Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

Informatik für Mathematiker und Physiker HS16

Exercise Sheet 12

Submission deadline: 15:15 - Tuesday 13th December, 2016

Course URL: http://lec.inf.ethz.ch/ifmp/2016/

Assignment 1 – Understanding Structs and Classes (4 points)

Consider the following definitions:

```
struct A {
  int a;
  double b;
  int c;
};

A str = {1, 1.5, 2};
int arr[] = {1, 1.5, 3};
int* a = arr;
```

```
class B {
public:
  B() : n(0) {};

  // PRE: ...
  // POST: ...
  void add (const int i) {
    ++n;
    std::cout << i << " ";
  }

  // PRE: ...
  // POST: ...
  int get () const {
    return n;
  }

private:
  int n;
};
```

```
class C {
public:
  C () {
    for (int i = 0; i < 128; ++i)
    arr[i] = 0;
  }

  // PRE: ...
  // POST: ...
  void add (const char c) {
    ++arr[c];
  }

  // PRE: ...
  // POST: ...
  int get (const char c) const {
    return arr[c];
  }

private:
  int arr[128];
};
```

a) Left box: For each of the provided expressions state their *type* and *value*! [based on: Exam Summer 2015, ex. 2]

i) `str.a * str.b`

ii) `str.b == arr[1]`

iii) `str.a * str.b / str.c`

iv) `arr[str.a] / str.c`

v) `*a / 2 - str.b`

b) Central box: What does the following code output?

```
int main () {
  B hi;
  B hello;
  hi.add(1); hi.add(2); hello.add(3);
```

```
  hi.add(4); hello.add(5);
  std::cout << hi.get() << " " << hello.get() << "\n";
  return 0;
}
```

c) Right box: Determine PRE- and POST-conditions for the methods `add` and `get` of class `C`!

This exercise can be handed in via Codeboard! However, if you prefer, you can also hand in your solutions on paper as before.

Submission: https://codeboard.ethz.ch/ifmp16E12T1

Assignment 2 – Writing Classes (4 points)

Each of the Codeboard-templates already contains a `main`-function with which your code shall work!

a) Write a class `Averager` that computes averages of given values of type `double`. It shall offer at least the members `add_value` (which provides another value of type `double` to the class) and `average_value` (which returns the average of the values provided so far as a `double`). [based on: Exam Summer 2012, ex.6]

I/O-Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2016/codeboard.html)

```
3
2 6 3
2 4 3.66667
```

Submission: https://codeboard.ethz.ch/ifmp16E12T2a

b) Write a class `Vector` that represents vectors in $\mathbb{R}^3$ (modelled with entries of type `double`). Your class shall provide the `operator+` for adding two vectors (which returns the sum as a new `Vector`), an `operator*` for multiplication with scalars of type `double`, as well as a member `get(char i)` to obtain the coordinates (where `i` is either `'x'`, `'y'` or `'z'`). It shall be constructible by passing the `x`, `y`, and `z` coordinates. Also provide `operator<<` and `operator>>` to output and input a vector with components `x`, `y`, `z` as `(x, y, z)`. You can of course implement more functions!

I/O-Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2016/codeboard.html)

```
sum
(1.5, 2.5, 3.5)
(4, 5, 6)
(5.5, 7.5, 9.5)
```

Submission: https://codeboard.ethz.ch/ifmp16E12T2b

Assignment 3 – Normalizing `rational` (4 points)

[Skript-Aufgabe 146]

Given the `struct rational` from the lecture, we want to have a function that normalizes a rational number, i.e. transforms it into the unique representation in which numerator and denominator are

relatively prime, and the denominator is positive. (0 shall be normalized to $\frac{0}{1}$.) For example,

$$\frac{21}{-14} \quad \mapsto \quad \frac{-3}{2}$$

There are two natural versions of this function:

```
// POST: r is normalized
void normalize (rational& r);

// POST: return value is the normalization of r
rational normalize (const rational& r);
```

The Codeboard submission link already contains the struct `rational` from the lecture. Implement one of the two functions given above. Then write a `main`-function which asks the user for a `rational` number and then outputs the normalized version of it.

Hint: You may use the function `gcd` from lecture 9, page 53, modified for arguments of type `int`.

I/O-Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2016/codeboard.html)

```
-33/-99
1/3
```

Submission: https://codeboard.ethz.ch/ifmp16E12T3

Assignment 4 – Extended Stack (4 points)

[based on: Skript-Aufgabe 159]

Extend the class `stack` from the lecture such that it provides the following functionality in addition to `push`, `pop`, `top`, `empty`:

```
// POST: number of elements in *this is returned
unsigned int size () const;

// POST: remove the element i wherever it occurs in *this. If
//       it doesn't occur, then *this is not modified.
void remove (const int i);

// POST: returns true if and only if s1 and s2 contain the same
//       keys in the same order
bool operator== (const stack& s1, const stack& s2);
```

The Codeboard submission link already contains a template which provides a `main`-function that tests your code, and which contains the code from the `stack` which you can modify and extend!

I/O-Examples (Explanation: http://lec.inf.ethz.ch/ifmp/2016/codeboard.html)

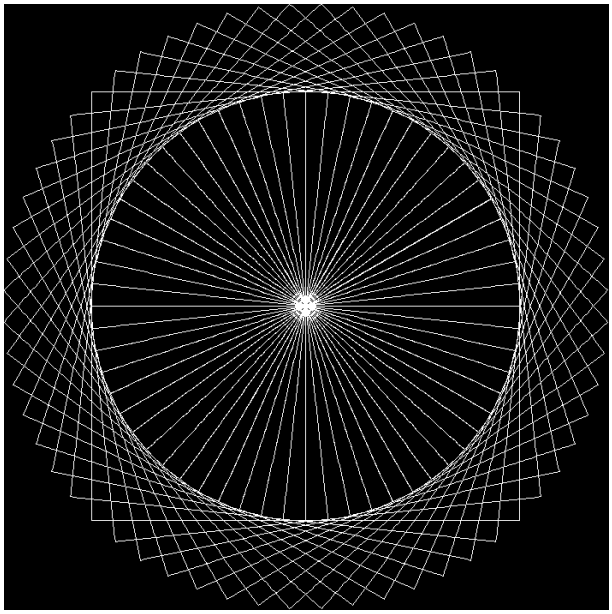
```
size
3 2 3 2 3
0
5
```

Submission: https://codeboard.ethz.ch/ifmp16E12T4

Challenge – LO Drawings! (8 points)

This challenge lets you design and draw turtle graphics images, using the simple language LO that is inspired by the classic programming language LOGO. Let us start with an example of a program in LO:

```
60(
  4(
    50(F)
    90(+)
  )
  6(+)
)
```



In LO, `n(...)` tells the turtle to repeat the commands in brackets `n` times. The "inner loop"

```
4(
  50(F)
  90(+)
)
```

therefore draws a square, and the whole program draws 60 squares, forming a circular pattern:

Here is an EBNF for the language LO:

```
program = repetitions "(" { statement } ")"
statement = command | program
command = "+" | "-" | "F" | "f" | "[" | "]"
repetitions = unsigned int
```

Command `F` tells the turtle to go one step forward, `+` to turn by one degree in counterclockwise direction. Command `f` corresponds to the turtle's jump command, `-` turns clockwise, `[` and `]` save and restore the current state.

Write a program `miracle.cpp` that transforms LO programs into turtle graphics by performing the following tasks:

1. Parse a given LO program and compute its "value" which is the corresponding sequence of turtle commands. For example, the program `4(f2(+))` has value `f++f++f++f++` (which doesn't draw anything, unfortunately...).
2. Let the turtle interpret a given sequence of turtle commands, producing the actual drawing.

All submissions (or a large selection thereof if there are too many) will be made into a video to be shown during a later lecture. Handing in the pictures to us works just as for the regular exercises in Codeboard. You can also hand in multiple submissions!

Click on the **Submission**-link at the bottom of this box to open Codeboard.

Submission: https://codeboard.ethz.ch/ifmp16E12LO