



СибГУТИ
Кафедра вычислительных систем

Дисциплины
"ЯЗЫКИ ПРОГРАММИРОВАНИЯ"
"ПРОГРАММИРОВАНИЕ"

Массивы

Преподаватель:

Перышкова Евгения Николаевна



Численное вычисление e^x (аккумуляция результата)

$$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

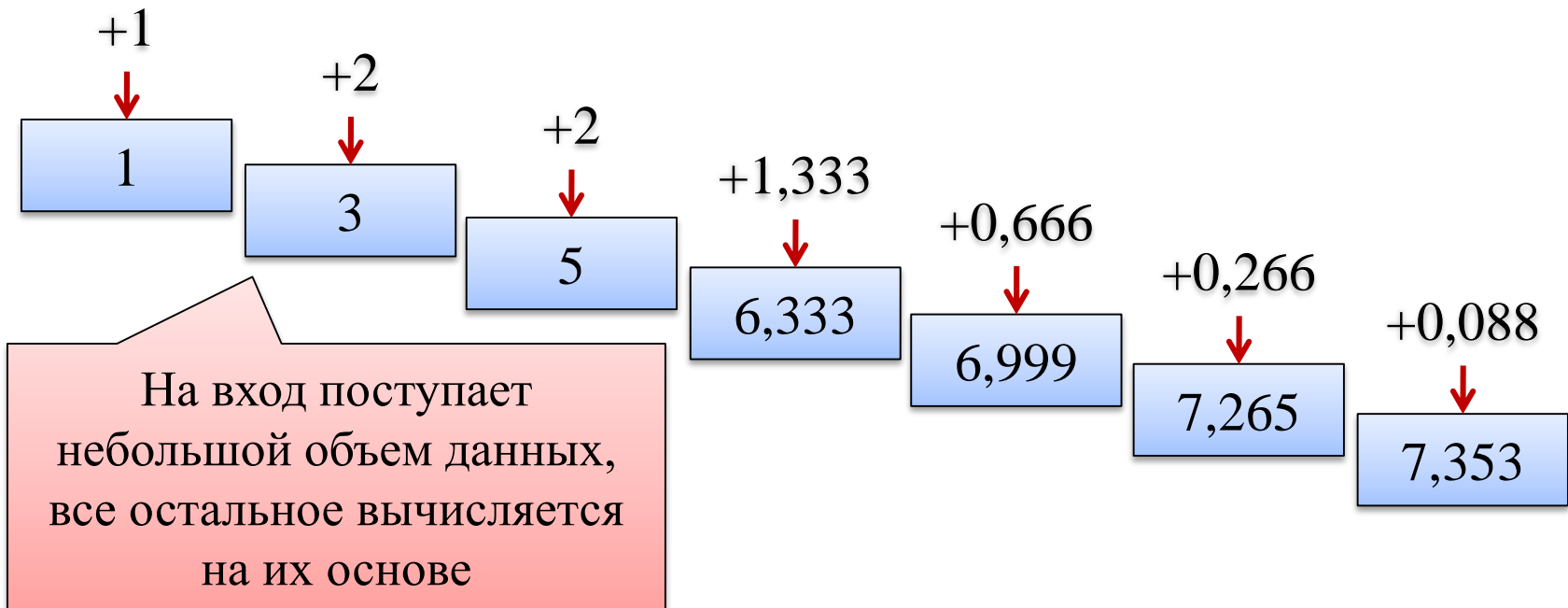
$$e^2 = 7,389056099$$

$$1 + \frac{2}{1} + \frac{4}{1 \cdot 2} + \frac{8}{1 \cdot 2 \cdot 3} + \frac{16}{1 \cdot 2 \cdot 3 \cdot 4} + \frac{32}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} + \frac{64}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} + \dots =$$
$$1 + 2 + 2 + 1,3(3) + 0,6(6) + 0,26(6) + 0,08(8) + \dots =$$
$$7,355555554 + \dots$$



Численное вычисление e^x (аккумуляция результата) (2)

$$1 + \frac{2}{1} + \frac{4}{1 \cdot 2} + \frac{8}{1 \cdot 2 \cdot 3} + \frac{16}{1 \cdot 2 \cdot 3 \cdot 4} + \frac{32}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} + \frac{64}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} + \dots =$$
$$1 + 2 + 2 + 1,3(3) + 0,6(6) + 0,26(6) + 0,08(8) + \dots =$$
$$7,355555554 + \dots$$





Поиск максимального элемента ("потокковая" обработка данных)

... 90 10 18 64 15 38 40 91 | 18 | ...

↓
18

...

... 90 10 18 64 | 15 | 38 40 91 18 ...

↓
15

...

... 90 | 10 | 18 64 15 38 40 91 18 ...

↓
10

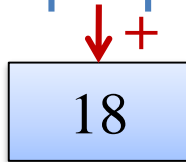
Данные, как поток
"протекают" через сито
(программу), которое
задерживает нужную
информацию

Для реализации
алгоритмов,
использующих такой
подход требуется
ограниченный набор
переменных

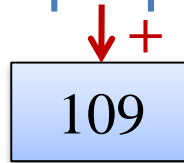


Вычисление суммы входных чисел ("поточковая" обработка + аккумуляция результата)

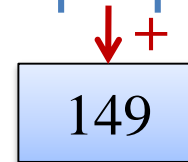
... 38 40 91 | 18 | ...



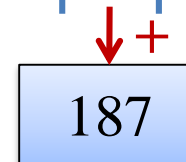
... 38 40 | 91 | 18 ...



... 38 | 40 | 91 18 ...



... | 38 | 40 91 18 ...



На вход подается
последовательность,
размер которой **не
известен** на этапе
создания программы!

Для ее обработки
достаточно однократно
учесть вклад каждого
элемента в общую сумму.



Задача сортировки (sorting problem)

Дано: последовательность из n чисел $\langle a_1, a_2, a_3, \dots, a_n \rangle$

Необходимо: переставить элементы последовательности так, чтобы для любых элементов новой последовательности $\langle a'_1, a'_2, a'_3, \dots, a'_n \rangle$ выполнялось соотношение:

$$a'_1 \leq a'_2 \leq a'_3 \leq \dots \leq a'_n$$

Алгоритм сортировки выбором

5	3	8	9	4
---	---	---	---	---

5	3	8	4	9
---	---	---	---	---

5	3	4	8	9
---	---	---	---	---

4	3	5	8	9
---	---	---	---	---

3	4	5	8	9
---	---	---	---	---

3	4	5	8	9
---	---	---	---	---



Операции над векторами (vector operations)

Вектором a n -мерного пространства называется кортеж из n элементов $a = (a_1, a_2, a_3, \dots, a_n)$, описывающих координаты конечной точки вектора, считая, что его начало расположено в точке $(0, 0, 0, \dots, 0)$.

Для векторов определены:

1) операция сложения $a + b$:

$$a + b = (a_1, a_2, a_3, \dots, a_n) + (b_1, b_2, b_3, \dots, b_n) = \\ (a_1 + b_1, a_2 + b_2, a_3 + b_3, \dots, a_n + b_n)$$

2) операция умножения на скаляр:

$$c \cdot a = c \cdot (a_1, a_2, a_3, \dots, a_n) = (c \cdot a_1, c \cdot a_2, c \cdot a_3, \dots, c \cdot a_n)$$

3) скалярное произведение (a, b) векторов:

$$(a, b) = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 + \dots + a_n \cdot b_n$$

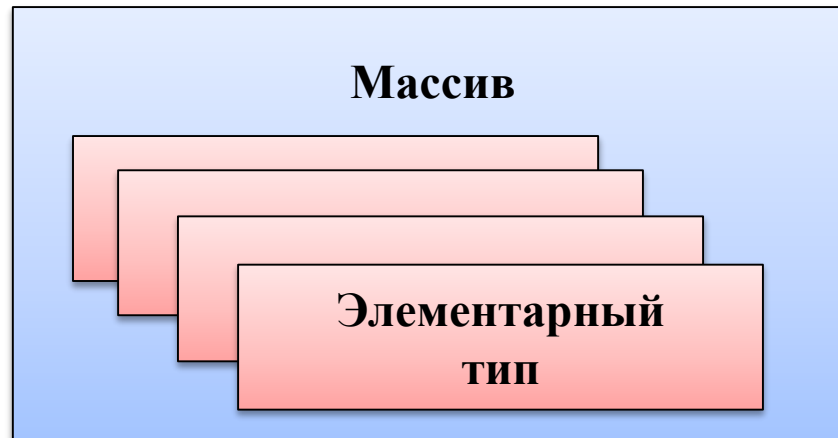
В результате выполнения любой из операций формируется кортеж из n переменных

Необходима возможность обрабатывать элементы в цикле



Требования к типу данных для хранения последовательностей и векторов

1. Хранение заданного количества n однотипных элементов под одним именем.
2. Механизм доступа к отдельным элементам, допускающий возможность автоматизации с применением циклических конструкций.





Массив данных

- Массивы данных в языках программирования высокого уровня – это **сложные типы данных**, предназначенные для хранения **наборов однотипных данных** в памяти программы.
- Доступ к конкретному элементу производится с использованием его **целочисленного индекса**.
- Индексы задают порядок следования элементов в массиве, поэтому его можно рассматривать как упорядоченное множество.
- Технически это последовательность **однотипных переменных**.
- В памяти элементы массива располагаются друг за другом непрерывно. Для каждого элемента i справедливо, что в памяти между ним и элементами с индексами $(i - 1)$ и $(i + 1)$ (если они существуют) не может находиться никаких ячеек данных.



Объявление массива

При объявлении массива компилятору необходимо предоставить следующую информацию:

1. Элементарный тип – тип данного элементов, составляющих массив.
2. Имя массива.
3. Количество элементов в массиве.
4. [*Необязательно*] начальное содержимое массива (инициализация).

Пример – массив с именем m из N элементов типа `int`:

```
int m[N];
```



Инициализация массива

При объявлении массива компилятору необходимо предоставить следующую информацию:

1. Элементарный тип – тип данного элементов, составляющих массив.
2. Имя массива.
3. Количество элементов в массиве (**может отсутствовать, если массив инициализирован**).
4. Начальное содержимое массива (инициализация).

Например: массив с именем *mas* из 5 элементов типа `float`, со значениями 1.1, 2.0, 3.5, 6.7, 8.1 объявляется так:

```
int mas[5] = {1.1, 2.0, 3.5, 6.7, 8.1};
```

или

```
int mas[] = {1.1, 2.0, 3.5, 6.7, 8.1};
```



Операция индексации (array subscripting)

- Основная часть операций над массивом производится поэлементно.
- Доступ к конкретному элементу производится с использованием операции индексации – []:

<имя-массива>[<индекс>].

Например, для массива:

```
float mas[] = { 0 1 2 3 4  
                1.1 2.0 3.5 6.7 8.1 }
```

доступ к элементу со значением 3.5, расположенному на 3-ей позиции в массиве осуществляется следующим образом:

```
x = mas[2];
```

Индексация элементов в языке СИ начинается с **НУЛЯ!**



Индексы

Важным достоинством операции индексации является то, что в качестве ее аргумента помимо констант:

```
int a[10]
a[0], a[1], a[2], a[3], a[4], ...,
```

можно использовать также выражения:

```
i = 0, j = 5, k = 2
a[i], a[j + k], a[i++], a[j - k]
```

Данное свойство дает массивам кардинальное преимущество перед простыми переменными, так как доступ к их элементам можно автоматизировать при помощи циклов



Индексы (2)

- В качестве индекса может использоваться **любое выражение, целого типа**: char, short, int, long.
- Индексы элементов массива языка СИ **начинаются с 0**.
- Индексы могут быть:
 - положительными, тогда обращение производится к ячейке, располагающейся **после первой**.
 - отрицательными, тогда обращение производится к ячейке, располагающейся **перед первой**.

...	a[-2]	a[-1]	a[0]	a[1]	a[2]	...
-----	-------	-------	------	------	------	-----

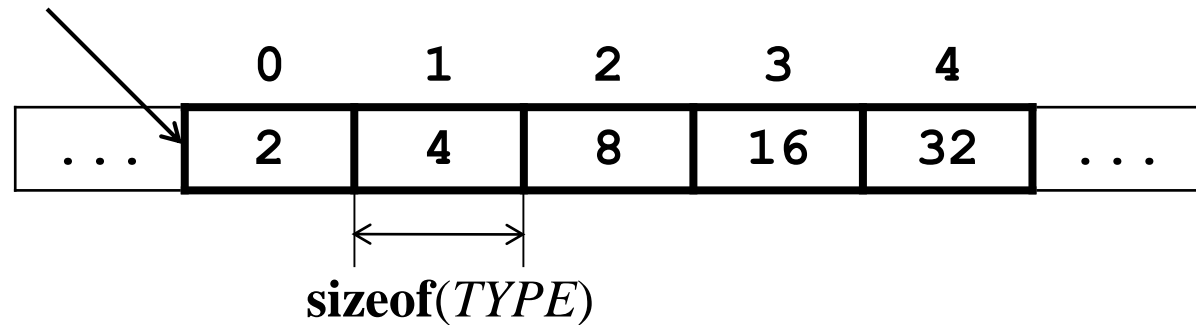


Принцип работы операции индексации

```
int a[] = {2, 4, 8, 16, 32}
```

a = 0xA000

Память
программы



Имя массива – указатель константа на первый элемент массива. В примере на рисунке адрес первой ячейки со значением "2" – 0xA000 (0x – признак шестнадцатеричной константы).

Зная адрес $\text{addr}(a)$ первого элемента, размер элементов и учитывая свойство непрерывного расположения элементов массива, можно определить адрес любого элемента по формуле:

$$\text{addr}(a[i]) = \text{addr}(a) + i \cdot \text{sizeof}(TYPE)$$



Обработка массивов

К массиву как единому целому может быть применена только операция индексации [] и оператор sizeof.

Любые другие действия требуют **поэлементной обработки!**

Например:

- ввод-вывод массива;
- присваивание массивов;
- сложение массивов по правилу векторов;
- поиск минимального/максимального элемента;
- сортировка массива;
- т.д.

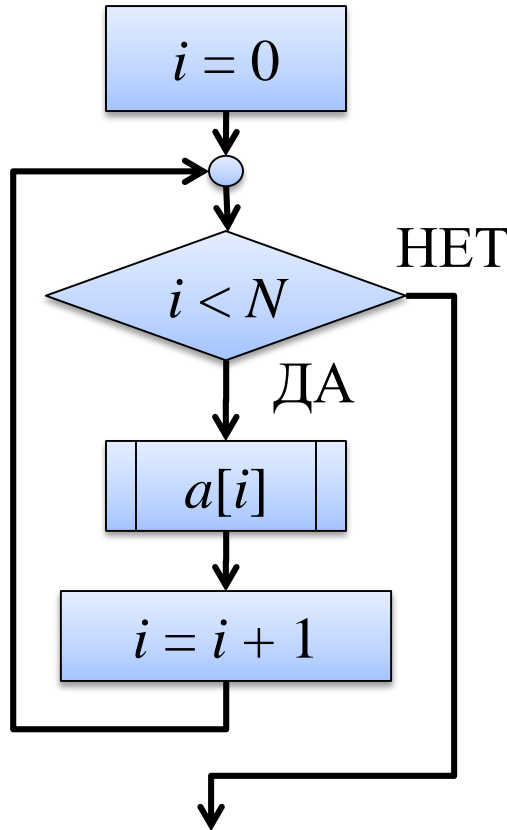


Обработка массивов (ЦИКЛЫ)

Большинство операций над массивами осуществляются **ПОЭЛЕМЕНТНО**.

Такой тип обработки предусматривает перебор всех допустимых индексов массива и применение однотипной операции к каждому элементу. Данная операция наиболее эффективно реализуется с применением циклов:

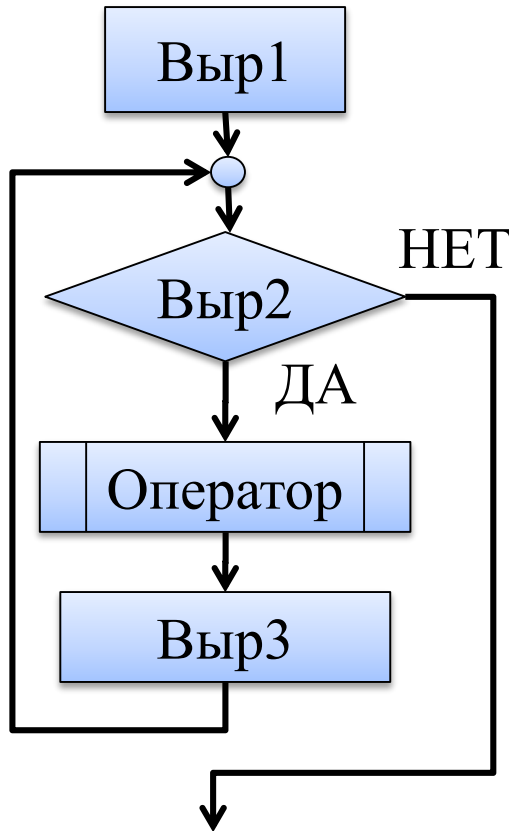
```
int a[10] = {1, 2, 3, 4, 5, 6};  
int i = 0;  
while( i < 6 ) {  
    обработать a[i];  
    i = i + 1;  
}
```





Цикл for

```
"for" "(" [Выр1] ";" [Выр2] ";" [Выр3] ")"  
Оператор.
```



Выр1 – выражение-инициализация. Выполняется **один раз** перед циклом.

Выр2 – логическое выражение. Условие продолжения цикла (аналогично циклу while).

Выр3 – выражение-последствие. Выполняется **после каждой** итерации.

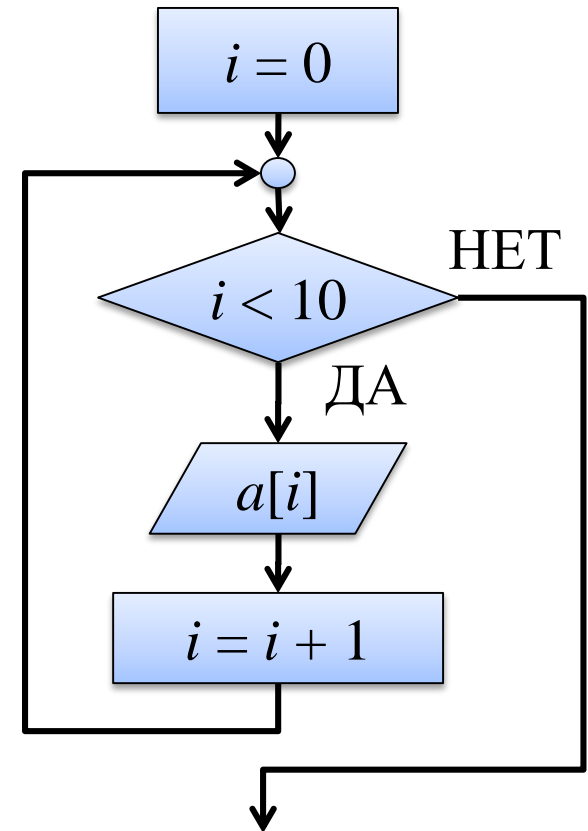
Оператор выполняется до тех пор, пока **Выр2** является ИСТИННЫМ.



Ввод-вывод массивов

- В функции `scanf` не предусмотрено спецификатора для ввода или вывода массива как единого целого. Это сделано из соображений универсальности и сохранения относительной простоты использования данной функции.
- Ввод-вывод массивов осуществляется поэлементно. Пример ввода:

```
int mas[10], i;  
for(i=0;i<10;i++){  
    scanf("%d",&mas[i]);  
}
```

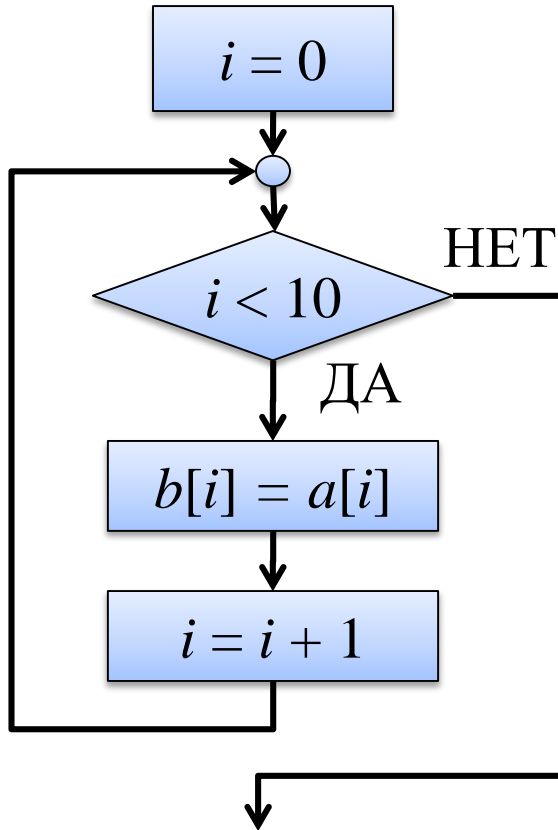




Присваивание массивов

Для того, чтобы присвоить одному массиву значение другого необходимо, чтобы были выполнены следующие условия:

1. Совместимость типов данных элементов (как и для переменных).
2. Совместимость по размеру (массив-приемник должен быть не меньше массива-источника).



```
int a[10], b[10], i;  
  
... // ввод a  
for(i=0;i<10;i++){  
    b[i] = a[i];  
}
```

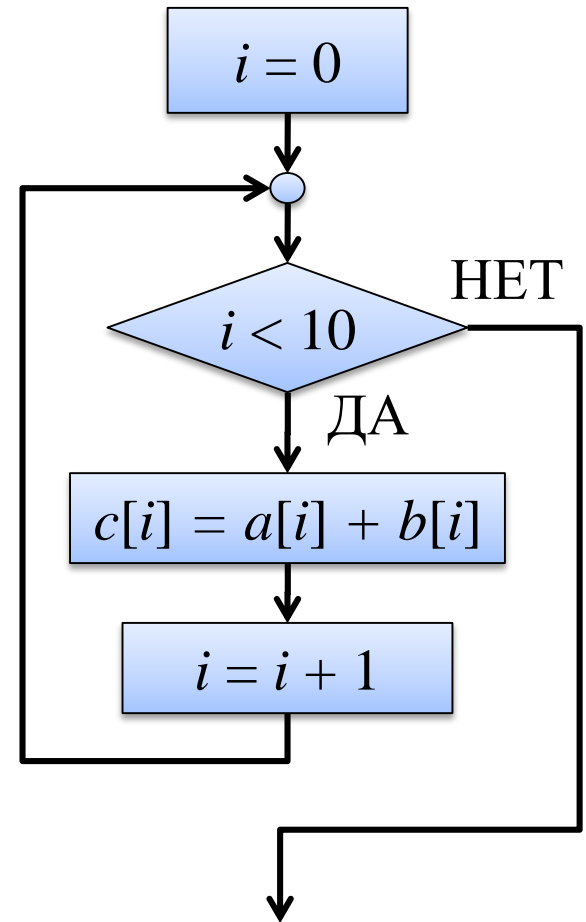


Сложение массивов по правилу векторов

Для выполнения операции сложения двух массивов требуется выполнение следующих условий:

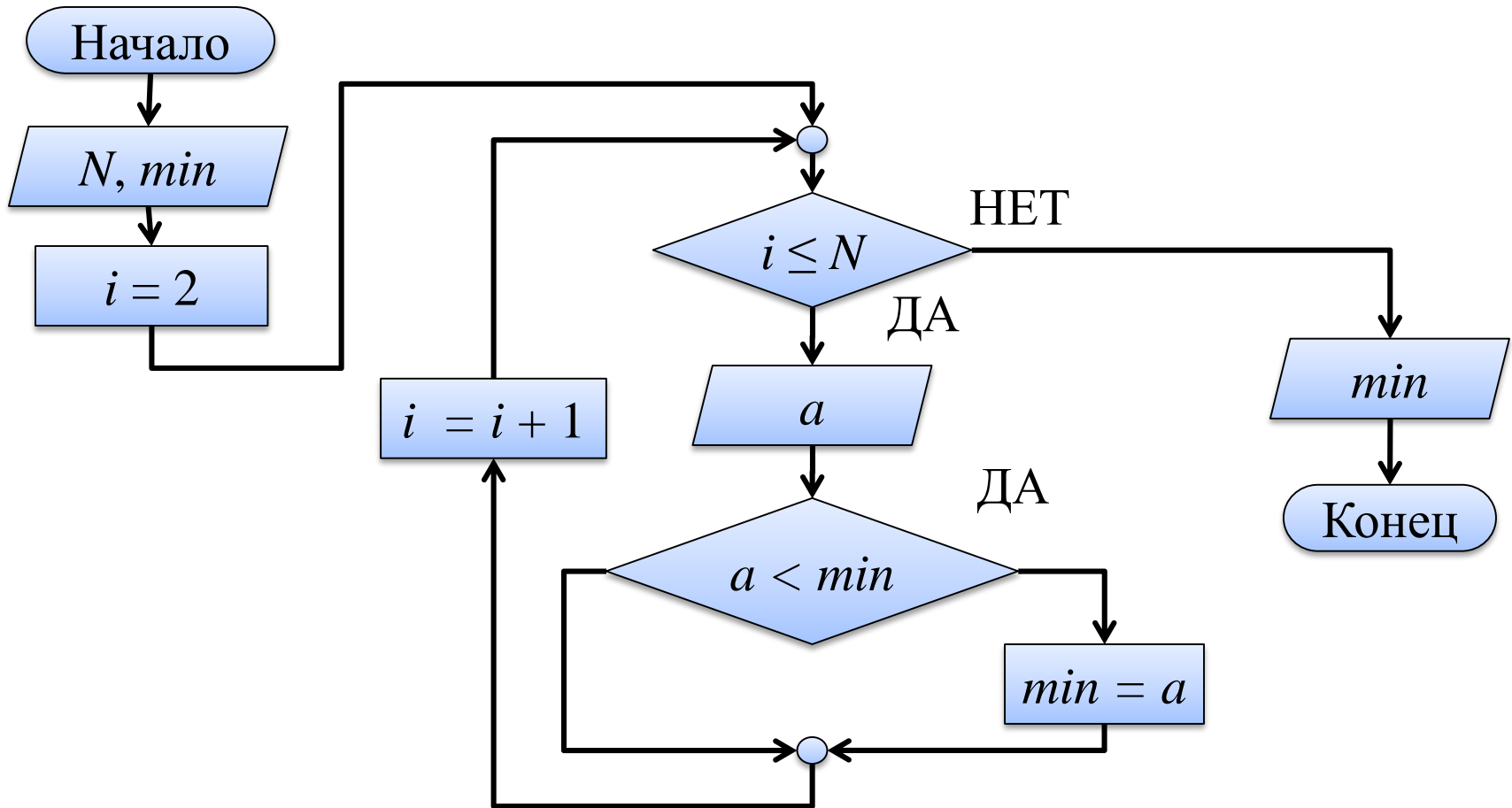
1. Совместимость типов данных элементов (как и для переменных).
2. Эквивалентность по размеру (массив-приемник и массивы-источники должны иметь одинаковый размер).

```
int a[10], b[10], c[10] , i;  
... // ввод a и b  
for(i=0;i<10;i++){  
    c[i] = a[i] + b[i]  
}
```





Поиск минимального элемента в последовательности [потоковый]





Поиск минимального элемента

Алгоритм поиска минимального элемента в массиве аналогичен рассмотренному ранее "потокковому" алгоритму поиска минимального элемента последовательности.

Важным отличием является то, что в массиве основной интерес представляет **индекс** искомого элемента, а не его значение. Информация о расположении элемента в массиве позволяет определить его значение, но не наоборот!

```
input  $n, a$   
 $imin \leftarrow 0$   
for  $i \leftarrow 1$  to  $n$  do  
    if  $a[imin] > a[i]$  then  
         $imin \leftarrow i$   
output  $imin, a[imin]$ 
```



Особенности инициализации массива

- Статические массивы можно объявлять с инициализацией, перечисляя значения их элементов в {} через запятую. **Если задано меньше элементов, чем длина массива остальные элементы считаются нулями:**

```
int a[10] = { 1, 2, 3, 4 }; /* и 6 нулей */
```

- Если при описании массива с инициализацией не указать его размер, он будет подсчитан компилятором:

```
int b[] = { 1, 2, 3 };
```




Особенности реализации массивов в языке СИ

- «СИ — инструмент, острый, как бритва: с его помощью можно создать и элегантную программу, и кровавое месиво»,
Б. Керниган.
- В связи со сравнительно низким уровнем языка многие случаи неправильного использования опасных элементов не обнаруживаются и не могут быть обнаружены ни при компиляции, ни во время исполнения.
- В СИ **не предусмотрено средств проверки индексов** массивов (проверки выхода за границы массива).
- Например, возможна запись в шестой элемент массива из пяти элементов, что, естественно, приведёт к непредсказуемым результатам.



Демонстрация выхода за границы массива

```
#include <stdio.h>
int main()
{
    int i, a[4] = {1,2,3,4}, b[] = {5,6,7,8};
    for(i=0;i<=4;i++)
        b[i] = (i+1)*(i+1);
    printf("a: ");
    for(i=0;i<4;i++)
        printf("%d ", a[i]);
    printf(" b: ");
    for(i=0;i<4;i++)
        printf("%d ", b[i]);
    printf("\n");
}
```

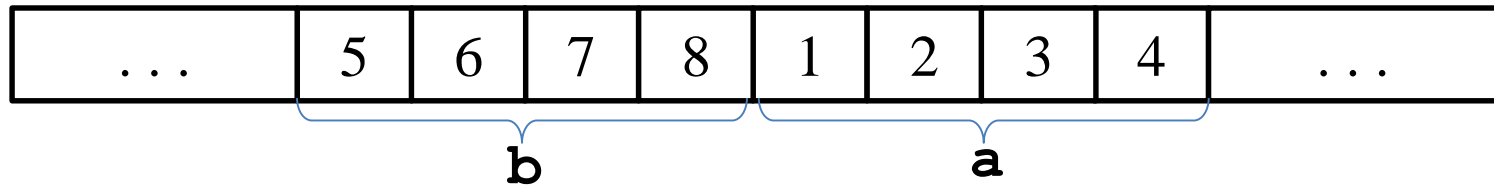
```
$ ./array_borders
```

```
a: 25 2 3 4 b: 1 4 9 16
```



Выход за границы массива

```
int i, a[4] = {1,2,3,4}, b[4] = {5,6,7,8};
```



```
for (i=0; i<=4; i++) {  
    b[i] = (i+1)*(i+1);  
}
```

