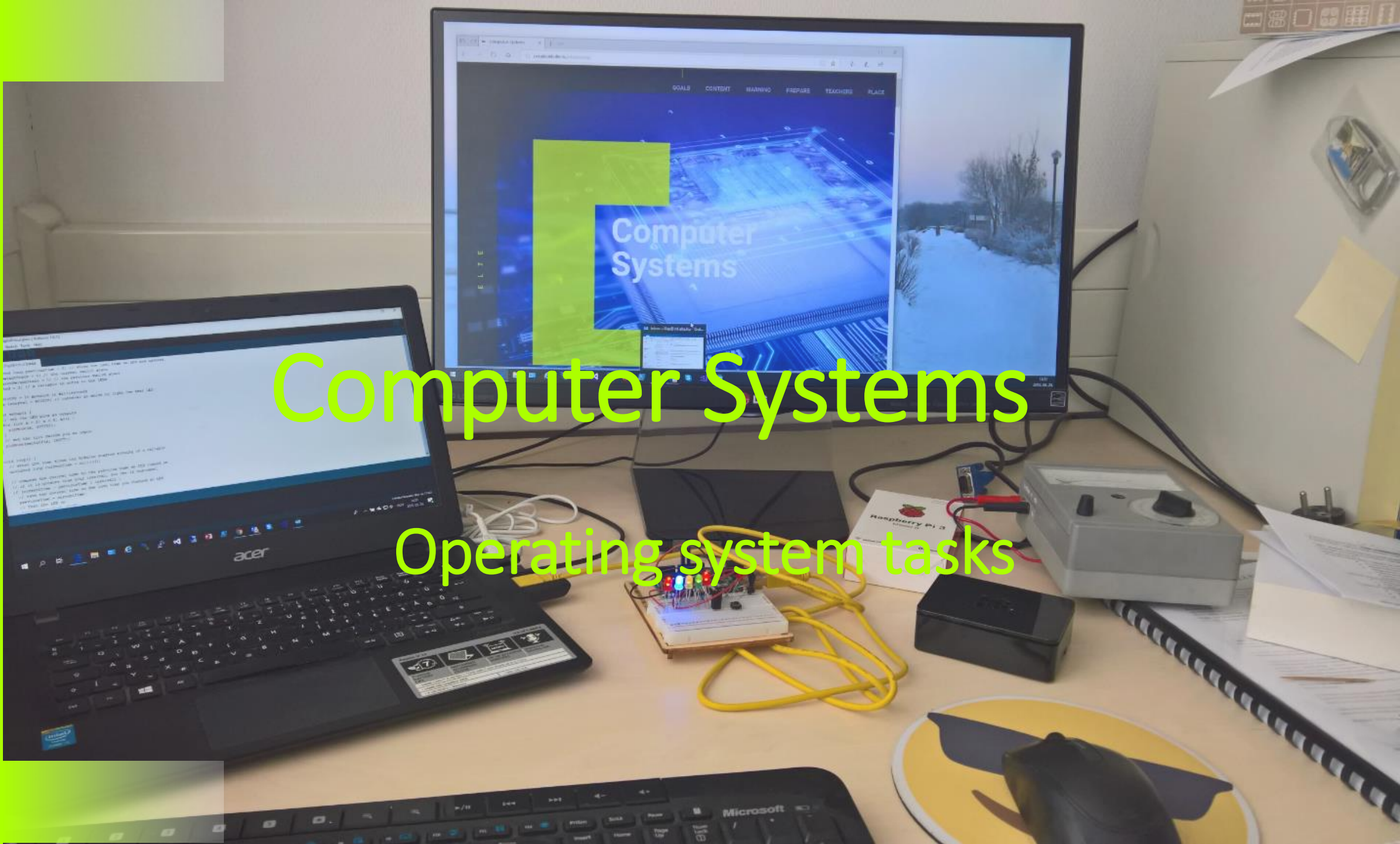
ELTE INFORMATIKA



Computer Systems

Operating system tasks

Review

- Computers, information, representation of numbers, code writing
- Architecture, operating system, client-server role
- Graphical, character connecttion, file systems
- Base commands, foreground and background processes
- I/O redirection, filters, regular expressions
- Variable, command substitution
- Arithmetical, logical expressions

What comes today?

- Control structures
- Branches
- Cycles
- Function definition
- …

A shell example

- What does the following script do?
- apple has a special structure:
  - X=Idared&y=colorisred
  - Where is this kind of formula used?

```
read apple
login=`echo $apple|cut -f1 -d\&|cut -f2 -d=`
pw=`echo $apple|cut -f2 -d\&|cut -f2 -d=`
#
cat <<till
Content-Type: text/html

<html>
<body bgcolor="#a1c1a1">
ill
 echo The login name is: $login , the password is:
$pw !
cat <<till
</body> </html>
till
```

- It cuts out from the input the login name (login) and the passwoprd (pw)!

A branch in shell script

- if
  instructions
  then
  instructions
  else
  instructions
  fi

```
if
        [ $x –lt 10 ]
then
        echo Less than 10
else
        echo Greater
fi
```

- Intendation (line breaks) is not obligatory, only suggested!

Conditional command execution

- if succes then othercommand fi
  - succes && othercommand
- if notsucces then othercommand fi
  - notsucces || othercommand
  - Example:

```
$ echo hello && echo kate
hello
kate
$ false || echo appletree # false:
exit 1
appletree
```

Multi-branch in Shell: case

- case $apple in

```
idared)     echo The apple is idared
            ;;
golden)     echo The apple is golden
            ;;
*)          echo This is an unknown type of apple
            ;;
esac
```

Branch example I.

- Task: Read in a number and decide whether it is positive, zero or negative!

```
#!/bin/sh
# if the first line is a remark, it can be the definition of the shell
read x  # read into variable x
if
    [ $x –lt 0 ]
then
    echo Variable X is negative, the value is: $x
else
        if
            [ $x –eq 0 ]
        then
            echo The value of the variable x is zero!
        else
            echo Variable x is positive, the value is: $x
        fi
fi
```

Branch example II.

- Any command execution can produce a value, and this value can be used as a logical test value of the branch!

```
if
 who | grep john >/dev/null
then
 echo john is logged in
else
 echo john is not logged in
fi
#
read x  #reading
case $x in
  [dD]*)          date     ;;
  [wW]*)          who      ;;
  l*|L*)          ls –l    ;;
 # small l or capital L
  *)      echo bad choice  ;;
esac
```

Cycles in shell: FOR

- **for** variable **in** datalist # general form of for cycle
  - **do**
    - instructions
  - **done**
- for i in `who` # the most frequently used form

```
do
 echo $i
done
```

- for i in apple pear peach # classical use of for cycle

```
do
  echo $i
done
```

writes out the fruits one after the other

FOR sample - seq

- for i in 1 2 3 4 5; do echo $i; done # 5 times loop
- How to create an N times loop?
  1. Or we have to write a script, producing 1…N output!
  2. Or we use the seq command!
- N=10; for i in `seq $N`;do echo $i; done # 1- 10 numbers
  - seq 2 6 # 2,3,4,5,6
  - seq 2 2 6 # 2,4,6
- More info: man seq

Bash - FOR

- for x in $(date) # Bash command substitution
  - do
    - cat $x
  - done
- for ((i=1;i<10;i++)) # C style for cycle
  - do
    - echo $i
  - done
- We recommend to use the classical (sh) cycle!

Cycles in shell: WHILE

- while instructions # general usage of while, if the last
  do # instruction is true, the cycle-seed will be instructions
  # executed
  done
- while
  echo –n Give me your name:
  read name # read from keyboard never gives a false value!!!
  do # to finish e.g.: ctrl+c
  echo Your name is: $name
  done

Cycles in shell: WHILE example I.

- If the read reads in the file-end symbol, the result will be false!
-

```
while
    echo –n Give me your name:
    read nev                # it will read in the file line by line
  do                        # at the end of the file the read gives
    back                            # a false value!
    echo Your name is: $nev
  done < names.txt
```

Cycles in shell: WHILE example II.

- If the parameter is a filename, then we list out it's content!

```
#!/bin/sh
#
while [ $# != 0 ]  # is there any other parameter?
do                 # yes
 if test -f $1     # $1 file?
 then
  echo $1 content:
  cat $1
 else
  echo $1 is not a file!
 fi
 shift  # it shifts the parameters to the left
 echo There are $# more parameters!
done
```

Cycles in shell: UNTIL

- While executes the cyccle–seed if the condition is true, until executes it if it is false!
- until

```
  instruction(s)
do
  cycle-seed instruction(s)
done
```

- Several instructions can be after until, if the last is false, the cycle-seed instructions are going to be executed!

Cycles(loops) in shell: UNTIL example

- We continue writing names into the file while not getting a „no”!
- What would be in the case of [$answer = [Nn]o]?
  - Nothing, because test instruction does not „like” regular expressions!

```
#!/bin/sh
#
# giving a starting value
answer=yes
until
 [ $answer = „no" ]
 do
  echo -n Give me your name:
  read name
  echo $name >>names.txt        # it creates the file
  echo Continue? \(yes/no\)     # if it does not exist
                                # \( is important
  read answer                   #
 done
```

BASH: Select

- Writing a menu structure (only in BASH)
- select i in apple peach plum
- do
- $i processing
- done
- Important: BREAK

pandora.inf.elte.hu - PuTTY

```
illes@pandora:~select i in Continue Exit; do echo $i; if
> [ $i = "Exit" ]
> then
> break
> fi
> done
1) Continue
2) Exit
#? 1
Continue
#? 2
Exit
illes@pandora:~
```

Jumping instructions in shell

- break
  - The cycle is terminated at break command and the program execution continues after done.
- continue
  - The remaining part of the cycle-seed is over-jumped and the execution of cycle continues..
- exit [n]
  - Exit the program and the return value will be n!

Function definition in shell I.

- C style function, processes the parameters similarly (as in shell script).
- We can not define parameters!
- Function value can be assigned by the return instruction!

```
illes@panda:~$ cat funcexample
#!/bin/sh
#
byebye()
{
 echo Bye-bye $1!
}
#
byebye Johny                    #Function call
#end of the script
illes@panda:~$ funcexample
Bye-bye Johny!
```

Function definition in shell II.

- A function may call another function!
  - hello function does not deal with its parameters!
- . scriptname calling:

  how to reach functions from the command line.
  - unset –f functionname

  # function deleting

szamalap.inf.elte.hu - PuTTY

```
illes@pandora:~,cat funcexample2
#!/bin/sh
#
byebye()
{
  echo Bye-bye $1!
}
#
hello()
{
  byebye Johny
}

hello Cate!
illes@pandora:~,. funcexample2
Bye-bye Johny!
illes@pandora:~, hello
```

Command execution options

- Sh funcexample # it will be executed without an execution permission!
- Sh –v funcexample
  - -v it writes out the command before the execution. The whole function too!
- Sh –x funcexample
  - At –x option it writes out only the tested logical expressions. (At –v it writed out the whole branch.)
- Sh –n funcexample
  - Syntax checking

More BASH

- Besides standard shell possibilities we mentioned the BASH implementation as well! E.g. for cycle
- We did not want to highlight each possibility of BASH!
  - For example such possibility is the usage of arrays!
- Full BASH reference is available from here likewise:

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

Script example I.

- What does the following script do?

```
for i
do
 case $i
 in
 [A-Z]*) F="$F $i";;
 [a-z]*) f="$f $i";;
 [0-9]*) break;;
 esac
done
echo $F
echo $f
```

Script example II: Let’s make a package!

- Example:

```
#!/bin/sh
#
echo '# package making‘
echo ‘# taking it to pieces again : sh ./filename
# We get the parameters one after the other
for i
do
  echo "echo $i 1>&2"
  echo "cat >$i <<'$i end’„
  # Whilest „ is the main aphostrope, ‚$i end’ is evaluated
  cat $i
  echo "$i end„  #End of here input
done
```

Usage of packing

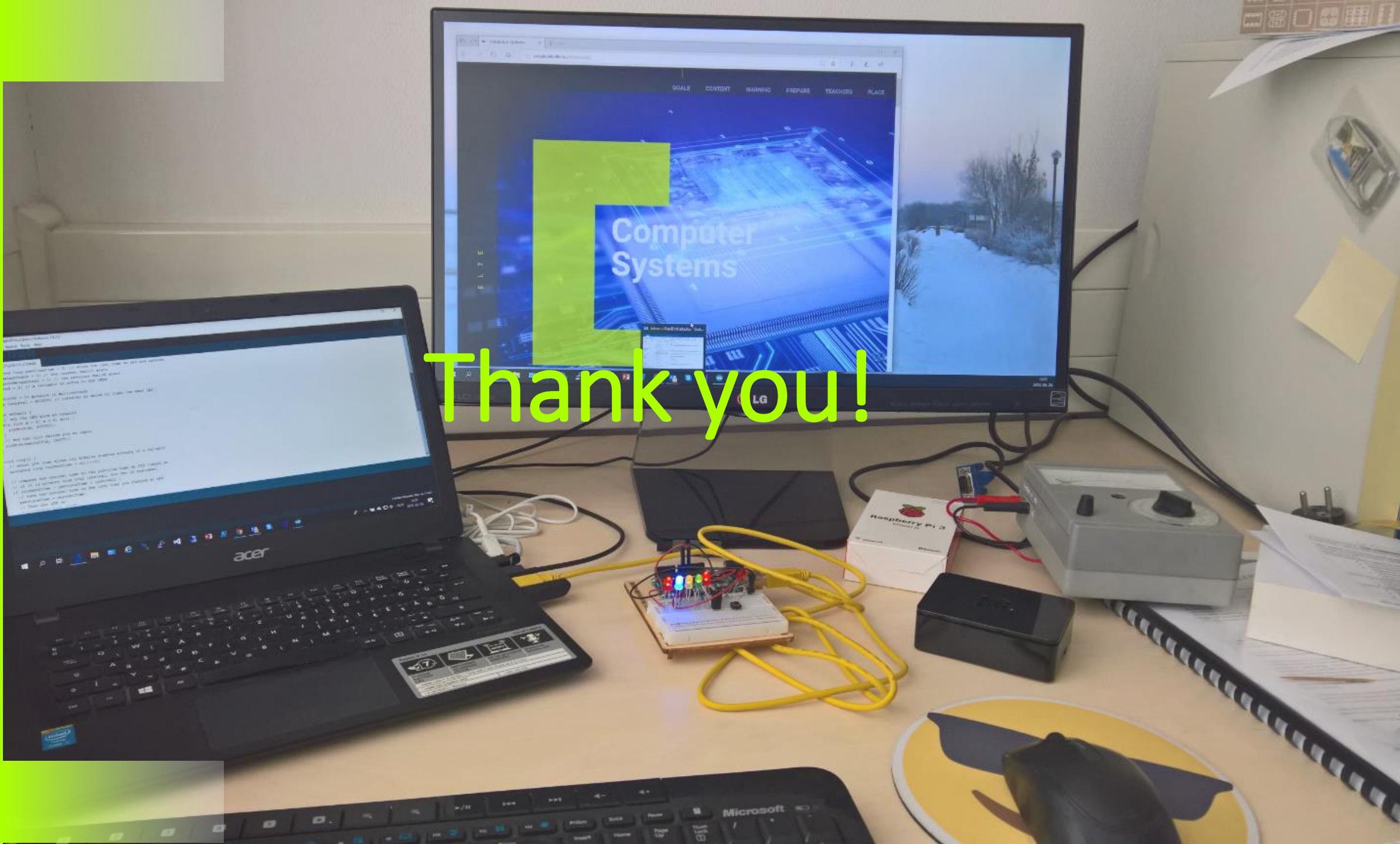
- The packing application will write out the result on the standard outputra!

```
$ pack forexample sorting #result on the screen
$ ….
$ pack forexample sorting > package #result into file
$ sh ./package          # unpacking
                        # if we do not have an x permission, we do have to start
                                # this way
```

Demonstration

- Live demo of the packing script usage

ELTE INFORMATIKA



Thank you!