Object-Oriented Programming for Scientific Computing

Dr. Linus Seelinger, Institute for Applied Mathematics, Heidelberg University

Note: I have moved the submission date to Wednesday, so I can give more detailed feedback during the subsequent tutorial.

Exercise 1: Basic Debugging 10 points

You can find three C++ files for download on the lecture website, namely `vector_broken.h`, `vector_broken.cc` and `testvector.cc`. These files contain several bugs. Try to find those bugs using GDB.

Download the three files and compile them with debug information:

```
g++ -std=c++11 -Og -g -o testvector vector_broken.cc testvector.cc
```

You can now start your program with GDB in TUI mode:

```
gdb -tui ./testvector
```

Entering `layout split` at the prompt (or `la sp`, almost all commands can be abbreviated) displays the assembly code equivalent of the program, and `layout src` (or `la sr`) removes the assembly window if you don't need / want it.

The most important GDB commands, with their abbreviation and possible arguments, are probably

- `break [b] <file:line, file:function>` (enable breakpoint at specified location, file may be omitted),
- `backtrace [bt]` (show hierarchy of called functions),
- `continue [c]` (continue running after break),
- `next [n]` (execute marked line),
- `print [p] <expression>` (print content of variable / object),
- `step [s]` (enter first function on marked line),
- `run [r] <arguments>` (start program with given arguments, if any),
- and `watch <expression>` (break if value of expression changes).

These commands are sufficient for this exercise, but you can find additional information at https://beej.us/guide/bggdb/#qref or any other GDB reference card on the internet. Note that `print` can be used to access members of objects, e.g. `p a.b` or `p a->b`, no need to step into some method for that — even if the member `b` is **`private`**.

Use GDB to find and correct the bugs in the provided source code, and document which bugs you found and how. There is a bug that is not covered by the tests, search for it. How would a test for this bug look like? What is problematic about the specific choice of test matrices in `testvector.cc`, what kind of bug are they unable to detect?

Note: the file `testvector.cc` does not contain bugs.

Exercise 2: C++ Quiz 5 points

On `https://cppquiz.org` you can find a quiz with C++ specific questions. In this exercise, answer the following questions:

Question 1: `https://cppquiz.org/quiz/question/197` (variable lifetime)

Question 2: `https://cppquiz.org/quiz/question/161` (Duff's Device)

Question 3: `https://cppquiz.org/quiz/question/9` (reference arguments)

Question 4: `https://cppquiz.org/quiz/question/113` (overload resolution)

Question 5: `https://cppquiz.org/quiz/question/5` (initialization order)

The questions are sorted (more or less) according to the structure of the lecture. For questions 1, 3, 4, and 5, write a short statement what information the question and its solution are trying to convey. Regarding question 2: inform yourself about the construct that is used. What is its purpose? Would you suggest using this in real-world code? Why, or why not?

Exercise 3: Rational Numbers 10 points

Write a class for rational numbers. The number should always be represented as a *fully reduced fraction* of the form

$$\frac{\text{numerator}}{\text{denominator}}$$

with denominator > 0.

(a) What is an appropriate data structure for rational numbers?

(b) Start by writing a function `int gcd(int,int)` (greatest common divisor), you will need it to reduce fractions.

- You can use the Euclidean algorithm to determine the greatest common divisor.
- For an algorithm see `https://en.wikipedia.org/wiki/Greatest_common_divisor`
- Implement this scheme as a recursive function.

(c) Write a class `Rational`, which represents a rational number. The constructor should have the numerator and the denominator as arguments. Be sure to check for valid input. In addition, the class has two functions `numerator()` and `denominator()` that return the values of the numerator and denominator. The class should have three constructors:

- a default constructor that initializes the fraction with 1,
- a constructor that initializes the fraction with a given numerator and denominator, and
- a constructor that initializes the fraction with a given whole number.

(d) Supplement the class with operators for `*=` `+=` `-=` `/=` and `==`.

(e) Use the newly implemented methods to implement free operators `*` `+` `-` `/`.

(f) Check your implementation using various test cases. Initialize three fractions

$$f_1 = -\frac{3}{12}, \quad f_2 = \frac{4}{3}, \quad f_3 = \frac{0}{1}.$$

Test the operators with the following examples:

$$f_3 = f_1 + f_2, \quad f_3 = f_1 \cdot f_2, \quad f_3 = 4 + f_2, \quad f_3 = f_2 + 5, \quad f_3 = 12 \cdot f_1, \quad f_3 = f_1 \cdot 6, \quad f_3 = \frac{f_1}{f_2}.$$

Print the result after each operation. The corresponding solutions are:

$$\frac{13}{12}, \quad -\frac{1}{3}, \quad \frac{16}{3}, \quad \frac{19}{3}, \quad -\frac{3}{1}, \quad -\frac{3}{2}, \quad -\frac{3}{16}.$$

Exercise 4: Farey Sequences 10 points

A Farey sequence F_N of degree N (or: the Farey fractions of degree N) is an ordered set of reduced fractions

$$\frac{p_i}{q_i} \qquad \text{with} \qquad p_i \leq q_i \leq N \qquad \text{and} \qquad 0 \leq i < |F_N|$$

and

$$\frac{p_i}{q_i} < \frac{p_j}{q_j} \qquad \forall\, 0 \leq i < j < |F_N|.$$

Use the class `Rational` from the previous exercise to write a function

```
void Farey(int N)
```

which calculates the Farey fractions up to degree N and prints the resulting Farey sequences up to degree N on the screen.

Algorithm: The sequences can be computed recursively. The first sequence is given by

$$F_1 = \left(\tfrac{0}{1}, \tfrac{1}{1}\right)$$

For a known sequence F_N one can get F_{N+1} by inserting an additional fraction $\frac{p_i + p_{i+1}}{q_i + q_{i+1}}$ between two consecutive entries $\frac{p_i}{q_i}$ and $\frac{p_{i+1}}{q_{i+1}}$ if $q_i + q_{i+1} = N + 1$ holds for the sum of denominators.

Example: Determining F_7 from F_6 results in the following construction:

$$F_6 = \left(\underbrace{\tfrac{0}{1}, \tfrac{1}{6}}_{\frac{1}{7}}, \tfrac{1}{5}, \underbrace{\tfrac{1}{4}, \tfrac{1}{3}}_{\frac{2}{7}}, \underbrace{\tfrac{2}{5}, \tfrac{1}{2}, \tfrac{3}{5}}_{\frac{3}{7} \text{ and } \frac{4}{7}}, \underbrace{\tfrac{2}{3}, \tfrac{3}{4}}_{\frac{5}{7}}, \tfrac{4}{5}, \underbrace{\tfrac{5}{6}, \tfrac{1}{1}}_{\frac{6}{7}} \right)$$

The new elements are:

$$\tfrac{0+1}{1+6} = \tfrac{1}{7} \;;\; \tfrac{1+1}{4+3} = \tfrac{2}{7} \;;\; \tfrac{2+1}{5+2} = \tfrac{3}{7} \;;\; \tfrac{1+3}{2+5} = \tfrac{4}{7} \;;\; \tfrac{2+3}{3+4} = \tfrac{5}{7} \;;\; \tfrac{5+1}{6+1} = \tfrac{6}{7}$$

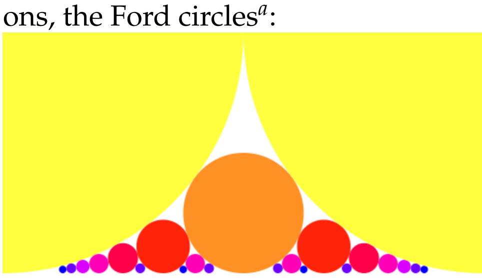
The sorted sequence then is:

$$F_7 = \left(\tfrac{0}{1}, \boldsymbol{\tfrac{1}{7}}, \tfrac{1}{6}, \tfrac{1}{5}, \tfrac{1}{4}, \boldsymbol{\tfrac{2}{7}}, \tfrac{1}{3}, \tfrac{2}{5}, \boldsymbol{\tfrac{3}{7}}, \tfrac{1}{2}, \boldsymbol{\tfrac{4}{7}}, \tfrac{3}{5}, \tfrac{2}{3}, \boldsymbol{\tfrac{5}{7}}\, \tfrac{3}{4}, \tfrac{4}{5}, \tfrac{5}{6}, \boldsymbol{\tfrac{6}{7}}, \tfrac{1}{1} \right)$$

For checking:
The Farey sequences up to degree 6

$$\begin{aligned}
F_1 &= \left(\tfrac{0}{1}, \tfrac{1}{1}\right) \\
F_2 &= \left(\tfrac{0}{1}, \tfrac{1}{2}, \tfrac{1}{1}\right) \\
F_3 &= \left(\tfrac{0}{1}, \tfrac{1}{3}, \tfrac{1}{2}, \tfrac{2}{3}, \tfrac{1}{1}\right) \\
F_4 &= \left(\tfrac{0}{1}, \tfrac{1}{4}, \tfrac{1}{3}, \tfrac{1}{2}, \tfrac{2}{3}, \tfrac{3}{4}, \tfrac{1}{1}\right) \\
F_5 &= \left(\tfrac{0}{1}, \tfrac{1}{5}, \tfrac{1}{4}, \tfrac{1}{3}, \tfrac{2}{5}, \tfrac{1}{2}, \tfrac{3}{5}, \tfrac{2}{3}, \tfrac{3}{4}, \tfrac{4}{5}, \tfrac{1}{1}\right) \\
F_6 &= \left(\tfrac{0}{1}, \tfrac{1}{6}, \tfrac{1}{5}, \tfrac{1}{4}, \tfrac{1}{3}, \tfrac{2}{5}, \tfrac{1}{2}, \tfrac{3}{5}, \tfrac{2}{3}, \tfrac{3}{4}, \tfrac{4}{5}, \tfrac{5}{6}, \tfrac{1}{1}\right).
\end{aligned}$$

There is a beautiful illustration of these fractions, the Ford circles[a]:



[a] see https://en.wikipedia.org/wiki/Ford_circle