Übungen zu Allgemeine Informatik I WS 1998/99

Andreas Franz Borchert
Universität Ulm

13. Januar 1999

Aufgabe 10: Santa Claus

Aufgabenstellung:

- Einzulesen ist eine quadratische Matrix, die als Wegematrix zu interpretieren ist, d.h. der Abstand der Orte `i` und `j` wird durch `m[i,j]` gegeben.

- Anschließend soll die Route von Santa Claus ermittelt und deren Länge berechnet werden. Die Reise beginnt am 1. Ort, wählt dann immer den nächstgelegenen Ort (bei mehreren Orten mit gleicher Entfernung ist der zuerst gegebene zu wählen) und kehrt am Ende wieder zurück.

Aufrufbeispiele

```
turing$ echo 0 4 3 5 0 1 6 2 0 | santa
    1 -->    3 [   3]
    3 -->    2 [   2]
    2 -->    1 [   5]
Insgesamt zurueckgelegt: 10
turing$ echo 0 4 3 7 2 0 8 2 9 2 0 7 1 4 9 0 | santa
    1 -->    3 [   3]
    3 -->    2 [   2]
    2 -->    4 [   2]
    4 -->    1 [   1]
Insgesamt zurueckgelegt: 8
```

- Die Größe der quadratischen Matrix wird von der Anzahl der eingegebenen Zahlen abgeleitet.
- Pro Zeile wird ein Weg mit Ausgangspunkt, Endpunkt und (in eckigen Klammern) zurückgelegter Wegstrecke ausgegeben.
- Zum Schluß wird die Gesamtstrecke angegeben.

Fehlersituationen

```
turing$ echo | santa
Keine Wegematrix vorgefunden!
turing$ echo 0 | santa
    1 -->    1 [   0]
Insgesamt zurueckgelegt: 0
turing$ echo 0 1 2 | santa
Die Wegematrix ist nicht quadratisch!
turing$ echo 0 1 2 3 | santa
Das 2. Element der Spur ist nicht 0!
turing$ creatematrix 33 | santa
Die Wegematrix ist zu gross!
turing$
```

- Jedes Programm (auch eine Lösung zu einer Übungsaufgabe) sollte – soweit mit vertretbaren Mitteln möglich – die Eingabedaten auf Plausibilität und/oder Korrektheit überprüfen.

- Auch Sonderfälle wie z.B. gar keine Eingabe sollten überlebt werden.

- Wenn es irgendwelche Beschränkungen gibt (z.B. die Maximalgröße irgendeiner Tabelle), dann sollte das Überschreiten zu sinnvollen Fehlermeldungen führen.

- `creatematrix` generiert eine zufällige Wegematrix in der gewünschten Größe (Quelltext folgt später).

Deklarationen

SantaClaus.m2

```
CONST
   ndigits = 4;
      (* es wird davon ausgegangen, dass Ortsnummern und
         Distanzen nicht mehr Ziffern benoetigen,
         wenn doch, dann wuerde die Schoenheit der Ausgabe
         darunter leiden
      *)
   maxlocations = 32;
      (* soviele Orte werden maximal unterstuetzt *)
TYPE
   Location = [0..maxlocations-1];
   LocationIndex = [0..maxlocations];
   Distance = CARDINAL;
   DistanceMatrix = ARRAY Location, Location OF Distance;

VAR
   distances: DistanceMatrix;
   noflocations: LocationIndex;
```

- Die wesentlichen Parameter sollten nach Möglichkeit zu Beginn als Konstanten definiert werden. Wichtig ist hier insbesondere `maxlocations`, das die Zahl der maximal unterstützten Orte angibt.

- Bei ganzzahligen Datentypen empfiehlt es sich immer – soweit möglich – genaue Bereiche anzugeben. Dann führen Programmierfehler eher zu sehr präzisen Fehlermeldungen (entweder durch den Compiler oder zur Laufzeit).

- In `distances` wird die Wegematrix abgelegt, `noflocations` gibt die Größe der Matrix an, die der Zahl der zu besuchenden Orte entspricht.

Rahmen

SantaClaus.m2

```
MODULE SantaClaus;

   FROM InOut IMPORT Write, WriteString, WriteCard, WriteLn,
      Done, ReadCard;

   (* ... Deklarationen (siehe vorherige Folie) ... *)

   PROCEDURE ReadMatrix(VAR matrix: DistanceMatrix;
                        VAR n: LocationIndex) : BOOLEAN;
      (* Einlesen einer quadratischen Matrix, deren Groesse
         zu Beginn noch nicht bekannt ist. Wir lesen solange
         Zahlen ein, soweit ReadCard klappt (Done = TRUE)
         und die Kapazitaet ausreicht.  In n wird die
         Groesse der Matrix abgelegt.  Bei Misserfolg wird
         eine Fehlermeldung ausgeben und FALSE
         zurueckgeliefert.
      *)
      (* ... folgt ... *)
   END ReadMatrix;

   PROCEDURE ComputeRoute(distances: DistanceMatrix;
                          noflocations: LocationIndex);
      (* ... folgt ... *)
   END ComputeRoute;

BEGIN
   IF ReadMatrix(distances, noflocations) THEN
      ComputeRoute(distances, noflocations);
   END;
END SantaClaus.
```

- Die Aufgabenstellung verteilt sich auf zwei Teilaufgaben: Das Einlesen der Matrix (einschließlich der Überprüfungen) und dem Berechnen der Route.

Einlesen der Matrix

SantaClaus.m2

```
CONST
   maxvalues = maxlocations * maxlocations;
      (* soviele Eintraege gibt es maximal in der Matrix *)
VAR
   (* Variablen zur Ablage der eingelesenen Werte *)
   buf: ARRAY [0..maxvalues-1] OF Distance;
   index: [0..maxvalues];
   value: Distance;
   i, j: Location; (* Indizes fuer matrix *)
   (* Variablen zur Bestimmung der Groesse der Matrix *)
   squarelen: LocationIndex;
   nextodd, counter: CARDINAL;
```

- Da die Größe der Matrix erst dann bekannt ist, wenn alle Werte eingelesen sind, werden sie zuerst in einem eindimensionalen Feld namens `buf` abgelegt, das `maxvalues` Einträge besitzt.
- Das Feld `buf` ist im Bereich von `0` bis `index-1` gefüllt. `index` entspricht somit auch der Zahl der bislang eingelesenen Werte.
- `value` ist der zuletzt eingelesene Wert.
- `i` und `j` werden später zur Indizierung der Ergebnismatrix verwendet.
- `squarelen` ist die Größe der Matrix, die von `0` beginnend laufend der Zahl der eingelesenen Werte angepaßt wird. Die Variablen `nextodd` und `counter` unterstützen dies.

Einlesen der Matrix

SantaClaus.m2

```
index := 0; squarelen := 0; counter := 0; nextodd := 1;
ReadCard(value);
WHILE Done DO
   (* Invarianten:
      buf[0 .. index-1] sind belegt, also index Werte
      squarelen^2 = index + counter
      (squarelen+1)^2 - squarelen^2 = nextodd
   *)
   IF index <= HIGH(buf) THEN
      buf[index] := value; INC(index);
      IF counter = 0 THEN
         INC(squarelen); counter := nextodd - 1;
         (* Der Abstand zwischen zwei aufeinanderfolgenden
            Quadratzahlen n^2 und (n+1)^2 ist 2n+1,
            somit ist der Abstand zwischen zwei
            aufeinanderfolgenden Abstaenden 2
         *)
         INC(nextodd, 2);
      ELSE
         DEC(counter);
      END;
   ELSE
      WriteString("Die Wegematrix ist zu gross!"); WriteLn;
      RETURN FALSE
   END;
   ReadCard(value);
END;
```

- `counter` gibt jeweils an, wieweit die nächste Quadratzahl entfernt ist.
- `nextodd` gibt die Distanz zur darauffolgenden Quadratzahl an.

Einlesen der Matrix

SantaClaus.m2

```
IF counter # 0 THEN
   (* somit folgt aus der Invariante: squarelen^2 # index *)
   WriteString("Die Wegematrix ist nicht quadratisch!");
   WriteLn;
   RETURN FALSE
END;
IF squarelen = 0 THEN
   WriteString("Keine Wegematrix vorgefunden!");
   WriteLn;
   RETURN FALSE
END;
FOR i := 0 TO squarelen-1 DO
   FOR j := 0 TO squarelen-1 DO
      matrix[i, j] := buf[i * squarelen + j];
      IF (i = j) & (matrix[i, j] # 0) THEN
         WriteString("Das "); WriteCard(i+1, 1);
         WriteString(". Element der Spur ist nicht 0!");
         WriteLn;
         RETURN FALSE
      END;
   END;
END;
n := squarelen;
RETURN TRUE
```

- Wenn alle Werte eingelesen sind, können die Werte in die Wegematrix nach Kenntnis ihrer Größe übertragen werden.

Berechnen der Route

SantaClaus.m2

```
PROCEDURE ComputeRoute(distances: DistanceMatrix;
                       noflocations: LocationIndex);
   VAR
      visited: ARRAY Location OF BOOLEAN;
      nofvisits: LocationIndex;
      location: Location;
      nextlocation: Location;
      length: Distance;
```

- `visited[location]` ist genau dann **TRUE**, wenn `location` bereits besucht worden ist.
- `nofvisits` ist die Zahl der erledigten Besuche.
- Die nächste Reise geht von `location` zu `nextlocation`.
- In `length` wird die Gesamtstrecke aufsummiert.

Berechnen der Route

SantaClaus.m2

```
BEGIN (* ComputeRoute *)
   FOR location := 0 TO noflocations-1 DO
      visited[location] := FALSE;
   END;
   length := 0; (* bislang noch nicht vorangekommen *)
   location := 0; (* Startpunkt der Reise *)
   nofvisits := 0;
   WHILE nofvisits < noflocations DO
      WriteLocation(location);
      visited[location] := TRUE;
      INC(nofvisits);
      IF nofvisits = noflocations THEN
         nextlocation := 0; (* und wieder zurueck *)
      ELSE
         nextlocation := NextLocation(location);
      END;
      INC(length, distances[location, nextlocation]);
      WriteString("-->"); WriteLocation(nextlocation);
      Write("[");
      WriteCard(distances[location, nextlocation], ndigits);
      Write("]"); WriteLn;
      location := nextlocation;
   END;
   WriteString("Insgesamt zurueckgelegt: ");
   WriteCard(length, 1); WriteLn;
END ComputeRoute;
```

- `NextLocation` ermittelt jeweils entsprechend der vorgegebenen Methode den nächsten Ort.

- `WriteLocation` gibt den jeweiligen Index um eins erhöht aus (Informatiker zählen ab 0, Anwender ab 1).

Berechnen der Route

SantaClaus.m2

```
PROCEDURE NextLocation(location: Location) : Location;
   VAR
      next: Location;
      mindistance: Distance;
      minlocation: Location;
BEGIN
   mindistance := MAX(Distance);
   FOR next := 0 TO noflocations-1 DO
      IF ~visited[next] THEN
         IF distances[location, next] < mindistance THEN
            mindistance := distances[location, next];
            minlocation := next;
         END;
      END;
   END;
   RETURN minlocation
END NextLocation;
```

- Von den noch nicht aufgesuchten Orten ist der nächstgelegene zurückzuliefern.

Generierung von Wegematrizen

CreateMatri.m2

```
MODULE CreateMatrix;

   FROM Arguments IMPORT InitArgs, FetchCard, AllArgs;
   FROM InOut IMPORT WriteCard, WriteLn;
   FROM RandomGenerator IMPORT Random;

   VAR
      n: CARDINAL;
      i, j: CARDINAL;
      distance: CARDINAL;

BEGIN
   InitArgs("squarelen");
   FetchCard(n);
   AllArgs;

   FOR i := 1 TO n DO
      FOR j := 1 TO n DO
         IF i = j THEN
            distance := 0;
         ELSE
            distance := Random(1, 9);
         END;
         WriteCard(distance, 2);
      END;
      WriteLn;
   END;
END CreateMatrix.
```

- `Random` liefert einen ganzzahligen Wert gleichverteilt aus dem gegebenen Intervall.

Aufgabe 11: Zeitdifferenz

```
turing$ timediff
Bitte Tag eingeben: 24
Bitte Monat eingeben: 12
Bitte Jahr eingeben: 1999

Heute ist der 12.1.1999.
Von heute bis zu Ihrem Wunschtag sind es nur 346 Tage.
turing$ timediff
Bitte Tag eingeben: 24
Bitte Monat eingeben: 12
Bitte Jahr eingeben: 2005

Heute ist der 12.1.1999.
Von heute bis zu Ihrem Wunschtag sind es nur 2538 Tage.
turing$
```

- Für ein einzulesendes Datum soll die Zeitdifferenz in Tagen zum heutigen Datum ausgegeben werden.

Vorgegebener Rahmen

TimeDiff.m2

```
MODULE TimeDiff;

   FROM Calendar IMPORT CurrentTime, ConvertTime,
      ConvertDate, Calendarday, Date, Day, Daytime, Month;
   FROM InOut IMPORT Write, ReadCard, WriteInt, WriteCard,
      WriteLn, WriteString;

   VAR
      date: Date;
      daytime: Daytime;
      today, future: Calendarday;
      tmp: CARDINAL;

   PROCEDURE CalcDays(today, future: Calendarday) : INTEGER;
      (* ... war auszufuellen ... *)
   END CalcDays;

BEGIN
   ConvertTime(CurrentTime(), date, daytime);
   ConvertDate(date, today);

   WriteString("Bitte Tag eingeben: ");
   ReadCard(tmp); future.day := tmp;
   WriteString("Bitte Monat eingeben: ");
   ReadCard(tmp); future.month := tmp;
   WriteString("Bitte Jahr eingeben: ");
   ReadCard(future.year); WriteLn;
   WriteString("Heute ist der ");
   WriteCard(today.day, 1); Write(".");
   WriteCard(today.month, 1); Write(".");
   WriteCard(today.year, 1); Write("."); WriteLn;
   WriteString("Von heute bis zu Ihrem Wunschtag sind es ");
   WriteInt(CalcDays(today, future), 0);
   WriteString(" Tage."); WriteLn;
END TimeDiff.
```

Fehlersituationen

```
turing$ timediff </dev/null
Bitte Tag eingeben:
runtime error in module TimeDiff at line 82:
CARDINAL value 0 out of [1..31]
Abort
turing$ timediff
Bitte Tag eingeben: 99

runtime error in module TimeDiff at line 82:
CARDINAL value 99 out of [1..31]
Abort
turing$ timediff
Bitte Tag eingeben: 31
Bitte Monat eingeben: 2
Bitte Jahr eingeben: 1999

Heute ist der 12.1.1999.
Von heute bis zu Ihrem Wunschtag sind es nur 50 Tage.
turing$
```

- Leider interessiert sich der vorgegebene Rahmen nicht für die Überprüfung der Eingabe.

Fehlersituationen

```
turing$ timediff2 </dev/null
Bitte Tag eingeben: Unerwarteter Fehler beim Einlesen!
turing$ timediff2
Bitte Tag eingeben: 99
Wert ausserhalb des Intervalls von [1..31]!
turing$ timediff2
Bitte Tag eingeben: 31
Bitte Monat eingeben: 2
Bitte Jahr eingeben: 1999

Ein solches Datum gibt es nicht!
turing$
```

- So könnte eine etwas bessere Reaktion auf Eingabefehler aussehen.

Überprüfungen

TimeDiff2.m2

```
MODULE TimeDiff2;
   FROM Calendar IMPORT CurrentTime, ConvertTime,
      ConvertDate, Calendarday, Date, Day, Daytime,
      Month, Year, DateOK;
   FROM InOut IMPORT Done, Write, ReadCard, WriteInt,
      WriteCard, WriteLn, WriteString;
   FROM SysExit IMPORT Exit;

   PROCEDURE ReadValue(min, max: CARDINAL) : CARDINAL;
      VAR input: CARDINAL;
   BEGIN ReadCard(input);
      IF ~Done THEN
         WriteString("Unerwarteter Fehler beim Einlesen!");
         WriteLn; Exit(1);
      END;
      IF (input < min) OR (input > max) THEN
         WriteString("Wert ausserhalb des Intervalls von [");
         WriteCard(min, 1); WriteString("..");
         WriteCard(max, 1); WriteString("]!"); WriteLn;
         Exit(1);
      END;
      RETURN input
   END ReadValue;

BEGIN
   WriteString("Bitte Tag eingeben: ");
   future.day := ReadValue(MIN(Day), MAX(Day));
   WriteString("Bitte Monat eingeben: ");
   future.month := ReadValue(MIN(Month), MAX(Month));
   WriteString("Bitte Jahr eingeben: ");
   future.year := ReadValue(MIN(Year), MAX(Year)); WriteLn;
   IF ~DateOK(future.year, future.month, future.day) THEN
      WriteString("Ein solches Datum gibt es nicht!");
      WriteLn; Exit(1);
   END; (* ... der Rest ist unveraendert ... *)
END TimeDiff2.
```

Berechnung der Differenz

TimeDiff.m2

```
PROCEDURE CalcDays(today, future: Calendarday) : INTEGER;
   (* liefert die Zahl der Tage zwischen today und future,
      wobei future auch in der Vergangenheit liegen darf;
      in diesem Fall wird ein negativer Wert zurueckgeliefert
   *)
   VAR
      md: ARRAY [1..12] OF Day;
      days: INTEGER;

   (* ... LeapYear und Days ... *)

BEGIN (* CalcDays *)
   md[1] := 31; md[2] := 28; md[3] := 31; md[4] := 30;
   md[5] := 31; md[6] := 30; md[7] := 31; md[8] := 31;
   md[9] := 30; md[10] := 31; md[11] := 30; md[12] := 31;

   days := (future.year - today.year) * 365;
   RETURN days + Days(future) - Days(today)
END CalcDays;
```

- Die Prozedur `Days` liefert für das jeweilige Datum die Zahl der Tage seit dem Beginn des jeweiligen Jahres **plus** die Zahl der Schalttage seit dem Jahre 0.

- Wenn zu der Differenz der beiden Werte von `Days` die Differenz der Jahre in Tagen (ohne Berücksichtigung von Schalttagen) hinzukommt, erhalten wir die gewünschte Differenz.

Berechnung der Differenz

TimeDiff.m2

```
PROCEDURE LeapYear(year: CARDINAL) : BOOLEAN;
BEGIN
   RETURN (year MOD 4 = 0) &
          ((year MOD 100 > 0) OR (year MOD 400 = 0))
END LeapYear;
```

- Jedes durch vier teilbare Jahr ist ein Schaltjahr, es sei denn es durch 100 teilbar, jedoch nicht durch 400.
- 1996 und 2000 sind Schaltjahre, 1800 und 1900 jedoch nicht.
- Die Zeit vor der Gregorianischen Kalenderreform wird nicht berücksichtigt.

Berechnung der Differenz

TimeDiff.m2

```
PROCEDURE Days(day: Calendarday) : INTEGER;
   (* liefert die Anzahl der Tage seit Beginn des
      zugehoerigen Kalendarjahres plus die Anzahl
      der Schalttage seit dem Jahr 0, wobei
      Kalenderreformen ignoriert werden
   *)
   VAR
      days: INTEGER;
      i: Month;
BEGIN
   (* Anzahl der Schaltjahre ermitteln *)
   days := day.year DIV 4 - day.year DIV 100 +
           day.year DIV 400;

   (* Anzahl der Tage im aktuellen Jahr hinzuzaehlen *)
   INC(days, day.day);
   FOR i := 1 TO day.month - 1 DO
      INC(days, md[i]);
   END;

   (* Fehlerkorrektur bei Schaltjahren notwendig, wenn wir
      im Januar oder Februar sind
   *)
   IF (day.month < 3) & LeapYear(day.year) THEN
      DEC(days);
   END;
   RETURN days
END Days;
```

Einfachere Lösung

DaysLeft.m2

```
MODULE DaysLeft;

   FROM Arguments IMPORT InitArgs, FetchString, AllArgs,
      Usage;
   FROM Calendar IMPORT Date, Daytime, CurrentTime,
      ConvertTime;
   FROM InOut IMPORT WriteString, WriteCard, WriteLn;
   FROM TimeIO IMPORT Done, SreadDate, ReleaseList, Append;

   VAR
      date: Date; (* Datum aus der Kommandozeile *)
      today: Date; (* heutiges Datum *)

   PROCEDURE InitTimeIO; (* ... folgt ... *)
   PROCEDURE ProcessArgs; (* ... folgt ... *)
   PROCEDURE CurrentDate() : Date; (* ... folgt ... *)

BEGIN
   InitTimeIO;
   ProcessArgs;
   today := CurrentDate();
   IF date >= today THEN
      WriteCard(date - today, 1); WriteLn;
   ELSE
      WriteString("Das Datum stammt aus der Vergangenheit.");
      WriteLn;
   END;
END DaysLeft.
```

- Es geht auch einfacher, wenn berücksichtigt wird, daß `Date` ein außerordentlich praktischer Datentyp ist :-)

Einfachere Lösung

DaysLeft.m2

```
PROCEDURE InitTimeIO;
BEGIN
   ReleaseList;
   Append("d.m.y"); (* 24.12.1999 *)
   Append("d.my");  (* 24. Dezember 1999 *)
   Append("d.m.");  (* 24.12. *)
   Append("d.m");   (* 24. Dezember *)
END InitTimeIO;

PROCEDURE ProcessArgs;
   PROCEDURE FetchDate(VAR date: Date);
      VAR string: ARRAY [0..31] OF CHAR;
   BEGIN
      FetchString(string);
      SreadDate(string, date);
      IF ~Done THEN
         Usage;
      END;
   END FetchDate;
BEGIN (* ProcessArgs *)
   InitArgs("date");
   FetchDate(date);
   AllArgs;
END ProcessArgs;
```

- Die Prozedur `SreadDate` aus dem Modul `TimeIO` kann ein frei formatiertes Datum aus einer Zeichenkette einlesen.

- Die unterstützen Datumsformate werden von `InitTimeIO` festgelegt.

- Wenn das Datum nicht stimmt, wird `Usage` aufgerufen.

Tips zur 12. Aufgabe: Kantige Kurven

```
turing$ fplot -h 25 -w 60 -x '-pi/2' '3*pi/2' 'sin(x)'
                          XXXXXXXX
                         XXXXXXXXXX
                       XXXXXXXXXXXXXX
                      XXXXXXXXXXXXXXXX
                     XXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXX
                    XXXXXXXXXXXXXXXXXXXX
                   XXXXXXXXXXXXXXXXXXXXXX
                  XXXXXXXXXXXXXXXXXXXXXXXX
                 XXXXXXXXXXXXXXXXXXXXXXXXXX
                XXXXXXXXXXXXXXXXXXXXXXXXXXXX
                XXXXXXXXXXXXXXXXXXXXXXXXXXXX
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
turing$
```

- Ziel ist es, eine kleine Anwendung zu entwickeln, die auf sehr einfache Weise Funktionen plottet.

Kommandozeile

```
turing$ fplot 2>&1 | fmt -60
Usage: fplot [-d xdelta] [-h hoehe] [-v varname] [-w weite]
[-x min max] function
turing$
```

-d *xdelta*	Abstand zwischen den einzelnen Punkten auf der X-Achse. Dies wird voreinstellungsgemäß in Abhängigkeit des Intervalls und der gewünschten Weite berechnet.
-h *höhe*	spezifiziert die zur Verfügung stehende Höhe. Die Voreinstellung ist bei 23.
-v *varname*	Der Variablenname, der beim Funktionsausdruck verwendet wird. Dies ist normalerweise "x".
-w *weite*	spezifiziert die zur Verfügung stehende Weite. Die Voreinstellung liegt bei 80. Diese Angabe wird hinfällig, wenn *xdelta* angegeben wird.
-x *min max*	gibt das darzustellende Intervall auf der X-Achse an. Die Voreinstellung liegt bei $[-1..1]$. Die Angabe von Ausdrücken ist dabei zulässig (siehe obiges Beispiel).

Als letztes folgt auf der Kommandozeile die darzustellende Funktion.

Das Modul *Arguments*

```
oberon$ argsdemo 2>&1 | fmt -60
Usage: argsdemo [-a card] [-b string] [-c] [-d card1 card2]
text
oberon$
```

ArgsDemo.m2

```
MODULE ArgsDemo;

   FROM Arguments IMPORT InitArgs, GetFlag, FetchCard,
      FetchString, AllArgs, Usage;

   VAR
      avalue: CARDINAL;
      bvalue: ARRAY [0..31] OF CHAR;
      cflag: BOOLEAN;
      dvalues: ARRAY [0..1] OF CARDINAL;
      text: ARRAY [0..79] OF CHAR;

   PROCEDURE ProcessArgs;
      (* ... *)
   END ProcessArgs;

BEGIN
   ProcessArgs;
END ArgsDemo.
```

Das Modul *Arguments*

ArgsDemo.m2

```
PROCEDURE ProcessArgs;
   VAR flag: CHAR;
BEGIN
   avalue := 0; bvalue := ""; cflag := FALSE;
   dvalues[0] := 0; dvalues[1] := 0;
   InitArgs("[-a card] [-b string] [-c] [-d card1 card2] text");
   WHILE GetFlag(flag) DO
      CASE flag OF
      | "a":   FetchCard(avalue);
      | "b":   FetchString(bvalue);
      | "c":   cflag := TRUE;
      | "d":   FetchCard(dvalues[0]); FetchCard(dvalues[1]);
      ELSE     Usage;
      END;
   END;
   FetchString(text); AllArgs;
END ProcessArgs;
```

- Mit `InitArgs` beginnt die Abarbeitung der Kommandozeile von vorne. Übergeben wird ein Teil der Zeile, die bei `Usage` ausgegeben wird.
- `GetFlag` holt die nächste Option, sofern vorhanden. Wenn keine Option vorliegt, wird **FALSE** zurückgeliefert.
- `FetchCard` und `FetchString` holen ein Argument ab. Wenn keines vorliegt, wird automatisch `Usage` aufgerufen.
- `Usage` gibt eine Fehlermeldung aus und beendet das Programm.
- `AllArgs` ruft `Usage` auf, wenn noch unbearbeitete Argumente übrig geblieben sind.

Das Modul *Functions*

```
oberon$ funcdemo
function: sin(x)+cos(y)
x = 1
y = 2
 4.253241482607540e-001
oberon$
```

- Mit dem Modul `Functions` ist es möglich, Ausdrücke einzulesen und zu bewerten.
- Dabei sind beliebig viele Variablen erlaubt, die einzeln gesetzt werden können (Voreinstellung ist `0.0`).
- Anschließend ist es möglich, die Funktion zu bewerten.

Das Modul *Functions*

FuncDemo.m2

```
MODULE FuncDemo;

   FROM Functions IMPORT Function, ParseFunction,
      FirstParam, NextParam, SetFuncParam, EvalFunction;
   FROM InOut IMPORT WriteString, WriteLn, ReadString;
   FROM RealInOut IMPORT ReadReal, WriteReal;
   IMPORT StdFuncs;

   VAR
      functxt, parname: ARRAY [0..63] OF CHAR;
      func: Function;
      val: REAL;

BEGIN
   WriteString("function: "); ReadString(functxt);
   IF ParseFunction(functxt, func) THEN
      FirstParam(func);
      WHILE NextParam(func, parname) DO
         WriteString(parname); WriteString(" = ");
         ReadReal(val);
         SetFuncParam(func, parname, val);
      END;
      WriteReal(EvalFunction(func), 1); WriteLn;
   ELSE
      WriteString("Syntaxfehler!"); WriteLn;
   END;
END FuncDemo.
```

Das Modul *Functions*

Die Grammatik in EBNF:

CondExpression = Expression
[“?” CondExpression “:” CondExpression] .

Expression = SimpleExpression
[RelOp SimpleExpression] .

SimpleExpression = [“+” | “-”] Term AddOp Term .

Term = Factor MulOp Factor .

Factor = Constant | IDENT | FunctionCall |
“~” Factor | “(” CondExpression “)” .

FunctionCall = IDENT “(” CondExpression
[“,” CondExpression] “)” .

RelOp = “=” | “#” | “<” | “>” | “<=” | “>=” .

AddOp = “+” | “−” | “|” .

MulOp = “*” | “/” | “&” .

Beispiele:

```
a+b
x < 0? 0: sqrt(x)
x < 0? -x: x
x = 0? 0: sin(x)/cos(x)
```

Abschließende Tips

```
turing$ fplot '2*3++x-7'
Fehler in dem Ausdruck:
2*3++x-7
    ^--- Fehlerposition
turing$
```

- Wenn `ParseFunction` **FALSE** zurückliefert, liegt ein Fehlervor. Die Fehlerposition ist über die Variable `errpos` des Moduls `Functions` zugänglich. Sie sollten dies in Ihrer Lösung ausnutzen, um eine hilfreiche Fehlermeldung zu erzeugen.

- Wenn `StdFuncs` nicht importiert wird, stehen die Standardfunktionen wie `sin(x)` oder Konstanten wie `pi` nicht zur Verfügung.

- Achten Sie darauf, daß nicht nur die Funktion mit `Functions` zu bearbeiten ist, sondern auch weitere Parameter. Nehmen Sie entsprechende Parameter mit `FetchString` entgegen und füttern Sie sie anschließend `ParseFunction` und `EvalFunction`, um den Wert zu ermitteln.

- Denken Sie daran, daß Sie mit dem Plotten erst beginnen können, nachdem Sie alle Funktionswerte kennen. Sie werden also hierfür ein Array benötigen und bei der Bewertung der Zwischenpunkte nebenbei das Minimum und das Maximum der Funktionswerte bestimmen.