Dirk Praetorius,
Michele Ruggeri

Wintersemester 2017/18
18./19.01.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1. Explain the differences between `public`-, `private`-, und `protected`-inheritance on the basis of a suitable exapmle.

Aufgabe 11.2. Implement the class `Person` which contains the members `name` and `address`. Derive from `Person` the class `Student`, that contains the additional data fields `matriculationNumber` and `study`. Derive from `Person` also the class `Worker` that contains the additional data fields `salary` and `work`. Write `set/get` functions, constructors, and destructors for all classes. Moreover, write a main progam to test your implementation!

Aufgabe 11.3. Implement the method `print` for the basis class `Person` from Exercise 11.2. The method should print to the screen name and address of a person. Redefine this function for the derived classes `Student` and `Worker` so that also the additional data fields of these classes are printed. Moreover, write a main programm to test the `print`-methods of the different classes.

Aufgabe 11.4. Derive the class `SquareMatrix` from the class `Matrix` from the lecture. This class is used to store square matrices and should contain all functionalities from the basis class `Matrix`. Test your implementation accurately!

Aufgabe 11.5. Consider the class `Matrix` and the derived class `SquareMatrix`.Implement the method `computeLU` that computes the LU-factorization in the class `SquareMatrix`. The method returns a matrix $R \in \mathbb{R}^{n\times n}$ of type `SquareMatrix` whose upper and lower triangular parts contain the entries of L and U.The diagonal of L does not need to be stored. Why? Not every matrix $A \in \mathbb{R}^{n\times n}$ has a normalized LU-factorization $A = LU$, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

In the case that such a factorization exists, it holds that

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{for } i = 1, \dots, n, \quad k = i, \dots, n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{for } i = 1, \dots, n, \quad k = i+1, \dots, n, \\ \ell_{ii} &= 1 \quad \text{for } i = 1, \dots, n, \end{aligned}$$

which can be verified by using the formula for the matrix-matrix multiplication. All the remaining entries of $L, U \in \mathbb{R}^{n\times n}$ are trivial. The determinant of a matrix $A \in \mathbb{R}^{n\times n}$ can be computed with the normalized LU-factorization. It holds $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Extend the class `SquareMatrix` by the method `det` which computes and returns the determinant. The matrix A should not be overwritten. Test your implementation appropriately!

Aufgabe 11.6. Extend the class `SquareMatrix` by the method `solve`, which computes the solution of a linear system of equations of the form $Ax = b$ according to the following strategy: First, compute the LU factorization $A = LU$. Then, then solve the system $Ly = b$ and finally $Ux = y$. Test your implementation accurately!

Aufgabe 11.7. What is the computational cost of your implementation of the method which solves a linear system of equation via a LU factorization (Exercise 11.6)? Use the $\mathcal{O}$-notation to write the result and justify your answer.

Aufgabe 11.8. What is the output of the following programme? Explain why!

```
#include <iostream>
using std::cout;
using std::endl;
class BasisClass {
    protected:
        int N;
    public:
        BasisClass() {
            N = 0;
            cout << "Standard constr. BasisClass" << endl;
        }
        BasisClass( int n) {
            N = n;
            cout << "Constr. BasisClass, N = " << N << endl;
        }
        ~BasisClass(){
            cout << "Destr. BasisClass, N = " << N << endl;
        }
        BasisClass( const BasisClass& rhs) {
            N = rhs.N;
            cout << "Copy constr. BasisClass" << endl;
        }
        BasisClass& operator=(const BasisClass& rhs) {
            N = rhs.N;
            cout << "Assignment operator BasisClass" << endl;
            return *this;
        }
        int getN() const { return N; }
        void setN( int N ) { this->N = N; }
};

class Derived : public BasisClass {
    public:
        Derived(){
            cout << "Standard constr. Derived" << endl;
        }
        Derived( int n):BasisClass(n) {
            cout << "Constr. Derived, N = " << N << endl;
        }
        ~Derived() {
            cout << "Destr. Derived, N = " << N << endl;
        }
        Derived( const Derived& rhs) {
            N = rhs.N+7;
            cout << "Copy constr. Derived" << endl;
        }
        Derived& operator=(const Derived& rhs) {
            N = rhs.N;
            cout << "Assignment operator Derived" << endl;
            return *this;
        }
```

```
};

Derived foo(Derived X){
    Derived tmp(5);
    tmp.setN(X.getN()*X.getN());
    return tmp;
}

int main() {
    Derived ah(10);
    {
        Derived gg(13);
        BasisClass bs;
        BasisClass mr=bs;
        ah=gg;
    }
    ah=foo(ah);

    return 0;
}
```