Informatics–3 LABS: C++ & CLI `prohlep@math.bme.hu`

Thursdays 10^{15} — 11^{45} ms–teams meeting in Hungarian

Fridays 10^{15} — 11^{45} ms–teams lab meeting in English

Requirements and exams are set by the lecturer, but some details are delegated to me.

The aim of the labs: to practice C++ in your own CLI environment at home.

Prerequisites for the labs

Deadline for completion: the last midnight before the first lab class.

- the total upgrading and refreshing your own system at home,

— first: refresh the already installed version of your system,
— then: upgrade it to the most recent version of your edition,
— finally: refresh this new version until it becomes up to date;
— in case of Windows: an editon of Win10, and version 20H2 is needed,
— in case of say Ubuntu desktop edition: version 20.10 is needed, the 20.04 is an LTS only;
— in May: you will upgrade further to 21H1 and 21.04 in case of Win10 and Ubuntu respectively.

Justification: the both of the native MS–TEAMS client and the Windows Subsystem Linux (Win10/WSL2) were developed intensively during the pandemic, and there is no reason to use the earlier less mature releases.

- install the native desktop client for MS-TEAMS onto your own home system

— source: `https://docs.microsoft.com/en-us/microsoftteams/get-clients` .

- join please the MS–TEAMS team called “TTK I3 Prőhle”,

— the corresponding “team code” will be sent to your official email you registered in Neptun,
— this team will be used **exclusively for laboratory meetings**,
— **do *not* use the chat and any other facilities** of this “TTK I3 Prőhle” team.

- in case of Windows: install the WSL2 facility, a CLI of Unix caliber,

— `https://en.wikipedia.org/wiki/Command-line_interface` ,
— any kind of Linux and Apple macOS have CLI of Unix caliber, due to their Unix origin.
— 2 steps: first install WIN10/WSL2, then install the Microsoft version os Ubuntu into it.
— Perform these installs (see next pages) extremely carefully, otherwise you will suck:
!! it is a frequent complaint that it does not work, however often the negligence is the reason.
!! Don’t complicate your life with outdated systems ($\approx$ flat earth, epidemic denial, etc);
!! not activated or expired activation systems can cause mystical errors ($\approx$ self punishment);
!! if “Windows display language” is not switched to English, then you find much less help;
!! rough-and-ready, approximate install attempts can bring the system into tricky puzzle :
!! mind the appropriate order of the installation actions, don’t forget the “reboot”s,
!! mind the kind of the user, whether it should be administrator or should be the plain user,
!! mind the host where you issue the command: righst at home, or a distant access of Leibniz.

- check whether your Windows is fresh and activated (and legal):

— Start menu / Settings / Home / System / About ... right side, scroll down: version "20H2"
— Start menu / Settings / Home / Update & Security / Activation ... state of activation
— Start menu / Settings / Home / Update & Security / Windows ... update status

- change "Windows display language" to English:

— Start menu / Settings / Accounts / Sync your settings ...
at "Language preferences" change "synchronization" to "OFF"
— Start menu / Settings / Time & Language / Language
change "Windows display language" to "English (United States)"
if the option USA is not there, then add it: "Preferred languages" / "Add a language"
— the format of time, date and other data can be changed independetly of the display language:
Start menu / Settings / Time & Language / Region ... "Change data formats"

- install WSL2 into Win10:

— https://en.wikipedia.org/wiki/Windows_Subsystem_for_Linux
— https://docs.microsoft.com/en-us/windows/wsl/install-win10
— WSL /1: Enable the installation and integration of WSL into the win10, i.e.: open a "PowerShell" as "Administrator" and run the command below, entered without line breaking:

```
dism.exe /online /enable-feature
              /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

— WSL /2: Enable the "Virtual Machine Platform" optional component, i.e.: run the command below, entered without line breaking, in the same "PowerShell" as "Administrator" above:

```
dism.exe /online /enable-feature
              /featurename:VirtualMachinePlatform /all /norestart
```

— WSL /3: Restart your machine to complete the WSL install and update to WSL–2.
— WSL /4: Set WSL 2 as your default version, i.e.:
issue the command below in a "PowerShell" as "Administrator":

```
wsl --set-default-version 2
```

— WSL /5.optional: If you see a message, that "`WSL 2 requires an update to its kernel component. For information please visit https://aka.ms/wsl2kernel`", then you have to update the "WSL–2 Linux kernel" before proceeding to the next non-optional step.
https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi
This msi–file above must be downloaded, then run, ... and when prompted for elevated permissions (= will run as "Administrator"), then select "yes" enabeling it for proper installation.

- install the MS-edition of Ubuntu 20.04 into WSL2:

— WSL /6: download first the "Ubuntu 20.04 LTS" install resource from the url below:
https://www.microsoft.com/store/apps/9n6svws3rx71
Warning: pressing the "download / letöltés" button, a "Microsoft Store" login session has to be

answered, but the "eduID" used to access ms–teams for university meetings is *not* appropriate here, you have to have as an everydau ms–windows user an own MicroSoft account for this store, ... it is easy to create one and is charge free until you want to download free softwares only. Your name concerning this store will be exacly the same, as the email address what you provide during the creation of this ms–store ms-account.

— WSL /7: Install this "Ubuntu 20.04" into the WSL2. I.e.:
  either click on the orange Ubuntu tile in the Start Menu, if it is already there,
  or issue the "`ubuntu2004`" command (without the quotations marks) at a *non-elevated* (= *not* as administrator) command-line prompt (cmd.exe).

This first invocation of "Ubuntu 20.04" takes rather nice amount of time, since before getting a proper Unix-alike terminal, the "Ubuntu 20.04" will be installed. During this install process, you will be prompted to give a name and a password for the main user of this "Ubuntu 20.04". Later, you can launch the installed "Ubuntu 20.04" in the same way as above, but there is no need to wait for the installing any more.

— WSL /8: make a tile (csempe) into the Start menu for WSL2/Ubuntu

WARNING: The tile of the proper WSL2/Ubuntu terminal and the tile of the plain WSL2 CLI can be easily mismatched. Be careful to use the orange colored WSL2/Ubuntu tile instead of the black–white–yellow penguin decorated tile of the plain WSL2 CLI.

— WSL /9: Refresh your Ubuntu system issuing the commands below in the "Ubuntu 20.04" terminal. You will be prompted to give your password in this local WSL2/Ubuntu.

```
sudo -s apt-get update
sudo -s apt-get -y dist-upgrade
sudo -s apt-get -y autoremove
sudo -s apt-get clean
```

- FOR EVERYONE — install C++ compiler into the CLI environment

— in case of Windows: we have already prepared a "WSL2/Ubuntu" tile into the "Start menu". Clicking that tile we get a character terminal, containing a CLI environment of unix caliber. in case of non-Windows: discover please, how to invoke a terminal by keystrokes of clicks.

— if you got an Ubuntu terminal (WSL2/Ubuntu or a Debian based linux), then issue the two commands below: install C++, and check the version of the compiler

```
sudo -s apt-get -y install g++
g++ --version
```

In case of a real desktop Ubuntu 20.10 we get the fresh 10.2.0 C++ compiler, while in case of WSL2/Ubuntu we get a slightly older version. With the spring update, the real desktop version will be even more current, while in the WSL2/Ubuntu will remain unchanged in WIN10/21H1.

— if you have a non-Ubuntu, or in general a non-Debaian based terminal, then discover using google how to install the preferably GNU C++ compiler, except for macOS.

— macOS / Clang (recommended) : https://www.youtube.com/watch?v=jF_RiRYk-Wo

— macOS / GNU C/C++ : https://www.youtube.com/watch?v=mnkySV0sZmU

- FOR EVERYONE — remote login to `leibniz.math.bme.hu` using plain password

— check whether you remember your account information (name and password) at Leibniz
if not, then send a seriously apologetic letter to the administrator `mgergi@math.bme.hu`

— test whether the pair of the name and the password you have really works:
https://webmail.math.bme.hu/
if you can login there successfully, then you have the correct information

— open a CLI termial, and issue the command below, replacing `name` with your account name
https://en.wikipedia.org/wiki/SSH_(Secure_Shell)

```
ssh -t name@leibniz.math.bme.hu
```

if this is the first time when you SSH to Leibniz from this particular system, then you will get an important "yes/no" question. You can answer this by 3 letters, namely "`y e s`", since a single "`y`" is not enough there in this situation. Check whether the fingerprint is the same as below:

```
ECDSA key fingerprint is SHA256:hHiKG+V1DukgomiHkGHn0Aw8+9vjqHtHGQtG6ec0uv8
```

https://en.wikipedia.org/wiki/Man-in-the-middle_attack ... if the fingerprint does not match, then a "man in the middle" attack can be the reason, and abort with 2 letters, namely "`n o`". If in other situation you don't know the fingerprint, then you have to collect it, using telephon or other tool, independent of your SSH session and the target server.

— exit from Leibniz, do not remain there:

```
exit
```

the keystroke Ctrl-d works as well.

- FOR EVERYONE — remote login to Leibniz using high security key exchange

— open a local unix terminal, and generate a high security key,

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -P ''
```

— copy the public part of this key to Leibniz, replacing "`name`" with your account name,

```
ssh-copy-id -i ~/.ssh/id_ed25519 name@leibniz.math.bme.hu
```

— if we copied correctly, say 2 apostrophes and not 1 double quotation mark after `-P`, then

```
ssh -t name@leibniz.math.bme.hu
exit
```

the first command will login without asking for the plain password. The second will logout.

IN SUMMARY: you should appear at the first lab class so that you can show all of follwings below, at the beginning of the lab class, using "desktop sharing" in native MS–TEAMS client:

— your system at home is absolutely fresh,
— you have a CLI of unix caliber,
— a C++ compiler is available in that CLI,
— from that CLI you can log into Leibniz without having to enter in your password.

THE LABOR DUTIES ON THE WEEK 01.

Commissioning your own home Unix account

It is a good habit first to check the working directory, since in the case of WSL2 Ubuntu it can happen, that accidentally we use the WSL2 icon instead of the Ubuntu icon. Also in other, windows free cases, better to make it clear, what is our home directory just after login, and to where the `cd` (the empty Change Directory) command will bring us.

```
pwd ; cd ; pwd                                      # Print Working Directory
mkdir i3                                            # MaKe DIRectory
cd i3                                               # Change Directory
cd ~/i3                                             # works from everywhere
```

If the results of the two print working directories separated by an empty change directory differs, then it idicates the risk that we are inside a Win10, but we clicked the WSL2 icon instead of the Ubuntu icon. Next we make the directory named `i3` for our i3-course. Since it is not a good practice to work in a single directory regardless of what we are doing. The last change directory shows how to change to this `i3` working directory, if we are not in the home of our account.

```
gcc -v
g++ -v
python3 -V
ssh -V
```

Test whether we have the Gnu Compiler Collection, the Python interpreter and an SSH-client, this way above. If not, then the missing stuffs must be installed as the system adminsitrator, called `root` in case of a contemporary Unix.

What follows below, the commands needed can vary from one kind of a a contemporary Unix to another one. The Ubuntu 20.04 (inside WSL2 as well) variant is there below.

```
id             # here you are a plain user of your system
sudo -s        # the password of the plain user is asked for
id             # now you should be the 'root' of your system
apt-get update
apt-get -y dist-upgrade
apt-get -y install g++      # ONLY IF the Gnu Compiler Collection is missing
apt-get -y install python3       # ONLY IF the Python interpreter is missing
apt-get -y install openssh-client           # ONLY IF ssh client is missing
apt-get -y autoremove
apt-get autoclean
id             # you are still the 'root' of your system
```

```
exit            # <<< do exit please, otherwise the later stuffs will go wrong
id              # you are reverted to the original plain user
```

In case of macOS, see: https://www.youtube.com/watch?v=mnkySV0sZmU for GNU C/C++ and https://www.youtube.com/watch?v=jF_RiRYk-Wo for Clang, which is the suggested one.

Commissioning your own Leibniz account

```
ssh -t name@leibniz.math.bme.hu                             # replace 'name'
touch .hushlogin                       # disable  the annoying login messages
mkdir i3                               # make a directory for the lab classes
cd i3 ; pwd                            # and test it
for prog in latex python3 gcc g++ bash ; do ${prog} --version ; echo ; done
exit                                                    # <<< do exit from LEIBNIZ
```

Make the i3 directory, and test it. Then demonstrate that we have all the compilers we need. And don't forget to exit finally.

One keystroke "Print Screen"

Whenever there is a need to ask for some help in informatics, the situation must be presented. The best way is to copy the character terminal content, and paste it into an email. However sometimes the demonstration of the situation can not be done on the level of characters, the graphical information does also matter. In these case the best solution is to make a screenshot.

Most graphical user interfaces have a one keystroke "print screen" option. Hence there is no need to use other solutions. Instead, do find (using google) the canonical solution built into your GUI. Be prepared, that in the normal cases the resulting image file is done silently without the need of pushing extra OK buttons. However, then you have to know without any aid, where to find the resulting image file.

- If you use a native Ubuntu with the Gnome3 GUI, then the pushing the plain <print screen> key (between the F12 and Scroll Lock) will save an actual screenshot into a PNG file located in `~/.desktop`, for example: `~/.desktop/Screenshot from 2021-02-04 10-51-56.png` .
- If you are working in Win10, then "Windows logo key + PrtScn" will do the job (holding the win logo key pushed, striking the PrtScn key shortly), and the resulting PNG file will be located in your personal `Pictures/Screenshots` folder, the name of the first screenshot will be `Screenshot (1).png` , behindering the comfortable use of CLI environments.

Accessing the windows files from the WSL2/Ubuntu

```
ls -al /mnt/                                                # ''This PC''
ls -al /mnt/c/                                              # win10 drive
ls -al /mnt/c/Users/                       # find your 'name' in this listing
ls -al /mnt/c/Users/name/Desktop/                           # replace 'name'
ls -al /mnt/c/Users/name/Documents/                         # replace 'name'
ls -al /mnt/c/Users/name/Downloads/                         # replace 'name'
ls -al /mnt/c/Users/name/Pictures/                          # replace 'name'
ls -al /mnt/c/Users/name/Pictures/Screenshots/              # replace 'name'
```

Don't forget to use <UpArrow>, <BackSpace> and <TAB> extensively.
In the above case: `l s` <Space> `- a l` <Space> `/ m n` <TAB> <Enter> <UpArrow> `c` <TAB> <Enter> <UpArrow> `U` <TAB> <Enter> <UpArrow> first-few-letters-of-your-name <TAB> `D e` <TAB> <Enter> <UpArrow> few-<BackSpace>s `D o c` <TAB> <Enter> few-<BackSpace>s `D o w` <TAB> <Enter> <UpArrow> few-<BackSpace>s `P i` <TAB> <Enter> <UpArrow> `S c` <TAB> <Enter>

The difference between SSH and SCP

The main similarity is that there is an account at a remote host in both cases. The main difference is that an SSH session provides a CLI — in typical case providint the `bash` command shell — equipped with the access rights of the remote account in question, while an SCP session provides a bidirectional file transfer facility between the remote files accessible using the accesss rights of the remote account and the local files. The both of SSH and SCP benefit great from using the key exchange autentification avoiding the need of typing a password by hand. See the local Unix man pages.

```
man ssh                                   # leave the reading with 'q' (Quit)
man bash                                  # leave the reading with 'q' (Quit)
man scp                                   # leave the reading with 'q' (Quit)
```

Upload a screenshot to Leibniz

```
cd ~/i3                                                         # anyways
mv /mnt/c/Users/name/Pictures/Screenshots/Screenshot*.png .     # Win10/WSL2
mv ~/.desktop/Screenshot*.png .         # < mind the dots ^     # Ubuntu/Gnome3
scp Screenshot*.png name@leibniz.math.bme.hu:                  # replace 'name'
ssh -t name@leibniz.math.bme.hu  # mind the ^ colon            # replace 'name'
mkdir  --parents   public_html/screenshots                 # use <TAB>s
mv Screenshot*.png public_html/screenshots                 # <UpArrow> and edit
chmod -f -R 'a-strwx,a+rX,u+w' public_html                 # use  'p' and <TAB>
chmod -f 0711 ~
exit                                                      # leave the remote CLI
```

The two `chmod` commands arrange the access rights in the remote filesystem such a way, that the uploaded screenshots can be viewed in any html browser (say firefox, safari, chrome) using the URL below:

`https://math.bme.hu/~name/`   do not forget to replace "`name`"

Download files from Leibniz

First we make dummy files at Leibniz, in order to have something to download.

```
cd ~/i3
ssh -t name@leibniz.math.bme.hu                              # replace 'name'
mkdir i3 ; cd i3
for x in $(seq 9) ; do echo $x >dummy_$x.txt ; done
ls -l
cat dummy_5.txt
exit                                                    # leave the remote CLI
scp -r name@leibniz.math.bme.hu:i3/'dummy*' .                # replace 'name'
ls -l                            # ^       ^ ^   mind these punctuation marks
cat dummy_5.txt
rm dummy_?.txt                                       # never leave any trash beind
ssh -t name@leibniz.math.bme.hu rm i3/'dummy*'      # remote cleaing command
```

The wildcard expressions must be protected from expanding by the local command interpreter, in order to ensure, that these expressions will be received by the `scp` or the `ssh` command itself. That's why the single quotes around `dummy*`.

An easy and dummy editor: the Nano

It is usually there in all of the various Unix distros. It's design goal is to require a minimal (nano) foot print in resources. In return it's editor services are very very limited.

```
nano                                              # eXit with '^X', i.e.: <ctrl> x
```

The first "`^O WriteOut`", i.e.: `<ctrl> o`, asks for a filename, and the later invocation of this "WriteOut" will provide this name, what we can alter if we want, and an `<enter>` will start the saving. The editor can be invoked specifying a file to edit.

```
date > testfile.txt
nano testfile.txt      # n a n o <space> t <TAB> if only this begins with t*
cat testfile.txt
rm testfile.txt        # never leave behind trash disturbing later, clean up
```

VIM, the primary Unix editor http://vimdoc.sourceforge.net/htmldoc/uganda.html

VIM, the VI IMproved is the rather old extended version of the classic VI that is available even on a scaled-down Unix system. This is why `vim` is important and worth knowing a bit.

From the comtemporary view it is a surprising editor, it has diverse operating modes, and in addition to that, the default mode — it is in at starting — is *not* the usual common editing mode, inserting the characters corresponding to the keystrokes.

```
sudo apt-get -y install vim                    # install "VIsual editor IMproved"
```

Leibniz provides `vim`, but on the own system we have to install it. However, *even* on Leibniz it is advised to have an own "`VIM Run Commands`" file, namely the `~/.vimrc` .

The main aim concerning our configuration effort is to set up the keystrokes `<F3>` and `<F4>` to execute an "exit without saving, ignoring silently the changes" and an "exit saving silently the pending changes", respectively. This is needed for the comfortable fluent work in `vim`.

```
nano .vimrc
```

Remain please in the editor just invoked, and type please in the folowing 9 lines below:

```
syntax on
set nobomb
set mouse=
set ttymouse=
set viminfo="NONE"
map <F3> :q!<Return>
map <F4> :wq<Return>
imap <F3> <Esc>:q!<Return>
imap <F4> <Esc>:wq<Return>
```

Now exit the Nano with saving, i.e.: `<ctrl>-o <enter> <ctrl>-x` ...

```
cat .vimrc
```

We should see exactly the same 9 lines as above. The last 4 lines do the desired keyboard mapping for VIM. The first line switches on the "syntax highlighting', the fifth line switches off saving and using the editing history. https://en.wikipedia.org/wiki/Syntax_highlighting

`type :help<Enter> for on-line help` (use <Ctrl>] to click the links)

Whilst this editor is self documenting, — see the quotation above from the middle of the starting screen of `vim`, — better to know a few things in advance. It has quite a few operating modes. There is no direct transition between these modes, but only via the default "normal" mode. The new changes to the file edited can be saved, and the editor can be leaved using the "command line" mode. The character-by-character editing is usually done in "insert" mode. Editing in larger units is done in default mode.

• the "normal", the default mode : everything the user types in normal mode is interpreted as commands, including those which switch the user to other modes. A few examples:

— `j` or `k` moves the cursor one line down or up respectively
— `5j` or `5k` moves the cursor five lines down or up respectively
— `gg` or `7gg` moves the cursor to the first or seventh line respectively
— `7G` or `G` moves the cursor to the seventh or last line respectively
— `x` or `9x` deletes one or nine characters respectively
— `dd` or `9dd` deletes one or nine lines respectively
— `ra` or `rb` or `rc` etc, replaces the character to `a` or `b` or `c` etc respectively
— `u` undoes changes up to the last time you were in normal mode
— `yy` or `7yy` yanks one or seven lines into the own separate cut and paste buffer of the editor
— `p` or `P` pastes the cut and paste buffer below or above the current line respectively

• `-- INSERT --` mode : can be leaved by <Esc>, and can be reached several ways, `i` insert before at the current cursor position, `I` insert at the beginning of the current line, `a` insert after the current cursor position, `A` insert at the end of the current line.

• `-- REPLACE --` mode : can be leaved by <Esc>, and can be reached by `R`, and then the characters will be replaced, overwrited starting at the cursor positon

• `-- VISUAL --` mode : can be leaved by <Esc> or by completing the selection by an editing command, and can be reached by `v`, then moving the cursor position the selection area can be changed, and finally, the first "non cursor–moving" keystroke will be interpreted as an editing command of the default normal mode, but the subject of the action will be the selected area, for example:

— `v` <cursor movings> `x`	will delete the selected area	≈ cut to copy–paste buffer (ctrl-x)
— `v` <cursor movings> `y`	will yank the selected area	≈ copy to copy–paste buffer (ctrl-c)

• `-- VISUAL LINE --` mode : can be leaved by <Esc> or by completing the selection by an editing command, and can be reached by <Shift>v namely `V`, and the operation is very similar to the plain "visual" mode, but the only essentian difference is that the selection can be altered in terms of lines instead of characters by characters. The "visual line" mode is extremly comfortable if we wish or have to think in lines as units

• `-- VISUAL BLOCK --` mode : can be leaved by <Esc> or by completing the selection by an editing command, and can be reached by pressing <Ctrl>`v`, and the operation is very similar to the "visual line" mode, the only essentian difference is that the shape of the selection always remains a block, more precisely a rectangular area, the two opposite corners of what is the current new position and the position where we entered the "visual block" mode

• `recording @x` the macro recording mode

— the goal of this mode is to record a sequence of editing actions, replayable by two keystrokes

— hidden danger of this mode is that right at the start of this mode nothing can be seen

— the other suprise of this mode is that <Esc> will not leave it but repeating the initial `q`

— `qx` starts a macro recording onto the letter `x`

— a macro recorded onto onto the letter `x` can be run by `@x`

— example: `qx` <some editing action> `q` <resume working for a while, and then> `3@x`

the resulting effect is that the "some editing action" will be replayed 3-times at the new context

• `:` the "command line" mode : can be leaved by <Esc>, and can be reached by `:` (colon)

— `:w filename.txt`	write the editor buffer to the given file
— `:q` or `:q!` or `:wq`	quit, or quit without questions, or save and quit respectively

See https://en.wikibooks.org/wiki/Learning_the_vi_Editor/Vim/Modes for further details, or in general https://en.wikibooks.org/wiki/Learning_the_vi_Editor .

After processing the "prerequisites for the labs" of 4 pages, and the first 6 pages of the "the labor duties on the week 01", after these 10 pages we are ready to start the real Informatics-3, the C++ programming in CLI environment. There are only 3 pages of C ++ in the first week.

Positional parameters and arguments in case of advanced CLI, like bash

One reason why a “composition-oriented user interface” (CLI) performs better than a “selection-oriented user interface” (GUI) is that the command line arguents can be dynamically altered according to the actual needs, while in case of a GUI the argumentum values are statically forged together with the menu or icon items. Hence a CLI is much more flexible than a GUI.

```
#! /bin/bash
echo "argv[0] : \"${0}\""
i=0
for arg in "$@" ; do
    i="$((i+1))"
    echo "argv[${i}] : \"${arg}\""
done
```

Using vim, write a file named `00_show_arguments.sh` with the content above.

```
vim 00_show_arguments.sh                          # exit with saving
cat 00_show_arguments.sh                          # c a t <space> 0 0 <TAB>
bash 00_show_arguments.sh                         # b a s h <space> 0 0 <TAB>
```

Notice how important are the prefixes of the filenames we use. The shorter prefixes of our filenames are pairweise different, the more comfortable becomes a CLI using <TAB>. Hence it is a crucial point to well design the nomenclature or our files. Think in advance, you get comfort later in return.

```
bash 00_show_arguments.sh date                    # <UpArrow> <space> d a t e
bash 00_show_arguments.sh $(date)                 # <UpArrow>    and edit it!
date
bash 00_show_arguments.sh "$(date)"               # <UpArrow> <UpArrow>  edit
bash 00_show_arguments.sh '$(date)'               # <UpArrow>        and edit
```

The aim of the above experiments is to demonstrate the difference between the structure of an actual command line, and the list of positional argument received by the program invoked by that command. The reason for this difference is, that — between our commandline and the invocation of the desired program — there are **seven kinds of expansion** performed by the shell `bash`: brace expansion, tilde expansion, parameter and variable expansion, command substitution, arithmetic expansion, word splitting, and pathname expansion.

```
man bash          # inside: / E X P A N S I O N <enter>, then a few times: n
```

— case of plain `date` : there is no kind of expansion, hence this `date` is the first argument.
— case of `$(date)` : a “command substitution” is performed, namely the `date` is executed, and it’s result substitutes the original `$(date)` string, providing 6 positional arguments.
— case of `"$(date)"` : a “command substitution” is performed, as above, but the resulting new string will be considered a single irreducibel word, despite of the spaces inside. Hence the whole output of `$(date)` will be the argument one.
— case of `'$(date)'` : any substitution is suspended, hence yields 1 argument only.

```
#! /usr/bin/env python3
import sys
argc = len( sys.argv )
for i in range( 0, argc ) :
    print ( "argv[%i] : \"%s\"" % ( i, sys.argv[i] ) )
# end of the body of 'for'
```

Using vim, write a file named `01_show_arguments.py` with the content above.

```
vim 01_show_arguments.py                                  # exit with saving
cat 01_show_arguments.py                                  # c a t <space> 0 1 <TAB>
python3 01_show_arguments.py                      # p y t h o n 3 <space> 0 1 <TAB>
python3 01_show_arguments.py date                 # <UpArrow> <space> d a t e
python3 01_show_arguments.py $(date)              # <UpArrow>          and edit it!
date
python3 01_show_arguments.py "$(date)"            # <UpArrow> <UpArrow>    and edit
python3 01_show_arguments.py '$(date)'            # <UpArrow>              and edit
```

Python-script gets the positional arguments the same way as a bash-script. For each of these two languages, bash an Python we have an interprer: the commands are translated to processor instructions on the fly. Hence these languages are slow and resource wasting. Since Python-alike interpreted languages are world wide popular in web servers as well, hence the interpreted languages are responsible for a non-negligible amount of energy wasting, hence CO2 and pollution.

Let us see the world of C/C++ languages, more suitable for high performance cases. When the performance and resource efficiency does matter, then the language has what is known as a "compiler" that relieves the runtime resources of unnecessary loads as much as possible.

```
#include <stdio.h>
int main( int argc, char * argv[] ) {
  for ( int i = 0; i < argc; ++i ) {
    printf( "argv[%d] : \"%s\"\n", i, argv[i] );
  }
  return 0;
}
```

Using vim, write a file named `02_show_arguments.c` with the content above. It is a source code written in plain C, what will be the minor topic of Info3, du to it's similarity to C++.

```
vim 02_show_arguments.c
cat 02_show_arguments.c
gcc -std=c11 02_show_arguments.c
file a.out
./a.out
./a.out date
```

```
./a.out $(date)
./a.out "$(date)"
./a.out '$(date)'
```

The gcc is the plain C compiler, -std=c11 tells that the source 02_show_arguments.c must be compiled according to the C standard as of 2011. The resulting binary is called a.out by default. In order to run a binary in the directory where we are, the plain name of the binary file is not enough due to safety reasons, hence it must be prefixed by ./, declaring explicitly that we really want to run something from the present working directory.

```
#include <iostream>
int main( int argc, char * argv[] ) {
  for ( int i = 0; i < argc; ++i ) {
    std::cout << "argv[" << i << "] : \"" << argv[i] << '"' << std::endl;
  }
  return 0;
}
```

Using vim, write a file named 03_show_arguments.cxx with the content above. It is a source code written in C++, what will be the main topic of Info3.

```
vim 03_show_arguments.cxx
cat 03_show_arguments.cxx
g++ -std=c++14 03_show_arguments.cxx
file a.out
./a.out
./a.out date
./a.out $(date)
./a.out "$(date)"
./a.out '$(date)'
```

The g++ is the C++ compiler, -std=c++14 tells that the source 03_show_arguments.cxx must be compiled according to the C++ standard as of 2014. The resulting binary is called a.out again by default.

Summing up we see, that any program started in a POSIX (= Portable Operating System Interface , https://en.wikipedia.org/wiki/POSIX) compatible environment get positional arguments and in the same way, regardless of the programming language used to write the source code of that program.

Fundamental resources

- https://math.bme.hu/~prohlep/i3/cplusplus-com.pdf
- https://en.cppreference.com/

This is the end of the week 01.

THE LABOR DUTIES ON THE WEEK 02.

From now on the role of the handout will change essentially:

- less imperative, your way will be more free how you work hard
- more advisory, and it is your problem if you do it in an other way
- less copy paste, more careful reading and understanding will be needed what to do
- handout will be a skeleton only of your independent work alone

The steps of a suggested way to study the source codes

— study the details of the source code and consider the related comments in first reading
— assembly the source code on the base of the handout and lab class
— compile the source code and run the program, ... fix the typing errors in the source
— repeat the edit – compile – run cycles several times, applying small changes to the source
- alter the literal constants in the sourd, alter the positional arguments given at the prompt
- interpret the results, rethink the comments
— remove smallish fragment of the source, reenter it from head, and check it with compilation
— remove larger fragment of the source, reenter it from head, and check it with compilation
— intentionally make minor syntactic or semantical error in the source, interpret the warnings
— perturbate the source in order to achieve a slight change in the functionality of the program

Terminal of 80 columns

Never kill a terminal using the mouse and the red X on the frame around the terminal. Instead, type `exit` and push <enter>, or type <ctrl> `d`. Each of these last two solutions sends a graceful request for quitting the actual work, while the red X on the frame sends a harmful `SIGKILL` signal (i.e.: `kill -9`) to the terminal, which in return will kill the program inside applying the same agressivity.

It is strongly recommended to adjust the background and foreground color of your terminal in oder to have dark (black) letters on light (white or light yellow) background. The focus system of our eyes is protected and works better on black letters over white paper, than white letter over black background.

The increasing the font size is also recommended, in order *not* to mismatch a dot and comma, or a colon and semicolon.

However the with of the terminal is *not* optional, it must be left on the traditional default of 80 columns. Hence the simple maximisation of the terminal windows is usually a very bad idea. Only the height of the terminal is suggested for maximalisation.

```
wc -L *.cxx                                  # length should be less than or equal 78
```

Learn the location of the ASCII characters on your keyboard

It is strongly recommended to use the English (US English, but the normal one, and *not* the international one) keyboard layout, since the national keyboard layouts obscure the access of the ASCII characters badly needed for fluent programming. Avoid using the /, *, -, + from the numeric keypad, find it please in the traditional letter area.

```
man ascii                                               # press 'q' to exit
```

!	(0x21)	logical negation
"	(0x22)	delimiters of string literals
#	(0x23)	first character of a preprocessor directive
$	(0x24)	...
%	(0x25)	integer division
&	(0x26)	bitwise and, logical and, logical assignment, reference operator
'	(0x27)	delimiters of character literals
(	(0x28)	opening parentheses of expressions, parameter or argument lists, initialisations
)	(0x29)	closing parentheses of expressions, parameter or argument lists, initialisations
*	(0x2a)	arithmetic operator, address resolution operator
+	(0x2b)	arithmetic operator + and ++
,	(0x2c)	the comma operator, called also listing operator
-	(0x2d)	arithmetic operator - and -, part of -> operator
.	(0x2e)	class- or object member selector
/	(0x2f)	arithmetic operator, part of one line- and block comments
:	(0x3a)	part of conditional expressions, leading character of inheritance lists
;	(0x3b)	grammatical synchron token between statements
<	(0x3c)	arithmetic operator, the << append operator
=	(0x3d)	assignment operator, part of diverse relational sympbols
>	(0x3e)	arithmetic operator, the >> append operator, -> member operator
?	(0x3f)	part of conditional expressions
@	(0x40)	...
[	(0x5b)	opening bracket of an array indexing
\	(0x5c)	escapes the lexical activity of the next character
]	(0x5d)	closing bracket of an array indexing
^	(0x5e)	bitwise XOR
_	(0x5f)	frequent member of usual identifier names
`	(0x60)	...
{	(0x7b)	opening curly brace of an inititialisation or a block
|	(0x7c)	bitwise or, logical or, logical assignment
}	(0x7d)	closing curly brace of an inititialisation or a block
~	(0x7e)	leading character in the names of destructors of classes, bitwise or logical negation

Shell script as an "Integrated Development Environment" in CLI

Collect please the `edit_compile_run.sh` shell script. This script will be upgraded during the semester, hence sometimes you have to recollect it when it is asked for. Inspect it by `view`, the full understanding of this script should normally belong to i1 course.

```
cd ~/i3
scp name@leibniz.math.bme.hu:~prohlep/public_html/i3/edit_compile_run.sh .
view edit_compile_run.sh                                    # push <F3> to exit
bash edit_compile_run.sh 03_show_arguments.cxx $(date)      # push <F4> to exit
ls                                                          # a.out  is deleted
```

The `edit_compile_run.sh` shell script first invokes the source code into `vim`, you can edit it if you wish. And if you want to try out your alterations so far, then push <F4> in order to save it and exit the editor. Then the shell script continues his work with compiling your source code into an executable. If this compilation is successful, then the resulting binary is executed using the positional arguments we specified, when the shell script was started.

If you accidentally destroyed your source code in the editor, then press <F3> to exit without saving this wrong version. Then press <UpArow> and <Enter>, and the earlier better version is there again.

If you wish to alter the source seeing the current behaviour of your program, then press <UpArow> and <Enter> to do a next round of the edit – compile – run cycles.

If you wish to alter the list of positional arguments in order to make a different trial of your program, then push <UpArow>, then reedit the list according your taste, en press <Enter> and <F3>. This system — in case of our small programs in this semester — is fast enough, not to complain that the executable is always deleted by our IDE script after the trial phase.

Develop as a first trial the following minimalistic C/C++ program of 32 bytes.

```
int main( void ) { return 42; }
```

To this end, issue the command `bash edit_compile_run.sh 04_return_42.cxx` , do not forget, that it is enogh to type: b a s h <space> e <tab> the-source-file-name <enter>. Then put `vim` into `-- insert --` mode, enter the only line of the program, push <F4>, and that is all.

```
g++ 04_return_42.cxx ; ./a.out ; echo $?
```

If you worked carefully, 42 should appear as a result of the three commands above.

The "make", the rule based programming aid for huge developing projects

In contrary to the most common, namely "sequence based" programming languages, the useful tool `make` has a "rule based" programming language, in order to describe the "prerequisite dependencies" among the numerous source files of a huge project. It is therefore convenient to use it to handle the question of what to compile before the others and how to compile.

This semester most of our project will be small, usually of a single source file, where the use of `make` makes no too much sense. But the shell script above is so comfortable in case of a single source file, what comfort simply cannot be achieved by using `make`.

Standard output and error output in C++

Now we extend the `03_show_arguments.cxx` program with an output to the error channel. Using a copy of the program, add the single line

```
  std::cerr << argc-1 << std::endl;
```

just inbetween the body of the `for` statement, and the final `return` statement, applying the CLI activity below:

```
cd ~/i3
cp 03_show_arguments.cxx 05_show_arguments.cxx
bash edit_compile_run.sh 05_show_arguments.cxx x1 x2 x3  # push <F4> to exit
```

If you have done the smallish change carefully, then the `diff` command below should signal a difference of a single line only. (the dollars and the last 3 lines are *not* part the CLI command what you have to issue)

```
$ diff 03_show_arguments.cxx 05_show_arguments.cxx
6a7
>   std::cerr << argc-1 << std::endl;
$
```

See how coincise the output of diff is: "6a7" means, that adding the displayed line just after the 6th line of the first file, we get the 7th line of the second file, and there are no more differences.

In fact, if there is not too much difference between the `first.cxx` and `second.cxx`, then on the base of the output of

```
diff first.cxx second.cxx
```

easy to reconstruct the second source on the base of the first. Be prepared, that in the future I will sometimes give you the content of a new source file giving you the differnces only with respect to a source file you have already.

Now it is time to play around a little in the CLI.

```
g++ 05_show_arguments.cxx
./a.out x1 x2 x3
./a.out x1 x2 x3  >/dev/null               # channel std::cout is redirected
./a.out x1 x2 x3 2>/dev/null               # channel std::cerr is redirected
./a.out x1 x2 x3 &>/dev/null               # the both of them are redirected
```

The standard output `std::cout` is for the essential results of the program, why it was written. The error output `std::cerr` is for signaling the problems arising during the execution.

Discover the structure of the "C++ reference" site

Since the site https://en.cppreference.com/ is slow, therefore I provide a static copy of it, can be collected from https://math.bme.hu/~prohlep/cxx/en/cpp.html. Here it is:

- https://math.bme.hu/~prohlep/cxx/en/cpp.html

Next job is to learn how to find the descriptions of diverse language elements of a C++ program.

0. Find the description of `#include` !
   https://en.cppreference.com/w/cpp/preprocessor/include
1. Find the description of `<iostream>` !
   https://en.cppreference.com/w/cpp/header/iostream
2. Find the description of `int` and `char` !
   https://en.cppreference.com/w/cpp/language/types
3. Find the description of `main` !
   https://en.cppreference.com/w/cpp/language/main_function
4. Find the description of `for` !
   https://en.cppreference.com/w/cpp/language/for
5. Find the description of `<<` !
   https://en.cppreference.com/w/cpp/io/basic_ostream/operator_ltlt
6. Find the description of `std::cout` !
   https://en.cppreference.com/w/cpp/io/cout
7. Find the description of `std::endl` !
   https://en.cppreference.com/w/cpp/io/manip/endl
8. Find the description of `std::cerr` !
   https://en.cppreference.com/w/cpp/io/cerr
9. Find the description of `return;` !
   https://en.cppreference.com/w/cpp/language/return

An oral explanation is delivered on the lab classes. Practice the use of `cppreference` using the program below. Make it first with `bash edit_compile_run.sh 06_stdbitset.cxx` !

```
#include <iostream>                                          // 06_stdbitset.cxx
#include <bitset>

void show_bitpattern( char const * const msg, short int const val ) {
  std::cout << msg << std::bitset<16>( val ) << std::endl;
}

int main( void ) {
  short int const a = 725; show_bitpattern( "    a = " , a );
  short int const b =  ~a; show_bitpattern( "   ~a = " , b );
  short int const c = b+1; show_bitpattern( "~a +1 = " , c );
  short int const d =  -a; show_bitpattern( "   -a = " , d );
  std::cout << "~a +1 == -a   " << ( c==d ? "TRUE" : "FALSE" ) << std::endl;
}
```

The “two’s complement” in detail: https://en.wikipedia.org/wiki/Two%27s_complement .

This is the end of the week 02.

THE LABOR DUTIES ON THE WEEK 03.

POSIX, the Portable Operating System Interface (1990)

Approaching the end of the 80's, the success of the Unix becoming an adult that time, triggered the need that the huge diveristy of different versions of Unix should be regulated somehow. The result is an industry standard fundamental even today, ISO/IEC IS 9945-1:1990 or ANSI/IEEE 1003.1-1990, depending on what standardisation system you prefer. The most recent edition is of 2017, see https://pubs.opengroup.org/onlinepubs/9699919799/ .

Linux and macOS are of course accepted or certified as POSIX compatible, while *none* of the MicroSoft operating system are. This is an essential disadvantage of DOS and Windows.

WSL2 is in fact a subsystem providing POSIX services, it can be installed additionally to Win10, and it was a main prerequisites for Windows users tarting this course. And in fact the real name of our course could be: **"Developing C++ programs using POSIX environment"**. See https://en.wikipedia.org/wiki/POSIX .

```
sudo apt-get -y install manpages-dev         # man pages of POSIX C interface
```

Learn to fork

Applying `bash edit_compile_run.sh 07_execvp.cxx` , develop the program below!

```
// this is a "POSIX"-only source code

#include <cstdio>
#include <unistd.h>

int main( void ) {
  printf( "BEFORE fork\tpid=%d\n", getpid() );
  auto const finf = fork(); // <<<<<<<<<<<<<<<<<<<<<<<<<<< the magic is here
  printf( "AFTER  fork\tpid=%d forkinfo=%d\n", getpid(), finf );
  sleep( 9 );          // imagine asif a lot of statements are executed here
  printf( "Whatever statement is here, it will be executed twice, "
          "pid=%d forkinfo=%d\n", getpid(), finf );
  return 0;
}
```

The magic of this program is in the statement "`auto const finf = fork();`", but not in the "`auto`" nor in the "`const`" keywords, but in the seemingly plain function call "`fork()`".

```
man 2 fork                                       # part of the POSIX C interface
```

Remember please the "`printf`" calls in the source code, since their output at runtime will explain us what the impact of a call to "`fork()`" is. Keep track where and how the output of these statemsnts appear. The key what it is worth while to track is the beginning of these output lines, which can be either "BEFORE" or "AFTER" or "Whatever", see:

```
BEFORE fork         pid=421
AFTER  fork         pid=421 forkinfo=422
AFTER  fork         pid=422 forkinfo=0
Whatever statement is here, it will be executed twice, pid=421 forkinfo=422
Whatever statement is here, it will be executed twice, pid=422 forkinfo=0
```

• "BEFORE" the fork, i.e.: "In the beginning was" running only single instance of our programe, what we started after compiling. Process identity (`pid`) of it can happen anything, say `421` in the above particula case when this handout was written.

• "AFTER" the fork, due to the `fork()`, already a second instances of our programe is running: it is an exact copy of the original one, and it begun it's separate life exactly at the source code location of the `fork()` in question, hence all the remaining statements of the source code is executed both by the process `421`, and also by the the process `422`. The only real difference of these two processes, that value of the variable `forkinfo` is in the original "parent" process equal to the process identity of the newborn "child" process, and in the case of this "child" process the same variable contins a "zero" signaling for that process, that it is a "child" process.

• "Whatever" statement comes after the so callaed "forking", the both of the "parent" and "child" process will execute it. We see, that there can be a minor difference in the result, since we can print out the `forkinfo`. Seems to be not too much. This behaviour of "forking" appears to be very ugly, but soon we will develop further this initial program, and then the "forking" turns out to be an essential and indispensable tool in multi tasking.

New C++ language elements in this code are: https://en.cppreference.com/w/cpp/language/auto and https://en.cppreference.com/w/cpp/language/cv . See the two new library functions as well:

```
man 2 getpid                                         # part of the POSIX C interface
man 3 sleep                                          # part of the POSIX C interface
```

Separate the program of the Child- and Parent processes

The "fork information" stored into the "`forkinfo`" variable can be used to separate the Child- and Parent process specific source code fragments. Hence the program will behave differently depending on whether the process is the parent forking the child process, or the process is the child born in the forking. To this end, an "`if`" branching is needed with respect to the value of the "`forkinfo`" variable. One branch will be the Parent specific source fragment, the other brancs will be the Child specific source fragment.

```
cp 07_execvp.cxx          08_execvp.cxx
bash edit_compile_run.sh 08_execvp.cxx
```

Delete first the whole line of the "`sleep( 9 );`" statement, and the two lines of "`printf`" statement below it. Insert the 7 new lines below, right before the "`return 0;`" statement.

```
  if ( finf < 0 ) {
    printf( "ERROR: forking a child process failed\n");
  } else if ( finf == 0 ) {
    printf( "Child  process\tpid=%d forkinfo=%d\n", getpid(), finf );
  } else {
    printf( "Parent process\tpid=%d forkinfo=%d\n", getpid(), finf );
  }
```

If you worked carefully, the “`diff`” command below should give the result what you see below:

```
$ diff 07_execvp.cxx 08_execvp.cxx
11,13c11,17
<   sleep( 9 );        // imagine asif a lot of statements are executed here
<   printf( "Whatever statement is here, it will be executed twice, "
<           "pid=%d forkinfo=%d\n", getpid(), finf );
---
>   if ( finf < 0 ) {
>     printf( "ERROR: forking a child process failed\n");
>   } else if ( finf == 0 ) {
>     printf( "Child  process\tpid=%d forkinfo=%d\n", getpid(), finf );
>   } else {
>     printf( "Parent process\tpid=%d forkinfo=%d\n", getpid(), finf );
>   }
$
```

• In case of Child process, the “fork information” is zero, hence the middle branch will be executed, and hence “`Child process ...`” will be printed by this Child process.

• In case of Parent process, the “fork information” is non–zero.

— If the “fork information” is negative, then the Parent remained alone, no Child process was born, due to som forking error, and “`ERROR: forking ...`” will be printed by theh Parent process, since the first branch will be executed.

— If the “fork information” is positive, then a Child was born getting the process id stored into the “fork information”, and the Parent process keeps runing, but from now on, *in parallell with it's child*, and “`Parent process ...`” will be printed by the Parent process, since the last branch will be executed.

The positive and zero branches can contain large programs, different from each other. This case these two different activities, the Parent- and the Child activities will be running in parallel to each other. Our next task is to put the directory listing activity into the Child branch.

The Child process can deliver an illusion of “ls” command, via execvp

Our next aim is to achieve the illusion, asif or program was the usual `ls` command, extended by a few diagnostic messages.

```
cp 08_execvp.cxx         09_execvp.cxx
bash edit_compile_run.sh 09_execvp.cxx
```

We will alter or extend the new copy of our source code at 4 locations, on the base of the `diff` information. Naturally, the `diff` will give the result below only if we've already done the appropriate changes at the 4 locations.

```
$ diff 08_execvp.cxx 09_execvp.cxx
5a6
> #include <sys/wait.h>
```

1. Insert the preprocessor directive above just below the 3 already existing `#include` directives.

```
man 2 wait                                                  # press 'q' to exit
```

The function called `wait` is declared in the header `sys/wait.h`, and the man page tell us that the function call "`wait( NULL )`" will wait until all the Child proesses exit, i.e.: they finished their own work.

```
7c8
< int main( void ) {
---
> int main( int, char * argv[] ) {
```

2. Change the parameter list declaration of "`main`" from emtpy to "`int, char * argv[]`", since we need to get the positional arguments in order to create the illusion of the "`ls`" command.

```
14a16,17
>     execvp( "ls", argv );
>     printf( "ERROR: call to execvp did not take away the control.\n" );
```

3. Append the two statements above to the "child process branch".

```
man 3 execvp                                                # press 'q' to exit
```

The call to "`execvp`" will *REPLACE* the current process — in our case it is the child process — with a freshly started instance of the "`ls`" command processing the "`argv`" list of positional arguments. From now on, this "`ls`" process will be the child process of the parent process, and the original child process will never reach the next statement, the "`printf`". Unless the call to "`ls`" somehow fails, because then the original child process will not be replaced, and the "`printf`" will print the appropriate error message.

```
16a20,22
>     wait( NULL );
>     printf( "\n\nSome destruction hidden behind the "
>             "'ls' command could happen here.\n\n" );
$
```

4. Append the two statements above to the "parent process branch". While the child process delivers the expected illusion of an "ls" command, the origianl Parent process can do what it

wishes. In our demonstrative case, it wait until the Child process fulfills the expected illusion of an “`ls`” command, and then a demonstrative only message is printed.

The Parent process can serve dirty goals in the background, via a 2nd forking

Our last aim is to omit the diagnostic only messages, and to hide the possibbly destructive branch into a *background* process, in order to allow the original Parent process to exit as soon as the expected illusion of an “`ls`” command is fulfilled, — and hence the user will get back the command prompt in the terminal, and the user will be relaxed, — but we can remain there in the *background*, potentially forever.

```
cp 09_execvp.cxx         10_execvp.cxx
bash edit_compile_run.sh 10_execvp.cxx
```

As before, we will again alter or extend the new copy of our source code at 5 locations, on the base of the `diff` information. Naturally, the `diff` will give the result below only if we've already done the appropriate changes at the 5 locations.

```
$ diff 09_execvp.cxx 10_execvp.cxx
9d8
<   printf( "BEFORE fork\tpid=%d\n", getpid() );
```

1. Remove the “`printf`” from just before the “fork”.

```
11d9
<   printf( "AFTER  fork\tpid=%d forkinfo=%d\n", getpid(), finf );
```

2. Remove the “`printf`” from just after the “fork”.

```
15d12
<     printf( "Child  process\tpid=%d forkinfo=%d\n", getpid(), finf );
```

3. Remove the “`printf`” from the begining of the Child branch.

```
19d15
<     printf( "Parent process\tpid=%d forkinfo=%d\n", getpid(), finf );
```

4. Remove the “`printf`” from the begining of the Parent branch.

```
21,22c17,22
<     printf( "\n\nSome destruction hidden behind the "
<             "'ls' command could happen here.\n\n" );
---
>     auto const finf2nd = fork();
>     if ( finf2nd == 0 ) {
>       sleep( 9 );
>       printf( "\n\nSome destruction hidden behind the "
>               "'ls' command could happen here.\n\n" );
>     }
$
```

5. As the last change to the source code, *REPLACE* the "`printf`" statement — be careful, it spans to 2 lines — below the "`wait`" statement, to a more elaborate solution. Namely to go first *background* using a 2nd *forking*, then sleep 9 seconds, and then "suddenly" surprise the unaware user by a message, that we could do — hidden into background — destructive actions as well.

Developing in small steps, having always a somehow working program

You may be surprised because of wasting quite a few lines of the source code, written earlier and then thrown away later. But that's exactly what happens in tailors. First, the dress is tailored and pinned together, then you test it, they alter it, and finally the dress is sewed and the pins are removed.

CONIO, CONsole Input Output systems

• Never mix the C style, C++ style and `ncurses` style console input output system, — unless you know and exactly understand how they can survive the presence of each other.
— C style: `#include <stdio.h>` and `printf` -alike functions
— C++ style: `#include <iostream>` and `<<` -alike operators
— `ncurses` style: `#include <ncurses.h>` and `addstr`, `mvaddch` -alike functions

```
sudo apt-get -y install libncurses-dev ncurses-doc
```

CONIO based on the "ncurses" library

Applying `bash edit_compile_run.sh 11_ncurses.cxx` , develop the program below!

```
// LINK OPTIONS -lncurses
#include <ncurses.h>
```

• The first line containing the special marker "`// LINK OPTIONS`" is extremly specific to our `edit_compile_run.sh` shell script, hence it is not a portable universal knowledge. The rest of this line informs our integrated developmnet script, that the "ncurses" library file must be linked into the final binary version of our program.
• The header "`ncurses.h`" provides the declarations needed for calling the the functions that belong to the "ncurses" library.

```
int main( void ) {
  initscr();
  endwin();
  return 0;
}
```

• The "ncurses" function "`initscr`" inititalises a 2-dimensioanal interface to the character terminal pretending as if it was a 2-dimensioanal matrix of position suitable for holding diverse ASCII (or Unicode) characters.
• The "ncurses" function "`endwin`" is the closing pair of the previous function "`initscr`", it liquidates the 2-dimensioanal "ncurses" interface.

The appropriate manual pages can be invoked with the CLI command below. Let us mention that this is not an exceptional case when two different request end up at the same manual page.

```
man 3 initscr                                                # press 'q' to exit
man 3 endwin                                                 # the SAME man page
```

After editing the source and compiling, when we run this progra, we perceive not to much is any. The reason for that is, that the program consists of initialisation and liquidation of the 2-dimensioanal "ncurses" interface, and nothing else.

The case of missing header inclusion or missing library link option

It is instructive to try out how the situation looks like when we accidentally forget to include the appropriate header, or forget to take action toward linking the appropriate library file.

- The case of missing header inclusion.

— Applying `bash edit_compile_run.sh 11_ncurses.cxx` , comment out the inclusion of the header "`ncurses.h`" and try it. The message is clear:

```
11_ncurses.cxx: In function ‘int main()’:
11_ncurses.cxx:6:3: error: ‘initscr’ was not declared in this scope
    6 |   initscr();
      |   ^~~~~~~
compilation terminated due to -Wfatal-errors.
there is no executable
```

— Notice that the missing header, our accidental error is *not* mentioned! Instead, the resulting error is described in detail. Usually it is our job to find out what caused the error.

— <UpArrow> and <Enter>, and don't forget to bring it out the `#include` from the jail of the comment marker what we inserted before.

- The case of missing library link option / 1st version.

— Applying `bash edit_compile_run.sh 11_ncurses.cxx` , insert a few extraneous characters into the middle of the marker "`// LINK OPTIONS`" and try it. The message is perhaps a bit less clear than before:

```
/usr/bin/ld: /tmp/ccTlISZm.o: in function ‘main’:
/work/pp/iszap/cxx/2021feb_i3/anyag/11_ncurses.cxx:6:
                                            undefined reference to ‘initscr’
/usr/bin/ld: /work/pp/iszap/cxx/2021feb_i3/anyag/11_ncurses.cxx:7:
                                            undefined reference to ‘endwin’
collect2: error: ld returned 1 exit status
there is no executable
```

— The guiding omen here is the pattern "`error:  ld returned 1 exit status`" — this is the unambiguous sign of that the error is not in the source code itself but at the linking phase, namely some library files were not linked into the final binary. In this kind of situation the "`undefined reference to`" information is the key to what is missing. The manual pages of

"initscr" and "endwin" will tell us what header these depends on, and this will yield us an idea what library file is missing from the list of "to be linked" items. As a last resource we can google what to link if "ncurses.h" is included.
— <UpArrow> and <Enter>, and don't forget to remove the extraneous characters from the marker "// LINK OPTIONS" and try it whether now it works again.

- The case of missing library link option / 2nd version.

It is instructive to compile our program "by hand" instead of our integrated development script.

```
g++ 11_ncurses.cxx                     # here we get the error seen earlier
g++ 11_ncurses.cxx -lncurses           # the library must be on the right side
g++ -lncurses 11_ncurses.cxx           # the left side produces a link error
```

Yet another "Hello World!" demonstrating the "ncurses" library

Next goal is to put something visible into our "ncurses" program.

```
cp 11_ncurses.cxx        12_ncurses.cxx
bash edit_compile_run.sh 12_ncurses.cxx
```

Insert the "addstr", "getch" and "flushinp" statements below just in between the existing "initscr" and "endwin" statements.

```
$ diff 11_ncurses.cxx 12_ncurses.cxx
6a7,17
>   addstr(
>     "If this text is\n"
>     "    in the upper left corner, and\n"
>     "    not below the last prompt,\n"
>     "then this program\n"
>     "    uses ncurses successfully.\n"
>     "\n"
>     "Press any key to exit!\n"
>   );
>   getch();
>   flushinp();
$
```

The manual pages of the 3 new functions can be accessed by the commands below.

```
man 3 addstr
man 3 getch
man 3 flushinp
```

This is the end of the week 03.

THE LABOR DUTIES ON THE WEEK 04.

Tick–tock model by Kernighan and Pike, 1984

• The spectacular steps in a development process are those, where the newer version of the program offers more functionality. However sometimes the internal structure of the source code needs substantial redesign, reorganization and rewriting. In these steps the best practice is to keep the externally viewable functionality exectly the same.

— `11_ncurses.cxx` ... *initial version* in our development process (see last week)
— `12_ncurses.cxx` ... *more functionality* (see last week)
— `dev_ncurses.cxx` ... a *digression* on the *safer technology*, based on Object Oriented C++
— `13_ncurses.cxx` ... still the *same functionality*, but using already the *safer technology*
— `14_ncurses.cxx` ... a *digression* on the preprocessor directives of the *safer technology*
— `15_xysnake.cxx` ... *more functionality*

• `https://en.wikipedia.org/wiki/The_Unix_Programming_Environment` (1984)
The book *"The Unix Programming Environment"* still — after 37 years — has a great impact on the present culture of developing programs. The chapter 8 *"Program Development"* (of 55 pages, 233–287) is still relevant, sentece by sentence, word by word. As the iconic example, a *"HOC, Higher Order Calculator"* is developed there 6 steps, stages. HOC is a language interpreter, implementing artithmetic, strings, flow control and recursively callable functions, with arguments. The language is extended gradually from stage 1 till stage 6.

As an exception, HOC3 (stage 3) and HOC4 (stage 4) implement exactly the same language, but uses a substantially different technology inside. The earlier technology in HOC3 was simple, but was not suitable to support any more the stepwise evolution of the language. The new technology in HOC4 is much more complex, however even a modern language could be implemented using that virtual machine technology.

• `https://en.wikipedia.org/wiki/Tick%E2%80%93tock_model` (2007)
A quarter century later, Intel begun to use clean development steps: either only to shrink the lithographic scale of the die, or only develop the microarchitecture of the processor, or only optimize the given desing.

C++ resource acquisition is initialization by object constructor

• Not secure enough technology is that our source code is responsible for closing the "ncurses" session if it was started. And flush the input channel just before the closing.

• There is no real danger in case of our last stage, `12_ncurses.cxx`. However, if the program is much mor complex, and perhaps uses *fork*-ed and *execvp*-ed processes like our program `10_execvp.cxx`, then it is already not so secure, which of the concurrent processes will close the “ncurses” session. Yet another variant of hazardous situations is that if the “ncurses” session is not necessarily started, hence it my become unclear at an unexpected exit location of the program, whether is there an “ncurses” session to close or not.

• An entirely automatic mechanism is needed for closing *sure* an “ncurses” session if it was started. Most of the better languages have some sort of answer to the need of *“safe removal”* of diverse, sometimes farirly complex resources.

• A widely supported solution is the *“OnExit”* function facility. At a glance, it looks elegant, natural and very safe. But this is only a wishful thinking: the errors will show up as we design, as we dream. The common property of the diverse *“OnExit”* is, that there is a guarantee, that if the program has to exit, then the duties listed in the *“OnExit”* mechanism will be executed just before the exit. — What’s the problem here, isn’t it exactly what we need?

The safety is only an illusion in case of *“OnExit”* — since in case of a more complex program, it changes heavily time to time, what needs final closing and what doesn’t.

• A bit better mechanism is the *“Finally”* facility, what can be attached to smaller units, and not only to the whole program. Still needs arrangements “by hand”, not automatic completely.

• The best solution was suggested by the author of C++, Bjarne Stroustrup, a former abstract algebraist (MSc). His favourite sentence is: *“resource acquisition is initialization by object constructor”*, that is why the title of this section is this famous sentence.

— in C++ no special mechanism is needed for automatic safety, since the automatic construction and destruction of *“stack objects”* offer the best mechanism one can imagine
— a suitable *“guardian object”* will provide the “ncurses” subsystem
— if that object exists, then the “ncurses” service is there and alive
— just when the object was created, the “ncurses” service was started by the *“constructor”*
— just when the object will be killed, the “ncurses” service will be closed by the *“destructor”*
— if the *“guardian object”* is a *“stack object”*, then there is no room for error!,
— since a *“stack object”* is automatically killed as soon the program flow exits a program block,
— regardless of whether the exit is intentional, or due to an error, or due to an external terror.
— an extra bonus what we get charge free from as a consequence of the basic behaviour of C++ is, that the diverse resources handled by separate *“guardian objects”* will be closed in the reverse order of their start. Since the *“stack objects”* use to die in opposite order of their births: first born last dies. This in fact a very logical and desired ordering, since a later born resource my need an earlier born resource till the last seconds of it’s life. Hence the reversing of the original ordering, the opposite order is usually desired, and almost never an error.

Handle ncurses by a global “guardian object”

Develop the new guardian technology in `dev_ncurses.cxx` on base of `12_ncurses.cxx` .

```
cp 12_ncurses.cxx         dev_ncurses.cxx
bash edit_compile_run.sh dev_ncurses.cxx
```

Our aim is to separate the “ncurses” initialization and closing from the “main” function into the consctructor and destructor respectively, of a suitable “guardian object”.

```
  initscr();                                              // delete this statement
```

- Delete the line above from the beginning of the “main” function.

```
  flushinp();                                             // delete this statement
  endwin();                                               // delete this statement
```

- Delete the 2 lines above from the end of the “main” function.

```
#include <cstdlib>
```

- Insert the #include above just below the alrady existing #include line.

```
class GlobalGuardNcurses {
} the_Global_Guardian_Object_For_Ncurses;
```

- Insert the 2 lines above between the last #include line and the “main” function.

— “the_Global_Guardian_Object_For_Ncurses” is a global variable, outside any function
— the object held by this variable is *“constructed”* before executing the “main” function
— the object held by this variable is *“destructed”* after executing the “main” function
— whatever happens, the *“destruction”* takes place **exactly iff** the *“construction”* before was successful, ... this is a really high level of security, no room for error
— the type of the variable is “GlobalGuardNcurses”, which is a class
— the object held in the variable is called an *“instance”* of this class
— this class is empty right now, since there is nothing in it’s body, encosed in curly braces
— next we insert a private memberfunction, and public constructor and destructor into it

```
  static void endNcurses( void ) {
    if ( ! isendwin() ) { refresh(); flushinp(); endwin(); }
  }
```

- Insert the 3 lines above into the body of the class, i.e.: inbetween the curly braces.

— enjoy the madness free syntax of the C/C++ grammar, the body can be a single line
— this is a private static member function of the class in question,
later this semester we learn what all of these does mean
— the aim of this function to close the “ncurses” service gracefully

```
man 3 isendwin
man 3 refresh
man 3 flushinp
man 3 endwin
```

```
public:
  GlobalGuardNcurses( void ) {
    initscr(); raw(); noecho(); intrflush( stdscr, false );
    keypad( stdscr, true ); curs_set( 0 ); std::atexit( endNcurses );
  }
```

- Insert the 5 lines above jut below the previous memberfunction, before the closing brace

— this is a public *"constructor"* of the class in question

— a member of a class is a *"constructor"* of that class, **iff** it's name is identical to the class

— the aim of this function to start the "ncurses" service well configured

```
man 3 initscr
man 3 raw
man 3 noecho
man 3 intrflush
man 3 keypad
man 3 curs_set
```

https://en.cppreference.com/w/cpp/utility/program/atexit

```
  ~GlobalGuardNcurses( void ) { endNcurses(); }
```

- Insert the line above just below the *"constructor"* we have there already.

— this is the public *"destructor"* of the class in question

— a member is the *"destructor"* of the class, **iff** it's name is ~ followed by the name of the class

— this destructor calls the member function "`endNcurses`" in order to close the "ncurses" service

```
#ifndef   GlobalGuardNcurses_IS_INCLUDED
#define   GlobalGuardNcurses_IS_INCLUDED true
```

- Insert these 2 lines above just before the first `#include` line.

```
#endif // GlobalGuardNcurses_IS_INCLUDED
```

- Insert the line above just below the closing curly brace of the body of the class definition.

— The lines of `#ifndef` and `#endif` form a pair of opening and closing parentheses,

— the whole block inbetween is read by the compiler **excactly iff** at the time of reading the opening part, the symbol "`GlobalGuardNcurses_IS_INCLUDED`" is *not* yet defined

— to avoid a later second reading of the enclosed block in question, the `#define` makes defined the block-guardian synbol "`GlobalGuardNcurses_IS_INCLUDED`"

- The source code is now complete, finish editing, compile and run:

— this program behaves exactly the same way as the previous one,

— but the technology inside is safer.

```
diff 12_ncurses.cxx dev_ncurses.cxx
```

This way we can investigate the differences between the original stage and the final stage.

The tricky but simple usage of our guardian environment

```
echo 1[12]_ncurses.cxx                                  # use <UpArrow> + editing
diff 1[12]_ncurses.cxx                                  # and <Enter> extensively
diff 1[12]_ncurses.cxx | grep '^>'
diff 1[12]_ncurses.cxx | grep '^>' | cut -c 3-
diff 1[12]_ncurses.cxx | grep '^>' | cut -c 3- | head -n -1
diff 1[12]_ncurses.cxx | grep '^>' | cut -c 3- | head -n -1 >13_ncurses.cxx
```

We have just collected our real task what needs to be surrounded by the merely technical stuffs.

```
bash edit_compile_run.sh 13_ncurses.cxx
```

- Invoke our development environment with respect to the newly created source file.

— Garnish first the existing code snippet with the main function.

```
int main( void ) {
```

- Insert the line above just before the “addstr” statement.

```
  return 0;
}
```

- Insert the 2 lines above just below the “getch” statement.

```
// LINK OPTIONS -lncurses
```

- Insert the line above just to the beginning of the sourc file.

```
#define   main dev_ncurses_main
#include "dev_ncurses.cxx"
#undef    main
```

- Insert the 3 lines above just inbetween the line of “// LINK OPTIONS” and the definition of the “main” function.

— The only but non trivial trick is here:

— the #define can be considered as a source interpretation command for the source reading: “from now on, whenever “main” is written, consider it asif “dev_ncurses_main” is written,

— hence in the “#include”-ed “dev_ncurses.cxx” file,
the function “dev_ncurses_main” is defined instead of the “main” function,
avoiding the **redefinition of ‘int main()’** collision with the rest of our code,

— then the #undef tells that “from now on, forget the tricky substitution of the symbol main”,

— hence, the “main” in 13_ncurses.cxx will be the only one, in spite of the “#include”.

- Finish editing, then compile and run.
- Try out what happens, when you comment out the “#define main dev_ncurses_main”.

```
13_ncurses.cxx:7:5: error: redefinition of ‘int main()’
    7 | int main( void ) {
      |     ^~~~
compilation terminated due to -Wfatal-errors.
```

The other main function — covered by our trick — can be accessed as well

```
cp 13_ncurses.cxx        14_ncurses.cxx
bash edit_compile_run.sh 14_ncurses.cxx
```

- Keep the “ncurses” garnish, and replace the core code snipplet in the body of the “main” function, on the base of the “diff” information below:

```
$ diff 13_ncurses.cxx 14_ncurses.cxx
8,16c8,15
```

```
<   addstr(
<     "If this text is\n"
<     "    in the upper left corner, and\n"
<     "    not below the last prompt,\n"
<     "then this program\n"
<     "    uses ncurses successfully.\n"
<     "\n"
<     "Press any key to exit!\n"
<   );
---
>   addstr( "First   the 'outermost main'  is started\n" );
>   addstr( "Here below comes the 'dev_ncurses_main':\n" );
>   addstr( "========================================\n" );
>   dev_ncurses_main();
>   addstr( "========================================\n" );
>   addstr( "The  'dev_ncurses_main'  has just exited\n" );
>   addstr( "\n" );
>   addstr( "Press any key to exit the outermost main\n" );
$
```

- Finish editing, then compile and run.
- Analyse and understand the other side of our `#define` – `#undef` trick:

— on the one hand, there are two "`main`" functions and no `dev_ncurses_main` in our code,
— while on the other hand, there is a call to the seemingly non existent `dev_ncurses_main`,
— any of the "*duplication*" and the "*non existence*" should cause a fatal compilation error,
— however there is no "*duplication*" and no "*non existence*",
— thanks to our **temporary** "`main` ⟶ `dev_ncurses_main`" translation trick.

- It is a good practice to use expressive and detailed diagnostic messages, examine our code.

Nurses: a snake of unbounded length

```
cp 14_ncurses.cxx        15_xysnake.cxx
bash edit_compile_run.sh 15_xysnake.cxx
```

- Empty the body of the "`main`" function, keeping naturally the final "`return 0;`" statement.

```
  snake_event_loop();
```

- Insert the statement above into the body of the "`main`" function,

— "`snake_event_loop()`" will call the function defined in the code snippet below.

```
void snake_event_loop( void ) {
  addstr( "\n\n\n\n\tPress 'q' or 'Q' to quit\n\n\t"
          "Press KEY_RIGHT or KEY_LEFT or KEY_UP or KEY_DOWN to play\n" );
  mvaddch( 0, 0, '@' ); move( 0, 0 );
  for ( int ch=getch(); (ch!='q')&&(ch!='Q'); ch=getch() ) {
    if ( ch==KEY_LEFT ) {
      try_to_crawl( 0, -1 );
    } else if ( ch==KEY_UP ) {
      try_to_crawl( -1, 0 );
    } else if ( ch==KEY_RIGHT ) {
      try_to_crawl( 0, +1 );
    } else if ( ch==KEY_DOWN ) {
      try_to_crawl( +1, 0 );
    }
  }
}
```

- Insert the code snippet of 16 lines above just before the "`main`" function.

— `man 3 mvaddch`

— `man 3 move`

— `man 3 getch` ... also for `KEY_RIGHT`, `KEY_LEFT`, `KEY_UP` and `KEY_DOWN`

— https://en.cppreference.com/w/cpp/language/for

— https://en.cppreference.com/w/cpp/language/if

— this "`for`" cycle is an "event loop", see: https://en.wikipedia.org/wiki/Event_loop

— the function "`try_to_crawl`" called 4 times above is definded in the coce snippet below.

```
void try_to_crawl( int crawl_row, int crawl_col ) {
  int head_row=0, head_col=0; getyx( stdscr, head_row, head_col );
  int max_row=0, max_col=0; getmaxyx( stdscr, max_row, max_col );
  int next_row = crop_to_interval( head_row + crawl_row, 0, max_row );
  int next_col = crop_to_interval( head_col + crawl_col, 0, max_col );
  if ( ( next_row != head_row ) || ( next_col != head_col ) ) {
    mvaddch( head_row, head_col, '#' );
    mvaddch( next_row, next_col, '@' );
    move( next_row, next_col );
  }
}
```

- Insert the code snippet of 11 lines above just before the "`snake_event_loop`" function.

— `man 3 getyx` ... a function-like macro, see below

— `man 3 stdscr`

— `man 3 getmaxyx` ... a function-like macro, see below

- Function-like macros https://en.cppreference.com/w/cpp/preprocessor/replace :

— the statement "`getyx( stdscr, head_row, head_col );`" looks like a function call,

— however it prescribes a source code replacement just before compilation, and the resulting
— new code will put the coordinates of the cursor position into "`head_row`" and "`head_col`",
— what could not happen if "`getyx`" was a real function.

- See "`man 3 curses`" for further details about the "ncurses" service,

— see also: https://pubs.opengroup.org/onlinepubs/7908799/xcurses/curses.h.html

```
#include <algorithm>
int crop_to_interval( int value, int imin, int imax ) {
  return std::min( std::max( value, imin ), imax-1 );
}
```

- Insert the code snippet of 5 lines above just before the "`try_to_crawl`" function.

— https://en.cppreference.com/w/cpp/algorithm/min
— https://en.cppreference.com/w/cpp/algorithm/max

- Finish editing, then compile and run, ... and enjoy it for a while.

Snake / TICK phase: same functionality, based on a more advanced technology

Next aim is to reduce name space pollution, the functions will become members of a class.

```
cp 15_xysnake.cxx         16_xysnake.cxx
bash edit_compile_run.sh 16_xysnake.cxx
```

— Warning: from now on the descriptions will be even less "blind copy paste of lines" friendly.

```
class SNAKE {
  int head_row=0;
  int head_col=0;
  static int crop_to_interval( int, int, int );
  void try_to_crawl( int, int );
  SNAKE( void );
public:
  static void snake_event_loop( void );
};
```

- Insert the class declaration of 9 lines above just before the "`crop_to_interval`" function.

— https://en.cppreference.com/w/cpp/language/static ... defer reading this till "`main`"

```
int SNAKE::crop_to_interval( int value, int imin, int imax ) {
void SNAKE::try_to_crawl( int crawl_row, int crawl_col ) {
void SNAKE::snake_event_loop( void ) {
```

- Move the 3 functions above into the name space of the class `SNAKE` by prepending `SNAKE::`.

```
  SNAKE::snake_event_loop();
```

- Prepending the name space scoping prefix "`SNAKE::`" to "`snake_event_loop`" in "`main`",

— the "`static`" member function can be called without having an object of type "`SNAKE`".

```
  head_row = next_row; head_col = next_col;
```

- Replace the line “`move( next_row, next_col );`” in the body of “`SNAKE::try_to_crawl`” to the line above, since the current position wonn’t be held in the cursor position any more,
- and delete the first line of the body of “`SNAKE::try_to_crawl`”, due to the same reason.

```
SNAKE::SNAKE( void ) {
  mvaddch( 0, 0, '@' );
}
```

- Insert this constructor just below the definition of “`SNAKE::crop_to_interval`”.

```
  SNAKE the_snake;
```

- The line “`mvaddch( 0, 0, '@' ); move( 0, 0 );`” of “`SNAKE::snake_event_loop`” must be replaced by the line above, since the constructor of “`SNAKE`” will do the job.

— The variable “`the_snake`” holds an object of type “`SNAKE`”, it is an “instance” of the class.

```
  the_snake.try_to_crawl( ..., ... );
```

- Prepend the object selector “`the_snake.`” prefix to the function call of the member function “`try_to_crawl`” — occuring 4 times in the body of “`SNAKE::snake_event_loop`”.

— Member function “`try_to_crawl`” of class “`SNAKE`” is called w.r.t. the object “`the_snake`”.

- Finish editing, then compile and run, ... and enjoy it for a while.

Snake / TOCK phase: a snake of finite bounded length = advanced functionality

The aim of this “tock phase” is to make the length of the snake bounded, hence it has to memorize it’s body, otherwise the cleanup after it’s tail can not be done.

```
cp 16_xysnake.cxx          17_xysnake.cxx
bash edit_compile_run.sh 17_xysnake.cxx
```

— Due to the object oriented technology taken into use in the previous “tick phase”, it is *not* a big issue any more to make the snake more advanced by equipping with memory.

```
#include <utility>
#include <list>
```

- Insert the 2 additional “`#include`” lines above just above the declaration of the class “`SNAKE`”.

— “`std::find`” ∈ “`<algorithm>`”: https://en.cppreference.com/w/cpp/algorithm/find

— “`std::pair`” ∈ “`<utility>`”, see https://en.cppreference.com/w/cpp/utility/pair

— “`std::list`” ∈ “`<algorithm>`”: https://en.cppreference.com/w/cpp/container/list

```
  typedef std::pair<int,int> xypoint;
  typedef std::list<xypoint> xysnake;
  std::size_t const maxlen;
  xysnake the_body;
  void keeplength( void );
```

- Insert the 5 lines above as just the new first 5 lines of the declaration of the class “`SNAKE`”,

— 2 new types are introduced, the “`xypoint`” and “`xysnake`”,

— “`maxlen`” is the intended length of the snake, after it’s growth phase,

— "the_body" of the snake is a list of points, the poinst are pairs of the coordinates,
— the member fuction "keeplength" handles the tail of the snake, keeping it's length bounded.

```
  SNAKE( std::size_t );
```

- Change the arrity "void" of the constructor "SNAKE" to the unary type "std::size_t",

```
void SNAKE::keeplength( void ) {
  if ( maxlen < the_body.size() ) {
    xypoint const tail = the_body.back();
    the_body.pop_back();
    auto const pos = std::find( the_body.cbegin(), the_body.cend(), tail );
    if ( pos == the_body.cend() ) {
      mvaddch( tail.first, tail.second, ' ' );
    }
  }
}
```

- Insert the definition of the member function just below the declaration of it's class.

— There is nothing to do unless the "size" of the list "the_body" is larger than the "maxlen",
— the 1st statement stores a copy of the last element of the list "the_body",
— the 2nd statement removes the last element from the list "the_body",
— the 3rd statement tries to "std::find" the "tail" in the remaining part of "the_body",
— if the "pos" found by "std::find" is beyond "the_body", i.e.: it is "the_body.cend()", then the old location of the tail can be and must be overwritten by a space character.

It is ensured that the view of the snake will not be messed up, even if the snake wraps itself.

```
    the_body.emplace_front( next_row, next_col );
    keeplength();
```

- Insert the 2 lines above just above the first "mvaddch" statement in the body of the large "if" statement, in the definition of "SNAKE::try_to_crawl".

— The new head information is attached to the "front" of the list of "the_body",
— and the sequence of the "todo list" is extended by handling the tail by "keeplength".

```
SNAKE::SNAKE( std::size_t m ) : maxlen(m) {
```

- Replace the leading line of the definition of the constructor "SNAKE::SNAKE".

— The phrase ": maxlen(m)" will initialise the data member "maxlen" by the value "m".

```
  the_body.emplace_front( 0, 0 );
```

- Insert the line above just before the "mvaddch" statement in the constructor "SNAKE::SNAKE".

— In the beginning the snake consists of a head only, but it must be recorded into "the_body".

```
  SNAKE the_snake ( 32 );
```

- Add the initialisation "(32)", to the introduction of the variable "the_snake" in "main".
- Finish editing, then compile and run, ... and enjoy it for a while.

This is the end of the week 04.

THE LABOR DUTIES ON THE WEEK 05.

Snake / TICK phase: the functionality is kept, the technology is developed

The aim of this "tick phase" is to enhance the class and it's constructor to support:

1. different starting points — implementing the "-1 convention" — to different snakes, and
2. different characters for marking the body, in order to distinguish the diverse snakes.

```
cp 17_xysnake.cxx         18_xysnake.cxx
bash edit_compile_run.sh 18_xysnake.cxx
```

— Let us alter the declaration of the class "SNAKE" at 3 locations.

```
  char const body;
```

- Insert this constant data member declaration inbetween "the_body" and "keeplength",

— a constant member can be initialised in the initialisation list of the constructor *only*.

```
  static int neg_from_end( int, int );
```

- Insert this static member function declaration, just below the "crop_to_interval",

— it does not need and will not get the object in question upon call, since it is "static".

```
  SNAKE( std::size_t, char='#', int=0, int=0 );
```

- Extend the parameter list of the constructor by 3 additional members,

— the last 2 of them can be omitted upon call, since they have defaults.

```
int SNAKE::neg_from_end( int value, int limit ) {
  return ( value < 0  ?  limit + value  :  value );
}
```

- Insert this member function definition just below the "SNAKE::crop_to_interval" one,

— the "-1 convention" is: if the offset is negative, then it is relative to the upper bound.

```
  mvaddch( head_row, head_col, body );
```

- Change the hash mark to the value of the data member "body", in "SNAKE::try_to_crawl".

```
SNAKE::SNAKE( std::size_t m, char b, int rr, int cc ) : maxlen(m), body(b) {
  int max_row=0, max_col=0; getmaxyx( stdscr, max_row, max_col );
  head_row = crop_to_interval( neg_from_end( rr, max_row ), 0, max_row );
  head_col = crop_to_interval( neg_from_end( cc, max_col ), 0, max_col );
  the_body.emplace_front( head_row, head_col );
  mvaddch( head_row, head_col, '@' );
}
```

- The header of the definition of the constructor is changed a little,
- the first 3 lines are new: they normalize the intended row and column values,
- the last 2 lines are almost the original ones, but now they use the intended values.

- Finish editing, then compile and run, ... and enjoy it for a while.

Snake / TOCK phase: 9 snakes in parallel = even more advanced functionality

The aim of this “tock phase” is to have 9 snakes, considering each other as an obstacle.

```
cp 18_xysnake.cxx        19_xysnake.cxx
bash edit_compile_run.sh 19_xysnake.cxx
```

— 4 minor changes in the sourc code will essentially enhance the functionality of the program.

```
    if ( mvinch( next_row, next_col ) == ' ' ) {
      // the 5 lines of the original body of the enclosing if statement
    }
```

- Insert an additional “`if`” statement into the member function “`SNAKE::try_to_crawl`”, the new “`if`” must be inside of the existing one, and
  it must surround the the 5 lines of the original body of the enclosing if statement.

— This additional condition will consider everything as an obstacle, except for the spaces.

```
  SNAKE snakes[] = {
    { 32, '1',  4, -9 }, { 32, '2',  2,  4 }, { 32, '3', -3,  4 },
    { 32, '4', -3, -5 }, { 32, '5',  2, -5 }, { 32, '6',  0,  0 },
    { 32, '7', -1,  0 }, { 32, '8', -1, -1 }, { 32, '9',  0, -1 }
  };
  std::size_t const ss = sizeof(snakes)/sizeof(snakes[0]);
  SNAKE * the_snake = & snakes[0];
```

- Change the single line “`SNAKE the_snake ( 32 );`” to the 7 lines above, in the definition of the “`SNAKE::snake_event_loop`”, ... the variable “`the_snake`” has become a pointer,

— a pointer to a member of the array “`snakes[]`”, of length “`ss`” (Size of Snakes).

```
      the_snake->try_to_crawl( 0, -1 );
      the_snake->try_to_crawl( -1, 0 );
      the_snake->try_to_crawl( 0, +1 );
      the_snake->try_to_crawl( +1, 0 );
```

- Change the member selector dot to an arrow, at 4 locations in “`SNAKE::snake_event_loop`”,

— since the “`the_snake`” holds a pointer to the object, instead of the object itself.

```
    } else {
      the_snake = & snakes[ ( ch - '1' ) % ss ];
    }                          // WARNING: this closing '}' is already overthere
```

- Append this “`else`” branch to the huge “`if`”-tree in the “`SNAKE::snake_event_loop`”,

— this will change which of the snakes the pointer “`the_snake`” is pointing to.

- Finish editing, then compile and run, ... and enjoy it for a while.

Ncurses: key codes and define key

```
cp 13_ncurses.cxx        20_keycode.cxx
bash edit_compile_run.sh 20_keycode.cxx
```

- Empty the body of the "`main`" function, keeping naturally the final "`return 0;`" statement.

```
  mvaddstr( 0, 0, "Press 'q' or 'Q' to quit" ); curs_set( 2 );
  for ( int row=1, ich=0; ( ich!='q' ) && ( ich!='Q' ) ; ++row ) {
    if ( LINES <= row ) { row = 1; }
    move( row, 0 ); clrtoeol();
    ich = getch();
    mvprintw( row, 0, "0x%04x", ich );  clrtoeol();
  }
```

- Insert the 7 lines above just into the beginning of the body "`main`" function,

— `man 3 lines`   summarizes the variables provided by "ncurses",

— `man 3 clrtoeol`   `CLeaR TO End Of Line`,

— `man 3 mvprintw`   it works much like the "`printf`".

- Finish editing, then compile and run, ... and test the operation of the various keys.

```
  int const vt100_home = 991;  define_key( "\x1b[1~",  vt100_home );
  int const vt100_end  = 992;  define_key( "\x1b[4~",  vt100_end );
  int const vt100_f3   = 993;  define_key( "\x1b[13~", vt100_f3 );
  int const vt100_f4   = 994;  define_key( "\x1b[14~", vt100_f4 );
```

- Insert the 4 lines above just into the beginning of the body of the "`main`" function,

— `man 3 define_key`   (by Thomas Dickey).

- Finish editing, then compile and run, ... and test the operation of the special keys.

Ncurses: exit on Ctrl-X, print in reversed, draw "line graphics" border

```
cp 13_ncurses.cxx         21_attribs.cxx
bash edit_compile_run.sh 21_attribs.cxx
```

- Empty the body of the "`main`" function, keeping naturally the final "`return 0;`" statement.

```
  move( LINES/2, COLS/2 ); curs_set( 2 );
  const char CTRL_X = 'X'-'@';
  for ( int ch=getch(); ch!=CTRL_X ; ch=getch() )  {
  }
}
```

- Insert the code snippet above into the body of "`main`", ... save, compile, run!

```
void draw_help( void ) {
  attron( A_REVERSE );
  mvaddstr( 1,1, " control-X   exit                           " );
  attroff( A_REVERSE );
}
```

- Insert the code snippet above just before the definition of "`main`", ... save, compile, run!

```
  draw_help();
```

- Insert the code line above as the first line of the body of “`main`”, ... save, compile, run!

```
void draw_frame( int startrow, int startcol, int endrow, int endcol ) {
  mvaddch( startrow, startcol, ACS_ULCORNER );
  for ( int i=startcol+1; i<endcol; ++i ) {
    addch( ACS_HLINE );
  }
  addch( ACS_URCORNER );
  for ( int i=startrow+1; i<endrow; ++i ) {
    mvaddch( i, startcol, ACS_VLINE );
    mvaddch( i, endcol, ACS_VLINE );
  }
  mvaddch( endrow, startcol, ACS_LLCORNER );
  for ( int i=startcol+1; i<endcol; ++i ) {
    addch( ACS_HLINE );
  }
  addch( ACS_LRCORNER );
}
```

- Insert the code snippet above just before the definition of “`draw_help`”, save, compile, run!

— `man 3 addch` ... see the section “Line Graphics”

```
  draw_frame( 0, 0, LINES-1, COLS-1 );
```

- Insert the code line above as the first line of the body of “`main`”, ... save, compile, run!
- Finish editing, then compile and run, ... and test it.

Ncurses: move around inside the line graphical border

```
cp 21_attribs.cxx        22_attribs.cxx
bash edit_compile_run.sh 22_attribs.cxx
```

— An “event loop” is introduced, similar one to what in “`15_xysnake.cxx`” was applied.

```
    int row=0, col=0; getyx( stdscr, row, col );
    switch ( ch ) {
    case KEY_UP:
      if ( 1 < row ) { move( row-1, col ); }
      break;
    case KEY_DOWN:
      if ( row < LINES-2 ) { move( row+1, col ); }
      break;
    case KEY_LEFT:
      if ( 1 < col ) { move( row, col-1 ); }
      break;
    case KEY_RIGHT:
      if ( col < COLS-2 ) { move( row, col+1 ); }
      break;
    case '\n':
      if ( row < LINES-2 ) { move( row+1, 1 ); } else {  move( row, 1 ); }
      break;
    default:
      break;
    }
```

- Insert the code snippet above into the empty body of the “`for`” cycle in the “`main`”,

— see: https://en.cppreference.com/w/cpp/language/switch .

```
  mvaddstr( 2,1, " arrow key   to move around         " );
```

- Insert the help information above just below the “`mvaddstr`” statement in “`draw_help`”.
- Finish editing, then compile and run, ... and test it.

Ncurses: print letters, animate the backspace

```
cp 22_attribs.cxx         23_attribs.cxx
bash edit_compile_run.sh 23_attribs.cxx
```

— The only non trivial task is to proceed into the next or previous line if it makes sense.

```
      if ( ( 32 <= ch ) && ( ch <= 126 ) ) {
        addch( ch );
        if ( col == COLS-2 ) {
          if ( row < LINES-2 ) {
            move( row+1, 1 );
          } else {
            move( row, col );
          }
        }
      }
```

- Insert the code snippet above into the empty body of the “`default:`” branch of the “`switch`”.

```
    case KEY_BACKSPACE:
      if ( 1 < col ) {
        move( row, col-1 );
        addch( ' ' );
        move( row, col-1 );
      } else if ( 1 < row ) {
        move( row-1, COLS-2 );
        addch( ' ' );
        move( row-1, COLS-2 );
      }
      break;
```

- Insert the "`case`" branch above just before the "`default:`" branch of the "`switch`".

```
  mvaddstr( 3,1, " letter      print a letter in mode " );
```

- Insert the help information above just below the last "`mvaddstr`" statement in "`draw_help`".
- Finish editing, then compile and run, ... and test it.

Ncurses: bold, reversed and underlined mode

```
cp 23_attribs.cxx         24_attribs.cxx
bash edit_compile_run.sh 24_attribs.cxx
```

— As a highlight of our preparations so far, we can try different ways of displaying the letters.

```
int main( void ) {
  const char CTRL_B = 'B'-'@';  bool bold = false;
  const char CTRL_R = 'R'-'@';  bool reverse = false;
  const char CTRL_U = 'U'-'@';  bool underline = false;
  const char CTRL_N = 'N'-'@';
```

- Insert the 4 lines above just before the "`for`" statement in the "`main`" function.

```
    case CTRL_B:
      bold ^= true;
      if ( bold ) {attron(A_BOLD);} else {attroff(A_BOLD);}
      break;
    case CTRL_R:
      reverse ^= true;
      if ( reverse ) {attron(A_REVERSE);} else {attroff(A_REVERSE);}
      break;
    case CTRL_U:
      underline ^= true;
      if ( underline ) {attron(A_UNDERLINE);} else {attroff(A_UNDERLINE);}
      break;
    case CTRL_N:
      bold=reverse=underline=false; attroff(A_BOLD|A_REVERSE|A_UNDERLINE);
      break;
```

- Insert the 4 "`case`" branches above just to the beginning of the body of the "`switch`".

— `man 3 attron` ... and ... `man 3 attroff`

— for ^= see: https://en.cppreference.com/w/cpp/utility/bitset/operator_logic

```
  mvaddstr( 4,1, " control-B   toggle Bold        mode " );
  mvaddstr( 5,1, " control-R   toggle Reversed    mode " );
  mvaddstr( 6,1, " control-U   toggle Underlined mode " );
  mvaddstr( 7,1, " control-N   back to Normal    mode " );
```

- Insert the help information above just below the last “`mvaddstr`” statement in “`draw_help`”.
- Finish editing, then compile and run, ... and test it.

Ncurses: foreground- and background colors, with “guardian objects”

```
cp 13_ncurses.cxx        25_colors.cxx
bash edit_compile_run.sh 25_colors.cxx
```

- Empty the body of the “`main`” function, keeping naturally the final “`return 0;`” statement.

```
short const colorcodes[] = {
  COLOR_BLACK, COLOR_RED, COLOR_GREEN, COLOR_YELLOW,
  COLOR_BLUE, COLOR_MAGENTA, COLOR_CYAN, COLOR_WHITE
};
```

- Insert the variable definition above just before the “`main`” function.

— The aim of this array of length 8 is to have an *own* enumeration of colors from 0 to 7.

— `man 3 color`

— https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/color.html

```
char const * const colornames[] = {
  "BLACK", "RED", "GREEN", "YELLOW", "BLUE", "MAGENTA", "CYAN", "WHITE"
};
```

- Insert the variable definition above just before the definition of “`colorcodes`”.

— The aim of this array to have the human readable names for the colors held in “`colorcodes`”.

```
class USE_COLORPAIR {
  short const pair_in_use;
public:
  USE_COLORPAIR( short index ) : pair_in_use( index ) {
    attron( COLOR_PAIR( pair_in_use ) );
  }
  ~USE_COLORPAIR( void ) {
    attroff( COLOR_PAIR( pair_in_use ) );
    attroff( A_BOLD | A_REVERSE | A_UNDERLINE | A_BLINK );
  }
};
```

- Insert the class definition above just before the “`main`” function.

— There is always a pair of foreground- and background colors in use for the next characters,

— the constructor of “`USE_COLORPAIR`” introduces such a pair of colors,

— the destructor of this class restores the state to the neutral default,

— the color pairs held in an array are identified by an index of type “`short`”.

```
  if ( start_color() != ERR ) {
    assume_default_colors( COLOR_WHITE, COLOR_BLACK );
    for ( short bg=0; bg<8; ++bg ) {
      mvprintw( 0, 10*bg, "%-10s", colornames[bg] );
      for ( short fg=0; fg<8; ++fg ) {
        short const pair_index = (short) ( 1 + 8*bg + fg );
        init_pair( pair_index, colorcodes[fg], colorcodes[bg] );
        USE_COLORPAIR tmp { pair_index };
        mvprintw( 2+3*fg, 10*bg, "  %-8s", colornames[fg] );
        attron( A_BOLD );
        mvprintw( 3+3*fg, 10*bg, "  %-8s", colornames[fg] );
      }
    }
    refresh();
    getch();
  }
```

- Insert the code snippet above into the body of the "`main`" function.

— It is up to our taste a particular color pair what "`pair_index`" identiy will get,

— "`init_pair`" function sets up the color pair at "`pair_index`" with the "`fg`" and "`bg`" colors,

— the object "`tmp`" will be a guardian object providing the color pair at "`pair_index`",

— `man 3 start_color`   `man 3 assume_default_colors`"   `man 3 init_pair` ,

— observe that 2 of the 3 manpage commands will open the same manpage as `man 3 color` .

- Finish editing, then compile and run, ... and test it.

— Notice, that the yellow and white are in fact brown and gray respectively, this is a tradition.

Ncurses: change the R–G–B parameters of the colors in the color palette

```
cp 25_colors.cxx          26_colors.cxx
bash edit_compile_run.sh 26_colors.cxx
```

- Our aim now is to make the yellow and white really yellow and white.

```
    init_color( COLOR_WHITE, 1000, 1000, 1000 );
    init_color( COLOR_YELLOW, 1000, 1000, 0 );
    refresh();
    getch();
```

- Insert the 4 lines above just below the "`getch();`" in the body of the "`main`" function.

— `man 3 init_color` the color indexed by "`COLOR_YELLOW`" will be 100% red + 100% green.

```
    short wr=0, wg=0, wb=0; color_content( COLOR_WHITE,  &wr, &wg, &wb );
    short yr=0, yg=0, yb=0; color_content( COLOR_YELLOW, &yr, &yg, &yb );
```

- Insert the 2 lines above just before the first "`init_color`",

— in order to record the R–G–B decomposition of the traditional yellow and white.

```
    GlobalGuardNcurses::endNcurses();
    printf( "%d %d %d %d\n", COLOR_WHITE,  wr, wg, wb );
    printf( "%d %d %d %d\n", COLOR_YELLOW, yr, yg, yb );
```

- Insert the 3 lines above just below the last “`getch`”,

— notice how the static member “`endNcurses`” of “`GlobalGuardNcurses`” can be called,

— finishing the “ncurses” session, we can write the information in the traditional way.

- Finish editing, then compile, ... and notice the error message.

— The problem is, that “`GlobalGuardNcurses::endNcurses`” is *not* “`public`”,

— we will make it “`public`”, ... where?, ... in “`dev_ncurses.cxx`” !

```
bash edit_compile_run.sh dev_ncurses.cxx
```

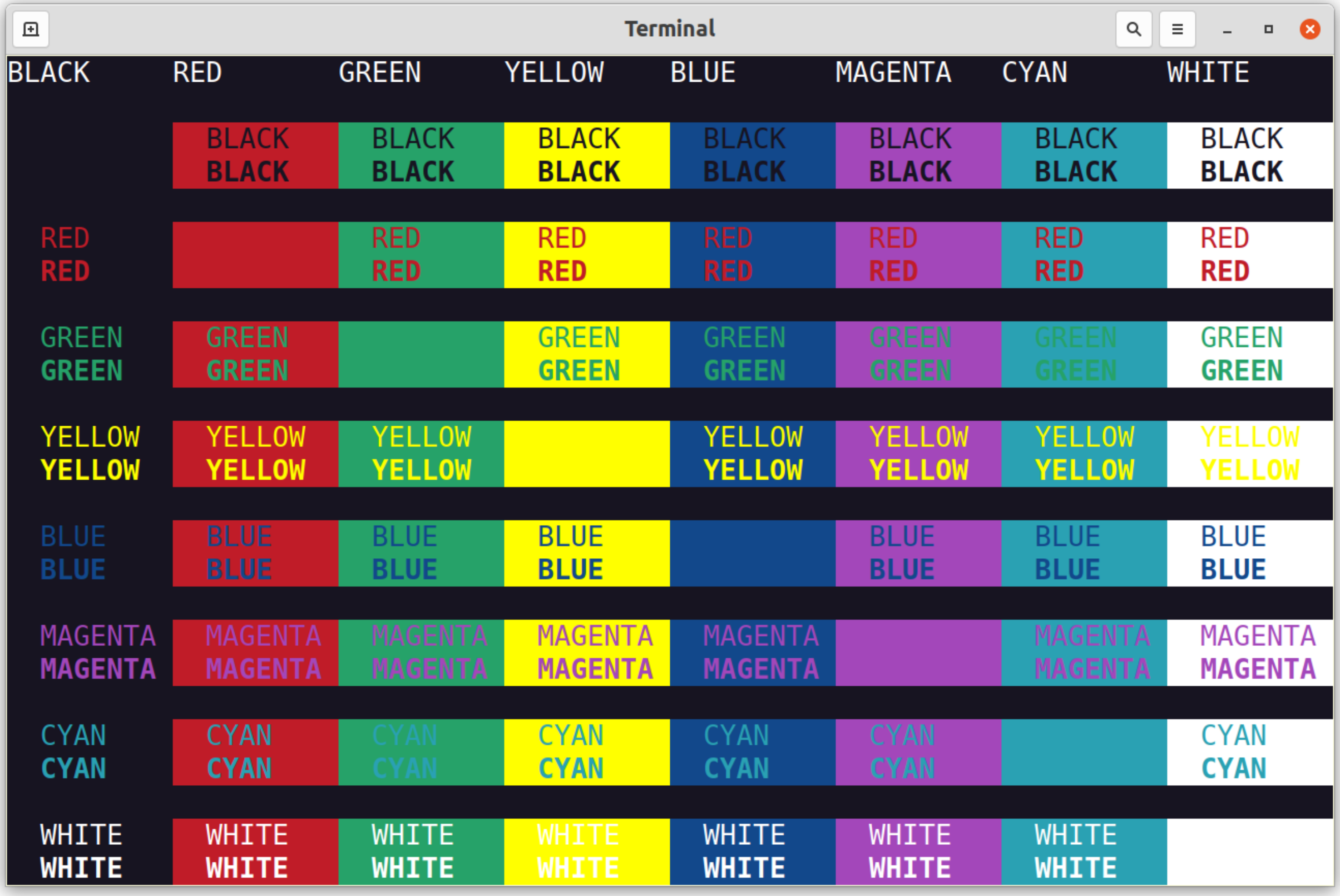
- Change the keyword “`class`” to “`struct`” in the definition of “`GlobalGuardNcurses`”,
- and remove the complete line of “`public:`” .

— The default visibility in a “`struct`” is “`public`”, hence “`endNcurses`” will be accessible.

- Finish editing, then compile and run.

```
bash edit_compile_run.sh 26_colors.cxx
```

- Finish editing, then compile and run, ... and test it.



```
BLACK     RED       GREEN     YELLOW    BLUE      MAGENTA   CYAN      WHITE

            BLACK     BLACK     BLACK     BLACK     BLACK     BLACK     BLACK
            BLACK     BLACK     BLACK     BLACK     BLACK     BLACK     BLACK

  RED                 RED       RED       RED       RED       RED       RED
  RED                 RED       RED       RED       RED       RED       RED

  GREEN     GREEN               GREEN     GREEN     GREEN     GREEN     GREEN
  GREEN     GREEN               GREEN     GREEN     GREEN     GREEN     GREEN

  YELLOW    YELLOW    YELLOW              YELLOW    YELLOW    YELLOW    YELLOW
  YELLOW    YELLOW    YELLOW              YELLOW    YELLOW    YELLOW    YELLOW

  BLUE      BLUE      BLUE      BLUE                BLUE      BLUE      BLUE
  BLUE      BLUE      BLUE      BLUE                BLUE      BLUE      BLUE

  MAGENTA   MAGENTA   MAGENTA   MAGENTA   MAGENTA             MAGENTA   MAGENTA
  MAGENTA   MAGENTA   MAGENTA   MAGENTA   MAGENTA             MAGENTA   MAGENTA

  CYAN      CYAN      CYAN      CYAN      CYAN      CYAN                CYAN
  CYAN      CYAN      CYAN      CYAN      CYAN      CYAN                CYAN

  WHITE     WHITE     WHITE     WHITE     WHITE     WHITE     WHITE
  WHITE     WHITE     WHITE     WHITE     WHITE     WHITE     WHITE
```

This is the end of the week 05.

THE LABOR DUTIES ON THE WEEK 06.

Collect first the source codes, and then analyse them.

```
scp -r NAME@leibniz.math.bme.hu:~prohlep/public_html/i3/* .    # replace NAME
```

Member access operators, function call conventions

```
bash edit_compile_run.sh 27_funcall.cxx
```

— https://en.cppreference.com/w/cpp/language/operator_member_access
— https://en.cppreference.com/w/cpp/language/operators
— https://en.cppreference.com/w/cpp/language/reference
— https://en.cppreference.com/w/cpp/language/reference_initialization
— https://en.cppreference.com/w/cpp/language/aggregate_initialization

Overview of the most typical class members

```
  bash edit_compile_run.sh 28_class.cxx
```

— https://en.cppreference.com/w/cpp/language/classes
— https://en.cppreference.com/w/cpp/language/data_members
— https://en.cppreference.com/w/cpp/language/member_functions
— https://en.cppreference.com/w/cpp/language/constructor
— https://en.cppreference.com/w/cpp/language/converting_constructor

- there are 6 special members:

— https://en.cppreference.com/w/cpp/language/destructor
— https://en.cppreference.com/w/cpp/language/default_constructor
— https://en.cppreference.com/w/cpp/language/copy_constructor
— https://en.cppreference.com/w/cpp/language/move_constructor
— https://en.cppreference.com/w/cpp/language/copy_assignment
— https://en.cppreference.com/w/cpp/language/move_assignment

```
vimdiff     28_class.cxx 29_class.cxx
bash edit_compile_run.sh 29_class.cxx
```

— see: “const- and volatile-qualified member functions” at
https://en.cppreference.com/w/cpp/language/member_functions
— https://en.cppreference.com/w/cpp/string/byte/strlen
— https://en.cppreference.com/w/cpp/string/byte/strncpy
— https://en.cppreference.com/w/cpp/memory/new/operator_new
— https://en.cppreference.com/w/cpp/memory/new/operator_delete
— https://en.cppreference.com/w/cpp/language/new
— https://en.cppreference.com/w/cpp/language/delete

```
vimdiff     29_class.cxx 30_class.cxx
bash edit_compile_run.sh 30_class.cxx
```

— getter & setter member functions

— warning: “`std_cout`” and “`test_1`” are not members of the class in question

— “`std_cout`” writes the address of the object, then the string and value held in the object

— “`N obj1;`” activates the default constructor, hence the string is empty and the value is zero

— “`obj1.UpdateWith( "foo bar baz", 42 );`” alters the string and the value accordingly

```
vimdiff     30_class.cxx 31_class.cxx
bash edit_compile_run.sh 31_class.cxx
```

— yet another 3 constructors, testing the diverse initialisation possibilities

— https://en.cppreference.com/w/cpp/language/initialization

```
vimdiff     31_class.cxx 32_class.cxx
bash edit_compile_run.sh 32_class.cxx
```

— copy constructor & copy assignment

— https://en.cppreference.com/w/cpp/language/this

— never forget the “`if ( this != & that ) {`” singularity test in assignment operators!

— why there is no need of singularity test in case of copy constructors?

```
vimdiff     32_class.cxx 33_class.cxx
bash edit_compile_run.sh 33_class.cxx
```

— move constructor & move assignment

— https://en.cppreference.com/w/cpp/utility/move

— compare the move- and copy versions!

— in case of move, we make no new copies, but we rob the original

— the object robbed out should be left in a consistent state, so we reset them to zero

```
vimdiff     33_class.cxx 34_class.cxx
bash edit_compile_run.sh 34_class.cxx
```

— debug information macros

— https://en.cppreference.com/w/cpp/preprocessor/replace

— the macro definition must be on one logical line, hence the “backslash enter”-s at the breaks

— https://en.cppreference.com/w/cpp/language/reinterpret_cast

— the decorations “`///////`” and “`<<<<`” identify the diagnostic messages, whether it is coming from the “`ALLOCATION_INFO`” or “`ASSIGNMENT_INFO`” macros respectively

This is the end of the week 06.

THE LABOR DUTIES ON THE WEEK 07.

0-20 points: week 7, presentation by desktop sharing on MS Teams

Due to my health conditions (I suffer in corona since two weeks) I can not conduct enough meetings needed to ineterview everybody separatedly.

Instead, there will be a written questionary during the time of the lab classes, i.e.: Thursday morning for the Hungarian students, and Friday morning for the English course attendies.

I will not start any ms–teams meeting for these lab classes. Instead, you will get a Neptun email: (1) where you will find the problems what you have to solve, and submit, (2) how soon and how to submit the answers.

There will be 7 question, each of 3 points. You have to be able to explain the notions occuring in our programs and the details of the programs we learned. Or you have to make minor alterations in the existing programs in order to achieve a different behaviour I ask for. Or you have to be able — writing or fixing small programs — to apply the solutions and tricks found in our example programs. Hence you have to understand the role of each fragment of our programs. In other words: not enough to know the programs, but you have to "own" each tiny portion of the source codes.

Solutions to the first three problems

— `https://math.bme.hu/~prohlep/i3/35_week07p1.cxx`

— `https://math.bme.hu/~prohlep/i3/36_week07p2.cxx`

— `https://math.bme.hu/~prohlep/i3/37_week07p3.cxx`

• You can get all the resources by the command suggested at the beginning of the 6th week:

```
scp -r NAME@leibniz.math.bme.hu:~prohlep/public_html/i3/* .   # replace NAME
```

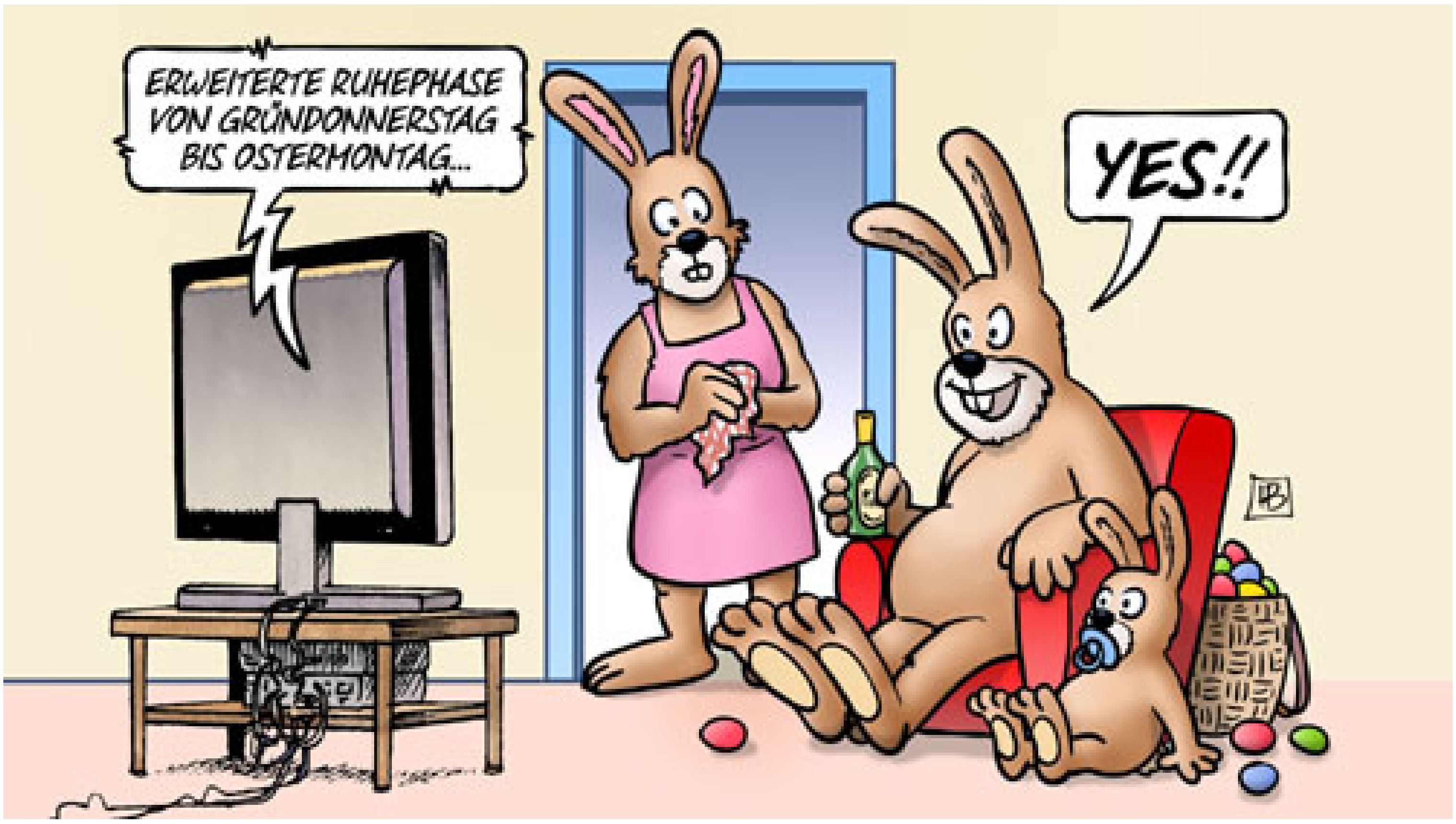
The cache of web browsers

Never forget about the caching system of web browsers. Learn to clear the cache of your favourite web browser if you want to collect the fresh version of what you are waiting for.

This is the end of the week 07.

THE LABOR DUTIES ON THE WEEK 08.

Good Thursday and Good Friday



This is the end of the week 08.

THE LABOR DUTIES ON THE WEEK 09.

- This week we make a few reflections concerning the last two lectures.

- Obtain please the new source codes, issuing the command below.

```
scp -r NAME@leibniz.math.bme.hu:~prohlep/public_html/i3/* .   # replace NAME
```

- The detailed explanation of the new programs will be delivered in the laboratory class.

Variadic arguments

— https://en.cppreference.com/w/cpp/language/overload_resolution

— https://en.cppreference.com/w/cpp/language/adl

— https://en.cppreference.com/w/cpp/language/variadic_arguments

— https://en.cppreference.com/w/cpp/utility/variadic

```
bash edit_compile_run.sh 38_variadic.cxx
```

```
bash edit_compile_run.sh 39_variadic.cxx
```

Time consuming linear search in aggregate object

— https://en.cppreference.com/w/c/preprocessor/replace

— https://en.cppreference.com/w/cpp/algorithm/upper_bound

— https://en.cppreference.com/w/cpp/container/vector

— https://en.cppreference.com/w/cpp/container/list

— https://en.cppreference.com/w/cpp/chrono/c/time

— https://en.cppreference.com/w/cpp/language/static_cast

— https://en.cppreference.com/w/cpp/numeric/random/srand

— https://en.cppreference.com/w/cpp/chrono/c/clock

— https://en.cppreference.com/w/cpp/numeric/random/rand

— https://en.cppreference.com/w/c/chrono/CLOCKS_PER_SEC

```
bash edit_compile_run.sh 40_memcache.cxx
```

Time consuming insertion into aggregate object

— https://en.cppreference.com/w/cpp/iterator/begin

— https://en.cppreference.com/w/cpp/iterator/end

```
vimdiff  40_memcache.cxx 41_memcache.cxx
bash edit_compile_run.sh 41_memcache.cxx
```

A well equipped class of polinoms of integer coefficienst

```
bash edit_compile_run.sh 42_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/default_constructor

— https://en.cppreference.com/w/cpp/language/destructor

```
vimdiff   42_polinom.cxx 43_polinom.cxx
bash edit_compile_run.sh 43_polinom.cxx
```

```
vimdiff   43_polinom.cxx 44_polinom.cxx
bash edit_compile_run.sh 44_polinom.cxx
```

— https://en.cppreference.com/w/cpp/container/vector

```
vimdiff   44_polinom.cxx 45_polinom.cxx
bash edit_compile_run.sh 45_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/copy_constructor

— https://en.cppreference.com/w/cpp/language/operator_assignment

```
vimdiff   45_polinom.cxx 46_polinom.cxx
bash edit_compile_run.sh 46_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/move_constructor

— https://en.cppreference.com/w/cpp/language/move_assignment

```
vimdiff   46_polinom.cxx 47_polinom.cxx
bash edit_compile_run.sh 47_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/operator_other

```
vimdiff   47_polinom.cxx 48_polinom.cxx
bash edit_compile_run.sh 48_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/operator_member_access

```
vimdiff   48_polinom.cxx 49_polinom.cxx
bash edit_compile_run.sh 49_polinom.cxx
```

— https://en.cppreference.com/w/cpp/io/basic_ostream/operator_ltlt2

```
vimdiff   49_polinom.cxx 50_polinom.cxx
bash edit_compile_run.sh 50_polinom.cxx
```

— https://en.cppreference.com/w/cpp/language/operator_arithmetic

```
vimdiff   50_polinom.cxx 51_polinom.cxx
bash edit_compile_run.sh 51_polinom.cxx
```

```
vimdiff   51_polinom.cxx 52_polinom.cxx
bash edit_compile_run.sh 52_polinom.cxx
```

```
vimdiff   52_polinom.cxx 53_polinom.cxx
bash edit_compile_run.sh 53_polinom.cxx
```

This is the end of the week 09.

THE LABOR DUTIES ON THE WEEK 10.

[not finished yet]