean;   i : Integer;
 Begin
       Writeln('Введите массив'); {Блок ввода массива}
       For  i:=1 to n do   Read (A[i]);
       Writeln('Введите элемент для поиска');
       Read (b);
       Flag := true;
       For  i:=1 to n  do
         If  A[i] = b then  begin  { прерывание цикла }
    Flag := false;   goto  1   end;
   1: If  Flag  then Writeln('Элемента ', b,
                             ' в массиве нет')
    else Writeln('Элемент ', b, ' стоит на ',
                       i, '-м месте');
 End.
```

Обработка массивов

Реверс массива

Задача: переставить элементы массива в обратном порядке.



Алгоритм:

сумма индексов N+1

поменять местами **A[1]** и **A[N]**, **A[2]** и **A[N-1]**, **…**

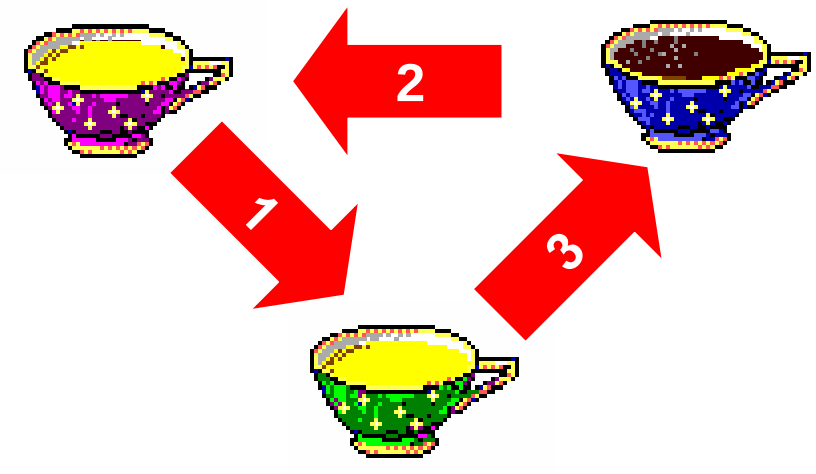
Псевдокод:

```
for i:=1 to N div 2 do
  { поменять местами A[i] и A[N+1-i] }
```

Что неверно?

Как переставить элементы?

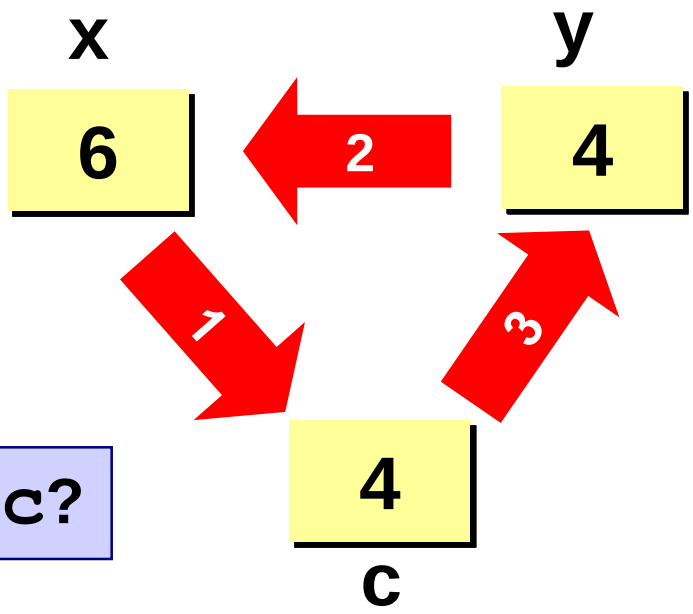
Задача: поменять местами содержимое двух чашек.



Задача: поменять местами содержимое двух ячеек памяти.

```
x := y;
y := x;
```

```
c := x;
x := y;
y := c;
```



Можно ли обойтись без c?

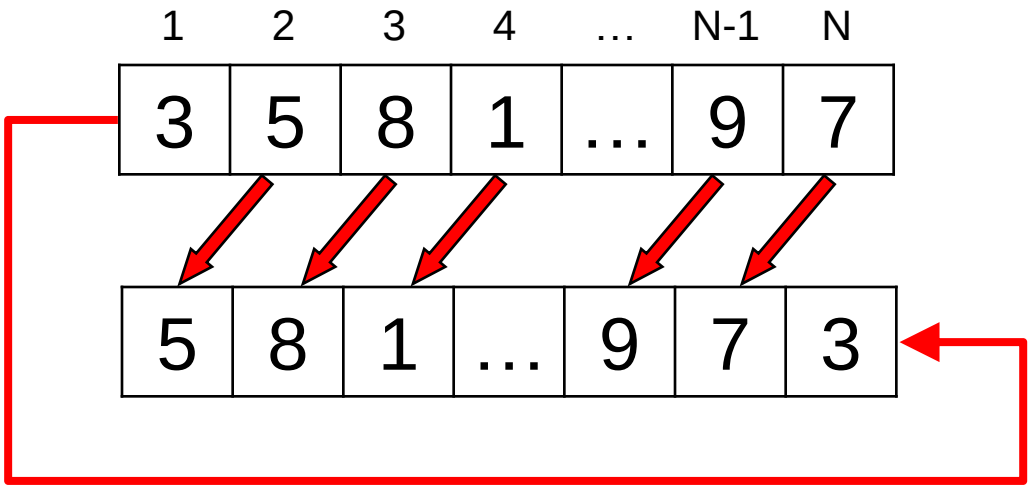
Программа

```
const N = 10;
var A: array[1..N] of integer;
    i, c: integer;
begin
  { заполнить массив }
  { вывести исходный массив }

  for i:=1 to N div 2 do begin
     c:=A[i]; A[i]:=A[N+1-i]; A[N+1-i]:=c;
   end;
end.
```

Циклический сдвиг

Задача: сдвинуть элементы массива влево на 1 ячейку, первый элемент становится на место последнего.



Алгоритм:

```
A[1]:=A[2]; A[2]:=A[3];… A[N-1]:=A[N];
```

Цикл:

почему не **N**?

```
for i:=1 to N-1 do
  A[i]:=A[i+1];
```

Что неверно?

Программа

```
const N = 10;
var A: array[1..N] of integer;
    i, c: integer;
begin
  { заполнить массив }
  { вывести исходный массив }

  c := A[1];
  for i:=1 to N-1 do A[i]:=A[i+1];
  A[N] := c;
end.
```

Способы перебора элементов массивов

Часто при работе с массивами задача ставится так, что требуется все элементы или их часть обработать одинаково. Для такой обработки организуется ***перебор элементов***.

Схему перебора элементов массива можно охарактеризовать:

- направлением перебора;
- количеством одновременно обрабатываемых элементов;
- характером изменения индекса.

По ***направлению перебора*** различают схемы:

- от первого элемента к последнему (от начала массива к концу);
- от последнего элемента к первому (от конца к началу);
- от обоих концов к середине.

В массиве одновременно можно обрабатывать один, два, три и т.д. элемента.

Часто в качестве параметра цикла используется индекс массива.

Обратите внимание также на то обстоятельство, что после изменения индекса его необходимо сразу же проверить на попадание в заданный диапазон, иначе возможны ошибки.

Общие правила организации перебора

В правильно построенной схеме обязательно должны присутствовать:

- блок установки начальных значений индексов массива,
- блок проверки индекса (индекс не должен выходить за границы индексов массива),
- блок изменения индекса для перехода к следующему элементу массива, причем, за блоком изменения индекса по времени выполнения должен располагаться блок проверки индекса на принадлежность интервалу, определенному границами массива.

Если будет нарушено хотя бы одно из перечисленных условий, то в процессе выполнения программы возникнут ошибки.

Случай 1

Перебрать элементы массива по одному, двигаясь от начала массива к концу.

Здесь индекс начального элемента 1, индекс последнего обрабатываемого элемента n, шаг перебора 1. Конечное значение (***кз***) параметра цикла при условии проверки окончания с помощью сравнения **<=** может быть вычислено по формуле:

(конечное значение - начальное значение + 1)/шаг = n.

Отсюда: ***(кз-1+1)/1=n*** или ***кз=n***.

Случай 1

Схема перебора может быть представлена в виде:

```
for  i:=1  to  n  do
    { обработка a[i] }
```

или:

```
i:=1;
while  i<=n  do  begin
    { обработка a[i] }
   i:=i+1
end;
```

Случай 1

Если условие окончания проверяется с помощью сравнения **<**, то конечное значение вычисляется так: ***(конечное значение - начальное значение)/шаг=n***.

Отсюда: ***(кз - 1)/1=n*** или ***кз=n+1***.

Схема перебора может быть представлена в виде:

```
i:=1;
while  i<n+1  do  begin
    { обработка a[i] }
  i:=i+1
end;
```

Случай 2

Перебрать элементы массива по одному, двигаясь от конца массива к началу.

```
for  i:=n  downto  1  do
      { обработка a[i] }
```

или:

```
i:=n;
while  i>=1  do  begin
    { обработка a[i] }
  i:=i-1
end;
```

Случай 3

Обработать массив по одному элементу, двигаясь с обоих концов к середине массива.

```
i:=1;  {установка нижней границы}
j:=n;  {установка верхней границы}
while  i<=j  do  begin
       { обработать элемент a[i] }
    i:=i+1;
       { обработать элемент a[j] }
    j:=j-1;
End;
```

Случай 4

Перебрать элементы массива с четными индексами, двигаясь от начала к концу.

Вариант 1. Здесь индекс начинает изменяться с четного числа, величина шага, равная двум, обеспечивает сохранение четности индекса.

```
i:=2;
while  i<=n  do begin
    { обработка a[i] }
  i:=i+2
end;
```

Вариант 2. Здесь внутрь цикла перебора вложен оператор, проверяющий четность индекса (работает медленнее).

```
for  i:=1  to  n  do
  if   i mod 2=0  then
     { обработка a[i] };
```

Случай 5

Перебрать элементы массива с четными индексами, двигаясь от конца массива к началу.

Вариант 1. Здесь установка начального значения - не простое присваивание, а условный оператор, позволяющий отыскать последний элемент массива с четным индексом.

```
if  n mod 2 =0 then
      i:=n
else  i:=n-1;
while  i>0  do  begin
    { обработка a[i] };
   i:=i-2
end;
```

Случай 5

Вариант 2. Условный оператор, устанавливающий начальное значение индекса, можно внести в тело цикла.

```
i:=n;
while  i>0  do  begin
  if  i mod 2 =0  then
    { обработка a[i] } ;
  i:=i-1
end;
```

Случай 6

Перебрать элементы массива с четным индексом, двигаясь с обоих концов массива к его середине.

Для решения этой задачи соединим схемы перебора, рассмотренные в случаях 4 и 5:

```
i:=2;
if  n mod 2 =0  then j:=n
else  j:=n-1;
while  i<=j  do begin
    { обработать элемент a[i] };
   i:=i+2;
    { обработать элемент a[j] };
   j:=j-2;
end;
```

Случай 7

Перебрать элементы массива с индексом, кратным k, двигаясь от начала массива к его концу.

Для решения этой задачи соединим схемы перебора, рассмотренные в случаях 4 и 5:

```
i:=k;
while  i<=n do begin
    {обработка a[i] }
   i:=i+k
end;
```

Случай 8

Перебрать соседние элементы массива, двигаясь от начала массива к концу (случай двух соседей).

Для массива из 5 элементов нужно последовательно обработать пары:

a[1] и ***a[2]***, ***a[2]*** и ***a[3]***, ***a[3]*** и ***a[4]***, ***a[4]*** и ***a[5]***.

Вариант 1.

```
for  i:=1  to  n-1  do
  { обработать a[i] и a[i+1] };
```

Вариант 2.

```
for  i:=1  to  n-1  do
  { обработать a[i-1] и a[i] };
```

Случай 8

Перебрать соседние элементы массива, двигаясь от начала массива к концу (случай трех соседей).

Для массива из 5 элементов нужно последовательно обработать пары:

a[1]*-*a[2]*-*a[3], ***a[2]*-*a[3]*-*a[4]***, ***a[3]*-*a[4]*-*a[5]***.

Вариант 1.

```
for i:=1 to n-2  do
   { обработать a[i] - a[i+1] - a[i+2] };
```

Вариант 2.

```
for i:=2 to n-1  do
   { обработать a[i-1] - a[i] - a[i+1] };
```

Вариант 3.

```
for i:=3 to n  do
   { обработать a[i-2] - a[i-1] - a[i] };
```