Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

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

Übung06: Informationstechnik (IT)

Tobias Schwalb & Michael Tansella

Institut für Technik der Informationsverarbeitung (ITIV)

Teil1: Besprechung der Übungsaufgaben 5.01-5.03

Inhalt: Übung06 – Teil 1

Karlsruher Institut für Technologie

- Aufg. 5.01: Verständnisfragen
- Aufg. 5.02: Stringmanipulation
- Aufg. 5.03: Studentendatenbank anhand einer verketteten Liste

Aufg. 5.01: Verständnisfragen Lsg. (1)

a) Bei der Ausführung der Anweisungen: **`string name; cin >> name;`** werden Zeichen von der Standardeingabe eingelesen und zwar

i. alle Zeichen einer Zeile ohne führende Zwischenraumzeichen Falsch

ii. genau ein Wort ohne führende Zwischenraumzeichen Richtig

iii. eine ganz Textzeile Falsch

b) Zum verketten zweier Objekte vom Typ **`string`** kann man den Operator **`+ oder +=`** verwenden.

c) Das erste Zeichen in einem String hat die Position 0.

Aufg. 5.01: Verständnisfragen Lsg. (2)

d) Für den Zugriff auf die einzelnen Zeichen in einem **`string`**-Objekt kann der Operator **`[]`** verwendet werden.

e) 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.

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

Aufg. 5.02: Stringmanipulation

- Schreiben Sie ein Programm, welches in einen String alle Vorkommen eines (anderen) Strings **findstring** sucht und die jeweiligen Positionen auf dem Bildschirm ausgibt. In nächsten Schritt soll dieser String durch einen anderen **replaceString** ersetzt werden. Die Länge der Strings können dabei beliebig (auch unterschiedlich) sein. Wenn **findstring** nicht im String gefunden wird soll eine Fehlermeldung ausgegeben werden.

- Beispielausgabe

Aufg. 5.02: Stringmanipulation Lsg. (1)

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string text, findstring, replacestring;
  int i = -1;

  cout << "Text eingeben : ";
  getline( cin, text );
  cout << "findstring    : ";
  cin >> findstring;
  cout << "replacestring : ";
  cin >> replacestring;

  if( text.find( findstring, 0 ) == -1 ) {
    cout << "Fehler: findString ist nicht vorhanden\n";
    return 1;
  }
```

Einbinden der Bibliothek für String

Stringvariablen zum Speichern der Eingaben

Einlesen einer Zeile mit Leerzeichen

Einlesen eines einzelnen Worts

Überprüfen, ob die Zeichenkette vorhanden ist

Programm mit Fehlermeldung beenden

Aufg. 5.02: Stringmanipulation Lsg. (2)

```
  cout << endl;

  do {
    i = text.find( findstring, i + 1 );
    if( i != -1 )
      cout << "String " << findstring
           << " gefunden an Pos: " << i << endl;
  } while( i >= 0 );

  i = -1;
  do {
    i = text.find( findstring, i + 1 );
    if( i != -1 )
      text.replace( i, findstring.length(), replacestring );
  } while( i >= 0 );

  cout << "Ergebnis: " << text << endl;

  return 0;
}
```

Schleife wird so oft durchlaufen, wie `i >= 0`

`findstring` suchen und Position in `i`
→ nicht gefunden Rückgabe `-1`

Position ausgeben

Zwei getrennte Schleifen, da `text` sonst zu früh verändert wird!

`findstring` durch `replacestring` ersetzen

Ergebnistext ausgeben

Aufg. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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. 5.03: 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