Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dr. K. Simon

Studiengang Maschinenbau und Verfahrenstechnik

1. Vordiplom, Informatik I

Musterlösung

Herbst 2001

Freitag, 28. September 2001

Aufgabe 1: Syntax, Typen, Prozeduren und Funktionen *(30 Punkte)*

a)

i)
1. Zeile 7.
2. Der Index eines Arrayzugriffs muss ganzzahlig sein, ist aber in diesem Fall vom Typ `double`, also nicht ganzzahlig.
3. In Zeile 4: `int i;` oder In Zeile 7: `a[int(i)] = int(i);`

ii)
1. Zeile 7.
2. Beim zweiten Schleifendurchlauf wird in Zeile 7 auf $b[-1]$ zugegriffen. Dieses Element existiert nur im Traum.
3. In Zeile 6: `i++` statt `i--`.

   Nicht gelten lassen wir ein if-Statement, das die Zuweisung vor $i < 0$ schützt, da dies die Endlosschleife in Zeile 6 nicht behebt.

iii)
1. Zeilen 10 und 11.
2. `A` ist ein Typ. Es wird aber in den Zeilen 10 und 11 als Variable verwendet.
3. Alle Vorkommen von `A.` in den Zeilen 10 und 11 müssen durch `c.` ersetzt werden.

iv)
1. Zeile 14 (Zeile 11 lassen wir auch gelten).
2. In Zeile 14 wird eine Zeigervariable dereferenziert, der in Zeile 11 eine ungültige Adresse (42) zugewiesen wurde. Das führt zum *Bus error*.
3. In Zeile 11 muss eine richtige Adresse besorgt werden. Statt `c= (A*)42;` sollte man `c= new A;` verwenden.

b)

i)
```
bool a[100];

int b;
int *c;
```

ii)
```
bool a;
char b;
struct {
  int a;
  float b;
} c;
```

c)

i)
```
// ************************************
// Kopfzeile
// ************************************

int maximum(int a[], int n)


// ************************************
// Aufruf
// ************************************
int c[100];
int max;

max= maximum(a,100);
```

ii)
```
// ************************************
// Kopfzeile
// ************************************
struct Bereich {
  int inf;
  int sup;
};

void bereich(int a[], int n, Bereich &res)


// ************************************
// Aufruf
// ************************************
int c[100];
Bereich range;

bereich(a,100,range);
```

Aufgabe 2: Records und Felder *(25 Punkte)*

a)

Variante 1

```
1   struct Wert {
2     int i;       // Zeilenindex
3     int j;       // Spaltenindex
4     double val;  // Wert
5
6   struct Duenn {
7     int m;        // Anzahl Zeilen
8     int n;        // Anzahl Spalten
9     int k;        // Anzahl Nicht-Nullen
10    Wert val[M];  // M Werte
11  }
```

Variante 2

```
1   struct Duenn {
2     int m;        // Anzahl Zeilen
3     int n;        // Anzahl Spalten
4     int k;        // Anzahl Nicht-Nullen
5     int i[M];     // Zeilenindices
6     int j[M];     // Spaltenindices
7     Wert val[M];  // M Werte
8   }
```

b)

```
1   void Matrix2Duenn(Matrix matrix1, Duenn &matrix2)
2   {
3     int i;                                  // Schleifenvariable
4     int j;                                  // Schleifenvariable
5     double val;                             // Zwischengespeicherter Wert
6
7     matrix2.k= 0;                           // Anzahl Nicht-Nullen ist 0
8     for (i= 0; i<matrix1.m; i++) {          // Fuer alle Zeilen
9       for (j= 0; j<matrix1.n; j++) {        // Fuer alle Spalten
10        val= matrix1.val[i][j];             // Merk dir den Wert
11        if (val != 0) {                     // Falls der Wert ungleich 0 ist
12          matrix2.val[matrix2.k].i= i;      // Merke Dir Zeilenindex
13          matrix2.val[matrix2.k].j= j;      // Merke Dir Spaltenindex
14          matrix2.val[matrix2.k].val= val;  // Merke Dir den Wert
15          matrix2.k++;                      // Das ist dann eine Nicht-Null mehr
16        }
17      }
18    }
19  }
```

Aufgabe 3: Logik *(29 Punkte)*

a)

A	B	C	$A \oplus (B \oplus C)$
0	0	0	***0***
0	0	1	***1***
0	1	0	***1***
0	1	1	***0***
1	0	0	***1***
1	0	1	***0***
1	1	0	***0***
1	1	1	***1***

A	B	C	$(A \oplus B) \oplus C$
0	0	0	***0***
0	0	1	***1***
0	1	0	***1***
0	1	1	***0***
1	0	0	***1***
1	0	1	***0***
1	1	0	***0***
1	1	1	***1***

b)

A	B	C	C	$A \oplus B \oplus C \oplus D$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

c)

Allgemeine Regel Ist die Anzahl der wahren Operanden in einer XOR-Verknüpfung *ungerade*, so ist das Ergebnis *wahr*. Wenn hingegen die Anzahl der wahren Operanden *gerade* ist, ist das Ergebnis *falsch*.

Überlegungsbeschreibung Diese Regel geht hervor sowohl aus der Definition für zwei Operanden (Zeilen 2 und 3) als auch aus den Wahrheitstabellen für drei Operanden (Zeilen 1, 2, 4 und 8) als auch aus den Wahrheitstabellen für vier Operanden (Zeilen 2, 3, 5, 8, 9, 12, 14 und 15).

Die Regel lässt sich mit vollständiger Induktion recht einfach beweisen: (1) Als Basis nehmen wir die Wahrheitstabelle für zwei Operanden. (2) Wir nehmen an, die Behauptung gelte für n Operanden. Das Ergebnis dafür sei R. Es bedeutet also $R = 0$: Anzahl wahrer Operanden in R ist gerade, $R = 1$: Anzahl wahrer Operanden in R ist ungerade. Sei A ein neuer Operand. Dann können wir also den Ausdruck $R \oplus A$ nach der definierenden Wahrheitstabelle wir folgt berechnen:

R	A	$R \oplus A$	
0	0	0	Anzahl wahrer Operanden bleibt gleich: gerade
0	1	1	Anzahl wahrer Operanden erhöht sich um 1: ungerade
1	0	1	Anzahl wahrer Operanden bleibt gleich: ungerade
1	1	0	Anzahl wahrer Operanden erhöht sich um 1: gerade

d)

Variante 1

```
1   bool XOR1(bool X[], int n)
2   {
3     int i;                // Schleifenvariable
4     int count;            // Anzahl wahrer Operanden
5
6     count= 0;             // Initialiserung der Anzahl
7     for (i= 0; i<n; i++)  // Schleife ueber Operanden
8       if (X[i])           // Wenn Operand wahr,
9         count++;          // erhoeht sich die Anzahl
10
11    return(count%2!=0);   // Ergebnis: Anzahl ist ungerade
12  }
13
```

Variante 2

```
1   bool XOR2(bool X[], int n)
2   {
3     int i;                  // Schleifenvariable
4     bool Y;                 // Ergebnis
5
6     Y= X[0];                // Initialiserung Ergebnis
7     for (i= 1; i<n; i++)    // Schleife ueber Operanden
8       Y= (Y && !X[i]) ||    // Y= Y XOR X[i]
9         (!Y && X[i]);
10
11    return(Y);              // Rueckgabe des Ergebnisses
12  }
```

Aufgabe 4: Rekursion *(25 Punkte)*

a)

$$c_2 = c_0c_1 + c_1c_0 = 1 \cdot 1 + 1 \cdot 1 = 2$$

$$c_3 = c_0c_2 + c_1c_1 + c_2c_0 = 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 5$$

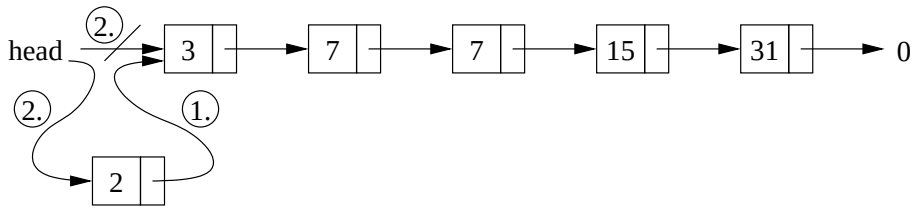
$$c_4 = c_0c_3 + c_1c_2 + c_2c_1 + c_3c_2 = 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 1 + 5 \cdot 1 = 14$$

b)
```
1   int catalanr(int n)
2   {
3     int s;                                  // Die zu berechnende Summe
4     int m;                                  // mittleres Paar
5     int i;                                  // Schleifenvariable
6
7     if (n==0) {                             // Falls n=0 ist (Rekursionsbasis)
8       return(1);                            // wissen wir den Wert
9     } else {                                // sonst (Rekursionsschritt)
10      s= 0;                                 // Initialisierung der Summe
11      i= 0;                                 // Wir beginnen mit i=0
12      while (i<(n-i-1)) {                   // Solange die Mitte nicht erreicht ist
13        s= s+2*catalanr(i)*catalanr(n-i-1); // Zaehle zweimal jedes Paar hinzu
14        i++;                                // naechstes Paar
15      }
16      if (i==(n-i-1)) {                     // Falls es eine Mitte gibt
17        m= catalanr(i);                     // Berechne das mittlere Element
18        s= s+m*m;                           // Zaehle das mittlere Paar hinzu
19      }
20      return(s);                            // Gib die Summe zurueck
21    }
22  }
```

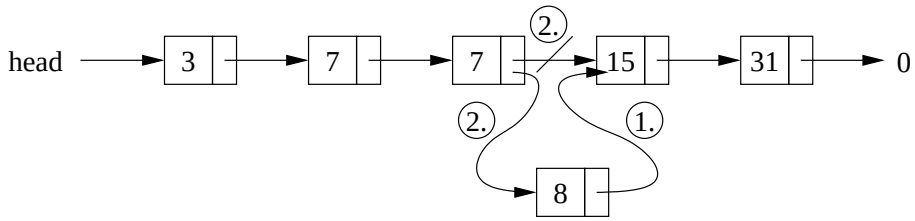
c)
```
1   int catalani(int n)
2   {
3     int c[100];                   // Array zum Speichern der Catalan-Zahlen
4     int i,j;                      // Schleifenvariablen
5
6     c[0]= 1;                      // c[0] wissen wir
7     for (i= 1; i<=n; i++) {       // Fuer alle Katalanzahlen von 1 bis n
8       c[i]= 0;                    // Initialisierung der Summe
9       for (j= 0; j<i; j++) {      // Summenschleife
10        c[i]= c[i]+c[j]*c[i-j-1]; // Zaehle zur Summe das aktuelle Paar hinzu
11      }
12    }
13    return(c[n]);                 // Das Resultat ist c[n]
14  }
```

Aufgabe 5: Dynamische Datenstrukturen: Sortierte Listen *(32 Punkte)*

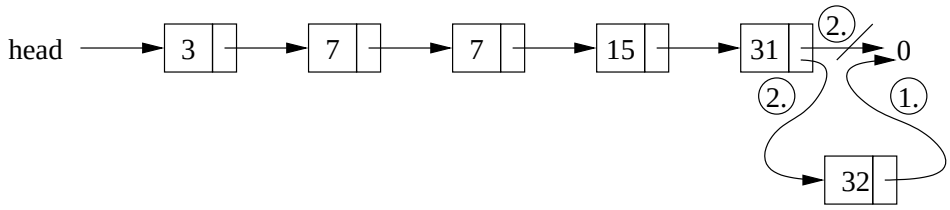
a) i)



ii)



iii)



b)
```
1   void insert(ListElem *&head, int key)
2   {
3     ListElem *focus;                          // Ein Element der Liste
4     ListElem *pre;                            // sein Vorgaenger
5     ListElem *n;                              // das neue Element
6
7     n= new ListElem;                          // das machen wir mal gerade
8     n->key= key;                              // und geben ihm den Schluessel
9     focus= head; pre= 0;                      // Dann starten wir ganz vorne
10    while ((focus!=0) && (key > focus->key)) {  // Solange wir nicht da sind
11      pre= focus;                             // der alte Focus wird zum Vorg.
12      focus= focus->next;                     // den Focus setzen wir neu
13    }
14    n->next= focus;                           // vor dem Focus muss es rein
15    if (pre==0) {                             // falls ganz vorne o. liste leer
16      head= n;                                // setzen wir den Kopf der Liste
17    } else {
18      pre->next= n;                           // sonsten denn Nachfolger des
19    }                                         // Vorgaengers
20  }
```

Rekursive Variante

```
1   void insert(ListElem *&head, int key)
2   {
3     ListElem *n;                    // das neue Element
4
5     if (head == 0) {                // leere Liste (resp. Listenende)
6       head= new ListElem;           // Der Kopf der Liste wird erzeugt
7       head->key= key;               // er bekommt den Schluessel
8       head->next= 0;                // es ist das einzige Element
9     } else if (key <= head->key) {  // Listenanfang
10      n= new ListElem;              // neues Element erzeugen
11      n->key= key;                  // Schluessel zuweisen
12      n->next= head;                // Nachfolger ist der Kopf
13      head= n;                      // das neue Element ist der Kopf
14    } else {                        // key > head->key:
15      insert2(head->next, key);     // weitersuchen
16    }
17  }
```

c)
```
1   ListElem *remove(ListElem *&head, int key)
2   {
3     ListElem *focus;                          // Ein Element der Liste
4     ListElem *pre;                            // Vorgaenger
5
6     focus= head;  pre= 0;                     // Wir fangen vorne an
7     while ((focus!=0) && (key > focus->key)) { // Solange wir noch nicht da sind
8       pre= focus;                             // Vorgaenger wird alter Focus
9       focus= focus->next;                     // Focus wird weitergesetzt
10    }
11
12    if ((focus!=0) && (key == focus->key)) {  // Wenn gefunden
13      if (focus==head) {                      // Vorne
14        head= head->next;                     // wird der Kopf verschoben
15      } else {                                // in der Mitte oder hinten
16        pre->next= focus->next;               // wird der Vorgaenger umgebogen
17      }
18    } else {                                  // Wenn nicht gefunden
19      focus= 0;                               // Out of focus ;-)
20    }
21    return(focus);                            // Zurueckgeben
22  }
```

Rekursive Variante

```
1   ListElem *remove(ListElem *&head, int key)
2   {
3     ListElem *p;                          // Zu entfernendes Element
4
5     if (head==0) {                        // wenn die Liste leer ist
6       return(0);                          // gibts nichts zu entfernen
7     } else if (key==head->key) {          // wenn das Element gefunden wurde
8       p= head;                            // entfernen wir den Kopf
9       head= head->next;                   // dann gibt es einen neuen Kopf
10      return(p);                          // Resultat ist das gefundene Element
11    } else if (key > head->key) {         // wenn wir weiter suchen muessen
12      return(remove2(head->next, key));   // suchen wir weiter
13    } else {                              // wenn wirs nicht gefunden haben
14      return(0);                          // geben wir das zurueck
15    }
16  }
```

lj, Februar 2001