ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik I - MaVt

Sommersemester 2007

Dr. Volker Roth
ETH Zürich, CAB G 81
CH-8092 Zürich

Tilman Lange <langet@inf.ethz.ch>

Exemplarische Prüfungsfragen zur Vorlesung Informatik I (D-MAVT)

Die vorliegende Fragesammlung soll Ihnen einen Eindruck davon vermitteln, was Sie in der Prüfung zu erwarten haben. Sie sollten alle Aufgaben lösen können. Bedenken Sie, daß Fragen über den in der Vorlesung noch nicht behandelten Stoff selbstverständlich auch in der Prüfung auftauchen können. Insofern kann (und will) diese Fragesammlung kein vollständiges Bild liefern.

Dieser Zettel zählt nicht als regulärer Übungszettel und ist insofern von der Zählung für das Testatkriterium ausgenommen. Es kann jedoch ein Bonuspunkt für das Erfüllen der Testatbedingungen erworben werden, indem zu wenigstens 50 % der Fragen (also 2 von 4 Aufgaben) auf diesem Zettel sinnvolle Lösungen abgegeben werden. Wie gewöhnlich ist hierzu eine Abgabe beim Übungsgruppenleiter bis zum 29./30. Mai erforderlich.

Aufgabe 1: C++-Syntax und komplexe Datentypen

a) In dieser Teilaufgabe geht es darum, Fehler in einem Programm zu finden und zu korrigieren. Im folgenden finden Sie vier verschiedene Implementierungen eines einfachen Primzahltests, welche alle einen (syntaktischen oder semantischen) Fehler enthalten.

Die Programmteile:

i)
```
1  bool isprime(int i) {
2    bool res = true;
3    int j = 2;
4    while (j<i) {
5      if (i%j = 0) {
6        // i ist durch j teilbar
7        res = false;
8      }
9    }
10   return res;
11 }
```

Fehler beim Compilieren:

```
prime1.cc: In function ‘bool
  isprime(int)’:
prime1.cc:5: non-lvalue in assignment
```

ii)
```
1  bool isprime(int i) {
2    int j;
3    while (j<i) {
4      if (i%j == 0) {
5        // i ist durch j teilbar.
6        return false;
7      }
8      j++;
9    }
10   return true;
11 }
```

Fehler bei der Ausführung:
Es werden sehr häufig Zahlen als prim angegeben, welche keine Primzahlen sind.

iii)
```
1  bool isprime(int i) {
2    for (int j=2; j<i; j++) {
3      if (i%j == 0)
4        // i ist durch j teilbar.
5        return false;
6      else
7        return true;
8    }
9  }
```

Fehler bei der Ausführung: Jede ungerade Zahl wird als prim angegeben.

iv)
```
1  bool isprime(int i) {
2    bool res = true;
3    for (int j=2; j<i; i++) {
4      if (i%j == 0)
5        // i ist durch j teilbar.
6        res = false;
7    }
8    return res;
9  }
```

Fehler bei der Ausführung: Das Programm läuft ewig und produziert keine Ausgabe.

Lösungsstruktur

- Identifizieren Sie die Programmzeile, in welcher sich der Fehler befindet, und erläutern Sie, was genau falsch ist (1 Punkt pro Programmteil).
- Korrigieren Sie den Fehler (1 Punkt pro Programmteil).

8 pts.

b) Betrachten Sie folgende Zuweisungsoperationen. Geben Sie zu jedem Ausschnitt eine mögliche Deklaration für alle vorkommenden Variablen an.

Lösungsstruktur: Pro vorkommenden einfachen Datentyp werden 0.5 bis 1 Punkt vergeben, für vollständig richtig deklarierte komplexe Datentypen bekommen Sie 2 Punkte. Pro Teilaufgabe werden maximal 2 Punkte vergeben.

Die Ausdrücke:

i)
```
c = new int;
d = *c;
```

ii)
```
b.me = &b;
b.data = "Hello World";
```

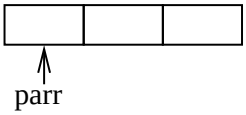
iii)
```
a.b[2] = 5.0f;
a.a = (a.b[2] > a.b[1]);
```

6 pts.

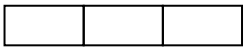
c) In dieser Tracing-Aufgabe können Sie ihr Wissen über Pointers und Arrays demonstrieren. Betrachten Sie das unten angegebene Programmstück, welches von oben nach unten abgearbeitet wird. Tragen Sie jeweils die Werte im Array `arr` in die Kästchen ein, und markieren Sie die Position, auf welche `parr` zeigt, mit einem Pfeil.

Verwenden Sie `u` für einen Array-Eintrag, der nicht definiert ist. Der Pointer in der ersten Skizze ist als Beispiel schon gegeben.

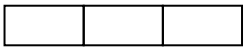
```
...
int arr[3];
int* parr;
parr = arr;
```



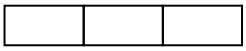

```
*parr = 7/3;
```



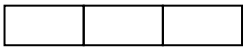
```
parr++;
```



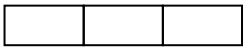
```
parr++;
```



```
arr[2] = 2;
```



```
*(--parr) = 5;
```



Lösungsstruktur: Insgesamt werden für diese Teilaufgabe maximal 5 Punkte vergeben. Fehlerhafte Angaben führen zu Punktabzug.

5 pts.

Aufgabe 2: Grosse Zahlen als doppelt verkettete Listen

Eine Variable vom Typ `int` hat eine bestimmte Grösse – typischerweise kann ein `int` Werte zwischen -2147483648 und +2147483647 annehmen. Um mit grösseren Zahlen rechnen zu können benötigt man eine andere Datenstruktur.

Sie sollen zuerst eine doppelt verkettete Liste implementieren, mit der beliebig viele Werte gespeichert und verarbeitet werden können. Anschliessend sollen zwei Funktionen auf Zahlen mit beliebig vielen Stellen programmiert werden.

a) **Implementieren einer doppelt verketteten Liste**

Die einzelnen Listenelemente sind wie folgt definiert:

```
struct element {
  element *prev, *next;
  int value;
};
```

Dabei ist in `value` immer eine Dezimalstelle der Zahl gespeichert; gültige Werte von `value` sind also nur 0 bis 9. `prev` und `next` sind Pointer auf die vorherige und die nachfolgende Ziffer (d.h. die Stellen "links" und "rechts" von der aktuellen Stelle). Für die hinterste Stelle wird `next=0` gesetzt; analog ist `prev` der ersten Stelle gleich `0`.
Mit dem Datentyp `element` wird dann eine Klasse `list` wie folgt definiert:

```
struct list {
    element *first, *last, *curr;

    list();                   // Konstruktor fuer eine leere Liste (Pointer auf 0 setzen!)
    int getDigit();           // gebe Ziffer des aktuellen Elementes zurueck,
                              // oder -1, falls die Liste leer ist oder
                              // curr auf kein gueltiges Element zeigt.
    bool addAtFront(int i);   // fuege eine Stelle an der Spitze der Zahl ein
                              // i ist eine positive, einstellige Zahl.
    void setToFront();        // setze curr auf das erste Listenelement
    void setToBack();         // setze curr auf das hinterste Listenelement
    bool hasElements();       // ueberpruefe, ob die Liste Elemente enthaelt
                              // gibt false zurueck, falls die Liste leer ist,
                              // und true sonst.
    bool moveForward();       // setze curr auf den Vorgaenger des aktuellen
                              // Elementes. Gibt false zurueck, falls das
                              // aktuelle Element das vorderste ist, sonst true
    bool moveBackward();      // setze curr auf den Nachfolger des aktuellen
                              // Elementes. Gibt false zurueck, falls das
                              // aktuelle Element das hinterste ist, sonst true
};
```

`first` zeigt auf das erste, und `last` auf das letzte Element (Stelle) der Liste (der gespeicherten Zahl). Beim Traversieren der Liste muss man sich jeweils das aktuelle Element merken. Dies geschieht in der Variable `curr`.

Implementieren Sie die folgenden Funktionen (i) - (iii) gemäss Spezifikation in der Definition im `struct list`. Achten Sie dabei besonders auf einen korrekten Umgang mit Pointern. Sie dürfen in Ihren Funktionen andere Funktionen, welche oben definiert sind, verwenden, ohne diese zu implementieren. Nehmen Sie bei der Implementation an, daß alle Parameter korrekt sind.

i) `list();`
ii) `int getDigit();`
iii) `bool addAtFront(int i);`

Lösungsstruktur: Für die ersten beiden Teilaufgaben werden maximal je 1.5 Punkte vergeben, für die dritte Teilaufgabe maximal 3 Punkte. Pro Syntaxfehler werden 0.5 Punkte abgezogen.

6 pts.

b) **Implementieren von beliebig grossen Zahlen mit einer doppelt verketteten Liste.** Mit Hilfe der Struktur `list` soll nun die Klasse `langzahl` implementiert werden. Diese ist wie folgt definiert:

```
class langzahl {
  private:
    list *theList;

  public:
    langzahl();                // Konstruktor fuer eine Zahl ohne Wert.
    void setValue(int v);      // setzt die Zahl auf den Wert v.
    langzahl add(langzahl l2); // addiert zwei lange Zahlen und gibt das Ergeb-
                               // nis als langzahl zurueck
    void print();              // gibt die Zahl auf dem Bildschirm aus.
};
```

Aufgabe Implementieren sie die folgenden beiden Funktionen.

i) `void print();`

ii) `langzahl add(langzahl l2);`
Tipp: Überlegen Sie sich, wie sie von Hand zwei Zahlen addieren. Beachten Sie insbesondere:
- Wann ist die Addition fertig?
- Was tun Sie, wenn eine Zahl kürzer ist als die andere?
- Was passiert mit Überträgen ("behalte 1")?

Lösungsstruktur: Die `print`-Funktion gibt maximal 4 Punkte, die `add`-Funktion maximal 8 Punkte. Pro Syntaxfehler wird ein halber Punkt abgezogen.

12 pts.

c) **Theoretische Überlegungen** In dieser Teilaufgabe sollen sie die implementierte Klasse `langzahl` auf ihre Laufzeit hin untersuchen. Begründen Sie jeweils ihre Antworten!

i) Sei `l1` eine `langzahl` mit n_1 Stellen. Wie gross ist die Laufzeit der Funktionen `l1.print()` (in $\mathcal{O}$-Notation)?

ii) Seien `l1`, `l2` zwei lange Zahlen mit n_1 bzw. n_2 Stellen. Wie gross ist die Laufzeit der Funktionen `l1.add(langzahl l2)` (in $\mathcal{O}$-Notation)?

iii) Falls statt einer Ziffer in jedem Element der doppelt verketteten Liste k Ziffern gespeichert würden, welches wäre dann die Laufzeit (wiederum in $\mathcal{O}$-Notation)?
Tipp: Überlegen Sie sich, wie die Laufzeit der Funktionen `print` und `add` von der Anzahl Elementen in der Liste abhängt. Geben Sie an, wie viele Listenelemente benötigt werden, um eine Zahl mit n Stellen zu speichern, und leiten Sie daraus die Laufzeit der Funktion ab.

Lösungsstruktur: In jeder Teilaufgabe werden für die richtige Laufzeit und für die richtige Begründung je ein Punkt vergeben. Sie können also maximal 6 Punkte erreichen.

6 pts.

Aufgabe 3: Suche in Feldern (Array == Feld)

Hier geht es um Suche in Feldern. Hierzu wird zunächst ein Array a der Länge n mit `double`-Einträgen betrachtet.

a) Geben Sie zunächst Funktionen

```
double max(double *a, int n);
double min(double *a, int n);
```

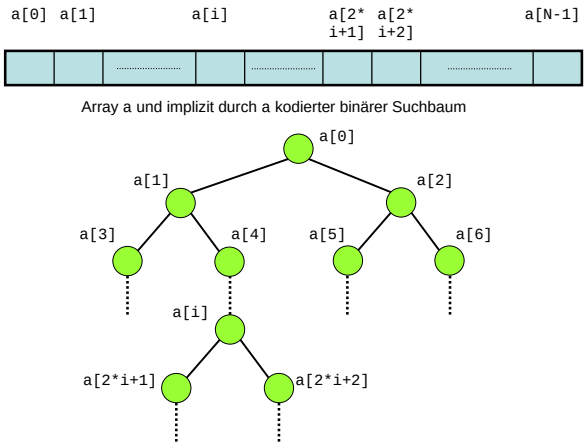
an, die das maximale/minimale Element im Feld a zurückliefern. Benennen Sie die Anzahl notwendiger Vergleiche ($<$,$>$).

Lösungsstruktur

- Erwartet wird die Angabe einer compilierbaren, kommentierten C++-Funktion als Abgabe (3 Punkte/Funktion).
- Begründen Sie die notwendige Vergleichsanzahl (1 Punkt).

7 pts.

b) Wir betrachten ein `double`-Feld a der Länge N (!), in das ein binärer Suchbaum zur Speicherung von n (!) Zahlen (i.A. $N \gg n$) auf folgende Art und Weise kodiert ist (vgl. Graphik).



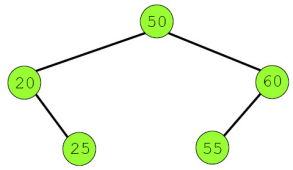
Nicht genutzte Elemente ($N - n$ viele) von a sind mit dem Wert `NAN` (Not-A-Number) belegt. Der Eintrag `NAN` ist wie ein `NULL` der Pointer-basierten Implementierung aus Vorlesung und Übung zu verstehen. Wir nehmen an, daß die Einträge von a folgende Struktur aufweisen:

$$\texttt{a[i]} \overset{<}{\underset{>=}{}} \texttt{a[2*i +1]}$$
$$\texttt{a[i]} \overset{>=}{\underset{<}{}} \texttt{a[2*i +2]}$$

für $i \geq 0$ und `a[i] != NAN`. Mit anderen Worten ist `a[i]` Wurzel eines (Teil-) Baumes, so ist `a[2*i +1]` Wurzel des linken und `a[2*i +2]` Wurzel des rechten Teilbaumes. Ist `a[2*i +1] == NAN`, so ist der linke Teilbaum leer. Analog ist der rechte Teilbaum leer, falls `a[2*i +2] == NAN`. Die Wurzel des Baumes ist in `a[0]` abgelegt. Ein konkreteres Beispiel:

0	1	2	3	4	5	6	7	8	9
50	20	60	NaN	25	55	NaN	NaN	NaN	NaN

Array a und implizit durch a kodierter binärer Suchbaum

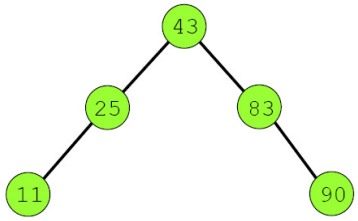


i) Zeichnen Sie den Baum der zu dem Feld

```
double a[] = {5,3,7,1,4,6,9};
```

gehört.

ii) Kodieren Sie den folgenden Baum im Array a der Länge $N = 10$



Lösungsstruktur Erwartet wird eine Skizze eines binären Suchbaumes. Für den zweiten Aufgabenteil wird die Skizze des zu dem abgebildetetem Baum korrespondierenden Feldes erwartet. Füllen Sie freie Felder mit `NAN` Einträgen auf.

6 pts.

c) In dieser Teilaufgabe verwenden wir die Suchbaumstruktur aus Aufgabe 3b weiter. a ist also ein Double-Feld der Länge N, in dem n Elemente in obiger Suchbaumstruktur abgelegt und $N - n$ freie (i.e. mit `NAN` belegte) Einträge vorhanden sind.

Implementieren Sie die C++-Funktionen

```
double max(double *a, int N);
double min(double *a, int N);
```

die Minimum und Maximum der n in a gespeicherten Elemente ausgeben. Ihre Funktionen sollen dabei den implizit durch a definierten Baum (s. Aufgabe 3b) traversieren und sich dabei die Suchbaum-Eigenschaft zunutze machen.

Lösungshinweis: Die Suchbaum-Eigenschaft besagt, daß es im linken Teilbaum nur Elemente gibt, die $\leq$ der Wurzel sind, während im rechten Teilbaum alle Elemente $>$ sind als die Wurzel. Machen Sie sich diese Eigenschaft zunutze! Verwenden Sie `bool isnan(double x)`, um zu testen, ob eine Variable den Wert `NAN` hat oder nicht.

Lösungsstruktur

- Erwartet wird die Angabe zweier compilierbarer, kommentierter C++-Funktionen. Sie können maximal 10 Punkte für die Funktionen erreichen (5 pro Funktion).
- Die Verwendung der rein sequentiellen Suche ohne Verwendung der Suchbaumstruktur ist *keine* zulässige Lösung.

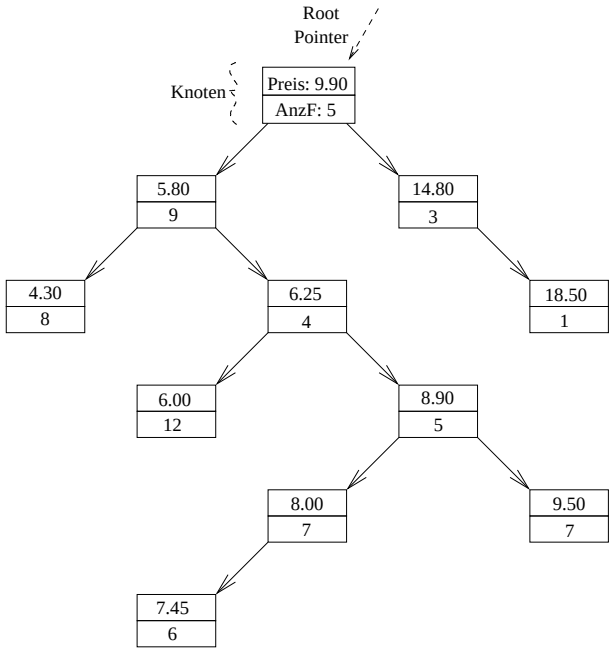
10 pts.

Aufgabe 4: Weinkellerverwaltung mittels Binärbaum

Nehmen Sie an, sie seien daran interessiert, ihren Weinkeller (der eine Vielzahl von Weinflaschen zu unterschiedlichen Einkaufspreisen enthält) mittels ihres neu erworbenen Informatik-I Wissens zu verwalten. Insbesondere wollen sie feststellen, wieviele Flaschen Wein es zu einem bestimmten Preis gibt. Benutzen Sie für die Lösung dieses Problems einen Binärbaum.

In diesem Binärbaum, sind die Elemente / Knoten nach Preisen geordnet. Jedes Element besitzt also als Schlüssel den **Preis**. Zusaetzlich gibt es einen Zähler, der angibt, wieviele Flaschen es zu diesem Preis gibt. Da wir einen Binärbaum konstruieren wollen, hat jedes Element höchstens zwei Nachfolgeknoten. Im linken Teilbaum werden diejenigen Flaschen verwaltet, die billiger als der Vorgängerknoten sind, und im rechten Teilbaum die jeweils teureren Flaschen.

Ein entsprechender Baum sieht beispielsweise folgendermassen aus:



a) Deklarieren Sie eine Struktur `element`, die Preis und Anzahl der Flaschen speichert. Zwei Zeiger sollen vorhanden sein (linker und rechter Nachfolgengeknoten).
Hinweis: Die Lösung hat die Form:

```
struct element{
  ...
};
```

4 pts.

Nehmen Sie im Folgenden an, dass ein solcher Binärbaum bereits exisitert. Der Wurzelknoten sei in einer globalen Variable `root` von Typ `element` gespeichert.

b) Implementieren Sie in C++ die Prozedur

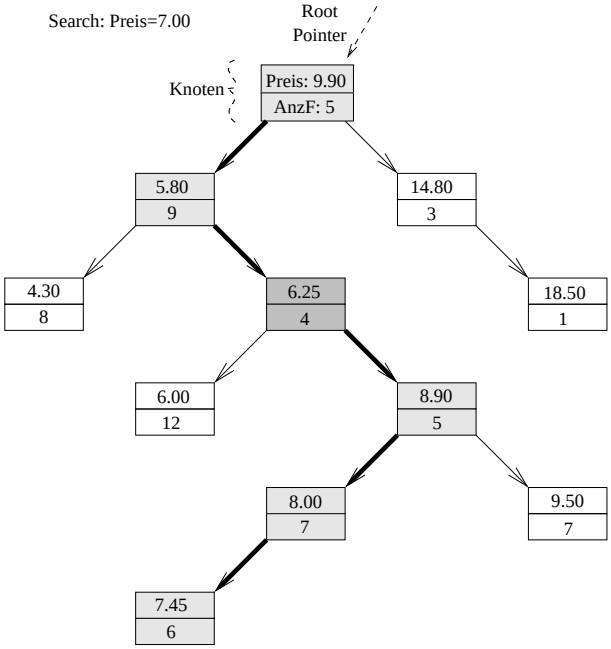
```
void InsertBottle(double price, int n);
```

Hinweis: Beim Einfuegen von neuen Flaschen zu einem gewissen Preis soll zuerst geprüft werden, ob schon ein Knoten zu diesem Preis existiert. Wenn ja, wird die Anzahl von Flaschen in diesem Knoten erhöht. Falls es noch keinen solchen Knoten gibt, dann soll ein neuer Knoten an der richtigen Stelle eingefügt werden.

5 pts.

c) Jetzt wollen wir wissen, wieviele Flaschen es zu einem gegeben Preis gibt. Bei erfolgreicher Suche soll die Anzahl von Flaschen und der entsprechende Preis ausgeben werden (mittels `cout`). Falls es keine Weine zu diesem Preis gibt, soll die Anzahl von Flaschen zum **nächst niedrigeren Preis** (und der nächst niedrigere Preis selbst) ausgegeben werden. Die Prozedur gibt den entsprechenden Knoten zurück.

Hinweis: Wegen der Anordnung der Knoten in einem Binärbaum, liegt der Knoten mit dem nächst niedrigeren Preis schon **auf dem Suchpfad** (siehe untenstehendes Beispiel). Es ist also zweckmässig, sich während der Suche zu merken, welcher Knoten zuletzt einen niedrigeren Preis hatte. Benützen Sie dafür die Hilfsvariable `lastlow`.



Implementieren Sie in C++ die Prozedur

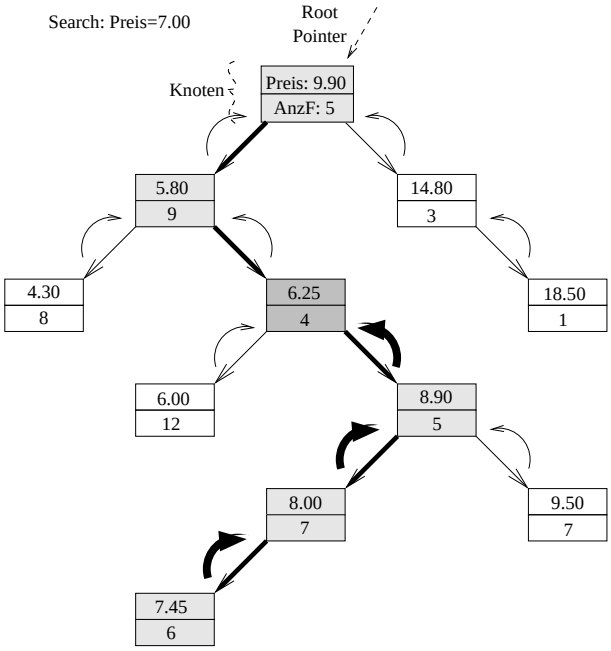
```
element FindBottles(double key);
```

6 pts.

Für manche Suchaufgaben ist es hilfreich, auch die jeweiligen Vorgängerknoten zu speichern. Das heisst, jeder Knoten hat einen zusätzlichen Pointer `parent`, der auf den Vorgängerknoten zeigt. Die entsprechend erweiterte Struktur sieht folgendmassen aus:

```
struct element{
 ...
 element *parent; /* Vorgaengerknoten */
 ...
};
```

d) Ähnlich zu Aufgabeteil 3 sollen Sie jetzt mit diesem neuen Pointer die Flaschen mit dem nächst kleineren Preis suchen (statt mit der Hilfsvariable `lastlow`). Der entsprechenden Baum zu dem Beispiel sieht folgenden massend aus:



Implementieren Sie die Prozedur

```
element FindBottlesUsingParent(double key);
```

4 pts.