C2110 *UNIX and programming*

Lesson 3 / Module 2

PS / 2020 Distance form of teaching: Rev2

Petr Kulhanek

kulhanek@chemi.muni.cz

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

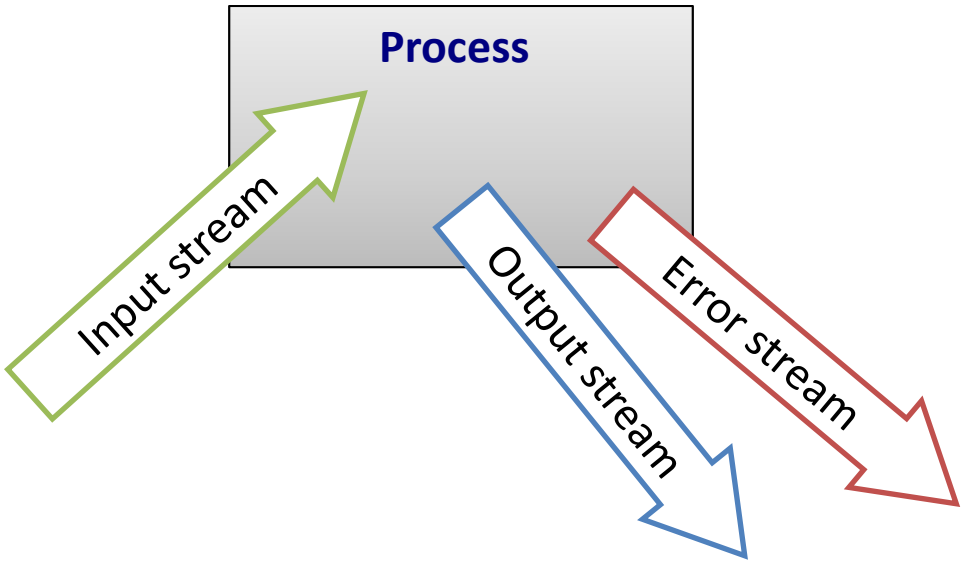
Streams, Redirects, Pipes

Process Communication

Process can communicate with the environment in a number of ways:

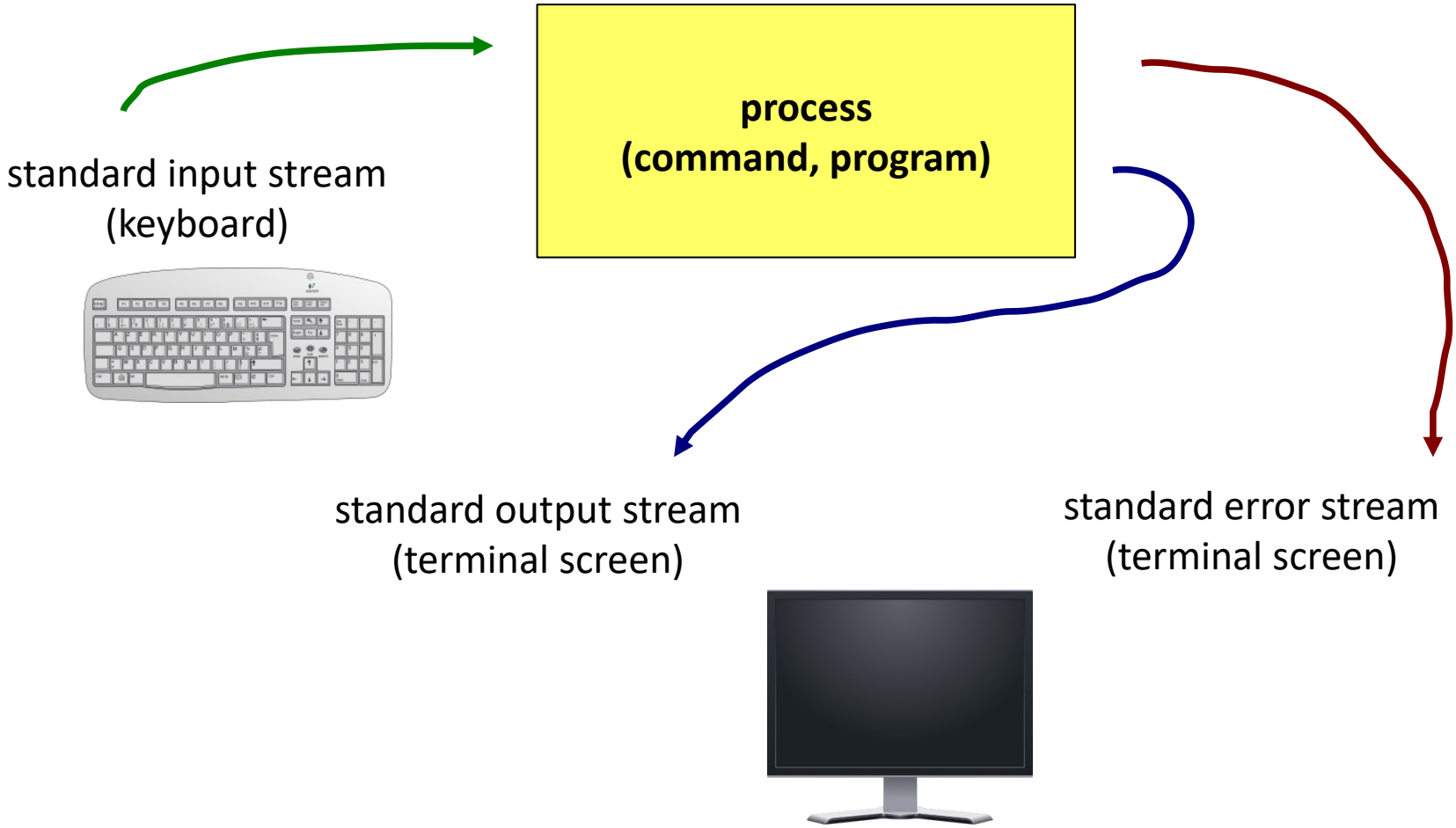
- GUI (Graphical User Interface = using the appropriate API)
- signals, shared memory, MPI (Message Passing Interface), etc.
- standard currents

One option is to read input data from **standard input current**, output data into **standard output** or **error stream**.



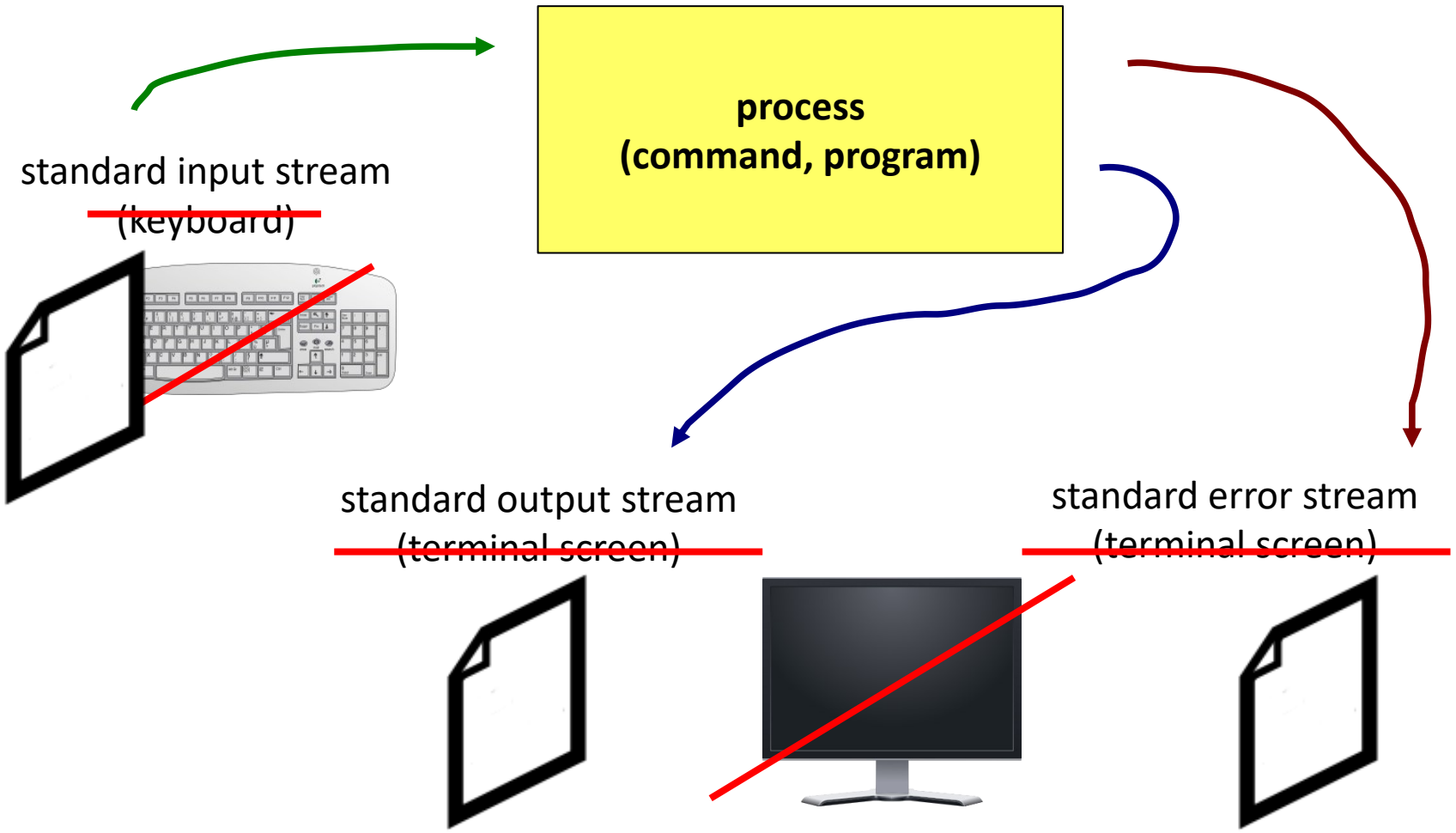
Standard Streams

Input-output streams serve the process for **communication** with its surroundings. Every process opens up **three standard streams**:



Redirection

Input-output streams can be redirected to use **files** instead of the keyboard or screen.



Redirection and Ending Input

Standard input redirection of program my_command from file **input.txt**.

```
$ my_command < input.txt
```

Standard input redirection of program my_command from script file.

```
.......
./my_command << EOF
first line text
second line text
third line text
EOF
......
```

mark indicating the end of the entry (selected by user)

text forming the read input

end of input, *mark **can not be** surrounded by spaces*

This method of redirection is especially useful in scripts, but it also works in the command line. The advantage is the expansion of variables in the read text.

Terminal (useful keyboard shortcuts):

Ctrl + D closes the input stream of the running process

Output Redirection

Standard output redirection of program my_command to a file **output.txt**. (File output.txt is created. If it already exists, its original content is **deleted**.)

```
$ my_command > output.txt
```

Standard output redirection of program my_command to a file **output.txt**. (File output.txt is created. If it already exists, output of the program my_command is **connected** to its end.)

```
$ my_command >> output.txt
```

Similar rules apply to standard **error** output, in this case the following operators are used:

```
$ my_command 2> errors.txt
$ my_command 2 >> errors.txt
```

Joining Output Streams

Standard output **and** standard program error output of my_command program can be redirected at the same time to a file **output.txt**.

```
$ my_command &> output.txt
```

```
$ my_command &>> output.txt
```

works in new versions bash

Alternative solutions for &>>: First, it is necessary to **redirect** standard output and then **join** standard error output with standard output.

```
$ my_command >> output.txt 2>&1
```

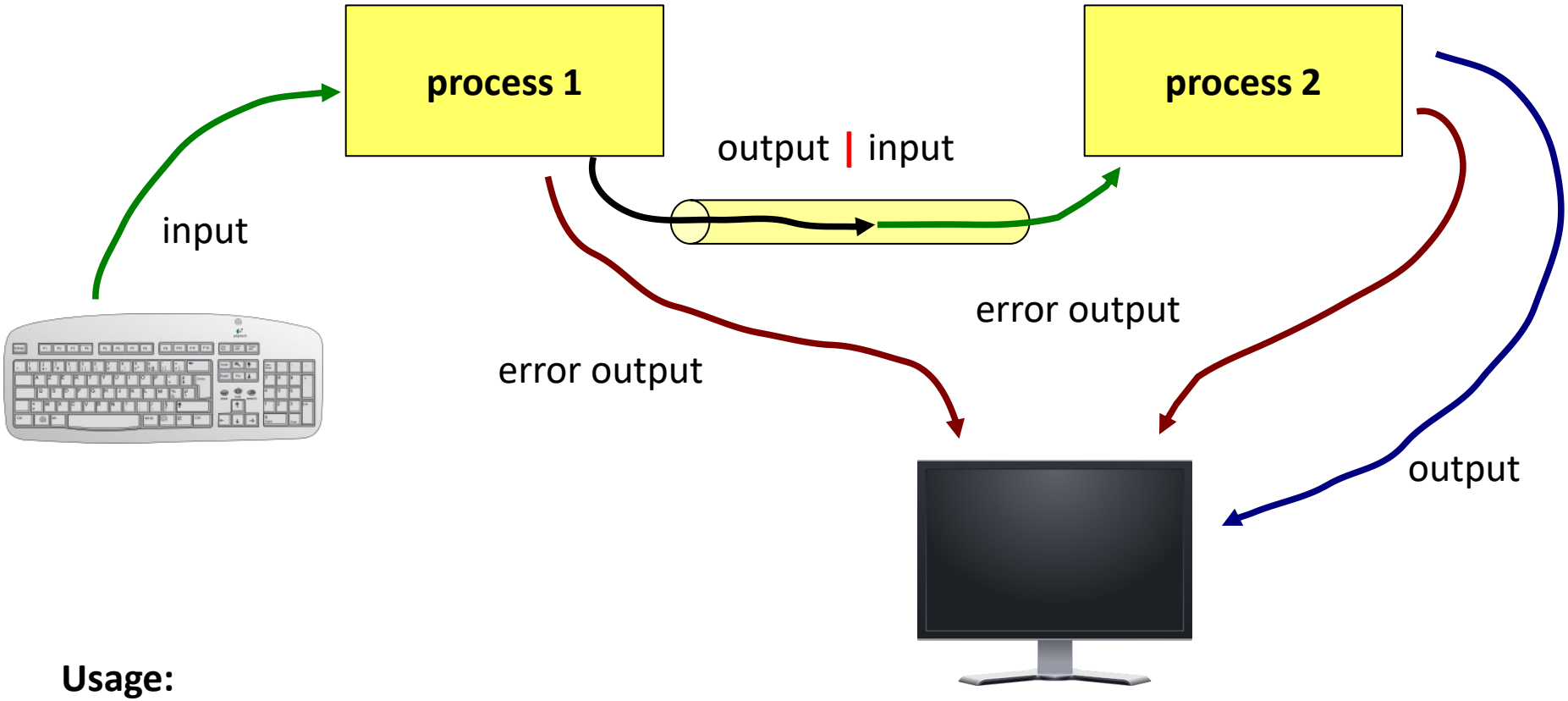
order is important!

~~`$ my_command 2>&1 >> output.txt`~~

does not work

Pipes

Pipes serve to combine the standard output of one process with the standard input of another process.

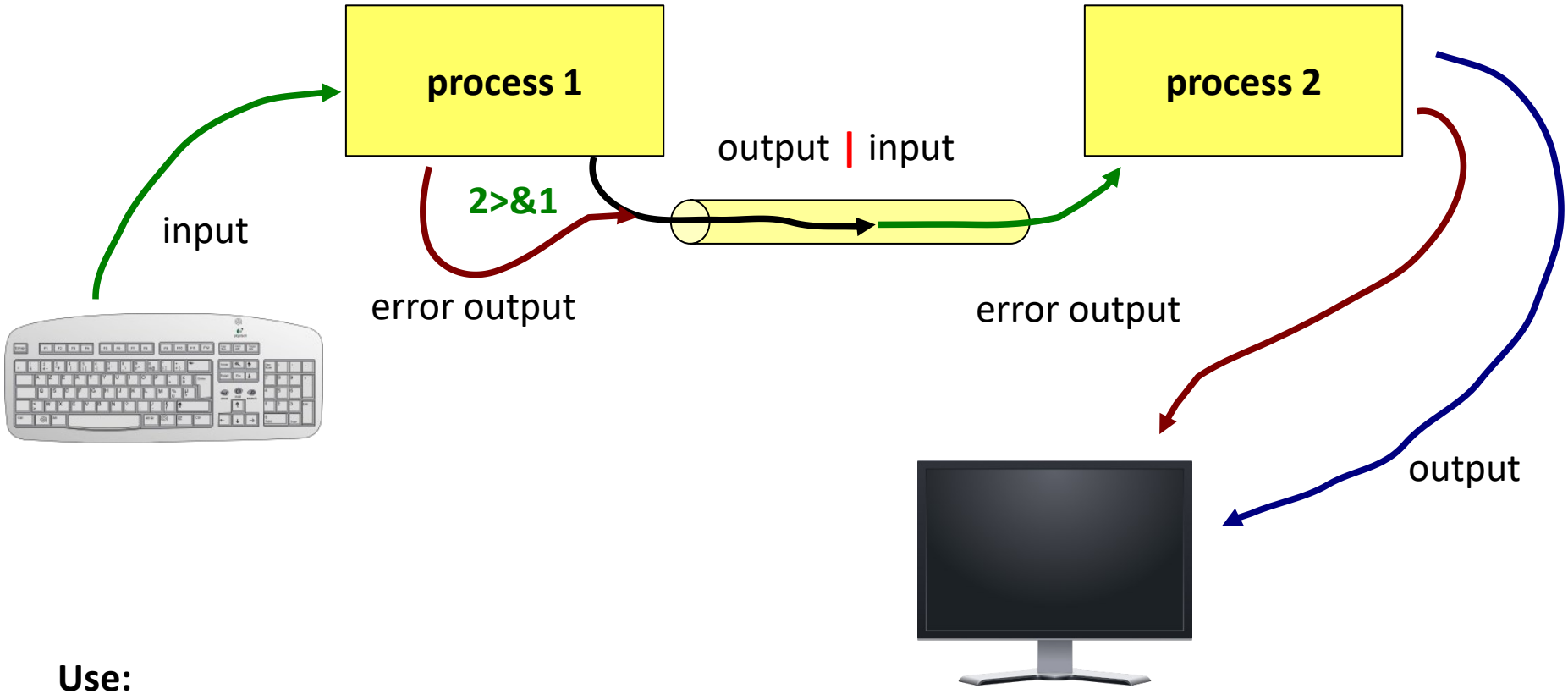


Usage:

```
$ command_1 | command_2
```

Pipes and Error Stream

The transmission of the standard error output via the pipe can be performed after its connection with the standard output.



Use:

```
$ command_1 2>&1 | command_2
```

Commands for Exercise

cat joins the content of several files into one (one after the other), or lists the contents of one file

paste joins the content of multiple files into one (side by side)

wc file information (number of lines, words and characters)

head prints the inital part of a file

tail prints the final part of the file

Examples of use:

```
$ cat file1.txt file2.txt
```
concatenates the content of the files file1.txt and file2.txt and prints the result on the screen

```
$ paste file1.txt file2.txt
```
concatenates the contents of file1.txt and file2.txt (side by side) and prints the result on the screen

```
$ wc file.txt
```
lists the number of lines, words, and characters in file.txt

```
$ head -15 file.txt
```
prints the first 15 lines of file.txt

```
$ tail -6 file.txt
```
prints the last 6 lines of file.txt

Commands for Exercise...

Command **tr** is used for transformation or deletion of characters from standard input. The result is sent to standard output.

Examples:

```
$ cat file.txt | tr --delete "qwe"
```

from the contents of **file.txt,** removes the characters "q", "w" and "e"

```
$ cat file.txt | tr --part "[:space:]"
```

from the contents of the file **file.txt** removes all whitespace

```
$ echo $PATH | tr ":" "\n"
```

in the text sent by the echo command, the characters ":" will be replaced by a newline character "\n "

Exercise 1

1. Find all files with the extension **.f90** which are in the directory **/home/kulhanek/Documents/C2110/Lesson03**. Save list of files to a file **~/Procesy/list.txt**
2. How many lines does the file **list.txt** contain?
3. Write the first two lines from the file **list.txt**, first onto the screen and then to the file **two_lines.txt**
4. Write only the third line of the file **list.txt**
5. In the directory **/proc**, find all files that begin with letters **cpu**. Remove the unauthorized access information from the listing by redirecting of the error current to **/dev/null**
6. List the directory names contained in the variable **PATH**, each on one line.