UNIVERSITÄT HEIDELBERG
Zukunft. Seit 1386.
IWR Interdisciplinary Center for Scientific Computing

IPI: Einführung in die Praktische Informatik
Wintersemester 2018
Prof. Björn Ommer
ommer@uni-heidelberg.de

Übungsblatt 9: Klassen

Abgabe am 9.1.2019, 13:00.

Aufgabe 1: Semester Rückblick **(10P)**

Im Laufe des Semesters wurden schon verschiedene Themen behandelt. Diese Aufgabe soll Ihnen wie auch uns dazu dienen einen Überblick zu bekommen wie viel der verschiedenen Themen noch präsent sind.
Folgendes gilt zu beachten:

- Jede Frage kann mehrere korrekte Antworten haben, mindestens jedoch eine.
- Es muss keine Erklärung für die Wahl der Antwort(en) gegeben werden.
- Jede Frage wird einzeln betrachtet und gibt, wenn sie korrekt beantwortet wurde 0.5 Punkte.
- Eine Frage ist korrekt beantwortet, wenn alle richtigen und keine falsche Antwortmöglichkeit gewählt wurde.
- Lösungen der Multiple Choice Aufgabe dürfen eingescannt als pdf abgegeben werden oder in Form einer Antwortentabelle wie in Tabelle 1 gezeigt. Bei Abgabe einer gescannten Lösung achten Sie bitte auf folgendes:
  - Um eine Frage zu beantworten kreuzen Sie diese bitte an: $\square \rightarrow \boxtimes$.
  - Sollten Sie eine Frage fehlerhaft angekreuzt haben und möchten dies korrigieren ohne zusätzlichen Papiermüll zu verursachen füllen Sie bitten das Kästchen komplett aus: $\boxtimes \rightarrow \blacksquare$.

Frage	Antworten
1	3
2	1, 2
3	1
⋮	⋮

Table 1: Beispiel einer Antwortentabelle für die Multiple Choice Aufgabe.

1: Gegeben sei das Alphabet `{a, b, A, B, S}`, die Axiom-Menge `{S}` und die Regeln `{S -> ABS, S -> AB, A -> aA, A -> a, B -> bA}`. Welche der folgenden Worte lassen sich ausgehend von den Axiomen mit den Regeln erzeugen?

- ☐ (1) aaaaba
- ☐ (2) abaaba
- ☐ (3) aababaa
- ☐ (4) aabaab

2: Gegeben sei der folgende endliche Automat über den Eingabesymbolen `{a,b,.}`, wobei `"."` das Ende einer Eingabe symbolisiert und `{WAHR, FALSCH}` Endzustände symbolisieren:

```
1  // Zustand:Eingabe:Folgezustand
2  q1:a:q2
3  q1:b:q3
4  q1:.:FALSCH
5  q2:a:q1
6  q2:b:q3
7  q2:.:WAHR
8  q3:a:q3
9  q3:b:q3
10 q3:.:WAHR
```

Welche der folgenden Aussagen ist korrekt?

- ☐ (1) Nur für Eingaben, bei denen das Symbol a vor dem Symbol b erscheint, ist der Endzustand WAHR
- ☐ (2) Nur für Eingaben mit einer ungeraden Zahl a oder mindestens einem b ist der Endzustand WAHR
- ☐ (3) Nur für Eingaben mit Symbol b vor Symbol . ist der Endzustand WAHR

3: Welche der folgenden Aussagen ist korrekt?

☐ (1) Ein terminierender, nichtdeterministischer Algorithmus ist nicht determiniert.

☐ (2) Fast alle Turingmaschinen halten für jede Eingabe an.

☐ (3) Nicht-Deterministische Automaten können nicht mehr Sprachen erkennen als deterministische Automaten.

4: Der Ausdruck `-(+(3, *(5, 8)), *(16, +(7, 9)))` ist in

☐ (1) Präfix - Schreibweise

☐ (2) Infix - Schreibweise

☐ (3) Postfix - Schreibweise

5: Wenn die Syntax stimmt,

☐ (1) stimmt auch die Semantik

☐ (2) gibt es beim Kompilieren keine Fehler

☐ (3) wird das Ergebnis rekursiv berechnet

6: `n!` kann

☐ (1) nur iterativ berechnet werden

☐ (2) nur rekursiv berechnet werden

☐ (3) sowohl iterativ, als auch rekursiv berechnet werden

7: Was wird ausgegeben:

```
1  void func(int zahl){
2      zahl = 0;
3  }
4
5  int main() {
6      int zahl = 5;
7      func(zahl);
8      print(zahl);
9  }
```

☐ (1) zahl

☐ (2) 0

☐ (3) 5

8: Was wird ausgegeben:

```
1  int func(int zahl){
2      return zahl-1;
3  }
4
5  int main() {
6      int zahl = 5;
7      print(func(zahl));
8  }
```

☐ (1) -1

☐ (2) 4

☐ (3) 5

9: Was wird ausgegeben:

```
1  void func()
2  {
3      int y=0;
4      for (int i=0; i<15; i++)
5      {
6          int y=y+4;
7      }
8      print(y);
9  }
10
11 int main() {
12     int zahl = 5;
13     func();
14 }
```

☐ (1) 0

☐ (2) 5

☐ (3) 60

10: Was wird ausgegeben:

```
1  void func() {
2      print(zahl);
3  }
4
5  int main() {
6      int zahl = 5;
7      func();
8  }
```

☐ (1) Fehler

☐ (2) 0

☐ (3) 5

11: Welche Aussage ist richtig?

☐ (1) Jede `for`- kann in eine `while`-Schleife transformiert werden.

☐ (2) Jede `while`- kann in eine `for`-Schleife transformiert werden.

☐ (3) Jede `for`- kann in eine `while`-Schleife transformiert werden und umgekehrt.

12: Welche dieser Feld-Zugriffe sind *gültig*? Ein Zugriff gilt auch als ungültig wenn er zwar kompiliert, aber zu einem ungültigen Speicherzugriff führen kann.

```
1 int feld[2];
```

☐ (1) `feld[-1]`

☐ (2) `feld[0]`

☐ (3) `feld[1]`

☐ (4) `feld[2]`

☐ (5) `feld[3]`

13: Zeichenketten werden in C und C++ durch ein `char`-Feld realisiert. Welches der folgenden `char`-Felder kann verwendet werden um die Zeichenkette `"Test"` (ohne die Anführungszeichen) korrekt abzuspeichern?

☐ (1) `char str[3]`

☐ (2) `char str[4]`

☐ (3) `char str[5]`

☐ (4) `char str[6]`

☐ (5) `char str[7]`

14: Welcher Datentyp eignet sich am besten als aussagenlogischer Wert?

☐ (1) integer

☐ (2) boolean

☐ (3) character

15: Gegeben sei der Befehl: `int f(float);`.
Welche von den vier folgenden Aussagen ist wahr?

- ☐ (1) Die Funktion `f` nimmt ein Argument vom Typ `int` entgegen und gibt einen Wert vom Typ `float` zurück.
- ☐ (2) Die Funktion `f` nimmt ein Argument vom Typ `float` entgegen und gibt einen Wert vom Typ `int` zurück.
- ☐ (3) Die Funktion `f` ist selbst vom Typ `float`.
- ☐ (4) Keine der obigen Antwortmöglichkeiten.

16: Der Wert `32.54` kann durch welchen Datentypen dargestellt werden?

- ☐ (1) `double`
- ☐ (2) `void`
- ☐ (3) `int`
- ☐ (4) `bool`

17: Welchen Wert hat der Ausdruck: `((false && true) || false || true)`?

- ☐ (1) `0`
- ☐ (2) `1`
- ☐ (3) `false`
- ☐ (4) Keine der obigen Antwortmöglichkeiten.

18: Betrachten Sie den folgenden Code-Aussschnitt:

```
1  int sum = 10, **acc = new int*[10];
2  *(acc+2) = &(*(&sum));
```

Welche der folgenden Vergleiche sind "`true`"?

- ☐ (1) `( &(*(acc[2])) == &sum )`
- ☐ (2) `( *(*(acc+2)) == *(&sum) )`
- ☐ (3) `( acc[0][2] == sum )`

19: Was ist die Ausgabe des folgenden Programms?

```
1 #include <iostream>
2 int main() {
3   const int i = 20, j = 15, *const ptr = &i;
4   (*ptr)++;
5   ptr = &j;
6   std::cout << i;
7 }
```

☐ (1) 20

☐ (2) Compile-Fehler in Zeile 4

☐ (3) Compile-Fehler in Zeile 5

20: Was wird hier ausgegeben?

```
1 void frage3() {
2   int** a, *b;
3   int c = 4, *d = &(c);
4   a = &d;
5   b = *a;
6   std:: cout << b << std::endl;
7 }
```

☐ (1) Die Speicheradresse von a

☐ (2) Die Speicheradresse von b

☐ (3) Die Speicheradresse von c

Aufgabe 2: Listen-Klasse **(10P)**

In der Vorlesung haben Sie Funktionen und Datenstrukturen kennengelernt `int`-Zahlen in einer einfach verketteten Liste zu verwalten.

a) Die in der Vorlesung vorgestellte Version hat den Nachteil, dass Funktionen und Daten getrennt sind. Schreiben Sie deshalb, basierend auf den in `intlist.cc` implementierten Funktionen, eine Klasse für eine Integerliste. Die Klasse sollte dabei mindestens folgendes Interface erfüllen:

```
1  class IntList
2  {
3  public:
4    // Konstruktor, erzeugt eine leere Liste
5    IntList();
6    // Destruktor, loescht gesamten Listeninhalt
7    ~IntList();
8    // Gibt Anzahl der Elemente zurueck
9    int getCount();
10   // Gibt zurueck, ob die Liste leer ist
11   bool isEmpty();
12   // Gibt die Liste aus
13   void print();
14   // Fuegt die Zahl 'element' an der (beliebigen) Position 'position' ein
15   void insert( int element, int position );
16   // Loescht das Element an der Position 'position'
17   void remove( int position );
18   // Gibt den Wert des Elements an der Position 'position' zurueck
19   int getElement( int position );
20 private:
21   // ... (hier folgen private Member der Klasse)
22 };
```

Überlegen Sie sich, welche privaten Daten und Methoden Sie brauchen und welche Funktionen Konstruktor und Destruktor ausführen sollten. Anhand der Methodensignaturen erkennen Sie bereits, dass der Benutzer nur mit `int`s zu tun hat, Dinge wie die Erzeugung von Listenelementen und das Hantieren mit Pointern ist Aufgabe Ihrer Klasse!

b) Es gibt in C++ eine Faustregel namens "Rule of Three"[1]. Diese Regel besagt, dass die drei Methoden

- Destruktor
- Copy-Konstruktor
- Zuweisungsoperator

immer zusammen auftreten sollten. Implementiert man eine dieser Methoden explizit, sollte man auch die anderen bereitstellen. Die dieser Regel zugrundeliegende Annahme ist folgende: Ist man für eine dieser Methoden mit der impliziten Standardvariante des Compiler nicht zufrieden, trifft dies höchstwahrscheinlich auch auf die anderen beiden Methoden zu. Dies ist meistens dann der Fall, wenn man mit Pointern hantiert.
Implementieren Sie für Ihre Klasse nun also die beiden fehlenden Methoden. Achten Sie bei der Implementierung des

- Copy-Konstruktors darauf, dass Sie eine *tiefe* Kopie[2] Ihrer Liste erstellen. Bloßes Kopieren der Pointer auf die Listenelemente (wie es der Compiler in seiner Default-Variante tun würde) reicht nicht aus, Sie müssen jedes Listenelement neu erstellen!
- Zuweisungsoperators darauf,
  - dass Sie die bereits vorhandenen Daten einer existierenden Liste "aufräumen", bevor Sie ihr neue Daten zuweisen, und
  - dass ein Objekt niemals "sich selbst" zugewiesen wird. Den Test, ob das aktuelle Objekt gleich einem als Referenz übergebenen anderen `other` ist, kann man relativ einfach mit `if (this != &other)` überprüfen.

[1] seit `C++11` eigentlich "Rule of Five", siehe
`http://en.wikipedia.org/wiki/Rule_of_three_%28C%2B%2B_programming%29`
[2] siehe `http://en.wikipedia.org/wiki/Object_copy`