# C2110 *UNIX and Programming*

## 5th lesson

## Programs *vs* scripts, algorithms, bash

Petr Kulhánek

kulhanek@chemi.muni.cz

National Centre for Biomolecular Research, Faculty of Science,
Masaryk University, Kamenice 5, CZ-62500 Brno

## Test I

# Test I

- **Test through ROPOT in IS** (ROPOT = Revision, Opinion Poll and Testing)

**Student - ROPOT - C2110 - Test 1c**

**Time limit - 20 minutes.**

**Only one set of questions can be built.**

**Continuously save your answers.**

**Evaluation can be done only once.**

**It is allowed and recommended:**

- to test commands in the terminal.
- to search in the manual pages, in your notes and presentations of the course.
- when in doubt, ask the teacher.

**It is not allowed:**

- to communicate with another person except the teacher.

# Contents

- **Programs vs. Scripts**
  - **compiled vs. interpreted languages, examples**
- **Introduction to Programming**
  - **algorithms, algorithm notation, data structures, operations, input/output operations**
- **Bash**
  - **interactive vs. non-interactive mode, direct and indirect running of scripts**

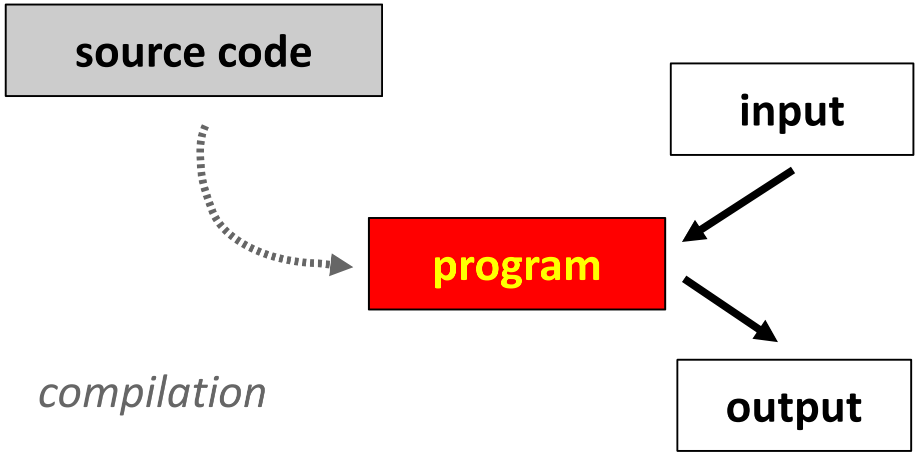
# Programs *vs* Scripts

# Programs *vs* Scripts

The **program** is a set of machine instructions processed directly by the processor. The program is obtained through the compilation of the source code written in a programming language.

**Compiled languages:**

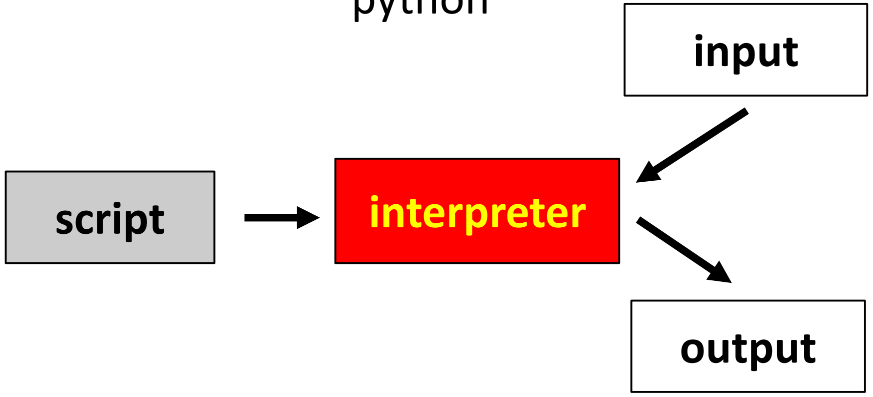
**C/C++**
**Fortran**



The **script** is a text file containing commands and control sequences that are executed by interpreter of the scripting language.

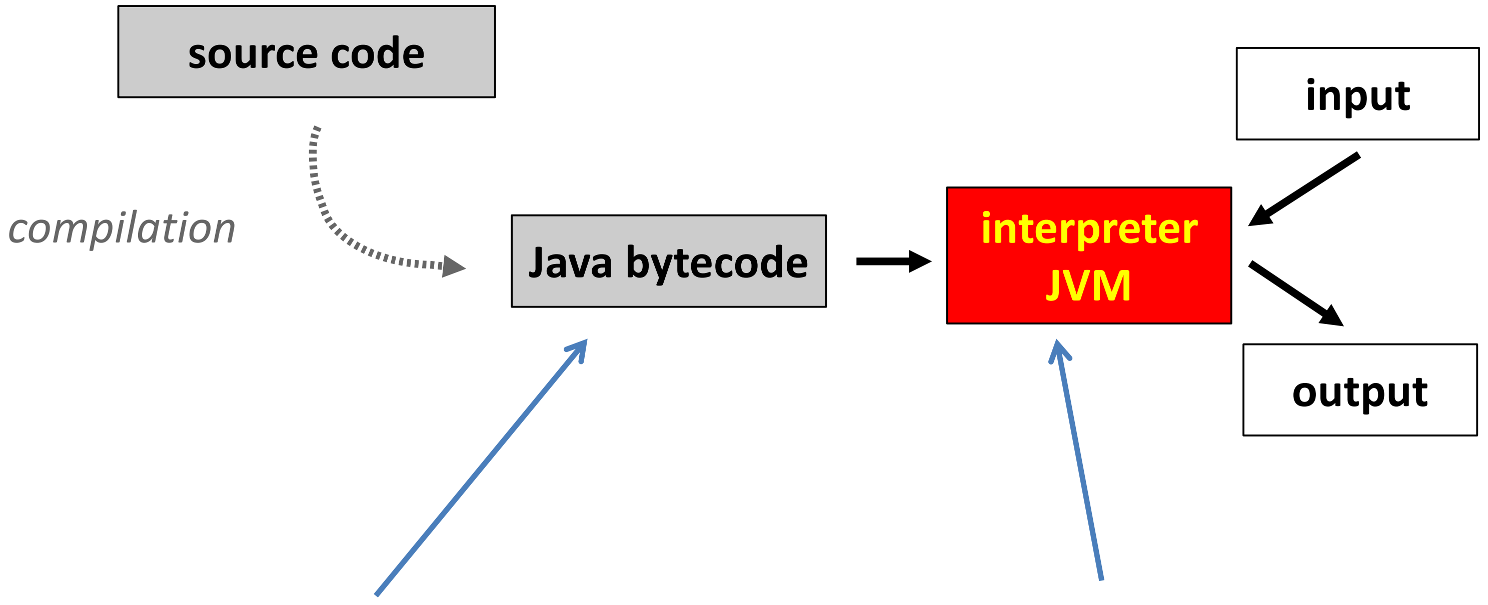
**Scripting languages:**

**bash**
**gnuplot**
**awk**
JavaScript
PHP
python



## What about JAVA?

There are also various combinations of these two approaches, for example the Java programming language.



**Java bytecode**
(platform independent - OS and HW)

**Java Virtual Machine**
Java bytecode interpreter
(program compiled for a given platform - OS and HW)

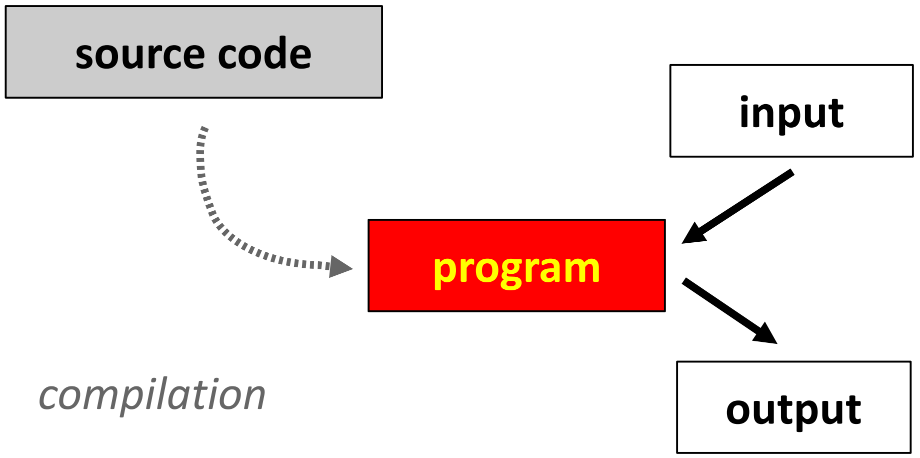
# Programs *vs* Scripts ...

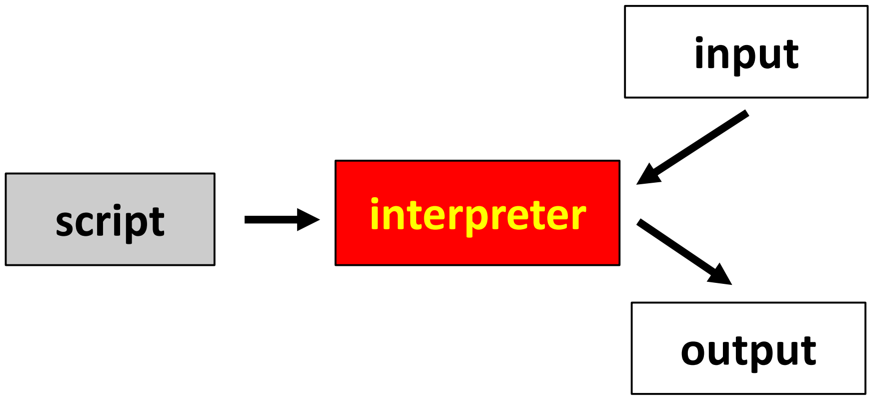
- easy optimization
- fast execution

- recompilation necessary
- not possible to create self-executable code

- recompilation is not required
- possible to create self-executable code

- problematic optimization
- slow execution





# Program in C

## Source code

```
#include <stdio.h>

int main(int argc,char* argv[])
{
  printf("This is program in C language!\n");
  return(0);
}
```

## Compilation

```
$ gcc program.c -o program
```

compiler of C language

name of file with compiled program

## Launch of program

```
$ ./program
```

File **program** must have permission for **execution**

## Script in bash

**Script**

```
#!/bin/bash

echo "This is script in Bash interpreter!"
```

**Launch of script**

```
$ bash script.bash
```

File **script.bash** does not need permission for **execution**

Bash interpreter

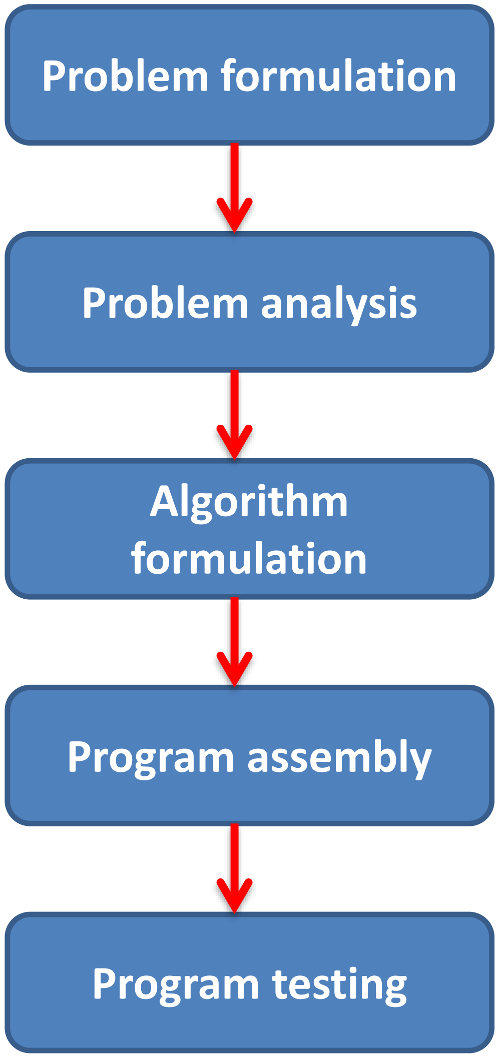
# Exercise I

1. Create directories called task01 and task02 in your home directory.
2. Save the program.c file into task01 and the skript.bash file into task02. The source of both files is
   /home/KULHANEK/Documents/C2110/Lesson05/programs
3. Compile the source code of a program written in C. Verify that the compiled program can be executed.
4. What is the size of the file containing the compiled program? Open the resulting file in a text editor (gedit). What does it contain?
5. Verify functionality of the script skript.bash by executing it.
6. Change functionality of program.c  and skript.bash, so they will print different output.

## Basics of Programming

https://cs.wikipedia.org/wiki/Algoritmus

# Algorithmization



**Problem formulation**
It is necessary to formulate requirements, specify the input data, desired results and the accuracy of the solution (for numerical task).

**Problem analysis**
Verification that the task is feasible for the expected input data. The most appropriate solution is proposed according to the nature of the task.

**Algorithm formulation**
Assembly of unambiguous sequence of operations that have to be performed in order to properly resolve the task.

**Program assembly**
Source code of the program in chosen programming language is created on the basis of the algorithm.

**Program testing**
Finding syntax errors in the source code and logic errors in design of the algorithm. Testing program functionality on the specified data.

# Algorithm

**Algorithm** is the precise manual or procedure for solving given type of problem. The term algorithm is mostly used in programming, where it describes theoretical principle of solving the problem (as opposed to the exact procedure in a particular programming language). In general, algorithm can appear in any other scientific discipline.

**Required properties:**

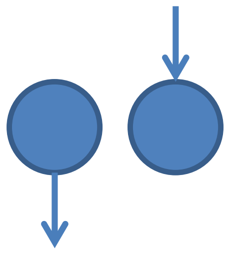
- **Determinism** - algorithm must be accurate, exact and unambiguous, i.e. next step is clearly determined at each point and the same input data must provide the same results. (Operation of algorithm must not depend on the goodwill of the person or the characteristics of the machine, where it was implemented).
- **Generality** - the algorithm is not used to solve just one task, but it is solution for whole group of jobs, which differ only by the input data. Input data can in some range vary.
- **Finiteness** - searched results must be obtained after a finite number of steps, algorithm must finish after a finite number of steps.

**Notation of algorithm:**

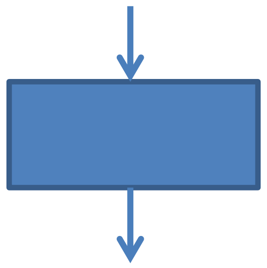
- words
- pseudocode
- graphic (flowcharts, etc.)

# Flowchart

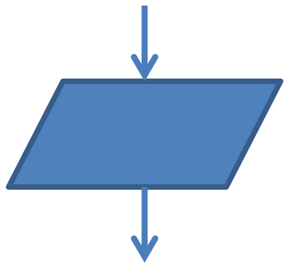
The flowchart is a graphical representation of an algorithm. The flowchart consists of blocks that are executed in the order given by the direction of flow lines from the beginning to the end.



terminal (start, end)

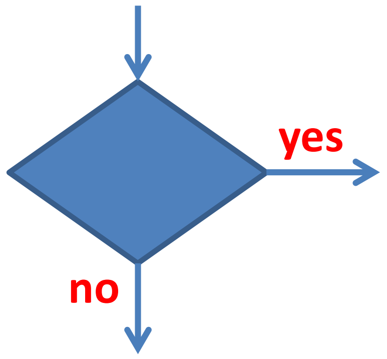


processing block



input/output block

Blocks are combined to unambiguously describe the sequence of operations that must be executed to correctly solve the task. Operations or groups of operations to be executed are written into individual blocks.

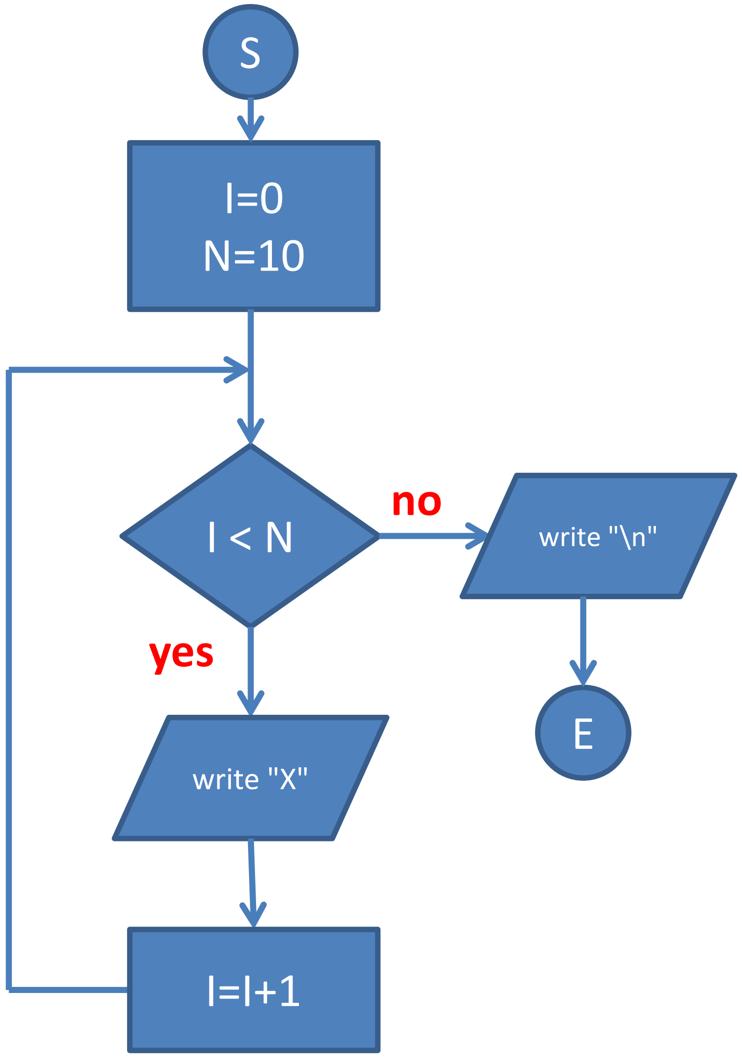


decision making block

Additional blocks exist.

# Examples

**Flowchart:**

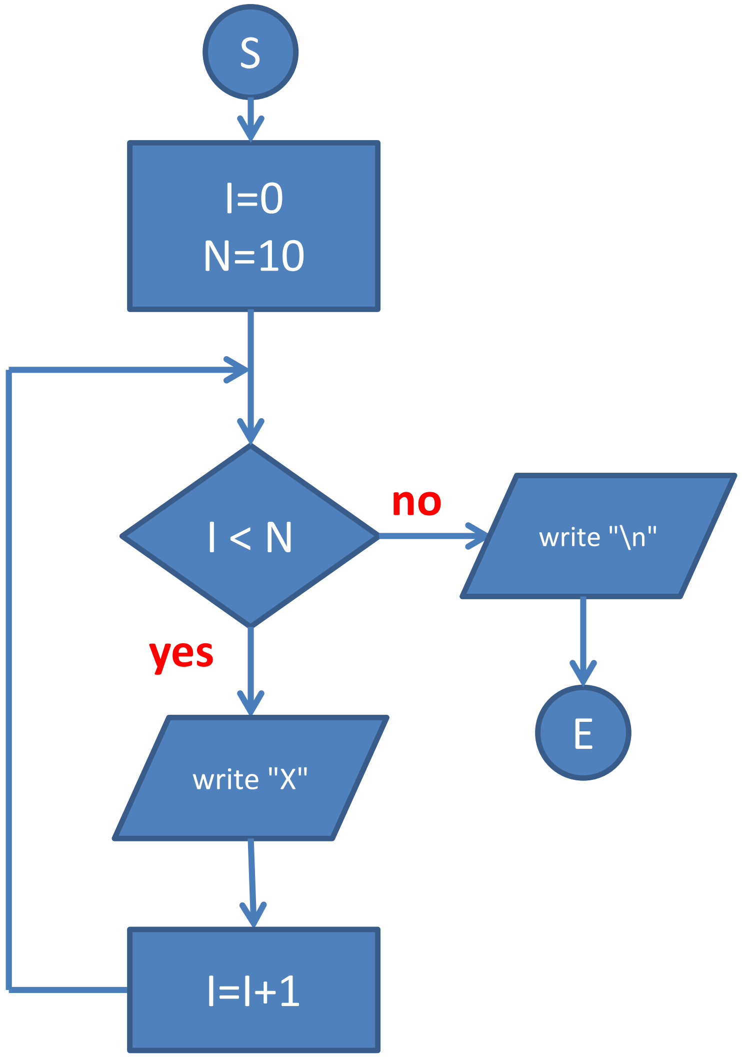


**Word description:**

1. Insert value 0 into variable I
2. Insert value 10 into variable N
3. Is the value of the variable I lower than N
   YES – Continue with step 4
   NO – Continue with step 7
4. Print character X.
5. Increase the value of the variable I by one.
6. Continue to step 3
7. Print end of the line
8. End

# Examples

## Algorithm



## Script in bash

```
#!/bin/bash

I=0
N=10

while [ $I -lt $N ]; do
     echo -n "X"
     ((I=I+1))
done
echo ""
```

**Result:**

```
$ ./my_script
XXXXXXXXXX
$
```

# Data Structures

Data structures are used for storing data. The basic data structures include:

a) **variable**
b) array
c) record
d) object

Variable is **named location** in the memory that **contains a value**. Each variable is of certain **type**, which limits possible operations with the variable. Type of the variable can be given when creating the variable (explicit type) or can be determined only when the variable is used (implicit type). The variable type affects the way of storing the data in the memory of a computer.

**Examples:**

| | |
|---|---|
| `A="magical phrase"` | text (string) |
| `B=5` | integer |
| `C=10.458` | real number (with floating point) |

```
D=B+C
```

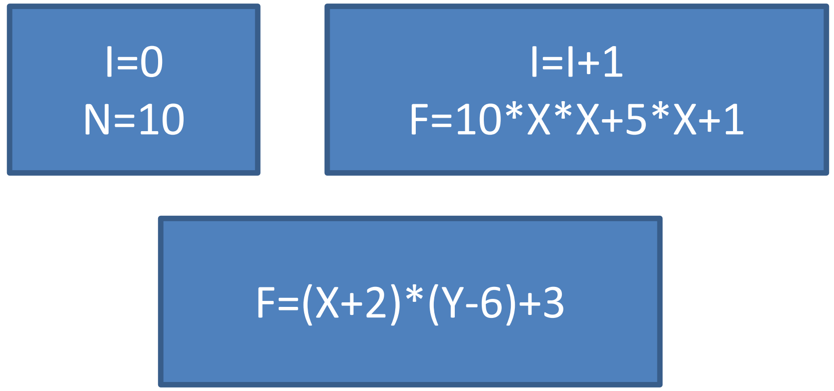
~~`E=A+B`~~

# Operations

processing block

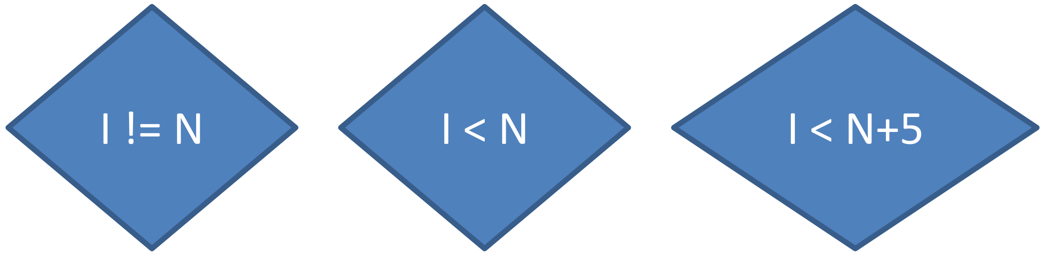
**Basic operation:**

| | |
|---|---|
| `=` | assignment |
| `+` | sum |
| `-` | difference |
| `*` | multiplication |
| `/` | division |



**Logic operation:**

| | |
|---|---|
| `==` | equal |
| `!=` | not equal |
| `<` | lower |
| `<=` | lower or equal |
| `>` | greater |
| `>=` | greater or equal |



decision making block

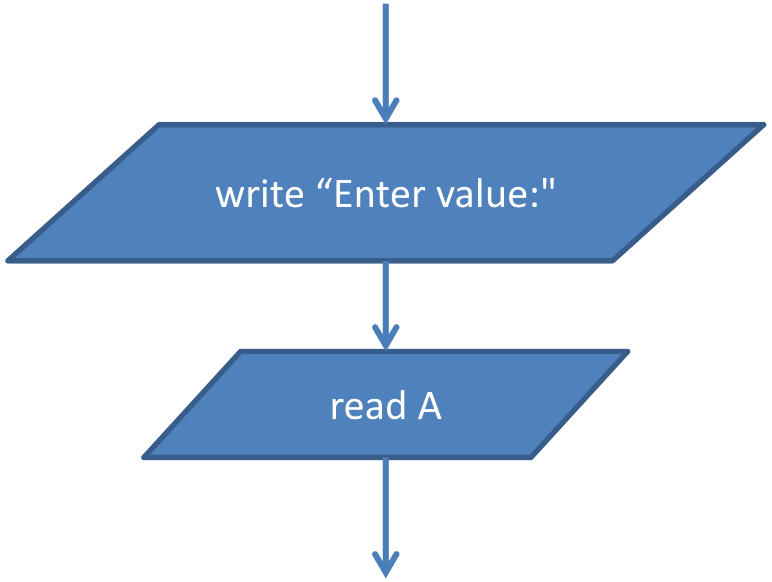
# Input

**Input** of a program can be an information entered by the user from the terminal, redirected from a file, or from another program by using pipes.

**Basic operations (pseudocode):**

`read var` Read value to variable var

**Example:**



program will print task for the user and will expects user’s input, which is then stored in the variable A.

Input defined in this way reflects the basic options of bash language. Usage of other input options is not recommend at this stage.

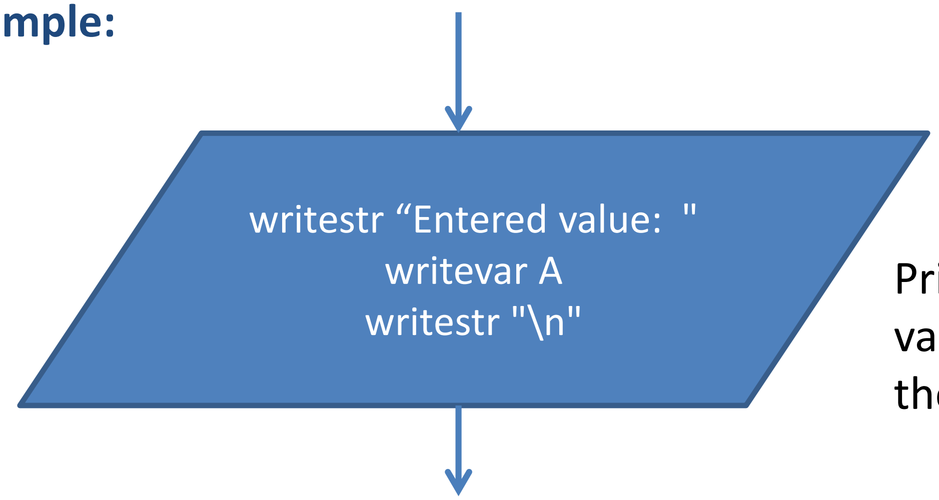
## Output

**Output** is printed to the terminal. The terminal acts as a printer, which can not take back an already printed character. In addition, the printer can go to the beginning of the new line by printing “\n” character to the terminal.

**Basic operations (pseudocode):**

| | |
|---|---|
| `writestr “string"` | prints characters (string) specified in quotation |
| `writevar var` | prints value of the variable var |

**Example:**



Prints text “Entered value" followed by value of variable A. Cursor will be move on the new line.

Output defined in this way reflects the basic options of bash language. Usage of other output options is not recommended at this stage.

# Bash

https://www.gnu.org/software/bash/

## Bash – Overview

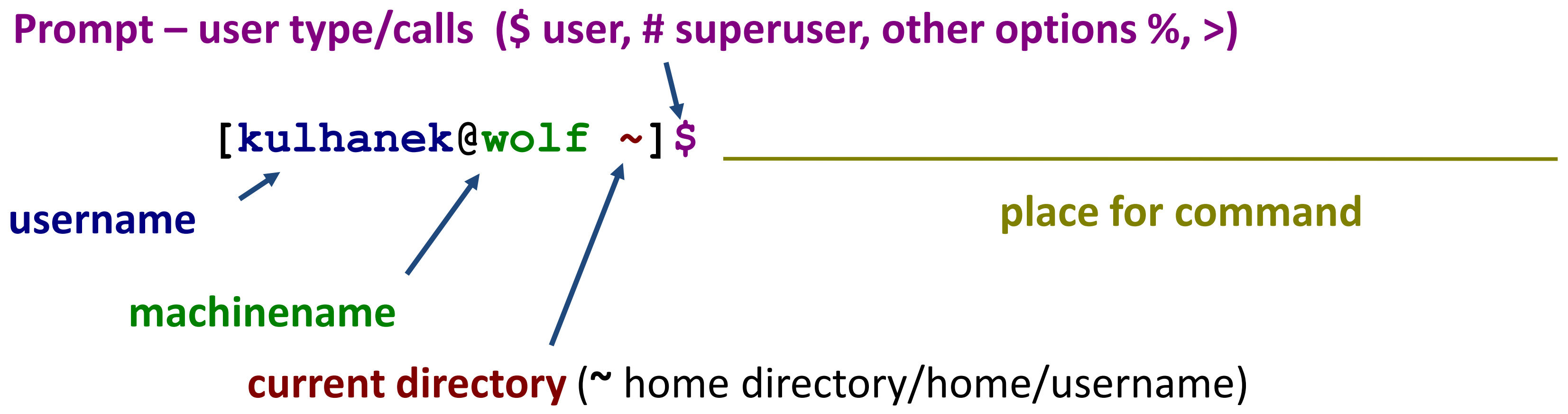
**Unix shell** (also command processor) is term of **text user interface**, which is the predecessor of the graphical user interface.
**Bash** is one of Unix shells.

**Bash** is POSIX shell with a number of extensions. It is designed for operating systems based on GNU project and can be run on the most Unix-like operating systems. It is used as implicit command interpreter in systems built on the Linux kernel, as well as in Mac OS X or Darvin system. It can be used in Microsoft Windows by using Subsystem for Unix Applications (SUA), or by POSIX emulation using software **Cygwin** and MSYS.

http://www.cygwin.com/

https://cs.wikipedia.org

# Interactive Mode



```
[kulhanek@wolf ~]$
```

Prompt – user type/calls ($ user, # superuser, other options %, >)
username
machinename
current directory (~ home directory/home/username)
place for command

Command is executed by pressing **Enter**.

**History:** list of previously used commands can be scrolled through by using the arrow keys up and down. A command from history can be reused or modified to use. History is also accessible by command **history.**

**Auto completion:** command line interpreter can try to complete itemized word by pressing Tab. It can be used to complete command names, paths and filenames (if first Tab does not give any output, more options exists and are displayed by repeated press of Tab).

Shell interprets (expands) **wild characters and other special characters**, before the command execution. Control structures of the bash language can be also run in the interactive mode.

Interactive mode is terminated by typing **exit**.

# Non-interactive mode

## 1) Indirect start

interpreter of the language is run with name of the script as argument

```
$ bash my_bash_script
```

Scripts do not have to have set flag x (executable).

## 2) Direct start

direct launch of script (shell automatically starts the interpreter).

```
$ chmod u+x my_bash_script
$./my_bash_script
```

Scripts have to have set flag **x** (executable) a **interpreter** is specified in the script.

```
#!/bin/bash

echo 'This is script in Bash interpreter!'
```

## Interpreter Specification

**Specification of the interpreter (first line of script):**

```
#!/absolute/path/to/script/interpreter
```

**Bash script**

```
#!/bin/bash

echo “This is Bash script!"
```

**Gnuplot script**

```
#!/usr/bin/gnuplot

set xrange[0:6]

plot sin(x)

pause -1
```

- If no script interpreter is specified during its direct start, system shell interpreter (bash) is used.
- Interpreter specified in the script is ignored during indirect start.
- It is recommended to always specify the interpreter in the script, because it is used by text editors to highlight syntax.

# Interpreter Specification, II

If the absolute path to the interpreter changes (e.g. when using software modules), it is possible to apply following structure:

```
#!/usr/bin/env interpreter
```

Interpreter must be in a directory defined in system variable PATH.

**Bash script**

```
#!/usr/bin/env bash

echo "This is script in Bash!"
```

**Gnuplot script**

```
#!/usr/bin/env gnuplot

set xrange[0:6]

plot sin(x)

pause -1
```

## Exercise II

1. Run the bash command in the terminal. What happened? Exit the second session with the exit command.
2. Find ID of a process, which is responsible for interpretation of the command line (use the ps command).
3. Terminate the process using the kill command (kill -9 PID, where PID is a process ID). What happened?
4. Run the script.bash script from the task02 directory directly.

# Conclusions

## Conclusions

- **Program** is a binary file directly executed by the processor. **Script** is a text file executed by an interpreter.
- **Algorithmization** is a way of transforming a task to a description (algorithm), which can be easily rewritten into a source code of a programming or scripting language. The basic principle is to **decompose the task into elementary parts** and to define unambiguous procedure of their execution.
- **Bash** is a Unix shell that interprets the command line and at the same time includes support for running scripts.

# Homework

- **Algorithmization**

# Homework - Instructions

1. Create flowcharts for the following tasks. In the flowcharts, use only blocks and operations including input-output, which are mentioned in this presentation.
2. Draw diagrams in an appropriate software:
   - e.g. program **dia** (sudo apt-get install dia)
   - online tools, e.g. **www.draw.io**
3. Upload the final diagrams into the Homework Vault L05-EN in IS. Use pdf format. The filename will be in the following format:

   SurnameL05x.pdf

   where x is the number of the task. Files that will be named in a different fashion, will not be controlled (automatic changes of name made by IS are allowed).
4. The deadline for submission is **November 5, 2018 23:59.**

**Recommendation:** Use decision-making block before processing block.

# Task 1

Print a square composed from **X** characters. Length of the square side is entered by user.

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

Ignore the fact, that this is not visually a square. Number of **X** characters per line and the number of lines must be the same.

# Task 2

Print a triangle composed from **X** characters in such a way that one leg of the triangle is placed on the left side and the other on the top. Length of triangle leg is entered by the user.

```
X X X X X X X X X X
X X X X X X X X X
X X X X X X X X
X X X X X X X
X X X X X X
X X X X X
X X X X
X X X
X X
X
```

# Task 3

Print a triangle composed from **X** characters in such a way that one leg of the triangle is placed on the left side and the other on the bottom. Length of triangle leg is entered by the user.

```
X
X X
X X X
X X X X
X X X X X
X X X X X X
X X X X X X X
X X X X X X X X
X X X X X X X X X
X X X X X X X X X X
```

# Task 4

Print a square outline composed from **X** characters. Length of the square side is entered by the user.

```
x x x x x x x x x x
x                 x
x                 x
x                 x
x                 x
x                 x
x                 x
x                 x
x                 x
x x x x x x x x x x
```

Ignore the fact, that this is not visually a square. Number of **X** characters per line and the number of lines must be the same.

## Task 5

Print a square outline and its diagonals composed from **X** characters. Length of the square side is entered by the user.

```
X X X X X X X X X X
X X             X X
X   X         X   X
X     X     X     X
X       X X       X
X       X X       X
X     X     X     X
X   X         X   X
X X             X X
X X X X X X X X X X
```

Ignore the fact, that this is not visually a square. Number of **X** characters per line and the number of lines must be the same.