Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. K. Simon

1. Vordiplom Abt. IIIA
Informatik I
Musterlösung

Herbst 1999

Freitag 24. September 1999

Aufgabe 1: C++-Syntax, Fehlererkennung, Schleifen, C++-Semantik, Typen *(26 Punkte)*

a) i) B: Zuweisung und kein Vergleich, aber Zuweisungen an Konstanten sind nicht möglich.

ii) A: $a = 6$

iii) D: Schleife wird $a = 1.5 < 2.0$ und bleibt im weiteren Verlauf immer kleiner als 2.0.

iv) A: $a = 3$, a wird nicht verändert (die *while*-Schleife wird übrigens nie durchlaufen)

b) i)
```
a= a%b+((b-a%b)/b)*b;
```
oder
```
a= a%b+(a%b==0)*b;
```

ii)
```
a= a*b;
```

iii)
```
a= a && b;
```
Hacks:
```
a= (a+b)/2;
a= a*b;
```

c) i)
```
int a;  oder double a;
int b;
int c;
```

ii)
```
bool a;   \\ resp. int a;
double b; \\ resp. float b oder int b;
char c;
```

iii)
```
char a[100];
        \\ Dim. beliebig
int b;
int c;
```

iv)
```
struct {
  bool b[100]; \\ resp. int b[100];
               \\ Dim. beliebig
  int c;
} *a;
bool c[100]; \\ resp. int, Dim. beliebig
int b;
```

Aufgabe 2: Logik *(16 Punkte)*

a)

$(a > 2)$	$(b < 1)$	$(c == -1)$	D	In welcher Zeile wird der Wert von `D` bestimmt?
0	0	0	0	14
0	0	1	1	14
0	1	0	0	12
0	1	1	0	12
1	0	0	1	9
1	0	1	0	7
1	1	0	1	9
1	1	1	0	7

b)

$$\begin{aligned} A &:= (a > 2) \\ B &:= (b < 1) \\ C &:= (c = -1) \end{aligned}$$

$$D = (\neg A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge \neg C)$$

Vereinfachung des Ausdrucks:

$$\begin{aligned} D &= (\neg A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge B \wedge \neg C) && (1) \\ &= (\neg A \wedge \neg B \wedge C) \vee ((A \wedge \neg C) \wedge (\neg B \vee B)) && (2) \\ &= (\neg A \wedge \neg B \wedge C) \vee (A \wedge \neg C) && (3) \end{aligned}$$

Das neue Programm lautet somit:

```
int a,b,c;
bool D;

[...]

D= ((a<=2) && (b>=1) && (c==-1)) || ((a>2) && (c!=-1));

[...]
```

Aufgabe 3: Arrays, Records, Prozeduren, Funktionen *(24 Punkte)*

a)

```
1   void ZeilenSumme(Matrix A, Vector &s)
2   {
3     int i,j;                          // Laufvariablen
4
5     s.n= A.m;                         // die Laenge von s ist bekannt.
6     for (i= 0; i<A.m; i++) {          // Fuer alle Zeilen von A
7       s.a[i]= 0;                      // setze das entspr. Element von s auf 0
8       for (j= 0; j<A.n; j++)          // Fuer alle Spalten von A
9         s.a[i]= s.a[i]+fabs(A.a[i,j];) // Addier A[i,j] zum entspr. El. von s.
10    }
11  }
```

b)

```
1   bool diagonaldominant(Matrix A)
2   {
3     Vector s;                         // Zeilensummenvektor
4     int i;                            // Laufvariable
5
6     Zeilensumme(A,s);                 // Berechnung der Zeilensummen von A.
7
8     i= 0;                             // Beginne mit der ersten Zeile,
9     while ((i<A.m) &&                 // Solange noch Zeilen uebrig sind
10           (s.a[i]<2*fabs(A.a[i][i]))) // und solange die Diagonaldominanz
11                                     // erfuellt ist,
12      i++;                           // gehe zu naechsten Zeile.
13
14    return(i==A.m);                   // Wenn die Matrix diagonaldominant ist,
15                                     // haben wir alle Zeilen abgearbeitet.
16  }
```

Aufgabe 4: Maximale Teilfolge *(16 Punkte)*

```
1   void maxteilfolge(const int A[], int N, int &i, int &j, int &s)
2   {
3     int ii, jj;                   // Laufvariablen
4     int ss;                       // Zwischensumme
5
6     i= 0; j= -1; s= 0;            // Initialisierung.
7     for (ii= 0; ii<N; ii++) {     // Fuer jeden moeglichen Anfang einer Teilfolge
8       ss= 0;                      // ist die Zwischensumme 0.
9       for (jj= ii; jj<N; jj++) {  // Fuer jede moegliche Endposition,
10        ss= ss+A[jj];             // zaehle das neue Element hinzu.
11        if (ss>s) {               // Wenn die Zwischensumme groesser ist
12                                  // als das bisherige Maximum,
13          i= ii;                  // ist es das
14          j= jj;                  // neue
15          s= ss;                  // Maximum
16      }
17    }
18  }
```

Aufgabe 5: Rekursive Umwandlung von Zahlen *(16 Punkte)*

```
1   char digits[]="0123456789ABCDEF";
2
3   void Convert(int n, int b)
4   {
5     if (n<b)               // Wenn n<b (Rekursionsbasis)
6       cout << digits[n];   // Gib die dem n zugehoerige Ziffer aus.
7     else {                 // ansonsten (Rekursionsvorschrift)
8       Convert(n/b,b);      // Konvertiere zuerst die Zahl vor der jetzigen Ziffer
9       cout << digits[n%b]; // gib die jetzige Ziffer (n mod b) aus.
10    }
11  }
12
```

Aufgabe 6: Dynamische Datenstrukturen: Bäume *(20 Punkte)*

a)

```
1   int Gewicht(TreeNode *r)
2   {
3     if (r == 0)                  // Wenn der Knoten nicht existiert,
4       return(0);                 // hat er kein Gewicht.
5     else {                       // ansonsten:
6       return(Gewicht(r->left)+   // hat der linke Teilbaum ein Gewicht,
7              1+                  // der Knoten selbst wiegt 1,
8              Gewicht(r->right)); // der rechte Teilbaum hat auch ein Gewicht.
9     }
10  }
```

b)

```
1   int Gewicht2a(TreeNode *r)
2   {
3     if (r==0)
4       return(0);
5     else {
6       quotient= Gewicht(r->left)/Gewicht(r);
7
8       if ((1/3<=quotient) && (quotient<=2/3))
9         cout << r->key;
10
11      return(Gewicht2a(r->left)+1+Gewicht2a(r->right));
12    }
13  }
14
```

oder

```
1   int Gewicht2(TreeNode *r)
2   {
3     int links, rechts, eigen;     // Zwischenspeicher fuer Gewichte
4     double quotient;              // Die Variable, um den Quotienten aufzunehmen
5
6     if (r == 0)                   // Wenn der Teilbaum nicht existiert,
7       return(0);                  // gibt es nichts zu machen.
8     else {                        // ansonsten:
9       links= Gewicht(r->left);    // der linke Teilbaum hat ein Gewicht
10      rechts= Gewicht(r->right);  // der rechte Teilbaum hat ein Gewicht
11      eigen= links+1+recht;       // das Eigengewicht ist die Summe + 1.
12
13      quotient= links/eigen;      // So berechnet man den Quotienten.
14      if ((1/3<=quotient) ||      // An diesen ist die Bedingung geknuepft
15          (quotient<=2/3))        // - wenn die wahr ist,
16        cout << r->key;           // geben wir den Schluessel des Knotens aus.
17
18      return(gewicht);            // und das Gewicht geben wir immer noch
19                                  // zurueck.
20    }
21  }
```

lj, September 1999