4.6 Bemerkungen zur UP–Technik

• Grundidee bisher: Algorithmus → Programm

– Pr. selbst ist Objekt der Verarbeitung
– das Objekt „Pr." kommuniziert mit seiner Umgebung

• Frage: Ist dies verallgemeinerbar?

komplexer Algor. ➔ Teilalgorithmen
a) jeder wieder ein Pr.-objekt
b) Gesamtprogramm zus.-gesetzt

⇒ Forderung: 1. Teile müssen definierbar sein
2. Es muss Regeln für ihr Zus.-setzen geben

• Realisierungsstufen:

1. Stufe: einfachste → Standardfunktionen und -prozeduren
(vordef. Programmobjekte)

muss nur lernen:
a) welche gibt es
b) was tun sie
c) wie anwenden (aufrufen)

2. Stufe: selbstdefinierbare Programmobjekte
Klar: Dazu entspr. Mechanismus der PS benötigt!

Def. 4.5:
Ein Teilalgorithmus, der in einem Programm definierbar und wiederholt anwendbar ist, heißt ***Unterprogramm***.

∃ 2 Arten von UP:

(1) geschlossene UP → Routinen
(2) offene UP → Makros

gemein haben beide 2 Ebenen:

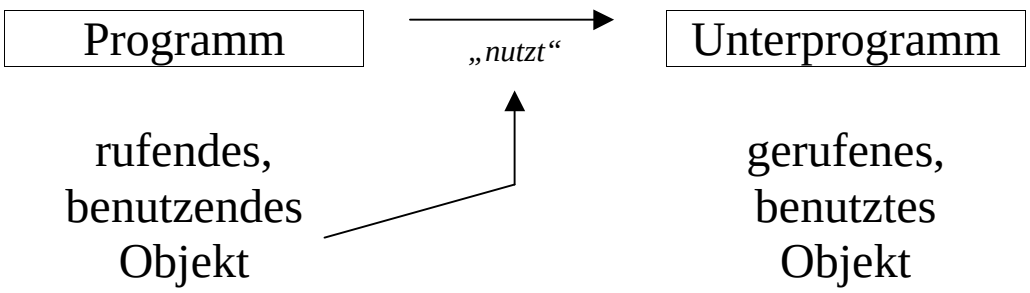
1. Ebene der Def. der UP:
   a) Teilalgor. als Zusf. von Anweisungen und DO
      (⇒ benannte Objekte)
   b) Regeln zur Kommunikation mit Umgebung
   (⇒ formale Parameter)

2. Ebene der Benutzung der UP: (hier Unterschied)
   Benutzung = Aufruf des UP (als Anweisung)

   bei (1) wird Anweis.-folge des UP zur Laufzeit vollständig abgearbeitet
   bei (2) wird Anweis.-folge des UP zur Übersetzungszeit an entspr. Stelle kopiert

   ! *Führt zur Übersetzungszeit zur Verlängerung des Pr.; Kommunikation mit Umgebung hier schon aufgelöst* !

• Nutzung der Teilalgor. erfordert klare Schnittstellenbeschreibung zw. rufendem und gerufenem Pr.:



Interface (Schnittstelle) dient zur Beschreibung der Ordnungsbeziehung in dieser Hierarchie

in ***Protokoll*** legt Schnittstelle die Art und Weise der Benutzerbeziehungen fest

⇒ *Schnittstellenbeschreibung*:
Def. des Objektes „UP“;
Nutzung des „UP“;

Routinen (in versch. PS) → Prozeduren
→ Funktionen

Bem.: Unterscheiden sich in Def. und Nutzung!

am Bsp. Pascal →

A) Prozeduren

1. Def. von Prozeduren:

PROCEDURE *prozname* (*parameterliste*);

vereinbarungsteil; {für lokale Objekte}

anweisungsteil; {Proz.-körper; enthält alg. Elemente}

Schnittstelle = *prozname* (*parameterliste*)

– Objektbezeichner; frei wählbar
– Parameterliste
  · definierte Anzahl
  · definierte Reihenfolge
  · definierte Typen
  · definierte Art des Parameteraustausches

Parameter:

· für flexible Anpassung eines UP an versch. Einsatzfälle

· Prozedur kann Eingabe- und Ausgabegrößen benötigen

· sind in Parameterliste in Art und Anzahl anzugeben

· heißen formale Parameter

· werden im HP nicht vereinbart

· formal = zur Beschreibung als formale Größen im Anweisungsteil („Platzhalter“)

in Mathem. $\rightarrow$

$f(x, a, b, c) = ax^2 + bx + c$

$\Rightarrow x, a, b, c$ – formale Parameter

$f(1, 2, 3, 4) = 2 \cdot 1^2 + 3 \cdot 1 + 4 = 9$

2. Nutzung von Prozeduren: (im rufenden Progr.)

durch (Prozedur-) ***Anweisung***

prozname (*aktuelle parameterliste*);

akt. Parameter:

- ersetzen bei Aufruf die formalen Parameter
- mit ihren akt. Werten wird die Prozedur ausgeführt
  ⇒ müssen im Vereinbarungsteil des ruf. Programms def. sein
  ⇒ müssen zur Zeit des Aufrufes einen Wert besitzen

Beziehung zw. akt. und form. Param.:

- gleiche Anzahl *{strenge Forderung}*
- übereinstimmende Reihenfolge *{wegen der Semantik der Param.}*
- übereinstimmende Typen
- Sicherung der Regeln des Parameteraustausches

Bsp.:

```
PROCEDURE  maximum (a, b : real; VAR max : real);
    BEGIN
        IF a > b THEN  max := a  ELSE max := b
    END;
```

im HP (ruf. Pr.), wo *x*, *t* zwei dort def. Variablen, wäre folg. Aufruf möglich:

maximum (19.3*2.7, sqr(*x*), *t*);

!! *x*, *t* als real–Var. vereinbart !!

3. Regeln für den Parameteraustausch:

- Parameter sind für die Kommunikation zw. rufendem und gerufenem Objekt (analog der Kommunik. zw. Pr. und Umwelt)

  rufend → gerufen: Input–Information
  gerufen → rufend: Output–Information
  dazu noch: InOut–Parameter;

- mit Proz.-aufruf ist ein Mechanismus verbunden, wie die aktuellen Werte an die Prozedur übergeben werden:

  z.B. die Prozedur *maximum* ⇒
  die Par. *a*, *b*, *max* im Kopf der Prozedur heißen formale Par.;
  die Par. im Proz.-aufruf 19.3*2.7, sqr(*x*), *t* aktuelle Par.;

- i.d.R. (auch in Pascal) 2 Möglichkeiten der ***Wertübergabe***:

(i) Wertsubstitution *(call by value)*

! Muß klar sein, welcher Wert →
– Bestimmung des Wertes des entspr. Parameters;
– Mitteilung des Wertes an die Prozedur;
! Nur bei Input–Information!
! In Pascal: Für alle Par. ohne VAR–Spezifikation!

Bsp. Prozedur *maximum* → *a*, *b* Wertparameter

– Ablauf/Mechanismus:

- Prozedur richtet „Hilfs-“Variable mit (*name*, *wert*, *typ*) als Platzhalter ein
  - dabei *name* nur der Prozedur bekannt
  - Wert bei Aufruf auf entspr. Speicherzelle, so dass er den formalen Parameter ersetzen kann

! Rufende Pr. kennt Verwendung des Wertes nicht!

! Wert kann auch aus Ausdruck erzeugt werden!

– Bez.:

· Art des Aufrufes: Wertaufruf (call by value)

· formal Param.: Wertparameter

– Bsp.: *maximum* →

wenn Wert der Var. *x* gleich 7, so bewirkt der Aufruf
maximum (19.3*2.7, sqr(*x*), *t*);
folgendes:

- · im UP wird Speicherplatz entspr. des Parametertyps geschaffen
- · Werte der akt. Par. berechnet →
  19.3 * 2.7 liefert 52.11 und sqr(*x*) liefert 49
  und den entspr. Speicherplätzen zugeordnet
- · damit wird UP ausgeführt
- · die im HP def. reelle Var. *t* erhält Output–Inf. von
  max = 52.11 zugewiesen

(ii) Variablensubstitution (call by reference)

! Alle in Liste der formal. Par. mit VAR gekennzeichneten Parameter sind Variablenparameter.

! Entspr. akt. Par. muß Variable des verlangten Typs sein.

! Parameter für Output-Inf. müssen notw. Variablenparameter sein.

Bsp. Prozedur *maximum* → *max* ist Variablenparameter

– Ablauf/Mechanismus/Wirkung:

- bei UP–Aufruf werden die Adressen(-nummern), d.h. der Speicherplatz derjenigen akt. Param., die den mit VAR gekennzeichneten formalen Param. entsprechen, an das UP übergeben {Speicherplatz für *t* wird *max* zugeordnet}
- werden im UP diese Param. verarbeitet, wird der entspr. Speicherplatz (des ruf. Pr.) (durch geruf. Pr.) referiert; und die dort befindliche Inf. verändert
  → Referenzaufruf (call by reference)
- nach Ende des UP stehen die Par.-werte für weitere Arbeit im rufenden Pr. zur Verfügung

! Rückgabe bzw. Output an ruf. Pr. nur vorgetäuscht!

(iii) Namenssubstitution (call by name)

– analog Makro–Technik:
textuelle Substitution, wenn Par. eine Fkt. ist; falls Par. eine Var., so wie Variablensubst.;
– in Pascal nicht;

4. Zusammenfassung und Bemerkungen:

- akt. Param. müssen im HP, formale Param. brauchen nicht vereinbart zu werden
- Obj., die nur Eingabegrößen für UP → Wertsubstitution
  ⇒ ohne VAR
- Obj., die nur Ausgabegrößen für UP → Variablensubstitution
  ⇒ mit VAR
- Typ–Namen für formale Param. bzw. Datentyp der akt. Param.:
  – einfache (CHAR, BYTE, BOOLEAN, INTEGER, REAL);
  – STRING (= STRING[255])
  – strukturierte (ARRAY, RECORD, STRING[*x*]) müssen global mit Typdeklaration umbenannt werden

- Par. gleichen Typs und gleicher Übergabeart durch Komma, sonst durch Semikolon getrennt
- Prozedur im Vereinbarungsteil des HP vereinbart, Aufruf im Anweisungsteil
- Prozedur wie HP in mehreren Schritten entwickeln
- READ(-LN), WRITE(-LN) sind Standard-Prozeduren

B) Funktionen

- oft eine Prozedur zu benutzen, die
  a) genau 1 Wert produziert, der
  b) innerhalb eines Ausdruckes aufzurufen ist

1. Def. von Funktionen: →Funktionsvereinbarung

FUNCTION *funktionsname* (*parameterliste*) : ***typname***;

vereinbarungsteil; *Schnittstelle* *Typbezeicher.*
anweisungsteil; → *Funktionskörper*

- Parameter: nur Input–Parameter;
- Funktion liefert Wert vom Typ *typname*;
- Beziehung zw. *funktionsname* und *typname*:
  Im Anweisungsteil muss *funktionsname* ein Wert zugewiesen werden, d. h. mindestens 1× dort
  funktionsname := *ausdruck*;
  ⇒ Interpretation des *funktionsname* wie eine Variable

{Proz.-name kommt innerhalb einer Prozedur nicht vor.}

2. Nutzung von Funktionen: →Funktionsaufruf

- rufen in einem Ausdruck durch
  funktionsname (*akt. parameterliste*);
  ! Parameterlisten können auch leer sein!

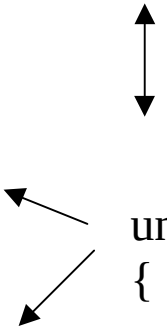
- Fkt. nur für nichtstrukturierte Datentypen, d. h. *typname* kann einfacher Typ, STRING oder Zeigertyp sein

- Zusfassung der Unterschiede zur Prozedur:
  (1) Fkt.-namen mind. 1× im Anweisungsteil ein Wert zugewiesen.
  (2) Keine strukt. Datentypen.
  (3) Fkt. innerhalb eines Ausdrucks aufrufen.

3. Funktionen mit Seiteneffekten:

```
PROGRAM  bsp;
VAR a, b : integer;
      FUNCTION     d (x : integer) : integer;
            BEGIN
                  d := a + x;
                  a := a + 1              → Seiteneffekt !
            END;
            !!! Schlechter Progr.-stil !

BEGIN {HP}
      a := 1;
      b := d (1)  +  d (a);
      WRITELN(Output, b);
      a := 1;
      b := d (a)  +  d (1);
      WRITELN(Output, b)
END.
```

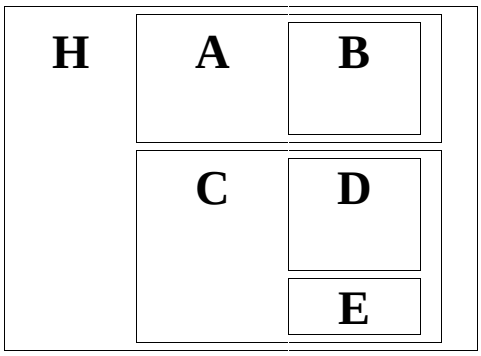


unterschiedliche Erg.
{ 6 bzw. 5 }

C) Gültigkeitsbereiche von Objekten

- prozedurale Abstraktion → hierarch. Strukturierung der Programme
  ⇒ entspr. Gültigkeitsbereich der Namen (Objekte);

- z.B.:



H : HP

A..E : UP

- Grundprinzip: Jeder vereinbarte Name gilt nur für den Block, in dem die Vereinbarung steht.

  Man sagt:
  Der Name ist für diesen Block lokal.
  Namen, die in umfass. Block def. sind, heißen global.

- allg.:
  ① Hierarchiebez. zw. den Blöcken – Über-, Unterord.-, Gleichstellung;
  ② im übergeordneten Block vereinb. Objekte gelten in allen untergeordneten; sie sind globale Obj.;
  alle Standardnamen, z. B. alle „oben“ def. Obj. sind „unten“ aufrufbar, also auch UP;
  ③ im untergeordn. Block vereinbarte Objekte sind dem übergeordneten verborgen → lokale Objekte;
  ④ Regelung im Kollisionsfall: Bei Namensgleichheit hat im UP das lokale den Vorrang;

! **Warnung**:
Verwendung globaler Größen in UP ist gefährlich

⇒ Seiteneffekte
⇒ besser Parametervermittlung zw. ruf. + gerufenem Pr.

für obiges Bsp. →

Objekt def. in	verfügbar in
H	H und A..E
A	A, B
B	B
C	C, D, E
D	D
E	E

D) Beispiele

1. Bestimmung von Nullstellen einer Funktion

geg.: $y = f(x)$; $[a, b]$; Genauigkeitsschranke $\varepsilon > 0$;
ges.: alle Nullstellen von $f(x)$ in $[a, b]$ mit Genauigkeit ε;

Lösung:

Idee →

Taste Intervall mit Schrittweite Δx ab.
Ist ein Intervall $[x, x + \Delta x]$ mit Nullstelle gefunden, so Approximation derselben bis auf ε–Genauigkeit durch Halbieren des Intervalls.

I) Pseudokode

```
BEGIN
      Anf.-werte lesen;          {a, b, ε, Δ x}
      x auf Intervallanfang setzen;
      WHILE Intervallende nicht erreicht DO
            BEGIN                {Suche}
                  einen Schritt nach rechts;
                  IF Nullstelle gefunden
                    THEN  Nullstelle drucken
                    ELSE  IF Nullstellenintervall gefunden
                                THEN  Nullstelle bestimmen
            END                  {Suche}
END.
```

II) Verfeinerung

- Anf.-werte lesen: *a*, *b*, *epsilon*, *deltax* einlesen;

- *x* auf Int.-anfang: *xl* := *a*; *yl* := *f* (*xl*); *xr* := *xl* + *deltax*;
  {1. Teilint. untersuchen} *yr* := *f* (*xr*);
  IF ABS (*yl*) < *epsilon* THEN Nullst. *xl* drucken;
  IF ABS (*yr*) < *epsilon* THEN Nullst. *xr* drucken;
  IF Nullst.int. gefunden THEN Nullst. bestimmen;

- 1 Schritt nach rechts:
  xl := *xr*; *xr* := *xr* + *deltax*;
  yl := *yr*; *yr* := *f* (*xr*);

- Nullstellenint. gefunden: *yr* * *yl* <= 0;
  {damit auch Fall erfasst, dass Nullstelle auf Int.-rand}

- Nullstelle bestimmen:

```
REPEAT
        mitte := (xl + xr) / 2;
        IF f (mitte) * yl > 0
              THEN  yl := f (mitte);   xl := mitte;
              ELSE   yr := f (mitte);   xr := mitte;
UNTIL ABS(f (mitte)) < epsilon;
Nullstelle drucken;
```

```
PROGRAM   Nullstellen;

  USES CRT;
  VAR a, b, epsilon, deltax, x1, x2, y1, y2: REAL;

  FUNCTION  f(x: REAL): REAL;
    BEGIN
     f := (x – 1.12) * (x – 2.13) * (x – 3.14)      {Nullstellen bei 1.12,
                                                      2.13 und 3.14}
    END;

  PROCEDURE anfwertelesen(VAR ug, og, eps, delta: REAL);
    BEGIN
     WRITE(´Untere Intervallgrenze gleich :  ´);
     READLN(ug);
     WRITE(´Obere Intervallgrenze gleich :  ´);
     READLN(og);
     WRITE(´Genauigkeit gleich :  ´); READLN(eps);
     WRITE(´Schrittweite gleich :  ´); READLN(delta)
    END;        {Ende anfwertelesen}
```

```
PROCEDURE nullstdrucken(x: REAL);
  BEGIN
   WRITELN(´Nullstelle bei x = ´, x:8:5, ´     y = ´, f(x):7:6)
  END;

PROCEDURE nullstbestimmen(xl, xr, yl, yr: REAL);
  VAR mitte: REAL;
  BEGIN
     REPEAT
        mitte := (xl + xr) / 2;
        IF  f(mitte) * yl > 0
          THEN  BEGIN   yl:=f(mitte); xl:=mitte  END
          ELSE  BEGIN   yr:=f(mitte); xr:=mitte  END
     UNTIL ABS(f(mitte)) < epsilon;
     Nullstdrucken(mitte)
  END;       {Nullstelle bestimmen}

PROCEDURE schrittnachrechts(VAR xl, xr, yl, yr: REAL);
  BEGIN
   xl := xr;   xr := xr + deltax;
   yl := yr;   yr := f(xr)
  END;

PROCEDURE anfsetzen(VAR xl, xr, yl, yr: REAL);
  BEGIN
   xl := a;
   yl := f(xl);
   xr := xl + deltax;
   yr := f(xr);
   IF  ABS(yl) < epsilon  THEN  nullstdrucken(xl);
   IF  ABS(yr) < epsilon  THEN  nullstdrucken(xr);
   IF yl * yr < = 0  THEN  nullstbestimmen(xl, xr, yl, yr)
  END;
```

```
BEGIN   {HP}
  CLRSCR;
  anfwertelesen(a, b, epsilon, deltax);
  anfsetzen(x1, x2, y1, y2);
  WHILE  (x1 < b)  DO
    BEGIN
     schrittnachrechts(x1, x2, y1, y2);
     IF  ABS(y2) < epsilon
       THEN  BEGIN
                          nullstdrucken(x2);
                          schrittnachrechts(x1, x2, y1, y2)
             END
       ELSE   IF  y1 * y2 <= 0
                THEN   nullstbestimmen(x1, x2, y1, y2)
    END;       {Ende suchen}
    READLN
END.
```

2. Mischen zweier Files

Geg.: 2 Files *i* und *j* mit Integer–Werten, die aufsteigend sortiert sind.

Ges.: Mischung von *i* und *j* zu einem aufsteigend sortierten File *k*.

I. Analyse und Problemdarstellung

Geg.:

Ziel:

k: | … | k_{l-1} | k_l | k_{l+1} | … | … |

wobei: a) $k_l \leq k_{l+1}$ für alle l

b) alle Elemente aus i und j in k vorhanden

II. Entwurf einer Lösung

(1) Initialisierung der File–Durchläufe für i und j ;
Vereinbarung / Anlegen eines leeren Files k ;

(2) Solange noch nicht i und j ganz durchlaufen:
wenn Pufferinhalt $i\hat{} < j\hat{}$, dann kopiere Inhalt von $i\hat{}$ nach k ; anderenfalls kopiere Inhalt von $j\hat{}$ nach k ;

(3) Solange i noch nicht ganz durchlaufen: kopiere restl. Komponenten nach k ;

(4) Solange j noch nicht ganz durchlaufen: kopiere restl. Komponenten nach k ;

(5) Abschluss aller Dateiarbeiten;

III. Programm (-teile)

```
TYPE    basistyp = integer;
        dateityp = FILE OF basistyp;
```

```
PROCEDURE mischen (VAR i, j, k : dateityp);
```

In Turbo Pascal müssen Files VAR–Parameter sein, sonst Compiler–Fehler 126.

```
VAR x, y : basistyp;

BEGIN
      RESET (i);
      RESET (j);
      REWRITE (k);
      WHILE NOT EOF(i) AND NOT EOF(j) DO
            BEGIN   READ(i, x);
                    READ(j, y);
                    IF x < y THEN BEGIN
                                      WRITE(k, x);
                                      SEEK(j, FILEPOS(j)–1)
                                  END
                             ELSE  BEGIN
                                      WRITE(k, y);
                                      SEEK(i, FILEPOS(i)–1)
                                  END
            END;
      WHILE NOT EOF(i) DO   BEGIN
                              READ(i, x);
                              WRITE(k, x)
                            END;
      WHILE NOT EOF(j) DO   BEGIN
                              READ(j, y);
                              WRITE(k, y)
                            END;
      CLOSE(i);
      CLOSE(j);
      CLOSE(k)
END;
```