RECHEN- UND
KOMMUNIKATIONSZENTRUM DER RWTH AACHEN

Jürgen Dietel, Willi Hanrath

Klausur zum C++-Kurs für MaTAs im SS 2007
am 29. Juni 2007
Dauer: 2 Stunden

Name:
Vorname:
Kenn-Nummer:
Matrikelnummer:
Unterschrift:

| | | | max. Punktzahl |
|---|---|---|---|
| Aufgabe | 1) | | (10=10) |
| Aufgabe | 2) | | (10=10) |
| Aufgabe | 3) | | (6=6) |
| Aufgabe | 4) | | (3+3+3=9) |
| Aufgabe | 5) | | (10=10) |
| Aufgabe | 6) | | (4+4+4+4+2=18) |
| Aufgabe | 7) | | (9+3+3+3=18) |
| Aufgabe | 8) | | (11+5+3=19) |
| maximale Summe: | | | (100) |

| Gesamtpunkte: | Note: |
|---|---|
| | |

**Aufgabe 1**: *10 Punkte*

**Standarddeklarationsdateien.** Welcher `#include`-Befehl ist jeweils nötig, damit man die genannte Variable, Konstante, Funktion oder Klasse aus der C++-Standardbibliothek im C++-Quelltext benutzen kann?

a) `std::cout`

b) `int system(const char* KommandoAlsCString)`

c) `double sqrt(double radikand)`

d) `std::ifstream`

e) `void exit()`

f) `EXIT_SUCCESS`

g) `size_t strlen(const char text[])`

h) `std::cin`

i) `std::ostream`

j) `int isalpha(int zeichen)`

**Aufgabe 2**: *10 Punkte*

**Fragen zu Zeigern und Referenzen.**

a) Definieren Sie einen Zeiger `p1` auf ein `char`:

b) Definieren Sie einen konstanten Zeiger `p2` auf ein `char`:

c) Definieren Sie einen Zeiger `p3` auf ein konstantes `char`:

d) Definieren Sie einen konstanten Zeiger `p4` auf ein konstantes `char`:

e) Definieren Sie eine Referenz `r1` auf ein `char`:

f) Definieren Sie eine Referenz `r2` auf ein konstantes `char`:

g) Gibt es konstante Referenzen, etwa auf ein `char`: `char & const r3 = ...;`?
ja ☐ nein ☐

h) Gibt es Zeiger auf Referenzen, etwa: `char &* p5 = ...;`?
ja ☐ nein ☐

i) Gibt es Referenzen auf Zeiger, etwa: `char *& r4 = ...;`?
ja ☐ nein ☐

j) Gibt es Referenzen auf Referenzen, etwa: `char && r5 = ...;`?
ja ☐ nein ☐

**Aufgabe 3**: *6 Punkte*

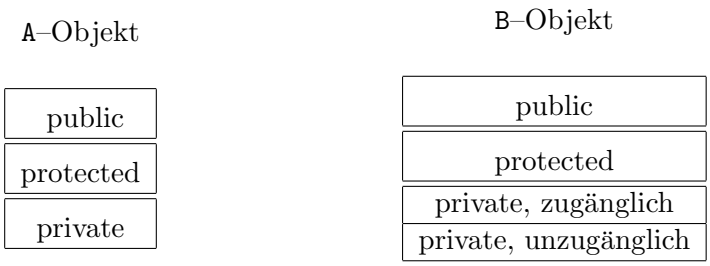
**Zugriffsabschnitte.** Es sei eine C++-Klasse gegeben. Tragen Sie in folgende Tabelle am Kreuzungspunkt von i. Zeile und j. Spalte ein $\times$ ein, wenn die in der i. Zeile genannte Funktion auf Komponenten der Klasse zugreifen darf, die zu dem in der j. Spalte genannten Zugriffsabschnitt der Klasse gehören:

| | `private` | `protected` | `public` |
|---|---|---|---|
| eigene Memberfunktion | | | |
| beliebige Funktion | | | |
| befreundete Funktion | | | |
| Memberfunktion einer beliebigen Klasse | | | |
| Memberfunktion einer direkt abgeleiteten Klasse | | | |
| Memberfunktion einer befreundeten Klasse | | | |

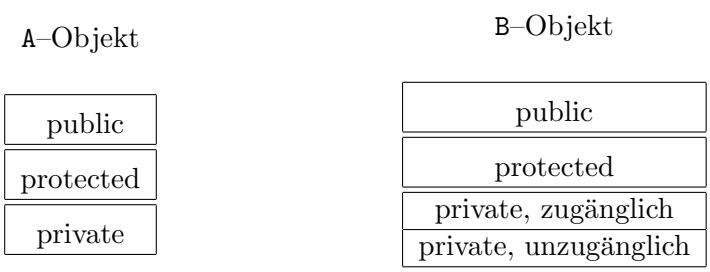
**Aufgabe 4**: *3+3+3=9 Punkte*

**Fragen zur Vererbung.** Eine Klasse verfügt bekanntlich über die drei Zugriffsabschnitte `public`, `protected` und `private`, wobei man den `private`–Abschnitt noch in (`private, zugänglich`) und (`private, unzugänglich`) unterteilen kann. Machen Sie durch Pfeile in folgenden Schaubildern klar, in welchem Zugriffsabschnitt eines `B`–Objektes die von der Klasse `A` geerbten Komponenten sind!

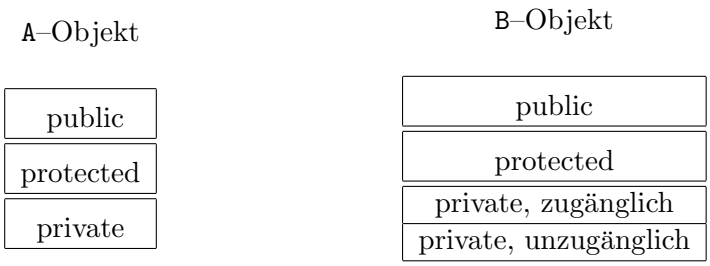
a) `class B : public A {...};`




b) `class B : protected A {...};`




c) `class B : private A {...};`



**Aufgabe 5**: *10 Punkte*

**Potentielle Arrayindizierungen.** Welche Zeilen in dem Programmausschnitt aus Listing 1, das auf den Definitionen von Listing 2 beruht:

```
42   a1 = a[1];
43   a2 = ar[2];
44   x  = (a+7)[0];
45   y  = (a+2)[-1];
46   a2 = (&ar+2)[0];
47   b2 = b[2];
48   c0 = c[0];
49   d3 = d[3];
50   h4 = h[4];
51   e  = anhaengen("hallo", "␣du")[7];
```

Listing 1: potentielle Indizierungen von Arrays in C++

```
1  #include <iostream>
2  #include <string>

4  using namespace std;

6  string anhaengen(const string& quelle,
7                   const string& ziel) {
8    return quelle + ziel;
9  }

11 int main() {

13   int    a[8];
14   int*   b = new int[3];
15   int*   c = new int[0];
16   int*   d = new int[2];
17   string h = "hallo";

19   int  a1 = -1;
20   int& ar = a1;
21   int  a2 = -2;
22   int  x  = -3;
23   int  y  = -4;
24   int  b2 = -5;
25   int  c0 = -6;
26   int  d3 = -7;
27   char h4 = 'y';
28   char e  = 'z';
```

Listing 2: Grundlage für potentielle Indizierungen von Arrays in C++

enthalten eine korrekte Indizierung, also eine korrekte Anwendung des Operators `[]`?

Dabei gilt eine Indizierung als nicht korrekt, falls sie vom Compiler abgelehnt wird oder zu einem Laufzeitfehler führen könnte.

ja nein

a) ☐ ☐ korrekte Indizierung in Zeile 42?
b) ☐ ☐ korrekte Indizierung in Zeile 43?
c) ☐ ☐ korrekte Indizierung in Zeile 44?
d) ☐ ☐ korrekte Indizierung in Zeile 45?
e) ☐ ☐ korrekte Indizierung in Zeile 46?
f) ☐ ☐ korrekte Indizierung in Zeile 47?
g) ☐ ☐ korrekte Indizierung in Zeile 48?
h) ☐ ☐ korrekte Indizierung in Zeile 49?
i) ☐ ☐ korrekte Indizierung in Zeile 50?
j) ☐ ☐ korrekte Indizierung in Zeile 51?

**Aufgabe 6**: *4+4+4+4+2=18 Punkte*

**Ein- und Ausgabe.**

a) Verfassen Sie eine C++-Funktion

```
void kopie()
```

die zeichenweise alles aus der Standardeingabe liest und auf die Standardausgabe schreibt.

Erlaubt sind dabei aus der C++-Standardbibliothek nur die mit Standardein- bzw. -ausgabe verbundenen Standardobjekte und dazugehörige Operatoren.

b) Erweitern Sie den Quelltext aus der vorigen Teilaufgabe so, daß sowohl die Eingabe als auch die Ausgabe sich auf je eine Datei beziehen.

Dabei sollen die Namen von Ein- bzw. Ausgabedatei Parameter der Funktion `kopie` vom Standardtyp `std::string` sein.

c) Werfen Sie selbstdefinierte Ausnahmen

```
Eingabefehler
```

oder

```
Ausgabefehler
```

wenn beim Öffnen der Dateien aus der vorigen Teilaufgabe Fehler auftreten.

Definieren Sie dazu die beiden Ausnahmen geeignet.

d) Schreiben Sie ein Testprogramm für die Funktion `kopie` aus dem vorigen Aufgabenteil, das die benötigten Dateinamen seiner Kommandozeile entnimmt und eventuell von `kopie` geworfene Ausnahmen passend behandelt.

e) Welche zusätzlichen Maßnahmen muß man in dem Quelltext aus Aufgabenteil 6c ergreifen, damit Leerraum (also z. B. Leerzeichen oder Zeilentrenner) nicht überlesen wird?

Achten Sie auch auf die jeweils nötigen **`#include`**-Befehle und Namensbereiche.

**Aufgabe 7**: *9+3+3+3=18 Punkte*

**Vektoren.** Es soll ein Array von ganzen Zahlen sortiert werden, hinter dem sich eine geeignet parametrisierte Konkretisierung der Standardklasse `vector` verbirgt, die folgende Eigenschaften besitzt:

- Sie stellt eine Verallgemeinerung der bekannten C-Arrays dar.
- Sie gehört zum Namensbereich `std` und wird durch `#include <vector>` deklariert.
- Der Standardkonstruktor erzeugt ein leeres Array (0 Elemente).
- Mit Hilfe von `void vector<T>::push_back(const T& wert)` kann man ein Element ans Ende eines Arrays anhängen.
- Die Größe eines solchen verallgemeinerten Arrays erhält man durch Aufruf der Elementfunktion `size()`.
- Der Indizierungsoperator ist für `vector` definiert.
- Bei der Sortierung darf angenommen werden, daß für den Elementtyp des Arrays sowohl Zuweisungs- als auch Vergleichsoperatoren definiert sind.

a) Schreiben Sie eine Funktion, die ein solches Array von ganzen Zahlen sortiert. Verwenden Sie als Verfahren Sortieren durch Auswahl (auch „Selection Sort“ genannt):
   i) Das Array besteht aus einem sortierten Teil vorne und einem unsortierten Teil hinten.
   ii) Im unsortierten Restarray wird das Minimum gesucht und mit dem ersten Element des Restarrays vertauscht. Dadurch wächst der sortierte Teil um ein Element.
   iii) Dieses Vorgehen wird so lange wiederholt, bis kein unsortierter Teil mehr existiert.

b) Definieren Sie den üblichen Ausgabeoperator für den Arraytyp, der der Sortierfunktion übergeben wird, und zwar so, daß z. B. das Array mit den Zahlen 5, 114 und 23 so:

```
(5,114,23)
```

   ausgegeben wird.

c) Verfassen Sie ein Testprogramm für Sortierung und Ausgabe mit dem Beispiel aus 7b: Erzeugen Sie dazu ein Array wie im Beispiel aus Aufgabenteil 7b, geben Sie es unsortiert aus (unter Benutzung des Operators aus Aufgabenteil 7b), sortieren Sie es (unter Benutzung der Funktion aus Aufgabenteil 7a) und geben Sie es zum Schluß nochmal aus.

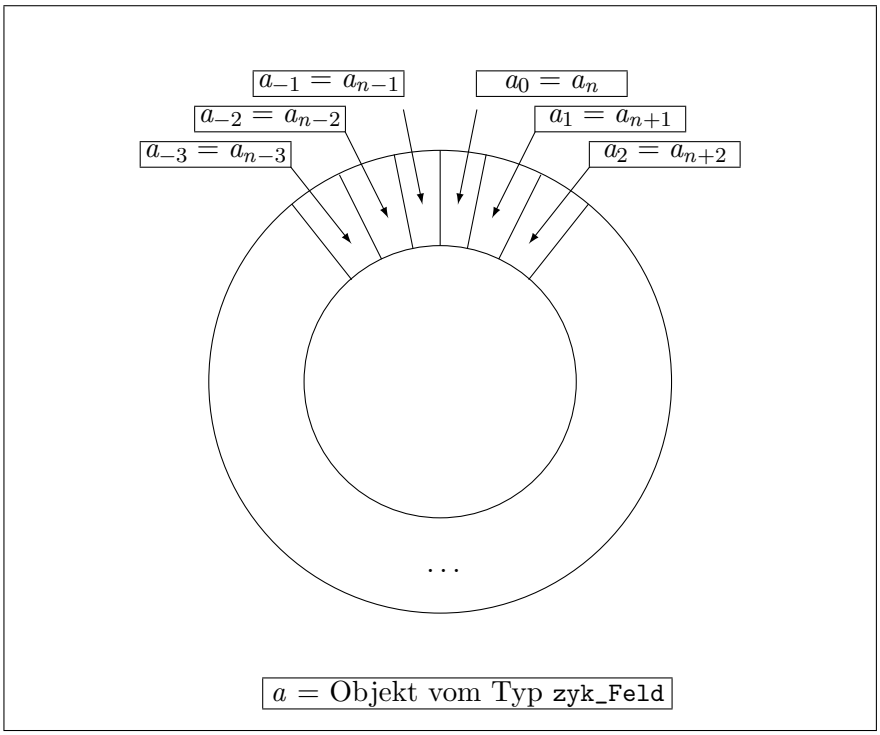
d) Welche Änderungen sind nötig, damit sowohl die Sortierung als auch der Ausgabeoperator auf Arrays von beliebigem Typ angewandt werden können? Notieren Sie dazu, wo und was geändert werden muß.

**Aufgabe 8**: *11+5+3=19 Punkte*

**Zyklisches Feld.** Entwerfen und implementieren Sie eine `template`–Klasse

```
template <typename T>
class zyk_Feld
{
  ...
};
```

zur Realisierung eines zyklischen Feldes einer positiven, ganzzahligen Länge `n` vom Typ `T`, d.h. eines Feldes mit einem Index–Zugriff der folgenden Art:




(Überlauf und Unterlauf sind nicht möglich, es wird einfach "*im Kreis herum*" weiter zugegriffen!)

Genauere Anforderungen:

a) Konstruktor mit `unsigned`–Argument für die Feldlänge, das eigentliche Feld vom Typ `T` ist dynamisch anzulegen,
   - im Fall der angegebenen Feldlänge 0 oder Speichermangel ist eine geeignet zu definierende Exception auszuwerfen (in Schnittstelle angeben!),
   - aufgrund der dynamischen Komponente sind weitere erforderliche Details zu berücksichtigen.

b) Feldzugriffs–Operatorfunktion `[]` mit `int`–Argument, soll für variable und konstante (zyklische) Felder verwendbar sein. Bei variablen Feldern soll das gelieferte Element abgeändert werden können, im Fall eines konstantes Feldes soll das gelieferte Feldelement nicht abgeändert werden können!

c) Kurzes Hauptprogramm–Fragment mit Erzeugung eines zyklischen Feldes und Abfangen möglicher Fehlerfälle.