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

Übung07: Informationstechnik (IT)

Tobias Schwalb / Michael Tansella

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
  - Christoph Roth, Tel. 608-42880, Email: christoph.roth@kit.edu

Inhalt: Übung07 – Teil 1

- Aufg. 6.01: Verständnisfragen
- Aufg. 6.02: OOP, Klassendefinition und Vererbung
- Aufg. 6.03: Dateiverarbeitung
- Aufg. 6.04: Arrays und Sortieren

Aufg. 6.01: Verständnisfragen Lsg. (1)

1. Die Methoden einer abgeleiteten Klasse können auf die private-Elemente der Basisklasse zugreifen. Falsch

2. Eine abgeleitete Klasse erbt eine public-Methode der Basisklasse nicht, wenn sie selbst eine public-Methode mit gleichem Namen besitzt. Falsch

3. Polymorphe Klassen werden in C++ mit Hilfe von virtual-Methoden implementiert.

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

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

Aufg. 6.01: Verständnisfragen Lsg. (2)

6. 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

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

8. Ein Algorithmus ist eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten.

9. Dynamische Finitheit besagt, dass das Verfahren zu jedem Zeitpunkt nur endlich viel Speicherplatz benötigen darf.

10. Komplexitätstheorie bezeichnet das Verhalten von Algorithmen bezüglich Ressourcenbedarf, wie Rechenzeit und Speicherbedarf.

Aufg. 6.01: Verständnisfragen Lsg. (3)

11. Berechenbarkeitstheorie besagt, dass:
   i. der Algorithmus bei denselben Voraussetzungen das gleiche Ergebnis liefern muss. Falsch
   ii. jeder Schritt des Verfahrens tatsächlich ausführbar sein muss. Falsch
   iii. das Verhalten bezüglich der Terminierung (ob der Algorithmus überhaupt jemals erfolgreich beendet werden kann) erfüllt sein soll. Richtig

12. Ein Algorithmus kann mit Pseudo Code oder Nassi-Shneiderman Diagrammen oder Ablaufdiagrammen beschrieben werden.

13. Der Quicksortalgorithmus benötigt, abgesehen von dem für die Rekursion benötigten Platz auf dem Aufruf-Stack, keinen zusätzlichen Speicherplatz. Richtig

14. Die Laufzeit des Quicksortalgorithmus ist abhängig von der Zerlegung des Feldes (Aufbau des Feldes - Zufall).

15. Merge Sort ist ein Sortieralgorithmus, der auf dem Prinzip Teile und Herrsche basiert.

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 );
```

Richtig / Falsch

Falsch:
1. Polymorphie funktioniert nur für Methoden, die die selbe Parameterliste und den selben Rückgabewert haben. Der Bezeichner virtual macht hier deshalb keinen Sinn.
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: Ein korrektes Beispiel für Polymorphie ist in Übung 5 Teil 2 Folie 11 dargestellt.

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

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!

```
C:\WINDOWS\system32\cmd.exe
Konstruktor der Klasse B
Konstruktor der Klasse D
Konstruktor der Klasse X
Bye, bye!
Destruktor der Klasse X
Destruktor der Klasse D
Destruktor der Klasse B
Drücken Sie eine beliebige Taste . . .
```

Aufg. 6.03: 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. 6.03: 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. 6.03: 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. 6.03: 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. 6.03: 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. 6.03: 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. 6.03: 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. 6.03: Dateiverarbeitung Lsg. (7)

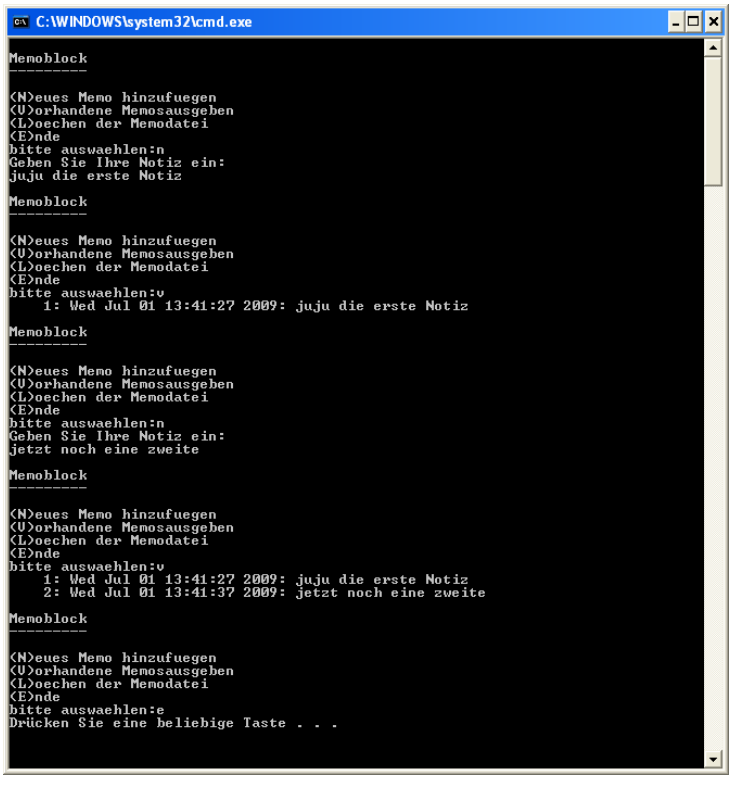
```
int main() {
  MemoBlock Memo;

  Memo.setfilename( "memo.txt" );

  Memo.menu();

  return 0;
}
```

Objekt der Klasse anlegen

Dateinamen festlegen

Menü starten

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



Aufg. 6.04: Arrays und Sortieren

- Schreiben Sie ein Programm, welches mit Hilfe eines InsertionSort-Algorithmus ein 1-dimensionales Array in aufsteigender Reihenfolge sortiert. Testen Sie anschließend Ihr Programm mit geeigneten Testarrays.
- Der folgende Pseudocode beschreibt den InsertionSort-Algorithmus.
- Hinweis: Beachten Sie, dass bei C++ ein Array an der Stelle 0 beginnt, dies ist nicht im Pseudocode berücksichtigt.

```
1 for j = 2 to length[A]
2 do key = A[j]
      //Füge A[j] ein in die sortierte Folge A[1 .. j – 1].
3     i = j – 1
4     while i > 0 and A[i] > key
5     do A[i + 1] = A[i]
6        i = i – 1
7     A[i + 1] = key
```


Aufg. 6.04: Arrays und Sortieren Lsg. (1)

```
int main() {
  long matrix[10] = {546, 21, 65, 1, 987, 88, 654, 5, 98, 123};
  int i, j, key;

  for( j = 1; j <= 9; j++ ) {
    key = matrix[j];
    i = j - 1;
    while( i > -1 && matrix[i] > key ) {
      matrix[i+1] = matrix[i];
      i = i - 1;
    }
    matrix[i+1] = key;
  }

  for( i = 0; i <= 9; i++ )
    cout << setw( 4 ) << matrix[i];
  cout << endl;

  return 0;
}
```

Startet bei 1, da Array bei 0 beginnt

Jedes Element nach dem Ersten

Schiebt die größeren Elemente nach hinten im Array

Fügt das Element an der richtigen Stelle ein

Arrayausgabe