Grundlagen der Programmierung (Vorlesung 14)

Ralf Möller, FH-Wedel

- Vorige Vorlesung
  - Verifikation von Anweisungen und Anweisungsfolgen
  - Schleifen
- Inhalt dieser Vorlesung
  - Funktionen und Prozeduren
- Lernziele
  - Grundlagen der systematischen Programmentwicklung

Schleifen

Beweisregel für while –Schleifen

falls

(1) $\{I \wedge B\} \;\; S \;\; \{I\}$

dann gilt $\{I\}$ while B do S end while $\{I \wedge \neg B\}$

Schleifen (2)

Terminierung zusätzliche Bedingungen

falls

Invariante $\{I \wedge B\} \;\; S \;\; \{I\}$

Fortschritt $\{I \wedge B \wedge t > T\} \;\; S \;\; \{t = T\}$

Beschränkung $I \wedge t \leq 0 \Rightarrow \neg B$

dann gilt die Nachbedingung $I \wedge \neg B$ und die Schleife terminiert:

$$\{I\} \;\; \textsf{while } B \textsf{ do } S \textsf{ end while} \;\; \{I \wedge \neg B\}$$

Variante t ist eine ganzzahlige Funktion
T ist eine Konstante

Ganzzahlige Division mit Rest mittels While

- Spezifikation
  - var x, y, q, r : N_0
  - $\{ y > 0 \}$ S $\{ q * y + r = x \wedge r < y \}$
  - V P
  - P: $q * y + r = x \quad \wedge \quad r < y$
  - $I \quad \wedge \quad \neg B$
  - Gut: Spezifikation schon in der richtigen Form
- $\{ V \}$ S_0 $\{ I \}$;
- $\{ I \}$ while B do $\{ I \wedge B \}$ S_1 $\{ I \}$ end while $\{ P \}$

Ganzzahlige Division mit Rest mittels While

- var x, y, q, r : N_0
- { y > 0 } q, r := 0, x { q * y + r = x }

  V So I

  B
- while r ≥ y do
- {q * y + r = x ∧ r >= y} q, r := q+1, r-y {q * y + r = x}
- end while S1
- {q * y + r = x ∧ r < y}

  P

  I ∧ ¬B

Terminierungsbeweis

- Für die Variante t wähle: r
- Zeige Fortschritt:
  - $\{I \wedge B \wedge t > T\}$ S1 $\{t = T\}$ muß korrekt sein, d.h.
    $\{I \wedge B \wedge r > T\}$ q, r := q+1, r-y $\{r = T\}$ muß korrekt sein
    - Also $r \leftarrow r-y$ in Nachbedingung einsetzen
    - Damit ergibt sich $r-y = T$
    - Es ergibt sich: $r > T$, wenn $y > 0$
  - $(I \wedge t \leq 0) \rightarrow \neg B$ muß gültig sein, d.h.
    $(I \wedge r \leq 0) \rightarrow r < y$ muß gültig sein
    - Unter der Voraussetzung, daß $r \leq 0$ und $y > 0$ ist $r < y$ immer erfüllt

repeat –Schleife aus einer Bedingung und einer Anweisung kann eine repeat –Schleife aufgebaut werden

Syntax

```
repeat
        Anweisung
until Bedingung
```

Semantik mit Hilfe der while –Schleife

```
Anweisung;
while ¬Bedingung do
        Anweisung
end while
```

Semantik	informell
(1)	die Anweisung (der Schleifenrumpf) wird ausgeführt
(2)	die Bedingung wird ausgewertet
(3a)	ist das Resultat true, so wird die Ausführung des Programms hinter der Schleife fortgesetzt
(3b)	ist das Resultat false, so wird die Ausführung der Schleife wiederholt
☞	der Schleifenrumpf wird $1, 2, \ldots$ mal ausgeführt.
☞	nicht für Schleifen zu gebrauchen, deren Rumpf möglicherweise nicht ausgeführt wird.
☞	while –Schleifen allgemeiner
⚠	repeat –Schleife **nicht** effizienter als while –Schleife

for –**Schleife** Zählschleife

Syntax

for $Variable := Ausdruck_1$ to $Ausdruck_2$ do
    Anweisung
end for

Variable	ist die Laufvariable
$Ausdruck_1$	ist der Startwert
$Ausdruck_2$	ist der Endwert
Anweisung	ist der Schleifenrumpf

Semantik Bedeutung: erklärt mit einer while –Schleife

```
Variable := Ausdruck1;
Endewert := Ausdruck2;
while Variable ≤ Endewert do
      Anweisung;
      Variable := Variable +1
end while
```

☞ while –Schleifen allgemeiner

5 Funktionen und Prozeduren

5.1 Modularität

Einzelteile	der Algorithmen unabhängig vom Algorithmus selbst
$\hookrightarrow$	unabhängig entwickelbar
$\hookrightarrow$	an verschiedenen Stellen einsetzbar im selben Algorithmus in verschiedenen Algorithmen
$\hookrightarrow$	Teile betrachten wie neue Elementaroperationen

Algorithmus

quadratsumme(x,y)	quadrat(x) + quadrat(y)
Benutzung	quadratsumme(3,4)
Name	des Algorithmus: **quadratsumme**
formale Parameter	**x** und **y**
Aufruf	quadratsumme(3,4)
aktuelle Parameter	**3** und **4**
Funktionen	sind Algorithmen, die einen Wert berechnen
Prozeduren	sind Algorithmen, die einen Zustand (globale Variablen) verändern

Vorteile	von Algorithmen mit Parametern (Module, Prozeduren und Funktionen)
Schrittweise Verfeinerung	mit **top–down** Vorgehen wird in natürlicher Weise unterstützt
Abgeschlossenheit	Ein Teilalgorithmus kann unabhängig von dem Kontext, in dem er verwendet wird, entwickelt werden
information hiding	ein Modul braucht nur die Schnittstelle und eine Beschreibung **was** der Modul macht, nach außen bekannt zu machen. **Wie** ein Modul eine Funktion berechnet (mit welchem Verfahren), wird versteckt

Urbildbereich (Parameter) und Bildbereich (Ergebnistyp)

- Beispiel:
  - Funktion sin
  - Urbildbereich: R (-> Parameter)
  - Bildbereich R, genauer: Intervall [-1, 1] (-> Typ)
- Mehr als ein Parameter möglich
- Stelligkeit einer Funktion
  - Anzahl der Parameter

Funktionen: Schreibweise

- Erklärung anhand von Beispielen
- quadrat(x : N_0) : N_0
  x * x

  Ergebnistyp
  Parametertyp
  (formaler) Parameter
  Ausdruck
  Parametertrenner

- quadratsumme(a : N_0 ; b : N_0) : N_0
  quadrat(a) + quadrat(b)

  Funktionsaufruf
  mit Aktualparameter

Vereinbarung: abkürzende Schreibweise

- quadratsumme($a, b : N_0$) : N_0
  quadrat(a) + quadrat(b)
- steht für
- quadratsumme($a : N_0 ; b : N_0$) : N_0
  quadrat(a) + quadrat(b)

Vorbedingungen und Nachbedingungen

- Zum Teil Vorbedingungen als Typen der Parameter formulierbar
- Parametertypen legen die Vorbedingungen meist nur partiell fest
- Problembeispiel: Division durch 0
- div(x : N_0 ; y : N_0): N_0
- Gleiches gilt für die Ergebnistypen

Funktionsrumpf

- Folge von Anweisungen
- mit "abschließendem" Ausdruck
- $min(x : N_0 ; y : N_0) : N_0$

```
if x > y
  then y
  else x
end if
```

Funktionen: Motivation

Auswechselbarkeit von Teil–Algorithmen ohne Veränderung der Funktionalität des Gesamtalgorithmus

$\hookrightarrow$

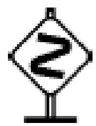
Wartbarkeit Veränderbarkeit, Erweiterbarkeit, Anpassung

Wiederverwendung Teilalgorithmen können in einer *Bibliothek (library)* von Algorithmen gespeichert und wiederverwendet werden

Information Hiding (Kapselung) (1)

Information verstecken ist eine wesentliche Aufgabe der Zerlegung in Teilalgorithmen

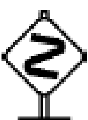
es wird nach außen nur die Schnittstelle des Algorithmus (die formalen Parameter und das Resultat) bekanntgemacht



einschließlich der Vorbedingungen

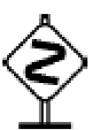
Welche Annahmen werden über die Parameter gemacht?

Information Hiding (Kapselung) (2)



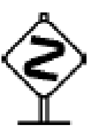
einschließlich der Invarianten

Welche globalen Variablen werden nicht verändert?



einschließlich der Nachbedingungen

Welche Eigenschaften haben die veränderten globalen Variablen?



die Realisierung (Implementierung) bleibt nach außen hin unbekannt (versteckt) und kann so verändert oder ausgewechselt werden

- Noch zu klären: Semantik des Funktionsaufrufs

Zuweisung und Arrays: Ergänzung

- Arraydenotation über Tupel
  - var f : array [0..2] of N_0; f := (42, 17, 9)
- Zuweisung von Arrays: Kopiersemantik
  - var f, g : array [0..2] of N_0;
  - f := (42, 17, 9);
  - g := f; ...
- Arrays als Werte von Funktionen (am Beispiel)
  - f(i : N0) : array [0..1] of N_0
    (i+1, i-1)
- Erweiterung der Zuweisung für Arrays
  - var a : array [0..1] of N0;   a := f(3)
  - var x, y : N_0; (x, y) := f(3)

Auswertestrategien

nicht strikte Auswertung	Boolescher Ausdrücke :
	Konjunktionen und Disjunktionen werden nur so weit ausgewertet, bis das Resultat bekannt ist.
$\hookrightarrow$	das Resultat ist auch dann definiert, wenn die Auswertung eines Teilausdrucks nicht definiert ist
$\mathbf{x \wedge y}$	$\Leftrightarrow$ if x then y else false
	$\Leftrightarrow$ if x then y else x
$\mathbf{x \vee y}$	$\Leftrightarrow$ if x then true else y
	$\Leftrightarrow$ if x then x else y

Lokale Variablen, Blöcke

- Bisher: Variablen "global" am Anfang eines Algorithmus vereinbart
- Nun: Variablen "lokal" nur für den Teilalgorithmus einer deklarierten Prozedur
- Notation:
  - begin
    var .... ; — Kopf des Blockes mit lokalen Variablen
    .... — Rumpf des Blockes (Sequenz von Anweisungen)
    end
- Blöcke können Anweisung oder Ausdruck sein

Blöcke: Freie und gebundene Variablen

- Variablen, die im Kopf eines Blockes aufgeführt sind, heißen gebunden (bzgl. eines Blockes)
- Variablen, die im Rumpf vorkommen, aber nicht gebunden sind, heißen frei (bzgl. eines Blockes)

Auswertestrategie für Blöcke

- Vor der Auswertung wird ein Block transformiert
- Für die lokalen Variablen werden neue, in der Auswertereihenfolge bisher nicht benutzte Namen vergeben
- Anschließend werden die in den Anweisungen im Rumpf des Blockes verwendeten lokalen Variablen durch die entsprechenden neuen Variablen ersetzt
- Ist die bei dieser Ersetzung betrachtete Anweisung wieder ein Block, so werden nur die freien Variablen dieses Blockes ersetzt

Funktionen: Auswertestrategie (1)

- Sei f definiert als $f(x_1: T_1, ..., x_n: T_n) : T_{result}$
- <Rumpf von f>
- Ein Funktionsaufruf $f(a_1, ..., a_n)$ einer n-stelligen Funktion f mit Aktualparametern $a_1, ..., a_n$ wird ausgewertet, indem aus den (formalen) Parametern und dem Rumpf der Funktion f folgender Block gebildet wird
  - begin

```
begin
   var x1: T1; x2: T2; ...; xn : Tn;
   <Rumpf von f>
end
```

- Anschließend wird der Block wie oben beschrieben zur Auswertung transformiert

Funktionen: Auswertestrategie (2)

- Es entsteht ein Block
  - begin
    var x_1': T_1; x_2': T_2; ...; x_n' : T_n;
    <Rumpf von f mit Ersetzungen>
    end
- Der Block wird wie folgt umgeformt und ausgewertet
  - begin
    var x_1': T_1; x_2': T_2; ...; x_n' : T_n;
    x_1', x_2', ..., x_n' := a_1', ..., a_n' ;
    <Rumpf von f mit Ersetzungen>
    end
- wobei a_1', ..., a_n' die Ergebnisse der entsprechenden Auswertung von a_1, ..., a_n sind

Parameter, Sichtbarkeit, Lebensdauer

- Der oben skizzierte Übergabemechanismus heißt Call-by-value
- Wertezuweisungen an Parameter sind möglich, da Parameter wie Blockvariablen mit Initialisierung zu behandeln sind
- Transformation garantiert Sichtbarkeitsregeln (Variablenskopus)
- Was passiert bei "Neueintritt" in den Block?
  - "Alte" Namen werden nicht neu vergeben
  - "Alte" Werte gehen also verloren

Ausdrücke vs. Anweisungen (Beispiel 2)

- amittel(g : array [0..n-1] of N_0) : N_0

```
begin
    var j, am : N0 ;
    j, am := 0, 0;
    while j < n do
        j, am := j + 1, ((j * am) + g[j]) / (j+1)
    end while;
    am
end
```

Zusammenfassung, Kernpunkte

- Blöcke
  - Auswertungsstrategie,
  - Sichtbarkeit (Skopus)
  - Lebensdauer
- Funktionen
  - Aktual- vs. Formalparameter
  - Auswertungsstrategie

Was kommt beim nächsten Mal?

- Prozeduren
- Rekursion