HPC - Skopje

date: Nov. 7, 2011

1. The goal of this small project consists in autonomous design an implementation of a programm. You should take into account the following items:

   - include source and header files of a given library into your project and **use the functions** therein,
   - program functions for your own library (source and header files) wrt. the given task,
   - work with input and output files,
   - test your modules/functions carefully,
   - document your code, explain the parameter lists in the header files and what the function is doing. I use the documentation style from doxygen, see also the brief introduction (Download).

   What's the functionality of your code?

   (a) Read discrete x-values from the given ASCII-files (use module *file_io*).

   (b) Calculate function values $f(x)$, discrete derivatives $f'(x)$ and discrete primitives $F(x) = \int\limits_0^x f(x)dx \quad$ numerically for a given function f , $\forall x$.
   For this purpose, you should write a module (header and source files) for calculus wherein the function under consideration is appears as function pointer in the parameter lists.

   (c) The resulting values for all x have to be stored in three separate ASCII-files. (This allows you to read the data into Matlab/Octave via `importdata` and to visualize them)

   (d) The module *file_io* has to be used and to be included in your project.

   Mathematical functions:

   $$p(x) = 4x^3 + 3x^2 \quad ; \qquad q(x) = \mathrm{e}^{-x}\sin(40x) \quad ; \qquad s(x) = x\sin(40/x)$$

   Files with input values x: *input_1_x.txt* and *input_2_x.txt*.

   Items a)-d) should be used with 2 different input data and 3 different functions, i.e., it might be of advantage to combine a)-d) into one function.

You will use the following formulas.

Numerical Integration

The numerical integration of $f(x)$ is done approximately, e.g., via the Riemann sums

$$\int_a^b f(x)dx \approx F_n(x) := \sum_{j=1}^{n} f(a+jh)\cdot h \tag{1}$$

with n denoting the number of equidistant subintervals of $[a,b]$ resulting in an subinterval length of $h = \frac{b-a}{n}$.

The accuracy of the numerical integration depends obviously on the number of subintervals n, i.e., **you have to increase** n ***while*** $|F_n(x) - F_{n-1}(x)|$ is larger than a given accuracy ε.

Suggestion for the parameter list: f, a, b, ε.
Validate your function with test data that can be checked easily!

Numerical Differentiation

The numerical differentiation of a function f in x can be approximated by the central difference

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad . \tag{2}$$

The stride h is given and $h = 0.1$ is a good starting guess for our data.

Once again the numerical differentiation have to accurate enough, i.e., you have to **decrease** $h > 0$ ***while*** the difference between the two last approximations of $f'(x)$ is larger than a given accuracy.

Suggestion for the parameter list: f, x, ε.
Validate your function with test data that can be checked easily!

2. Design and implement a class for rational numbers (I called it `bruch`) containing all four basic arithmetic operations such that the following code can be used without any changes (you have to include the appropriate header file(s)).

```
1   int main()
2   {   // (enumerator, denominator) [germ.:(Zaehler, Nenner)] as Parameter list
3       const bruch a(4,3), b(7,-2);
4             bruch c(a+b), d(-3,0);
5
6       d = a*b;
7       cout << "a = " << a << endl;
8       cout << "b = " << b << " has enumerator " << b.getZaehler() << endl;
9       cout << "c = " << c << endl;
10      cout << "d = " << d << endl;
11      cout << "e = " << d/(b-a)-a*c << endl;
12
13      return 0;
14  }
```

Implement all methods/operators by yourself although you could use also the assignment operator and constructors implicitly generated by the compiler.

- Distinguish between declaration and implementation (header and source file).
- Deactivate first the lines 4–11 in the code above and start with declaration/implementation of parameter constructor and standard constructor (which fraction is equivalent to 0 ?).
- Line 4 requires an additional copy constructor and the addition operator `operator+` .
- Line 6 requires the assignment operator `operator=` and multiplication operator `operator*`.
- Lines 7–10 use the output operator `operator<<` that uses your class as parameter, i.e., it is not a method of your class. Here, you should use the access methods (as `getZaehler()`) appropriately
  Check the correctness of the results after arithmetics!
- More operators are needed in line 11.

After implementation of all basic items above we think about improvements and protection of the class:

- What happens if the denominator is equal to 0?
  That have to be taken into account in the constructors and arithmetic operators.
- The instance `b(7,-2)` would be stored as $\frac{7}{-2}$. Change the parameter constructor (and operators) such that the fraction possesses always a positive denominator.
- Instance d has the value $\frac{-28}{6}$ and this fraction can be reduced to $\frac{-14}{3}$.
  Implement this reduction as a `private` method by first determining the ggT (greatest common divisor) followed by using this number. Reduce the fraction in all methods and constructors wherein it makes sense but do it in a smart way.

- Take care for the division by 0 and handle that case.

∗ How can you check for leaving the admissible range of the chosen data type in temporary results? Can you catch these cases without using exception handling?

3. Rewrite your class `bruch` from exercise 2 as template class. Test your class with template parameters `long long int`, `short int`, `signed char` similar to the main program in exercise 3 2.

4. Define an STL vector with your class `bruch` as type of the elements and read its data from file input_22_a.txt. Adapt module *file_io* for that purpose, i.e., by rewriting the functions therein as template functions. The data in the file are arranged such that enumerator and denominator (Zähler, Nenner) of the first rational are followed by enumerator and denominator of the second rational, etc.. (Hint: Your class will need an input operator.)

   Use the **STL algorithms** and the methods of your container `vector` to finish the following subtasks (always check for the correct result).

   (a) Output of the read vector a via a separate function `operator<<` .

   (b) Copy vectors a onto a second vector b [STL?].

   (c) Sort vector b in increasing order [STL].

   (d) Remove all multiple elements from b [STL].

   (e) Copy all positive elements from b into a vector c [STL].

   (f) Determine the largest and the smallest element of a [STL].

   (g) Determine the largest and the smallest element of a with respect to its abolute value and remove the two elements [STL].

   (h) Sort vector c in decreasing order [STL].

   (i) Count the number of elements in a which are equal $\frac{1}{4}$ [STL].

   Hints:

   - Try the above subtasks first with `vector<int>` initialized with appropriate data. If that works than solve them with your class `bruch` (even with the template class).
   - Read the description (and examples!!) of the following algorithms/methods in www and use them whenever possible: `unique`, `copy`, `remove_copy_if`, `find`, `remove`, `remove_if`, `sort`, `max`, `max_element`, `vector<T>::erase` .
   - Add the needed comparison operators to your class `bruch` (needed for sorting and test for equality).