Advanced Programming Techniques
ExAdvPT
Winter Term 2018/2019

Assignment sheet A

Assignments that are marked with *StudOn submission* are **mandatory** and must be submitted via StudOn in time – please see the StudOn page for deadlines.

Warm-Up

Please be aware that during the exercises of AdvPT, we will give only support for the linux machines in the CIP-Pool. If you have never worked with the linux terminal, we recommend to go threw some tutorials frist.

Furthermore we expect you to know the difference between stack and heap, i.e. what the usage of malloc/free and new/delete means for the program.

And last but not least, you should know how to write Makefiles.

On StudOn you will find some links regarding those toppics, however feel free to explore this subjects on your own with your favourite search engine.

1 Coding: Range sum

Write a program that queries the user for two numbers and sums the numbers in that range (including the first number, excluding the last number).

2 Coding: Factorial

Write a program that prompts the user to enter a number and then uses a `for` loop to calculate the factorial of the given number and writes it to the standard output. Verify your program at least against the following *test cases*:

$$0! = 1 \quad ; \quad 1! = 1 \quad ; \quad 6! = 720 \quad ; \quad 12! = 479001600$$
$$13! = 6227020800 \quad ; \quad 21! \approx 5.1091e19$$
$$35! \approx 1.0333e40 \quad ; \quad -1! = ?$$

3 Punctuation

The program shall read a line from standard input and print the line to standard output without punctuation. The resulting program should be usable as a filter like this:

```
./punctuation < with_punct.txt > no_punct.txt
```

Have a look at the following STL Header: `<cctype>`

4 Matrix-matrix product

StudOn submission

Part 1: Matrix Class

Implement a matrix class in `Matrix.h` fulfilling the following requirements:

- Matrix entries shall be of `double` data type.
- Store the matrix data using a plain C-pointer member. Do **not** use `std::vector` or smart pointers.
- Matrix elements should be stored consecutively in a row wise order. That means all elements of the first row are placed consecutively in memory, directly followed by the elements of the second row, etc.
- Your matrix should be copyable and assignable. Follow the "Rule of three (five)".
- Implement the following arithmetic operators: +, -, *, +=, -=, *= with their usual semantics. Use `assert` to test that both matrices have matching sizes.
- Implement the (in)equality operators == and != .
- Your matrix class shall contain the two functions `rows()` and `cols()`, which return the number of rows and columns respectively.
- Matrix instances should be printable to `std::ostream`. In the output, columns should be separated by one ore more spaces/tabs, rows should be separated by newlines.
- Matrix instances should be readable from `std::istream`. The reader should at least support the format specified above.
- Compile your matrix class together with the provided testcase:
  `g++ -std=c++14 -Wall -pedantic MatrixTest.cpp`.
  Your implementation has to compile on the Linux CIP-Pool computers with these compiler flags and without warnings!

Make sure your implementation passes the provided testcase! Run your program also with `-fsanitize=address` and with `valgrind` to detect memory leaks or invalid accesses. Hint: In some environments `valgrind` may report false positives. You may want to check this by comparing the `valgrind` output of your program to the `valgrind` output of a minimal program containing only an empty main function.

Part 2: Multiply matrices read from standard input:

The purpose of the program is to read two matrices from standard input and write their product to standard output.

The program itself should be written to `MatrixProduct.cpp` and shall first read three integral numbers, in the following denoted s_1, s_2, and s_3. They specify the dimensions of the matrices, $m_1 \in \mathcal{R}^{s_1 \times s_2}, m_2 \in \mathcal{R}^{s_2 \times s_3}, m_3 \in \mathcal{R}^{s_1 \times s_3} = m_1 m_2$.

It then reads $s_1 \times s_2$ numbers that are used to populate m_1 row by row, and then $s_2 \times s_3$ numbers that are used to populate m_2 analogously. Handle all possible input errors and print a comprehensible error message in case of an incorrect input. The program now computes m_3 before it is eventually written to standard output using the format indicated above.

Store your solutions in files named `Matrix.h` and `MatrixProduct.cpp`, pack them into a zip file called `assignmentA4.zip` and submit them via StudOn. Please follow the file namings strictly since your submission is corrected automatically.

C++ Variables and Basic Types

5 Literal constants

StudOn submission

Determine the type of each of these literal constants:

(a) `-10` _____

(b) `-10U` _____

(c) `false` _____

(d) `-10.` _____

(e) `-10E-2` _____

(f) `'\t'` _____

6 Names

StudOn submission

Which, if any, of the following identifiers are valid?

(a) `int double = 3.14159;` ☐ correct ☐ invalid

(b) `bool catch-22;` ☐ correct ☐ invalid

(c) `float Float = 3.14F;` ☐ correct ☐ invalid

(d) `char _;` ☐ correct ☐ invalid

(e) `char 1_or_2 = '1';` ☐ correct ☐ invalid

7 Code Fragment: sum i

StudOn submission

Given the following program fragment, which values are printed?

```
int i = 100, sum = 0;
for( int i = 0; i != 10; ++i )
    sum += i;
std::cout << "  i: " << i << std::endl;
std::cout << "sum: " << sum << std::endl;
```

`i:` _____

`sum:` _____

8 Code Fragment: sum ii

StudOn submission

What is the output of the following program fragment?

```
const unsigned int length1 = 10U, length2 = 12U;
unsigned int sum = 0U;
for( unsigned int i = 0U; i < length1-length2; ++i )
    sum += i;
std::cout << "sum: " << sum << std::endl;
```

sum: _____

9 Code Fragment: sum iii

What is the output of the following program fragment?

```
unsigned int sum = 0U;
for( unsigned int i=100U; i>=0U; --i )
    sum += i;
std::cout << "sum: " << sum << std::endl;
```