technische universität dortmund

Department of Computer Science
Chair 11 / Algorithm Engineering

Dr. Carsten Gutwenger

Winter 2011/12

Object-oriented Programming
Assignment Sheet No. 6

Date: November 29

Exercise 6.1 (Function Overloading)

Implement overloaded `power()` functions, which compute `a` to the power of `b` for various data types. For example, you could implement the following functions:

- `long long power(int a, unsigned int b);`
- `float power(float a, unsigned short b);`
- `string power(string a, unsigned int b);`
  e.g. `"xyz"` to the power of 3 is `"xyzxyzxyz"`

Write a test program that demonstrates your implementations.

Exercise 6.2 (Function Overloading and Number Types)

Implement overloaded `print_vector()` functions, which print a vector of different integer and floating point types. For floating point types, it should also be possible to specify the precision. For example, you could implement the following functions:

- `void print_vector(const vector<int> &v);`
- `void print_vector(const vector<float> &v);`
- `void print_vector(const vector<float> &v, int precision);`

Write a test program that demonstrates your implementations.

Exercise 6.3 (Function Overloading and Const References)

Implement overloaded functions that compute the *scalar product* (also known as *dot product*) of two vectors. Provide functions for `int` and `float` vectors. Write a test program that demonstrates the functions.

The scalar product of two vectors $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ is defined as $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$. Please note that both vectors must have the same length.

Exercise 6.4 (Recursive Functions)

It is also allowed that a function calls itself. Such a function is called a *recursive* function. Of course, if the function would call itself over and over again, it would never terminate. Therefore, there must be some kind of *termination condition* in the function to avoid this.

Euclid's algorithm for computing the *greatest common divisor* (denoted by $\gcd(a,b)$ in the sequel) of two integer numbers a and b can easily be described as a recursive function. It is based on the following, simple recursive formula:

$$\gcd(a,b) = \begin{cases} a & \text{if } b = 0 \\ \gcd(b,\ a \bmod b) & \text{otherwise} \end{cases}$$

In this case, $b =$ is the termination condition, since we do not call gcd recursively once $b = 0$.

Implement a recursive function `int gcd(int a, int b)` applying Euclid's algorithm and write a test program that lets the user enter two integers and computes their greatest common divisor using `gcd`.