Basic elements of the C language

Basics of Programming 1

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

10. September, 2025

Contents

1 Introduction
- Requirements

2 Basic terms
- The imperative programming paradigm
- The algorithm
- The data – constants and variables
- Expressions
- Programming languages

3 Structured programming
- Introduction
- Definition
- Elements of structured programs
- Theorem of structured programming
- The structogram

4 C language basics
- History
- The first program
- Variables
- Scanning data, inputting

Chapter 1

Introduction

Contact

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- BME Faculty of Electrical Engineering and Informatics
  - Department of Electron Devices
- Márton Németh
  - email: nemeth.marton@vik.bme.hu
- Important webpages for the course:
  - The main webpage (slides, exercises, solutions): `http://www.eet.bme.hu/bop`
  - The portal managing the assignments: `https://cprog.eet.bme.hu`

Requirements

- Laboratory practices
  - Active participation on at least 70% of the labs
  - Entrance test at the beginning of the labs, Max. number of absences/falied entrance tests: 4
- Classroom practices
  - Participation on at least 70% of the practices
  - 6 mid-term "small tests" in total (10 pts each)
  - The sum of best 4 must be above 20 pts
  - No re-take available
  - Small tests on 4th, 6th, 8th, 10th, 11th, 13th week's practice

Requirements

- "Big test"
  - There will be 2 mid-term "big tests"
  - Option 1: sum of the scores $\geq 50\%$
  - Option 2: the score of the second one $\geq 50\%$
  - The one with lower score can be re-taken
  - 1st midterm for CE students on 21st of October, from 18:00; for EE and PE students on 27th of October from 18:00
  - 2nd midterm for PE and EE students on 11th of December, from 18:00, for CE students 12th of October 8:00
  - retake (just one can be retaken) 18th of December 10:00

Requirements

- Homework project
  - Milestones are published, extra score if deadlines are met
  - Choice
  - Specification
  - Current state (AKA: Skeleton)
  - Final submission (13th week)
  - Face-to-face presentation is mandatory!
  - Plagisarism check for every submission, it is an individual work.
  - More details on the homepage

Recommended literature

Any book about Standard C programming language
in your own language

Chapter 2

Basic terms

Programming

How to make ham and eggs?

```
1 Bake some ham in hot vegetable oil
2 Add three eggs to it
3 ...
```

How to bake the ham in oil?

```
1 Get a frying pan
2 Put it on the stove (fire)
3 Add some vegetable oil to it
4 Bring it to boiling
5 Put some ham in it
6 Wait until the ham gets a bit brown
```

How to bring the oil to boiling?

```
1 Light the fire
2 Wait a little bit
3 Is the oil hot enough?
4 If not, go back to line 2
```

The impreative programming paradigm

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Programming

We tell the computer what to do

Programming paradigms

These are the principles, that we use to create the program

- Imperative programming ← This is what we learn
- Functional programming
- Object-oriented programming
- etc. . .

Imperative programming

We tell the computer step-by-step, what to do

- by defining an algorithm

The process of programming

We will always take these steps during the course:

1. We describe the task
2. We construct an algorithm for solving the task
3. We create the program – we create the code of the algorithm

The process of programming

In more details:

1. We describe the task
2. We give an exact specification of the task
3. We select the right data structure for modelling the problem
4. We construct an algorithm for solving the task
5. We select an effective programming language for coding (in this course: C)
6. We create the code of the algorithm (coding)
7. We test the program

Algorithm

Algorithm (method)

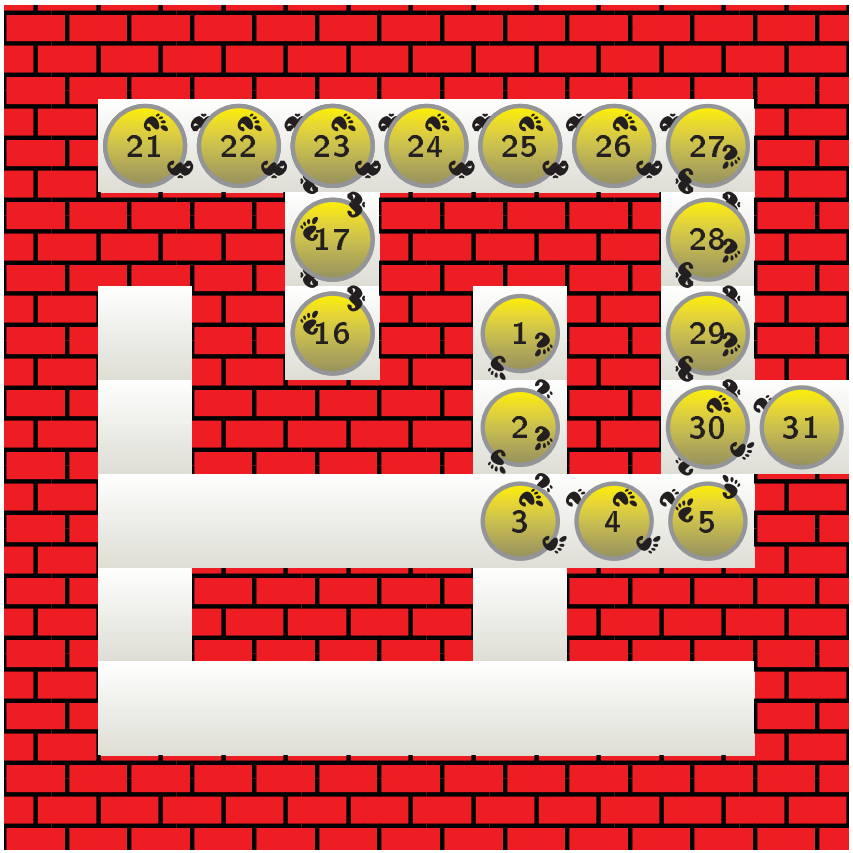
A finite sequence of steps, that can be performed mechanically and leads to the solution

- Before coding we check if the algorithm
  - right – it gives solution to our problem (and not to something else)
  - complete – it gives solution in all possible cases
  - finite – it will end in finite number of steps
- It is not enough to try, you also have to prove it!

Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

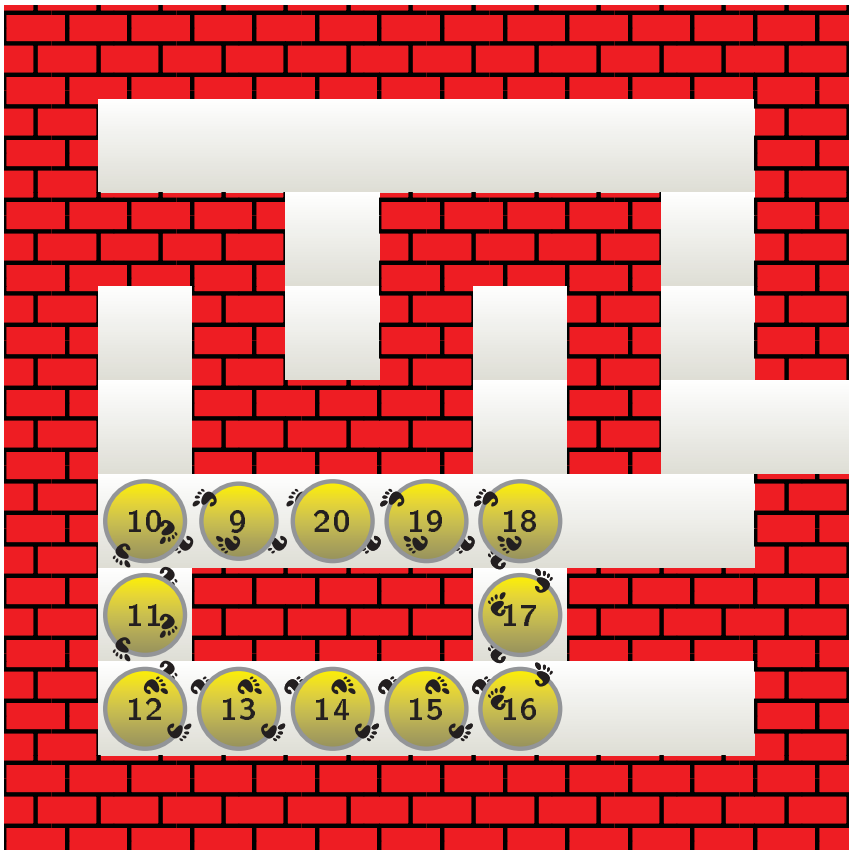
- Task: Escape from the dark maze (labyrinth)
- Solution: Push the left shoulder to the wall, and walk forward until you get out.



Algorithms

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Task: Escape from the dark maze (labyrinth)
- Solution: Push the left shoulder to the wall, and walk forward until you get out.



Algorithms – examples

- Even if the algorithm is right (correct), complete (gives solution if exists) and finite, it might be not manageable (not tractable)
- It is important to be finite also in practice, which means
  - should be finished within acceptable time
  - should work with reasonable amount of data

Description of algorithms

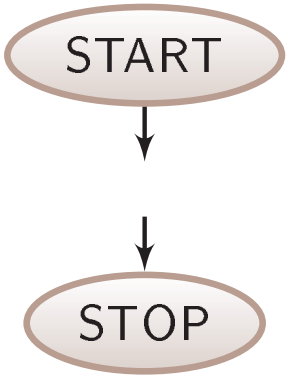
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pseudo-code is a language independent way of describing algorithms
- it is written in a normal (human) language, but it is constructed precisely

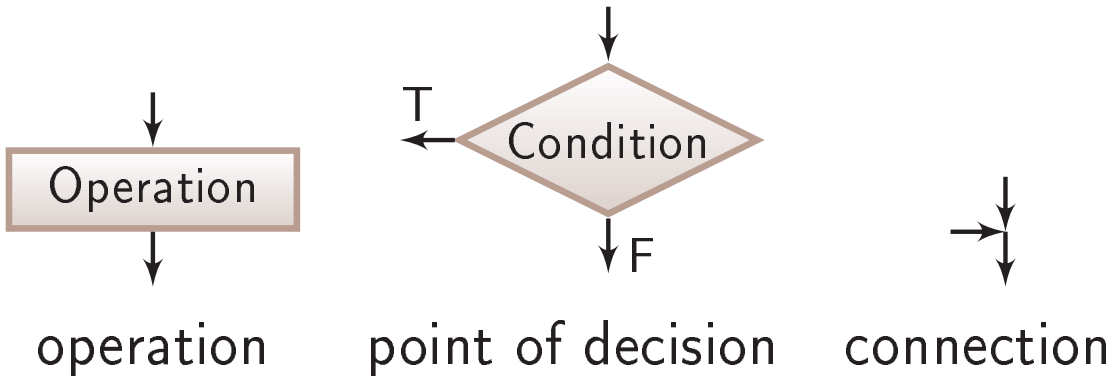
```
1  Get a frying pan
2  Put it on the stove (fire)
3  Add some vegetable oil to it
4  Light the fire
5  Wait a little bit
6  Is the oil hot enough?
7  If not, go back to line 5
8  Put some ham in it
9  Wait until the ham gets a bit brown
10 Add three eggs to it
```

Description of algorithms

- Flow-chart is a tool for describing algorithms in a graphical way
- The flow-chart of a program with one input and one output is placed between START and STOP elements

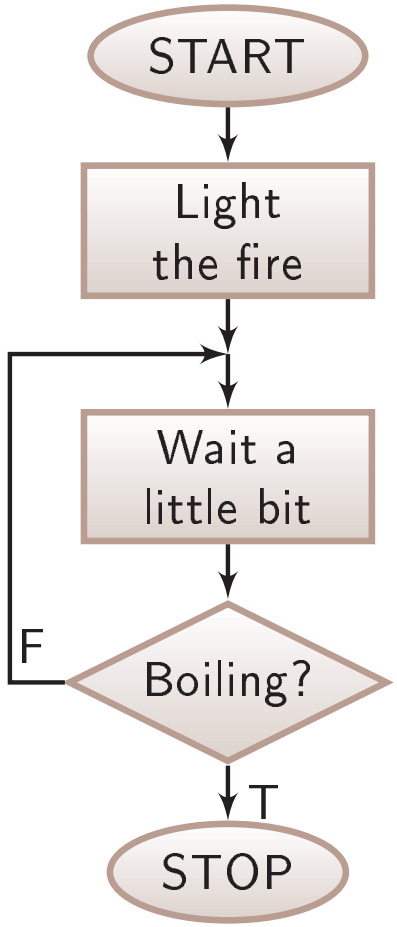


- A flow-chart consists of the following elements



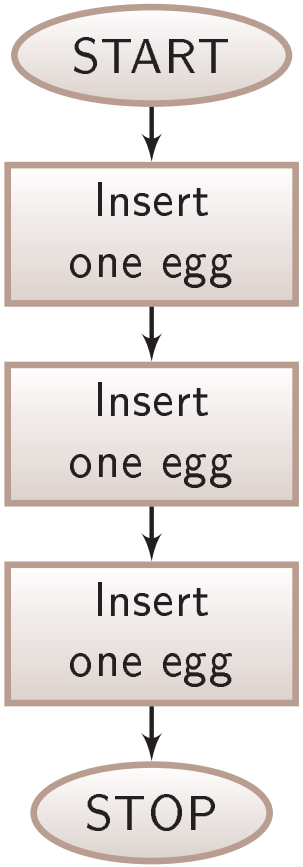
Flow-chart – example

- Construct the flow-chart of boiling water



Flow-chart – example

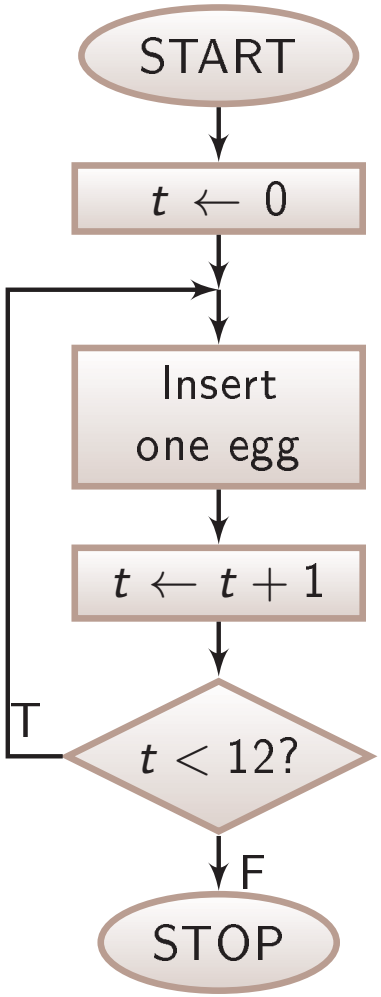
- How do we insert 3 eggs?



Flow-chart – example

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- How do we insert 12 eggs?
- Let's use t for counting the number of eggs inserted!



What is data?

- Algorithm works on data, with data

Data

Data is everything from the outside world that we somehow represent and store on our computer.

- The data has
  - type (number, letter, colour, . . .)
  - value
- The data determines
  - the set of values the data may have
  - the operations that can be performed on the data

Types – examples

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

type	values	operations
number	$0, -1, \mathrm{e}, \pi, \ldots$	addition, subtraction, ..., comparison, sorting
character	a, A, b, γ, ...	comparison, sorting
logical	{true, false}	negation, conjunction (AND), disjunction (OR)
colour	red, blue, ...	comparison
temperature	cold, warm, hot, ...	comparison, sorting

Expressions

Expression

We can form expressions from constants and variables by using the appropriate operations

- An expression can be evaluated, it has type and value.
- The operations are determined by the operators, the operators work on the operands.
- Examples for expressions

expression	type	value	remark
$2 + 3$	number	5	
$-a$	number	-3	if $a = 3$
$2 * (a - 2)$	number	2	if $a = 3$
true AND false	logical	false	

Expressions

- The type of the expression is not always (not necessarily) the same as the type of the operands. Mixed expressions:

expression	type	value	remark
$2 < 3$	logical	true	
$(a - 2) \neq 8$	logical	true	if $a = 3$

- There are strict rules for forming expression (syntax). Wrong (not interpretable) expressions:

expression	error
$3/$	binary operator (/) with one operand
$\text{red} < 2$	colour $<$ number
$3 \cdot \text{warm}$	number $\cdot$ temperature
$(2 < 3) + 5$	logical $+$ number

Programming languages

Programming languages

Mathematical formalism that can be interpreted by the computer

- It is similar to spoken languages, in order to be easily understandable and to be easily constructed
- Small vocabulary, very strict grammar (syntax)

Syntax and semantics

- Syntax error (grammatical error)
  - We don't follow the rules of the programming language, the program is not interpretable, it is not executable.
  - Syntax errors are easy to detect.
  - In most of the cases it can be corrected easily, quickly.
- Semantic error (interpretation error)
  - The program is executable, it performs something, but it does not do exactly what we have specified.
  - Semantic error is typically hard to detect and hard to correct.
  - Program testing is a profession.

Chapter 3

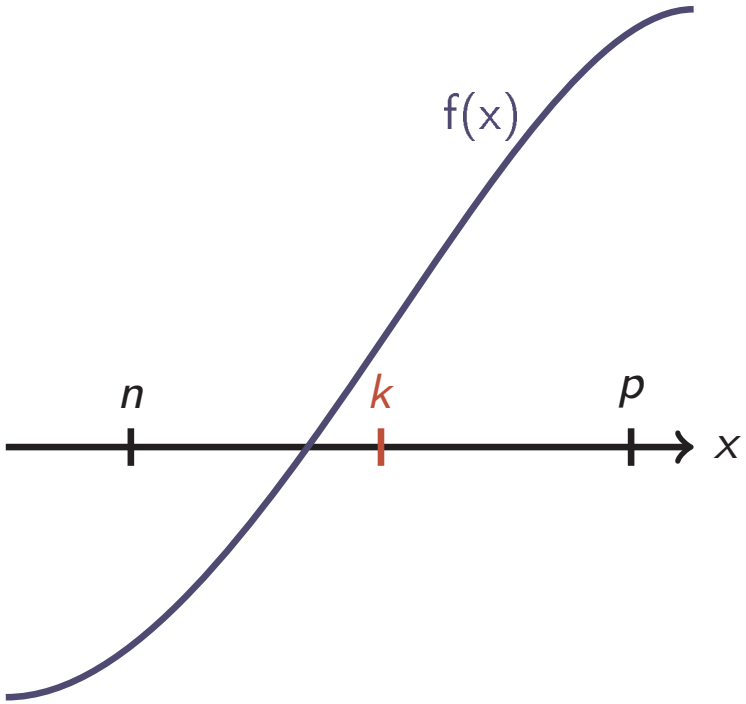
Structured programming

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.

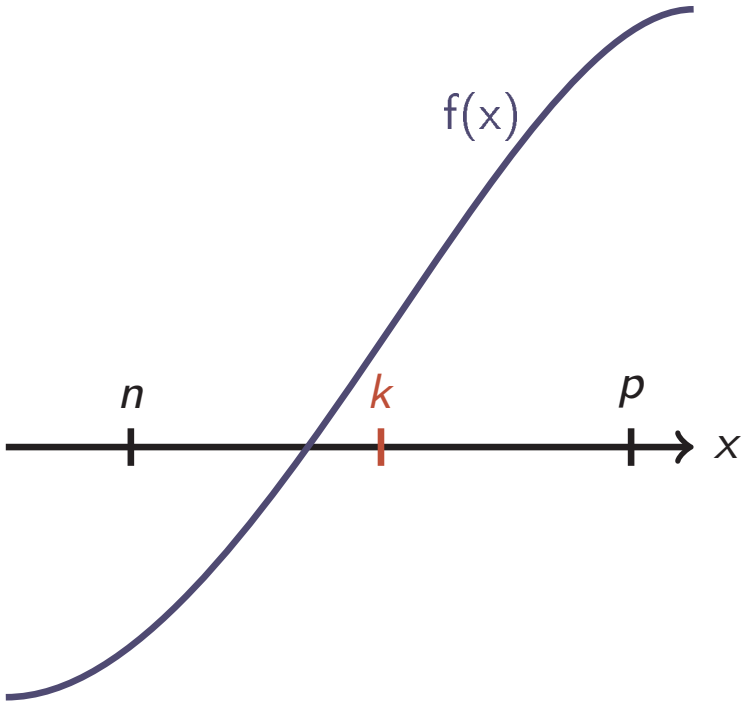



```
1  p-n < eps?
2  IF TRUE, JUMP TO 10
3  k ← (n+p) / 2
4  f(k) < 0?
5  IF TRUE, JUMP TO 8
6  p ← k;
7  JUMP TO 1
8  n ← k;
9  JUMP TO 1
10 The zero is: n
```

Algorithms

Finding zeros – a different approach

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




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

Structured vs unstructured

- Two programs of the same algorithm

```
1  WHILE p-n > eps, repeat
2    k ← (n+p) / 2
3    IF f(k) > 0
4      p ← k;
5    OTHERWISE
6      n ← k;
7  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
```

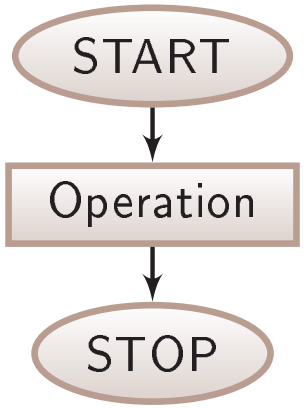
- Structured program
  - easy to maintain
  - complex control
  - higher level
- Unstructured program
  - spaghetti-code
  - easy control
  - ”hardware-level”

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

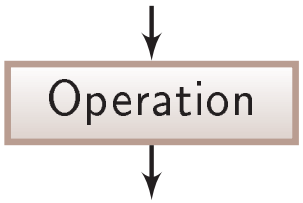


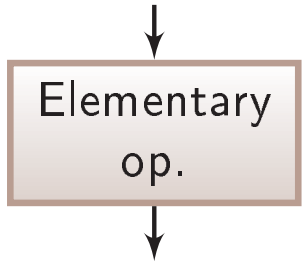
- All structured programs follow this simple scheme:
- The structure of the program is determined by the inner structure (layout) of Operation.
- Operation can be:
  - Elementary operation (action)
  - Sequence
  - Loop or repetition
  - Selection

Elements of structured programs

Elementary operation

that cannot be further expanded






```
Elementary op.
```

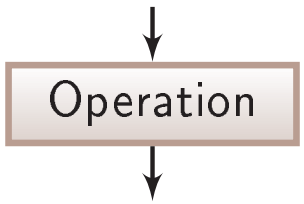
- The empty operation (`don't do anything`) is also an elementary operation

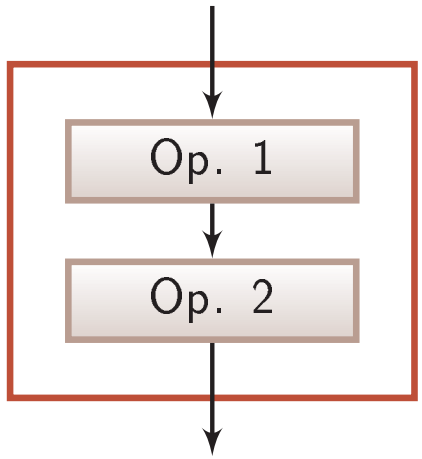
Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Sequence

Execution of two operations after eachother, in the given order

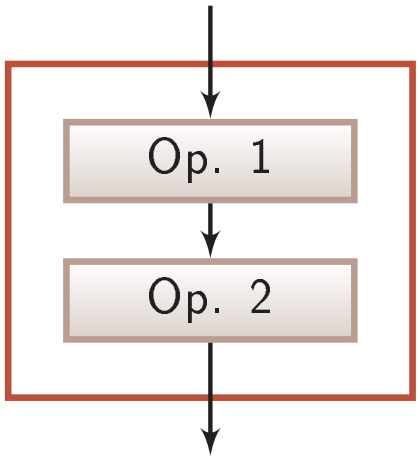


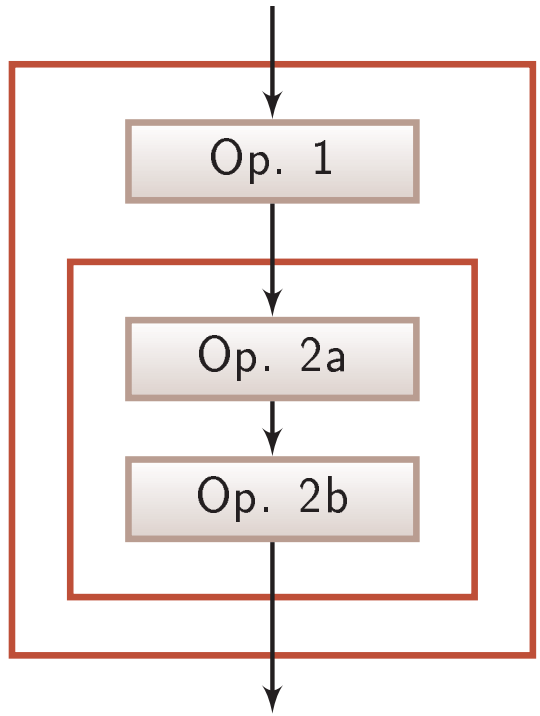



```
Op. 1
Op. 2
```

Elements of structured programs

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





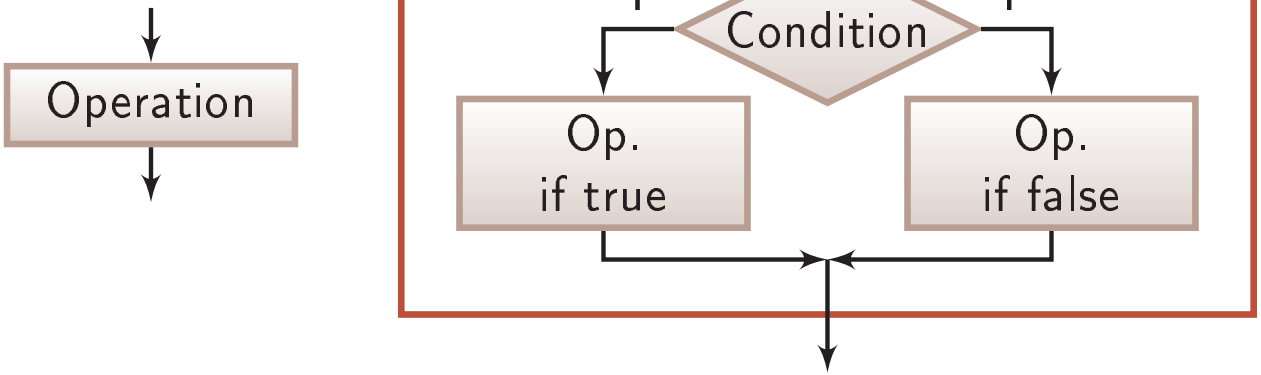
- The expansion can be continued, so a sequence can be an arbitrary long (finite) series of operations.

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Condition-based selection

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

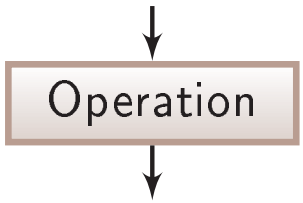


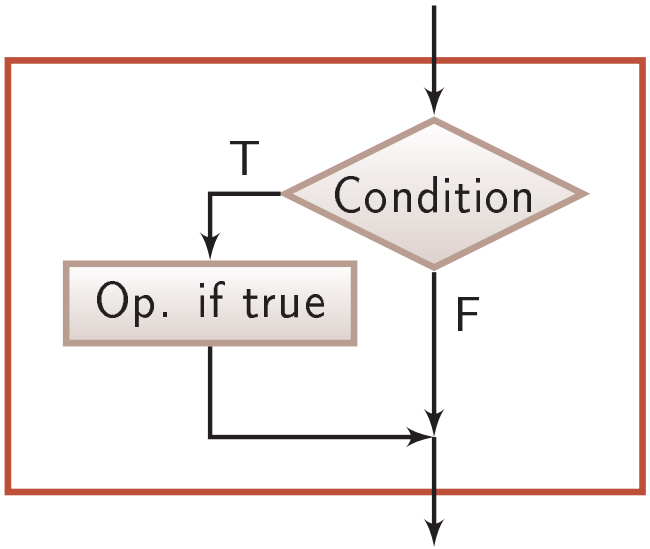

```
IF Condition
  Op. if true
ELSE
  Op. if false
```

Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- One of the branches can also be empty.





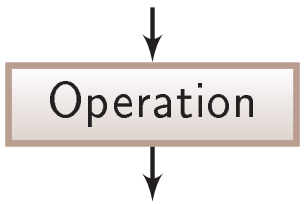

```
IF Condition
  Op. if true
```

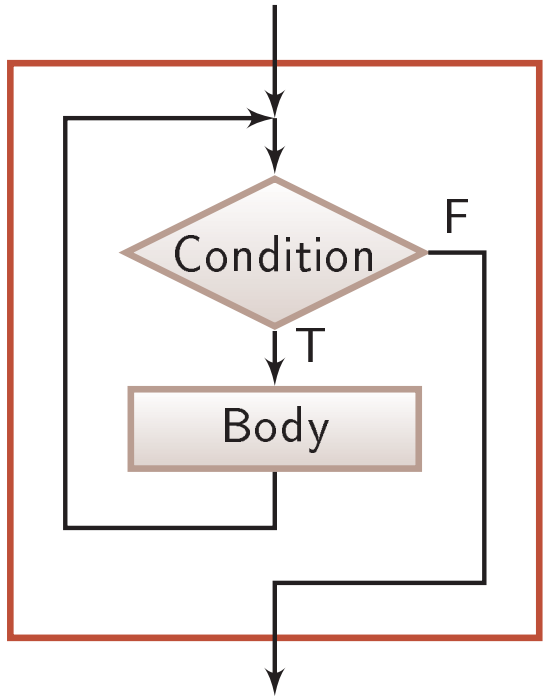
Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Top-test loop

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






```
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.

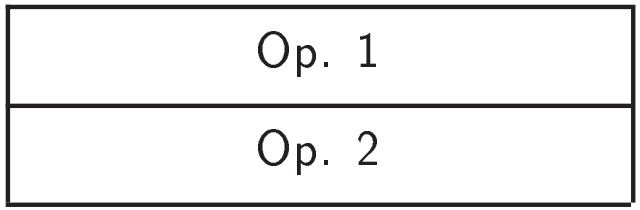
The structogram

- The flowchart
  - a tool for describing unstrutured programs
  - can be 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

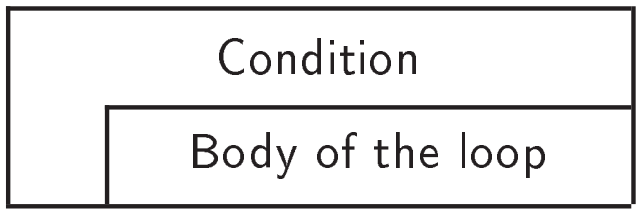
The structogram

- The program is a rectangle

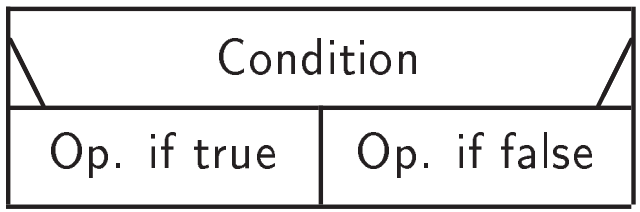
- it can be expanded into more rectangles with the elements below
  - Sequence



  - Top-test loop



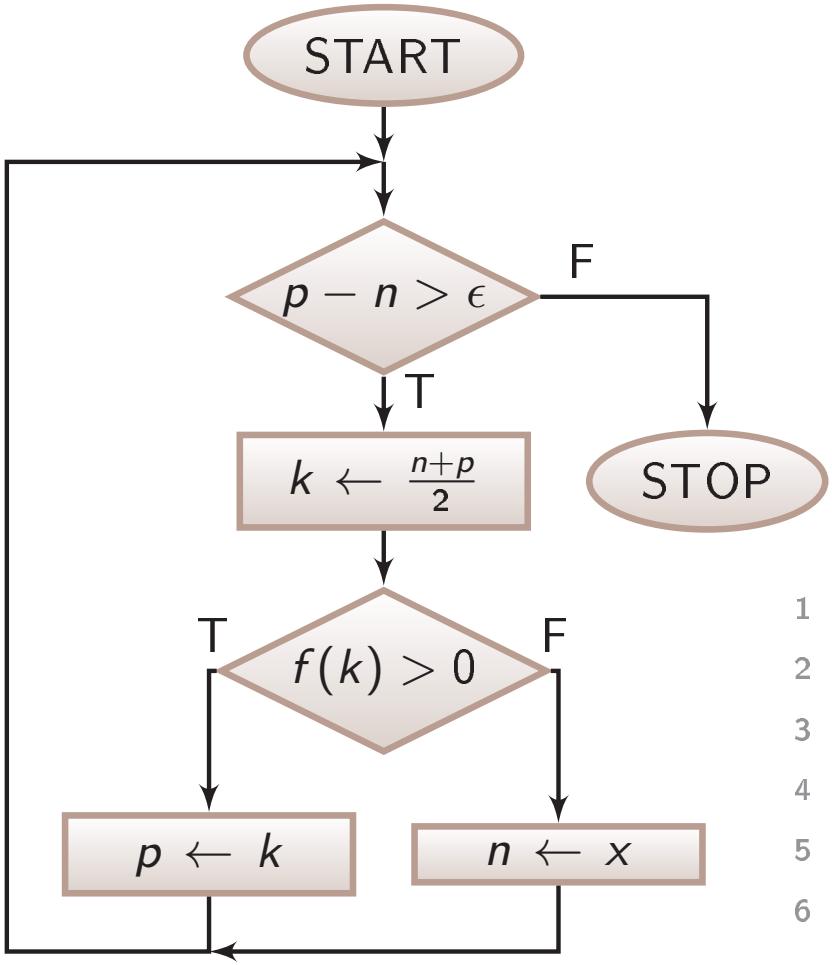
  - Selection

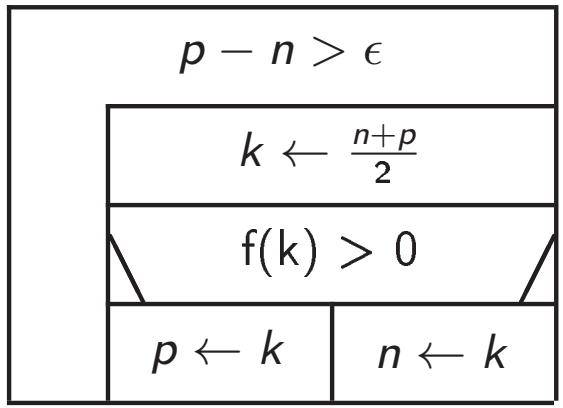


The structogram

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

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






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

Chapter 4

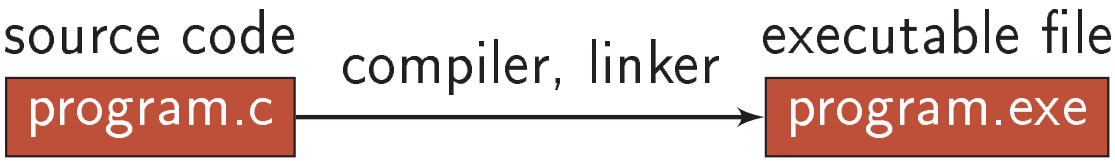
C language basics

Short history of C programming language

- 1972: Start of development at AT&T Bell Labs
  Most of the UNIX kernel was created in C
- 1978: K&R C – Brian Kernigham, Dennis Ritchie:
  The C Programming Language
- 1989: Standardization: ANSI X3.159-1989
- 1999: C99-standard:
  new data types (complex)
  international character encoding
  arrays with variable sizes
  ...
- 2007–: C1X standard, 2011: C11 standard
  C++ compatibility
  multi-thread programs
  ...

Main features of C

- Compiled language



- "small language": few (10) instructions, a lot of (>50) operators
- concise syntax ("zipped")
  - hard to read (must pay attention)
  - easy to make a mistake
  - hard to find a mistake
- it gives a code that can be optimized efficiently and runs fast
- easy to implement for different platforms

The first C program

- The source code of the minimum-program

```
1  /* first.c -- The first program */
2
3  int main()
4  {
5    return 0;
6  }
```

link

- The program starts, and after that it ends (finishes its run)
- between /* and */ there are comments: messages for the programmer
- `int main()` – All C programs starts like this
  - `int Main()` – and not like this. C is "case sensitive"
- `{ }` – block, it encloses the program body
- `return 0;` – It marks the end of the program

The first C program

- . . . that actually does something

```c
/* Helloworld.c -- My first program */
#include <stdio.h> /* needed for printf */

/* The main program */
int main()
{
  printf("Hello world!\n"); /* Printing */
  return 0;
}
```

link

- After compiling and running it gives the following output:

```
Hello world!
```

- `#include` – to insert other C program parts
- `printf` – printing, `\n` – new line (line feed)

A more complicated one

- Instructions in a sequence

```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!
```

Printing the value of a variable

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
#include <stdio.h>
int main()
{
  int n;          /* declaring an integer var., called n */
  n = 2;          /* n <- 2 assignement of value */
  printf("The value is: %d\n", n); /* printing */
  n = -5;         /* n <- -5 assignement of value */
  printf("The value is: %d\n", n); /* printing */
  return 0;
}
```

link

```
The value is:  2
The value is:  -5
```

- `int n` – declaration of variable.
  `int` (integer number) is the type, `n` is the identifier
- `n = 2` – assignement of value, variable `n` takes value of expression "2"

. . . continued


```c
#include <stdio.h>
int main()
{
  int n;          /* declaring an integer var., called n */
  n = 2;          /* n <- 2 assignement of value */
  printf("The value is: %d\n", n); /* printing */
  n = -5;         /* n <- -5 assignement of value */
  printf("The value is: %d\n", n); /* printing */
  return 0;
}
```

link

- `printf(<format>, <what>)` – printing the value of expression `<what>` in the given `<format>` format
  - `%d` – decimal (decimal number system)

Block and declaration

Structure of the block

```
{
  <declarations>
  <instructions>
}
```

```c
{
  /* declarations */
  int n;

  /* instructions */
  n = 2;
  printf("%d\n", n);
}
```

Block and declaration

Structure of declaration

<type name> <identifier> [= <initial value>]$_{opt}$;

```
1  int n;                          /* not initialized */
2  int number_of_dogs = 2;         /* initialized */
```

- value of n is garbage from memory at the beginning
- value of number_of_dogs is 2 at the beginning

Inputting data

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
/* square.c -- square of a number */
#include <stdio.h>
int main()
{
  int num;       /* declaring an integer var. */
  printf("Please give an integer value: "); /* info */
  scanf("%d", &num);                        /* inputting */
  /* printing the value of 2 expressions */
  printf("The square of %d is: %d\n", num, num*num);
  return 0;
}
```

link

```
Please give an integer value:  8
The square of 8 is:  64
```

- scanf(<format>, &<where to>) –
  Inputting (scanning) data in <format> format and putting it into <where to> variable

Inputting data

- This is another option, that gives the same result.

```
1 #include<stdio.h>int main(){int num; printf
2 ("Please give an integer value: ");scanf("%d",
3     &num);printf("The square of %d is: %d\n",
4             num,num*num);return
5                     0;}
```

link

- Not recommended, unmanagable

Thank you for your attention.