Structured programs – Elements of C language

Basics of Programming 1

DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

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

September 16, 2020

Contents

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

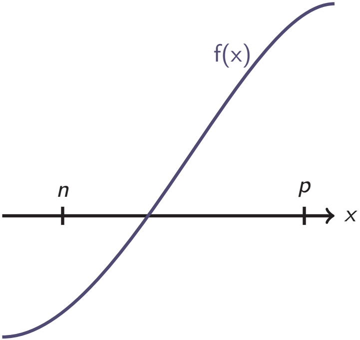
Chapter 1

Structured programming

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

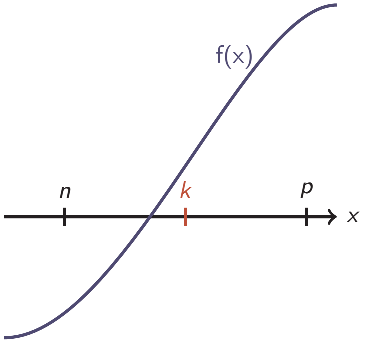



```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

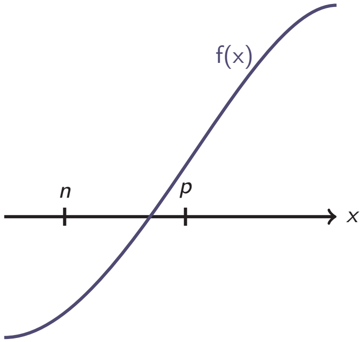



```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



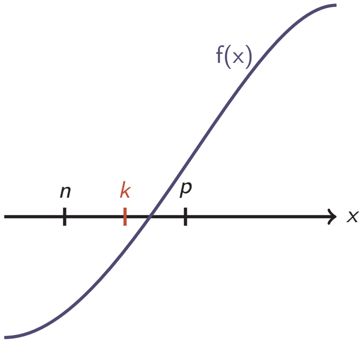

```
p-n < eps?
IF TRUE, JUMP TO 10
k ← (n+p) / 2
f(k) < 0?
IF TRUE, JUMP TO 8
p ← k;
JUMP TO 1
n ← k;
JUMP TO 1
The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



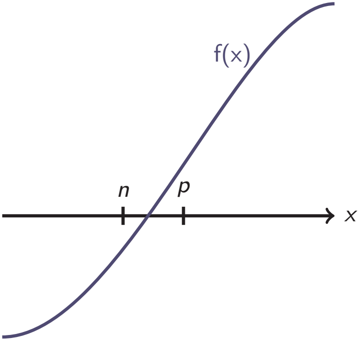

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

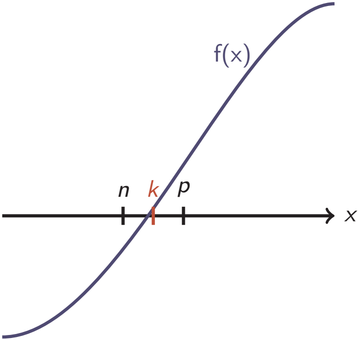



```
p-n < eps?
IF TRUE, JUMP TO 10
k ← (n+p) / 2
f(k) < 0?
IF TRUE, JUMP TO 8
p ← k;
JUMP TO 1
n ← k;
JUMP TO 1
The zero is: n
```

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



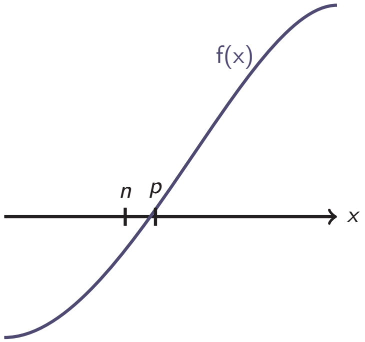

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

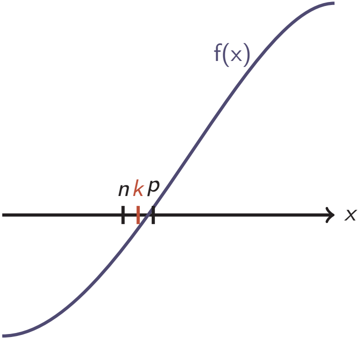



```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

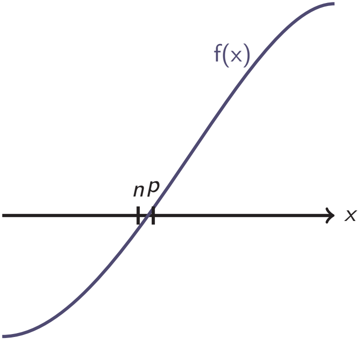



```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

Finding zeros of functions

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



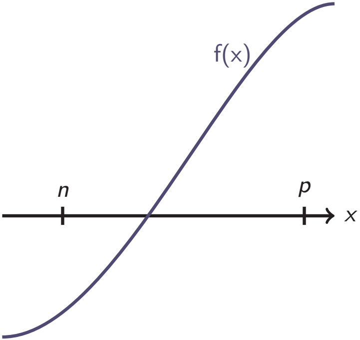

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



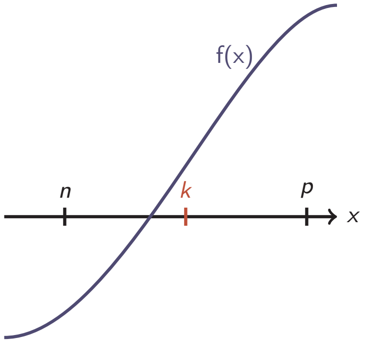

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



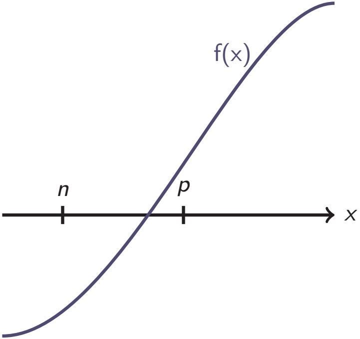

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

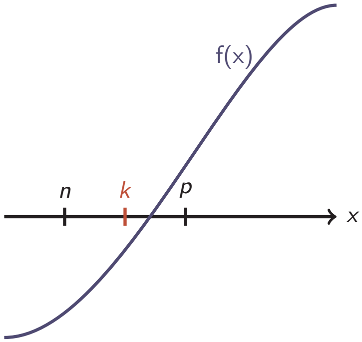



```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

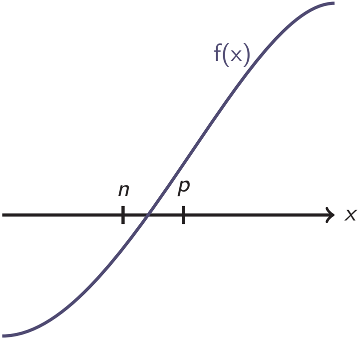



```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

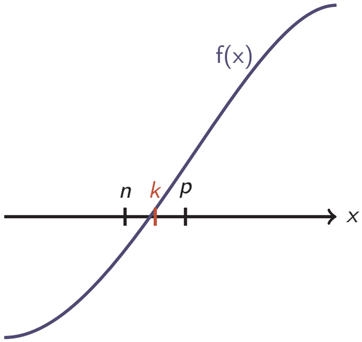



```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

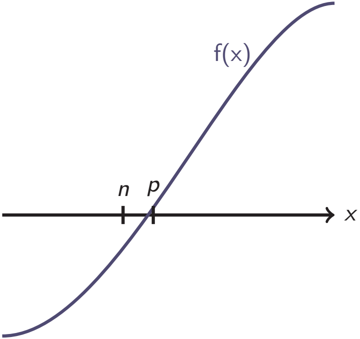



```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.

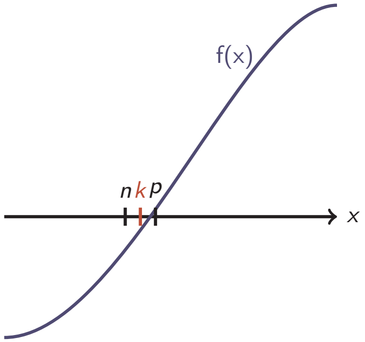



```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.



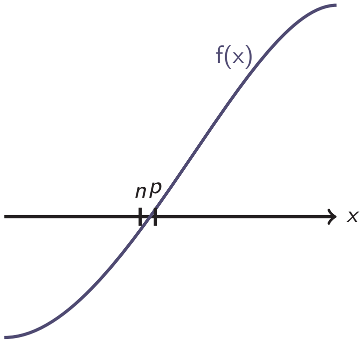

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Finding zeros – a different approach

We are searching the zeros of function $f(x)$, a monotonically increasing function, between points n and p, with ϵ accuracy.




```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

Structured vs unstructured

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Two programs of the same algorithm

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

```
p-n < eps?
IF TRUE, JUMP TO 10
k ← (n+p) / 2
f(k) < 0?
IF TRUE, JUMP TO 8
p ← k;
JUMP TO 1
n ← k;
JUMP TO 1
The zero is: n
```

Structured vs unstructured

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Two programs of the same algorithm

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

- Structured program
- Unstructured program

Structured vs unstructured

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Two programs of the same algorithm

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

- Structured program
  - easy to maintain
- Unstructured program
  - spaghetti-code

Structured vs unstructured

- Two programs of the same algorithm

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

- Structured program
  - easy to maintain
  - complex control

- Unstructured program
  - spaghetti-code
  - easy control

Structured vs unstructured

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Two programs of the same algorithm

```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
The zero is: n
```

```
p-n < eps?
IF TRUE, JUMP TO 10
k ← (n+p) / 2
f(k) < 0?
IF TRUE, JUMP TO 8
p ← k;
JUMP TO 1
n ← k;
JUMP TO 1
The zero is: n
```

- Structured program
  - easy to maintain
  - complex control
  - higher level

- Unstructured program
  - spaghetti-code
  - easy control
  - "hardware-level"

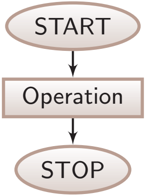
Structured vs unstructured

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Hardware level languages
  - Lot of simple instructions
  - Easy control (`JUMP`; `IF TRUE, JUMP`)
  - Unstructured layout
  - The processor can interpret only this
- Higher level languages
  - Rather few, but complex instructions
  - More difficult control (`WHILE...REPEAT...`; `IF...THEN...ELSE...`)
  - Structured layout
  - The processor is unable to interpret it.
- The compiler transforms a high level structured program into a hardware level program, that is equivalent to the original one.
- We create a high level structured program,
  we use the compiler to translate it,
  and we execute the hardware level code.

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



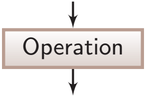
- All structured programs follow this simple scheme:
  - The structure of the program is determined by the inner structure (layout) of Operation.
    - Operation can be:
      - Elementary operation (action)
      - Sequence
      - Loop or repetition
      - Selection

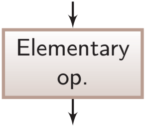
Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Elementary operation

that cannot be further expanded





`Elementary op.`

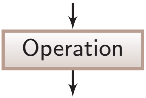
- The empty operation (`don't do anything`) is also an elementary operation

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Sequence

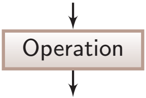
Execution of two operations after eachother, in the given order

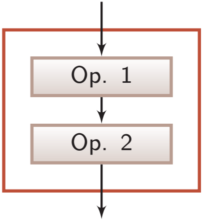


Elements of structured programs

Sequence

Execution of two operations after eachother, in the given order



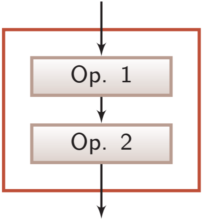


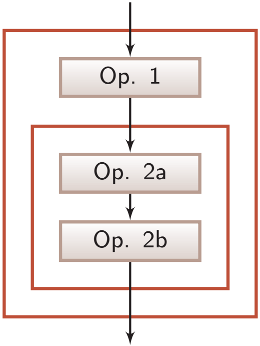

```
Op. 1
Op. 2
```

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

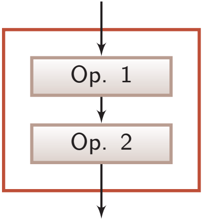
- Each element of the sequence itself is an operation, so they can be expanded into a sequence

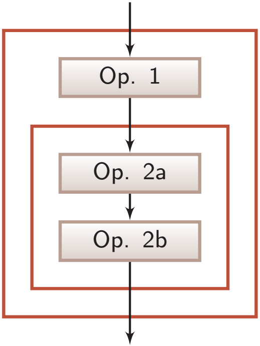




Elements of structured programs

- Each element of the sequence itself is an operation, so they can be expanded into a sequence



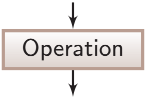


- The expansion can be continued, so a sequence can be an arbitrary long (finite) series of operations.

Elements of structured programs

Condition-based selection

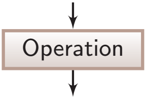
Execution of one of two operations, depending on the logical value of a condition (true or false)

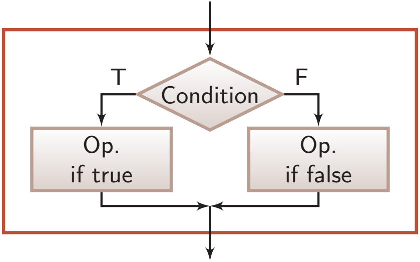


Elements of structured programs

Condition-based selection

Execution of one of two operations, depending on the logical value of a condition (true or false)

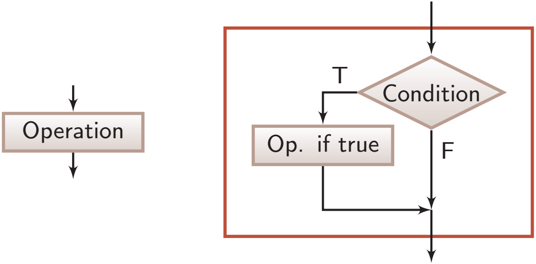





```
IF Condition
  Op. if true
ELSE
  Op. if false
```

Elements of structured programs

- One of the branches can also be empty.



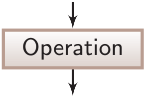

```
IF Condition
  Op. if true
```

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Top-test loop

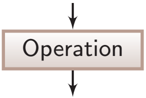
Repetition of an operation as long as a condition is true.

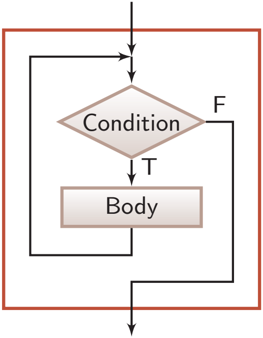


Elements of structured programs

Top-test loop

Repetition of an operation as long as a condition is true.






```
WHILE Condition
  Body of loop
```

Elements of structured programming

Theorem of structured programming

By using only

- elementary operation,
- sequence,
- selection, and
- loop

ALL algorithms can be constructed.

The structogram

- The flowchart
  - a tool for describing unstrutured programs
  - can ba translated (compiled) into an unstructured program immediately (IF TRUE, JUMP)
  - structued elements (esp. loops) are hard to recognize within it
- The structogram
  - a tool for representing structured programs
  - only a structured program can be represented by it
  - it is easily translated into a structured program

The structogram

- The program is a rectangle

The structogram

- The program is a rectangle

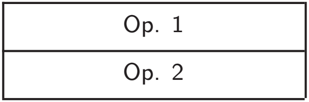
- it can be expanded into more rectangles with the elements below

The structogram

- The program is a rectangle

- it can be expanded into more rectangles with the elements below

- Sequence



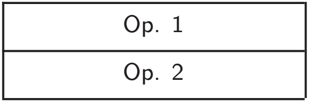
The structogram

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

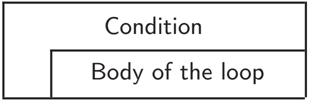
- The program is a rectangle

- it can be expanded into more rectangles with the elements below

- Sequence



- Top-test loop



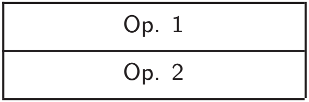
The structogram

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

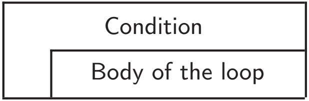
- The program is a rectangle

- it can be expanded into more rectangles with the elements below

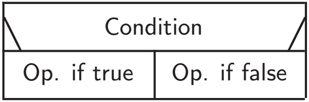
- Sequence



- Top-test loop

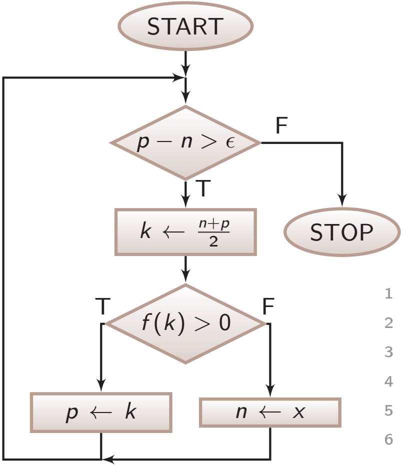


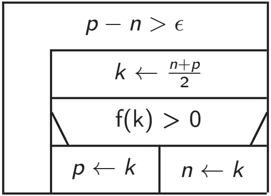
- Selection



The structogram

- Finding zeros – flowchart, structogram, structured pseudo-code






```
WHILE p-n > eps, repeat
  k ← (n+p) / 2
  IF f(k) > 0
    p ← k;
  OTHERWISE
    n ← k;
```

Chapter 2

Structured programming in C

Sequence in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Forming a sequence is listing instructions one after eachother

```c
/* football.c -- football fans */
#include <stdio.h>
int main()
{
  printf("Are you");  /* no new line here */
  printf(" blind?\n"); /* here is new line */
  printf("Go Bayern, go!");
  return 0;
}
```

link

```
Are you blind?
Go Bayern, go!
```

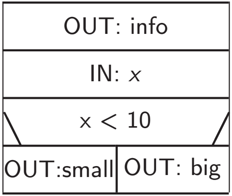
Selection control in C – the `if` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!

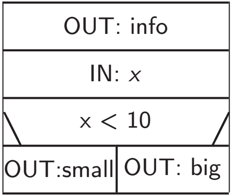
Selection control in C – the `if` statement

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!



Selection control in C – the `if` statement

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!

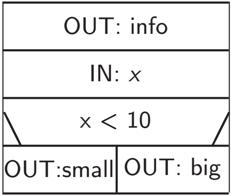



```
Let x be an integer
OUT: info
IN: x
IF x < 10
  OUT: small
OTHERWISE
  OUT: big
```

Selection control in C – the `if` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!




```
Let x be an integer
OUT: info
IN: x
IF x < 10
  OUT: small
OTHERWISE
  OUT: big
```

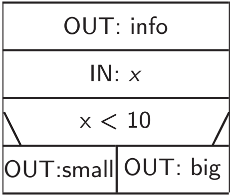
```c
#include <stdio.h>
int main()
{
  int x;
  printf("Please enter a number: ");
  scanf("%d", &x);
  if (x < 10)
/* condition */
    printf("small"); /*true branch*/
  else
    printf("big"); /*false branch*/
  return 0;
}
```

link

Selection control in C – the `if` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!




```
Let x be an integer
OUT: info
IN: x
IF x < 10
  OUT: small
OTHERWISE
  OUT: big
```

```c
#include <stdio.h>
int main()
{
  int x;
  printf("Please enter a number: ");
  scanf("%d", &x);
  if (x < 10)
/* condition */
    printf("small"); /*true branch*/
  else
    printf("big"); /*false branch*/
  return 0;
}
```

link

```
Please give an integer number: 5
small
```

Selection control – the if statement

Syntax of the if statement

if (<condition expression>) <statement if true>
[else <statement if false>]$_{opt}$

```
if (x < 10)           /* condition */
  printf("small"); /* true branch */
else
  printf("big"); /* false branch */
```

Selection control – the if statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the if statement

```
if (<condition expression>) <statement if true>
[ else <statement if false> ]opt
```

```c
if (x < 10)          /* condition */
  printf("small"); /* true branch */
else
  printf("big"); /* false branch */
```

Selection control – the if statement

Syntax of the if statement

if (<condition expression>) <statement if true>
[else <statement if false>]$_{opt}$

```
if (x < 10)           /* condition */
  printf("small");    /* true branch */
else
  printf("big"); /* false branch */
```

Selection control – the if statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the if statement

```
if (<condition expression>) <statement if true>
[ else <statement if false> ]opt
```

```c
if (x < 10)            /* condition */
  printf("small"); /* true branch */
else
  printf("big"); /* false branch */
```

Selection control – the if statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the if statement

```
if (<condition expression>) <statement if true>
[ else <statement if false> ]opt
```

```c
if (x < 10)           /* condition */
  printf("small"); /* true branch */
else
  printf("big"); /* false branch */
```

```c
if (a < 0)        /* creating absolute value */
  a = -a;
/* no false branch */
```

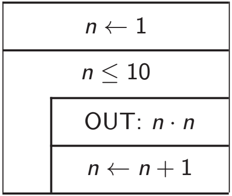
Top-test loop in C – the `while` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the square of the integer numbers between 1 and 10!

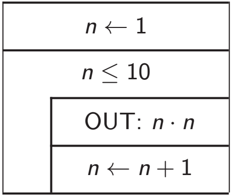
Top-test loop in C – the `while` statement

Let's print the square of the integer numbers between 1 and 10!



Top-test loop in C – the `while` statement

Let's print the square of the integer numbers between 1 and 10!

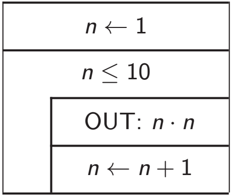



```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

Top-test loop in C – the `while` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

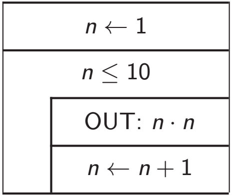
```c
#include <stdio.h>
int main()
{
  int n;
  n = 1; /* initialization */
  while (n <= 10) /* condition */
  {
    printf("%d ", n*n);/* printing */
    n = n+1;
/* increment */
  }
  return 0;
}
```

link

Top-test loop in C – the while statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

```c
#include <stdio.h>
int main()
{
  int n;
  n = 1; /* initialization */
  while (n <= 10) /* condition */
  {
    printf("%d ", n*n);/* printing */
    n = n+1; /* increment */
  }
  return 0;
}
```

link

```
1 4 9 16 25 36 49 64 81 100
```

Top-testing loop – the while statement

Syntax of the while statement

```
while (<condition expression>) <instruction>
```

- If `<instruction>` is a sequence, we enclose it in a {block}:

```c
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

Top-testing loop – the while statement

Syntax of the while statement

```
while (<condition expression>) <instruction>
```

- If <instruction> is a sequence, we enclose it in a {block}:

```
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

- In language C an instruction always can be replaced with a block.

Top-testing loop – the while statement

Syntax of the while statement

```
while (<condition expression>) <instruction>
```

- If <instruction> is a sequence, we enclose it in a {block}:

```c
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

- In language C an instruction always can be replaced with a block.

Top-testing loop – the while statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the while statement

```
while (<condition expression>) <instruction>
```

- If <instruction> is a sequence, we enclose it in a {block}:

```
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

- In language C an instruction always can be replaced with a block.

A complex application

- By using sequence, loop and selection, we can construct everything!

A complex application

- By using sequence, loop and selection, we can construct everything!
- We know enough to construct the algorithm of finding the zeros in C!

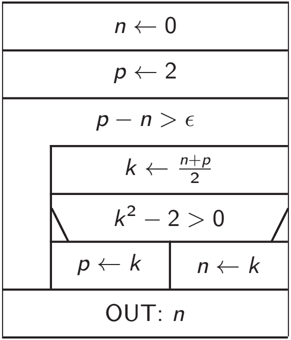
A complex application

- By using sequence, loop and selection, we can construct everything!
- We know enough to construct the algorithm of finding the zeros in C!
- A new element: a type for storing real numbers is called `double` type (to be learned later)

```
double a;            /* the real number */
a = 2.0;             /* assignement of value */
printf("%f", a);     /* printing */
```

Finding zero of a function

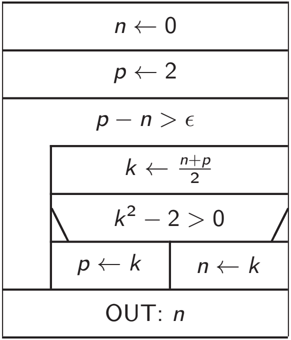
We are searching the zeros of function $f(x) = x^2 - 2$, between points $n = 0$ and $p = 2$, with $\epsilon = 0{,}001$ accuracy.



Finding zero of a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

We are searching the zeros of function $f(x) = x^2 - 2$, between points $n = 0$ and $p = 2$, with $\epsilon = 0{,}001$ accuracy.




```c
#include <stdio.h>

int main()
{
  double n = 0.0, p = 2.0;
  while (p-n > 0.001)
  {
    double k = (n+p)/2.0;
    if (k*k-2.0 > 0.0)
      p = k;
    else
      n = k;
  }
  printf("The zero is: %f", n);

  return 0;
}
```

link

Chapter 3

Other structured elements

Elements of structured programs

- We have seen that the structured elements we had learned so far are enough for everything.
- Only for a higher comfort, we introduce new elements, that of course origin from the earlier ones.

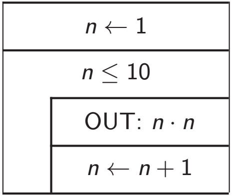
Top-test loop in C – the `for` statement

Let's print the square of the integer numbers between 1 and 10!

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

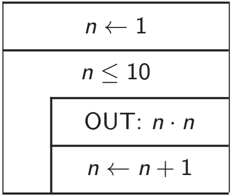
Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

Top-test loop in C – the for statement

Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

Because the structure of

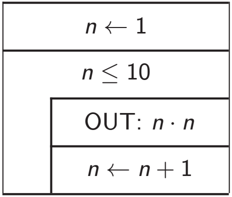
- Initializations
- As long as Condition is TRUE
  - Operation
  - Increment

is very common in programming, we simplify its application with a new statement.

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the square of the integer numbers between 1 and 10!




```
Let n be integer
from n=1, WHILE n<=10, one-by-one
  OUT: n*n
```

```c
#include <stdio.h>
int main()
{
  int n;
  for (n = 1; n <= 10; n = n+1)
    printf("%d ", n*n);
  return 0;
}
```

link

```
1 4 9 16 25 36 49 64 81 100
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 1

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 1

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 1

```
1
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 2

```
1
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 2

```
1
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 2

```
1 4
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 3

```
1 4
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 3

```
1 4
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 3

```
1 4 9
```

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

```
n:  4
```

```
1 4 9
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 4

```
1 4 9
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 4

```
1 4 9 16
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 5

```
1 4 9 16
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 5

```
1 4 9 16
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 5

```
1 4 9 16 25
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 6

```
1 4 9 16 25
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 6

```
1 4 9 16 25
```

Top-test loop in C – the `for` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 6

```
1 4 9 16 25 36
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 7

```
1 4 9 16 25 36
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 7

```
1 4 9 16 25 36
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 7

```
1 4 9 16 25 36 49
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 8

```
1 4 9 16 25 36 49
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 8

```
1 4 9 16 25 36 49
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 8

```
1 4 9 16 25 36 49 64
```

Top-test loop in C – the `for` statement

Syntax of the `for` statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 9

```
1 4 9 16 25 36 49 64
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 9

```
1 4 9 16 25 36 49 64
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 9

```
1 4 9 16 25 36 49 64 81
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 10

```
1 4 9 16 25 36 49 64 81
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 10

```
1 4 9 16 25 36 49 64 81
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 10

```
1 4 9 16 25 36 49 64 81 100
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 11

```
1 4 9 16 25 36 49 64 81 100
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

```
n:  11
```

```
1 4 9 16 25 36 49 64 81 100
```

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 11

```
1 4 9 16 25 36 49 64 81 100
```

Multiplication table

Let's print the $10 \cdot 10$ multiplication table!

Multiplication table

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)

Multiplication table

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)
- In every row
  - we print into 10 columns (`col = 1, 2, 3, ...10`)

Multiplication table

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)
- In every row
  - we print into 10 columns (`col = 1, 2, 3, ...10`)
  - In every column
    - We print the value of `row*col`

Multiplication table

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)
- In every row
  - we print into 10 columns (`col = 1, 2, 3, ...10`)
  - In every column
    - We print the value of `row*col`
  - After this we have to start a new line

Multiplication table

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)
- In every row
  - we print into 10 columns (`col = 1, 2, 3, ...10`)
  - In every column
    - We print the value of `row*col`
  - After this we have to start a new line

```
int row;
for (row = 1; row <= 10; row=row+1)
{
  int col;      /* declaration at beginning of block */
  for (col = 1; col <= 10; col=col+1)
    printf("%4d", row*col); /* printing with size 4 */
  printf("\n"); /* this is not inside the for */
}
```

link

Multiplication table

- It might be advantageous to enclose in a block even one single instruction, because it might make the code more understandable!

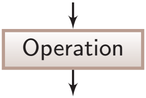
```c
int row;
for (row = 1; row <= 10; row=row+1)
{
  int col;      /* declaration at beginning of block */
  for (col = 1; col <= 10; col=col+1)
  {
    printf("%4d", row*col); /* printing with size 4 */
  }
  printf("\n");
}
```

link

Elements of structured programs

Bottom-test loop

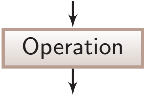
Repetition of an operation as long as a condition is true.

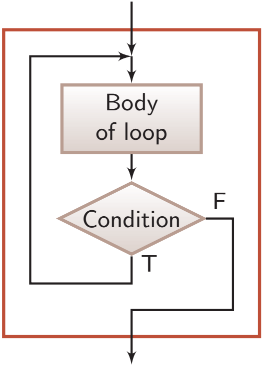


Elements of structured programs

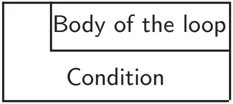
Bottom-test loop

Repetition of an operation as long as a condition is true.



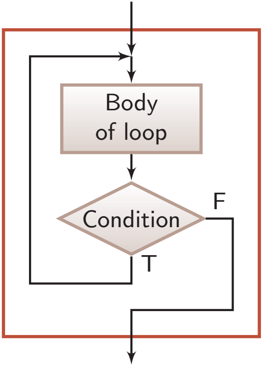


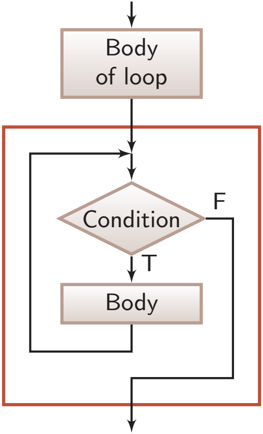

```
REPEAT
  Body of loop
WHILE Condition
```



Elements of structured programs

- It can be traced back to sequence and a top-test loop





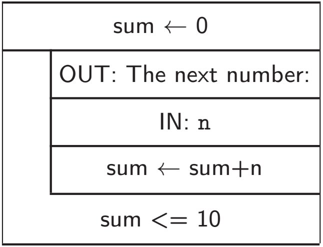
Bottom-test loop – the do statement

Let's read positive integer numbers! We stop if the sum of the numbers is larger than 10.

Bottom-test loop – the do statement

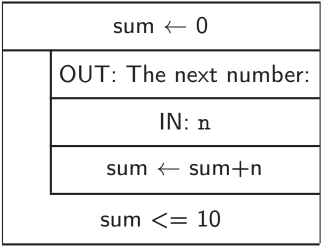
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's read positive integer numbers! We stop if the sum of the numbers is larger than 10.



Bottom-test loop – the do statement

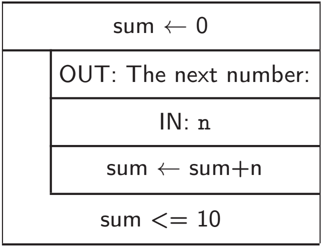
Let's read positive integer numbers! We stop if the sum of the numbers is larger than 10.




```
sum ← 0
REPEAT
  OUT: Info
  IN: n
  sum ← sum+n
WHILE sum ≤ 10
```

Bottom-test loop – the do statement

Let's read positive integer numbers! We stop if the sum of the numbers is larger than 10.




```
sum ← 0
REPEAT
  OUT: Info
  IN: n
  sum ← sum+n
WHILE sum ≤ 10
```

```c
#include <stdio.h>
int main()
{
  int sum = 0, n;
  do
  {
    printf("The next number: ");
    scanf("%d", &n);
    sum = sum+n;
  }
  while (sum <= 10);
  return 0;
}
```

link

Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the do statement

```
do <instruction> while (<condition expression>);
```

```c
do
{
  printf("The next number: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```

Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the do statement

```
do <instruction> while (<condition expression>);
```

```
do
{
  printf("The next number: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```

Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the do statement

```
do <instruction> while (<condition expression>);
```

```c
do
{
  printf("The next number: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```

Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the do statement

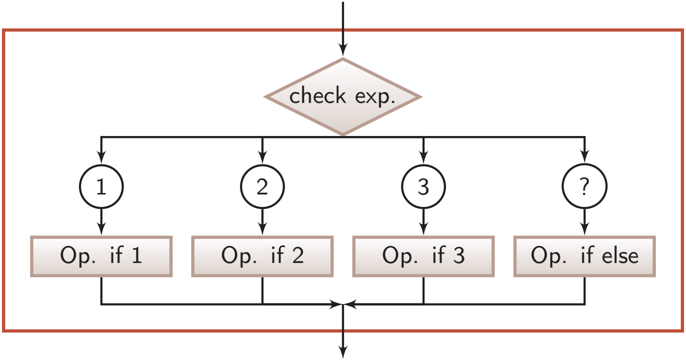
```
do <instruction> while (<condition expression>);
```

```c
do
{
  printf("The next number: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```

Elements of structured programs

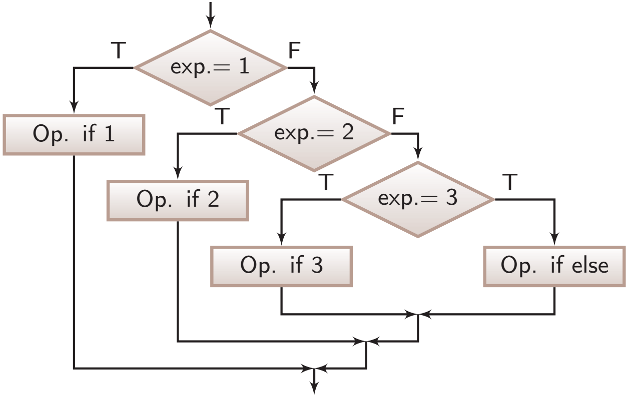
Integer-value based selection

Execution of operations depending on the value of an integer expression



Elements of structured programs

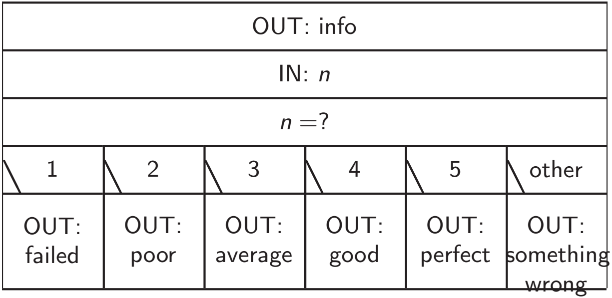
- It can be constructed as nested selections



Integer-value based selection – the `switch` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's assign (connect) written evaluations to grades given in numbers!



Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's assign (connect) written evaluations to grades given in numbers!

```c
#include <stdio.h>
int main() {
  int n;
  printf("Please enter the grade: ");
  scanf("%d", &n);
  switch (n)
  {
    case 1: printf("failed"); break;
    case 2: printf("poor"); break;
    case 3: printf("average"); break;
    case 4: printf("good"); break;
    case 5: printf("perfect"); break;
    default: printf("something wrong");
  }
  return 0;
}
```

link

Integer-value based selection – the switch statement

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The `break` instructions are not part of the syntax. If we omit them, the `switch` will remain syntactically correct, but it will not provide the same result as before:

```c
switch (n)
{
  case 1: printf("failed");
  case 2: printf("poor");
  case 3: printf("average");
  case 4: printf("good");
  case 5: printf("perfect");
  default: printf("something wrong");
}
```

link

```
Please enter the grade:  2
pooraveragegoodperfectsomething wrong
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The constant expressions are only entry points, and from this point on, all instructions are executed until the first `break` or until the enf of the block:

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2:
  case 3:
  case 4:
  case 5: printf("passed"); break;
  default: printf("something wrong");
}
```

link

```
Please enter the grade:  2
passed
```

Thank you for your attention.