Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. Dr. h.c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Übung 06: Informationstechnik (IT)

Daniel Grimm

Teil 1: Besprechung Aufgabenblatt 6

Aufg. 6.01: Verständnisfragen Lsg. (1)

a) Der Operator `new` erwartet als Operand
   i. den Namen des Objekts, das dynamisch erzeugt werden soll. Falsch
   ii. den Typ des anzulegenden Objekts. Richtig
   iii. die Größe des Objekts in Anzahl Bytes. Falsch

b) Mit `new` reservierter Speicher wird mit dem Operator `delete` wieder freigegeben.

c) Wird für ein dynamisch reserviertes Objekt der `delete`-Operator nicht aufgerufen, so wird der dynamisch reservierte Speicherbereich
   i. nicht wieder durch das Programm freigegeben. Richtig
   ii. automatisch freigegeben, wenn er nicht mehr verwendet wird. Falsch

Aufg. 6.01: Verständnisfragen Lsg. (2)

d) Jedem Objekt einer Basisklasse kann ein Objekt einer Kindklasse zugewiesen werden und umgekehrt. → Falsch

e) Polymorphe Klassen werden in C++ mit Hilfe von `virtual`-Methoden implementiert.

f) Die C++ Standard Template Library (STL) bietet „Container“, welche nur bei bestimmten Datentypen angewendet werden können. → Falsch

g) Nennen Sie 3 Klassen der STL.
map, list, stack, queue, ….

Aufg. 6.01: Verständnisfragen Lsg. (3)

h) Nennen Sie einen Vorteil und einen Nachteil einer verketteten Liste gegenüber einem dynamisch angelegten Array.
Eine verkette Liste kann einfach in der Größe dynamisch verändert werden, wohingegen ein Array weniger Speicherplatz belegt.

i) Beim Einfügen und Löschen eines Elements in einer verketteten Liste werden keine Elemente verschoben, sondern lediglich Zeiger versetzt.

j) Erklären Sie kurz die Begriffe FIFO und LIFO in Bezug auf die STL.
FIFO-Prinzip (First In, First Out): Lesen nur in selber Reihenfolge wie beim Schreiben
LIFO-Prinzip (Last In, First Out): Lesen nur in umgekehrter Reihenfolge wie beim Schreiben

Aufg. 6.01: Verständnisfragen Lsg. (4)

k) Welches Prinzip wird in einer Warteschlange (Queue) verwendet? Beschreiben Sie dieses kurz.

FIFO Prinzip (First-In First-Out), in einer Warteschlange kann eine beliebige Anzahl von Objekten gespeichert werden, die gespeicherten Objekte können nur in der gleichen Reihenfolge wieder gelesen werden, wie sie gespeichert wurden.

- Aufg. 6.01: Verständnisfragen

Aufg. 6.02: OOP, Klassendefinition und Vererbung

Lsg. (1)

a) Angenommen die folgenden Klassen sind in der Header-Datei **"myClasses.h"** definiert:

```
class Polynom {
  int x;
public:
  Polynom();
  virtual int calc( int a );
};
```

```
class Add : public Polynom {
  int y;
public:
  Add();
  int calc();
};
```

Für ein Objekt **obj** der Klasse **Add** ist dann folgende Anweisung zulässig:

```
int res = obj.calc( 2 );
```

→ Falsch

Falsch:

1. Polymorphie funktioniert nur für Methoden, die die selbe Parameterliste und den selben Rückgabewert haben.
2. In der Klasse **Add** wird die geerbte Funktion **calc(int)** von der eigenen Funktion **calc()** überdeckt. Deshalb ist der Aufruf mit Parameter ungültig.

Info: Möglich wäre hingegen **int res = obj.Polynom::calc(2);**

Aufg. 6.02: OOP, Klassendefinition und Vererbung

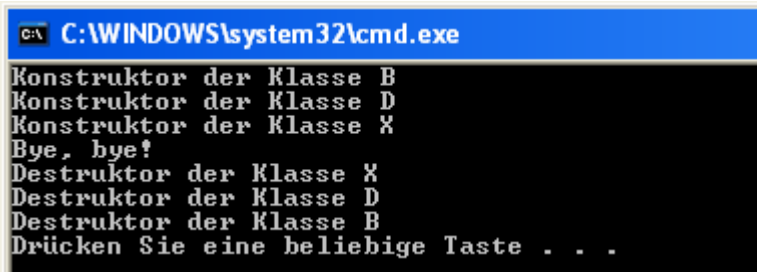
Lsg. (2)

b) Was gibt das folgende Programm auf dem Bildschirm aus?

```
#include <iostream>

using namespace std;

class B {
public:
  B() { cout << "Konstruktor der Klasse B \n"; }
  ~B() { cout << "Destruktor der Klasse B \n"; }
};

class D : public B {
public:
  D() { cout << "Konstruktor der Klasse D \n"; }
  ~D() { cout << "Destruktor der Klasse D \n"; }
};

class X {
private:
  D d;
public:
  X() { cout << "Konstruktor der Klasse X \n"; }
  ~X() { cout << "Destruktor der Klasse X \n"; }
};

int main() {
  X xObj;
  cout << "Bye, bye!" << endl;
  return 0;
}
```

Beim Anlegen vom Objekt **xObj** der Klasse **X** gilt die folgende Reihenfolge:
1) Anlegen des Objekts **d**
1.1) Aufruf von **B()** der Basisklasse **B**
1.2) Aufruf von **D()** der Klasse **D**
2) Aufruf von **X()** der Klasse **X**

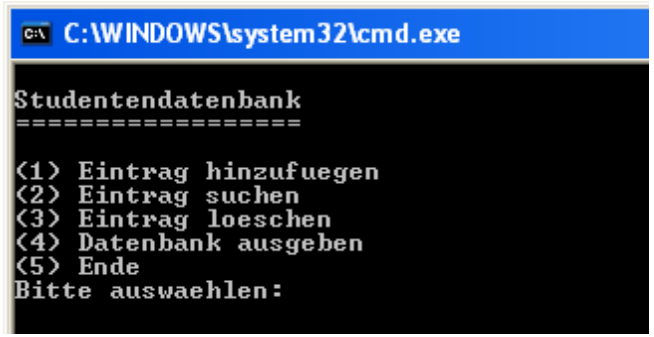
Beim Beenden des Programms wird der Destruktor aufgerufen und dann der Speicher für die angelegten Objekte freigegeben.
Es gilt die umgekehrte Reihenfolge wie beim Anlegen!

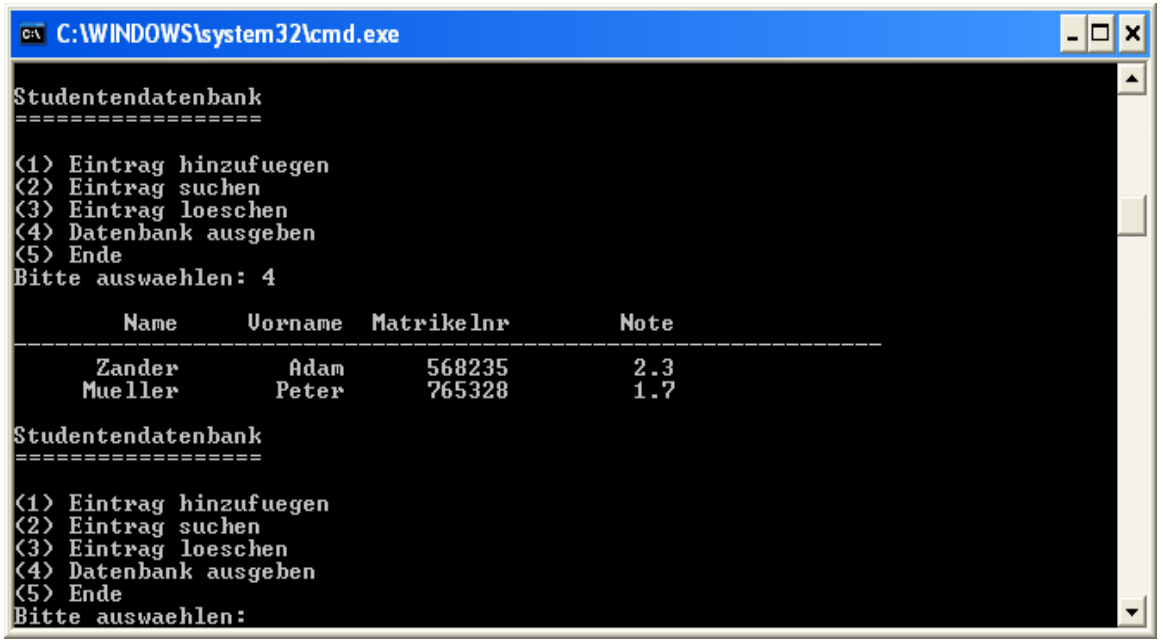


- Aufg. 6.02: OOP, Klassendefinition und Vererbung

Aufg. 6.03: Studentendatenbank

- In dieser Aufgabe soll eine Studentendatenbank in Form einer verketteten Liste unter Zuhilfenahme der STL-Klasse **`list`** aufgebaut werden. Verwenden Sie zudem Iteratoren. Die elementare **`Student`** Datenstruktur (Objekt einer Klasse) soll folgende Attribute über den jeweiligen Studenten beinhalten: Vorname, Nachname, Matrikelnummer und Note.





Aufg. 6.03: Studentendatenbank Lsg. (1)

```
//Klasse zum Speichern der Daten eines Studenten
class Student {
  private:
    string name;
    string vorname;
    int matrikelnr;
    double note;

  public:
    Student();
    ~Student();

    const string& getName() { return name; };
    const string& getVorname() { return vorname; };
    int getMatrikelnr(){ return matrikelnr; };
    double getNote() { return note; };

    void setData( string vorname, string name, int matrikelnr, double note );
};
```

student.h

Attribute zum Speichern der spezifischen Daten

get-Methoden

Aufg. 6.03: Studentendatenbank Lsg. (2)

```
Student::Student() {
  name = "";
  vorname = "";
  matrikelnr = 0;
  note = 0;
}

Student::~Student() {
}

void Student::setData( string vorname, string name, int matrikelnr, double note ) {
  this->name = name;
  this->vorname = vorname;
  this->matrikelnr = matrikelnr;
  this->note = note;
}
```

student.cpp

Konstruktor

Initialisierung der Attribute

Leerer Destruktor

Einzelne Set-Methode für alle Werte

Zugriff auf Klassenattribute über **this**-Zeiger

Aufg. 6.03: Studentendatenbank Lsg. (3)

```
//Klasse zum Speichern der Datenbank
#include "student.h"
#include <list>

class Studentenverwaltung {
  private:
    list<Student> studentenliste;

  public:
    void addStudent( const string& vorname, const string& name,
                     int matr, double note );
    void deleteStudent( list<Student>::iterator it );
    void printStudent( list<Student>::iterator it ) const;
    void print();
    list<Student>::iterator findStudent( int matrikel );
};
```

studentenverwaltung.h

Einbinden der **<list>** STL Bibliothek

list Container als Datenbank

Methoden zum Verwalten der Datenbank

Gibt Iterator auf den gefundenen Studenten zurück. Iterator „zeigt“ auf Student

Aufg. 6.03: Studentendatenbank Lsg. (4)

studentenverwaltung.cpp

```
void Studentenverwaltung::addStudent( const string& vorname, const string& name,
                                      int matr, double note ) {
  Student s;
  s.setData( vorname, name, matr, note );
  studentenliste.push_back( s );
}

void Studentenverwaltung::deleteStudent( list<Student>::iterator it ) {
  if( it != studentenliste.end() ){
    studentenliste.erase(it);
  } else {
    cout << "Student nicht gefunden" << endl;
  }
}
```

Daten dem neuem Element zuweisen

Student am Ende der Liste hinzufügen

Student auf den Iterator **`it`** zeigt aus der Liste löschen, wenn **`it`** nicht auf Listenende zeigt (**`end()`**)

Aufg. 6.03: Studentendatenbank Lsg. (5)

studentenverwaltung.cpp

```
list<Student>::iterator Studentenverwaltung::findStudent( int matrikel ) {
  list<Student>::iterator it;
  for ( it = studentenliste.begin(); it != studentenliste.end(); ++it ) {
    if ( it->getMatrikelnr() == matrikel ) {
      return it;
    }
  }
  return studentenliste.end();
}

void Studentenverwaltung::printStudent( list<Student>::iterator it ) const {
  if ( it != studentenliste.end() ) {
    cout << "Vorname: " << it->getVorname() << ", Name: "
         << it->getName() << ", Matrikel-Nr.: " << it->getMatrikelnr()
         << ", Note: " << it->getNote() << endl;
  } else {
    cout << "Student nicht gefunden" << endl;
  }
}
```

Elemente nacheinander durchgehen

Matrikelnummer überprüfen

Student nicht gefunden → Rückgabe des Iterators auf Listenende

Attribute des Studenten an Position `it` ausgeben

Aufg. 6.03: Studentendatenbank Lsg. (6)

studentenverwaltung.cpp

```
void Studentenverwaltung::print () const {
  cout << endl << setw(12) << "Name" << setw(12) << "Vorname" << setw(12)
       << "Matrikelnr" << setw(12) << "Note" << endl;
  cout << "-------------------------------------------------------" << endl;

  if ( ! studentenliste.empty() ) {
    for ( it = studentenliste.begin(); it != studentenliste.end(); ++it ) {
      cout << setw(12) << it->getName() << setw(12)
           << it->getVorname() << setw(12)
           << it->getMatrikelnr() << setw(12)
           << it->getNote() << endl;
    }
  } else {
    cout << "Die Datenbank ist leer" << endl;
  }
}
```

Tabellenkopf ausgeben

Liste nicht leer?

Daten-Attribute jedes Studenten ausgeben

Aufg. 6.03: Studentendatenbank Lsg. (7)

main.cpp

```
int main() {
  Studentenverwaltung* database = new Studentenverwaltung;
  int auswahl;

  do {
    cout << endl;
    cout << "Studentendatenbank" << endl;
    cout << "------------------" << endl << endl;
    cout << "(1) Eintrag hinzufuegen" << endl;
    cout << "(2) Eintrag suchen" << endl;
    cout << "(3) Eintrag loeschen" << endl;
    cout << "(4) Datenbank ausgeben" << endl;
    cout << "(5) Ende" << endl;
    cout << "Bitte auswaehlen: ";
    if ( cin >> auswahl ) {
      switch( auswahl ) {
        case 1: daten_hinzufuegen( database ); break;
        case 2: daten_suchen( database ); break;
        case 3: daten_loeschen( database ); break;
        case 4: daten_ausgabe( database ); break;
        case 5: break;
        default: cout << "Ihre Eingabe ist falsch, die Option steht nicht zur Verfuegung" << endl;
      }
    } else {
      cout << "Sie haben keine Zahl eingegeben!" << endl;
    }
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
  } while( auswahl != 5 );
  delete database;
}
```

Ausgabe des Menüs

Einlesen der Auswahl des Benutzers

Je nach der Auswahl des Benutzers wird die entsprechende Funktion aufgerufen

Bei Fehleingabe wird eine Fehlermeldung ausgegeben

Wichtig: Eingabepuffer nach Fehleingabe leeren (Einbinden von `<limits>`)

Abbruchbedingung der `while`-Schleife

Freigeben des Datenbankspeichers

Aufg. 6.03: Studentendatenbank Lsg. (8)

```
void daten_hinzufuegen( Studentenverwaltung* database ) {
  string name;
  string vorname;
  int matrikelNr;
  double note;

  cout << endl << "Bitte die neuen Daten eingeben" << endl;
  cout << "Name: ";
  cin >> name;
  cout << "Vorname: ";
  cin >> vorname;
  cout << "Matrikelnummer: ";
  cin >> matrikelNr;
  cout << "Note: ";
  cin >> note;

  database->addStudent( vorname, name, matrikelNr, note );
}
```

main.cpp

Einlesen der Daten von der Tastatur

Aufruf der Methode zum Hinzufügen der Daten zur verketteten Liste

Aufg. 6.03: Studentendatenbank Lsg. (9)

```
void daten_suchen( Studentenverwaltung* database ) {
  int matrikelNr;

  cout << "Bitte zu suchende Matrikelnummer eingeben: ";
  cin >> matrikelNr;

  list<Student>::iterator gesucht = database->findStudent( matrikelNr );
  database->printStudent(gesucht);
}
```

main.cpp

Student mit entsprechender Matrikelnummer in der Liste vorhanden?

Ausgabe der Daten des gefundenen Studenten durch Delegation an die Methode der Datenbank

Aufg. 6.03: Studentendatenbank Lsg. (10)

```
void daten_loeschen( Studentenverwaltung* database ){
  int matrikelNr;

  cout << "Bitte zu loeschende Matrikelnummer eingeben: ";
  cin >> matrikelNr;


  list<Student>::iterator gesucht = database->findStudent( matrikelNr );
  database->deleteStudent( gesucht );
}

void daten_ausgabe( Studentenverwaltung* database ) {
  database->print();
}
```

main.cpp

Student mit entsprechender Matrikelnummer in der Liste vorhanden?

Löschen des gefundenen Elements

Weiterleitung der Methode an die Datenbank

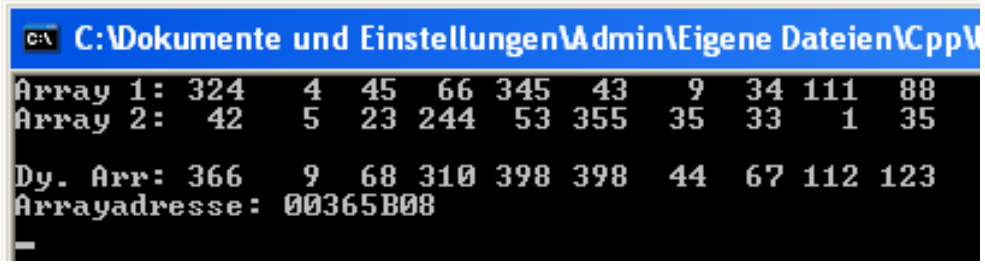
- Aufg. 6.03: Studentendatenbank

Aufg. 6.04: Dynamische Arrays

- Schreiben Sie eine Funktion, welche elementweise die Summe zweier gleich langer Arrays in ein dynamisch reserviertes Array schreibt und einen Zeiger auf das neue Array zurückgibt. Der Typ der Arrayelemente ist `long`.

- Der Funktion werden beide Arrays und ihre gemeinsame Länge übergeben. Testen Sie die Funktion, indem Sie die Arrays mit Werten Ihrer Wahl initialisieren und die Summe der Arrays anzeigen. Geben Sie danach den Speicherplatz explizit frei.

- Beispielausgabe:



Aufg. 6.04: Dynamische Arrays Lsg. (1)

```
#include <iomanip>
#include <iostream>

using namespace std;

long* element_sum( long arr_1[], long arr_2[], int laenge );

int main() {
  long wert_1[10] = {324, 4, 45, 66, 345, 43, 9, 34, 111, 88};
  long wert_2[10] = {42, 5, 23, 244, 53, 355, 35, 33, 1, 35};
  long* summ = NULL;

  cout << "Array 1: ";
  for( int i = 0; i < 10; i++ ) {
    cout << setw( 4 ) << wert_1[i];
  }

  cout << endl << "Array 2: ";
  for( int i = 0; i < 10; i++ ) {
    cout << setw( 4 ) << wert_2[i];
  }
```

Zum Speichern der Adresse des dynamischen Arrays

Array 1 ausgeben

`setw()`: Legt die Breite eines Ausgabefelds fest → Einbinden der Bibliothek **`<iomanip>`** ist erforderlich

Array 2 ausgeben

Aufg. 6.04: Dynamische Arrays Lsg. (2)

```
  summ = element_sum( wert_1, wert_2, 10 );

  cout << endl << endl << "dy. Arr: ";
  for( int i = 0; i < 10; i++ ) {
    cout << setw( 4 ) << summ[i];
  }

  cout << endl << "Arrayadresse: " << summ << endl;

  delete[] summ;
  summ = NULL;
  return 0;
}
```

Funktion aufrufen, Arrays und Länge werden übergeben

dynamisches Array ausgeben

Speicher des dynamisch erzeugten Arrays wieder freigeben

```
long* element_sum( long arr_1[], long arr_2[], int laenge ) {
  long* p = new long[laenge];

  for( int i = 0; i < laenge; i++ ) {
    p[i] = arr_1[i] + arr_2[i];
  }

  return p;
}
```

dynamisches Array anlegen

Summe bilden und in Array ablegen

Adresse des dynamisch erzeugten Arrays zurückgeben

- Aufg. 6.04: Dynamische Arrays

Aufg. 6.05: STL Dictionary

- Programmieren Sie unter Zuhilfenahme der STL-Klasse **`map`** ein Wörterbuch englisch-deutsch / deutsch-englisch. Verwenden Sie hierbei **Iteratoren**. Der Nutzer gibt ein Suchwort ein und dazu wird die passende Übersetzung angezeigt. Immer wenn der Nutzer ein Suchwort eingibt, das nicht im Wörterbuch steht, bekommt er die Gelegenheit, das Wort zusammen mit der Übersetzung in das Wörterbuch aufzunehmen. Beim Beenden des Programms sollen alle Wortpaare in einer Textdatei gespeichert werden. Zu Beginn des Programms sollen alle Wortpaare aus dieser Textdatei eingelesen werden, falls sie bereits existiert.

- Hinweis: Textdateien lesen / schreiben mittels C++ nicht klausurrelevant.

Aufg. 6.05: STL Dictionary Lsg. (1)

- Speicherung während der Laufzeit des Programms

- **`map<key, value>`**
  - **`key`** = deutsches Wort
  - **`value`** = englisches Wort
  - Zugriff über Iterator

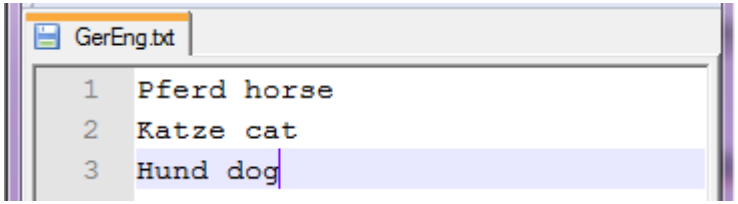
- Beispiel:

```
map<string, string> myMap;
myMap["Hund"] = "dog";
map<string, string>::iterator
  myIterator;
myIterator = myMap.begin();
cout << myIterator->first;  //"Hund"
cout << myIterator->second; //"dog"
```

- Speicherung nach Beendigung des Programms

- Textdatei - Protokoll
  - Ein Wortpaar pro Zeile
  - 1. deutsches Wort
  - 2. Leerzeichen
  - 3. englisches Wort
  - 4. end line

- Beispiel:



Aufg. 6.05: STL Dictionary Lsg. (2)

Textdatei

Interaktion mit Textdatei

Dictionary

- lesen der Textdatei
- speichern in **`map`**
- suchen in **`map`**
- speichern in Textdatei

verwaltet

Benutzer

Interaktion mit Benutzer

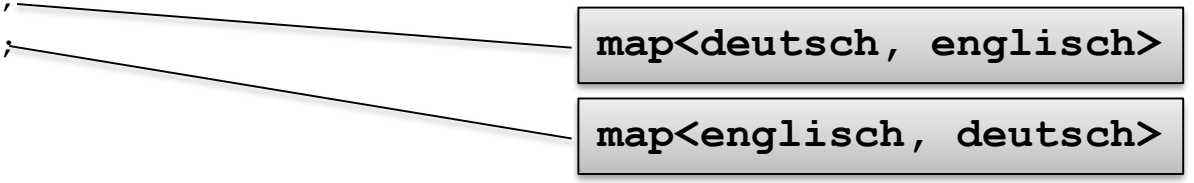
Translator

- übergibt Suchbegriff und neue Wortpaare an Dictionary
- gibt Übersetzung aus

Aufg. 6.04: STL Dictionary Lsg. (3)

dictionary.h

```
class Dictionary {

private:
  map<string, string> gerEngMap;
  map<string, string> engGerMap;
  string filename;

public:
  Dictionary( string filename );
  string searchGerman( string searchString );
  string searchEnglish( string searchString );
  void add( string germanWord, string englishWord );
  void readFile();
  void saveToFile();
};
```



Aufg. 6.04: STL Dictionary Lsg. (4)

```
string Dictionary::searchEnglish( string searchString ) {
  map<string, string>::iterator mapIterator;
  mapIterator = engGerMap.find( searchString );

  if ( mapIterator! = engGerMap.end() ) {
    return mapIterator->second;
  }
  else {
    return "not in map";
  }
}

string Dictionary::searchGerman( string searchString ) {
  map<string, string>::iterator mapIterator;
  mapIterator = gerEngMap.find( searchString );

  if ( mapIterator != gerEngMap.end() ) {
    return mapIterator->second;
  } else {
    return "not in map";
  }
}

void Dictionary::add( string germanWord, string englishWord ) {
  gerEngMap[germanWord] = englishWord;
  engGerMap[englishWord] = germanWord;
}
```

dictionary.cpp

passender Iterator

`find(..)` gibt Position des Suchbegriffs (**`key`**) oder **`end()`** zurück

über **`first`** und **`second`** wird der **`key`** bzw. der **`value`** ausgelesen

Zugriff auf Map:
`map[key] = value`

Aufg. 6.04: STL Dictionary Lsg. (5)

dictionary.cpp

```
void Dictionary::saveToFile() {
  fstream myFile( filename.c_str(), ios::out );
  map<string, string>::iterator mapIterator;

  for( mapIterator = gerEngMap.begin();
                     mapIterator != gerEngMap.end();
                     ++mapIterator ) {
    myFile << mapIterator->first << " " << mapIterator->second << endl;
  }
  myFile.close();
}
```

`begin()` zeigt auf das 1. Element der **`map`**

`end()` auf das Ende der **`map`** und nicht auf das letzte Element

über **`first`** und **`second`** wird der **`key`** bzw. der **`value`** ausgelesen und durch ein Leerzeichen getrennt in eine Zeile geschrieben

Aufg. 6.04: STL Dictionary Lsg. (6)

dictionary.cpp

```
Dictionary::Dictionary( string _filename ) {
  filename = _filename;
  readFile();
}

void Dictionary::readFile() {
  fstream myFile( filename.c_str(), ios::in );
  if( myFile ) {
    string germanWord, englishWord;
    while( myFile >> germanWord >> englishWord ){
      gerEngMap[germanWord] = englishWord;
      engGerMap[englishWord] = germanWord;
    }
    myFile.clear();
  } else {
    myFile.open( filename.c_str(), ios::out );
  }
  myFile.close();
}
```

öffne Datei zum Lesen

falls Datei existiert

lese alle Wertepaare

lösche error flags

falls Datei noch nicht existiert

erzeuge Datei

schließe Datei

Aufg. 6.04: STL Dictionary Lsg. (7)

translator.h

```
class Translator{
public:
  Translator( string _filename );
  ~Translator();
  void menu();
  void save();
private:
  string filename;
  Dictionary* myDictionary;
  void search();
};
```

Aufg. 6.04: STL Dictionary Lsg. (8)

translator.cpp

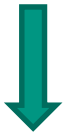
```
Translator::Translator( string _filename ) {
  myDictionary = new Dictionary( _filename );
}

void Translator::save() {
  myDictionary->saveToFile();
}

Translator::~Translator(){
  /*Beim Zerstören des Translators auch Speicher
  * des Dictionary wieder freigeben
  */
  delete myDictionary;
}
```

Auch Speicher des Dictionary wieder freigeben!

Aufg. 6.04: STL Dictionary Lsg. (9)

translator.cpp

```
void Translator::search() {
  string searchWord;
  bool wordFound = false;
  cout << "Bitte deutsches oder englisches Suchwort eingeben: ";
  getline( cin, searchWord );
  cin.sync();
  cout << endl;

  string germanTranslation=myDictionary->searchEnglish( searchWord );
  string englishTranslation=myDictionary->searchGerman( searchWord );

  if( germanTranslation != "not in map" ) {
    cout << "Die deutsche \x9A \bbersetzung lautet: " << germanTranslation << endl;
    wordFound = true;
  }

  if( englishTranslation != "not in map" ) {
    cout << "Die englische \x9A \bbersetzung lautet: " << englishTranslation << endl;
    wordFound = true;
  }
```



Aufg. 6.04: STL Dictionary Lsg. (10)

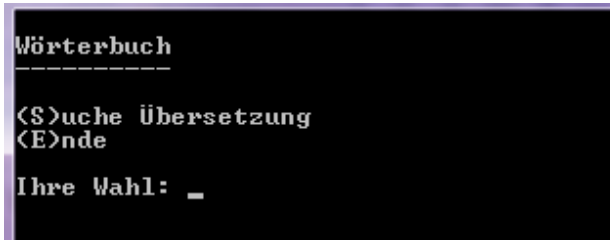
translator.cpp

```
if( !wordFound ) {
   char add = 'a';
   string germanWord, englishWord;
   do {
     cout << "Begriff nicht im W\x94rterbuch! Hinzuf\x81gen? (y/n)";
     cin >> add;
     cin.ignore();
     cout << endl;
   } while(( add != 'y' ) && ( add != 'n' ));

   if( add == 'y' ) {
     cout << "Bitte das deutsche Wort eingeben: ";
     getline( cin, germanWord );
     cin.sync();
     cout << "Bitte das englische Wort eingeben: ";
     getline( cin, englishWord );
     cin.sync();
     myDictionary->add( germanWord, englishWord );
     cout << "Neues Wortpaar hinzugef\x81gt" << endl;
   }
  }
}
```

Aufg. 6.04: STL Dictionary Lsg. (11)

translator.cpp

```
void Translator::menu() {
  char auswahl = 'a';
  do {
    cout << endl << "W\x94rterbuch" << endl;
    cout << "----------" << endl << endl;
    cout << "(S)uche \x9A \bbersetzung" << endl;
    cout << "(E)nde" << endl << endl;
    cout << "Ihre Wahl: ";
    cin >> auswahl;
    cin.ignore();
    switch( auswahl ) {
      case 'S':
      case 's':
        search();
        system( "pause" );
        break;
      case 'e':
      case 'E': break;
      default:
        cout << "Keine g\x811tige Eingabe!" << endl;
        system( "pause" );
    }
    system( "cls" );
  } while( auswahl != 'e' && auswahl != 'E' );
}
```



Überprüfung, ob es beim Einlesen einen Fehler gab, fehlt hier (und an manchen anderen Stellen)!

Aufg. 6.04: STL Dictionary Lsg. (12)

```
int main() {
  Translator* myTranslator;
  myTranslator = new Translator( "GerEng.txt" );
  myTranslator->menu();
  myTranslator->save();
  delete myTranslator;
  return 0;
}
```

main.cpp

Speicher des dynamisch erzeugten Translators freigeben!

Der hier gezeigte Quellcode ist nicht 100% vollständig → Alle Quellcode-Dateien zu allen Aufgaben finden Sie wie immer auf der Lernplattform ILIAS

Aufg. 6.04: STL Dictionary Lsg. (13)

```
Wörterbuch
----------

(S)uche Übersetzung
(E)nde

Ihre Wahl: s
Bitte deutsches oder englisches Suchwort eingeben: Hund

Die englische Übersetzung lautet: dog
Drücken Sie eine beliebige Taste . . .
```

Begriff ist bereits im Wörterbuch

```
Wörterbuch
----------

(S)uche Übersetzung
(E)nde

Ihre Wahl: s
Bitte deutsches oder englisches Suchwort eingeben: Katze

Begriff nicht im Wörterbuch! Hinzufügen? (y/n)y

Bitte das deutsche Wort eingeben:Katze
Bitte das englische Wort eingeben:cat

Neues Wortpaar hinzugefügt
Drücken Sie eine beliebige Taste . . .
```

Begriff nicht im Wörterbuch und wird auf Nachfrage hinzugefügt

- Aufg. 6.04: STL Dictionary

Vielen Dank für Ihre Aufmerksamkeit

Daniel Grimm
Karlsruher Institut für Technology (KIT) – ITIV
daniel.grimm@kit.edu