Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dr. K. Simon

Studiengang Maschinenbau und Verfahrenstechnik

1. Vordiplom, Informatik I

Musterlösung

Frühling 2002

Freitag, 8. März 2002

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

a)

i)
1. Zeile 9.
2. Hinter `b--` fehlt ein Semikolon.
3. In Zeile 9: `b--;`

ii)
1. Zeile 9.
2. Hinter `c[b-1` fehlt eine schliessende eckige Klammer.
3. In Zeile 9: `c[b-1]`.

iii)
1. Zeile 12.
2. Mit `b--` wird der Pointer herabgesetzt. Wir wollwn aber den Inhalt der Pointervariablen herabsetzen.
3. In Zeile 12: `(*b)--;`.
   Wir lassen auch `*b--;` gelten, weil die Studis wahrscheinlich die stärkere Bindung von `--` gegenüber `*` nicht kennen.

iv)
1. Zeile 10.
2. In Zeile 10 wird die falsche Variable (`a`) dekrementiert.
3. Zeile 10: `b--;`

b) i)
```
bool *a;
bool b;
bool c[100];
```

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

c) i)
```
/////////////
// Kopfzeile
/////////////
void skalarprodukt (
                    double vec1[],
                    double vec2[],
                    int n,
                    double &erg)


//////////////
// Aufruf
//////////////
double v1[10], v2[10], erg;
skalarprodukt(v1,v2,erg,10);
```

ii)
```
/////////////
// Kopfzeile
/////////////
struct Complex {
  double Re, Im;
};

double arg(Complex zahl)


//////////////
// Aufruf
//////////////
Complex zahl;
double winkel;

winkel= arg(zahl);
```

Aufgabe 2: Logik *(26 Punkte)*

a) i) $!A = !(A \;\&\&\; A) = A \barwedge A$

ii) $A \;\&\&\; B = !(A \barwedge B) = (A \barwedge B) \barwedge (A \barwedge B)$

iii) $A \;||\; B = !!(A \;||\; B) = !(!A \;\&\&\; !B) = (A \barwedge A) \barwedge (B \barwedge B)$

b) i)

A	B	C	$(A \;\&\&\; B \;\&\&\; !C) \;||\; (A \;\&\&\; !B \;\&\&\; C) \;||\; (A \;\&\&\; !B \;\&\&\; !C)$
0	0	0	***0***
0	0	1	***0***
0	1	0	***0***
0	1	1	***0***
1	0	0	***1***
1	0	1	***1***
1	1	0	***1***
1	1	1	***0***

ii) $A \;\&\&\; !(B \;\&\&\; C)$

iii) $(A \barwedge (B \barwedge C)) \barwedge (A \barwedge (B \barwedge C))$

Aufgabe 3: Funktionen und Prozeduren *(27 Punkte)*

a)
```
1   int TestTripel(int a, int b)
2   {
3     double c;             // Gesuchte Zahl c
4     c = sqrt(a*a + b*b);  // Berechne c als Wurzel aus a^2+b^2
5     if(c == int(c))       // Wenn c eine ganze Zahl ist
6       return int(c);      // Gib den ganzzahligen Wert von c zurueck
7     else                  // Sonst
8       return 0;           // Gib 0 zurueck
9   }
```

b) **Variante 1**

```
1   struct Tripel {  // Ein pythagoraeisches Tripel
2     int a,b,c;     // die drei Ganzzahlen des Tripels
3   }
```

Variante 2

```
1   int tripel[3];  // ein Array als Pythagoraeisches Tripel
```

c) **Variante 1**

```
1   void PythTripel(int n, Tripel p[])
2   {
3     int a, b, c;                 // Tripel (a, b, c)
4     int count;                   // Anzahl der pyth. Tripel
5     count = 0;                   // Initialisierung der Anzahl
6     for(a = 1; a < n; a++)       // Fuer alle a zwischen 1 und n
7     {
8       for(b = a+1; b <= n; b++)  // Fuer alle b von a+1 bis n
9       {
10        c = TestTripel(a, b);    // Berechnung von c (Aufgabe a)
11        if(c != 0)               // Falls c von 0 verschieden ist
12        {
13          p[count].a = a;        // Merke Dir den Wert von a
14          p[count].b = b;        // Merke Dir den Wert von b
15          p[count].c = c;        // Merke Dir den Wert von c
16          count++;               // Das ist dann ein Tripel mehr
17        }
18      }
19    }
20  }
```

Variante 2

```
1   void PythTripel(int n, int p[][3])
2   {
3     int a, b, c;                 // Tripel (a, b, c)
4     int count;                   // Anzahl der pyth. Tripel
5     count = 0;                   // Initialisierung der Anzahl
6     for(a = 1; a < n; a++)       // Fuer alle a zwischen 1 und n
7     {
8       for(b = a+1; b <= n; b++)  // Fuer alle b von a+1 bis n
9       {
10        c = TestTripel(a, b);    // Berechnung von c (Aufgabe a)
11        if(c != 0)               // Falls c von 0 verschieden ist
12        {
13          p[count][0] = a;       // Merke Dir den Wert von a
14          p[count][1] = b;       // Merke Dir den Wert von b
15          p[count][2] = c;       // Merke Dir den Wert von c
16          count++;               // Das ist dann ein Tripel mehr
17        }
18      }
19    }
20  }
```

Aufgabe 4: Rekursion *(30 Punkte)*

a) **Vorgehen** Wir suchen alle Zerlegungen, die mit 1, 2, 3 und 4 beginnen. *Rekursion:* Um alle Zerlegungen zu bestimmen, die mit einer bestimmten Zahl beginnen, ziehen wir diese von der 4 ab und erhalten damit ein verkleinertes Zerlegungsproblem der neuen Zahl, das wir auf die gleiche Art lösen. *Abbruchbedingung:* Ist die zu zerlegende Zahl 0, so sind wir mit der aktuellen Zerlegung fertig.

Variante *Rekursion:* Wir bekommen alle Zerlegungen einer Zahl, wenn wir alle Zerlegungen der kleineren Zahl nehmen. Einmal addieren wir bei jeder Zerlegung einen neuen Summanden 1,einmal erhöhen wir den letzten Summanden um 1.*Abbruchbedingung:* Die Zerlegung der Zahl 1 ist 1.

Die Zerlegungen

1+1+1+1	2+1+1	3+1	4
1+1+2			
1+2+1	2+2		
1+3			

Variante

1+1+1+1	1+1+2
2+1+1	2+2
1+2+1	1+3
3+1	4

b)
```
1   void zerlege(int zahl, int zerlegung[], int anzahl)
2   {
3     int i;                              // Zaehlvariable
4
5     if (zahl==0) {                      // Wenn nichts mehr zu zerlegen ist,
6       cout << zerlegung[0];             // geben
7       for (i= 1; i< anzahl; i++) {      // wir die
8         cout << "+" << zerlegung[i];    // bisherige
9       }                                 // Zerlegung
10      cout << endl;                     // aus.
11    } else {                                 // ANSONSTEN (Rekursionsschritt)
12      for (i= 1; i<=zahl; i++) {             // zaehlen wir den ersten Summ. hoch
13        zerlegung[anzahl]= i;                // wir merken ihn uns
14        zerlege(zahl-i,zerlegung,anzahl+1); // und zerlegen die restliche Zahl.
15      }
16    }
17  }
```

Variante

```
1   void zerlege(int zahl, int zerlegung[], int anzahl)
2   {
3     int i;                                  // Zaehlvariable
4
5     if (zahl==0) {                          // Wenn die Zahl 0 ist
6       cout << zerlegung[0];                 // Geben
7       for (i=1; i<anzahl; i++) {            // wir
8         cout << "+" << zerlegung[i];        // die
9       }                                     // Zerlegung
10      cout << endl;                         // aus
11    } else {                                // ANSONSTEN:
12      zerlegung[anzahl]= 1;                 // Neuer Summand 1 und
13      zerlege(zahl-1,zerlegung,anzahl+1); //   Zerlegungen mit kleinerer Zahl
14
15      if (anzahl>0) {                       // Wenn eine Zerlegung angefangen wurde
16        zerlegung[anzahl-1]++;              // erhoehen wir den letzten Summanden
17        zerlege(zahl-1,zerlegung,anzahl); // und zerlegen die kleinere Zahl
18      }
19    }
20  }
```

c)
```
1   void zerlege(int zahl, int zerlegung[], int anzahl)
2   {
3     int i;                            // Zaehlvariable
4     int start;                        // Zaehlbeginn
5
6     start= 1;                         // Wir beginnen bei 1 zu zaehlen.
7     if (anzahl>1)                     // Wenn die Zerlegung Werte enthaelt,
8       start= zerlegung[anzahl-1];     // beginnen wir mit dem letzten Wert.
9
10    if (zahl==0) {                    // Wenn nichts mehr zu zerlegen ist,
11      cout << zerlegung[0];           // geben
12      for (i= 1; i< anzahl; i++) {    // wir die
13        cout << "+" << zerlegung[i];  // bisherige
14      }                               // Zerlegung
15      cout << endl;                   // aus.
16    } else if (start<=zahl) {         // Sonst: wenn noch gezaehlt werden kann
17      for (i= start; i<=zahl; i++) {       // zaehlen wir den ersten Summ. hoch
18        zerlegung[anzahl]= i;              // wir merken ihn uns
19        zerlege(zahl-i,zerlegung,anzahl+1); // und zerlegen die restliche Zahl.
20      }
21    }
22  }
```

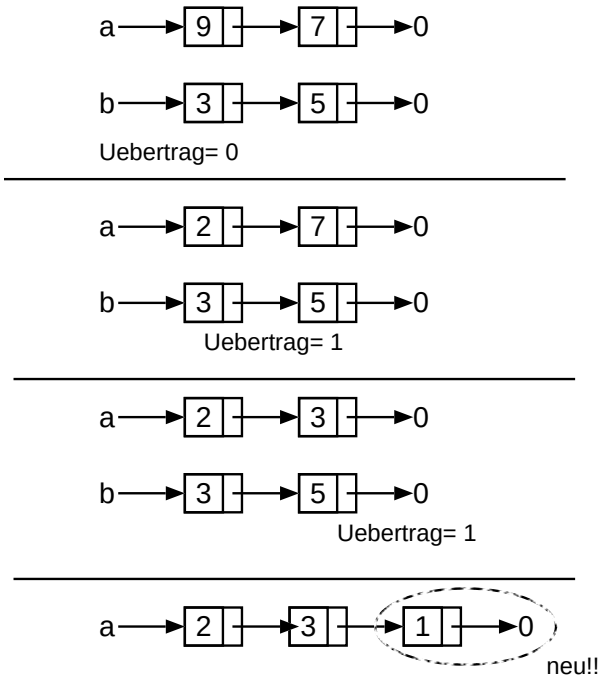
Variante

```
1   void zerlege(int zahl, int zerlegung[], int anzahl)
2   {
3     int i;                                // Zaehlvariable
4
5     if (zahl==0) {                        // Wenn die Zahl 0 ist
6       cout << zerlegung[0];               // Geben
7       for (i=1; i<anzahl; i++) {          // wir
8         cout << "+" << zerlegung[i];      // die
9       }                                   // Zerlegung
10      cout << endl;                       // aus
11    } else {                              // ANSONSTEN:
12      zerlegung[anzahl]= 1;               // Neuer Summand 1 und
13      zerlege(zahl-1,zerlegung,anzahl+1); //   Zerlegungen mit kleinerer Zahl
14
15      if (anzahl==1 ||                    // Wenn erst ein Summand vorhanden ist
16          (anzahl>1 &&                    // oder die vorhergehenden Summanden
17           zerlegung[anzahl-2]>zerlegung[anzahl-1])) { // streng absteigend sind
18        zerlegung[anzahl-1]++;            // erhoehen wir den letzten Summanden
19        zerlege(zahl-1,zerlegung,anzahl); // und zerlegen die kleinere Zahl
20      }
21    }
22  }
```

Aufgabe 5: Dynamische Datenstrukturen: Listen *(29 Punkte)*

a)
```
1   struct Ziffer {    // Eine Ziffer der Zahl
2     int wert;        // Der Wert einer Ziffer
3     Ziffer *next;    // die naechst vordere Ziffer
4   };
5
6   Ziffer *zahl;      // So speichern wir eine Zahl
```

b)



c)
```
1   void addition(ziffer *z1, ziffer *z2)
2   {
3     int uebertrag=0;      // Uebertrag der Addition am Anfang 0
4     int temp;             // temp. Erg. der Addition zweier Ziffern.
5     ziffer *vorg=0;       // Vorgaenger
6
7     while(z1!=0) {        // Solange es noch Ziffern gibt,
8       temp=               // Addieren wir
9         z1->wert+         // Ziffer erste Zahl
10        z2->wert+         // Ziffer zweite Zahl
11        uebertrag;        // Uebertrag
12      z1->wert=h%10;      // mod von 10 kommt in die Ziffer des Ergebnisses
13      uebertrag=h/10;     // Uebertrag ist div
14      vorg=z1;            // Vorgaenger merken
15      z1=l1->next;        // Gehen wir bei beiden Zahlen eine Ziffer weiter
16      z2=l2->next;
17    }
18    if(uebertrag>0) {             // wenn noch ein Uebertrag uebrig ist
19      vorg->next=new(ziffer);     // erzeugen neue Ziffer
20      vorg->next->next=0;         // Nachfolger ist 0
21      vorg->next->wert=uebertrag; // der Wert ist der Uebertrag
22    }
23  }
```

lj, März 2002