Vector algorithms - Arrays

Basics of Programming 1

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

27 September, 2023

Content

Chapter 1

Type conversion

What is that?

In some cases the C-program needs to convert the type of our expressions.

```
1  long func(float f) {
2    return f;
3  }
4
5  int main(void) {
6    int i = 2;
7    short s = func(i);
8    return 0;
9  }
```

In this example: `int` → `float` → `long` → `short`

- `int` → `float` rounding, if the number is large
- `float` → `long` may cause overflow, rounding to integer
- `long` → `short` may cause overflow

Converting types

- Basic principle

Converting types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Basic principle
  - preserve the value, if possible

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

- Conversion with one operand (we have seen that)

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

- Conversion with one operand (we have seen that)
  - at assignment of value

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

- Conversion with one operand (we have seen that)
  - at assignment of value
  - at calling a function (when actualising the formal parameters)

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

- Conversion with one operand (we have seen that)
  - at assignment of value
  - at calling a function (when actualising the formal parameters)
- Conversion with two operands (eg. `2/3.4`)

Converting types

- Basic principle
  - preserve the value, if possible
- In case of overflow
  - the result is theoretically undefined

- Conversion with one operand (we have seen that)
  - at assignment of value
  - at calling a function (when actualising the formal parameters)
- Conversion with two operands (eg. `2/3.4`)
  - evaluating an operation

Conversion with two operands

The conversion of the two operands to the same, common type happens according to these rules

operand one	the other operand	common, new type
`long double`	anything	`long double`
`double`	anything	`double`
`float`	anything	`float`
`unsigned long`	anything	`unsigned long`
`long`	anything (`int`, `unsigned`)	`long`
`unsigned`	anything (`int`)	`unsigned`
`int`	anything (`int`)	`int`

Type conversions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Example for conversion

```
1 int a = 3;
2 double b = 2.4;
3 a = a*b;
```

Type conversions

Example for conversion

```
1 int a = 3;
2 double b = 2.4;
3 a = a*b;
```

1. $3 \rightarrow 3.0$

Type conversions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Example for conversion

```
1 int a = 3;
2 double b = 2.4;
3 a = a*b;
```

1. $3 \rightarrow 3.0$
2. $3.0 * 2.4 \rightarrow 7.2$

Type conversions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Example for conversion

```
1 int a = 3;
2 double b = 2.4;
3 a = a*b;
```

1. $3 \rightarrow 3.0$
2. $3.0 * 2.4 \rightarrow 7.2$
3. $7.2 \rightarrow 7$

Chapter 2

Sequential processing

Data vector arriving sequentially

- There are 2 options for determining the number of elements

Data vector arriving sequentially

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- There are 2 options for determining the number of elements
  1. First we read the number of elements, and after it we read the data elements

4	renault	opel	kia	fiat

Data vector arriving sequentially

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- There are 2 options for determining the number of elements
  1. First we read the number of elements, and after it we read the data elements

4	renault	opel	kia	fiat

  2. We use a loop to read and process the data elements, until we don't receive a previously specified, special (different from all other) data element

renault	opel	kia	fiat	end

Data vector arriving sequentially

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- There are 2 options for determining the number of elements
  1. First we read the number of elements, and after it we read the data elements

4	renault	opel	kia	fiat

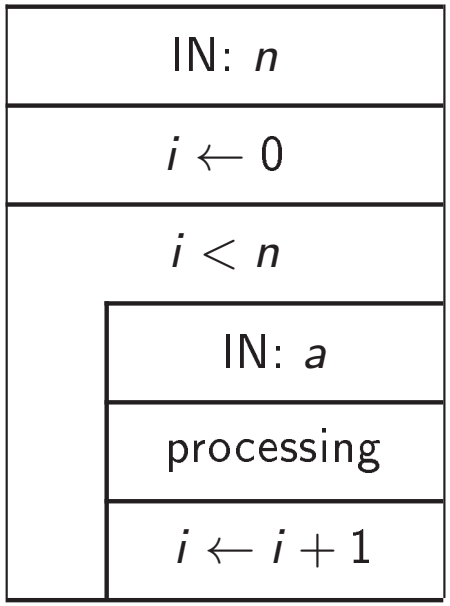
  2. We use a loop to read and process the data elements, until we don't receive a previously specified, special (different from all other) data element

renault	opel	kia	fiat	end

     This is called a series with termination or series with termination symbol

Processing a data vector

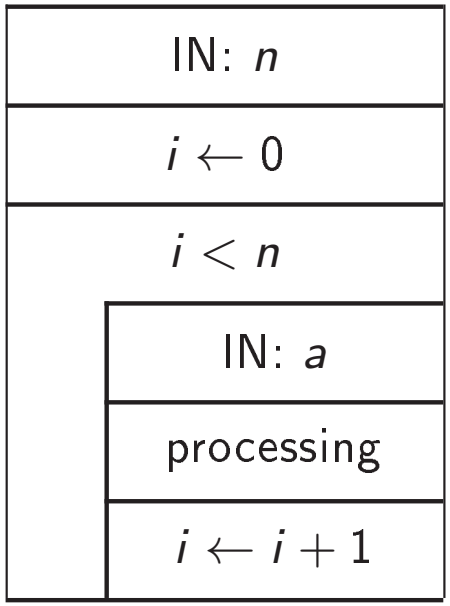
Vector with known size



- Notations:
  - n: number of data elements
  - a: data read
  - i: loop counter

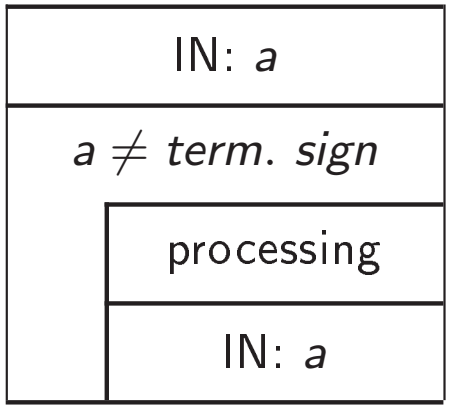
Processing a data vector

Vector with known size



- Notations:
  - n: number of data elements
  - a: data read
  - i: loop counter

Vector with termination



- Notations
  - a: data read (scanned)

A little remark

- We will learn these later in details, but until that...

A few types in C

`int` Type for storing integer values,
read (scan) and print with `%d` format code

`double` Type for storing real numbers,
read (scan) with `%lf`, print with `%f` format code

`char` Type for storing text characters read (scan) and printf with `%c` format code

A little remark

- We will learn these later in details, but until that...

A few types in C

`int` Type for storing integer values, read (scan) and print with `%d` format code

`double` Type for storing real numbers, read (scan) with `%lf`, print with `%f` format code

`char` Type for storing text characters read (scan) and printf with `%c` format code

A few operators in C

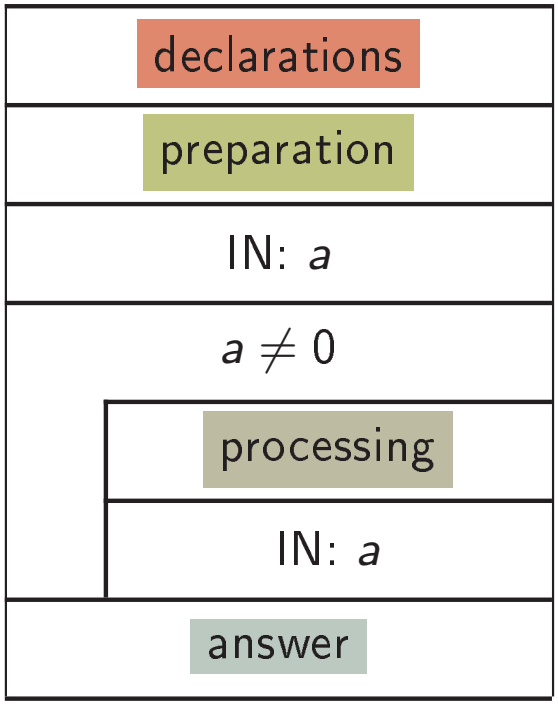
`==` (equal to) checking equality

`!=` (not equal to) checking difference

`&&` (logical AND) conjunction

A framework for processing a vector with termination

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- We have to figure out only the coloured parts, the rest is always the same

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    /* declarations */
7    /* preparation */
8    scanf("%d", &a);
9    while (a != 0)
10   {
11     /* processing */
12     scanf("%d", &a);
13   }
14   /* answer */
15   return 0;
16 }
```

Sum of elements

declarations

We create a variable for storing the sum.

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations */
  /* preparation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Sum of elements

declarations

We create a variable for storing the sum.

preparation

At the beginning we set it to 0.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int sum;
  /* preparation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Sum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the sum.

preparation

At the beginning we set it to 0.

processing

We increase it with the read (scanned) data.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int sum;
  sum = 0;
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Sum of elements

declarations

We create a variable for storing the sum.

preparation

At the beginning we set it to 0.

processing

We increase it with the read (scanned) data.

answer

We print the result.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int sum;
  sum = 0;
  scanf("%d", &a);
  while (a != 0)
  {
    sum = sum + a;
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Sum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the sum.

preparation

At the beginning we set it to 0.

processing

We increase it with the read (scanned) data.

answer

We print the result.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int sum;
  sum = 0;
  scanf("%d", &a);
  while (a != 0)
  {
    sum = sum + a;
    scanf("%d", &a);
  }
  printf("%d", sum);
  return 0;
}
```

link

Arithmetic product of elements

declarations

We create a variable for storing the product.

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations */
  /* prepearation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Arithmetic product of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the product.

preparation

At the beginning we set it to 1.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int prod;
  /* prepearation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Arithmetic product of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the product.

preparation

At the beginning we set it to 1.

processing

We multipy it with the read (scanned) data.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int prod;
  prod = 1;
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Arithmetic product of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the product.

preparation

At the beginning we set it to 1.

processing

We multipy it with the read (scanned) data.

answer

We print the result.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int prod;
  prod = 1;
  scanf("%d", &a);
  while (a != 0)
  {
    prod = prod * a;
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Arithmetic product of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the product.

preparation

At the beginning we set it to 1.

processing

We multipy it with the read (scanned) data.

answer

We print the result.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int prod;
  prod = 1;
  scanf("%d", &a);
  while (a != 0)
  {
    prod = prod * a;
    scanf("%d", &a);
  }
  printf("%d", prod);
  return 0;
}
```

link

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's determine the average of the elements!
  - We have to remember the sum and the number of elements all the time.
  - Both are 0 at the beginning.
  - In every cycle we have to increase the sum with the read (scanned) data, and increase the number of elements by 1.
  - Finally, we print out the quotient of the sum and the number (divide sum by the number of elements).

Average of elements

- Let's determine the average of the elements!
  - We have to remember the sum and the number of elements all the time.
  - Both are 0 at the beginning.
  - In every cycle we have to increase the sum with the read (scanned) data, and increase the number of elements by 1.
  - Finally, we print out the quotient of the sum and the number (divide sum by the number of elements).
- Warning! In C language
  - 8/3=2 (integer division)
  - `8.0/3.0 = 8.0/3 = 8/3.0 = 2.6666...` (real division)
  - for this reason is better to store the sum as a real number

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create two variables for storing the sum and the number of elements.

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations
                  */
  /* preparation
                  */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                    */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create two variables for storing the sum and the number of elements.

preparation

We set both sum and number to 0.

```c
#include <stdio.h>

int main(void)
{
  int a;
  double sum;
  int n;
  /* preparation
                  */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                    */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create two variables for storing the sum and the number of elements.

preparation

We set both sum and number to 0.

processing

We increase the sum with the read (scanned) data, and increase the number of elements by 1.

```c
#include <stdio.h>

int main(void)
{
  int a;
  double sum;
  int n;
  sum = 0.0;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                    */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create two variables for storing the sum and the number of elements.

preparation

We set both sum and number to 0.

processing

We increase the sum with the read (scanned) data, and increase the number of elements by 1.

answer

We print the quotient of the sum and the number.

```c
#include <stdio.h>

int main(void)
{
  int a;
  double sum;
  int n;
  sum = 0.0;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    sum = sum + a;
    n = n+1;
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Average of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create two variables for storing the sum and the number of elements.

preparation

We set both sum and number to 0.

processing

We increase the sum with the read (scanned) data, and increase the number of elements by 1.

answer

We print the quotient of the sum and the number.

```c
#include <stdio.h>

int main(void)
{
  int a;
  double sum;
  int n;
  sum = 0.0;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    sum = sum + a;
    n = n+1;
    scanf("%d", &a);
  }
  printf("%f", sum/n);
  return 0;
}
```

link

Counting

- Let's count the number of elements that satisfy a given condition!
  - We have to remember the number of the appropriate elements,
  - that is 0 at the beginning,
  - and it is increased by 1, if another appropriate element arrives (logical test).
  - Finally, we print out the count (number of elements).
- As an example, let's count the numbers that have 2 digits!

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's count the number of elements that satisfy a given condition!
  - We have to remember the number of the appropriate elements,
  - that is 0 at the beginning,
  - and it is increased by 1, if another appropriate element arrives (logical test).
  - Finally, we print out the count (number of elements).
- As an example, let's count the numbers that have 2 digits!
- The right condition is:

```
1  a >= 10 && a <= 99 /* && : logical AND */
```

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the count (nr. of elements).

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations */
  /* preparation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                  */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the count (nr. of elements).

preparation

At the beginning we set it to 0.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int n;
  /* preparation */
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                  */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the count (nr. of elements).

preparation

At the beginning we set it to 0.

processing

If the element has 2 digits, we increase the count (nr. of elements) by 1.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int n;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    /* processing
                    */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the count (nr. of elements).

preparation

At the beginning we set it to 0.

processing

If the element has 2 digits, we increase the count (nr. of elements) by 1.

answer

We print the count.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int n;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    if (a>=10 && a<=99)
      n = n+1;
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Counting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the count (nr. of elements).

preparation

At the beginning we set it to 0.

processing

If the element has 2 digits, we increase the count (nr. of elements) by 1.

answer

We print the count.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int n;
  n=0;
  scanf("%d", &a);
  while (a != 0)
  {
    if (a>=10 && a<=99)
      n = n+1;
    scanf("%d", &a);
  }
  printf("%d", n);
  return 0;
}
```

link

Determining the minimum

Let's determine the minimum of the elements!

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- Let's set it to 5000 (there surely won't be any larger than that)!

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- ~~Let's set it to 5000 (there surely won't be any larger than that)!~~

  We can only do this if it is given in the specification!

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- ~~Let's set it to 5000 (there surely won't be any larger than that)!~~

  We can only do this if it is given in the specification!

It is better to modify the structure:

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- ~~Let's set it to 5000 (there surely won't be any larger than that)!~~

  We can only do this if it is given in the specification!

It is better to modify the structure:

- At first we read (scan) the first element, and we initialize the minimum value with it.

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- ~~Let's set it to 5000 (there surely won't be any larger than that)!~~

  We can only do this if it is given in the specification!

It is better to modify the structure:

- At first we read (scan) the first element, and we initialize the minimum value with it.
- If the next data element is smaller than the minimum, we rewrite the minimum to this new value

Determining the minimum

Let's determine the minimum of the elements!

- We have to remember the minimum all the time
- ~~Let's set it to 5000 (there surely won't be any larger than that)!~~

  We can only do this if it is given in the specification!

It is better to modify the structure:

- At first we read (scan) the first element, and we initialize the minimum value with it.
- If the next data element is smaller than the minimum, we rewrite the minimum to this new value
- Finally, we print the minimum

Minimum of elements

declarations

We create a variable for storing the minimum.

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations */
  scanf("%d", &a);
  /* preparation */
  while (a != 0)
  {
    /* processing
                  */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Minimum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the minimum.

preparation

it is after the first `scanf` now!

At the beginning we set it to the value of the first element.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    int min;
7    scanf("%d", &a);
8    /* preparation */
9    while (a != 0)
10   {
11     /* processing
12                    */
13     scanf("%d", &a);
14   }
15   /* answer */
16   return 0;
17 }
```

link

Minimum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the minimum.

preparation

it is after the first `scanf` now!

At the beginning we set it to the value of the first element.

processing

If new element is smaller than min, min $\leftarrow$ element.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    int min;
7    scanf("%d", &a);
8    min=a;
9    while (a != 0)
10   {
11     /* processing
12                    */
13     scanf("%d", &a);
14   }
15   /* answer */
16   return 0;
17 }
```

link

Minimum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the minimum.

preparation

it is after the first `scanf` now!

At the beginning we set it to the value of the first element.

processing

If new element is smaller than min, min $\leftarrow$ element.

answer

We print the minimum.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    int min;
7    scanf("%d", &a);
8    min=a;
9    while (a != 0)
10   {
11     if (a < min)
12       min = a;
13     scanf("%d", &a);
14   }
15   /* answer */
16   return 0;
17 }
```

link

Minimum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the minimum.

preparation

it is after the first `scanf` now!

At the beginning we set it to the value of the first element.

processing

If new element is smaller than min, min $\leftarrow$ element.

answer

We print the minimum.

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int a;
6    int min;
7    scanf("%d", &a);
8    min=a;
9    while (a != 0)
10   {
11     if (a < min)
12       min = a;
13     scanf("%d", &a);
14   }
15   printf("%d", min);
16   return 0;
17 }
```

link

Maximum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the maximum.

```c
#include <stdio.h>

int main(void)
{
  int a;
  /* declarations */
  scanf("%d", &a);
  /* preparation */
  while (a != 0)
  {
    /* processing
                  */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Maximum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the maximum.

preparation

At the beginning we set it to the value of the first element.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int max;
  scanf("%d", &a);
  /* preparation */
  while (a != 0)
  {
    /* processing
                  */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Maximum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the maximum.

preparation

At the beginning we set it to the value of the first element.

processing

If new element is larger than max, max $\leftarrow$ element.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int max;
  scanf("%d", &a);
  max=a;
  while (a != 0)
  {
    /* processing
                    */
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Maximum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the maximum.

preparation

At the beginning we set it to the value of the first element.

processing

If new element is larger than max, max ← element.

answer

We print the maximum.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int max;
  scanf("%d", &a);
  max=a;
  while (a != 0)
  {
    if (a > max)
      max = a;
    scanf("%d", &a);
  }
  /* answer */
  return 0;
}
```

link

Maximum of elements

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

declarations

We create a variable for storing the maximum.

preparation

At the beginning we set it to the value of the first element.

processing

If new element is larger than max, max ← element.

answer

We print the maximum.

```c
#include <stdio.h>

int main(void)
{
  int a;
  int max;
  scanf("%d", &a);
  max=a;
  while (a != 0)
  {
    if (a > max)
      max = a;
    scanf("%d", &a);
  }
  printf("%d", max);
  return 0;
}
```

link

Processing of characters

Let's wlite a ploglam that bulls
that prints the text arriving from input to the output in a way that all `'r'` letters are replaced by `'l'`.

- Differences from previous programs

Processing of characters

Let's wlite a ploglam that bulls
that prints the text arriving from input to the output in a way that all `'r'` letters are replaced by `'l'`.

- Differences from previous programs
  - The program will read characters until the newline `'\n'` character arrives

Processing of characters

Let's wlite a ploglam that bulls
that prints the text arriving from input to the output in a way that all `'r'` letters are replaced by `'l'`.

- Differences from previous programs
  - The program will read characters until the newline `'\n'` character arrives
  - There will be an answer on the output at every step of the processing loop

Processing of characters

Let's wlite a ploglam that bulls
that prints the text arriving from input to the output in a way that all `'r'` letters are replaced by `'l'`.

- Differences from previous programs
  - The program will read characters until the newline `'\n'` character arrives
  - There will be an answer on the output at every step of the processing loop
  - The value of this answer will be the read (scanned) character or character `'l'`, if the scanned character was an `'r'`.

Processing of characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's wlite a ploglam that bulls
that prints the text arriving from input to the output in a way that all `'r'` letters are replaced by `'l'`.

- Differences from previous programs
  - The program will read characters until the newline `'\n'` character arrives
  - There will be an answer on the output at every step of the processing loop
  - The value of this answer will be the read (scanned) character or character `'l'`, if the scanned character was an `'r'`.
- Think about upper- and lowercase letters too!

Processing of characters

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
#include <stdio.h>

int main(void)
{
  char a;
  scanf("%c", &a);
  while (a != '\n')
  {
    switch(a)
    {
    case 'R': printf("L"); break; /* ' " */
    case 'r': printf("l"); break;
    default: printf("%c", a);
    }
    scanf("%c", &a);
  }
  return 0;
}
```

link

A slightly different task

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!

A slightly different task

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!
- We can determine the average only after reading the entire data vector.

A slightly different task

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!
- We can determine the average only after reading the entire data vector.
- After this we have to go through all elements, in order to be able to collect the smaller ones (smaller than average).

A slightly different task

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!
- We can determine the average only after reading the entire data vector.
- After this we have to go through all elements, in order to be able to collect the smaller ones (smaller than average).
- We have to store the read (scanned) data elements.

A slightly different task

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!
- We can determine the average only after reading the entire data vector.
- After this we have to go through all elements, in order to be able to collect the smaller ones (smaller than average).
- We have to store the read (scanned) data elements.
- Obviously, this is not the right solution:

```
1 int a, b, c, d, e, f, g, h, i;
2 scanf("%d%d%d%d%d", &a, &b, &c, &d... /* No! No! No! */
```

A slightly different task

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's write a program, that counts, how many numbers have a value lower than the avarage of all numbers coming from the input!
- We can determine the average only after reading the entire data vector.
- After this we have to go through all elements, in order to be able to collect the smaller ones (smaller than average).
- We have to store the read (scanned) data elements.
- Obviously, this is not the right solution:

```
1 int a, b, c, d, e, f, g, h, i;
2 scanf("%d%d%d%d%d", &a, &b, &c, &d... /* No! No! No! */
```

- the correct approach is to make any element easily accessible with uniform name and using indexes (a_1, a_2, a_3, ... a_i).

Chapter 3

Arrays

Arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The concept of the array (datavector)

- linear data structure
- finite sequence of data of the same type, stored in the memory one after the other
- access of elements is by indexing, in arbitrary order

a_0	a_1	a_2	...	a_{n-1}

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

1

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

1

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

1

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

1

Syntax of arrays

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

- <number of elements> is a constant expression, it is already known when compiling (writing) the code!

1

Syntax of arrays

Declaration of array

```
<type of element> <identifier of array> [<number of elements>];
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
```

- <number of elements> is a constant expression, it is already known when compiling (writing) the code!
- This means that there is NO[1] such declaration as

```
1 int n = 5;
2 double data[n]; /* WRONG, n is not constant,
3                    it is variable */
```

[1] Actually the C99-standard makes it possible, but we don't.

Syntax of arrays

Access of elements of the array

```
<identifier of array> [<index of element>]
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Access of elements of the array

```
<identifier of array> [<index of element>]
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Syntax of arrays

Access of elements of the array

```
<identifier of array> [<index of element>]
```

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Syntax of arrays

Access of elements of the array

```
<identifier of array> [<index of element>]
```

- In case of an array with n elements, indexes run from 0 to $n-1$

data[0]	data[1]	data[2]	...	data[n-1]

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Syntax of arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Access of elements of the array

```
<identifier of array> [<index of element>]
```

- In case of an array with n elements, indexes run from 0 to $n-1$

data[0]	data[1]	data[2]	...	data[n-1]

- <index of element> can be a non-constant expression too, and that makes it useful!

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Syntax of arrays

Access of elements of the array

```
<identifier of array> [<index of element>]
```

- In case of an array with n elements, indexes run from 0 to $n-1$

data[0]	data[1]	data[2]	...	data[n-1]

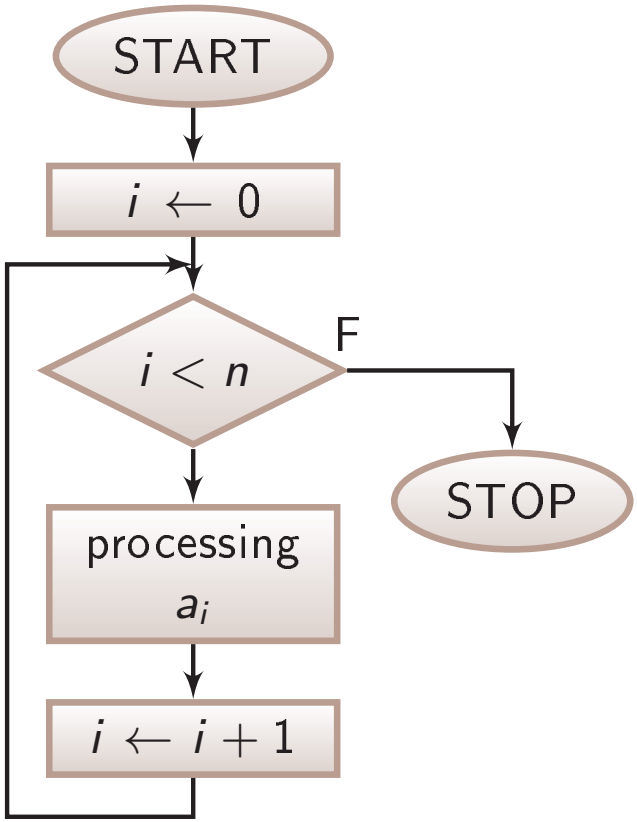
- <index of element> can be a non-constant expression too, and that makes it useful!
- With the elements of the array we can work in the same way as with a standalone variable

```
1 /* Array named 'data', storing 5 double values */
2 double data[5];
3
4 data[0] = 2.0;
5 data[1] = data[0];
6 data[i] = 3*data[2*q-1];
```

Traversing through an array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Traversing: accessing and processing each element of the array, one after the other

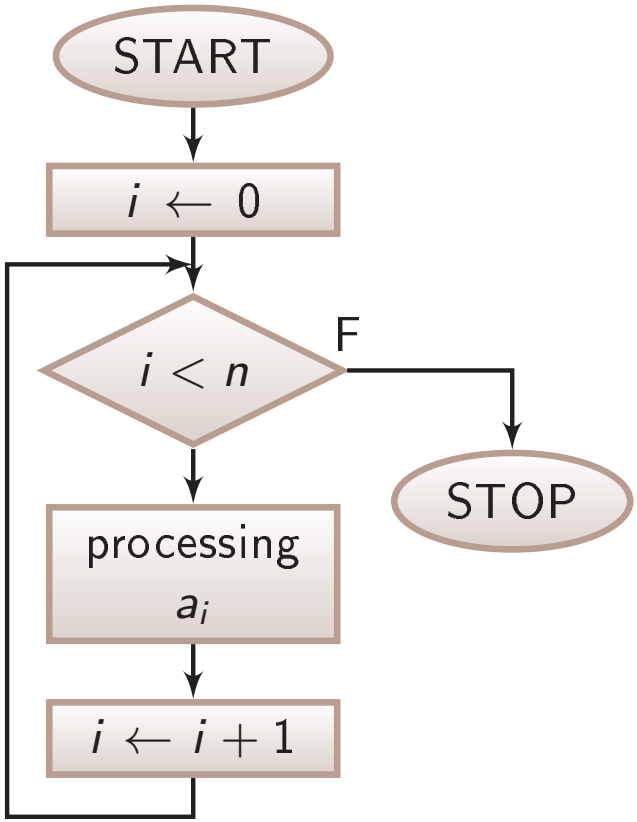


- Notations
  - n: constant size
  - a: the array
  - i: loop counter

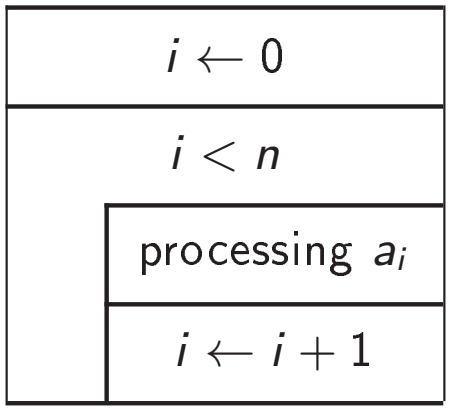
Traversing through an array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Traversing: accessing and processing each element of the array, one after the other



- Notations
  - n: constant size
  - a: the array
  - i: loop counter



- This is a `for` loop!

Traversing through an array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Realisation of traversing is suitably done with a `for` loop in the following way:

```
1  double array[10];                /* array of 10 elems */
2  int i;                           /* loop counter */
3  for (i = 0; i < 10; i = i+1) /* i = 0,1,...,9 */
4  {
5    /* processing array[i] */
6  }
```

Traversing through an array

- Realisation of traversing is suitably done with a `for` loop in the following way:

```
1  double array[10];                /* array of 10 elems */
2  int i;                           /* loop counter */
3  for (i = 0; i < 10; i = i+1) /* i = 0,1,...,9 */
4  {
5    /* processing array[i] */
6  }
```

- Example: Fill up an array with read (scanned) data

```
1  double array[10];
2  int i;
3  for (i = 0; i < 10; i = i+1)
4  {
5    scanf("%lf", &array[i]);
6  }
```

Traversing through an array

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's determine the average of the elements stored in the array!

```
1  double mean = 0.0;
2  for (i = 0; i < 10; i = i+1)
3  {
4    mean = mean + array[i];
5  }
6  mean = mean / 10;
```

Traversing through an array

- Let's determine the average of the elements stored in the array!

```
1 double mean = 0.0;
2 for (i = 0; i < 10; i = i+1)
3 {
4   mean = mean + array[i];
5 }
6 mean = mean / 10;
```

- Let's count the elements that are smaller than the average!

```
1 int n = 0;
2 for (i = 0; i < 10; i = i+1)
3 {
4   if (array[i] < mean)
5     n = n + 1;
6 }
```

Counting elements smaller than average

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
#include <stdio.h>

int main(void)
{
  /* declarations */
  double array[10];
  int i, n;
  double mean;

  /* filling up the array */
  for (i=0; i<10; i=i+1)
    scanf("%lf", &array[i]);

  /* calculating average */
  mean = 0.0;
  for (i=0; i<10; i=i+1)
    mean = mean + array[i];
  mean = mean / 10;

  /* counting */
  n = 0;
  for (i=0; i<10; i=i+1)
  {
    if (array[i] < mean)
      n = n+1;
  }

  /* answer */
  printf("%d", n);
  return 0;
}
```

link

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's write a program that decides whether it is true, that...
  - all elements of the vector have a given feature
  - none of the elements of the vector has a given feature
  - some elements of the vector has a given feature
  - some elements of the vector does not have a given feature

Decision

- Is it true, that all elements of the n-sized[2] data array are greater than 10?

[2] size usually means the number of the elements of the array.

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Is it true, that all elements of the n-sized[2] data array are greater than 10?

```
1  answer ← TRUE
2  For each i between 0 and n-1
3    IF data[i] <= 10
4      answer ← FALSE
5  OUT: answer
```

[2] size usually means the number of the elements of the array.

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Is it true, that all elements of the n-sized[2] data array are greater than 10?

```
1  answer ← TRUE
2  For each i between 0 and n-1
3    IF data[i] <= 10
4      answer ← FALSE
5  OUT: answer
```

- In C language there is no separate type for storing true/false values (boolean), we use `int` instead
  - 0 → FALSE
  - everything else → TRUE

[2] size usually means the number of the elements of the array.

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Is it true, that all elements of the n-sized[2] data array are greater than 10?

```
1  answer ← TRUE
2  For each i between 0 and n-1
3    IF data[i] <= 10
4      answer ← FALSE
5  OUT: answer
```

```
1  int answer = 1;
2  for (i=0; i<n; i=i+1)
3    if (data[i] <= 10)
4      answer = 0;
5  printf("%d", answer);
```

- In C language there is no separate type for storing true/false values (boolean), we use `int` instead
  - 0 → FALSE
  - everything else → TRUE

[2] size usually means the number of the elements of the array.

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Is it true, that all elements of the n-sized[2] data array are greater than 10?

```
1  answer ← TRUE
2  For each i between 0 and n-1
3    IF data[i] <= 10
4      answer ← FALSE
5  OUT: answer
```

```
1  int answer = 1;
2  for (i=0; i<n; i=i+1)
3    if (data[i] <= 10)
4      answer = 0;
5  printf("%d", answer);
```

- In C language there is no separate type for storing true/false values (boolean), we use `int` instead
  - 0 → FALSE
  - everything else → TRUE
- What if already the first (index 0) element turns out to be ≤ 10?

[2] size usually means the number of the elements of the array.

Decision

- a more efficient solution: we are checking only until the result is not yet certain.

Decision

- a more efficient solution: we are checking only until the result is not yet certain.

```
1  answer ← TRUE
2  i ← 0
3  UNTIL i < n AND answer TRUE
4    IF data[i] <= 10
5      answer ← FALSE
6    i ← i+1
7  OUT: answer
```

Decision

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- a more efficient solution: we are checking only until the result is not yet certain.

```
1  answer ← TRUE
2  i ← 0
3  UNTIL i < n AND answer TRUE
4    IF data[i] <= 10
5      answer ← FALSE
6    i ← i+1
7  OUT: answer
```

```
1  int answer = 1, i = 0;
2  while (i<n && answer==1)
3  {
4    if (data[i] <= 10)
5      answer = 0;
6    i = i+1;
7  }
8  printf("%d", answer);
```

Decision

- the same in a different way, without `answer` variable

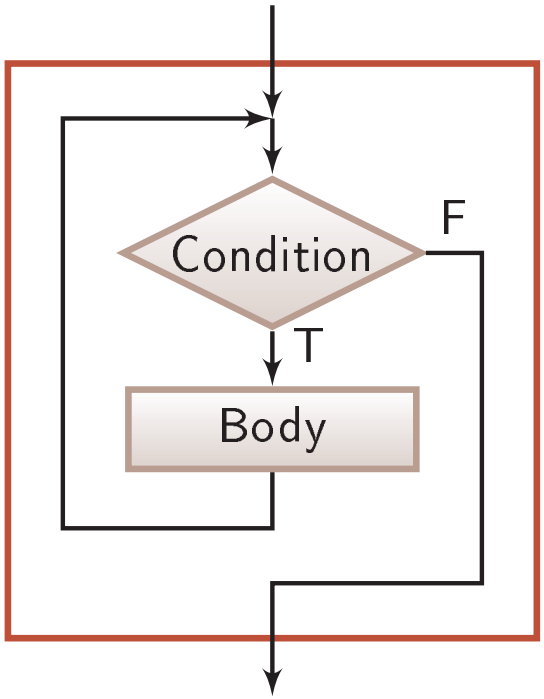
```
1  for (i=0; i<n; i=i+1)
2  {
3    if (data[i] <= 10)
4      break;
5  }
6  printf("%d", i == n);        /* reached the end? */
```

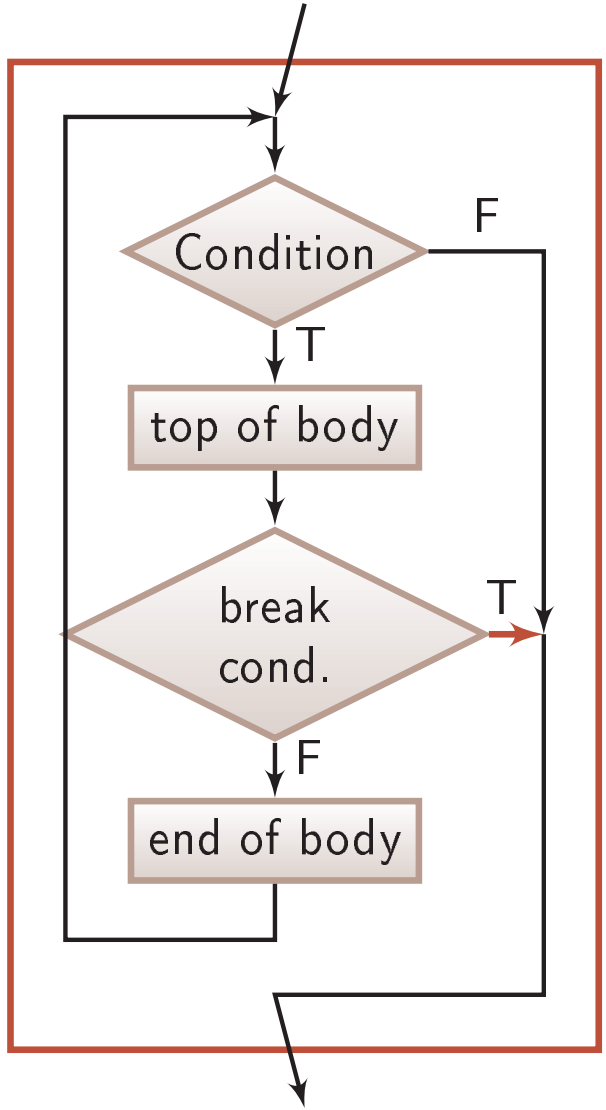
- The `break` statement interrupts (breaks) the execution of the cycle (`for`, `while`, `do`) that contains the break itself, and jumps to the next instruction

  it is not a structured element, therefore we use it only if it is unavoidable!

Top-test loop without and with break

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES





Decision


```
1  for (i=0; i<n; i=i+1)
2  {
3    if (data[i] <= 10)
4      break;
5  }
6  printf("%d", i == n);        /* reached the end? */
```

- Let's note that
  - when `break` jumps out of the `for` loop, the value of *i* is not incremented, so the answer will be right even if we jump out at the last element of the array.
  - In C language the type of a logical expression (`i == n`) is `int`:
    - FALSE $\rightarrow$ 0
    - TRUE $\rightarrow$ 1

Initial value

- If we declare an array in the way we learned it, its content will be uninitialized, in other words garbage from memory.

```
1 int numbers[5]; /* random content, memory garbage */
```

Initial value

- If we declare an array in the way we learned it, its content will be uninitialized, in other words garbage from memory.

```
1 int numbers[5]; /* random content, memory garbage */
```

This is not a problem, but we must not use the elements before filling them up with valid data.

Initial value

- If we declare an array in the way we learned it, its content will be uninitialized, in other words garbage from memory.

```
1 int numbers[5]; /* random content, memory garbage */
```

This is not a problem, but we must not use the elements before filling them up with valid data.

- Similarly to scalar variables, we can initialize the array at the point of declaration:

```
1 int numbers[5] = {1, -2, -3, 2, 4};
```

Initial value

- If we declare an array in the way we learned it, its content will be uninitialized, in other words garbage from memory.

```
1 int numbers[5]; /* random content, memory garbage */
```

This is not a problem, but we must not use the elements before filling them up with valid data.

- Similarly to scalar variables, we can initialize the array at the point of declaration:

```
1 int numbers[5] = {1, -2, -3, 2, 4};
```

- Only at this point (and only here!) we can omit the size, because it can be determined from the length of our list:

```
1 int numbers[] = {1, -2, -3, 2, 4};
```

Initial value

- If we declare an array in the way we learned it, its content will be uninitialized, in other words garbage from memory.

```
1 int numbers[5]; /* random content, memory garbage */
```

This is not a problem, but we must not use the elements before filling them up with valid data.

- Similarly to scalar variables, we can initialize the array at the point of declaration:

```
1 int numbers[5] = {1, -2, -3, 2, 4};
```

- Only at this point (and only here!) we can omit the size, because it can be determined from the length of our list:

```
1 int numbers[] = {1, -2, -3, 2, 4};
```

- And this is also valid:

```
1 int numbers[5] = {1, -2, -3 /* garbage, garbage */};
```

Collation

- Let's collect separately, in another vector the elements, that have a given feature!
- Let's print out the number of copied elements!
- Let the name of the source array that contains integers be `data`, and its size is 5.
- Let the name of the destination array be `selected`, and set its size to 5 – it should be obviously enough.
- Let's collect the negative elements separately!

Collation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We should traverse the data array, as learned.
- n denotes the number of elements that have been copied to the selected array.
- At the beginning, value of n is 0, it is increased at each copy.

```
1  int data[5] = {-1, 2, 3, -4, -7};/* declarations */
2  int selected[5];
3  int i, n;
4  n = 0;                                /* preparation */
5  for (i = 0; i < 5; i=i+1)             /* traversing */
6  {
7    if (data[i] < 0)                    /* investigation */
8    {
9      selected[n] = data[i];            /* copy */
10     n = n+1;
11   }
12 }
13 printf("Number of negatives: %d", n);/* answer */   link
```

Collation

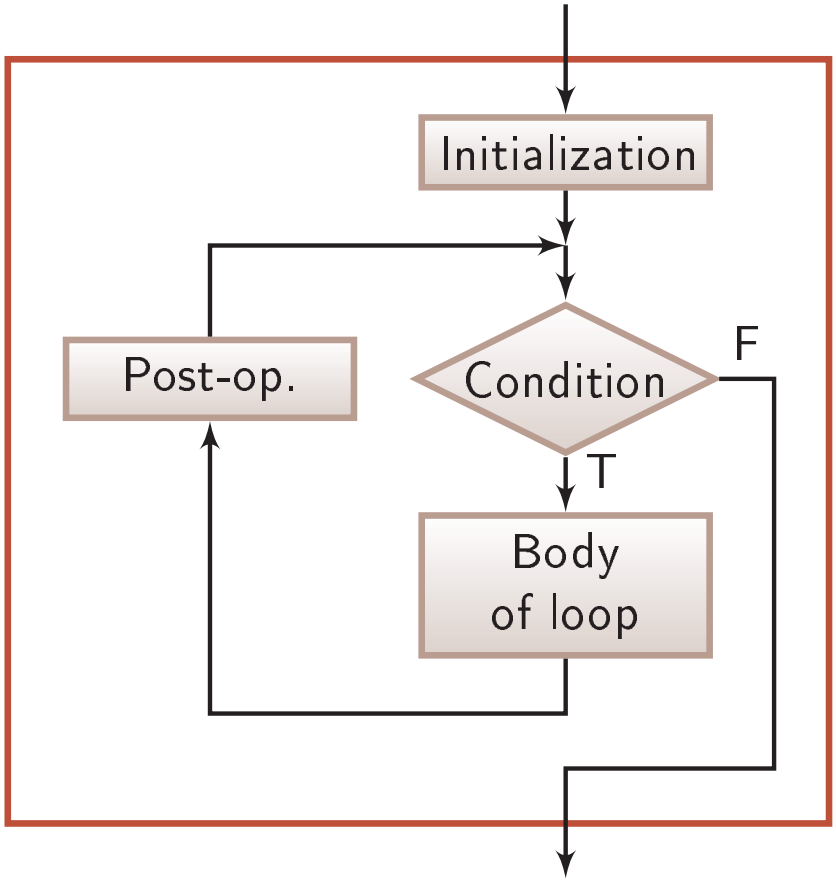
- A solution in a different approach:

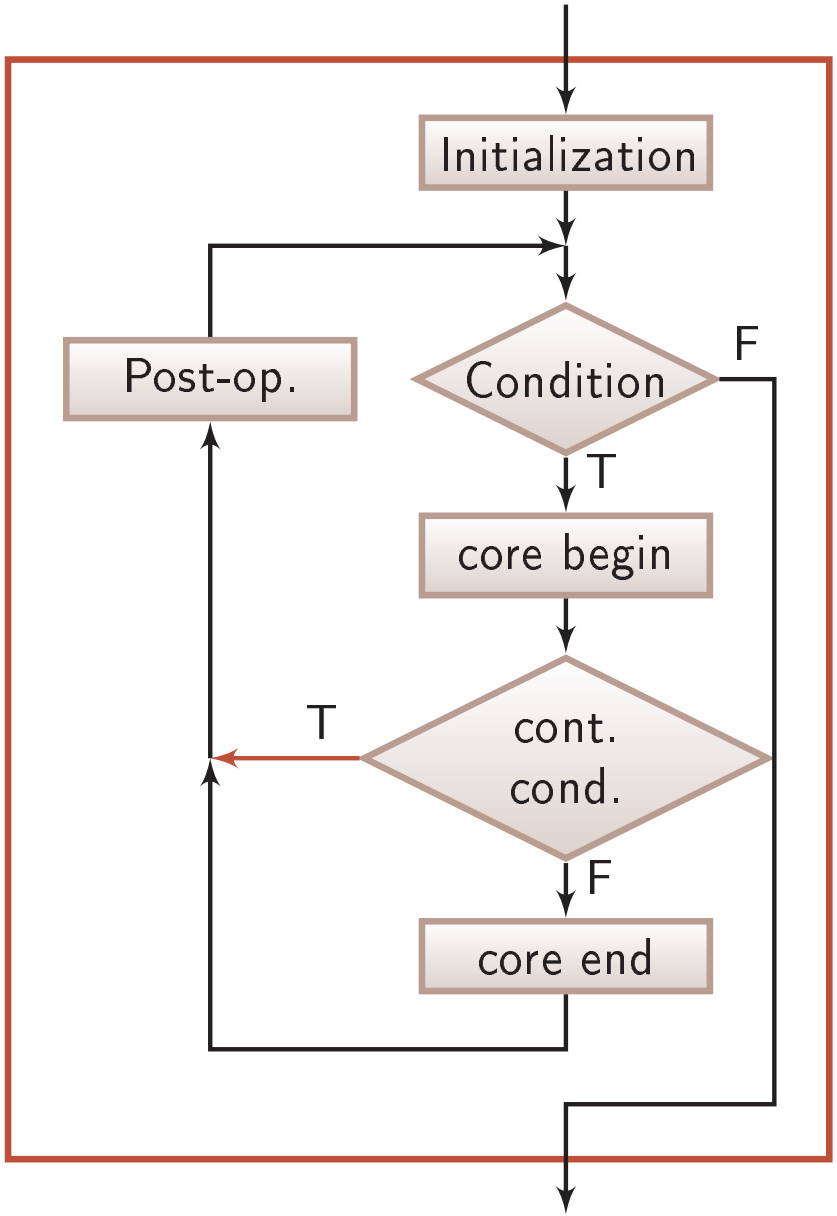
```
1  n = 0;                                   /* preparation */
2  for (i = 0; i < 5; i=i+1)                /* traversing */
3  {
4    if (data[i] >= 0)                      /* investigation */
5      continue;
6    selected[n] = data[i];                 /* copy */
7    n = n+1;
8  }
9  printf("Number of negatives: %d", n);/* answer */   link
```

- The `continue` statement interrupts (breaks) the execution of the body of the cycle (`for`, `while`, `do`) that contains the continue itself, and continues the cycle with the next iteration
  This is also not a structured element, use it moderately!
- It breaks the execution of the body of the loop, when using in a `for` loop, the post-operation is executed.

for loop without and with continue

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES





In-place separation

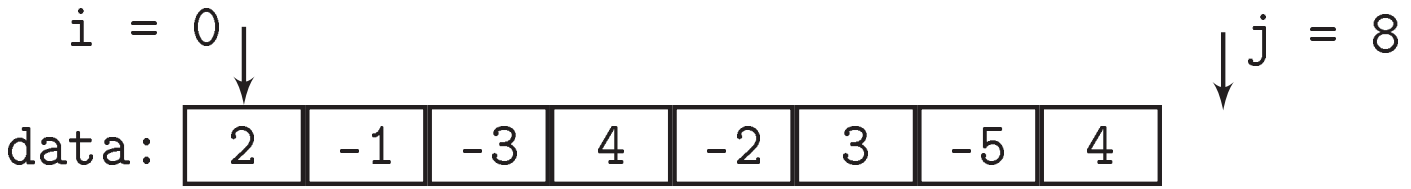
- Let's separate the elements of the `data` array, so we have all negative elements at the rear part (end) of the array!
- Let's print out the position of the first negative element!

In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

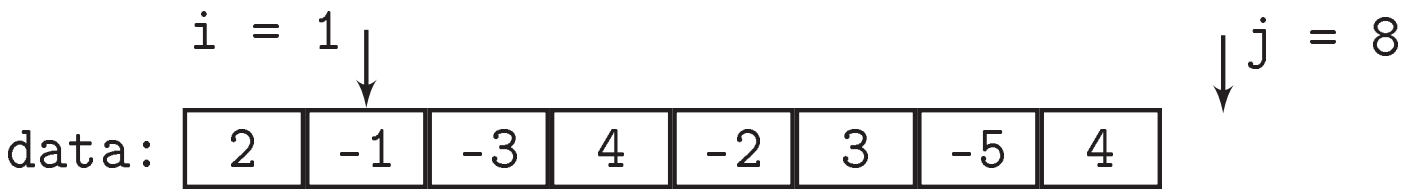


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

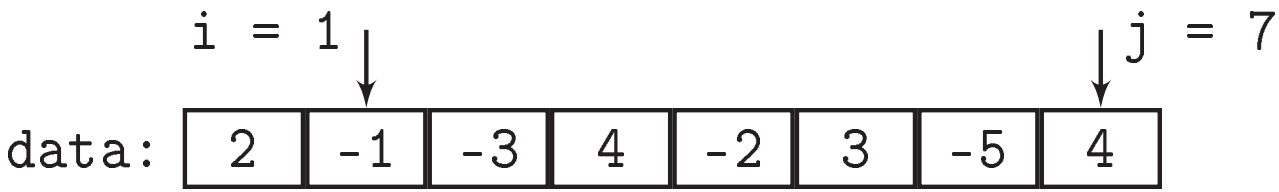


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

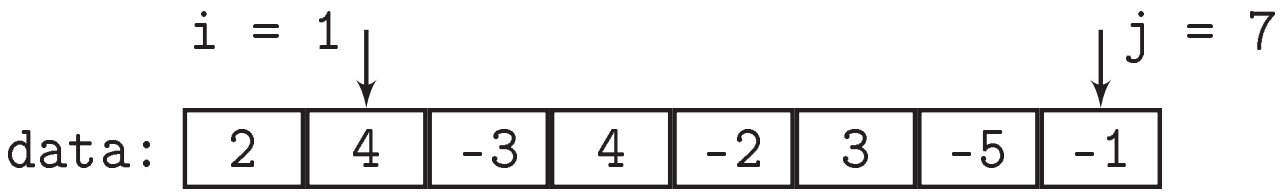


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

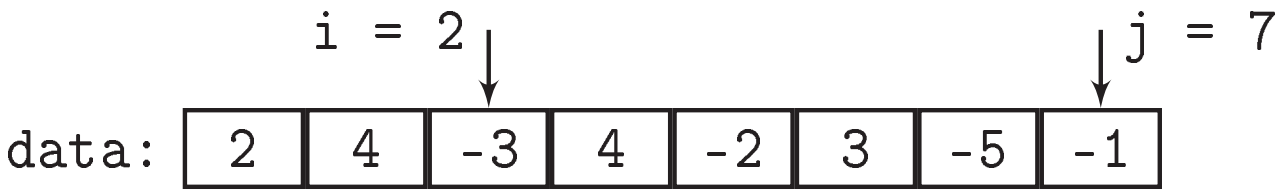


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

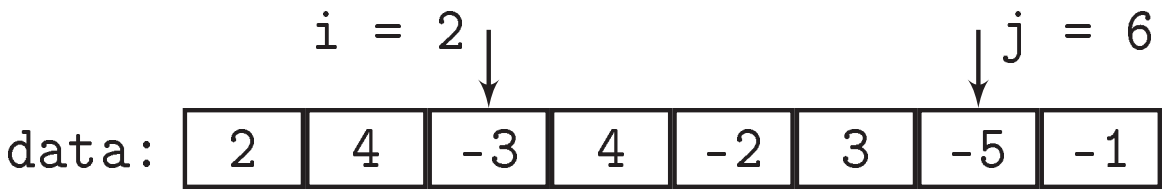


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

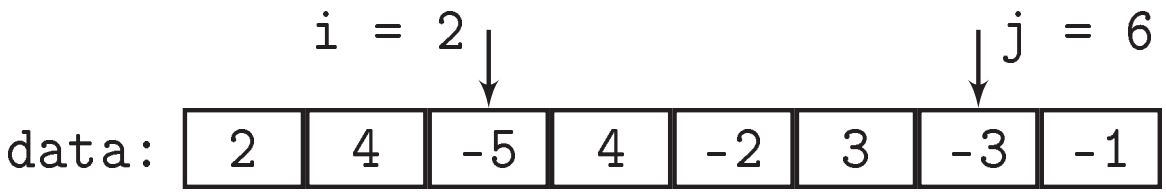


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

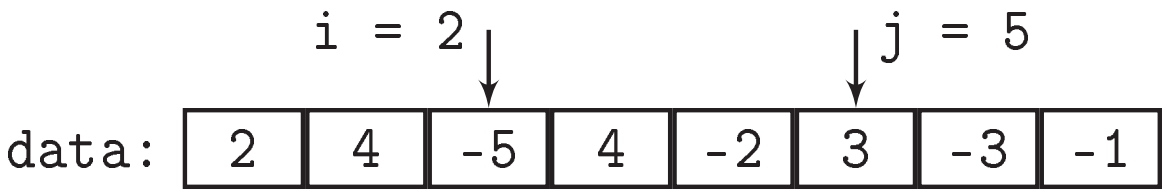


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

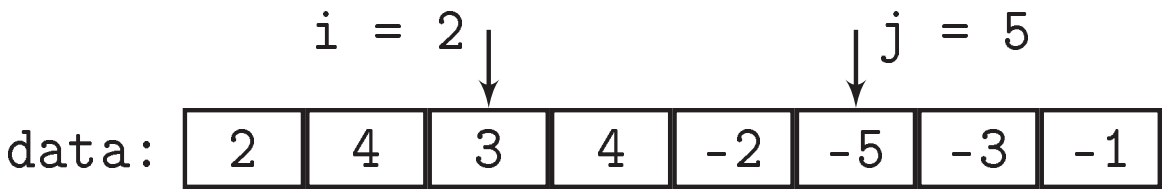


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

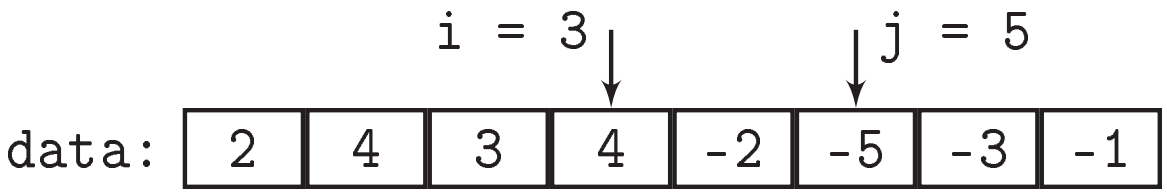


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

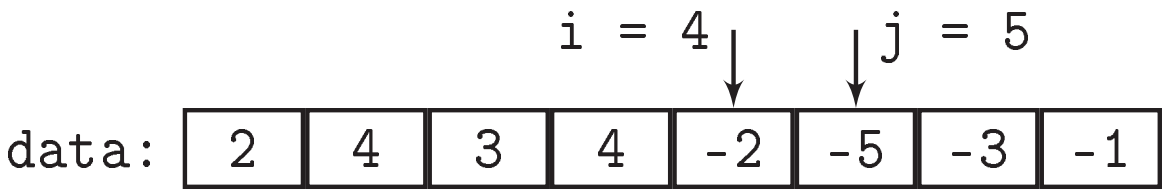


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size

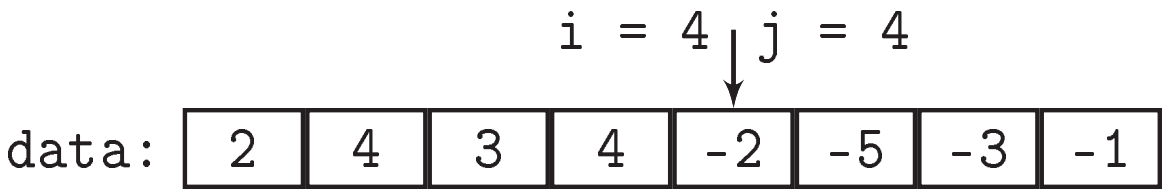


In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Testing with a vector of $n = 8$ size



In-place separation

- The algorithm

```
1  i ← 0;
2  j ← n;
3  WHILE i < j
4    IF data[i] >= 0
5      i ← i+1;
6    ELSE
7      j ← j-1;
8      data[i] ↔ data[j]
9  OUT: i
```

- Complete? Finite? – let's prove it!
  - In every cycle i or j is incremented $\rightarrow$ finite, n steps
  - i is incremented, if it points a non-negative element, $\rightarrow$ to the left from i there are only non-negative elements
  - after j is incremented, the pointed value is replaced by a negative one $\rightarrow$ from j onwards, there are only negative elems
  - If i and j meet, the array is separated

In-place separation

- Let's create the source code, it is fun!

```
1  int i = 0, j = 8;
2  while (i < j)
3  {
4    if (data[i] >= 0)
5      i=i+1;
6    else
7    {
8      int xchg;
9      j=j-1;
10     xchg = data[i];     /* exchange the values */
11     data[i] = data[j]; /* learn it well! */
12     data[j] = xchg;
13   }
14 }
15 printf("Index of 1st negative element is: %d", i);
```

link

Thank you for your attention.