Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Übung07: Informationstechnik (IT)

Institutsleitung
Prof. Dr.-Ing. K. D. Müller-Glaser
Prof. Dr.-Ing. J. Becker
Prof. Dr. rer. nat. W. Stork

Tobias Schwalb & Timo Sandmann & Stephan Werner

Teil1: Besprechung der Übungsaufgaben 6.01 - 6.04

Ankündigung: Tutoren zum Praktikum gesucht

Karlsruher Institut für Technologie

- Aufgaben
  - Betreuung des Praktikums in den Poolräumen des RZ
  - Tutoren-Schulung wird als Schlüsselqualifikation anerkannt
- Voraussetzungen
  - Fundierte Kenntnisse in C++ (v.a. Objektorientierung)
  - Spaß am Erklären von Problemstellungen
- Ansprechpartner
  - Tobias Gädeke, Tel. 608-48248, Email: tobias.gaedeke@kit.edu

Inhalt: Übung07 – Teil 1

- Aufg. 6.01: Verständnisfragen
- Aufg. 6.02: Dateiverarbeitung
- Aufg. 6.03: STL
- Aufg. 6.04: Verkettete Liste

Aufg. 6.01: Verständnisfragen Lsg. (1)

a) Eine Datei ist aus der Sicht eines C++ Programms eine Folge von
   i. Bytes Richtig
   ii. Datensätzen Falsch
   iii. int-Werten Falsch

b) Die Methoden, Operatoren und Manipulatoren, die Sie im Zusammenhang mit `cin` und `cout` bereits verwendet haben, stehen auch für File-Streams zur Verfügung. Richtig

c) Wenn eine Datei im Default-Modus, d.h. ohne explizite Angabe eines Eröffnungsmodus eröffnet wird, ist die aktuelle Dateiposition
   i. nicht festgelegt Falsch
   ii. der Dateianfang Richtig
   iii. das Dateiende Falsch

d) Zur Abfrage, ob beim Lesen das Dateiende bereits erreicht wurde, kann die Methode `eof()` aufgerufen werden.

e) Die C++ Standard Template Library (STL) ein eine Sammlung von Klassen Templates, die es dem Programmierer erlauben einfache Standard-datenstrukturen und Algorithmen zu verwenden. Richtig

Aufg. 6.01: Verständnisfragen Lsg. (2)

f) Die STL bietet „Container“, welche nur bei bestimmten Datentypen angewendet werden können. Falsch

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

h) 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

i) 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. 6.02: Dateiverarbeitung

- Erstellen Sie ein Programm, das als Memoblock verwendet werden kann. Dabei soll das Programm ein Menü bereitstellen, im welchem der Benutzer auswählen kann, ob er seine bisherigen Memos lesen oder eine neue Memo hinzufügen möchte.
  - Beim Hinzufügen soll das Programm die Datei *„memo.txt“* zum Schreiben ab Dateiende öffnen. Wenn die Datei nicht vorhanden ist, soll sie angelegt werden. Anschließend soll das Programm eine Zeile vom Benutzer abfragen und diese mit einem führenden Zeitstempel (siehe Hinweis) in die Datei schreiben. Anschließend wird die Datei wieder geschlossen.
  - Beim Lesen soll die Datei „memo.txt“ geöffnet werden, jede Zeile aus der Datei gelesen werden und mit einer führenden Zeilennummer auf dem Bildschirm ausgegeben werden. Die Zeile enthält dabei den Zeitstempel und den jeweiligen Text. Anschließend wird die Datei wieder geschlossen.
  - Falls ein Fehler beim Dateizugriff auftritt, soll dieser mit einer entsprechenden Fehlermeldung auf dem Bildschirm angezeigt werden.
- Hinweis: Die aktuelle Zeit als String erhalten Sie durch den Aufruf der Standardfunktion `time()` oder `ctime()` der Bibliothek `<ctime>`.

Aufg. 6.02: Dateiverarbeitung Lsg. (1)

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>

using namespace std;

class MemoBlock{
private:
  string filename;
  inline void openError();
  inline void writeError();
  inline void readError();
  void schreiben();
  void lesen();
  void loeschen();

public:
  MemoBlock();
  ~MemoBlock();
  void setfilename( string _filename );
  void menu();
};
```

Zur Ermittlung / Ausgabe der Zeit

Speicherung des Dateinamens

Fehlerfunktionen

Eingabe und speichern eines Memos

Lesen aller Memos

Löschen aller Memos

Benutzermenü

Aufg. 6.02: Dateiverarbeitung Lsg. (2)

- Fehlerfunktionen

Jeweils Fehlermeldung ausgeben und Programm beenden

```
inline void MemoBlock::openError() {
  cerr << "Fehler beim Oeffnen der Datei " << filename << endl;
  exit( 1 );
}

inline void MemoBlock::writeError() {
  cerr << "Fehler beim Schreiben in die Datei " << filename << endl;
  exit( 2 );
}

inline void MemoBlock::readError() {
  cerr << "Fehler beim Lesen der Datei " << filename << endl;
  exit( 3 );
}
```

Aufg. 6.02: Dateiverarbeitung Lsg. (3)

```
void MemoBlock::schreiben() {
  string line, zeit;
  fstream memoFile( filename.c_str(), ios::out | ios::app );

  if( !memoFile ) {
    openError();
  }

  time_t sec = time( NULL );
  zeit = ctime( &sec );
  zeit.erase( zeit.length() - 1, 1 );

  cout << "Geben Sie Ihre Notiz ein:\n";
  getline( cin, line );

  if( !( memoFile << zeit << ": " << line << endl ) ) {
    writeError();
  }

  memoFile.close();
}
```

Lesen und Abspeichern eines Memos

Datei zum Schreiben „ab Dateiende“ öffnen

Fehler beim Öffnen?

Ermittlung und Formatierung der Zeit

Notiz vom User abfragen

Zeit und Memo in die Datei schreiben mit Fehlerabfrage

Datei schließen

Aufg. 6.02: Dateiverarbeitung Lsg. (4)

```
void MemoBlock::lesen() {
  string line;
  int number = 0;

  fstream memoFile( filename.c_str(), ios::in );

  if( !memoFile ) {
    openError();
  }

  while( getline( memoFile, line ) ) {
    cout.width( 5 );
    cout << ++number << ": " << line << endl;
  }

  if( !memoFile.eof() ) {
    readError();
  }

  memoFile.close();
}
```

Lesen aller Memos

Datei zum Lesen öffnen

Fehler beim Öffnen?

Lesen einer Zeile aus der Datei

Ausgabebreite für die Nummer festlegen

Nummer und gelesene Zeile ausgeben

Fehler beim Lesen? (nicht am Dateiende)

Datei schließen

Aufg. 6.02: Dateiverarbeitung Lsg. (5)

```
void MemoBlock::loeschen() {
  fstream memoFile( filename.c_str(), ios::out );

  if( !memoFile ) {
    openError();
  }

  memoFile.close();
}

MemoBlock::MemoBlock() {
}

MemoBlock::~MemoBlock() {
}

void MemoBlock::setfilename( string _filename ) {
  filename = _filename;
}
```

Inhalt der Datei löschen (nicht Datei selbst)

Datei zum Schreiben (ab Dateianfang) öffnen und wieder schließen

Leerer Konstruktor

Leerer Destruktor

Festlegen des Dateinamen (privates Attribut)

Aufg. 6.02: Dateiverarbeitung Lsg. (6)

```
void MemoBlock::menu() {
  char auswahl = 'a';

  do {
    cout << endl;
    cout << "Memoblock" << endl;
    cout << "---------" << endl << endl;
    cout << "(N)eues Memo hinzufuegen" << endl;
    cout << "(V)orhandene Memosausgeben" << endl;
    cout << "(L)oechen der Memodatei" << endl;
    cout << "(E)nde" << endl;
    cout << "bitte auswaehlen:";
    cin >> auswahl;
    cin.clear();
    cin.sync();

    switch( auswahl ) {
      case 'N':
      case 'n': schreiben(); break;
      case 'V':
      case 'v': lesen(); break;
      case 'L':
      case 'l': loeschen(); break;
      case 'e':
      case 'E': break;
      default: cout << "Keine gueltige Eingabe!";
    }
  } while( auswahl != 'e' && auswahl != 'E' );
}
```

Alles läuft in der Schleife bis zur Eingabe von `'e'` oder `'E'`

Menü ausgeben

Nach der Auswahl entsprechend die Methoden aufrufen

Fehlermeldung für ungültige Eingabe

Aufg. 6.02: Dateiverarbeitung Lsg. (7)

```
int main() {
  MemoBlock Memo;
```

Objekt der Klasse anlegen

```
  Memo.setfilename( "memo.txt" );
```

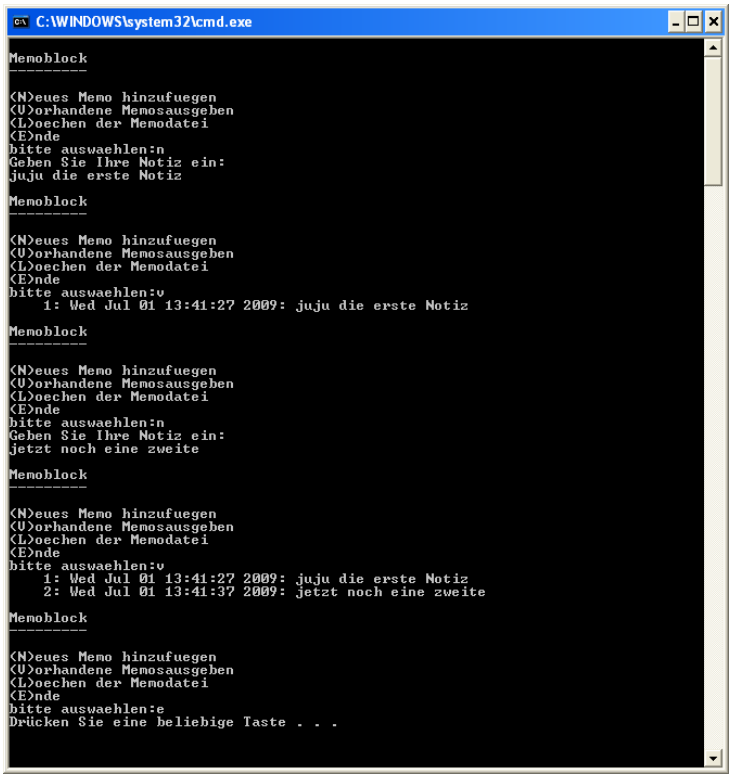
Dateinamen festlegen

```
  Memo.menu();
```

Menü starten

```
  return 0;
}
```

Hinweis: mit dem Befehl
`system( "cls" );`
kann man die
Bildschirmausgabe löschen



Aufgabe 6.03: STL

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.

Aufg. 6.03: Speicherung Lsg.(1)

Speicherung während der Laufzeit des Programms

- map<key, value>
  - key = deutsches Wort
  - value = englisches Wort
  - Zugriff über Iterator
- Beispiel:

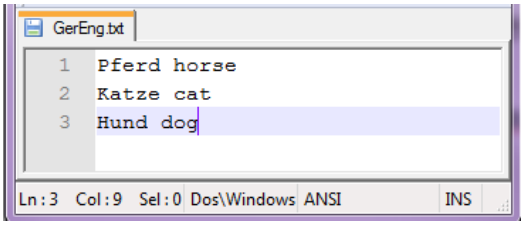
```
map<string,string> myMap;
myMap["Hund"] = "dog";

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. 6.03: Aufteilung in 2 Klassen 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. 6.03: Dictionary.h Lsg.(3)

```
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();
};
```

map<deutsch, englisch>

map<englisch, deutsch>

Aufg. 6.03: Dictionary.cpp Lsg.(4)

```
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. 6.03: Dictionary.cpp Lsg.(5)

```
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";
  }
}
```

passender Iterator

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

Aufg. 6.03: Dictionary.cpp Lsg.(6)

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

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. 6.03: Translator.h Lsg.(7)

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

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

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


Aufg. 6.03: Translator.cpp Lsg.(8)

```
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. 6.03: Translator.cpp Lsg.(9)

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


Aufg. 6.03: Translator.cpp Lsg.(10)

```
void Translator::menu() {
  char auswahl = 'a';
  do {
    cout << endl;
    cout << "W\x94rterbuch" << endl;
    cout << "----------" << endl << endl;
    cout << "(S)uche \x9A \bbersetzung" << endl;
    cout << "(E)nde" << endl << endl;
    cout << "Ihre Wahl: ";
    cin >> auswahl;
    cin.clear(); cin.sync();
    switch( auswahl ) {
      case 'S':
      case 's': {
        search();
        break;
      }
      case 'e':
      case 'E': break;
      default: {
        cout << "Keine g\x81ltige Eingabe!" << endl;
        system( "pause" );
      }
    }
    system( "cls" );
  } while( auswahl != 'e' && auswahl != 'E' );
}
```

Aufg. 6.03: main.cpp Lsg.(11)

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

    return 0;
}
```


Aufg. 6.03: Beispiel Lsg.(12)

```
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 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: Studentendatenbank Lsg. (1)

student.h

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

  public:
    Student();
    ~Student();

    inline const string& getName() { return name; }
    inline const string& getVorname() { return vorname; }
    inline int getMatrikelnr() { return matrikelnr; }
    inline double getNote() { return note; }
    inline Student* getNext() { return next; }

    inline void setNext( Student* nt )  { next = nt; }
    void setData( string name, string vorname, int matrikelnr, double note );

    void displayData();
};
```

Attribute zum Speichern der spezifischen Daten

Nachfolger-Zeigervariable
→ Zum Aufbau der verketteten Liste

get-Methoden

set-Methoden

Ausgabe aller Daten eines Studenten

Aufg. 6.04: Studentendatenbank Lsg. (2)

student.cpp

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

Student::~Student() {
}

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

void Student::displayData() {
  cout << "Name         : " << name << endl;
  cout << "Vorname      : " << vorname << endl;
  cout << "Matrikelnr.  : " << matrikelnr << endl;
  cout << "Note         : " << note << endl;
}
```

Konstruktor

Initialisierung der Attribute

Neues Element hat keinen Nachfolger

Leerer Destruktor

Einzelne Set-Methode für alle Werte

Zugriff auf Klassenattribute über `this`-Zeiger

Ausgabe der Werte eines Studenten

Aufg. 6.04: Studentendatenbank Lsg. (3)

studDatabase.h

```
//Klasse zum Speichern der Datenbank
class StudDatabase {
  private:
    Student* start;
    Student* ende;

  public:
    StudDatabase();
    ~StudDatabase();

    void hinzufuegen( string name, string vorname, int matrikelnr,
                      double note );
    Student* suchen( int matrikelnr );
    void loeschen( Student* zuLoeschen );
    void ausgabe();

};
```

Zeiger für die verketteten Liste aus **Student**-Elementen

Methoden zum Verwalten der verketteten Liste

Aufg. 6.04: Studentendatenbank Lsg. (4)

studDatabase.cpp / 1

```
StudDatabase::StudDatabase() {
  start = NULL;
  ende = NULL;
}

StudDatabase::~StudDatabase() {
  while( start != NULL ) {
    loeschen( start );
  }
}

void StudDatabase::hinzufuegen( string name, string vorname, int matrikelnr,
                                double note ) {
  Student* neu = new Student;
  if( start == NULL ) {
    start = neu;
  } else {
    ende->setNext( neu );
  }

  ende = neu;

  neu->setData( name, vorname, matrikelnr, note );
}
```

Verkettete Liste ist am Anfang leer

Löschen der Elemente der verketteten Liste bei Zerstörung der Datenbank

Neues Element dynamisch anlegen

Verkette Liste ist komplett leer?

Neues Element ist der Nachfolger des aktuellen letzten Elements

Neues Element ist das neue letzte Element der Liste

Daten dem neuem Element zuweisen

Aufg. 6.04: Studentendatenbank Lsg. (5)

studDatabase.cpp / 2

```
void StudDatabase::loeschen( Student* zuLoeschen ) {
  Student* zeigt_geloescht = start;

  if( zuLoeschen == start && zuLoeschen == ende ) {
    start = NULL;
    ende = NULL;
    delete zuLoeschen;
  } else if( zuLoeschen == start ) {
    start = zuLoeschen->getNext();
    delete zuLoeschen;
  } else {
    for( zeigt_geloescht = start; zeigt_geloescht->getNext() != zuLoeschen;
         zeigt_geloescht = zeigt_geloescht->getNext() );
    zeigt_geloescht->setNext( zuLoeschen->getNext() );
    delete zuLoeschen;
    if( zeigt_geloescht->getNext() == NULL ) {
      ende = zeigt_geloescht;
    }
  }
  cout << "--> Loeschung Erfolgreich" << endl;
}
```

Zum Speichern des vorherigen Elements

Zu löschende Element ist das Einzige in der Liste

Zu löschende Element ist das Erste in der Liste

Zu löschende Element ist in der Liste

Zu löschende Element ist das Letzte in der Liste

Aufg. 6.04: Studentendatenbank Lsg. (6)

studDatabase.cpp / 3

```
Student* StudDatabase::suchen( int matrikelnr ) {
  Student* gesucht = NULL;
  for( Student* ptr_stud = start; ptr_stud != NULL; ptr_stud = ptr_stud->getNext() ) {
    if( matrikelnr == ptr_stud->getMatrikelnr() ) {
      gesucht = ptr_stud;
      break;
    }
  }
  return gesucht;
}

void StudDatabase::ausgabe() {
  Student* ptr_stud = start;
  cout << endl << setw( 12 ) << "Name" << setw( 12 ) << "Vorname" << setw( 12 )
       << "Matrikelnr" << setw( 12 ) << "Note" << endl;
  cout << "-----------------------------------------------------------------" << endl;

  if( start == NULL ) {
    cout << "--> Datenbank ist Leer" << endl;
  } else {
    for( ptr_stud = start; ptr_stud != NULL; ptr_stud = ptr_stud->getNext() ) {
      cout << setw( 12 ) << ptr_stud->getName() << setw( 12 ) << ptr_stud->getVorname()
           << setw( 12 ) << ptr_stud->getMatrikelnr() << setw( 12 ) << ptr_stud->getNote()
           << endl;
    }
  }
}
```

Elemente nacheinander durchgehen

Matrikelnummer überprüfen

Gefundenes Element zurückgeben, wenn kein Element gefunden = NULL

Tabellenkopf ausgeben

Verkettete Liste leer?

Elemente nacheinander durchgehen

Daten-Attribute ausgeben

Aufg. 6.04: Studentendatenbank Lsg. (7)

main.cpp / 1

```
int main() {
  StudDatabase* database = new StudDatabase;
  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: ";
    cin >> auswahl;
    cin.clear();
    cin.sync();

    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";
    }
  } 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

Abbruchbedingung der While-Schleife

Freigeben des Speichers der Datenbank

Aufg. 6.04: Studentendatenbank Lsg. (8)

main.cpp / 2

```
void daten_hinzufuegen( StudDatabase* database ) {
  string name;
  string vorname;
  int matrikelNr;
  int 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->hinzufuegen( name, vorname, matrikelNr, note );
}

void daten_suchen( StudDatabase* database ) {
  int matrikelNr;

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

  Student* gesucht = database->suchen( matrikelNr );
  if( gesucht != NULL ) {
    gesucht->displayData();
  } else {
    cout << "Matrikelnummer in Datenbank nicht gefunden" << endl;
  }
}
```

Einlesen der Daten von der Tastatur

Aufruf der Methode zur Hinzufügen der Daten zur verketteten Liste

Element mit entsprechender Matrikelnummer in der Liste vorhanden?

Ausgabe der Daten des gefundenen Elements

Aufg. 6.04: Studentendatenbank Lsg. (9)

main.cpp / 3

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

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

  Student* gesucht = database->suchen( matrikelNr );
  if( gesucht != NULL ) {
    database->loeschen( gesucht );
  } else {
    cout << "Matrikelnummer in Datenbank nicht gefunden" << endl;
  }
}

void daten_ausgabe( StudDatabase* database ) {
  database->ausgabe();
}
```

Element mit entsprechender Matrikelnummer in der Liste vorhanden?

Löschen des gefundenen Elements

Weiterleitung der Methode an die Datenbank

Alle Quellcode-Dateien zu dieser Aufgabe finden Sie auch auf der Lernplattform eStudium