Tut 9 (Tutor) 2nd Edition

Monday, 17. December 2018 20:51

0. ORGANISATORISCHES

- 2. Block-Test
  → Start: Mo. 07.01.19 10^{00}
  → Ende: So. 13.01.19 23^{59}

> **I.V.**
> - Abstrakte Klassen
> - I/O in Dateien
> - Vektoren

1. ABSTRAKTE KLASSEN

- Wortwörtlich zu abstrakt
- Tut. Bsp zu umfangreich & nicht anschaulich
  → VL.-Bsp auch zu chaotisch
- Eigenes Bsp. zur Herleitung einer abstrakten Klasse
- Wir erstellen eine Klasse Form

```
class Form {
public:
    virtual double berechneUmfang() = 0;
};
```

- und eine Klasse Kreis & Rechteck, die von Form erbt

```
class Kreis : public Form {
public:
    double radius;
    double berechneUmfang(){
        return 2* M_PI * radius;
    }
};

class Rechteck: public Form{
    double a,b;
    double berechneUmfang{
        return 2* a * b;
    }
};
```

- Man sieht, dass es nur sinnvoll ist, berechneUmfang in Kindklassen (bestimmte, definierte Formen) zu implementieren, nicht aber in Form
- Objektinstanzen von 'Form' sind auch nicht sinnvoll
  → Instanziierung unterbinden
  → virtual ... = 0();

> **ABSTRAKTE KLASSEN**
>
> `virtual double abstrakteMethode () = 0;`
>
> Keine Implementierung
> 'rein virtuell'
>
> ... sind Klassen mit mind. einer rein virtuellen Methoden
>
> → Klasse nicht instanziierbar
> → Sinn: Vorgabe von zu impl. Methoden für Kindklassen
> ↳ Interfaces (Schnittstellen)

2. VEKTOREN

- In HA5 haben wir vector.c implementiert
  → Array, dass mit Inhalt befüllt wird und dynamisch vergrößert wurde, wenn kein Platz mehr
  → Qualvolle Angelegenheit: malloc neues Array, mit altem Inhalt befüllen, freigeben
- In C++ kann man die Klasse vektor nutzen
- Vektor ist ein Array, das automatisch wächst & schrumpft
  → Elemente reinschieben, auslesen & löschen

> **KLASSE VECTOR IN C++**
>
> ... bietet automatisch wachsend/schrumpfendes Array
>
> `#include <vector>`
> ↱ Template: beliebiger, gewünschter DT
> `vector <double > vec;`

- <typ> ist ein 'Template'. Hier kann man einen gewünschten DT eintragen
  → Verwendet in dynamic_cast <Student *> (s1);
- Syntax:

```
vec.push_back(3.14f);      // 3.14 hinzufügen
double x = vec.at(0);      // 0. Element auslesen
```

- Mehr Methoden & Bedienungsanleitung in der HA
  → HA: Studierendendatenbank mit vector

3. EIN- UND AUSGABE IN DATEIEN

- endlich praktisch: Kommunikation mit der Außenwelt
- Bsp. 1: HA09, Vektoraufgabe
  → Wir sind 'ne Uni & verwalten Studierende
  → Benutzer kann Studierende aus einer .txt-Datenbank einlesen
  ↳ +: neue hinzufügen, nach Nr. suchen, Löschen, in DB speichern
  → C-Version öffentlich!
- Bsp. 2 Guitar Tuner
  → Liest wave-Datei Bit für Bit ein
  → Berechnet Frequenzspektrum der Wave und speichert Ergebnis in txt
  → Einlesen in MATLAB z.B., Peak auswerten
- Bsp. 3 Fraktal - Generator
  → Nicht nur .txt. Auch Bilder wie .bmp möglich
  → Gemeinsam kompilieren
- Wie liest und schreibt man (Text)-Dateien?

> **I/O IN DATEIEN**
>
> ... erfolgt in filestreams
>
> `#include <fstream>`

- Streams sind uns bekannt:
  → stringstream für String 'bauen'
  → I/O-Stream für Konsolen Ein- und Ausgabe

Schreiben mit ofstream

```
ofstream outFile("test.txt");
outfile << "bla" << endl;
outfile.close();
```

- Beim lesenden Zugriff wird ifstream verwendet:

Lesen mit ifstream

```
ifstream inFile;
inFile.open("bla.txt");
string in;
getline(inFile, in);
double d;
inFile >> d;
```

- Bsp. fileio1.cpp:
  → beispiel.txt beinhaltet String & paar Zahlen
  → Werden eingelesen mit ifstream und auf Konsole ausgegeben
- Bsp. fileio2.cpp:
  → beispiel.txt wird kopiert
  → while (infile) erfüllt wie es Inhalt gibt
  → mit ofstream::app wird zum bestehenden Inhalt angefügt
- Aufg. 2a vervollständigen
  → Jedem ASCII-Char ist einem Zahlenwert 0-127 zugeordnet
  ↳ Bekannt aus Caesar-Aufgabe

```
ifstream file;
string art;
file.open(filename);
while (file){
    int i;
    file >> i;
    art += (char) i;
}
file.close();
return art;
```

- Merry X-Mas!