Advanced Programming

https://github.com/pcafrica/advanced_programming_2023-2024

Lecture 1 (26/09): intro to the course, unix shell, build process

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/01/01-intro_unix.md

Slide 32: Compiled VS interpreted languages
A compiled language (such as C) is harder to write but its execution is faster, while for a interpreted language (such as R, Python, MatLab) it's the other way round.
A compiled language has a compiler that compiles the source file producing an executable which runned gives an output.
An interpreted languages has an interpreter that interprets the source file producing the output.

Slide 33: Build process
The user gives a (or more) source file to the preprocessor, that gives a (or more) preprocess source file to the compiler. The compiler generates an object file which is given to the linker. The linker link all the object files generated from different source files with libraries; the loader loads shared libraries and execute the executable.

Slide 40:
Apri il terminale, quello che scrivi tu è preceduto da %

```
sara@Air-di-Sara-2 ~ % x=2
sara@Air-di-Sara-2 ~ % echo "Hello world"
Hello world
sara@Air-di-Sara-2 ~ % echo x
x
sara@Air-di-Sara-2 ~ % echo $x
2
```

quando metti $ davanti a x ti stampa il suo valore
Slide 47:
pwd: Print working directory

ls: fa la lista dei contenuti della directory in cui sei

cd: change directory. Se scrivi solo cd torni alla home, se no per scegliere la directory la scrivi subito dopo

cd .. : torni indietro

```
sara@Air-di-Sara-2 ~ % pwd
/Users/sara
sara@Air-di-Sara-2 ~ % ls
Applications
Creative Cloud Files
Creative Cloud Files testadobe908@gmail.c
b2b52c2764fd0cfb4076dd1d13b6
Creative Cloud Files wobic73874@jwsuns.co
f211a07ff82bee67df03a8b1f61
Desktop
Documenti
Documents
Downloads
Library
Mono_Preprocessing_WithoutFlat.ssf
Movies
Music
Pictures
Public
VirtualBox VMs
bin
sara@Air-di-Sara-2 ~ % cd Desktop
sara@Air-di-Sara-2 Desktop % cd ..
sara@Air-di-Sara-2 ~ %
```

Slide 48:
Sempre sul terminale, se scrivi *mkdir new_folder* crei una nuova cartella.
Se fai *mv new_folder raccoglitore/* sposti la nuova cartella nella directory chiamata raccoglitore.

```
sara@Air-di-Sara-2 ~ % mv Desktop/
Desktop/    Documenti/  Documents/  Downloads/
```

Se fai *mv D* e premi **tab**, ti mostra tutte le cose che iniziano con la D. È **estremamente utile quando le cose hanno nomi lunghi**.

Slide 50:
Open *vim* or *nano* from the terminal just typing them. Let's use vim.
C'è la **modalità comandi** in cui sei immessa appena entri (e ci torni premendo **esc**) e c'è la **modalità di inserimento** (ci entri scrivendo ***i***).
Dalla modalità comandi scrivi *:w* (che è il comando per scrivere un nuovo file) e il nome del file col luogo dove lo vuoi salvare, esempio *:w ~/Desktop/my_script.sh*.
Ora vai alla modalità di inserimento e scrivi:

```
#!/bin/zsh

a="hello world"
echo $a
```

The **first line of the script tells the shell which interpreter to use**, here #!/bin/bash where bash in MacOS is zsh so you'll write ***#!/bin/zsh***.
Infine premi esc e scrivi :wq per salvare e uscire.

1. Sara Serafino
   8 ottobre 2023 alle ore 13:55:54
   Dare i permessi di esecuzione

Now to execute a script in the terminal you type */my/path/name_of_the_script.ext*. When we're in the same folder you just type *./name_of_the_script.ext*. So here *./my_script.sh*.

1 Non abbiamo i **permessi**, affinché venga executed write ***chmod +x name_of_the_script.ext*** (where +x adds the execution permission).
Now rewriting *./my_script.sh* it executes the file.
Another way of executing the script is *source my_script.sh*; this way exports the local variables which means that now if we type *echo $a* we see "hello world", while with the previous command we did not due to locality, we just saw an empty line. This is because source copies and pastes the whole content.
Let's simulate the interaction of two sub-processing, so create another script my_script2.sh.

```
#!/bin/zsh

echo $a
```

In the first one let's add the execution of the second one.

```
#!/bin/zsh

a="hello world"
echo $a

./my_script2.sh
```

Now go to the terminal, make the second script executable as before (*chmod +x my.script2.sh*).
Executing *./my_script.sh* you'll see that it shows "hello world" followed by an empty line, which was what happened with local variables.
If I need the **variables accessible for all the sub-processes**, in the line of its definition I type ***export*** *a="hello world"*.

```
pcafrica@pcafrica-pop-os:~/Desktop$ ./my_script.sh
hello world
pcafrica@pcafrica-pop-os:~/Desktop$ echo $a

pcafrica@pcafrica-pop-os:~/Desktop$ source my_script.sh
hello world
pcafrica@pcafrica-pop-os:~/Desktop$ echo $a
hello world
pcafrica@pcafrica-pop-os:~/Desktop$ chmod +x my_script2.sh
pcafrica@pcafrica-pop-os:~/Desktop$ ./my_script.sh
hello world

pcafrica@pcafrica-pop-os:~/Desktop$ ./my_script.sh
hello world
hello world
```

Slide 52:
Into one of the scripts, let's define a **function** where arg1, argn are the arguments called with *$1 $n*.

```
my_fun() {
        echo $1 $2
        echo "This is my_fun!"
}

my_fun arg1 arg2 ... argn
```

You can define a **new command** (here myls) with ***alias*** *myls="ls -lA"*.

Slide 54:
head and tail print the first or last 10 lines, if you want a different number type:
head or ***tail*** and then *-n 3 nomefile.ext* per avere le prime 3.
grep *"echo" nomefile.ext* prints the lines that contain the word between " " inside the file.
For **more than one word** write *grep "echo\|a" nomefile.ext*.
For **word-something-word** write *grep "echo.*is" nomefile.ext*.
Since *cat* concatenates and reads a file and outputs its content, and *wc* reads a list of files and generates one or more of newline count, word count, byte count; if we want to **count the number of lines in a file *cat my_script.sh | wc***. If you want the lines add *-l* after that.
file is a command that tries to detect what is the content of the given input and classify its type.

Slide 55:
pipe operator (**|**) forwards the output of one command to another.

```
pcafrica@pcafrica-pop-os:~/Desktop$ file /bin/bash > bashtype.txt
pcafrica@pcafrica-pop-os:~/Desktop$ cat bashtype.txt
/bin/bash: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, in
terpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=33a5554034feb2af38e8c75872058883b2988
bc5, for GNU/Linux 3.2.0, stripped
pcafrica@pcafrica-pop-os:~/Desktop$ grep "echo" my_script2.sh >> bashtype.txt
pcafrica@pcafrica-pop-os:~/Desktop$ cat bashtype.txt
/bin/bash: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, in
terpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=33a5554034feb2af38e8c75872058883b2988
bc5, for GNU/Linux 3.2.0, stripped
echo $a
    echo $1 $2
    echo "This is my_fun!"
```

file /bin/bash **>** *bashtype.txt* creates a new file that we can print with *cat bashtype.txt*.
We can **append** new lines to a file with *grep "echo" my_script2.sh* **>>** *bashtype.txt*.

```
pcafrica@pcafrica-pop-os:~/Desktop$ cat foo.sh
cat: foo.sh: No such file or directory
pcafrica@pcafrica-pop-os:~/Desktop$ cat foo.sh > out
cat: foo.sh: No such file or directory
pcafrica@pcafrica-pop-os:~/Desktop$ cat foo.sh 2> err
```

Standard messages go in the standard outputs, **error messages** go in the error outputs. In order to redirect these outputs to a file:
cat foo.sh > out
It gives an error because > only redirects standard outputs.
If we want to print error messages to a file ***cat foo.sh 2> err***
This will become useful with C++ for isolating the two of them and printing them in separate files.
You can combine them with *cat foo.sh > full 2>&1*.

Laboratory 1 (28/09):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/01/01-intro_unix.md

Slide 2: Exercise 1: basic Bash commands.

1. Navigate to your home folder: *cd*
2. Create a folder named test 1: *mkdir test1*
3. Navigate to test1 and create a new directory test2: *cd test 1*
   mkdir test2
4. Navigate to test2 and go up one directory: *cd test2*
   mkdir new_folder
   cd new_folder
5. Create the following files f1.txt, f2.txt, f3.dat, f4.md, README.md, .hidden: *touch f1.txt f2.txt f3.dat f4.md README.md .hidden*
6. List all files including hidden ones: *ls -a*
7. List only .txt files: *ls *.txt*
8. Move README.md to folder test2: *mv README.md* ***../*** (**muove di una cartella indietro**)
9. Move all .txt files to test2 in one command: *mv *.txt ../*
10. Remove f3.dat: *rm f3.dat*
11. Remove all contents of test1 and the folder itself in one command: *rm -r test1*

Slide 3: Exercise 2: dataset exploration
After downloading the dataset structured from https://github.com/logpai/loghub/blob/master/HPC/HPC_2k.log_structured.csv , perform

the following analyses using only Bash commands (grep, echo, wc, cut, sort, uniq).

1. Find out how many unique node names are present in the dataset. *cut -f 3 -d ',' HPC_2k.log_structured.csv | sort -u | wc -l* (cut suddividi il file, f è il field di cui voglio la colonna 3, –d è il delimitatore che voglio ',', **sort** ordina in ordine alfabetico, **-u** prende solo quelli unici, wc word count -l conta le linee)
2. Export the list from the previous point to a file named nodes.log: *cut -f 3 -d ',' HPC_2k.log_structured.csv | sort -u > nodes.log*
3. Determine the number of times the "unavailable" event (E13) has been reported: *cut -f 5 -d ',' HPC_2k.log_structured.csv | grep "unavailable" | wc -l*
4. Identify the number of unique nodes that have reported either event E32 or event E33: *grep "E32\|E33" HPC_2k.log_structured.csv | cut -f 3 -d ',' | sort -u | wc -l*
5. Calculate how many times the node "gige7" has reported a critical event (E15): *grep "E32\|E33" HPC_2k.log_structured.csv | cut -f 9 -d ',' | grep "E15" | wc -l*
6. Find out how many times the "node-2" node has been reported in the logs *cut -f 3 -d ',' HPC_2k.log_structured.csv | grep -w "node-2" | wc -l* (**grep -w** prende esattamente quella parola e non altre)

Alternative way to do it (like the teacher did):

```
0.  filename=HPC_2k.log_structured.csv
    if [ ! -f "${filename}" ]
    then wget https://raw.githubusercontent.com/logpai/loghub/master/HPC/${filename}
    fi
1.  unique_nodes=$(tail -n +1 ${filename} | cut -d, -f3 | sort | uniq)
    echo "Number of unique nodes: $(echo "${unique_nodes}" | wc-l)"
2.  echo "${unique_nodes}" > nodes.log
3.  echo "Number of \"unavailable\" events reported: $(grep "E13" ${filename} | wc -l)"
4.  echo "\Number of unique nodes that have reported events E32 or E33: \$(grep "E32\|E33" ${filename} | cut -d, -f3 | sort | uniq | wc -l)"
```

*5. echo "\Number of times that node \"gige7\" has reported event E15: \ $(grep "gige7.*E15" ${filename} | wc -l)"*
6. echo "Number of reports for \"nodes-2\": $(grep "node-2," ${filename} | wc -l)"

La soluzione è:

```
2023-09-28 15:42:25 (4,43 MB/s) - 'HPC_2k.log_structured.csv' saved [217818/
217818]

Number of unique nodes: 299
Number of "unavailable" events reported: 12
Number of unique nodes that have reported events E32 or E33: 55
Number of times that node "gige7" has reported event E15: 15
Number of reports for "nodes-2": 2
```

Slide 4-5: Exercise 3: creating a backup script
Create a Bash script (using basename, tar, read, mkdir, cp, echo) that automates the process of creating a backup of a specified directory. The script should accomplish the following tasks:

1. Create a new Bash script file named backup.sh: Inside the script:
    1. Receive the directory to backup as an input argument
    2. Create a timestamped (***YYYYMMDD_hhmmss*** with ***date + %Y%m%d_%H%M%S***) backup folder (*backup_$timestamp*) inside a specified backup directory that you can define at the beginning of your script
    3. Copy all files and directories from the user-specified directory to the backup folder.
    4. Compress the backup folder into a single archive file (*backup_$timestamp.tar.gz*)
    5. Display a message indicating the successful completion of the backup process
2. Test your script by running it in your terminal. Ensure it performs all the specified tasks correctly.
3. Implement error handling in your script. For example, check if the specified input directory exists.

Svolgimento:

Su terminale scrivi *vim backup.sh*
Dopo aver creato il file backup.sh per runnarlo scrivi a terminale
chmod +x backup.sh
./backup.sh

Il file backup.sh è:

```
#!/bin/bash

#receive the directory to backup
read -p 'Which directory do you want to backup?' input_directory

#check if the directory exists
while [ ! -d "$input_directory" ]
do
echo "That is not a directory."
read -p ' Which one do you want to backup?' input_directory
done

output_directory="${HOME}/backup"
#if it does not exist create it
mkdir -p ${output_directory}

#extract the last part of the directory name with basename
#for example if we have /home/user/Documents, basename is Documents
directory_name=$(basename ${input_directory})

#Generate the timestamped backup directory name
backup_directory=${directory_name}_$(date +%Y%m%d_%H%M%S)

echo "Copying to backup folder..."
cp -r ${input_directory} ${output_directory}/${backup_directory}

echo "Creating backup archive..."
#save and change to output_directory
#dev/null is an empty file where you put things you don't want to see
pushd $output_directory > /dev/null

#create a tar archive where c compress, z is the gzip file format, f specifies
 the output file
#.tar.gz compress the backup_directory into a single archive file
tar czf ${backup_directory}.tar.gz $backup_directory

#print ending of backup
echo "Backup finished"

#go back to the original folder
popd > /dev/null
```

Slide 5-6: Exercise 4: collaborative file management on GitHub

1. Form groups of 2-3 members.
2. Designate one member to create a new **repository** and click the + button in the top right corner, ensure everyone **clones** it: ***git clone git@github.com:saraserafino/ex4AdvProg.git***
3. In a sequential manner, each group member should create a file with a distinct name and push it to the online repository while the remaining members pull the changes:
    1. *cd ex4AdvProg*
    2. *touch Sara.txt*
    3. *git add Sara.txt*
    4. *git* ***commit -m*** *"Sara created a file"* **salva e commenta o fai solo localmente!!**
    5. *git push*
    6. *git pull* quando gli altri fanno un cambiamento
4. Repeat step 3 but this time each participant should modify a different file than the ones modified by the other members of the group. *echo "Sto aggiungendo questa riga" >> file.ext; cat file.ext; git add file.ext; git commit -m "Sara modified file.ext"; git push*
5. Now let's work on the same file main.cpp (create it with *touch main.cpp; git add main.cpp; git commit -m "Sara created main.cpp"; git push*). Each person should create a hello world main.cpp that includes a personalized greeting with your name. To prevent conflicts do the following:
    1. **Create a unique branch** using the command ***git checkout -b new_branch*** (if you already created it you can return on it with the same command without -b)
    2. Develop your code and push your branch to the online repository: *echo "Hello world, I'm Sara" >> main.cpp; cat main.cpp; git add main.cpp; git commit -m "Sara updated main.cpp"; git push origin new_branch*
    3. Once everyone has finished their work, **merge your branch into the main branch** using the following commands: ***git checkout main; git pull origin main***; ***git merge new_branch; git push origin main***

4. If you're not the first, you'll have **conflicts to remove** (marked with <<<<<<< ======= >>>>>>>), review this sections deciding what to keep (open in vim, nano or similar), remove the markers and adjust, then go back to the terminal: ***git add main.cpp; git commit -m "File without conflicts"; git push origin main***

Lecture 2 (03/10): introduction to C++

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/02/02-c++_intro.md

```cpp
#include <iostream>

int main ()
{
    std::cout << "Hello, world!"
              << std::endl;
    return 0;
}
```

Slide 7:
Here on the right there's the hello_world.cpp file.
From terminal write the following commands to **compile and run** the program:
g++ hello_world.cpp -o my_program -o specifies the output file name
./my_program

```
→  Advanced Programming g++ hello_world.cpp -o my_program
→  Advanced Programming ./my_program
Hello, world!
```

If you want, before the int main you can write *using namespace std*, in this way you won't need std:: every time (so the above .cpp in the main would just be *cout << "Hello, world!" << endl;*)

Slide 10:

```cpp
#include <iostream>

int main ()
{
    // int a = 10; // direct inizialization
    // int a(10); // constructor-style inizialization
    int a{10}; // uniform inizialization. This is the best way to write it
    std::cout << "Hello, world!"
              << a << std::endl;
    return 204;
}
```

Change the file into this

If it says **"expected ';' at the end of declaration"** of int a{10}; it's because it has C++ 98 so writing on the terminal ***g++ -std=c++11 hello_world.cpp -o my_program*** you force to apply C++ 11 stuff and everything is ok. Note that you have to do it every time.
That final return 204 it's because if everything is ok you write return 0, otherwise any other number.

Slide 24-25:
Always remember to delete the pointer!!
Actually after the program ends the memory is deleted anyway, unless you're in a loop that continues to cause memory leaks until you don't have it anymore.

Slide 26:
Reference is a different name to the same variable, so you can use pointers without their dangers; though pointers are faster.
In the end ref=5, a=5, b=10 because ref was referring to a.

Slide 55:
In the **header file** you have **only** the **declaration**:

```
// file sum.hpp
int sum(int a, int b)
```

In the **source file** you **actually define it**:

```
// file sum.cpp
#include "sum.hpp"
int sum(int a, int b) {return a + b;}
```

The main file is:

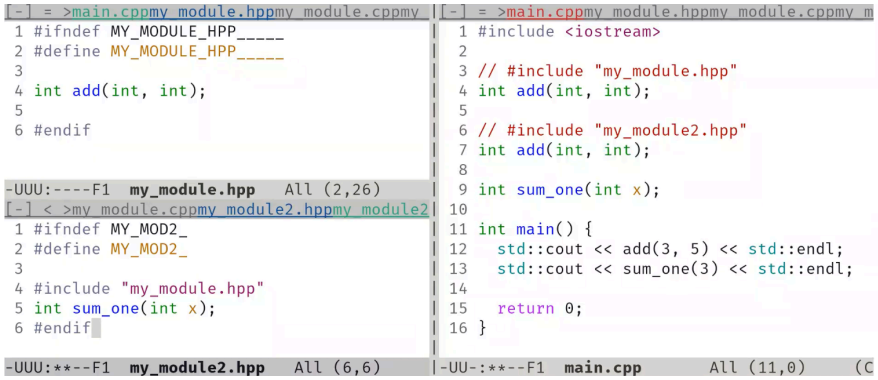
```
#include "sum.hpp"
int main () {
        int a = 2;
        int b = 3;
        c=sum(a, b);
```

return 0;
}

Per creare un'altra funzione che parta da una già costruita:
// file module.cpp
#include "sum.hpp" // la si include
int sum_and_print(int a, int b) {
int c = sum(a, b);
std::cout << c << std::endl;
return c;
}

The advantage of this subdivision - called **modular programming** - is that you have to change something, you have to only recompile that changed file, not everything.

Slide 59:
*#ifndef my_module_h_* means if it's not defined (usually you name it after the file); therefore we define it with *#define my_module_h_* .
Don't forget the *#endif* .



Slide 62:
namespace is how to give a label to things and use them.

Laboratory 2 (05/10):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/02/02-c++_intro.md

Slide 2: Exercise 1: control structures
Write a C++ program that converts a given temperature from Celsius to Fahrenheit or viceversa.
Then to compile and run write from terminal:
g++ -std=c++11 temperature_converter.cpp -o my_program
./my_program

The file temperature_converter.cpp is:

```
#include <iostream>
#include <string>

int main()
{
    double temperature;
    std::string unit;
    std::cout << "Enter a temperature: “;
    std::cin >> temperature;
    std::cout << "Specify if the unit of temperature is Celsius (C) or Fahrenheit (F): “;
    std::cin >> unit;
    // perform the conversion based on the provided unit
    double temperature_converted;
    if (unit == "C" || unit == "celsius" || unit == "Celsius") {
        temperature_converted = (9.0 / 5.0) * temperature + 32;
        std::cout << "The temperature " << temperature << "°C is " << temperature_converted << "°F" << std::endl;
    }
    else if (unit == "F" || unit == "fahrenheit" || unit == "Fahrenheit") {
        temperature_converted = (5.0 / 9.0 ) * (temperature - 32);
        std::cout << "The temperature " << temperature << "°F is " << temperature_converted << "°C" << std::endl;
    }
    else {
        std::cout << "You typed neither Celsius nor Fahrenheit." << std::endl;
        return 1; // we exit due to the error
    }
    return 0;
}
```

Slide 3: Exercise 2: memory management
Write a C++ program that dynamically allocates memory for an array of integers
The file is mm_ex.cpp:

```cpp
#include <cstdlib> // For rand() and srand().
#include <ctime>   // For time().
#include <iostream>

// Function to find and display the maximum and minimum values in an array.
void find_max_min(int *arr, unsigned int size, int &max_val, int &min_val) {
  if (size == 0) {
    std::cout << "The array is empty." << std::endl;
    return;
  }
  max_val = arr[0]; // Initialize max_val to the first element.
  min_val = arr[0]; // Initialize min_val to the first element.
  for (unsigned int i = 1; i < size; ++i) {
    if (arr[i] > max_val) {
      max_val = arr[i]; // Update max_val if a larger element is found.
    } else if (arr[i] < min_val) {
      min_val = arr[i]; // Update min_val if a smaller element is found.
    }
  }
}
int main() {
  unsigned int size; // unsigned guarantees that it is unsigned;
  // unfortunately if you type a negative number the program freezes
  // Prompt the user for the size of the array.
  std::cout << "Enter the size of the array: ";
  std::cin >> size;
  if (size == 0) {
    std::cout << "Invalid array size. Please enter a positive integer." << std::endl;
    return 1; // Exit with an error code.
  }
  // Dynamically allocate memory for the array.
  int *arr = new int[size];
  // Seed the random number generator with the current time.
  srand(time(nullptr));
  // Fill the array with random integers.
  for (unsigned int i = 0; i < size; ++i) {
    arr[i] = rand() % 100; // Generates random integers between 0 and 99.
  }
  // Find and display the maximum and minimum values in the array.
```

2. Sara Serafino
8 ottobre 2023 alle ore 14:55:14
Come implementare .hpp e .cpp relativo

```
  int max_val, min_val;
  find_max_min(arr, size, max_val, min_val);
  std::cout << "Array elements:" << std::endl;
  for (unsigned int i = 0; i < size; ++i) {
    std::cout << arr[i] << " ";
  }
  std::cout << std::endl << "Maximum value: " << max_val <<
std::endl;
  std::cout << "Minimum value: " << min_val << std::endl;
  // Deallocate the dynamically allocated memory.
  // If forgotten, a memory leak will occur.
  delete[] arr;
  return 0;
}
```

Slide 4: Exercise 3: complete the missing statistics calculator
You are provided with a partially implemented program (statistics.hpp) that calculates and displays statistics for a set of numbers, fill it in.
2 Implement statistics.cpp from the provided statistics.hpp:

```
#ifndef STATISTICS_HPP_
#define STATISTICS_HPP_
#include <vector>
namespace stat {
// Function prototypes.
double calculate_mean(const std::vector<double> &numbers);
double calculate_median(const std::vector<double> &numbers);
double calculate_standard_deviation(const std::vector<double>
&numbers);
} // namespace stat
#endif // STATISTICS_HPP_
```

Now statistics.cpp is:

```
#include "statistics.hpp"
#include <algorithm> // we need it for sort
#include <cmath> // we need it for sqrt

namespace stat {
// Function to calculate the mean of a set of numbers.
double calculate_mean(const std::vector<double> &numbers) {
  double sum = 0.0;
  for (const double &num : numbers) {
    sum += num; // aka sum=sum+num
  }
  return sum / numbers.size();
}
// Function to calculate the median of a set of numbers
double calculate_median(const std::vector<double> &numbers) {
  std::vector<double> sorted_numbers = numbers;
  // std::vector encapsulates dynamic size arrays
```

```
  std::sort(sorted_numbers.begin(), sorted_numbers.end());
  // std::sort sorts the elements in the range (first,last)
  size_t size = sorted_numbers.size();
  // size_t stores the maximum size of the object “size"
  if (size % 2 == 0) {
    // If the number of elements is even, average the middle two
numbers.
    size_t mid = size / 2;
    return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2.0;
  } else {
    // If the number of elements is odd, return the middle number.
    return sorted_numbers[size / 2];
  }
}
// Function to calculate the standard deviation of a set of
numbers.
double calculate_standard_deviation(const std::vector<double>
&numbers) {
  double mean = calculate_mean(numbers);
  double variance = 0.0;
  for (const double &num : numbers) {
    variance += (num - mean) * (num - mean);
  }
  return std::sqrt(variance / numbers.size());
}
} // closing namespace stat
```

main.cpp is:

```
#include “statistics.hpp"
#include <iostream>
#include <sstream> // we need it for istringstream

int main() {
  std::cout << "Statistics calculator" << std::endl;
  while (true) {
    std::cout << "Enter a set of numbers separated by spaces (or
'q' to quit): “;
    std::string input;
    std::getline(std::cin, input);
    // getline reads characters from an input stream (here
std::cin)
    // and places them into a string (here input)
    if (input == "q" || input == "Q") {
      break;
    }
    // Split the input string into a vector of numbers.
    std::vector<double> numbers;
    std::istringstream iss(input); // input string stream
    double num;
    while (iss >> num) {
      numbers.push_back(num);
```

3. Sara Serafino
8 ottobre 2023 alle ore 14:55:43
Come scrivere build.sh quando sta tutto nella stessa cartella

```
      // adds a new element (here num)
      // at the end of a vector (here numbers)
    }
    if (numbers.empty()) {
      std::cout << "Invalid input. Please enter numbers." <<
std::endl;
      continue;
    }
    // Calculate and display statistics
    const double mean = stat::calculate_mean(numbers);
    const double median = stat::calculate_median(numbers);
    const double stddev =
stat::calculate_standard_deviation(numbers);
    std::cout << "Mean: " << mean << std::endl;
    std::cout << "Median: " << median << std::endl;
    std::cout << "Standard Deviation: " << stddev << std::endl;
  }  // end of while
  return 0;
}
```

3 Lastly **using build.sh it's faster**:

```
#!/bin/bash
set -x
g++ -std=c++11 statistics.cpp main.cpp -o main
set +x
if [ $? -eq 0 ]; then
    echo "Build successful! You can run the program using ./main"
else
    echo "Build failed."
fi
```

From terminal we give the permissions with *chmod +x build.sh* then write *./build.sh* and lastly, as said from build.sh, we write *./main*.

Slide 5: Exercise 4: struct
Organize your code by separating the struct definition, data initialization and display function into different files or modules. Define a struct called student that represents information about a student including their name, age and grade average. This is student.hpp:

```
#ifndef STUDENT_HPP_
#define STUDENT_HPP_
#include <string>
struct Student {
  std::string name;
  unsigned int age;
  double grade_average;
};
```

```
#endif // STUDENT_HPP_
```

Create a std::vector of 5 student objects and initialize them with sample data. This is respectively data.hpp and data.cpp:

```
#ifndef DATA_HPP_
#define DATA_HPP_
#include “student.hpp"
#include <vector>
extern std::vector<Student> students_data;
#endif // DATA_HPP_
```

```
#include "data.hpp"
std::vector<Student> students_data = {{"Alice", 18, 95.5},
                                      {"Bob", 19, 88.0},
                                      {"Charlie", 20, 75.5},
                                      {"David", 21, 91.0},
                                      {"Eve", 22, 82.5}};
```

Write a function to display the information of all students in the array. This is respectively display.hpp and display.cpp:

```
#ifndef DISPLAY_HPP_
#define DISPLAY_HPP_
#include “student.hpp"
#include <vector>
void display_students(const std::vector<Student> &students);
#endif // DISPLAY_HPP_
```

```
#include “display.hpp"
#include <iostream>
void display_students(const std::vector<Student> &students) {
  for (const Student &student : students) {
    std::cout << "Name: " << student.name << std::endl;
    std::cout << "Age: " << student.age << std::endl;
    std::cout << "Grade Average: " << student.grade_average <<
std::endl;
    std::cout << "--------------" << std::endl;
  }
}
```

Finally the main.cpp is:

```
#include “data.hpp"
#include “display.hpp"
#include “student.hpp"
int main() {
  // Display the information of all students.
  display_students(students_data);
  return 0;
}
```

And build.sh is the same as the exercise before, with obviously all the files data.cpp display.cpp main.cpp at the 3rd line instead of the previous ones.

Slide 6: Exercise 5: code organization
Write a C++ program that simulates a simple calculator. Define functions for addition, subtraction, multiplication and division. Allow the user to enter two numbers and choose an operation to perform and display.
calculator.hpp is:

```
#ifndef CALCULATOR_HPP
#define CALCULATOR_HPP
namespace Calculator {
double add(const double &a, const double &b);
double subtract(const double &a, const double &b);
double multiply(const double &a, const double &b);
double divide(const double &a, const double &b);
}; // namespace Calculator
#endif // CALCULATOR_HPP
```

calculator.cpp is:

```
#include “calculator.hpp"
#include <iostream>
namespace Calculator {
double add(const double &a, const double &b) { return a + b; }
double subtract(const double &a, const double &b) { return a -
b; }
double multiply(const double &a, const double &b) { return a *
b; }
double divide(const double &a, const double &b) {
  if (b == 0) {
    std::cerr << "Division by zero is not allowed." << std::endl;
    // cerr stands for character error stream
    std::exit(1); // terminates immediately the whole program
execution
    // while return would have returned only from this current
function
  }
  return a / b;
}
} // namespace Calculator
```

main.cpp is:

```
#include “calculator.hpp"
#include <iostream>
int main() {
```

4. Sara Serafino
8 ottobre 2023 alle ore 14:56:31
Ora build.sh è un po' diverso perché deve prendere i file da diverse cartelle

```
  char operation;
  double num1, num2, result;
  std::cout << "Enter two numbers: “;
  std::cin >> num1 >> num2;
  std::cout << "Choose an operation (+, -, *, /): ";
  std::cin >> operation;
  switch (operation) {
  case ‘+':
    result = Calculator::add(num1, num2);
    break;
  case ‘-':
    result = Calculator::subtract(num1, num2);
    break;
  case ‘*':
    result = Calculator::multiply(num1, num2);
    break;
  case ‘/':
    result = Calculator::divide(num1, num2);
    break;
  default:
    std::cerr << "Error: Invalid operation." << std::endl;
    return 1;
  }
  std::cout << "Result: " << result << std::endl;
  return 0;
}
```

It’s required this breakdown of the project structure:

- `calculator/`
  - `src/`
    - `main.cpp`
    - `calculator.cpp`
  - `include/`
    - `calculator.hpp`
  - `build/` (build artifacts, such as object files and executables).
  - `build.sh` (a Bash script that compiles and properly links the code)

4

so we have a different build.sh to before because **build.sh has to take files from directories**:

```
#!/bin/bash
set -x
mkdir -p build
g++ -Iinclude/ src/calculator.cpp -c -o build/calculator.o
g++ -Iinclude/ src/main.cpp -c -o build/main.o
g++ -std=c++11 build/calculator.o build/main.o -o build/calculator
set +x
if [ $? -eq 0 ]; then
    echo "Build successful! You can run the program using ./build/
calculator”
else
    echo "Build failed."
```

```
fi
```

Lecture 3 (10/10): object-oriented programming, classes and access control in C++

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/03/03-c++_classes.md

Slide 9-10:

```
#include <iostream>
#include <string>
class Car {
public:
    std::string model;
    unsigned int year;
// The thing about objects is that you can
// implement functions that act on an object.
    void print_info() {
        std::cout << "Car model: " << model
        << ", year" << year;
    }
};
int main () {
    Car c; //creating an object c of the class Car
    c.year = 2020; //object_name.member_or_variable_name
    c.model = "BMW series 1";
    c.print_info(); //invoke a method
    return 0;
}
```

Slide 12:

Defining void f() {*static int x=0; x++}*

When outside we call *f(); f(); f();* every time it increments it, so you get x=3; while if there wasn't static it would have been 1 every time.

Slide 15:

Changing the void function:

```
void print_info(std::string model) {
        std::cout << "Car model: "
        << this->model // this points to the first model
        << ", year" << year;
    }
```

Slide 19:

The initializer list is the preferred way to go.

Slide 20:
With the copy constructor we tell to the compiler and to our program what we want to do when we copy it.
We allocate the memory with its value *int x=5* (copy assignment), while with *int x; x=5* we initialize x after defining (copy constructor).

Slide 25:
~FileHandler is the destructor. Inside of it we tell it that if the file is still open, it closes it. In a modular programming, the destructor is only in the header (.hpp) file.
std::ofstream file represents the file to be opened.

Slide 39:
<=> isn't if and only if but it's the spaceship operator which returns:
-1 if the first number < second number
0 if they're =
1 if the first number > second number

Slide 44-48-49:
If outside *class BankAccount* we want to access balance, which was declared private with a keyword, we can't. For example if you write *std::cout << balance;* it gives an error; unless you write *BankAccount account; account.get_balance()*.

Slide 47:
In your code use only one between class and struct, at the beginning declare explicitly if public or private.

Laboratory 3 (12/10):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/03/03-c++_classes.md

The 5 exercises of this lab operate all on the same thing. They ask the following:

1. Create a class named *DataProcessor* with private data members for a data array and its size. The data array should be represented as a *double *data*.
2. Implement a constructor that takes an array of floating-point numbers and its size as input and initializes the class data members.
3. Implement a copy constructor, a copy assignment operator and the destructor.
4. Add a method *n_elements()* that returns the number of elements in the array.
5. Test all these functionalities in the *main* function by creating proper instances of *DataProcessor* and displaying the results.
6. Add methods to compute minimum and maximum values, the mean and the standard deviation of the data.
7. Add tests to validate these new functionalities.
8. Organize the *DataProcessor* class by separating declarations and definition into separate header (*data_processor.hpp*) and source (*data_processor.cpp*) files.
9. Create a main program file that includes the header and demonstrates the use of the *DataProcessor* class for data analysis.
10. Compile the program using *g++ std=c++11 –O3 -Wall -Wpedantic data_processor.cpp main.cpp -o data_processor* where -O3 means that we're performing the highest level of optimization (from 0 to 3, 3 has faster execution times); -Wall enables all the warnings about constructions; -Wpedantic issues all the warnings and rejects all programs that use forbidden extensions.
11. Overload the output stream operator << as a *friend* function to allow printing the list of values in the stored data, separated by a comma.
12. Overload the *[]* operator to allow indexing and accessing individual data elements; this operator will be used for both read and write access.
13. Overload the + operator in the *DataProcessor* class to allow adding two *DataProcessor* objects. The result should be a new *DataProcessor* object containing the element-wise sum of the data

arrays. The operator should also print an error if the two operands do not have the same size.

14. Add tests to validate these new functionalities.
15. Ensure the const-correctness of all member variables and methods by adding proper *const* qualifiers.
16. Add a *static* member function *get_n_instances()* that returns how many instances of *DataProcessor* objects are currently active.
17. Implement a free function *double compute_correlation(const DataProcessor &dp1, const DataProcessor &dp2);* that computes the Pearson correlation coefficient between two datasets with the same size.
18. Add tests to validate these new functionalities.

The data_processor.hpp is:

```
#ifndef DATA_PROCESSOR_HPP__
#define DATA_PROCESSOR_HPP__
#include <algorithm> // to use min_element and max_element
#include <cassert> // to use assert
#include <numeric> // to use accumulate
#include <ostream> // to use ostream
class DataProcessor {
public:
  // Constructor
  DataProcessor(const double *input_data, const unsigned int
&input_size);
  // Copy constructor
  DataProcessor(const DataProcessor &other);
  // Copy assignment operator
  DataProcessor &operator=(const DataProcessor &other);
  // Destructor. It is fully written in the .hpp and
  // not mentioned in the .cpp
  ~DataProcessor() {
    --n_instances;
    delete[] data;
    data = nullptr;
  }
  // Sum operator
  DataProcessor operator+(const DataProcessor &other) const;
  // Write access operator
  double &operator[](const unsigned int &index) {
    assert(index >= 0 && index < size);
    return data[index];
  }
  // Read access operator.
  const double &operator[](const unsigned int &index) const {
    assert(index >= 0 && index < size);
    return data[index];
```

```
  }
  unsigned int n_elements() const { return size; }
  double min() const { return *std::min_element(data, data +
size); };
  // returns the smallest element in range [data, data + size)
  double max() const { return *std::max_element(data, data +
size); };
  // returns the largest element in range [data, data + size)
  double compute_mean() const {
    const double sum = std::accumulate(data, data + size, 0.0);
    // accumulate values in range [data, data + size) giving
    // as initial value for the accumulator 0.0
    return sum / size;
  }
  double compute_std_dev() const;
  static unsigned int get_n_instances() { return n_instances; }
//a friend function can access private and protected data of a
class
  friend std::ostream &operator<<(std::ostream &os, const
DataProcessor &dp);
//std::ostream is needed because you may don't want to overload
//the operator << for all individual stream types, but only for
//the common base class of them which has the << functionality
private:
  static unsigned int n_instances;
  unsigned int size;
  double *data;
};
double compute_correlation(const DataProcessor &dp1, const
DataProcessor &dp2);
#endif /* DATA_PROCESSOR_HPP__ */
```

The data_processor.cpp is:

```
#include "data_processor.hpp"
#include <cmath>
unsigned int DataProcessor::n_instances = 0;
DataProcessor::DataProcessor(const double *input_data,
                             const unsigned int &input_size)
  : size(input_size), data(new double[size]) {
  // writing in this way we copy not just the pointer but all
  // the elements inside
  ++n_instances;
  // Copy from input_data to data
  for (unsigned int i = 0; i < size; ++i) {
    data[i] = input_data[i];
      }
}
// Copy constructor
DataProcessor::DataProcessor(const DataProcessor &other)
// const so you don't modify it
    : size(other.size), data(new double[size]) {
```

```
  ++n_instances;
  for (unsigned int i = 0; i < size; ++i) {
    data[i] = other[i];
  }
}
// Copy assignment operator
DataProcessor &DataProcessor::operator=(const DataProcessor
&other) {
  if (this != &other) {
    delete[] data;
    size = other.size;
    data = new double[size];
    for (unsigned int i = 0; i < size; ++i) {
      data[i] = other[i];
    }
  }
  return *this; // this argument return is conventional
  // so it returns the copy of the object
}
DataProcessor DataProcessor::operator+(const DataProcessor &other)
const {
  assert(size == other.size); // Check if sizes match
  DataProcessor result(data, size); // Copy the current object
  for (unsigned int i = 0; i < size; ++i) {
    result[i] += other[i];
  }
  return result;
}
// Compute the standard deviation
double DataProcessor::compute_std_dev() const {
  const double mean = compute_mean();
  double sum_squared_diff = 0.0;
  for (unsigned int i = 0; i < size; ++i) {
    sum_squared_diff += (data[i] - mean) * (data[i] - mean);
  }
  return std::sqrt(sum_squared_diff / size);
}
std::ostream &operator<<(std::ostream &os, const DataProcessor
&dp) {
  for (unsigned int i = 0; i < dp.size; ++i) {
    os << dp[i];
    if (i < dp.size - 1) {
      os << ", ";
    }
  }
  return os;
}
double compute_correlation(const DataProcessor &dp1, const
DataProcessor &dp2) {
  assert(dp1.n_elements() == dp2.n_elements()); // Check if sizes
match
  const unsigned int n = dp1.n_elements();
```

```
  // Calculate the means of both datasets
  const double mean1 = dp1.compute_mean();
  const double mean2 = dp2.compute_mean();
  // Calculate the sums of the cross-products of differences
  double sum_cross_products = 0.0;
  double sum_diff1_squared = 0.0;
  double sum_diff2_squared = 0.0;
  for (unsigned int i = 0; i < n; ++i) {
    const double diff1 = dp1[i] - mean1;
    const double diff2 = dp2[i] - mean2;
    sum_cross_products += diff1 * diff2;
    sum_diff1_squared += diff1 * diff1;
    sum_diff2_squared += diff2 * diff2;
  }
  // Calculate the Pearson correlation coefficient.
  const double correlation =
      sum_cross_products /
     (std::sqrt(sum_diff1_squared) +std::sqrt(sum_diff2_squared));
  return correlation;
}
```

The main.cpp is:

```
#include "data_processor.hpp"
#include <iostream>
int main() {
  const double input1[] = {2.43, -0.86, 7.19, 4.57, 1.68, 9.32,
5.75};
  const double input2[] = {0.73, -0.45, 0.12, 0.88, -0.67, 0.34,
-0.92};
  const DataProcessor dp1(input1, 7); // Testing constructor
  const DataProcessor dp2 = dp1; // Testing copy constructor
  DataProcessor dp3(input2, 7);
  dp3 = dp1; // Testing copy assignment operator
  std::cout << "dp1: " << dp1 << std::endl;
  std::cout << "dp2: " << dp2 << std::endl;
  std::cout << "dp3: " << dp3 << std::endl;
  std::cout << "Number of elements in dp1: " << dp1.n_elements()
<< std::endl;
  std::cout << "Number of elements in dp2: " << dp2.n_elements()
<< std::endl;
  std::cout << "Number of elements in dp3: " << dp3.n_elements()
<< std::endl;
  std::cout << "Minimum value of dp1: " << dp1.min() << std::endl;
  std::cout << "Maximum value of dp1: " << dp1.max() << std::endl;
  std::cout << "Mean of dp1: " << dp1.compute_mean() << std::endl;
  std::cout << "Standard deviation of dp1: " <<
dp1.compute_std_dev() << std::endl;
  const DataProcessor dp4(input2, 7);
  std::cout << "dp4: " << dp4 << std::endl;
  {
    DataProcessor dp5 = dp1 + dp4;
```

```
    std::cout << "dp5: " << dp5 << std::endl;
    std::cout << "Minimum value of dp5: " << dp5.min() <<
std::endl;
    std::cout << "Maximum value of dp5: " << dp5.max() <<
std::endl;
    std::cout << "dp5[3]: " << dp5[3] << std::endl;
    dp5[3] = 0.0;
    std::cout << "dp5[3]: " << dp5[3] << std::endl;
  }
  std::cout << "Number of DataProcessor instances: "
            << DataProcessor::get_n_instances() << std::endl;
  const double correlation = compute_correlation(dp1, dp4);
  std::cout << "Correlation between dp1 and dp4: " << correlation
<< std::endl;
  return 0;
}
```

Making it run with *./data_processor* it gives:

```
dp1: 2.43, -0.86, 7.19, 4.57, 1.68, 9.32, 5.75
dp2: 2.43, -0.86, 7.19, 4.57, 1.68, 9.32, 5.75
dp3: 2.43, -0.86, 7.19, 4.57, 1.68, 9.32, 5.75
Number of elements in dp1: 7
Number of elements in dp2: 7
Number of elements in dp3: 7
Minimum value of dp1: -0.86
Maximum value of dp1: 9.32
Mean of dp1: 4.29714
Standard deviation of dp1: 3.22301
dp4: 0.73, -0.45, 0.12, 0.88, -0.67, 0.34, -0.92
dp5: 3.16, -1.31, 7.31, 5.45, 1.01, 9.66, 4.83
Minimum value of dp5: -1.31
Maximum value of dp5: 9.66
dp5[3]: 5.45
dp5[3]: 0
Number of DataProcessor instances: 4
Correlation between dp1 and dp4: 0.251191
```

Lecture 4 (17/10): inheritance and polymorphism

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/04/04-c++_inheritance_polymorphism.md

Slide 11:
Composition is very similar to aggregation but it's stronger since the part can't exist independently.

Slide 23:

First we destroy derived class members, then base class members because derived classes are dependent on the base class.

```
#include <iostream>
class Base {
public:
    Base() { std::cout << "Constructing base class" <<  std::endl;
}
    ~Base() { std::cout << "Destroying base class" << std::endl; }
};
class Derived: public Base {
public:
    Derived() { std::cout << "Constructing derived class" <<
std::endl; }
    ~Derived() { std::cout << "Destroying derived class" <<
std::endl; }
};
int main () {
    Derived d;
    return 0;
}
```

Slide 26:
The main goal of implementing inheritance is that there's polymorphism which allows a class of derived type to act like a class of base type.

Slide 33:
In the main:

```
Polygon *p = new Rectangle(); // il * sta per pointer
    const double a = p->area();
    // whenever there's a new you have to delete. delete p isn't
enough, due to the structure of the polygon it doesn't destruct
stuff in rectangle
```

In the base class:

```
    virtual ~Polygon() = default; // the destructor is implemented
by default.
```

Remember to mark the destructor in the base class a virtual when using polymorphisms!!

Laboratory 4 (19/10):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/04/04-c++_inheritance_polymorphism.md

Slide 2-3: Exercise 1: a framework for automatic differentiation

Implement a C++ framework for computing derivatives of arbitrary functions using a polymorphic approach. The goal is to create a structure that can handle common arithmetic operations and their derivatives.
Define an abstract base class *ADExpression* with two pure virtual functions: *double evaluate()* that returns the value of the variable and *double evaluate_derivative()* that returns its derivative.
Implement a concrete class *Scalar* that inherits from *ADExpression* and represents a scalar variable and its derivative.
Implement the following operation classes that also inherit from *ADExpression*:

- *sum, difference, product, division*: represent respectively the addition, subtraction, multiplication, division of two *ADExpression* objects.
- *power*: represents raising an *ADExpression* object to a constant exponent.

In the main function demonstrate the usage of these classes. Here it is:

```
#include <iostream>
#include <cmath> // to use std::pow
class ADExpression {
public:
    virtual ~ADExpression() = default;
    virtual double evaluate() const = 0;
    // while if you implement it in this other way:
    // virtual double evaluate(const double &x) = 0;
    // by doing f.evaluate(3.0) in the main, you
    // evaluate its value in 3
    virtual double derivative() const = 0; // pure methods
};
class Scalar : public ADExpression {
public:
    Scalar(double val, double der = 0.0)
    : val(val), der(der) {} // Constructor
    double evaluate() const override { return val; }
    double derivative() const override { return der; }
private:
    double val;
    double der;
};
class Sum : public ADExpression {
public:
    Sum(const ADExpression &t1, const ADExpression &t2)
    : term1(t1), term2(t2) {} // Constructor
    double evaluate() const override {
```

```
        return term1.evaluate() + term2.evaluate();
    }
    double derivative() const override {
        return term1.derivative() + term2.derivative();
    }
private:
    const ADExpression &term1;
    const ADExpression &term2;
};
class Difference : public ADExpression {
public:
    Difference(const ADExpression &t1, const ADExpression &t2)
    : term1(t1), term2(t2) {} // Constructor
    double evaluate() const override {
        return term1.evaluate() - term2.evaluate();
    }
    double derivative() const override {
        return term1.derivative() - term2.derivative();
    }
private:
    const ADExpression &term1;
    const ADExpression &term2;
};
class Product : public ADExpression {
public:
    Product(const ADExpression &f1, const ADExpression &f2)
    : factor1(f1), factor2(f2) {} // Constructor
    double evaluate() const override {
        return factor1.evaluate() * factor2.evaluate();
    }
    double derivative() const override {
        return  (factor1.derivative() * factor2.evaluate()) +
                (factor1.evaluate() * factor2.derivative());
    }
private:
    const ADExpression &factor1;
    const ADExpression &factor2;
};
class Division : public ADExpression {
public:
    Division(const ADExpression &num, const ADExpression &den)
    : numerator(num), denominator(den) {} // Constructor
    double evaluate() const override {
        return numerator.evaluate() / denominator.evaluate();
    }
    double derivative() const override {
        return  (numerator.derivative() * denominator.evaluate()
                - numerator.evaluate() * denominator.derivative())
                / (denominator.evaluate() *
denominator.evaluate());
    }
private:
```

```
    const ADExpression &numerator;
    const ADExpression &denominator;
};
class Power : public ADExpression {
public:
    Power(const ADExpression &b, int exp)
    : base(b), exponent(exp) {} // Constructor
    double evaluate() const override {
        return std::pow(base.evaluate(), exponent);
    }
    double derivative() const override {
    return  exponent * base.derivative() *
            std::pow(base.evaluate(), exponent - 1);
    }
private:
    const ADExpression &base;
    int exponent;
};
// if you wanted, you could implement this operator
/*
operator+(const ADExpression &a, const ADExpression &b) {
    double val = a.evaluate() + b.evaluate();
    double der = a.derivative() + b.derivative();
};
*/
int main() {
    // set x = 2.0 and its derivative = 1.0
    Scalar x(2.0, 1.0);
    // Define the following polynomials
    // f(x) = 2 x^3 - 3 x^2 + 4 x - 5
    const auto two = Scalar(2);
    const auto three = Scalar(3);
    const auto four = Scalar(4);
    const auto five = Scalar(5);

    const auto x2 = Power(x,2);
    const auto x3 = Power(x,3);

    const auto _4x = Product(four, x);
    const auto _3x2 = Product(three,x2);
    const auto _2x3 = Product(two, x3);

    const auto f1 = Difference(_2x3,_3x2);
    const auto f2 = Sum(f1,_4x);
    const auto f = Difference(f2,five);

    // Compute and print its value and derivative at set x
    std::cout << "f(" << x.evaluate() << ") = " << f.evaluate() <<
std::endl;
    std::cout << "f'(" << x.evaluate() << ") = " << f.derivative()
<< std::endl;
```

```
    // g(x) = 1 / x^2
    const auto one = Scalar(1.0);
    auto g = Division(one, x2);

    // Compute and print its value and derivative at set x
    std::cout << "g(" << x.evaluate() << ") = " << g.evaluate() <<
std::endl;
    std::cout << "g'(" << x.evaluate() << ") = " << g.derivative()
<< std::endl;

    return 0;
}
```

Slide 4-9: Exercise 2: inheritance and polymorphism for data analysis
Create a C++ program that models different types of data sources and transformation objects.
Define an abstract class *DataSource* with attributes and methods that represent common properties of data sources. Create derived classes such as *FileDataSource* and *ConsoleDataSource*.
Define an abstract class *DataTransformer* with a virtual method for data transformation. Create derived classes such as *LinearScaler*, *LogTransformer* and *StandardScaler* that implement specific data transformation methods.
Test all functionalities by prompting the user with proper questions.
Define an abstract class *DataSource* with a string attributo *name*, a vector *data*, a method *display_info()* and a pure virtual method *real_data()*. Implement a constructor and a virtual destructor in the *DataSource* class. Create derived classes *FileDataSource* and *VectorDataSource* that inherit from *DataSource*. Implement constructors for all classes to model different data source types. For example, *FileDataSource* should initialize a *filename* and an input file, *ConsoleDataSource* should have a default constructor. Implement destructors for the derived classes. For example the *FileDataSource* constructor should open the file and its destructor should close it. Implement the *read_data()* methods. *FileDataSource::read_data()* should import values from a file (see *data.txt* as an example), whereas *ConsoleDataSource::read_data()* should read a list of values from the standard input. Create objects of these classes and demonstrate inheritance. For example, create a *FileDataSource* object and call *display_info()* and *read_data()* methods to read and display data from

a file. Ensure that resources associated with data sources are properly managed during construction and destruction.
Define a base class *DataTransformer*, bound to *DataSource* by polymorphic composition. *DataTransformer* should have a pure virtual method *transform()* that transforms the *data* vector in the corresponding *DataSource*. Create derived classes *LinearScaler*, *LogTransformer* and *StandardScaler* that inherit from *DataTransformer*. Override the *transform()* method in each derived class to provide specific data transformation. For example, *LinearScaler* scales the data by multiplying them by a given scaling factor, *LogTransformer* applies a logarithmic transformation and sets negative entries to 0, and *StandardScaler* performs standardization to the [0,1] interval. Create objects of these classes and demonstrate their use to transform the previously defined *DataSource* object.
Use the *DataSource* and the *DataTransformer* hierarchy polymorphically. In particular, the program should prompt the user to import data either from a file or from console, and to select the transformation method. Display the original and the transformed data. You can implement: in case a *FileDataSource* is selected, prompt the user to specify the filename from the console. In case a *LinearScaler* is selected, prompt the user to specify the scaling factor from the console. Add a new class to the *DataSource* hierarchy to import a given field from an input CSV file (see *data.csv* as example). Write a class or method to overwrite the original data from the text or CSV file with transformed ones.

Find solution in folder lab04/ex2

Lecture 5 (24/10): functions. Templates and generic programming

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/05/05-c++_functions_templates.md

Slide 12:

5. Sara Serafino
8 novembre 2023 alle ore 12:49:48
template in headers

Whenever you're dealing with functions remember that you're dealing with pointers.

Slide 14:
Member function pointers ensure us that we're referring to a member of a specific class.

Slide 22:

```
int main() {
    // regular variables
    auto f = [] (double x) { return x * x * x; };
    int i = 3; // if we want to use i inside f but don't
    // want to change its value, we capture it by copy with [i]
    auto f = [i] (double x) { return x * i; };
    // actually the compiler has converted this function to a
functor
     auto f = [&i] (double x) {
        i = 2; //doesn't change the i outside
        return x * i;
     };
    return 0;
}
```

Slide 23:

```
#include <algorithm>
int main() {
    std::for_each (v.begin(), v.end(), f) {
        for(a = v.begin(); a = v.end() - 1)
            f(a);
    }
    return 0;}
```

Slide 33:
In polymorphism code can adapt, while in generic programming everything is done at compiling time so it takes more time (but it's slightly more efficient).

Slide 54:
5 Separate declarations *module.hpp* and definitions ***module.tpl.hpp*** when **templates** are long and complex, then add *#include "module.tpl.hpp"* at the end of *module.hpp*.

Laboratory 5 (26/10):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/05/05-c++_functions_templates.md

Slide 2: Exercise 1: lambda functions

Starting from exercises/05/hints/ex1.cpp, create a C++ program that calculates the total cost of a given list of products. Use *std::accumulate* to calculate the total cost of the products, passing a custom lambda function. Display the results after each step to check for correctness.

```cpp
#include <iostream>
#include <numeric> // for using accumulate
#include <string>
#include <vector>
class Product {
public:
  std::string name;
  double price;
};
void print(const std::vector<Product> &products) {
  for (const Product &product : products) {
    std::cout << product.name << ": \t$" << product.price <<
std::endl;
  }
}
int main() {
  // Define a list of products.
  const std::vector<Product> products = {{"Smartphone", 799.99},
                                         {"Laptop", 1299.99},
                                         {"Tablet", 349.99},
                                         {"Headphones", 99.99},
                                         {"Smartwatch", 249.99}};

  std::cout << "Original list of products:" << std::endl;
  print(products);
  // Compute total cost.
  const double total_cost =
      std::accumulate(products.begin(), products.end(), 0.0,
                      [](const double &sum, const Product
&product) {
                        return sum + product.price;
                      });
  std::cout << std::endl << "Total cost: $" << total_cost <<
std::endl;
  return 0;
}
```

Slide 3: Exercise 2: function pointers, functors and lambda functions

Starting from exercises/05/hints/ex2.cpp, develop a library management system with search capabilities using lambdas and functors. Using *std::sort*, sort the books:

- in ascending order based on year, using a function pointer as a comparator;
- in descending order based on year, using a lambda function as a comparator;
- in ascending order based on the author name, using a functor pointer as a comparator.

Using *std::copy_if* fill the *filtered_books* variable by extracting from *books* only the books written by a specific author. Implement the search functional using lambdas.

Display the results after each step to check for correctness.

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
class Book {
public:
  std::string title;
  std::string author;
  std::string genre;
  unsigned int publication_year;
};

bool compare_by_year_ascending(const Book &a, const Book &b) {
  return a.publication_year < b.publication_year;
}

class ComparatorByName {
public:
  bool operator()(const Book &a, const Book &b) const {
    return a.author < b.author;
  }
};

void print(const std::vector<Book> &books) {
  for (const Book &book : books) {
    std::cout << "  Title: " << book.title << " | Author: " <<
book.author
              << " | Genre: " << book.genre
              << " | Year: " << book.publication_year <<
std::endl;
  }
}
```

```cpp
int main() {
  // Define a list of books.
  std::vector<Book> books = {
      {"Go set a watchman", "Harper Lee", "Fiction", 2015},
      {"Animal farm", "George Orwell", "Satire", 1945},
      {"To kill a mockingbird", "Harper Lee", "Fiction", 1960},
      {"1984", "George Orwell", "Science fiction", 1949},
      {"Pride and prejudice", "Jane Austen", "Romance", 1813},
      {"Sense and sensibility", "Jane Austen", "Romance", 1811}};

  std::cout << "Original list of books:" << std::endl;
  print(books);

  // Sort in ascending order by year.
  std::sort(books.begin(), books.end(),
compare_by_year_ascending);
  std::cout << std::endl
            << "Books sorted in ascending order by year:" <<
std::endl;
  print(books);

  // Sort in descending order by year.
  std::sort(books.begin(), books.end(), [](const Book &a, const
Book &b) {
    return a.publication_year > b.publication_year;
  });
  std::cout << std::endl
            << "Books sorted in descending order by year:" <<
std::endl;
  print(books);

  // Sort in ascending order by author name.
  std::sort(books.begin(), books.end(), ComparatorByName{});
  std::cout << std::endl
            << "Books sorted in ascending order by author name:"
<< std::endl;
  print(books);

  // Filter books by author.
  const std::string author_to_filter = "Jane Austen";
  std::vector<Book> filtered_books;
  std::copy_if(books.begin(), books.end(),
std::back_inserter(filtered_books),
               [author_to_filter](const Book &book) {
                 return book.author == author_to_filter;
               });

  if (filtered_books.empty()) {
    std::cout << std::endl
              << "No books by " << author_to_filter << " found."
<< std::endl;
  } else {
```

```
    std::cout << std::endl
              << "Books by " << author_to_filter << ":" <<
std::endl;
    print(filtered_books);
  }
  return 0;
}
```

Slide 4-5: Exercise 3: function wrappers, templates
The exercises/05/hints/ex3/ folder provides a partial C++ implementation of the Newton method to approximate the root(s) of a function f, i.e. to solve the problem f(x)=0. Newton's method, starting with an initial guess for the root(s) of the function, denoted as x^(0), repeatedly refine the estimate using the formula x^(k+1) = x^k - f(x^k)/f'(x^k). The iterations continue until the difference between two consecutive estimates |x^(k+1) - x^k | is smaller than a predefined tolerance. If the condition is not met within a maximum number of iterations, the algorithm failed to reach convergence and the solver returns *NaN*.
Fill in the missing parts to make the program work with real valued scalar functions.
Use the program to solve f(x)=x^2 - 1 = 0 starting from x^0 = 0.5
Templatize the solver to be able to deal with more general functions, such as complex valued functions. Use the program to solve f(x) = x^2 + 1 = 0, starting from x^0 = 0.5 + 0.5i.
How would organize the project files? Is it better to keep everything in header files, or splitting declarations and definitions in header and source files by providing explicit instantiations? Try both alternatives.

The header file newton.hpp is:

```
#ifndef NEWTON_HPP__
#define NEWTON_HPP__
#include <functional> // for using function
// T is whichever type of function
template <typename T> class NewtonSolver {
public:
  NewtonSolver(const std::function<T(const T &)> &f,
               const std::function<T(const T &)> &df, const T &x0,
               const double &tolerance = 1e-12,
               const unsigned int &max_iterations = 100);

  T solve();
```

```
private:
  // f is the function we're searching the root of
  const std::function<T(const T &)> f;
  // its derivative
  const std::function<T(const T &)> df;
  const T x0; // initial value
  const double tolerance;
  const unsigned int max_iterations;
};
#endif
```

newton.cpp is:

```
#include "newton.hpp"
#include <cmath>
#include <complex>
template <typename T>
NewtonSolver<T>::NewtonSolver(const std::function<T(const T &)>
&f,
                              const std::function<T(const T &)>
&df,
                              const T &x0, const double
&tolerance,
                              const unsigned int &max_iterations)
    : f(f), df(df), x0(x0), tolerance(tolerance),
      max_iterations(max_iterations) {}

template <typename T> T NewtonSolver<T>::solve() {
  T x = x0;

  unsigned int it = 0;
  while (it < max_iterations) {
    const T delta = f(x) / df(x);
    x -= delta;
    if (std::abs(delta) < tolerance) {
      return x;
    }
    ++it;
  }
  // Indicates failure to converge.
  return std::numeric_limits<T>::quiet_NaN();
}
template class NewtonSolver<double>;
template class NewtonSolver<std::complex<double>>;
```

main.cpp is:

```
#include "newton.hpp"
#include <complex>
#include <functional>
#include <iostream>
int main() {
  // Function with real root: f(x) = x^2 - 1 = 0.
```

```
  {
    // Define function and its derivative.
    auto f = [](const double &x) { return x * x - 1.0; };
    auto df = [](const double &x) { return 2.0 * x; };
    // Create a NewtonSolver instance and find the root.
    const double x0 = 0.5;
    NewtonSolver<double> solver(f, df, x0);
    const double root = solver.solve();
    // Indicates failure to converge
    if (root != std::numeric_limits<double>::quiet_NaN()) {
      std::cout << "Approximate root: " << root << std::endl;
    } else {
      std::cout << "Failed to converge to a root." << std::endl;
    }
  }
  // Function with complex root: f(x) = x^2 + 1 = 0.
  {
    // Define function and its derivative.
    auto f = [](const std::complex<double> &x) { return x * x +
1.0; };
    auto df = [](const std::complex<double> &x) { return 2.0 *
x; };
    // Create a NewtonSolver instance and find the root.
    const std::complex<double> x0{0.5, 0.5};
    NewtonSolver<std::complex<double>> solver(f, df, x0);

    const std::complex<double> root = solver.solve();
    if (root !=
std::numeric_limits<std::complex<double>>::quiet_NaN()) {
      std::cout << "Approximate root: " << root << std::endl;
    } else {
      std::cout << "Failed to converge to a root." << std::endl;
    }
  }
  return 0;
}
```

Slide 6: Exercise 4: template metaprogramming
Use template metaprogramming to calculate the factorial of an integer at compile time. Define a template class for calculating the factorial of an integer. Instantiate the template for the integers 5, 7 and 10. Use ***static_assert* to ensure that values are computed at compile time rather than runtime**. Print the results of the factorials.

```
#include <iostream>
// Template class to calculate the factorial at compile time.
template <unsigned int N> class Factorial {
public:
  static constexpr unsigned int value = N * Factorial<N -
1>::value;
```

```
};
// Specialization for Factorial<0> (base case).
template <> class Factorial<0> {
public:
  static constexpr unsigned int value = 1;
};

int main() {
  constexpr unsigned int n1 = 5;
  constexpr unsigned int n2 = 7;
  constexpr unsigned int n3 = 10;

  static_assert(Factorial<n1>::value == 120, "Factorial of 5 is
not 120");
  static_assert(Factorial<n2>::value == 5040, "Factorial of 7 is
not 5040");
  static_assert(Factorial<n3>::value == 3628800,
                "Factorial of 10 is not 3628800");

  std::cout << "Factorial of " << n1 << " is: " <<
Factorial<n1>::value
            << std::endl;
  std::cout << "Factorial of " << n2 << " is: " <<
Factorial<n2>::value
            << std::endl;
  std::cout << "Factorial of " << n3 << " is: " <<
Factorial<n3>::value
            << std::endl;
  return 0;
}
```

Lecture 6 (31/10): The Standard Template Library

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/06/06-c++_stl.md

Slide 11:
The size of a vector is how many elements it contains; the capacity is usually larger because it reserves memory for other elements to be added; if you want to avoid reallocation of the memory, you can just resize the capacity of the vector with reserve.

Slide 12:
There is no way to change the size of an array during lifetime because its specified during compiled time.

Slide 31:

```
std::vector<double> x(10);
/*
x.begin()
x.end()
*/
// Let's define an iterator using this syntax:
std::vector<double>::iterator;
    it = x.begin();
*it ---> x[0]; // dereference operator
*(it++) --> x[1]
*(++it) --> x[2]

for(unsigned int i = 0; i < x.size(); ++i)
    x[i] = …;

for (auto it = x.begin(); it != x.end() ; ++it)
    *it = …;

// Range based for loop (since c++17)
for (auto el : x)
    el = ...;
```

Slide 33:

```
for (auto it = x.crbegin(); it != x.crend() ; ++it)
    std::cout << *it << std::cout; // can only use it for reading
```

Slide 52:

```
std::begin(x) // free function
std::end(x)
class MyClass {
    class MyCLassIterator {
        operator->() {...}
        operator++() {...}
     };
    MyClassIterator begin() {...}
    MyClassIterator end() {...}
};
MyClass x;
for (auto el : x)
    el = ...;
```

Lecture 7 (07/11): Move semantics, smart pointers, STL utilities

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/07/07-c++_stl2_move.md

Slide 8-10:
To use *std::unique_ptr* you have to include the library *memory*. It deletes it when it goes out of scope.
std::move is needed to move a pointer from an object to one other because we can't have more than one pointer pointing to the same resource.

```
#include <memory>
int main() {
    int *p = new int(10);
    std::cout << *p;
    // ...
    delete p;
    // make_unique makes a pointer with the value to be
    // given to the constructor
    std::unique_ptr<int> p2 = std::make_unique<int>(10);
    // the memory will be destroyed when p2 goes out of scope

    int * y = p2.get();
    int x = *p2;
    return 0;
}
```

Slide 13:
The main difference is that shared pointers can be copied unlike unique pointers.

Slide 18:
The reference must be initialized at the same location where it's declared, so you can't write *int &c; c=a;* instead you can write *std::reference_wrapper<int> c; c=a; c.get()=15;*

Slide 23:
l stands for left, r stands for right

Slide 27:
Two examples of rvalue are a value returned by copy and non-string literals.

Slide 33:

```
#include <iostream>
```

```
void f(int &a) {
    std::cout << "f(int &a)" << std::endl; }
void f(const int &a) {
    std::cout << "f(const int &a)" << std::endl; }
int main() {
    int x = 10;
    f(x); // 1st void
    f(10); // 2nd void
    // why?
    int &a = x; // 1st void used; is allowed and has priority
    const int &a = x; // 2nd void used; is allowed;
    int &a = 10; // 1st void; is not allowed
    const int &a = 10; // 2nd void; is allowed;
    return 0;
}
```

Why don't we produce a new reference type for which we can have a temporarily bound swap non const reference? To avoid copies.

Slide 41:

```
int main() {
    int x = 10;
    f(x); // 1st void
    f(10); // 2nd void

    int y = std::move(x); // forcing the fact that x is moved
    std::vector<double> x(10000);
    std::vector<double> y = std::move(x); // after this x will be
empty
```

Laboratory 6 (08/11):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/06/06-c++_stl_move.md

Slide 2: Exercise 1: Monte Carlo estimate of π
Consider the square [0,1]^2 and the quarter-circle centered at (0,0) with radius 1. Generate random points within the square. Count how many of these random points fall within the quarter-circle. After generating a sufficient number of random points, you can estimate π with 4 times the number of points inside the quarter-circle divided by the total number of generated points. To improve estimation accuracy, try to increase the number of random points in your simulation.

```
#include <iomanip> // for setprecision
#include <iostream>
```

```cpp
#include <random>
int main() {
  // Initialize random number generator.
  std::random_device rd;
  std::default_random_engine gen(rd());
  // distribution we sample from
  std::uniform_real_distribution<double> distribution(0, 1);

  // Number of random points to generate.
  const unsigned int n_points = 1e7;
  unsigned int n_points_inside_circle = 0;

  for (unsigned int i = 0; i < n_points; ++i) {
    // Generate a random point.
    const double x = distribution(gen);
    const double y = distribution(gen);

    // Check if the point is inside the circle.
    if (x * x + y * y <= 1) {
      ++n_points_inside_circle;
    }
  }

  // Calculate the estimated value of π.
  const double estimated_pi = 4.0 * n_points_inside_circle /
n_points;

  // Print the result with decimal precision
  std::cout << "Estimated value of π: " << std::setprecision(16)
<< estimated_pi
            << std::endl;
  return 0;
}
```

Slide 3-4: Exercise 2: std::set

In a building security system, door locks are opened by entering a 4-digit access code into a keypad. The access code's validation process is handled through an Access object with the following interface:

```cpp
class Access
{
public:
    void activate(unsigned int code);
    void deactivate(unsigned int code);
    bool is_active(unsigned int code) const;
};
```

Each employee is assigned a unique access code, which can be activated using the *activate()* function. When an employee leaves the

company, their access code can be deactivated using the *deactivate()* function.

Your task is to implement the *Access* class as described above. Write a test program that accomplishes the following:

1. Create an instance of the *Access* object.
2. Activate the access codes 1234, 9999, and 9876.
3. Prompt the user to enter an access code, and read the code from the console.
4. Inform the user whether the door opens successfully.
5. Repeat the last two steps until the door successfully opens.
6. Deactivate the code that worked. Also, deactivate the code 9999 and activate the code 1111.
7. Repeat steps 3 and 4 until the door successfully opens.

```
#include <iostream>
#include <unordered_set>
class Access {
public:
  void activate(unsigned int code) { active_codes.insert(code); }

  void deactivate(unsigned int code) { active_codes.erase(code); }

  bool is_active(unsigned int code) const {
    return active_codes.count(code) > 0;
  }

private:
  std::unordered_set<unsigned int> active_codes;
  // since we don't need the order, this is better than std::set
};

int main() {
  Access access_system;
  // Activate access codes.
  access_system.activate(1234);
  access_system.activate(9999);
  access_system.activate(9876);

  unsigned int entered_code;
  bool door_opened = false;

  while (!door_opened) {
    std::cout << "Enter your access code: ";
    std::cin >> entered_code;

    if (access_system.is_active(entered_code)) {
      std::cout << "Door opened successfully!" << std::endl;
```

```
      door_opened = true;
      access_system.deactivate(entered_code);
    } else {
      std::cout << "Access code is not valid. Please try again."
<< std::endl;
    }
  }

  access_system.deactivate(9999);
  access_system.activate(1111);

  door_opened = false;

  while (!door_opened) {
    std::cout << "Enter your access code: ";
    std::cin >> entered_code;

    if (access_system.is_active(entered_code)) {
      std::cout << "Door opened successfully!" << std::endl;
      door_opened = true;
    } else {
      std::cout << "Access code is not valid. Please try again."
<< std::endl;
    }
  }
  return 0;
}
```

Slide 5-6: Exercise 3: std::map

In the previous exercise, the customer using the security system wants to associate an access level with each access code. Users with higher access levels should be able to open doors to more security-sensitive areas of the building compared to users with lower access levels. Start with your solution from the previous exercise and make the following modifications to the *Access* class:

```
class Access
{
public:
    void activate(unsigned int code, unsigned int level);
    void deactivate(unsigned int code);
    bool is_active(unsigned int code, unsigned int level) const;
};
```

The *is_active()* function should return *true* if the specified access code has an access level greater than or equal to the specified access level. If the access code is not active at all, it should return *false*.

Now, update the main program to perform the following tasks:

1. After creating an instance of the *Access* object, activate code 1234 with access level 1, code 9999 with access level 5, and code 9876 with access level 9.
2. Prompt the user to enter an access code, and read the code from the console.
3. Assuming a door requires access level 5 for entry, print whether it opened successfully.
4. Repeat the last two steps until the door opens.
5. Deactivate the code that worked, change the access level of code 9999 to 8, and activate code 1111 with access level 7.
6. Prompt the user for an access code, read the code from the console.
7. Assuming a door requires access level 7 for entry, print whether it opened successfully.
8. Repeat the last two steps until the door opens.

```
#include <iostream>
#include <map>
class Access {
public:
  void activate(unsigned int code, unsigned int level) {
    access_codes[code] = level;
  }

  void deactivate(unsigned int code) { access_codes.erase(code); }

  bool is_active(unsigned int code, unsigned int level) const {
    auto it = access_codes.find(code);
    if (it != access_codes.end()) {
      return it->second >= level;
    }
    return false;
  }

private:
  std::map<unsigned int, unsigned int> access_codes;
};

int main() {
  Access access_system;

  // Activate access codes with access levels.
  access_system.activate(1234, 1);
  access_system.activate(9999, 5);
  access_system.activate(9876, 9);
```

```
  unsigned int entered_code;
  unsigned int required_level = 5;
  bool door_opened = false;

  while (!door_opened) {
    std::cout << "Enter your access code for level " <<
required_level << ": ";
    std::cin >> entered_code;

    if (access_system.is_active(entered_code, required_level)) {
      std::cout << "Door opened successfully!" << std::endl;
      access_system.activate(9999, 8);
      access_system.activate(1111, 7);
      door_opened = true;
      access_system.deactivate(entered_code);
    } else {
      std::cout << "Access code is not valid for the current level
"
                << required_level
                << ". Please try again." << std::endl;
      }
  }
```

```
  required_level = 7;
  door_opened = false;

  while (!door_opened) {
    std::cout << "Enter your access code for level " <<
required_level << ": ";
    std::cin >> entered_code;

    if (access_system.is_active(entered_code, 7)) {
      std::cout << "Door opened successfully!" << std::endl;
      door_opened = true;
    } else {
      std::cout << "Access code is not valid for the current
level."
                "Please try again." << std::endl;
      }
  }
  return 0;
}
```

Slide 7: Exercise 4: STL containers and algorithms

1. Generate a vector: Create a vector named *random_numbers* that contains 100 random integers between 0 and 9.
2. Sort the vector: Create a new vector named *sorted_numbers* by sorting the random_numbers vector in ascending order, with repetitions.

3. Remove duplicates while sorting: Create a new vector named *sorted_unique_numbers* by sorting the *random_numbers* vector and removing duplicate entries.
4. Remove duplicates without sorting: Create a new vector named *unsorted_unique_numbers* by printing unique entries from the *random_numbers* in the same order they appear, without repetitions.

```cpp
#include <algorithm>
#include <iostream>
#include <random>
#include <unordered_set>
#include <vector>
template <typename Container>
void print(const std::string &string, const Container &cont) {
  std::cout << string << ":" << std::endl;
  for (const auto &v : cont) {
    std::cout << v << " ";
  }
  std::cout << std::endl << std::endl;
}

int main() {
  // Initialize random number generator.
  std::default_random_engine gen(42);
  std::uniform_int_distribution<unsigned int> distribution(0, 9);

  // Generate vector of 100 random integers between 0 and 9.
  std::vector<unsigned int> random_numbers(100);
  for (unsigned int &num : random_numbers) {
    num = distribution(gen);
  }
  print("Original vector", random_numbers);

  // Or you could have done it in this other way
  /*
  std::vector<int> random_numbers;
  random_numbers.reserve(100); // capacity = 100, size = 0
  for (size_t i = 0; i < random_numbers.capacity(); ++i)
    random_numbers.push_back(distribution(gen));
  */

  // or in this other way
  /*
  std::vector<int> random_numbers;
  for (size_t i = 0; i < random_numbers.size(); ++i)
    random_numbers[i] = distribution(gen);
  */

  // Sort the vector.
```

```
  std::vector<unsigned int> sorted_numbers(random_numbers);
  std::sort(sorted_numbers.begin(), sorted_numbers.end());
  print("Sorted numbers", sorted_numbers);

  // Remove duplicates while sorting.
  std::vector<unsigned int> sorted_unique_numbers(random_numbers);
  std::sort(sorted_unique_numbers.begin(),
sorted_unique_numbers.end());
  auto unique_end =
      std::unique(sorted_unique_numbers.begin(),
sorted_unique_numbers.end());
  sorted_unique_numbers.erase(unique_end,
sorted_unique_numbers.end());
  print("Sorted unique numbers", sorted_unique_numbers);

  // Remove duplicates without sorting.
  std::vector<unsigned int> unsorted_unique_numbers;
  std::unordered_set<unsigned int>
unique_numbers(random_numbers.begin(),
                                    random_numbers.end());

  for (const unsigned int &element : random_numbers) {
    if (unique_numbers.find(element) != unique_numbers.end()) {
      unique_numbers.erase(element);
      unsorted_unique_numbers.push_back(element);
    }
  }
  // or it could have been done in this way
  /*
  for (auto v : random_numbers) {
    if (unique_numbers.count(v)) {
      unsorted_unique_numbers.push_back(v);
      unique_numbers.erase(v);
    }
  }
  */
  print("Unsorted unique numbers", unsorted_unique_numbers);
  return 0;
}
```

Slide 8: Exercise 5: Word frequency analysis
The file *input.txt* contains a list of random complete sentences in English. Develop a C++ program that reads the file, calculates the frequency of each word in the text, and outputs the word-frequency pairs to a new file in a dictionary format.
Write a C++ program to process the input text file by splitting it into words and counting the occurrences of each unique word. Spaces and punctuation should be discarded.

The program should generate a new file (named *output.txt*) containing the word-frequency pairs in a dictionary format. Each line in the output file should consist of a word followed by its frequency, separated by a colon or any other suitable delimiter.
(Bonus): sort the output by frequency, in descending order. If two words have the same frequency, then sort them alphabetically.

```cpp
#include <algorithm>
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

int main() {
  // Define file paths.
  const std::string input_file_path = "/input.txt";
  // if it was in a folder behind you would have written with ../
before:
  // const std::string input_file_path = "../input.txt";
  const std::string output_file_path = "ex5_output.txt";

  // Open input file.
  std::ifstream input_file(input_file_path);
  if (!input_file.is_open()) {
    std::cerr << "Error opening input file: " << input_file_path
<< std::endl;
    return 1;
  }

  // Create a map to store word-frequency pairs.
  std::map<std::string, int> word_frequency_map;

  // Read input text file.
  std::string line;
  while (getline(input_file, line)) {
    // Split the line into words.
    std::istringstream iss(line); // take the line
    std::string word; // divide it into words
    // since you don't know how many words there are, use a while
    while (iss >> word) {
      // Convert word to lowercase for case-insensitive analysis
      // and remove punctuation.
      std::transform(word.begin(), word.end(), word.begin(),
                     [](char ch) -> char {
                       if (ispunct(ch)) // if it's a punctuation
                         return '\0'; // string terminator
                       return tolower(ch); // converts to
lowercase
```

```
                    });

      // Increment word frequency. operator[] automatically
allocates
      // a new entry if it was not present yet.
      word_frequency_map[word]++;
    }
  }

  // Close input file.
  input_file.close();

  // Sort the output by frequency, in descending order.
  // If two words have the same frequency, then sort them
alphabetically.

  // Being an associative container, a std::map cannot be sorted.
  // So let's convert it to a std::vector, and sort it instead.
  std::vector<std::pair<std::string, int>> word_frequency_vector(
      word_frequency_map.begin(), word_frequency_map.end());
  std::sort(word_frequency_vector.begin(),
word_frequency_vector.end(),
            [](const auto &a, const auto &b) {
              return (a.second > b.second || a.first < b.first);
            });

  // Open output file.
  std::ofstream output_file(output_file_path);
  if (!output_file.is_open()) {
    std::cerr << "Error opening output file: " << output_file_path
<< std::endl;
    return 1;
  }

  // Write word-frequency pairs to output file.
  for (const auto &[word, frequency] : word_frequency_vector) {
    output_file << word << ": " << frequency << std::endl;
  }

  // Close output file.
  output_file.close();
  return 0;
}
```

Slide 9: Exercise 6: Move semantics for efficient data transfers
Define a class Vector that represents a one-dimensional vector of double values, stored as a raw pointer *double * data*.

1. Implement a move constructor for the *Vector* class that transfers ownership of the underlying data from the source vector to the

destination vector. The move constructor should ensure that the source vector's data is no longer accessible after the transfer.

2. Define a copy and a move assignment operator for the *Vector* class that allows for the efficient transfer of ownership of the underlying data from one Vector object to another. Similar to the move constructor, the move assignment operator should ensure that the source vector's data is no longer accessible after the transfer.
3. Compare the performance of copying and moving large vectors using both copy semantics and move semantics. Measure the time taken to copy and move vectors by increasing the input size from 2^20 to 2^30 elements. Analyze the results and observe the performance gain achieved by using move semantics.

```
#include <chrono> // for measuring time
#include <cmath> // for power
#include <iostream>
class Vector {
public:
  // Constructor.
  Vector(const unsigned int &size = 0) : data(new double[size]),
size(size) {}

  // Move constructor.
  Vector(Vector &&other) noexcept {
    // noexcept means that it doesn't throw exceptions
    data = other.data;
    size = other.size;
    other.data = nullptr;
    other.size = 0;
  }

  // Copy assignment operator.
  Vector &operator=(const Vector &other) noexcept {
    if (this != &other) {
      delete[] data;
      size = other.size;
      data = new double[size];
      for (unsigned int i = 0; i < other.size; ++i) {
        data[i] = other.data[i];
      }
    }
    return *this;
  }

  // Move assignment operator.
  Vector &operator=(Vector &&other) noexcept {
```

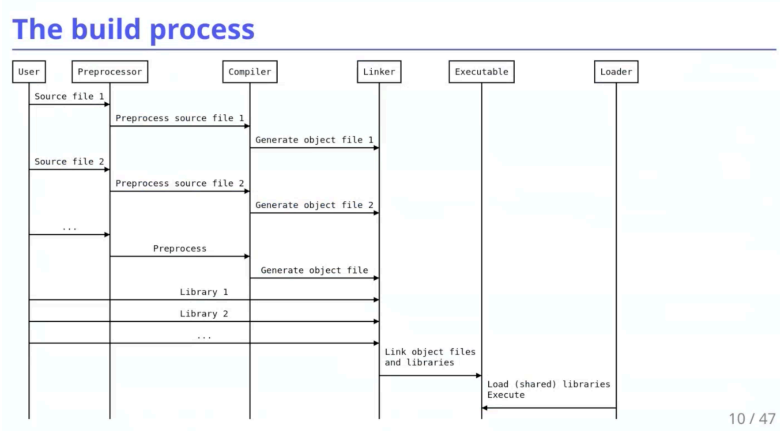
```
    if (this != &other) {
      delete[] data;
      data = other.data;
      size = other.size;
      other.data = nullptr;
      other.size = 0;
    }
    return *this;
  }

  // Destructor.
  ~Vector() { delete[] data; }

private:
  double *data;
  unsigned int size;
};

void measure_copy(unsigned int size) {
  Vector source(size);
  Vector destination(size);
  auto start = std::chrono::high_resolution_clock::now();
  destination = source;
  auto end = std::chrono::high_resolution_clock::now();
  std::cout << "Time taken to copy vector of size " << size << ":
"
            <<
std::chrono::duration_cast<std::chrono::milliseconds>(end -start)
                    .count()
            << " milliseconds." << std::endl;
}

void measure_move(const unsigned int &size) {
  Vector source(size);
  Vector destination(size);
  auto start = std::chrono::high_resolution_clock::now();
  destination = std::move(source);
  auto end = std::chrono::high_resolution_clock::now();
  std::cout << "Time taken to move vector of size " << size << ":
“ << std::chrono::duration_cast<std::chrono::microseconds>(end
-start)
                    .count()
            << " microseconds." << std::endl;
}

int main() {
  for (unsigned int i = 20; i < 31; ++i) {
    measure_copy(std::pow(2, i));
    measure_move(std::pow(2, i));
    std::cout << std::endl;
  }
  return 0;
```

```
}
```

Lecture 8 (09/11): Libraries: principles, generation and use

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/08/08-c++_libraries.md

Slide 10: When talking about static libraries, we stop at the



penultimate
column, while with shared libraries we go on with the last column.

Slide 12:
The eigen.cpp is:

```cpp
#include <Eigen/Dense>
#include <iostream>
int main() {
  Eigen::MatrixXd m(2, 2);
  m(0, 0) = 3;
  m(1, 0) = 2.5;
  m(0, 1) = -1;
  m(1, 1) = m(1, 0) + m(0, 1);
  std::cout << "The matrix is:" << std::endl << m << std::endl;
  return 0;
}
```

Slide 15:
You have to specify that main.cpp is in other folder with *g++ -Imylib/ -c main.cpp*
nm main.cpp takes all the singles defined in that file and prints them.

The flag -*C* is to print without caring about overloads.

Slide 19:
-L points to the directory containing the library; -l points to the name of the library without directory and extension.

Slide 23:
Invoking the linker through the g++ command invokes c++
ar creates an archive

Slide 38:
PIC means that the generated object file can be put into a shared library, so it can be loaded at runtime in a different address of the memory.
Always put it in shared libraries.

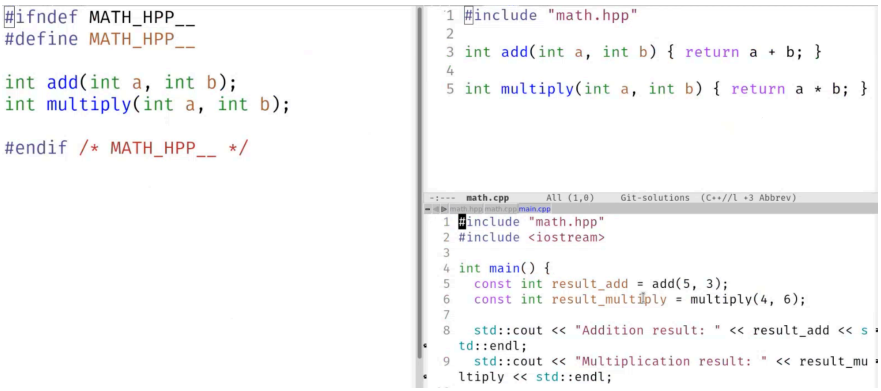
Slide 39:
g++ main.o -Lmylib/ -lmylib -o main just checks if this library contains all the symbols that are unresolved in main.o, it doesn't inject them in the final executable. To instruct the loader: *LD_LIBRARY_PATH+=:$(pwd)/mylib/ ./main*

Laboratory 7 (16/11): introduction to GNU Make + Lab

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/07/07-c++_make_libraries.md

Slide 4:
math.hpp, math.cpp and main.cpp are those:

As usual type
g++ -c math.cpp
g++ -c main.cpp
g++ math.o main.o -o main
./main



```
#ifndef MATH_HPP__
#define MATH_HPP__

int add(int a, int b);
int multiply(int a, int b);

#endif /* MATH_HPP__ */
```

```
#include "math.hpp"

int add(int a, int b) { return a + b; }

int multiply(int a, int b) { return a * b; }
```

```
-:---  math.cpp      All (1,0)     Git-solutions  (C++//l +3 Abbrev)
#include "math.hpp"
#include <iostream>

int main() {
  const int result_add = add(5, 3);
  const int result_multiply = multiply(4, 6);

  std::cout << "Addition result: " << result_add << s
td::endl;
  std::cout << "Multiplication result: " << result_mu
ltiply << std::endl;
```

If you modify math.hpp you have to recompile math.cpp and then main.cpp, redo the linking (*g++ math.o main.o -o main*) and so on with others, there is a faster way with a Makefile.

Slide 7:
These things (remember the tab!) are written in the Makefile, so from terminal then you'll only write *make* and it will do everything.

Slide 8:
DEPS are the **dependencies**.
Every time you want to recompile with other stuff, do ***make clean*** and then again ***make***. Remark that clean is not a file, it's a command.

The Makefile can be:

```
%.0: %.cpp $(DEPS)
    $(CXX) -c $(CPPFLAGS) $(CXXFLAGS) $< -o $@
```

Which means that if you type *make math.o* it does

```
math.o: math.cpp math.hpp
    g++ -c [flags…] math.cpp -o math.o
```

Slide 9:
all means that if you don't specify in runs the thing said after, in this case (*all: main*) is the main.

6. Sara Serafino
17 novembre 2023 alle ore 15:31:55
how to download and extract zip

Slide 10:
wildcard *.cpp takes all .cpp files in the folder and makes a list of them.
OBJ generates a .o file for each .cpp file.
If you modify only one main.o, you only have to do *make main*

Slide 14:
Usual Makefile prototype to link against a library:

```
CXX=g++
CPPFLAGS=-Imath/
CXXFLAGS=-std=c++17 -Wall -Wpedantic -Werror
LDFLAGS=-Wl,-rpath,math/ -Lmath/ # For dynamic linking.
# LDFLAGS=-Lmath/ -static        # For static linking.
LDLIBS=-lmath

SRC=main.cpp
OBJ=$(SRC:.cpp=.o)

all: main

main: $(OBJ)
    $(CXX) $(CXXFLAGS) $^ $(LDFLAGS) $(LDLIBS) -o $@

%.o: %.cpp
    $(CXX) -c $< $(CPPFLAGS) $(CXXFLAGS) -o $@

clean:
    rm -f *.o main
```

Slide 17: Exercise 1: building and using muParserX
Download and extract muParserX :
6 To download *wget https://github.com/beltoforion/muparserx/archive/refs/tags/v4.0.12.tar.gz*
To extract *tar xvzf nomefile.tar.gz*
The source files of *muParserX* are located inside the *muparserx-4.0.12/parser/* folder. In that folder, write a *Makefile* to compile *muParserX* into a shared library:
So for starting from a prototype of Makefile write, from the folder /parser/: *cp ../../../../advanced_programming_2023-2024/exercises/07/examples/makefile_library/math/Makefile .*
Then open it with vim (*vim Makefile*) and modify it (in the solutions it's called "Makefile_muparser"):

7. Sara Serafino
17 novembre 2023 alle ore 18:15:18
used by Mac

```
CXX=g++
CPPFLAGS=-I.
CXXFLAGS=-std=c++17 -Wall -Wpedantic -Werror

SRC=$(wildcard *.cpp)
OBJ=$(SRC:.cpp=.o)
DEPS=$(wildcard *.h)

LIB_NAME_SHARED=libmuparserx.dylib

all: $(LIB_NAME_SHARED)

$(LIB_NAME_SHARED): $(OBJ)
    g++ $(CXXFLAGS) -shared -dynamiclib $^ -o $@

%.o: %.cpp $(DEPS)
    $(CXX) -c -fPIC $(CPPFLAGS) $(CXXFLAGS) $< -o $@

clean:
    rm -f *.o $(LIB_NAME_SHARED)
```

Once done write *make* and then *make clean* for the next step.
Write a Makefile that compiles and links the program in *hints/ex1.cpp* with *muParserX* .
We now link ex1.cpp to it:
cd
cd Desktop/Advanced\ Programming/lab\ exercises/lab07/hints
g++ -I${HOME}/Desktop/Advanced\ Programming/lab\ exercises/lab07/muparserx-4.0.12/parser/ -c ex1.cpp
Where ex1.cpp is:

```
#include "mpParser.h"
#include <iostream>
int main() {
  // Create the parser instance.
  mup::ParserX p;
  // Create some basic values.
  mup::Value a(2.0);
  mup::Value c(mup::cmplx_type(1, 1));
  mup::Value s("Hello World");
  mup::Value f(1.1);
  // Now add the variable to muParser.
  p.DefineVar("a", mup::Variable(&a));
  p.DefineVar("c", mup::Variable(&c));
```

```
  p.DefineVar("s", mup::Variable(&s));
  p.DefineVar("f", mup::Variable(&f));
  p.SetExpr("a + c * strlen(s) - f");
  for (unsigned int i = 0; i < 10; ++i) {
    // Evaluate the expression and change the value of
    // the variable c in each turn.
    c = mup::cmplx_type(1.1 * i, 1.1 * i);
    const mup::Value result = p.Eval();
    // Print the result.
    std::cout << result << std::endl;  }
  return 0;
}
```

Redirect the loader so to run the executable:
g++ ex1.o -L${HOME}/Desktop/Advanced\ Programming/lab\ exercises/lab07/muparserx-4.0.12/parser/ -lmuparserx -o ex1
LD_LIBRARY_PATH+=:${HOME}/Desktop/Advanced\ Programming/lab\ exercises/lab07/muparserx-4.0.12/parser ./ex1

Or you can write the last two lines in a Makefile like:

```
CXX=g++
CPPFLAGS=-Imuparserx/parser
CXXFLAGS=-std=c++17 -Wall -Wpedantic -Werror
LDFLAGS=-Wl,-rpath,muparserx/parser -lmuparserx/
parser
LDLIBS=-lmuparserx

SRC=ex1.cpp
OBJ=$(SRC:.cpp=.o)

all: ex1

ex1: $(OBJ)
    $(CXX) $(CXXFLAGS) $^ $(LDFLAGS) $(LDLIBS)-o $@

%.o: %.cpp
    $(CXX) -c $< $(CPPFLAGS) $(CXXFLAGS) $< -o $@

clean:
    rm -f *.o ex1
```

Slide 18: Exercise 2: shared libraries

The *hints/ex2/* directory contains a library that implements a gradient descent algorithm for linear regression, accompanied by a source file *ex2.cpp* utilizing this library. Unfortunately, the gradient descent code within the library contains a bug. Your tasks are:

1. Compile the library and test file, using the provided Makefiles. For this just write *make* inside the folder of ex2, it will tell you that it didn't converge.
2. Inspect the code to locate the bug within the gradient descent algorithm. Seeing the algorithm you should put -= instead of += in line 17-18.
3. Once the bug is identified, fix it within the code. Then, compile an updated version of the library, incorporating the bug fix. In the Makefile of gradient_descendent there is versioning: *LIB_VERSION_MAJOR=1 LIB_VERSION_MINOR=0* so you change the last one in *1* because it's version 1.1. So now it looks like:

```
CXX=g++
CPPFLAGS=-I.
CXXFLAGS=-std=c++17 -Wall -Wpedantic -Werror

SRC=$(wildcard *.cpp)
OBJ=$(SRC:.cpp=.o)
DEPS=$(wildcard *.hpp)

LIB_NAME=libgradient_descent.so
LIB_VERSION_MAJOR=1
LIB_VERSION_MINOR=1

LIB_NAME_REAL=$(LIB_NAME).$(LIB_VERSION_MAJOR).$
(LIB_VERSION_MINOR)

all: $(LIB_NAME_REAL)

$(LIB_NAME_REAL): $(OBJ)
    g++ $(CXXFLAGS) -shared $^ -Wl,-soname,$
(LIB_NAME).$(LIB_VERSION_MAJOR) -o $@
    ln -sf $(LIB_NAME_REAL) $(LIB_NAME).$
(LIB_VERSION_MAJOR)
    ln -sf $(LIB_NAME).$(LIB_VERSION_MAJOR) $
(LIB_NAME)
```

```
%.o: %.cpp $(DEPS)
    $(CXX) -c -fPIC $(CPPFLAGS) $(CXXFLAGS) $< -o $@

clean:
    rm -f *.o *.so*
```

4. Execute the test case to verify that the bug fix successfully addresses the issue (just write *make*). Please note that, since we are dealing with a shared library, this verification should be conducted without the need for recompilation or relinking of the test file.

Slide 19: Exercise 3: order matters

The *hints/ex3/* directory contains a source file *ex3.cpp* that uses a library *graphics_lib* , which depends on another library *math_lib* .

1. Generate a static library *libmath.a* .
2. Generate a static library *libgraphics.a* .
3. Compile ex3.cpp into an object file *ex3.o* .
4. Link *main.o* against *libmath.a* and *libgraphics.a* to produce the final executable.

What is the correct order for passing *ex3.o*, *libmath.a*, and *libgraphics.a* to the linker to successfully resolve all the symbols? Would the same considerations apply if dynamic linking (shared libraries) were used instead of static linking?

In the end the command will be *g++ ex3.o -L. -lgraphics_lib -lmath -o ex3*

Slide 20: Exercise 4: dynamic loading

This exercise showcases dynamic loading, the building block for implementing a plugin system.

The *hints/ex4/* contains a module *functions* containing the definition of three mathematical functions. The source file *functions.cpp* gets compiled into a shared library *libfunctions.so*, using C *linkage* to prevent name mangling .

Notably, when compiling the source file *ex4.cpp* into an executable, there is no need to link against *libfunctions.so* .

1. Fill in the missing parts in *ex4.cpp* to dynamically load the library.

2. Prompt the user for the function name to evaluate at a given point, selecting from the ones available in the library.
3. Perform the evaluation and print the result.
4. Release the library.

ex4.cpp is:

```
#include <dlfcn.h> // dynamic library
#include <iostream>
#include <string>
/*
void check_error(int error_code) {
  auto error = dlerror();
  if (error != nullptr) {
    std::cerr << "Error: " << error << std::endl;
    std::exit(error_code);
  }
}
*/
int main() {
  // Load libfunctions.so.
  void *handle = dlopen("./libfunctions.so", RTLD_LAZY);
  // RTLD_LAZY does it at runtime
  char* error = dlerror();
  if (error != nullptr)
    std::cerr << error << std::endl;

  // Prompt the user for the function to evaluate.
  std::string answer;
  std::cout << "cube, square or square_root: ";
  std::cin >> answer;

  // Load the proper symbol from libfunctions.so
  auto fun = reinterpret_cast<double (*) (const double &)>(
      dlsym(handle, answer.c_str()));
  // .c:str() converts a string into an array of characters
  // dlsym returns a pointer to void
  // reinterpret_cast interprets it as a function pointer

  char* error = dlerror();
  if (error != nullptr)
    std::cerr << error << std::endl;

  // Prompt the user for the value to evaluate the function at.
  double value;
  std::cout << "x = ";
  std::cin >> value;

  // Evaluate function and print result.
  const double result = fun(value);
```

8. Sara Serafino
19 novembre 2023 alle ore 15:50:01
link against dynamic library

```
  std::cout << answer << "(" << value << ") = " << result <<
std::endl;

  // Close libfunctions.so.
  dlclose(handle);

  return 0;
}
```

Write from terminal:

8 *g++ ex4.cpp -ldl –o ex4*
./ex4

functions.hpp is:

```
#ifndef FUNCTIONS_HPP__
#define FUNCTIONS_HPP__

// extern "C" needed to avoid name mangling.
extern "C" {
double square(const double &x);

double cube(const double &x);

double square_root(const double &x);
}
#endif /* FUNCTIONS_HPP__ */
```

C++ allows you to define multiple overloads of the same function (for example int of the above) but it wasn't allowed in C, which is the language of these files. Let's say we don't put *extern "C"*. Compiling
g++ -c functions.cpp
g++ functions.o -shared -o libfunctions.so
nm libfunctions.so
It prints info about input and output not human readable because it has other symbols (called mangling). Writing *extern "C"* means the compiler treats these functions as if they were compiled by a C compiler, so the mangling doesn't occur.

Lecture 9 (21/11): introduction to CMake, optimization, debugging, profiling, testing

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/09/09-c++_cmake_optimization_debug_test.md

Slide 5:
Makefile is not platform independent, while CMake is.

Slide 6:
Downloading Doxygen and GSL. In Doxygen there is the Cmake, while in GSL there is *configure*.

In the folder of gsl we write *./configure –prefix=/opt/gsl/ –enable-shared –disable-static CXX=g++* where with *–prefix* you can specify in which folder you want to install it and since we want only dynamic libraries we write *–disable-static*.
If any of the checks that it does fails, it stops.
We end up with a Makefile which is really long and human unreadable. Doing ***make -j<N>*** it **compiles everything to build the library**. In this case we do *make -j16* where 16 is the number of processor.
To install it in folders we write *sudo make install*, now the system has created a directory named "/opt/gsl/" with include, lib and share folders.
Let's do a similar thing for Doxygen. First of all let's create a folder *build*. In build write *cmake* followed by the name of the folder in which CMakeLists.txt is; in this case *cmake ../*. This is the equivalent of the previous configure; also here you can specify flags. (If you don't have cmake, install it).
If it gives error saying that it can't find FLEX, do *sudo apt install flex*. Apt means that somebody already did it, you just download. You redo *cmake ../* and it works. You then will have a Makefile automatically generated.
If you **want to specify the folder for installing** cmake, write ***cmake .. -DCMAKE_INSTALL_PREFIX=/****opt/doxygen/*. Now you can run *make -j6*

Slide 9:
We have 3 different folders: build, source and installation (contains only files needed to be used by other projects).

Slide 10:
A target can be an executable or a library.

Slide 12:
With set you define local variables.

Slide 13:
Otherwise you can define cache variables which take a variable from inside the Cmake and are controllable by user from outside.

Slide 15:
If you want to overwrite a library path you can do it this way, even though it's not good.

Slide 19:
We're starting to see how to inject variables from outside into C++

Slide 20:
With fatal error the compilation stops.
Adding this ***VERBOSE=1*** it prints the whole commands that is being executed.

Slide 21:
If everyone has a library but in different folders, use FindPackage.
Here in the example my module uses only filesystem and graph.

Slide 22:
If it's true that Boost is found, it does those things.

Slide 26:
If you put your code online remember the license!

Slide 30:
0 means no optimization (exactly the code that I wrote), 3 is the highest level (and we can't control how the program runs it, but typically it's the fastest), s generates the smallest possible executable.

Slide 31:
Unrolling small loops is more efficient.

Slide 32:
In this way we save time but consume a bit more memory (one extra variable), but it's much better to evaluate the value just once than every time in the loop.

Slide 34-35:
There's no better strategy, it depends on you compiler: some can vectorize.

Slide 38:
Static analysis tools only look at the code without compiling it.

Slide 40:
In debuggers you can set breakpoints like "run until it comes this thing".

Laboratory 8 (23/11):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/08/08-c++_cmake_optimization_debug_test.md

Slide 3: Exercise 1
Use CMake on ex1 of last time.
Since there is a CMakeLists.txt, instead of writing a Makefile, we can exploit it. Do the same:

```
cd Desktop/Advanced\ Programming/TPL/muparserx-4.0.12
mkdir build
cd build
cmake ../ -DCMAKE_INSTALL_PREFIX=/opt/muparserx
```

```
make -j8
```

sudo make install copier hpp and libraries and summaries in the specified folder.
Remove the Makefiles of ex1 of last time because we don't need it anymore (I did a copy in Lab08). Create CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.12)

project(example_muparser VERSION 1.0
    DESCRIPTION "some project description"
    LANGUAGES CXX)

set(MUPARSERX_ROOT "/opt/muparserx")

add_executable(ex_mup ex1.cpp)
# first target name, then source files to compile for generating
the executable target
target_include_directories(ex_mup
    PRIVATE
    ${MUPARSERX_ROOT}/include)

target_link_directories(ex_mup
    PRIVATE
    ${MUPARSERX_ROOT}/lib) # like writing -L/opt/muparserx/lib

target_link_libraries(ex_mup muparserx) # like writing -lmuparserx
```

Change the include of ex1.cpp from

```
#include "mpParser.h"
```

to

```
#include "muparserx/mpParser.h"
```

Now like before do:

```
mkdir build
cd build
cmake ../
make
./ex_mup
```

And it prints the results of the exercise. If the user has the library in another folder, remove all to remake the cmake (*rm -rf* *), in the CMakeLists.txt modify 7th line adding cache… (can also be a boolean) and removing the path that can be set later:

```
set(MUPARSERX_ROOT "" CACHE STRING "Path to muparserx
installation.")
cmake .. -DMUPARSERX_ROOT=/opt/muparserx
```

Where any user can set the path and don't modify the CMakeLists.txt. cmake also generated an important file: CMakeCache.txt that contains all the variables etc internally used by cmake.
We can also compile using different levels of optimizations. By default it's the optimized version. You can have multiple make directories with different flags. After cancelling everything, let's build another directory named build_debug and:

```
cmake .. -DMUPARSERX_ROOT=/opt/muparserx -DCMAKE_BUILD_TYPE=Debug
```

And you can enable whatever flag with:

```
cmake .. -DMUPARSERX_ROOT=/opt/muparserx -DCMAKE_BUILD_TYPE=Debug
-DCMAKE_CXX_FLAGS="-std=c++17 -Wall"
```

At the beginning use the debug version to correct bugs, then when realizing use the higher level of optimization so that it is faster.

Now do the same with ex3 of last time.

```
cmake_minimum_required(VERSION 3.12)

project(Ex07_3 VERSION 1.0
  DESCRIPTION "Exercise 3 from session 07"
  LANGUAGES CXX)

add_library(graphics STATIC graphics_lib.cpp)
add_library(math STATIC math_lib.cpp)
target_link_libraries(graphics math)

add_executable(ex3 ex3.cpp)
target_link_libraries(ex3 graphics)
```

Slide 7:
The CPU works in a way that all variables must be contained in a single row or integer multiples of a single row.
The padding is an empty memory that fills the memory until it can be done in the above way.
So instead of having 8 bytes, we have 12 bytes.

Slide 7:
Just changing the order of definitions of int/char etc the memory used is less due to what explained before.

Slide 10: Exercise 2: Optimization

matrix.cpp is

```
#include "matrix.hpp"
#include <cassert>
Matrix Matrix::transpose() const {
  Matrix result(n_cols(), n_rows());
  for (size_t i = 0; i < result.n_rows(); ++i) {
    for (size_t j = 0; j < result.n_cols(); ++j) {
      result(i, j) = (*this)(j, i);
    }
  }
  return result;
}
Matrix operator*(const Matrix &A, const Matrix &B) {
  assert(A.n_cols() == B.n_rows());
  Matrix result(A.n_rows(), B.n_cols());
  // for (size_t i = 0; i < result.n_rows(); ++i)
  //   for (size_t j = 0; j < result.n_cols(); ++j)
  //     for (size_t k = 0; k < A.n_cols(); ++k)
  //       result(i, j) += A(i, k) * B(k, j);
  const Matrix At = A.transpose();
  for (size_t i = 0; i < result.n_rows(); ++i)
    for (size_t j = 0; j < result.n_cols(); ++j)
      for (size_t k = 0; k < A.n_cols(); ++k)
        result(i, j) += At(k, i) * B(k, j);
  return result;
}
std::vector<double> operator*(const Matrix &A, const
std::vector<double> &x) {
  assert(A.n_cols() == x.size());
  std::vector<double> y(A.n_rows());
  for (size_t i = 0; i < y.size(); ++i)
    for (size_t j = 0; j < A.n_cols(); ++j)
      y[i] += A(i, j) * x[j];
  return y;}
```

ex2.cpp is

```
#include <chrono>
#include "matrix.hpp"
int main(int argc, char **argv) {
  const size_t size = 2000;
  Matrix A(size);
  Matrix B(size);
  for (size_t i = 0; i < size; ++i) {
    A(i, i) = 10.0;
    A(i, size - 1) = 30.0;
    B(i, 0) = 1.0;
    B(i, i) = 3.0;
  }
  {
    const auto start = std::chrono::high_resolution_clock::now();
```

9. Sara Serafino
27 novembre 2023 alle ore 11:45:04
run the debugger

```
    const Matrix D = A.transpose();
    const auto end = std::chrono::high_resolution_clock::now();
    const auto duration =
        std::chrono::duration_cast<std::chrono::milliseconds>(end
- start).count();
    std::cout << "Transpose time: " << duration << " [ms] " <<
std::endl;
  }
  {
    const auto start = std::chrono::high_resolution_clock::now();
    const Matrix C = A * B;
    const auto end = std::chrono::high_resolution_clock::now();
    const auto duration =
        std::chrono::duration_cast<std::chrono::milliseconds>(end
- start).count();
    std::cout << "Multiply time: " << duration << " [ms] " <<
std::endl;
  }
  return 0;
}
```

Using *valgrind –tool=callgrind* we generate a file *callgrind* which is human unreadable but doing *kcachegrind callgrind.out.59129* it shows a graphic representation of what is happening.
Code coverage tells you how many times each line of code and each function were executed. See the Makefile_coverage that summarizes the steps to do: no optimization, add the flag *–coverage*.
From terminal write: *make listclear; make -f Makefile_coverage; ./ex2*.
In the Makefile_coverage there is a tool called ***lcov*** that generates an unreadable test_coverage_info file. With ***genhtml*** we get a webpage with relevant info: *genhtml* test_coverage.info *–output test_coverage*.
Slide : Exercise 3: Debugging

9 To **run the debugger**, a common debugger is gbd:
g++ -O0 -g student1.cpp -o student1
gbd ./student1
It opens an interface with command that can be executed. We begin with *run*. As said in the student1.cpp we want a breakpoint at line 6 i.e. for the program to stop, thus: *break student1.cpp:6* where 6 is the line at which we want to break. We can *run* again and it stops at "this->name = name;". We can *print name* and it says "Alice", we can *print this->name* and it says "" because it's not been executed yet.
Then ***next*** means **only run the next line of code**.

10. Sara Serafino
27 novembre 2023 alle ore 12:04:23
memory leaks

***tui enable* shows the portion of code** that you're actually running.
continue continues running until the next breakpoint.
The *delete* in the code calls the destructor and you can see it by typing *step*.
If the program has encountered an error, with *backtrace* we print all the calls up to where the problem is.
10 ***valgrind –tool=memcheck –leak-check=full*** *./student2* checks the memory leaks and tells where it is.

Slide : Exercise 4: Testing

Lecture 10 (30/11): introduction to Python. Built-in data types. Variables, lists, tuples, dictionaries and sets. Control structures. Functions. Docstrings.

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/10/10-py_intro.md

Slide 24:
Lists can be modified, tuples not.

Slide 66:
Also strings once defined can't be modified in characters.

Slide 67:
You can modify the inner list of a tuple, but not the general list.

Slide 80:
args must follow a specific order.

Laboratory 9 (30/11):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/09/09-py_intro.md

Slide 2: Exercise 1: numerical types, strings

```
#1: 5 to the power of 5
print (5**5)
```

```
3125
```

```
#2: remainder from diving 73 by 6
print(76 % 6)
```

```
4
```

```
#3: how many times does the whole number 3 go into 123?
print (123 // 3)
# What is the remainder of diving 123 by 3?
print (123 % 3)
```

```
41
0
```

```
#4: split the string
s = "apple#banana#cherry#orange"
# into a list by splitting on the # character
print(s.split("#"))
```

```
['apple', 'banana', 'cherry', ‘orange']
```

```
#5: Use string methods to extract the website domani from an email
email = "pafrica@fakemail.com"
print(email.split("@")[-1].split(".com")[0])
```

```
fakemail
```

```
#6: Use f-strings to print "The speed of light is 2.9.. m/s"
thing = "light"
speed = 299792458 # m/s
print(f"The speed of {thing} is {speed:2e} m/s")
```

```
The speed of light is 2.997925e+08 m/s
```

Slide 3: Exercise 1: lists, dictionaries, tuples

```
#7: Use indexing to grab the word "AdvProg"
l = [10, [3, 4], [5, [100, 200, ["AdvProg"]], 23, 11], 1, 7]
print(l[2][1][2]) #printa fra ' '
print(*l[2][1][2]) #printa il valore
```

```
['AdvProg']
AdvProg
```

```
#8: Grab the word "AdvProg"
d = {
    "outer": [
        1,
        2,
        3,
```

```
        {"inner": ["this", "is", "inception", {"inner_inner": [1,
2, 3, "AdvProg"]}]},
    ]
}

print(d['outer'][3]['inner'][3]['inner_inner'][3])
```

```
AdvProg
```

```
#9: Why does the following cell return an error?
t = (1, 2, 3, 4, 5)
t[-1] = 6
#because it's a tuple so you can't modify it
```

Slide 4: Exercise 1: if-else

```
#10: Given the variable "language" which contains a string, use
if-elif-else
# to write a program that prints...
language = "Python"
if language.lower() == "python":
    print("I love snakes!")
elif language.lower() == "c++":
    print("Are you a pirate?")
else:
    print(f"What is {language}?")
```

```
I love snakes!
```

Slide 5: Exercise 2: functions, comprehension

```
#1: create a function that grabs the website domain from a url
string
def website(url):
    return url.split(".")[1]

print(website("www.google.com"))
```

```
google
```

```
#2: create a function that accepts two integers and returns true
if a
# is divisible by b without remainder
def divisible(a, b):
    return True if a % b == 0 else False

print(divisible(10, 3))
print(divisible(6, 3))
```

```
False
True
```

```
#3: use list comprehension to square every number of this list:
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
print([_ ** 2 for _ in l])
```

[1, 4, 9, 16, 25, 36, 49, 64, 81]

```
#4: write a list comprehension that creates a list of only words
# that start with a capital letter
names = ['Steve Irwin', 'koala', 'kangaroo', 'Australia',
'Sydney', 'desert']
print([_ for _ in names if _[0].isupper()])
```

['Steve Irwin', 'Australia', 'Sydney']

Slide 6: Exercise 2: comprehension, exceptions

```
#5: use dictionary comprehension to create a dictionary of the
form
# {'key-0': 0, 'key-1': 1, etc}
keys = [f"key-{k}" for k in range(10)]
vals = range(10)
print({k:v for k, v in zip(keys, vals)})
```

{'key-0': 0, 'key-1': 1, 'key-2': 2, 'key-3': 3, 'key-4': 4, 'key-5': 5, 'key-6': 6, 'key-7': 7, 'key-8': 8, 'key-9': 9}

```
#6: write a try/except to catch the error generated from the
following code
# and print "I caught you"
# catch only the specific error, not all errors
try:
    5 / 0
except ZeroDivisionError:
    print("I caught you!")
```

I caught you!

Slide 7: Exercise 2: generators, loops, comprehension

```
#7: create a generator function that yields number from 0 to n, in
batches of
# lists of maximum 10 numbers at a time
def listgen(n):
    counter = 0
    numbers = list(range(n))
    while counter <= n // 10:
        yield numbers[(10 * counter) : (10 * (counter+1))]
        counter += 1

g = listgen(100)
print(next(g))
print(next(g))
print(next(g))
```

```
g = listgen(5)
print(next(g))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
[0, 1, 2, 3, 4]
```

```
#8: given this 3x4 matrix implemented as a list of lists, write a
list
# comprehension that creates its transpose
matrix = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
print([[row[i] for row in matrix] for i in range(4)])
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Slide 8: Exercise 2: functions

```
#9: create a function that takes all the integers a user enters
# and returns their sum. However if one of the values is 13, it
doesn't
# count towards the sum, nor do any values to its rights
def lucky_sum(*args):
    if 13 in args:
        return sum(args[:args.index(13)])
    return sum(args)

print(lucky_sum(1, 2, 3, 4))
print(lucky_sum(1, 13, 3, 4))
print(lucky_sum(13))
```

```
10
1
0
```

Slide 9: Exercise 2: loops, comprehension

```
#10: print only the EEG signal and events separately from this
nested list
# using for loop and list comprehension
two_channel_eeg_signal1 = [8, 9]
event1 = 1

two_channel_eeg_signal2 = [3, 3]
event2 = 2

two_channel_eeg_signal3 = [2, 3]
event3 = 2

nested_list = []
nested_list.append([two_channel_eeg_signal1, event1])
nested_list.append([two_channel_eeg_signal2, event2])
```

```
nested_list.append([two_channel_eeg_signal3, event3])
print("EEG signal: ", nested_list)

# (1) Using a for loop.
eeg_signals = []
events = []

for item in nested_list:
    #eeg_signals.append(item[0]) # Create a list of lists.
    eeg_signals.extend(item[0]) # Flattened.
    events.append(item[1])

print("EEG signals (using for loop):", eeg_signals)
print("Events (using for loop):", events)

# (2) Using list comprehension.
#eeg_signals = [item[0] for item in nested_list] # Create a list
of lists.
eeg_signals = [value for item in nested_list for value in item[0]]
# Flattened.
events = [item[1] for item in nested_list]

print("EEG signals (using list comprehension):", eeg_signals)
print("Events (using list comprehension):", events)
```

```
EEG signal:  [[[8, 9], 1], [[3, 3], 2], [[2, 3], 2]]
EEG signals (using for loop): [8, 9, 3, 3, 2, 3]
Events (using for loop): [1, 2, 2]
EEG signals (using list comprehension): [8, 9, 3, 3, 2, 3]
Events (using list comprehension): [1, 2, 2]
```

Slide 10: Exercise 3: sets, functions

```
#!/usr/bin/env python3

def symmetric_difference(set1, set2):
    """
    Compute the symmetric difference between two sets manually.

    Parameters
    ----------
    - set1 : set
             the first set
    - set2 : set
             the second set

    Returns
    -------
    set
        the symmetric difference between set1 and set2

    Examples
    --------
```

```
    >>> symmetric_difference({1, 2, 3, 4}, {3, 4, 5, 6})
    {1, 2, 5, 6}
    """

    # Elements in set1 but not in set2.
    diff1 = set1.difference(set2)

    # Elements in set2 but not in set1.
    diff2 = set2.difference(set1)

    # Symmetric difference is the union of the two differences.
    symmetric_diff = diff1.union(diff2)

    return symmetric_diff

# Example usage:
set_a = {1, 2, 3, 4}
set_b = {3, 4, 5, 6}

result_manual = symmetric_difference(set_a, set_b)
print("Symmetric difference (manual):", result_manual)
print("Symmetric difference (built-in):",
set_a.symmetric_difference(set_b))
```

```
Symmetric difference (manual): {1, 2, 5, 6}
Symmetric difference (built-in): {1, 2, 5, 6}
```

Lecture 11 (05/12): Object-oriented programming. Classes, inheritance and polymorphism. Modules and packages.

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/11/11-py_classes_modules.md

Slide 6:
In Python the constructor is ***def __init__*** and must take ***self*** as first input argument.
The actual list of all members is defined inside the constructor body.

Slide 8:
All variables and methods are public by default.

Slide 11:

You can call self whatever you want, here is called cls. Same thing as before but we didn't invoke the constructor.

Slide 13:
__add__ sums between two objects of the same type; it's an overload of the addition operator.

Slide 14:
__eq__ compares if two objects are the same.
__call__ makes a class act like a function f-string.

Slide 15:
__getitem__ returns something related to the item with the constructor.
__setitem__ takes the value to assign to the second argument

Slide 21:

```
class MyClass:
    a = 2
    __b = 4

obj = MyClass()
obj.a = 10
print(obj.a)

# but we can't do the same with b
#obj.__b = 10

print(dir(obj))
# prints all the methods inside

# but we can write this
obj._MyClass__b = 10
```

Slide 45:
Calling this module.py:

```
a = 2
def f():
    print("hello")

class MyClass():
    pass
```

We can write from terminal *import module*, then to access variables of this module you can write *module.a*, *module.f()*, *obj = module.MyClass()*.

Slide 49:
Writing *module.__name__* it prints the name.
In module.py:

```
a = 2
def f():
    print("hello")

class MyClass():
    pass

if __name__ == '__main__':
    f()
    a = 3
    obj = MyClass()
```

Writing *python module.py* it prints hello, while doing *import module* this part won't be executed.

Slide 50:
dir(math) prints all the functions available in the module math.
dir() prints everything defined in the module.

Laboratory 10 (07/12):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/10/10-py_classes_modules.md

Slide 2: Exercise 1: generators for the solution of ODEs
Solving the differential equation u'=-sin u by applying the explicit Euler method results in the recursion u_{n+1}=u_n - h sin (u_n).
Write a generator that computes the solution values u_n for a given initial value u_0=1 and a given value of the time step h=0.1.
Implement a function decorator *step_counter* that counts how many time steps have been performed

```
#!/usr/bin/env python3
from math import sin
def step_counter(generator_func):
    def wrapper(*args, **kwargs):
```

```
        gen = generator_func(*args, **kwargs)
        wrapper.steps = 0
        while True:
            wrapper.steps += 1
            print(f"Step number: {wrapper.steps}")
            yield next(gen)
            print()
    return wrapper
@step_counter
def explicit_euler(u0, h):
    u = u0
    while True:
        u = u - h * sin(u)
        yield u
gen = explicit_euler(u0=1.0, h=0.1)
for _ in range(10):
    print(next(gen))
```

Yield is used **to create sequences of values without memorizing the whole sequence**.
When *next()* is written, the execution begins, not from the first lines as in functions.

Slide 3-4-5: Exercise 2: Polynomial class
Implement a class called Polynomial that represents polynomials and has the following features:

1. Constructor: the class should have a custom constructor that takes variable coefficients as arguments. The coefficients should be provided in increasing order of degree.

```
from time import time

class Polynomial:
    def __init__(self, *coefficients):
        self.coefficients = coefficients
```

2. String representation: implement the __repr__ method to provide a string representation of the polynomial. The string should display the polynomial in a human-readable form. For example, for the polynomial with coefficients [1,2,3] the string representation should be 1+2x+3x^2

```
    def __repr__(self):
        terms = [f"{a}x^{i}" if i > 1 else f"x" if i == 1 else
str(a) for i, a in enumerate(self.coefficients)]
        return " + ".join(terms)
```

3. Addition and multiplication: implement the __add__ and __mul__ methods to allow addition and multiplication of polynomials. The methods should return a new polynomial.

```
    def __add__(self, other):
        common_len = min(len(self.coefficients),
len(other.coefficients))
        new_coefficients = [a + b for a, b in
zip(self.coefficients, other.coefficients)]
        remaining_coefficients = (
            self.coefficients[common_len:] +
other.coefficients[common_len:]
        )
        new_coefficients += remaining_coefficients
        return Polynomial(*new_coefficients)
```

```
    def __mul__(self, other):
        new_coefficients = [0] * (len(self.coefficients) +
len(other.coefficients) - 1)
        for i, a in enumerate(self.coefficients):
            for j, b in enumerate(other.coefficients):
                new_coefficients[i + j] += a * b
        return Polynomial(*new_coefficients)
```

4. Class method to create from string: implement a @classmethod called *from_string* that creates a Polynomial object from a string representation. Assume that the input string will be a polynomial in the form of a + bx + … + cx^(n-1) + dx^n

```
    @classmethod
    def from_string(cls, poly_str):
        terms = poly_str.split("+")
        coefficients = [int(term.split("x")[0]) for term in terms]
        return cls(*coefficients)
```

5. The base class Polynomial should be extended by two subclasses:
   - StandardPolynomialEvaluator implements the standard polynomial evaluation method: P(x) = a_0 + a_1 x + a_2 x^2 + … + a_n x^n

```
class StandardPolynomialEvaluator(Polynomial):
    def evaluate(self, x_list):
        return [sum(a * (x ** i) for i, a in
enumerate(self.coefficients)) for x in x_list]
```

   - HornerPolynomialEvaluator implements Horner's rule for polynomial evaluation: P(x) = a_0 + x (a_1 + x (a_2 + … + x (a_{n-1} + x a_n …))

```
class HornerPolynomialEvaluator(Polynomial):
    def evaluate(self, x_list):
```

```
        y_list = []
        for x in x_list:
            result = 0
            for a in reversed(self.coefficients):
                result = result * x + a
            y_list.append(result)
        return y_list
```

6. Implement a *measure_time* decorator which measures the time taken by a function to execute

```
def measure_time(func):
    def wrapper(*args, **kwargs):
        start_time = time()
        result = func(*args, **kwargs)
        end_time = time()
        print(f"Time taken by {func.__name__}: {end_time -
start_time} seconds.\n")
        return result
    return wrapper

@measure_time
def evaluate_polynomial(poly_evaluator, x_list):
    return poly_evaluator.evaluate(x_list)
```

7. Instantiate objects of both StandardPolynomialEvaluator and HornerPolynomialEvaluator with the same set of coefficients
8. Apply the *measure_time* decorator to a function that takes a *PolynomialEvaluator* object and evaluates it at a given list of points
9. Evaluate the polynomial at the same 1000 points using both methods and compare the results. Raise an assertion error if the results do not match
10. Use the decorated function to evaluate the polynomial using both the standard method and Horner's rule and observe the logged results and execution times

```
poly1 = Polynomial(1, 2, 3)
poly2 = Polynomial(2, 3, 4)
print(poly1 + poly2)
print(poly1 * poly2, '\n')

poly3 = Polynomial.from_string("1 + 2x + 3x^2")
print(poly3, '\n')

standard_evaluator = StandardPolynomialEvaluator(1, 2, 3)
horner_evaluator = HornerPolynomialEvaluator(1, 2, 3)
```

```
print("Standard evaluation")
result_standard = evaluate_polynomial(standard_evaluator,
range(1000))
print("Horner evaluation")
result_horner = evaluate_polynomial(horner_evaluator, range(1000))

assert result_standard == result_horner
```

Slide 6: Exercise 3:
Refactor the existing data processing code provided into a modular package with multiple modules, functions, classes.

1. Refactoring: refactor the code into a modular package *dataprocessor* with the following modules:

- *__init__.py* : entry point for the package, import necessary functions, classes and data. Implement *__all__*

```
from .operations import process_data, additional_operation
from .data_analysis import DataAnalyzer
import pandas as pd

__all__ = ['process_data', 'additional_operation', 'DataAnalyzer',
'pd']
```

- *operations.py* : contains functions for data processing and analysis

```
import pandas as pd

def process_data(df, column1, column2, result_column):
    """
    Perform data processing by adding values from column1 and
column2,
    and store the result in the specified result_column.

    Parameters:
    - df: pd.DataFrame
        Input DataFrame containing data.
    - column1: str
        Name of the first column.
    - column2: str
        Name of the second column.
    - result_column: str
        Name of the column to store the result.

    Returns:
    - pd.DataFrame
        DataFrame with the processed data.
    """
    df[result_column] = df[column1] + df[column2]
    return df
```

```
def additional_operation(df, column1, column2, additional_column1,
additional_column2):
    """
    Perform additional operations on the DataFrame by multiplying
column1 and column2
    and storing the result in additional_column1, and dividing
column1 by column2
    and storing the result in additional_column2.

    Parameters:
    - df: pd.DataFrame
        Input DataFrame containing data.
    - column1: str
        Name of the first column.
    - column2: str
        Name of the second column.
    - additional_column1: str
        Name of the column to store the multiplication result.
    - additional_column2: str
        Name of the column to store the division result.

    Returns:
    - pd.DataFrame
        DataFrame with the additional operations applied.
    """
    df[additional_column1] = df[column1] * df[column2]
    df[additional_column2] = df[column1] / df[column2]
    return df
```

- *data_analysis.py* : introduce a class DataAnalyzer that encapsulates data processing and analysis functionalities.

```
class DataAnalyzer:
    """
    Class for analyzing data in a DataFrame.

    Parameters:
    - result_column: str
        Name of the column containing the result values.
    """
    def __init__(self, result_column):
        self.result_column = result_column

    def analyze_data(self, df):
        """
        Analyze the data in the DataFrame by calculating the mean
        and maximum values of the specified result_column.

        Parameters:
        - df: pd.DataFrame
            Input DataFrame containing data.
```

```
        Returns:
        - tuple
            A tuple containing mean and maximum values.
        """
        mean_val = df[self.result_column].mean()
        max_val = df[self.result_column].max()
        return mean_val, max_val
```

2. Documentation: provide docstrings for functions and classes. Explain the purpose and usage of each function and configuration option. It's the thing between """ """
3. Test cases: create a test main.py script to demonstrate the usage of the package

```
from dataprocessor import process_data, additional_operation,
DataAnalyzer, pd

# Configuration settings.
data_config = {
    'column1': 'A',
    'column2': 'B',
    'result_column': 'Result',
    'additional_column1': 'Additional1',
    'additional_column2': 'Additional2',
}

data = pd.DataFrame({
    data_config['column1']: [1, 2, 3, 4],
    data_config['column2']: [5, 6, 7, 8],
})

def main():
    # Process data.
    processed_data = process_data(data, data_config['column1'],
data_config['column2'], data_config['result_column'])

    # Analyze data.
    analyzer = DataAnalyzer(data_config['result_column'])
    mean, max_val = analyzer.analyze_data(processed_data)
    print(f"Mean: {mean}, Max: {max_val}")

    # Additional operation.
    additional_data = additional_operation(processed_data,
data_config['column1'], data_config['column2'],

data_config['additional_column1'],
data_config['additional_column2'])

    print("Final processed data:")
    print(additional_data)
```

```
if __name__ == "__main__":
    main()
```

Lecture 12 (12/12): Introduction to NumPy and SciPy for scientific computing. Data visualization. Introduction to pandas for data analysis.

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/12/12-py_scicomp.md

Slide 21:
Fancy indexing says the indices you want to extract.
The boolean contains true for the position you want to extract and false for the one you don't want.

Slide 32:
B = A is like the reference in C++, you better create a deep copy with *C = np.copy(A)*

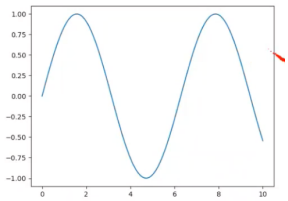
Slide 34:
enumerate returns a tuple where the first element is the index of the element you want to access and the second element is the value itself.

Slide 36:
The function *vectorize* of NumPy can help you vectorizing a function because it's not always easy as the Heaviside function.

Slide 37:
.any() returns true if there is at least one element that satisfies the condition
.all() returns true if all of them satisfy the condition.

Slide 51:
Don't overwrite *lambda* as the teacher erroneously did, because it already exists.

Slide 64:
matplotlib works well with numpy so generally you import both libraries.
After writing *plt.plot(x,y)* you have to write *plt.show()* if you actually want to see the plot.
The one in this slide (with just those commands above and not the labels) is



Slide 79:
.loc extracts a new dataframe, while *.at* returns the value contained in the specific cell in the dataframe. For these you specify the labels, with position (*.iloc .iat*) you specify the position.

Laboratory 11 (14/12):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/11/11-py_scicomp.md

Slide 2: Exercise 1: NumPy

```
import numpy as np
# 1: Array creation and manipulation
# Create a NumPy array of shape 5x5 filled with random int in
[1,10]
A = np.random.randint(1,11, size=(5,5))
print(A)
# Extract second row, third column and the diagonal elements
second_row = A[1, :]
third_column= A[:, 2]
diagonal = np.diag(A)
# Reshape it into a 1D array of shape 1x25
A.reshape((1,25))

# 2: Linear algebra operations
# Generate two 3x3 matrices with random integers from 1 to 10
B = np.random.randint(1,11, size=(3,3))
C = np.random.randint(1,11, size=(3,3))

# Perform element-wise and matrix-matrix multiplication
print(f"Element wise product = {B * C}")
print(f"Matrix product = {B @ C}")
# or np.dot(B,C)
# Create a 3x3 matrix with random values, compute its inverse and
det
D = np.random.rand(3,3)
print(f"Inverse: {np.linalg.inv(D)}")
```

```
print(f"Determinant: {np.linalg.det(D)}")

# 3: Statistical analysis
# Generate a 1D NumPy array with 20 random integers between 1 and
100
random_array = np.random.randint(1,101,20)
# Calculate mean, median, standard deviation and variance
print(f"Mean: {np.mean(random_array)}")
print(f"Median: {np.median(random_array)}")
print(f"Standard deviation: {np.std(random_array)}")
print(f"Variance: {np.var(random_array)}")
```

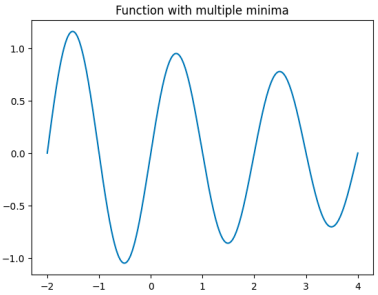
Slide 3: Exercise 2: SciPy. Ex 1: Solving a linear system of equations

```
from scipy.sparse import diags
import numpy as np

# 1: Solving a linear system of equations
# Define a 100x100 sparse tridiagonal matrix A, with 2 over the
# main diagonal and -1 over the first lower and upper diagonals
n = 100
diagonals = [2 * np.ones(n), -1 * np.ones(n-1), -1 * np.ones(n+1)]
A = diags(diagonals, [0, 1, -1], format='csr')
# [0, 1, -1] is the offsets i.e. the diagonals to set
# CSR is the format for the sparse matrix
from scipy.sparse.linalg import spsolve
# Let b = Ax_{ex} where x_{ex}=[1,..,1]^T in R^100
x_ex = np.ones(n)
b = A @ x_ex

# Solve the linear system Ax=b
x = spsolve(A, b)
from scipy.linalg import norm
# Compute the residual ||b-Ax|| in norm 1, 2 and infinity
residual = b - A @ x

norm1_res = norm(residual, 1)
norm2_res = norm(residual, 2)
norm_inf_res = norm(residual, np.inf)

# Compute the error ||x-x_{ex}|| in norm 1, 2 and infinity
error = x - x_ex

norm1_err = norm(error, 1)
norm2_err = norm(error, 2)
norm_inf_err = norm(error, np.inf)
```

Slide 3: Exercise 2: SciPy. Ex 2: Function optimization

```
# 2: Function optimization
import numpy as np
```

```python
# Consider the function f(x)=sin(πx)exp(-x/10) over [-2,4]
def f(x):
    return np.sin(np.pi * x) * np.exp(-x /10)
# Plot the function using Matplotlib to visually identify
potential minima
import matplotlib.pyplot as plt
x = np.linspace(-2, 4, 1000)
y = f(x)
plt.plot(x, y)
plt.title("Function with multiple
minima")
plt.show()
```




```python
# Use scipy.optimize.minimize with
different initial guesses to find these
minima
from scipy.optimize import minimize
result1 = minimize(f, x0=-0.5)
result2 = minimize(f, x0=1.5)
result3 = minimize(f, x0=3.5)

print(f"x1 = {result1.x[0]}, f(x1) = {result1.jac[0]}")
print(f"x2 = {result2.x[0]}, f(x2) = {result2.jac[0]}")
print(f"x3 = {result3.x[0]}, f(x3) = {result3.jac[0]}")
```

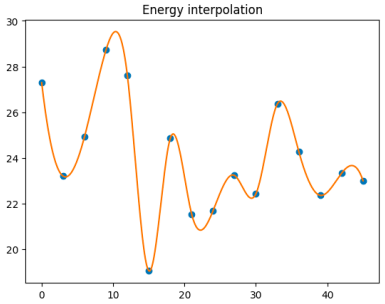
```
x1 = -0.5101287856932691, f(x1) = -8.344650268554688e-07
x2 = 1.4898712445912434, f(x2) = -4.172325134277344e-07
x3 = 3.4898712651457853, f(x3) = -2.086162567138672e-07
```

Slide 4: Exercise 2: SciPy. Ex 3: Data interpolation and integration

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from scipy.integrate import quad
# 3: Data interpolation and integration
# An electric vehicle charging station erogates the following
series
# of energy measurements over time
time = np.arange(0, 46, 3)
energy = np.array([27.29, 23.20, 24.93, 28.72, 27.60, 19.06,
24.85, 21.54, 21.69, 23.23, 22.43, 26.36, 24.28, 22.36, 23.33,
23.00])

# Use SciPy to build a cubic interpolator of these data points
f_interpolator = interp1d(time, energy, kind='cubic')
# It returns a callable object
# So f_interpolar(25) calculates it in 25

# Evaluate the interpolator over 1000 equispaced nodes
```

```
# between 0 and 45 and plot values obtained
time_interp = np.linspace(0, 45, 1000)
# right endpoint is included
energy_interp = f_interpolator(time_interp)

plt.plot(time, energy, 'o', time_interp,
energy_interp, '-')
plt.title("Energy interpolation")
plt.show()

# Integrate the interpolant over (0,45)
print(quad(f_interpolator, 0, 45)[0])
```

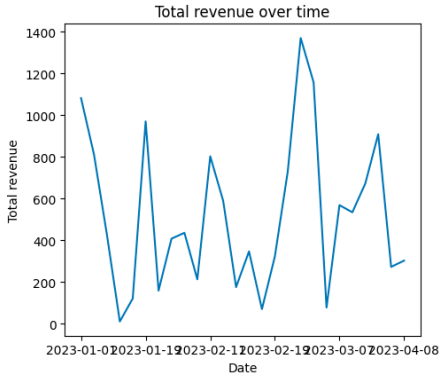
```
1074.8394630292078
```

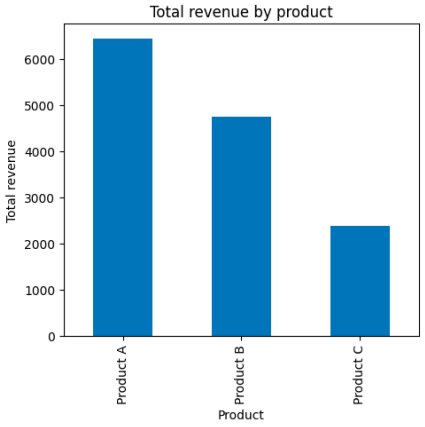


Slide 5: Exercise 3: pandas. Ex 1: DataFrame operations and visualization

```
import pandas as pd
# 1: DataFrame operations and visualization
# Import the sales_data.txt dataset as a pandas DataFrame
sales_data = pd.read_csv('sales_data.csv')

# Extract data from the 'South' region
filtered_sales_data = sales_data[sales_data['Region'] == 'South']
# Sort them by descending 'Quantity'
sorted_sales_data =
filtered_sales_data.sort_values(by=['Quantity'], ascending=False)
# Add a column 'Total revenue' = 'Quantity' x 'Price'
sorted_sales_data['Total revenue'] = sorted_sales_data['Quantity']
* sorted_sales_data['Price']
#print(sorted_sales_data)

# Visualize trends of 'Total revenue' by 'Data' (line plot)
# and by 'Product' (bar plot)
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sorted_sales_data.groupby('Date')['Total
revenue'].sum().plot(kind='line')
plt.title('Total revenue over time')
plt.xlabel('Date')
plt.ylabel('Total revenue')

plt.subplot(1, 2, 2)
sorted_sales_data.groupby('Product')['Total
revenue'].sum().plot(kind='bar')
plt.title('Total revenue by product')
plt.xlabel('Product')
plt.ylabel('Total revenue')
plt.tight_layout()
```

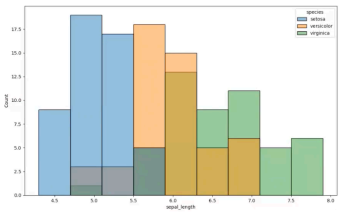
```
plt.show()
```





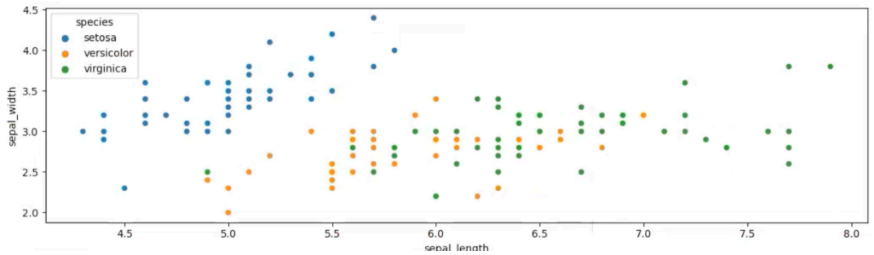
Slide 5: Exercise 3: pandas. Ex 2: Exploratory data analysis with the iris set

```
import seaborn as sns
import matplotlib.pyplot as plt
# 2: Exploratory data analysis with the iris dataset
# Load the iris dataset from seaborn
iris = sns.load_dataset('iris')
# Returned as a pandas DataFrame

# Group the data by 'species' and compute summary statistics
# for sepal_length and sepal_width
grouped_iris = iris.groupby('species').agg({'sepal_length':
['mean', 'std'],
                                            'sepal_width':
['mean', 'std']})
```



```
# Use seaborn to plot the histogram of
the sepal length distribution for each
species
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(data=iris,
x='sepal_length', hue='species',
element='step', stat='density',
common_norm=False)
plt.title('Sepal length distribution by
species')

#  Use seaborn to generate a scatter plot of sepal width vs sepal
length
plt.subplot(1, 2, 2)
```

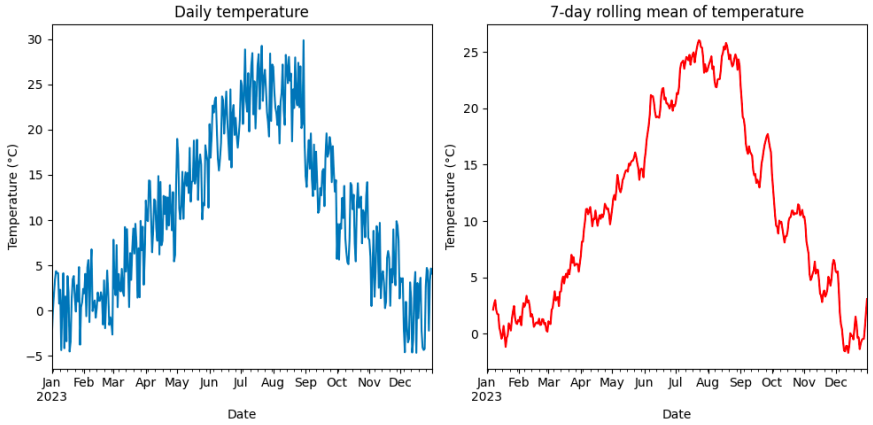
```
sns.scatterplot(data=iris, x='sepal_length', y='sepal_width',
hue='species')
plt.title('Sepal width vs. length')
plt.show()
```



Slide 6: Exercise 3: pandas. Ex 3: Time series analysis with real data

```
import pandas as pd
import matplotlib.pyplot as plt
# 3: Time series analysis with real data
# Import the weather_data.csv dataset
weather_data = pd.read_csv('weather_data.csv',
parse_dates=['Date'], index_col='Date')

# Resample the dataset to compute monthly averages
monthly_avg = weather_data.resample('M').mean()
# M stands for monthly

# Compute a 7-day rolling mean
rolling_mean = weather_data.rolling(window=7).mean()

# Visualize the original data and the rolling mean using line
plots
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
weather_data['Temperature'].plot(kind='line', label='Original')
plt.title('Daily temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')

plt.subplot(1, 2, 2)
rolling_mean['Temperature'].plot(kind='line', label='7-day rolling
mean', color='red')
plt.title('7-day rolling mean of temperature')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.tight_layout()
plt.show()
```

11. Sara Serafino
23 dicembre 2023 alle ore 16:25:30
compilation of C++ and Python



Lecture 13 (19/12): Integrating C++ and Python codes

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/lectures/13/13-py_c++.md

Slide 19:
m is just a placeholder
"add" is the Python function, it points to add (second argument).

Slide 21:
`python3 -m pybind11 –includes` tells us the compile-time flags to be passed to the compiler: `-I/usr/include/python3.10 -I/usr/lib/python3/dist-packages/pybind11/include`. You can do a shortcut of it writing
`g++ -03 -fPIC -shared $(python3 -m pybind11 –include) matrix_multiplication.cpp -o matrix_ops.so` where $ takes the output and copies it in place.
Or you can do it manually with `-I/path/to/pybind11`.

Slide 57:
Compilation is slightly different now:

```
mkdir build
cd build
cmake ..
```

```
make
cd ../
PYTHONPATH=build/ python3 test.py
```

Interactive part with examples:
All the examples come with a CMakeLists.txt, example 6 comes also with setup.py that can be runned with `python setup.py build_ext –inplace` or with `python setup.py install –user` or with `pip install –user .`

Laboratory 12 (21/12):

https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exercises/12/12-py_c++.md

Slide 2: Exercise 1: binding classes and magic methods
Provide Python bindings using pybind11 for the code provided as the solution to ex 1 from session 3.
Bind the *DataProcessor* class and its member functions. using a lambda function, expose a constructor taking a Python list as an input, to be converted to a *std::vector* and invoking the actual constructor.
Provide Python bindings for the addition (*__add__*), the read (*__getiitem__*) and write (*__setitem__*) access, and the output stream (*__str__*) operators.
Package the Python module with the compiled C++ library using *setuptools*.
Write a Python script to replicate the functionalities implemented in the *main.cpp* file.

data_processor.hpp is exactly as before.
data_processor.cpp is as before plus the Python interface:

```cpp
#include "data_processor.hpp"
#include <cmath>
#include <sstream>
#include <pybind11/pybind11.h> // this is new
#include <pybind11/stl.h> // this is new

// This is the same C++ interface as before

unsigned int DataProcessor::n_instances = 0;

DataProcessor::DataProcessor(const double *input_data,
                             const unsigned int &input_size)
```

```
    : size(input_size), data(new double[size]) {
  ++n_instances;
  for (unsigned int i = 0; i < size; ++i) {
    data[i] = input_data[i];
  }
}

DataProcessor::DataProcessor(const DataProcessor &other)
    : size(other.size), data(new double[size]) {
  ++n_instances;
  for (unsigned int i = 0; i < size; ++i) {
    data[i] = other[i];
  }
}

DataProcessor &DataProcessor::operator=(const DataProcessor
&other) {
  if (this != &other) {
    delete[] data;
    size = other.size;
    data = new double[size];
    for (unsigned int i = 0; i < size; ++i) {
      data[i] = other[i];
    }
  }
  return *this;
}

DataProcessor DataProcessor::operator+(const DataProcessor &other)
const {
  assert(size == other.size); // Check if sizes match.
  DataProcessor result(data, size); // Copy the current object.
  for (unsigned int i = 0; i < size; ++i) {
    result[i] += other[i];
  }
  return result;
}

double DataProcessor::compute_std_dev() const {
  const double mean = compute_mean();

  double sum_squared_diff = 0.0;
  for (unsigned int i = 0; i < size; ++i) {
    sum_squared_diff += (data[i] - mean) * (data[i] - mean);
  }

  return std::sqrt(sum_squared_diff / size);
}

std::ostream &operator<<(std::ostream &os, const DataProcessor
&dp) {
  for (unsigned int i = 0; i < dp.size; ++i) {
```

```
    os << dp[i];
    if (i < dp.size - 1) {
      os << ", ";
    }
  }
  return os;
}

double compute_correlation(const DataProcessor &dp1, const
DataProcessor &dp2) {
  assert(dp1.n_elements() == dp2.n_elements()); // Check if sizes
match.
  const unsigned int n = dp1.n_elements();
  // Calculate the means of both datasets.
  const double mean1 = dp1.compute_mean();
  const double mean2 = dp2.compute_mean();
  // Calculate the sums of the cross-products of differences.
  double sum_cross_products = 0.0;
  double sum_diff1_squared = 0.0;
  double sum_diff2_squared = 0.0;
  for (unsigned int i = 0; i < n; ++i) {
    const double diff1 = dp1[i] - mean1;
    const double diff2 = dp2[i] - mean2;
    sum_cross_products += diff1 * diff2;
    sum_diff1_squared += diff1 * diff1;
    sum_diff2_squared += diff2 * diff2;
  }

  // Calculate the Pearson correlation coefficient.
  const double correlation =
      sum_cross_products /
      (std::sqrt(sum_diff1_squared) *
std::sqrt(sum_diff2_squared));

  return correlation;
}

// -----------------
// Python interface
// -----------------

namespace py = pybind11;
PYBIND11_MODULE(data_processor, m) {
  m.doc() = "pybind11 example plugin";

  py::class_<DataProcessor>(m, "DataProcessor")
      .def(py::init([](const std::vector<double> &a) {
        return new DataProcessor(a.data(), a.size());
      }))
      .def("__add__", &DataProcessor::operator+)
      .def("n_elements", &DataProcessor::n_elements)
      .def("min", &DataProcessor::min)
```

12. Sara Serafino
24 dicembre 2023 alle ore 12:57:59
Here __getitem__ is a const method with a pointer to a member function of the DataProcessor class that takes an unsigned int const reference and returns a const double

13. Sara Serafino
24 dicembre 2023 alle ore 13:01:29
Here for __setitem__ you need something more: provide as input DataProcess and the index. Python syntax here has only two arguments so the value must be provided with a lambda function.

```
        .def("max", &DataProcessor::max)
        .def("compute_mean", &DataProcessor::compute_mean)
        .def("compute_std_dev", &DataProcessor::compute_std_dev)
        .def_static("get_n_instances",
&DataProcessor::get_n_instances)
12      .def("__getitem__",
                (const double &(DataProcessor::*)(const unsigned int &)
const) &
                    DataProcessor::operator[])
13      .def("__setitem__", [](DataProcessor &dp, unsigned int
index,
                               double value) { dp[index] = value; })
        .def("__repr__", [](const DataProcessor &dp) {
          std::ostringstream ss;
          ss << dp;
          return ss.str();
        });

  m.def("compute_correlation", &compute_correlation);
}
```

main.py:

```
import data_processor as dp

input1 = [2.43, -0.86, 7.19, 4.57, 1.68, 9.32, 5.75]
input2 = [0.73, -0.45, 0.12, 0.88, -0.67, 0.34, -0.92]
dp1 = dp.DataProcessor(input1)
dp2 = dp.DataProcessor(input2)
print(f"dp1: {dp1}")
print(f"dp2: {dp2}")
print(f"Number of instances: {dp1.get_n_instances()}")
print(f"Minimum value of dp1: {dp1.min()}")
print(f"Maximum value of dp1: {dp1.max()}")
print(f"Mean of dp1: {dp1.compute_mean()}")
print(f"Standard deviation of dp1: {dp1.compute_std_dev()}")
dp3 = dp1 + dp2
print(f"dp3: {dp3}")
print(f"dp3[3]: {dp3[3]}")
dp3[3] = 0.0
print(f"dp3[3]: {dp3[3]}")
print(f"Correlation between dp1 and dp2:
{dp.compute_correlation(dp1, dp2)}")
del dp2
print(f"Number of instances: {dp1.get_n_instances()}”)
```

Slide 3: Exercise 2: binding class templates and exceptions
Provide Python bindings using pybind11 for the code provided as the solution to ex 3 from session 5.

Modify the *NewtonSolver::solve()* method in order to throw a *std::runtime_error* exception instead of returning *NaN* when failed to converge to a root.
Bind the *NewtonClass* class and its member functions, providing explicit instantiations for *double* and *std::complex<double>* numbers. The Python interface should provide consistent default arguments.
Python bindings should be implemented in a separate *newton_py.cpp* file. Translate the *std::runtime_error* C++ exception to a *RuntimeError* Python exception.
Use CMake to setup the build process.
Write a Python script to replicate the functionalities implemented in the *main.cpp* file.
Verify that exception handling works properly.

newton.hpp and newton.cpp are exactly as before.
newton_py.cpp is its Python interface:

```
#include "newton.hpp"
#include <pybind11/complex.h>
#include <pybind11/functional.h>
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>

namespace py = pybind11;

PYBIND11_MODULE(newton_solver, m) {
  py::class_<NewtonSolver<double>>(m, "NewtonSolverDouble")
      .def(py::init<const std::function<double(const double &)> &,
                    const std::function<double(const double &)> &,
                    const double &, const double &, const unsigned
int &>(),
           py::arg("f"), py::arg("df"), py::arg("x0"),
           py::arg("tolerance") = 1e-12, py::arg("max_iterations")
= 100)
      .def("solve", &NewtonSolver<double>::solve);

  py::class_<NewtonSolver<std::complex<double>>>(m,
"NewtonSolverComplex")
      .def(py::init<const std::function<std::complex<double>(
                        const std::complex<double> &)> &,
                    const std::function<std::complex<double>(
                        const std::complex<double> &)> &,
                    const std::complex<double> &, const double &,
                    const unsigned int &>(),
           py::arg("f"), py::arg("df"), py::arg("x0"),
```

```
            py::arg("tolerance") = 1e-12, py::arg("max_iterations")
= 100)
        .def("solve", &NewtonSolver<std::complex<double>>::solve);

  // Translate C++ exception to Python exception.
  py::register_exception<std::runtime_error>(m, "RuntimeError");
}
```

main.py:

```
import newton_solver
import cmath

# Function f(x) = x^2 - 1 = 0 and its derivative.
def function(x):
    return x * x - 1

def derivative(x):
    return 2 * x

# Create a NewtonSolver instance for real functions and find the
root.
x0 = 0.5
solver_real = newton_solver.NewtonSolverDouble(function,
derivative, x0, 0, 10)

try:
    root = solver_real.solve()
    print(f"Approximate root: {root}")
except RuntimeError as e:
    print("Error: ", str(e))

# Create a NewtonSolver instance for complex functions and find
the root.
x0 = complex(0.5, 0.5)
solver_complex = newton_solver.NewtonSolverComplex(function,
derivative, x0)

try:
    root = solver_complex.solve()
    print(f"Approximate root: {root}")
except RuntimeError as e:
    print("Error: ", str(e))
```

Slide 4: Exercise 3: binding with external libraries
Implement C++ functions using the Eigen library to perform matrix-matrix multiplication and matrix inversion.
Provide Python bindings using pybind11 for the code implemented.
Use *CMake* and *setuptools* to setup the build process.

Write a Python script to test the performance of the Eigen-based operations. Implement a *log_execution_time* decorator to print the execution time of a function.
Compare the execution time of these operations to equivalent operations in NumPy (e.g., *numpy.matmul* for multiplication and *numpy.linalg.inv* for inversion). Use a large matrix (e.g., 1000 x 1000) of random integers between 0 and 1000 for the test.

eigen_ops.hpp:

```
#ifndef EIGEN_OPS_HPP__
#define EIGEN_OPS_HPP__

#include <Eigen/Dense>

Eigen::MatrixXd matrix_multiply(const Eigen::MatrixXd& m1, const
Eigen::MatrixXd& m2);
Eigen::MatrixXd matrix_invert(const Eigen::MatrixXd& m);

#endif /* EIGEN_OPS_HPP__ */
```

eigen_ops.cpp:

```
#include "eigen_ops.hpp"
#include <pybind11/eigen.h>
#include <pybind11/pybind11.h>

using Eigen::MatrixXd;

MatrixXd matrix_multiply(const MatrixXd &m1, const MatrixXd &m2) {
  return m1 * m2;
}

MatrixXd matrix_invert(const MatrixXd &m) { return m.inverse(); }

// -----------------
// Python interface
// -----------------
namespace py = pybind11;

PYBIND11_MODULE(eigen_ops, m) {
  m.def("matrix_multiply", &matrix_multiply);
  m.def("matrix_invert", &matrix_invert);
}
```

main.py:

```
import numpy as np
import time
import eigen_ops
```

```python
def log_execution_time(func):
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"{func.__name__} executed in {end - start}
seconds.")
        return result
    return wrapper

@log_execution_time
def test_eigen_product(A, B):
    return eigen_ops.matrix_multiply(A, B)

@log_execution_time
def test_eigen_inversion(A):
    return eigen_ops.matrix_invert(A)

@log_execution_time
def test_numpy_product(A, B):
    return np.matmul(A, B)

@log_execution_time
def test_numpy_inversion(A):
    return np.linalg.inv(A)

n = 1000
A = np.random.randint(0, 1001, size=(n, n))
B = np.random.randint(0, 1001, size=(n, n))

test_eigen_product(A, B)
test_eigen_inversion(A)
test_numpy_product(A, B)
test_numpy_inversion(A)
```

Slide 5: Exercise 4: code obfuscation
Without running it, what's the output resulting from the execution of the code contained in *wish.cpp*?

The output (runned with `g++ -std=c++17 wish.cpp -o wish && ./wish`) is:

Exam simulation (09/01):

Part 1: Open-ended questions: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-09_simulation/part1.md

My answers:

```
# Part 1: Open-ended questions

## Question 1
What is a preprocessor and what is it used for?<br>
Given a C++ source file `code.cpp`, which shell command can be used to export the output of the preprocessor to a file `code_preprocessed.cpp`?

## Answer
A preprocessor is a program or set of directives which is part of the build process and performs preliminary operations on the source code before passing it to the compiler. It has the role of handling directives (like including header files with `#include`, conditional compilation `#ifdef #ifndef #else #endif`, compiler-specific directives `#pragma`) and macros (defining them with `#define`). It's used for code reusability and organization.<br>
The shell command to export the output of the preprocessor is:
```bash
g++ code.cpp -o code_preprocessed.cpp
```
```

Question 2
Describe what is meant by *undefined behavior* in C++.
Give examples of common programming mistake in C++ that can lead to undefined behavior.

Answer
It means that there is a situation for which the language doesn't prescribe a specific outcome, generating any result.
Some examples can be:
```bash
// Trying to access memory beyond the allocated one
std::vector<int> v;
v.resize(2, 5);
std::cout << v[6];

// Using uninitialized variables
int x;
int y = x + 2;

// Modifying a const object
const int pi = 3.1415926;
pi = 3.14

// Type mismatch
int x = 5;
std::string saluto = "ciao";
double sum = x + saluto;
```

Question 3
Compare and contrast `std::unique_ptr`, `std::shared_ptr`, and `std::weak_ptr`.
When should each be used? Provide some prototypical examples of use.

Answer
Smart pointers control the lifespan of the resource they point to. `std::unique_ptr` means that the owned resource gets destroyed when the pointer goes out of scope; while for `std::shared_ptr` it happens when the last pointer owning that resource is destroyed. In fact in this last one, you have several objects that refer to the same resource. `std::weak_ptr` is a non-owning (thus weak) pointer to a shared resource; to access the referenced object it must be converted to a shared pointer; it's used to safely observe the shared object without affecting the ownership.
An example of the use of `std::unique_ptr` is:
```bash
class Polygon {};
```

```
class Triangle : public Polygon {};
class Square : public Polygon {};

int main () {
  std::unique_ptr<Polygon> polygon;
  std::string t;
  std::cin >> t;
  switch (t[0]) {
    case "T": // as in Triangle
      polygon = std::make_unique<Triangle>(...);
      break;
    case "S": // as in Square
      polygon = std::make_unique<Square>(...);
      break;
  }
  // Create another unique_ptr for swapping
  std::unique_ptr<Polygon> polygon2 =
std::make_unique<Triangle>();
  // You can swap ownership
  std::swap(polygon, polygon2);
return 0;
}
```

An example of use of `std::shared_ptr` is (part of my StatisticsModule of Homework2):

```bash
class CSVHandler {
public:
  // Declare StatOp as a friend class of CSVHandler in order to
access its private members
  friend class StatOp;
protected:
  // Shared pointer between the two classes
  const std::shared_ptr<CSVHandler> CSVfile;
}

class StatOp {
public:
    // Constructor
    StatOp(const std::shared_ptr<CSVHandler> CSVfile);
}

int main () {
    // Create shared pointer
    const std::shared_ptr<CSVHandler> CSVfile =
std::make_shared<CSVHandler>(input_file_path);
    return 0;
}
```

An example of use of `std::weak_ptr` is:

```bash
std::shared_ptr<int> sptr = std::make_shared<int>(5);
std::weak_ptr<int> weak = sptr; // Get pointer to new data without
taking ownership
if (sptr = weak.lock()) {
  // Use the locked sptr to access the shared data
} else {
  // The shared data is out of scope
}
```

Question 4
What's the difference between *Aggregation* and *Inheritance* in
object-oriented programming?
Provide some conceptual example for each of the two types of
relationship.

Answer
Aggregation is a "has-a" relationship where one class contains the
other as a part, but the contained object can exist independently;
while inheritance is a "is-a" relationship where the derived class
inherits properties and behaviours from the base class, therefore
it can't exist without the base class and it's stronger.
An example of aggregation is the University class that may
aggregate Department classes:
```bash
class Department {
public:
  Department(std::string name) : name(name) {}
private:
  std::string name;
};
class University {
public:
  void addDepartment(const Department& department) {
    departments.push_back(department);
  }
private:
  std::vector<Department> departments;
};
```
An example of inheritance is (part of my Homework1):
```bash
class SparseMatrix { // abstract class
public:
    // Constructor
    SparseMatrix(std::vector<double>& values, std::vector<unsigned
int>& columns);

    // Define the common method of COO and CSR as virtual
```

```
    virtual unsigned int get_num_rows() const = 0;
protected:
// the vector values contains all the nonzero values
std::vector<double> values;
// the vector columns contains their corresponding columns indices
std::vector<unsigned int> columns;
};

class SparseMatrixCOO : public SparseMatrix {
public:
    // Constructor
    SparseMatrixCOO(std::vector<double>& values,
std::vector<unsigned int>& rows, std::vector<unsigned int>&
columns);

    // Overriding the pure virtual method
    unsigned int get_num_rows() const override;

private:
// the vector rows contains the row indices
std::vector<unsigned int> rows;
};

class SparseMatrixCSR : public SparseMatrix {
public:
   // Constructor
   SparseMatrixCSR(std::vector<double>& values,
std::vector<unsigned int>& row_idx, std::vector<unsigned int>&
columns);

   unsigned int get_num_rows() const override;

private:
// the vector row_idx contains the cumulative number of nonzero
entries up to the i-th row excluded
std::vector<unsigned int> row_idx;
};
```

Question 5
Explain the concept of dynamic memory management in C++.
Discuss the differences between `new`/`delete` and `malloc`/
`free`. Which scenarios are appropriate for each?

Answer
It's the allocation and deallocation of memory at runtime using
pointers. It's more flexible than static memory allocation (i.e.
at compile time) because the program can adapt to varying data
sizes and structures during execution.

`new`/`delete` is type safe, while `malloc`/`free` is not and it doesn't even invoke constructors and destructors. The first automatically determines the size, while the second requires that information. For this reason `malloc`/`free` is more appropriate when dealing with simple memory allocations where constructors are not invoked. When it isn't so, it's better to use `new`/`delete`, or even better smart pointers.

Question 6
What are lambda functions in C++ and Python?
How do they differ between the two languages and how are they used?

Answer
Lambda functions are similar to the MatLab anonymous functions and they are used to create short inlined functions quickly, thus only when the function takes up one line (for Python) or a few (for C++) of code.
In both languages we don't need to specify the return type, with the difference that in Python we don't write anything, while in C++ we write `auto` before the definition and we can also specify with `->` (example later).
In Python they are created with the keyword lambda, for example `add_one = lambda x: x + 1`.
In C++ the square brackets are used to capture specification, which allows to use variables in the enclosing scope inside the lambda either by value or by reference. For example, for doing the same thing done with Python, we can write:
```bash
auto f = [] (double x) { return x + 1; };
```
Which can be also specified with
```bash
auto f = [] (double x) -> double { return x + 1; };
```
If we want to capture all the variables by making a copy we can write:
```bash
auto g = [=] (float x, float y) { return x * y; };
```
And so on with other symbols inside [].

Question 7
What are the advantages of using NumPy arrays over standard Python lists for scientific computing?

Demonstrate with an example how to perform element-wise operations on NumPy arrays.

Answer
NumPy arrays offer an efficient way to store and manipulate numerical data, offering advantages over the traditional Python lists in terms of less memory, more speed and convenience. It also provides support for multidimensional arrays, allowing an easy representation of matrices.
An element-wise multiplication between two random matrices 3x3, can be simply done with:
```bash
import numpy as np
A = np.random.randint(1, 11, size=(3,3))
B = np.random.randint(1, 11, size=(3,3))
multiplication = A * B
```

Question 8
Discuss the capabilities of the Matplotlib library in Python.
Provide an example of how to create a line plot and a scatter plot using Matplotlib.

Answer
Matplotlib is a library for data visualization that offers a wide range of plotting tools with various types of plots like scatter plots, bar plots, histograms; it provides a high-level interface for drawing attractive statistical graphics.
A line plot can be:
```bash
import matplotlib.pyplot as plt
import numpy as np
x_values = np.linspace(0, 10, 100)
# Sine function for line plot
y_values_line = np.sin(x_values)
# Cosine function for scatter plot
y_values_scatter = np.cos(x_values)

# Creating Line plot
plt.figure(figsize=(8, 4))
plt.plot(x_values, y_values_line, label='Line Plot', color='blue', linestyle='-', linewidth=2)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line plot example')
plt.legend()
plt.show()

# Creating Scatter plot
plt.figure(figsize=(8, 4))
```

```
plt.scatter(x_values, y_values_scatter, label='Scatter Plot',
color='red', marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter plot example')
plt.legend()
plt.show()
```

Question 9
What are list comprehensions in Python and how do they work?
Rewrite a for loop that filters and transforms a list into an
equivalent list comprehension.

Answer
List comprehensions are a convenient and compact way to build
lists, tuples, sets and dictionaries in just one line. For example
instead of writing:
```bash
message = ["The", "cat", "is", "purring"]
first_letters = []
for word in message:
    first_letters.appen(word[0])
print(first_letters)
```
We can just write
```bash
letters = [word[0] for word in message]
print(letters)
```

Question 10
How does pybind11 handle type conversions between Python and C++?

Provide an example of passing complex data types (like a list or a
dictionary) from Python to C++ using pybind11.

Answer
The conversions are handled through a PYBIND11_MODULE macro that,
once defined, creates a function that will be called when an
import statement is issued from within Python. For example:
```bash
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>

// ----------------
// Regular C++ code
// ----------------
```

```
void do_something(const std::vector<int> &list) {
  // ...
}

// Wrap as Python module.
PYBIND11_MODULE(example, m) {
  m.doc() = "pybind11 example plugin";

  m.def("do_something", &do_something, "Do something");
}

// In test.py:
import example
list = [1, 2, 3, 4]
example.do_something(list)
```

Part 2: Programming exercise: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-09_simulation/part2.md

Exam simulation (11/01):

Part 1: Open-ended questions: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-11_simulation/part1.md

My answers:

```
## Question 1
What is a preprocessor and what is it used for?<br>
Given a C++ source file `code.cpp`, which shell command can be used to export the output of the preprocessor to a file `code_preprocessed.cpp`?

## Answer
A preprocessor is a program or set of directives which is part of the build process and performs preliminary operations on the source code before passing it to the compiler. It has the role of handling directives (like including header files with `#include`, conditional compilation `#ifdef #ifndef #else #endif`, compiler-specific directives `#pragma`) and macros (defining them with `#define`). It's used for code reusability and organization.<br>
The shell command to export the output of the preprocessor is:
```bash
g++ code.cpp -o code_preprocessed.cpp
```
```

Question 2
Describe what is meant by *undefined behavior* in C++.
Give examples of common programming mistake in C++ that can lead to undefined behavior.

Answer
It means that there is a situation for which the language doesn't prescribe a specific outcome, generating any result.
Some examples can be:
```cpp
// Trying to access memory beyond the allocated one
std::vector<int> v;
v.resize(2, 5);
std::cout << v[6];

// Using uninitialized variables
int x;
int y = x + 2;

// Modifying a const object
const int pi = 3.1415926;
pi = 3.14

// Type mismatch
int x = 5;
std::string saluto = "ciao";
double sum = x + saluto;
```

Question 3
Compare and contrast `std::unique_ptr`, `std::shared_ptr`, and `std::weak_ptr`.
When should each be used? Provide some prototypical examples of use.

Answer
Smart pointers control the lifespan of the resource they point to. `std::unique_ptr` means that the owned resource gets destroyed when the pointer goes out of scope; while for `std::shared_ptr` it happens when the last pointer owning that resource is destroyed. In fact in this last one, you have several objects that refer to the same resource. `std::weak_ptr` is a non-owning (thus weak) pointer to a shared resource; to access the referenced object it must be converted to a shared pointer; it's used to safely observe the shared object without affecting the ownership.
An example of the use of `std::unique_ptr` is:
```cpp
```

```cpp
class Polygon {};
class Triangle : public Polygon {};
class Square : public Polygon {};

int main () {
  std::unique_ptr<Polygon> polygon;
  std::string t;
  std::cin >> t;
  switch (t[0]) {
    case "T": // as in Triangle
      polygon = std::make_unique<Triangle>(...);
      break;
    case "S": // as in Square
      polygon = std::make_unique<Square>(...);
      break;
  }
  // Create another unique_ptr for swapping
  std::unique_ptr<Polygon> polygon2 =
std::make_unique<Triangle>();
  // You can swap ownership
  std::swap(polygon, polygon2);
return 0;
}
```

An example of use of `std::shared_ptr` is (part of my StatisticsModule of Homework2):

```cpp
class CSVHandler {
public:
  // Declare StatOp as a friend class of CSVHandler in order to access its private members
  friend class StatOp;
protected:
  // Shared pointer between the two classes
  const std::shared_ptr<CSVHandler> CSVfile;
}

class StatOp {
public:
    // Constructor
    StatOp(const std::shared_ptr<CSVHandler> CSVfile);
}

int main () {
    // Create shared pointer
    const std::shared_ptr<CSVHandler> CSVfile =
std::make_shared<CSVHandler>(input_file_path);
    return 0;
}
```

An example of use of `std::weak_ptr` is:
```cpp
std::shared_ptr<int> sptr = std::make_shared<int>(5);
std::weak_ptr<int> weak = sptr; // Get pointer to new data without
taking ownership
if (sptr = weak.lock()) {
  // Use the locked sptr to access the shared data
} else {
  // The shared data is out of scope
}
```

Question 4
What's the difference between *Aggregation* and *Inheritance* in
object-oriented programming?
Provide some conceptual example for each of the two types of
relationship.

Answer
Aggregation is a "has-a" relationship where one class contains the
other as a part, but the contained object can exist independently;
while inheritance is a "is-a" relationship where the derived class
inherits properties and behaviours from the base class, therefore
it can't exist without the base class and it's stronger.
An example of aggregation is the University class that may
aggregate Department classes:
```cpp
class Department {
public:
  Department(std::string name) : name(name) {}
private:
  std::string name;
};
class University {
public:
  void addDepartment(const Department& department) {
    departments.push_back(department);
  }
private:
  std::vector<Department> departments;
};
```
An example of inheritance is (part of my Homework1):
```cpp
class SparseMatrix { // abstract class
public:
    // Constructor
    SparseMatrix(std::vector<double>& values, std::vector<unsigned
int>& columns);
```

```
    // Define the common method of COO and CSR as virtual
    virtual unsigned int get_num_rows() const = 0;
protected:
// the vector values contains all the nonzero values
std::vector<double> values;
// the vector columns contains their corresponding columns indices
std::vector<unsigned int> columns;
};

class SparseMatrixCOO : public SparseMatrix {
public:
    // Constructor
    SparseMatrixCOO(std::vector<double>& values,
std::vector<unsigned int>& rows, std::vector<unsigned int>&
columns);

    // Overriding the pure virtual method
    unsigned int get_num_rows() const override;

private:
// the vector rows contains the row indices
std::vector<unsigned int> rows;
};

class SparseMatrixCSR : public SparseMatrix {
public:
   // Constructor
   SparseMatrixCSR(std::vector<double>& values,
std::vector<unsigned int>& row_idx, std::vector<unsigned int>&
columns);

   unsigned int get_num_rows() const override;

private:
// the vector row_idx contains the cumulative number of nonzero
entries up to the i-th row excluded
std::vector<unsigned int> row_idx;
};
```

Question 5
Explain the concept of dynamic memory management in C++.
Discuss the differences between `new`/`delete` and `malloc`/
`free`. Which scenarios are appropriate for each?

Answer
It's the allocation and deallocation of memory at runtime using
pointers. It's more flexible than static memory allocation (i.e.
at compile time) because the program can adapt to varying data
sizes and structures during execution.

`new`/`delete` is type safe, while `malloc`/`free` is not and it doesn't even invoke constructors and destructors. The first automatically determines the size, while the second requires that information. For this reason `malloc`/`free` is more appropriate when dealing with simple memory allocations where constructors are not invoked. When it isn't so, it's better to use `new`/`delete`, or even better smart pointers.

Question 6
What are lambda functions in C++ and Python?
How do they differ between the two languages and how are they used?

Answer
Lambda functions are similar to the MatLab anonymous functions and they are used to create short inlined functions quickly, thus only when the function takes up one line (for Python) or a few (for C++) of code.
In both languages we don't need to specify the return type, with the difference that in Python we don't write anything, while in C++ we write `auto` before the definition and we can also specify with `->` (example later).
In Python they are created with the keyword lambda, for example `add_one = lambda x: x + 1`.
In C++ the square brackets are used to capture specification, which allows to use variables in the enclosing scope inside the lambda either by value or by reference. For example, for doing the same thing done with Python, we can write:
```cpp
auto f = [] (double x) { return x + 1; };
```
Which can be also specified with
```cpp
auto f = [] (double x) -> double { return x + 1; };
```
If we want to capture all the variables by making a copy we can write:
```cpp
auto g = [=] (float x, float y) { return x * y; };
```
And so on with other symbols inside [].

Question 7
What are the advantages of using NumPy arrays over standard Python lists for scientific computing?

Demonstrate with an example how to perform element-wise operations on NumPy arrays.

Answer
NumPy arrays offer an efficient way to store and manipulate numerical data, offering advantages over the traditional Python lists in terms of less memory, more speed and convenience. It also provides support for multidimensional arrays, allowing an easy representation of matrices.
An element-wise multiplication between two random matrices 3x3, can be simply done with:
```python
import numpy as np
A = np.random.randint(1, 11, size=(3,3))
B = np.random.randint(1, 11, size=(3,3))
multiplication = A * B
```

Question 8
Discuss the capabilities of the Matplotlib library in Python.
Provide an example of how to create a line plot and a scatter plot using Matplotlib.

Answer
Matplotlib is a library for data visualization that offers a wide range of plotting tools with various types of plots like scatter plots, bar plots, histograms; it provides a high-level interface for drawing attractive statistical graphics.
A line plot can be:
```python
import matplotlib.pyplot as plt
import numpy as np
x_values = np.linspace(0, 10, 100)
# Sine function for line plot
y_values_line = np.sin(x_values)
# Cosine function for scatter plot
y_values_scatter = np.cos(x_values)

# Creating Line plot
plt.figure(figsize=(8, 4))
plt.plot(x_values, y_values_line, label='Line Plot', color='blue', linestyle='-', linewidth=2)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line plot example')
plt.legend()
plt.show()

# Creating Scatter plot
plt.figure(figsize=(8, 4))
```

```
plt.scatter(x_values, y_values_scatter, label='Scatter Plot',
color='red', marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter plot example')
plt.legend()
plt.show()
```

Question 9
What are list comprehensions in Python and how do they work?
Rewrite a for loop that filters and transforms a list into an
equivalent list comprehension.

Answer
List comprehensions are a convenient and compact way to build
lists, tuples, sets and dictionaries in just one line. For example
instead of writing:
```python
message = ["The", "cat", "is", "purring"]
first_letters = []
for word in message:
    first_letters.appen(word[0])
print(first_letters)
```
We can just write
```python
letters = [word[0] for word in message]
print(letters)
```

Question 10
How does pybind11 handle type conversions between Python and C++?

Provide an example of passing complex data types (like a list or a
dictionary) from Python to C++ using pybind11.

Answer
The conversions are handled through a PYBIND11_MODULE macro that,
once defined, creates a function that will be called when an
import statement is issued from within Python. For example:
```cpp
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>

// -----------------
// Regular C++ code
// -----------------
```

```
void do_something(const std::vector<int> &list) {
  // ...
}

// Wrap as Python module.
PYBIND11_MODULE(example, m) {
  m.doc() = "pybind11 example plugin";

  m.def("do_something", &do_something, "Do something");
}

// In test.py:
import example
list = [1, 2, 3, 4]
example.do_something(list)
```

Part 1 simulation 2: Open-ended questions

Question 1
In a UNIX shell, what is the difference between the single pipe (`command1 | command2`) and the double pipe (`command1 || command2`) operators?
Provide a minimal working example for each of the two.

Answer
The single pipe is like the composition of a function: the output of the command1 gets forwarded as input for the command2. For example `cut -f 3 -d ',' file.csv | sort | wc -l` uses cut to subdividing a file (with some flags for how to do it), then its output gets sorted, lastly it applies a word count (actually a line count due to the flag) to the sorted values.
The double pipe is the logical "or" operation. For example to verify if a file exists and create it if it does not: `[ -e file.txt ] || touch file.txt`.

Question 2
Which types of constructor can be defined for a C++ class?
Provide some examples for a class representing a 3D tensor.

Answer
Default constructor: takes no arguments; if no constructor is provided, C++ generates a default one automatically using default values (such as empty for strings, zero for numbers).
Parametrized constructor: takes one or more parameters to initialize member variables based on the provided values.
Copy constructor: creates a new object as a copy of an existing object of the same class, taking a reference to an object of the

same class as a parameter. It is invoked when objects are copied, passed by value or initialized with other objects.
Copy assignment operator: constructor used for copy assignment
Move constructor: it allows to transfer the resources from one object to another
Destructor: special member function used (and automatically called) to clean up resources held by an object before it goes out of scope or is explicitly deleted. If it's done inside an abstract class, it must be virtual.

```cpp
class Tensor3D {
private:
    int x_size, y_size, z_size;
    std::vector<std::vector<std::vector<double>>> data;

public:
    // Default Constructor
    Tensor3D() : x_size(0), y_size(0), z_size(0) {}

    // Parameterized Constructor
    Tensor3D(int x, int y, int z) : x_size(x), y_size(y), z_size(z) {
        data.resize(x, std::vector<std::vector<double>>(y, std::vector<double>(z, 0.0)));
    }

    // Copy Constructor
    Tensor3D(const Tensor3D& other) : x_size(other.x_size), y_size(other.y_size), z_size(other.z_size), data(other.data) {}

    // Move Constructor
    Tensor3D(Tensor3D&& other) noexcept : x_size(0), y_size(0), z_size(0), data() {
        swap(*this, other);
    }

    // Copy Assignment Operator
    Tensor3D& operator=(const Tensor3D& other) {
      if (this != &other) {
        Tensor3D temp(other);
        swap(*this, temp);
      }
      return *this;
    }

    // Destructor
    ~Tensor3D() {}
};
```

Question 3
Write a C++ function template called sum that accepts a half-open range $[begin, end)$ (as iterators) of int objects and returns the sum of the sequence.

Answer
```cpp
int sum(Iterator begin, Iterator end) {
  int result = 0;
  for (Iterator i = begin; it != end; ++it)
    result += *it;
  return result;
}
```

Question 4
What is the meaning of the following C++ code?

```cpp
template <typename T, template <typename> class Container>
class ContainerWrapper {
private:
    Container<T> data;
};
```

How to instantiate a `ContainerWrapper` object?

Answer
This code defines a template classContainerWrapper with two template parameters: the type of the elements T and another template class called Container. It contains a private member data of type Container. Thanks to these templates, the ContainerWrapper can contain whichever type of Container.
A ContainerWrapper object can be instantiated as:

```cpp
ContainerWrapper<type1, container> wrapper;
```

where type1 and type2 are two types (like int, double, std::string, std::vector etc), in particular type2 is the same type of Container.

Question 5

In C++, what are the key differences in behavior and usage when functions return values by direct value, by pointer, and by reference?

Answer
When returning by value, it returns a copy of the value, so it's better to use it for small, non-mutable data.
When returning by pointer, it returns the pointer to that value, so it's usually used for dynamically allocated data.
When returning by reference, it returns the reference to that value, so it's used for efficiency and direct modification of data.

Question 6
What are the similarities and differences in object-oriented features between C++ and Python?
How does the implementation of concepts like inheritance and polymorphism vary in these two languages?

Answer
* A main difference is the lack of pointers in Python, because it automatically manages the memory, contrary to C++.
* Python offers also magic methods, which are special methods automatically invoked in response to certain events or operations, like **__add__**, **__eq__**, **__getitem__** etc.
* In Python all class members are public and all methods are virtual, which means they can be overloaded anytime. In C++ members can also be private or protected, which is also why you can declare a class as friends of another one (to access its protected members).
* In the class methods, Python requires a first name that refers to the object itself and it's called self; it's the equivalent of the this pointer in C++.
* What said above is also for the constructor in Python, which must take self as first argument and initializes every member with self.nameofmember.
* They both support multiple inheritance.
* While in C++ it is specified with `class DerivateClass : public BaseClass`, in Python it's `class DerivateClass(BaseClass)`.
* If there isn't a constructor for the derived class, the one of the abstract is automatically called in Python.
* When calling a function of the BaseClass, Python can either use its name or the function super().functionToCall.

Question 7

How does Boolean indexing work in NumPy for vectorized data selection?
Give an example of filtering data in an array based on a condition.

Answer
With NumPy an array of boolean values can be used as an index of another array, where each element of the boolean array indicates whether or not to select the elements from the array (respectively True and False).

```python
a = np.arange(1,100)
b = a % 2 == 0 # When printed returns [True, False, True, False,...]
c = a[b] # When printed returns odd numbers
```

Question 8
How to handle an exception in Python?
Illustrate the syntax of raising and catching exceptions with an example.

Answer
You can raise an error when something bad is being done, and when executing that operation, you can use a try-except block to handle the exception.

```python
def division(x, y):
  if y == 0:
    raise ValueError("Can't divide by zero.")
  return x/y

try:
  result = division(10,8)
except ValueError as e:
  print("Error: ", str(e))
```

Question 9
The following Python code snippet is meant to create a list named `another_list` with elements 1 and 2.
Find the error and fix it.

```python
def append_to_list(element, target_list=[]):
    target_list.append(element)
```

````
    return target_list

my_list = append_to_list(1)
another_list = append_to_list(2)
```

## Answer
The target_list in the argument should not be instantiated as [] but as None, otherwise when a list is given, it overwrites it (in the main here it doesn't happen because it's used this default value).
Furthermore the request was to append 1 and 2 to the same list, here it is not done. So the code should be:

```python
def append_to_list(element, target_list=None):
    if target_list is None:
      target_list = []
    target_list.append(element)
    return target_list

another_list = append_to_list(1)
another_list = append_to_list(2, another_list)
```

---

## Question 10
How does pybind11 handle default arguments in C++ functions when exposed to Python?

## Answer
It allows it, specifying the default value when defining the C++ function in the pybind11 module, for example writing:
```cpp
PYBIND11_MODULE(module, m) {
  m.def("function", &function, py::arg("a"), py::arg("b") = 42);
}
```
````

Part 2: Programming exercise: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-11_simulation/part2.md
````

Exam simulation (11/01):

Part 1: Open-ended questions: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-17/part1.md

My answers:

````
# Part 1: Open-ended questions

## Question 1
In Object Oriented Programming, what is *polymorphism*? In C++, what is the difference between dynamic and static polymorphism?
<br>
Provide a real-world example for at least one of the two techniques.

## Answer
Polymorphism allows objects of different classes to be treated as objects of a common base class, promoting code flexibility and working with objects at a higher level of abstraction.
The difference between dynamic and static is that dynamic is done at runtime (so it can call method based on the type of object), while static is at compile time.
An example of dynamic polymorphism is when using abstract and derived classes:
```cpp
class Polygon {
public:
    virtual double area() { //something;
    }
};
class Square : public Polygon {
public:
    double area() override { return side * side; }
private:
    double side;
};
void f(const Polygon &p) {
    const double a = p.area();
    // ...
}

int main (){
Square s;
f(s); // Polymorphism converts 'const Square &' to 'const Polygon &'
return 0;
}
```
````

Question 2
Using template metaprogramming, write a C++ code that, without any use of `for` or `while` loops, prints the first 100 integer numbers.

Answer
```cpp
template <unsigned int n>
class printInt {
public:
    static void print() {
        printInt<n - 1>::print(); // Recursive call
        std::cout << n << ' ';
    }
};

// Base case where to stop
template <>
class printInt<1> {
public:
    static void print() {
        std::cout << 1 << ' '; // Print the last int i.e. 1
    }
};

int main() {
    printInt<100>::print();
    std::cout << std::endl;
    return 0;
}
```

Question 3
What is *move semantics* in C++? What is the meaning of *lvalue* and *rvalue* expressions, and how are they related to move semantics?
Provide an example where move semantics might be beneficial.

Answer
Move semantics means transferring ownership of some resource to another object.
An lvalue is an expression that may appear on the left (that's why the name) and on the right-hand side of an assignment. It refers to a memory location and allows us to take its address via the & operator. Examples of it are the value held in a variable, the returned value reference of a function.

An rvalue is an expression that can only appear on the right-hand side of an assignment. Examples of it are the returned value of a function.
They're related to move semantics because they are move operators. Move semantics might be beneficial when swapping may be costly, so instead of doing
```cpp
void swap(Matrix& a, Matrix& b) {
Matrix tmp{a}; // Make a copy of a.
a = b; // Copy assign b to a.
b = tmp; // Copy assign tmp to b.
}
```
We can do
```cpp
void swap_
with
_
move(Matrix& a, Matrix& b) {
// Swap number of rows and columns.
double* tmp = a.data; // Save the pointer.
a.data = b.data; // Copy the pointer.
b.data = tmp; // Copy the saved pointer.
}
```
In this way we just swap the pointers, saving memory and operations, but only for this specific situation since we can't write a function template (we don't know how data is stored in T).

Question 4
Given the following C++ code, which *unit tests* would you perform and why?

```cpp
class Geometry {
public:
    double calculate_circle_area(double radius) {
        if (radius < 0) {
            throw std::invalid_argument("Radius cannot be negative.");
        }
        return 3.14159 * radius * radius;
    }

    double calculate_rectangle_area(double length, double width) {
        if (length < 0 || width < 0) {
            throw std::invalid_argument("Length and width cannot be negative.");
```

```
        }
        return length * width;
    }
};
```

Answer
I would do the following unit tests:
Call calculate_circle_area with a negative value and check that I get std::invalid_argument exception, to assess that a negative radius is correctly handled.
Call calculate_circle_area with some positive radius values and assess that the method returns the correct area. To be sure, I would also check that the absolute difference between the actual value and the expected value is less than a small threshold (for example 10 to the power of -9).
```cpp
std::vector<double> radiuses {0,1,2,4,8,55};
std::vector<double> expected {0, 3.14159, 12.56636, 50.26544, 201.06176, 9503.30975};
Geometry g;
const double threshold = 1e-9;
unsigned int i = 0;
for(double& radius : radiuses){
    double actual = g.calculate_circle_area(radius);
    assertLess(std::abs(expected[radius] - actual), threshold);
}
```
I would then call calculate_rectangle_area with a negative value for height, then call it with a negative value for width, and finally call it with both negative values for height and width. Every time it must throw `std::invalid_argument exception` to assess that negative rectangle sides are correctly handled.
Repeat what done for calculate_circle_area with calculate_rectangle_area, with an array of pairs of height and width and an array of expected areas. Every time the difference between the actual area and the expected area must be lower than a small threshold.

Question 5
What is the difference between local variables and cache variables in CMake? What are shell environment variables and how to access them from a CMake script?
Provide some examples.

Answer
As the name says, local variables have a limited scope and are not visible outside of their definition, while cache variables store their values, so they can be used to interact with the command

line setting options on or off (for example when using a module like the request for the homework2: `option(USE_MODULEC "Use the Numerical Integration Module" ON)`). An example of local variable in the CMake is `set(localvariable "Local Variable")`.
Shell environment variables are set in the environment of the shell from which CMake is invoked and they are accessible using the $ENV keyword (`message("This is a shell environment variable: $ENV{ENV_VARIABLE}")`)

Question 6
Compare the two following Python functions. Which one is more *Pythonic*? Which one is more efficient and why?

```python
def square1(lst):
    result = []
    for i in lst:
        result.append(i * i)
    return result

def square2(lst):
    return [i * i for i in lst]
```

Answer
The second one is more "pythonic" because list comprehensions are a particularity of Python. It's also more efficient because it's just one line of code instead of 4, you don't need to define another variable (like result in square1) but you just use the one given as input argument.

Question 7
In Python, what is the difference between positional and keyword arguments?
How to enforce that a function receives an input parameter of a _given_ type (e.g., an integer or a string)?

Answer
A positional argument is an argument of a function which is passed via the position of it in the function call. For example
```python
def function(arg, arg2):
    pass
function(12,”arg”)
```
12 and “arg” will be positional arguments.

Keyword arguments are arguments passed to the function via their name, independent of the position of it in the function call. For example:
`function(arg2="arg",arg1=12)`. Here the arguments have the same value but they have been passed to the function independently of the position.
In python you can indicate which type is expected using this notation `def function(arg: type), for example def function(arg: int) `.
To enforce you could assert that an argument is an instance of a specific type: `assert isinstance(arg,type)`

Question 8
What is *context management* in Python and which methods should a class expose in order to be able to use it?

Answer
A context management, used with the magic methods __enter__ and __exit__, is used for resource acquisition and release in a `with` statement, when some resources need to be properly managed, for example when you want to open a file.

Question 9
Vectorize the following Python function that calculates the square of each element in a list, so that it can efficiently work with NumPy arrays.
What are some advantages and disadvantages of the vectorized function?

```python
def square_elements(my_list):
    squared_list = []
    for item in my_list:
        squared_list.append(item ** 2)
    return squared_list
```

Answer
It is enough to write np.square(my_list) and the function np.square will return a numpy array containing the squared elements of the list.
Vectorizing functions using numpy leads to a faster code because numpy is highly optimised. Moreover, the output is a numpy array, which can be concatenated with other numpy operations.
Some disadvantages might be the fact that it is more difficult to debug vectorized code.

Question 10
Given the following compilation command, what is the meaning of each line/flag and what are they used for?

```bash
g++ \
    -std=c++11 \
    -O3 \
    -shared \
    -fPIC \
    $(python3 -m pybind11 --includes) \
    $(python3-config --cflags --ldflags --libs) \
    example.cpp \
    -o example$(python3-config --extension-suffix)
```

Answer
```bash
g++ \ # indicates the compiler to use
    -std=c++11 \ # tells the compiler to use C++ 11
    -O3 \ # tells the debugger to perform the highest level of optimizations
    -shared \ # tells it's a shared library
    -fPIC \ # compiler flag to use for shared libraries (PIC stands for Position Independent Code)
    $(python3 -m pybind11 --includes) \ # includes the modules written with pybind11
    $(python3-config --cflags --ldflags --libs) \ #must include respectively compiler flags, linker flags, libraries
    example.cpp \ # is the source file to compile
    -o example$(python3-config --extension-suffix) # -o example tells that the output will have name example
```
This whole command takes a source file example.cpp and compiles it with c++11, adding a module created with pybind11 and a shared library.

Part 2: Programming exercise: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-17/part2.md

Exam (17/01):
Part 1: Open-ended questions: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-17/part1.md

My answers (with scores):

```
# Part 1: Open-ended questions

## Question 1
In Object Oriented Programming, what is *polymorphism*? In C++,
what is the difference between dynamic and static polymorphism?
<br>
Provide a real-world example for at least one of the two
techniques.

## Answer (1.25/1.5)
Polymorphism allows objects of different classes to be treated as
objects of a common base class, promoting code flexibility and
working with objects at a higher level of abstraction.
The difference between dynamic and static is that dynamic is done
at runtime (so it can call method based on the type of object),
while static is at compile time.
An example of dynamic polymorphism is when using abstract and
derived classes:
```cpp
class Polygon {
public:
    virtual double area() { //something;
    }
};
class Square : public Polygon {
public:
    double area() override { return side * side; }
private:
    double side;
};
void f(const Polygon &p) {
    const double a = p.area();
    // ...
}

int main (){
Square s;
f(s); // Polymorphism converts 'const Square &' to 'const Polygon
&'
return 0;
}
```

Question 2
Using template metaprogramming, write a C++ code that, without any
use of `for` or `while` loops, prints the first 100 integer
numbers.
```

Answer (1.25/1.5)
```cpp
template <unsigned int n>
class printInt {
public:
    static void print() {
        printInt<n - 1>::print(); // Recursive call
        std::cout << n << ' ';
    }
};

// Base case where to stop
template <>
class printInt<1> {
public:
    static void print() {
        std::cout << 1 << ' '; // Print the last int i.e. 1
    }
};

int main() {
    printInt<100>::print();
    std::cout << std::endl;
    return 0;
}
```

Question 3
What is *move semantics* in C++? What is the meaning of *lvalue* and *rvalue* expressions, and how are they related to move semantics?
Provide an example where move semantics might be beneficial.

Answer (1.25/1.5)
Move semantics means transferring ownership of some resource to another object.
An lvalue is an expression that may appear on the left (that's why the name) and on the right-hand side of an assignment. It refers to a memory location and allows us to take its address via the & operator. Examples of it are the value held in a variable, the returned value reference of a function.
An rvalue is an expression that can only appear on the right-hand side of an assignment. Examples of it are the returned value of a function.
They're related to move semantics because they are move operators.
Move semantics might be beneficial when swapping may be costly, so instead of doing
```cpp
void swap(Matrix& a, Matrix& b) {
```

```
Matrix tmp{a}; // Make a copy of a.
a = b; // Copy assign b to a.
b = tmp; // Copy assign tmp to b.
}
```
We can do
```cpp
void swap_
with
_
move(Matrix& a, Matrix& b) {
// Swap number of rows and columns.
double* tmp = a.data; // Save the pointer.
a.data = b.data; // Copy the pointer.
b.data = tmp; // Copy the saved pointer.
}
```
In this way we just swap the pointers, saving memory and
operations, but only for this specific
situation since we can't write a function template (we don't know
how data is stored in T).

Question 4
Given the following C++ code, which *unit tests* would you perform
and why?

```cpp
class Geometry {
public:
    double calculate_circle_area(double radius) {
        if (radius < 0) {
            throw std::invalid_argument("Radius cannot be
negative.");
        }
        return 3.14159 * radius * radius;
    }

    double calculate_rectangle_area(double length, double width) {
        if (length < 0 || width < 0) {
            throw std::invalid_argument("Length and width cannot
be negative.");
        }
        return length * width;
    }
};
```

Answer (1.5/1.5)
I would do the following unit tests:

Call calculate_circle_area with a negative value and check that I get std::invalid_argument exception, to assess that a negative radius is correctly handled.
Call calculate_circle_area with some positive radius values and assess that the method returns the correct area. To be sure, I would also check that the absolute difference between the actual value and the expected value is less than a small threshold (for example 10 to the power of -9).
```cpp
std::vector<double> radiuses {0,1,2,4,8,55};
std::vector<double> expected {0, 3.14159, 12.56636, 50.26544, 201.06176, 9503.30975};
Geometry g;
const double threshold = 1e-9;
unsigned int i = 0;
for(double& radius : radiuses){
    double actual = g.calculate_circle_area(radius);
    assertLess(std::abs(expected[radius] - actual), threshold);
}
```
I would then call calculate_rectangle_area with a negative value for height, then call it with a negative value for width, and finally call it with both negative values for height and width. Every time it must throw `std::invalid_argument exception` to assess that negative rectangle sides are correctly handled.
Repeat what done for calculate_circle_area with calculate_rectangle_area, with an array of pairs of height and width and an array of expected areas. Every time the difference between the actual area and the expected area must be lower than a small threshold.

Question 5
What is the difference between local variables and cache variables in CMake? What are shell environment variables and how to access them from a CMake script?
Provide some examples.

Answer (1.25/1.5)
As the name says, local variables have a limited scope and are not visible outside of their definition, while cache variables store their values, so they can be used to interact with the command line setting options on or off (for example when using a module like the request for the homework2: `option(USE_MODULEC "Use the Numerical Integration Module" ON)`). An example of local variable in the CMake is `set(localvariable "Local Variable")`.
Shell environment variables are set in the environment of the shell from which CMake is invoked and they are accessible using the $ENV keyword (`message("This is a shell environment variable: $ENV{ENV_VARIABLE}")`)

Question 6
Compare the two following Python functions. Which one is more *Pythonic*? Which one is more efficient and why?

```python
def square1(lst):
    result = []
    for i in lst:
        result.append(i * i)
    return result

def square2(lst):
    return [i * i for i in lst]
```

Answer (1.25/1.5)
The second one is more "pythonic" because list comprehensions are a particularity of Python. It's also more efficient because it's just one line of code instead of 4, you don't need to define another variable (like result in square1) but you just use the one given as input argument.

Question 7
In Python, what is the difference between positional and keyword arguments?
How to enforce that a function receives an input parameter of a _given_ type (e.g., an integer or a string)?

Answer (1.5/1.5)
A positional argument is an argument of a function which is passed via the position of it in the function call. For example
```python
def function(arg, arg2):
    pass
function(12,”arg”)
```
12 and “arg” will be positional arguments.
Keyword arguments are arguments passed to the function via their name, independent of the position of it in the function call. For example:
`function(arg2=“arg”,arg1=12)`. Here the arguments have the same value but they have been passed to the function independently of the position.
In python you can indicate which type is expected using this notation `def function(arg: type), for example def function(arg: int) `.

To enforce you could assert that an argument is an instance of a specific type: `assert isinstance(arg,type)`

Question 8
What is *context management* in Python and which methods should a class expose in order to be able to use it?

Answer (1.25/1.5)
A context management, used with the magic methods **__enter__** and **__exit__**, is used for resource acquisition and release in a `with` statement, when some resources need to be properly managed, for example when you want to open a file.

Question 9
Vectorize the following Python function that calculates the square of each element in a list, so that it can efficiently work with NumPy arrays.
What are some advantages and disadvantages of the vectorized function?

```python
def square_elements(my_list):
    squared_list = []
    for item in my_list:
        squared_list.append(item ** 2)
    return squared_list
```

Answer (1.5/1.5)
It is enough to write np.square(my_list) and the function np.square will return a numpy array containing the squared elements of the list.
Vectorizing functions using numpy leads to a faster code because numpy is highly optimised. Moreover, the output is a numpy array, which can be concatenated with other numpy operations.
Some disadvantages might be the fact that it is more difficult to debug vectorized code.

Question 10
Given the following compilation command, what is the meaning of each line/flag and what are they used for?

```bash
g++ \
```

```
    -std=c++11 \
    -O3 \
    -shared \
    -fPIC \
    $(python3 -m pybind11 --includes) \
    $(python3-config --cflags --ldflags --libs) \
    example.cpp \
    -o example$(python3-config --extension-suffix)
```

Answer (1.25/1.5)
```bash
g++ \ # indicates the compiler to use
    -std=c++11 \ # tells the compiler to use C++ 11
    -O3 \ # tells the debugger to perform the highest level of
optimizations
    -shared \ # tells it's a shared library
    -fPIC \ # compiler flag to use for shared libraries (PIC
stands for Position Independent Code)
    $(python3 -m pybind11 --includes) \ # includes the modules
written with pybind11
    $(python3-config --cflags --ldflags --libs) \ #must include
respectively compiler flags, linker flags, libraries
    example.cpp \ # is the source file to compile
    -o example$(python3-config --extension-suffix) # -o example
tells that the output will have name example
```
This whole command takes a source file example.cpp and compiles it
with c++11, adding a module created with pybind11 and a shared
library.

Part 2: Programming exercise: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-01-17/part2.md

Exam (13/02):

Part 1: Open-ended questions: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-02-13/part1.md

My answers (with scores):

Part 1: Open-ended questions

Question 1

Describe a specific task you automated using a shell script.
What was the task, and how did your script improve the process?
Share the script and explain its functionality.

Answer (1.5/1.5)
In the first homework for this course, I used a shell script in order to build everything together. The task of the homework was to create a base class SparseMatrix and its two derived classes SparseMatrixCOO and SparseMatrixCSR that allow you to write a sparse matrix with two different methods.
My build.sh improved the process allowing me to build what required faster and in a more efficient way. The build.sh was:
```bash
#!/bin/bash
set -x
mkdir -p build
g++ -std=c++17 -Iinclude/ src/SparseMatrix.cpp -c -o build/SparseMatrix.o
g++ -std=c++17 -Iinclude/ src/SparseMatrixCOO.cpp -c -o build/SparseMatrixCOO.o
g++ -std=c++17 -Iinclude/ src/SparseMatrixCSR.cpp -c -o build/SparseMatrixCSR.o
g++ -std=c++17 -Iinclude/ src/main.cpp -c -o build/main.o
g++ -std=c++17 -Wall -Wpedantic build/SparseMatrix.o build/SparseMatrixCOO.o build/SparseMatrixCSR.o build/main.o -o build/HomeworkSF
set +x
if [ $? -eq 0 ]; then
    echo "Build successful! You can run the program using ./build/HomeworkSF"
else
    echo "Build failed."
fi
```
With this, in the terminal, you first give the permissions with `chmod +x build.sh`, then write `./build.sh`, lastly run it with `./build/HomeworkSF`.
In this way, the procedure of preprocessing and compiling (and eventually linking if there was a library) is automated, you just have to run the build file and then load the program.

Question 2
Design a minimal C++ class hierarchy for a simple game engine that includes entities like `Player`, `Enemy`, and `Item`.
Discuss how polymorphism, inheritance, or other forms of relationship between classes can be used to organize and extend the engine.

Answer (0.5/1.5)

The design might be:
```cpp
class ActiveObject{
    public:
        virtual void doAction() = 0;
        virtual void render() = 0;
        unsigned long numPolygons;
        virtual ~ActiveObject() = default;
}
class Item : public ActiveObject{
    public:
        Item() {...};
        void doAction override{...};
        void render() override{...};
}
class Player : public ActiveObject{
    public:
        Player() {};
        void doAction override{...};
        void render() override{...};
}
class Enemy : public ActiveObject{
    public:
        Enemy() {};
        virtual void doAction override{...};
        void render() override{...}
}
```
Where ActiveObject is an abstract base class which represents a rendered object capable of performing an action. All the ohter classes inherit from this base class.
- Polymorphism can be used for example when all the object in a scene must perform their specific action. Every item of an array containing all the ActiveObjects in a scene have just to call .doAction.
- Inheritance can be used when rendering an object. Every ActiveObject must be rendered, but in different ways. For example if the game is a "first person shooter" just the part of the player visible in the screen can be rendered (hands, legs); while an Item must always be rendered.

Question 3
Provide an example of how you've used a functor or lambda function in C++ to simplify your code, especially in the context of STL algorithms.
Explain your choice and its benefits.

Answer (1.5/1.5)
For example, I used a lambda function in the second assignemnt of the course. In the statistic operations module, when I needed to

compute the median of a column of a dataset, I had to sort the column to find the median value. What I did was:

```cpp
// Sorting the data based on the index of the variant
    std::sort(sortedData.begin(), sortedData.end(), [](const auto &a, const auto &b) {
        if (a.index() == 0 && b.index() == 0) {
            return std::get<double>(a) < std::get<double>(b);
        } else if (a.index() == 1 && b.index() == 1) {
            return std::get<std::string>(a) < std::get<std::string>(b);
        } else {
            // If they're different types, sort by index
            return a.index() < b.index();
        }
    });
```

Since the column had either `double` or `std::string` values, I had to create a custom comparator with a lambda function, which checked the type of the values that were compared. If they were `double`, I compared them by their value. If they were `std::string`, I compared them by the < operator of the `std::string` class. Using the STL and lambdas was very convinient, as I did not need to write my sorting algorithm, but just define a custom comparator "on the fly". The code was more readable and I could use a sorting algorithm which has been implemented to be very efficient, which is a great advantage considering that the dataset was quite large.

Question 4

Consider the following C++ code snippet.
Detect as many errors as you can and fix them.

```cpp
class Shape {
public:
    virtual void print() const = 0;
};

class Circle : public Shape {
public:
    Circle(double r) : radius(r) {}

    virtual void print() const override {
        std::cout << "A circle with radius " << radius << "." << std::endl;
    }

private:
    double radius;
```

```
};

auto s1 = &Shape();
s1->print();

auto s2 = new Circle(0.5);
s2->print();

delete s1;
delete s2;
```

Answer (1.5/1.5)
```cpp
class Shape {
public:
    virtual void print() const = 0;
    virtual ~Shape() = default; // the virtual constructor was missing
};

class Circle : public Shape {
public:
    Circle(double r) : radius(r) {}

    void print() const override {
        std::cout << "A circle with radius " << radius << "." << std::endl; // virtual is not necessary if no class inherits from Circle
    }

private:
    double radius;
};

// You cannot instantiate an object of an abstract class
//auto s1 = &Shape();
//s1->print();

auto s2 = new Circle(0.5);
s2->print();

//delete s1;
delete s2;
```

Question 5
The following C++ code snippet implements the Babylonian method to approximate the square root of a given number at compile time, exploiting template metaprogramming.

However, the method only works for integer square roots. How would you use it to approximate, e.g., $\sqrt{2}$ at compile time?

```cpp
#include <iostream>
#include <type_traits>

template <unsigned int N, unsigned int X, unsigned int NextX = (X + N / X) / 2>
struct BabylonianMethod : BabylonianMethod<N, NextX> {};

// Specialization to stop recursion.
template <unsigned int N, unsigned int X>
struct BabylonianMethod<N, X, X> {
    static constexpr unsigned int value = X;
};

// Helper template to initiate the Babylonian method with a good enough first guess.
template <unsigned int N>
using Sqrt = BabylonianMethod<N, N / 2 + 1>;

int main() {
    // Compute square root of 16 at compile time.
    constexpr unsigned int sqrt16 = Sqrt<16>::value;
    std::cout << "Square root of 16 is " << sqrt16 << std::endl;

    return 0;
}
```

Answer (0/1.5)
I would change the template definitions to accept floating point numbers, in this way
```cpp
#include <iostream>
#include <type_traits>

template <typename T, T N, T X, T NextX = (X + N / X) / 2>
struct BabylonianMethod : BabylonianMethod<T, N, NextX> {};

template <typename T, T N, T X>
struct BabylonianMethod<T, N, X, X> {
    static constexpr T value = X;
};

template <typename T, T N>
using Sqrt = BabylonianMethod<T, N, N / 2 + 1>;

int main() {
    // Compute square root of 2.0 at compile time.
    constexpr double sqrt2 = Sqrt<double, 2>::value;
```

```
    std::cout << "Square root of 2 is " << sqrt2 << std::endl;

    return 0;
}
```
Note that there might be problems when the algorithm is converging but cannot reach the same exact value of the floating point solution. A compile-time check with a small epsilon would lead to a more stable and consistent algorithm.

Question 6
Explain how you would convert an existing `Makefile` project to use CMake, highlighting the steps involved in creating a `CMakeLists.txt` file.
Discuss benefits and disadvantages, if any, of this transition.

Answer (1.25/1.5)
I would firstly create a `CMakeLists.txt` file in the root directory; then in this file I would specify the minimum reuired CMake Version `cmake_minimum_required(VERSION 3.27.1)` and the `project name project(ProjectTitle)`. After this general settings, I would create a CMakeFile.txt file for every submodule that I want to create. In every CMake file I would add executables or libraries to define the targets of my project, link libraries or set compiler flags. After setting all the CMakeLists.txt files, I would create a build directory where the generated MakeFile can be used to build the project.
Using CMake has several advantages: it is platform independent, so the project can be easily ported to different systems; CMake utilizes also a clearer and more concise way to specify build settings, which makes the project more mantainable. Especially the creation of submodules (setting cache variables on/off) gave me a great advantage when developing projects, allowing me to mantain and build seperate modules within the same project with ease.

Question 7
Share your experience with debugging or profiling a performance issue in a C++ or Python application.
Which tools or techniques did you use, and how did you resolve the issue?

Answer (1.5/1.5)
After the first exam, I tried to do again at home the second part, which involved the implementation of the Gradient Descent algorithm. The class was not working well, it was not converging at all. So I tried to use the gdbc tool in order to debug the `optimize` function. I put a breakpoint in the function and then checked the value of the maximum number of iterations to check if

they were correctly set. Using `print maxIterations` gave me value 0, so the algorithm was not executing even one iteration. Thanks to the debugger I narrowed the problem down, and discovered that I did not set correctly the algotithm parameters. After fixing this issue, the algorithm worked as expected.

Question 8
Explain the difference between list comprehensions and generator expressions in Python.
Give an example of a scenario where using a generator would be significantly more memory-efficient than a list comprehension.

Answer (1.5/1.5)
Both list comprehensions and generator expression could be used to create lists based on some conditions or specific functions to apply to every item. List comprehensions actually creates the whole list, like here:
```python
l = [0,1,2,5,100]
l2 = [x**2 for x in l]
for value in l2:
    print(l2)
```
l2 will contain all the squared values of the original list l.
If a generator expression is used:
```python
l = [0,1,2,5,100]
l2 = (x**2 for x in l)
for value in l2:
    print(l2)
```
l2 would not contain the squared values but it would generate them at every iteration of the for cycle, so just when needed. In this way generator expressions are much more memory efficient than list comprehensions. For example when processing a very large dataset, generator expressions would be the perfect fit, since there would not be necessary to load the entirety of the dataset into memory, but just process its records when needed.

Question 9
Given the following Python code, which uses a custom class decorator to modify or add methods to classes, analyze how the decorator affects both the `BaseClass` and the `ChildClass` with respect to the methods `greet` and `farewell`.
Considered the intricacies of inheritance, method overriding, and dynamic attribute modification introduced by the decorator, explain what happens when instances of these classes call the `greet` and `farewell` methods.

```python
def class_decorator(method_name):
    def wrapper(cls):
        original_method = getattr(cls, method_name, None)
        if original_method:
            def new_method(self, *args, **kwargs):
                return f"Modified behavior: {original_method(self, *args, **kwargs)}"
            setattr(cls, method_name, new_method)
        else:
            def new_method(self):
                return f"New method added to {cls.__name__}"
            setattr(cls, method_name, new_method)
        return cls
    return wrapper

@class_decorator("greet")
class BaseClass:
    def greet(self):
        return "Hello from BaseClass"

@class_decorator("greet")
@class_decorator("farewell")
class ChildClass(BaseClass):
    def farewell(self):
        return "Goodbye from ChildClass"
```

Answer (1.5/1.5)
The decorator modifies the original method of a class if a method with that name is present in the class. Otherwise, it adds the method to the class.
- When `BaseClass` calls `greet()` it outputs `Modified behavior: Hello from BaseClass` because the method is already present.
- When `ChildClass` calls `greet()` it outputs `Modified behavior: Modified behavior: Hello from BaseClass` because the method has been inherited by the BaseClass and was modified originally by the decorator.
- When `ChildClass` calls `farewell()` it outputs `Modified behavior: Goodbye from ChildClass` because the method is already present but just in the ChildClass.

Question 10
Is it possible to use pybind11 to execute Python code from within C++ code?
What are possible use cases in the field of data science and scientific computing?

Answer (1.5/1.5)

````
Yes it is possible, I did it in the homework 3 for implementing
the Gauss Legendre integration method since the one I implemented
in C++ had some memory leaks. For doing so I defined its class in
a slightly different way: instead of just defining the constructor
in Python, I added a way to compute nodes and weights using NumPy.
The code was the following
```python
from numpy.polynomial import legendre
import numpy as np

# ----------------
# Python interface
# ----------------

# GaussLegendre is better implemented with NumPy
# Note: it can only compute nodes and weights in [-1,1]
class GaussLegendre(Quadrature):
    def __init__(self, a, b, nBins):
        Quadrature.__init__(self)
        # Call numpy.polynomial.legendre.leggauss
        self.nodes,self.weights = np.polynomial.leggauss(nBins -
1)
```
In the main.py, I then defined the function to compute an integral
in this way:
```python
# For Gauss-Legendre, the interval is fixed to [-1,1] due to
numpy.polynomial.legendre.leggaus's
# definition which was overloaded in the C++ method (see
IntegrationMethods.py for the overload)
@execution_time
def test_GaussLegendre_cpp(function, trueValue, nBins):
    GL = moduleC.GaussLegendre(-1, 1, nBins)
    integrationValue = moduleC.IntegrateGaussLegendre(function,
GL)
    error = abs(trueValue - integrationValue)
    return integrationValue, error
```
Where moduleC was the module I created for that functions and
classes.
As I did with my code, possible use cases in the field of data
science and scientific computing can be the implementation of some
methods that in C++ require more compilation-time or are more
complex. Moreover, Python libraries are made by experts, therefore
using them you're less prone to make mistakes.
````

Part 2: Programming exercise: https://github.com/pcafrica/advanced_programming_2023-2024/blob/main/exams/2024-02-13/part2.md