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 05: Informationstechnik (IT)

Marc Weber

Teil 1: Besprechung Aufgabenblatt 5

Aufg. 5.01: Verständnisfragen Lsg. (1)

a) Die Methoden einer abgeleiteten Klasse können auf die **`private`**-Elemente der Basisklasse zugreifen. → Falsch

b) Eine abgeleitete Klasse erbt eine **`public`**-Methode der Basisklasse nicht, wenn sie selbst eine **`public`**-Methode mit gleichem Namen besitzt. → Falsch (unterschiedlicher Prototyp oder bei gleichem Prototyp über Basisklassenoperation)

c) Methoden einer abgeleiteten Klasse können auf **`protected`** Elemente der Basisklasse zugreifen. Die **`protected`** Elemente sind aber von außerhalb der Basisklasse und deren Kindklassen nicht zugreifbar. → Richtig

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

Aufg. 5.01: Verständnisfragen Lsg. (2)

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

f) 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

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

Aufg. 5.01: Verständnisfragen Lsg. (3)

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

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

- Aufg. 5.01: Verständnisfragen

Aufg. 5.02: 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. 5.02: OOP, Klassendefinition und Vererbung

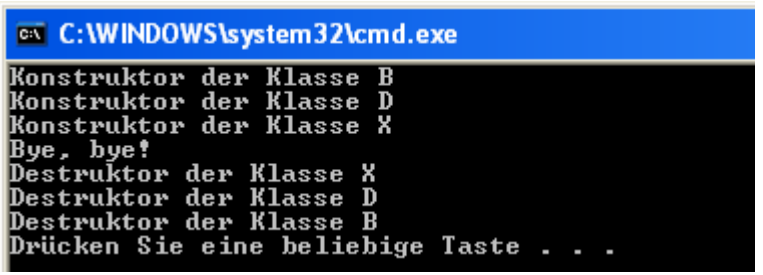
Lsg. (2)

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. 5.02: OOP, Klassendefinition und Vererbung

Aufg. 5.03: 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 kann dabei beliebig (auch unterschiedlich) sein. Wenn **`findstring`** nicht im String gefunden wird, soll eine Fehlermeldung ausgegeben werden.

- Beispielausgabe

Aufg. 5.03: 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" << endl;
    return 1;
  }
```

Einbinden der Bibliothek für String

Stringvariablen zum Speichern der Eingaben

Einlesen einer Zeile mit Leerzeichen

Einlesen eines einzelnen Worts

Prüfen, ob die Zeichenkette vorhanden ist

Programm mit Fehlermeldung beenden

Aufg. 5.03: 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 und sich dadurch Positionen evtl. verschieben!

`findstring` durch `replacestring` ersetzen

Ergebnistext ausgeben

Bonus: Das Programm erzeugt unter gewissen Umständen eine Endlosschleife. Wann?

- Aufg. 5.03: Stringmanipulation

Vielen Dank für Ihre Aufmerksamkeit

Marc Weber
Karlsruher Institut für Technology (KIT) – ITIV
marc.weber3@kit.edu