

# ***Grundlagen der Theoretischen Informatik***

Sebastian Iwanowski  
FH Wedel


**Kap. 3: Verifikationstechniken**  
**Teil 4: Modularisierung und funktionale Programmierung**

# Modularisierung

**Grundlegende Idee: Zerlege ein Problem in Teilprobleme**

**1. Beispiel:** Schreib einen Brief

Briefkopf;  
Anrede;  
Briefinhalt;  
Briefschluss



**Prozeduren**

***Die 4 Teilprobleme können unabhängig voneinander gelöst werden und wiederverwendet werden.***

**Vorteile:**

- 1) Übersichtlichkeit**
- 2) Wiederverwendbarkeit**

# Modularisierung

## Grundlegende Idee: Zerlege ein Problem in Teilprobleme

**2. Beispiel:** Berechne zu einem Kalenderdatum  $D$  und einer Tageszahl  $n$  das Kalenderdatum  $D+n$

```
LiesEingabedatum;  
LiesTageszahl;  
BerechneAusgabedatum;  
GibAusgabedatumAus;
```

**Problem:** Hier müssen Daten zwischen den Prozeduren ausgetauscht werden.

**Lösung:** Prozeduren mit Parametern

```
LiesEingabedatum (in);  
LiesTageszahl (n);  
BerechneAusgabedatum (in, n, out);  
GibAusgabedatumAus (out);
```

# Modularisierung

## Prozeduren mit Parametern

```
LiesEingabedatum (in);  
LiesTageszahl (n);  
BerechneAusgabedatum (in, n, out);  
GibAusgabedatumAus (out);
```

### Eingabeparameter:

- dienen der Übermittlung von Werten aus dem aufrufenden Programm an die Prozedur

### Ausgabeparameter:

- dienen der Übermittlung von Werten aus der Prozedur an das aufrufende Programm

### 1. Übergabetechnik: Call by reference (Variablenparameter)

- Prozedur und aufrufendes Programm teilen sich denselben Speicherplatz.
- In Pascal wird diese Übergabetechnik bei den VAR-Parametern angewandt.
- In manchen Programmiersprachen gibt es nur diese Übergabetechnik.

# Modularisierung

## Prozeduren mit Parametern

```
LiesEingabedatum (in);  
LiesTageszahl (n);  
BerechneAusgabedatum (in, n, out);  
GibAusgabedatumAus (out);
```

### Eingabeparameter:

- dienen der Übermittlung von Werten aus dem aufrufenden Programm an die Prozedur

### 2. Übergabetechnik: Call by value (Wertparameter)

- Prozedur und aufrufendes Programm legen für denselben Parameter verschiedene Speicherplätze an.
- Beim Aufruf wird der Wert des aufrufenden Programms in den entsprechenden Speicherplatz der Prozedur kopiert.
- Eine Rückgabe von Werten durch die Prozedur ist auf diese Weise **nicht** möglich (also nur als Eingabeparameter verwendbar).
- In Pascal wird diese Übergabetechnik bei den Nicht-VAR-Parametern angewandt.
- In manchen Programmiersprachen gibt es nur diese Übergabetechnik.

# Modularisierung

## Parameteridentifikation zwischen aufrufendem Programm und Prozedur

im aufrufenden Programm:

*Aktuelle Parameter*

```
BerechneAusgabedatum (in, n, out);
```

*Wertübergabe bzw.  
Adressübergabe  
(je nach verwendeter  
Übergabetechnik)*

in der Prozedur:

```
BerechneAusgabedatum (inDate, count, outDate);
```

*Formale Parameter*

- Es werden jeweils die Parameter identifiziert, die an der gleichen Position in der Parameterliste stehen.

# Modularisierung

## Anforderungen an die Parameter

### an die formalen Parameter (in der Prozedur):

- Parameter müssen Variablennamen sein.
- Bei Call by reference steht dieser Variablenname für die Speicheradresse, die übergeben wurde.
- Bei Call by value steht dieser Variablenname für einen in der Prozedur neu eingerichteten Speicherplatz, in den der Wert des aufrufenden Programms kopiert wird.

### an die aktuellen Parameter (im aufrufenden Programm):

- Bei **Call by reference** müssen die Parameter **Variablennamen** sein. Sie stehen für die Speicheradresse, die an die Prozedur übergeben wird.
- Bei **Call by value** dürfen die Parameter **beliebige Wertausdrücke** sein. Der Ausdruck muss vor Prozedurbeginn ausgewertet werden und wird dann in den neu eingerichteten Speicherplatz der Prozedur kopiert.

# Modularisierung

## Warum gibt es 2 Parameterübergabetechneken in derselben Programmiersprache ?

### für Pascal: Warum gibt es Wertparameter ?

- Wertparameter können bequemer mit aktuellen Parametern bedient werden (beliebige Ausdrücke sind zulässig).

*Programmieren wird **bequemer***

- Falls eine Variable als Wertparameter übergeben wird, dann kann man sich sicher sein, dass diese auch von einer nicht ganz durchsichtigen Prozedur auf keinen Fall verändert wird.

*Programmieren wird **zuverlässiger***

***Die mathematisch exakte Verifikation von Programmen mit call-by-reference-Parametern ist praktisch unmöglich !***



# Modularisierung

## Wann sollte man welche Technik einsetzen ?

**Empfehlung:** *Parameter sollten nur zur Übergabe von Werten an die Prozedur benutzt werden und nicht zur Übergabe von Rückgabewerten an das aufrufende Programm.*

*aus Übersichtlichkeits- und Zuverlässigkeitsgründen*

**Folgerung:** Dann braucht man ja eigentlich nur call by value ... ?

**Achtung:** Call by reference ist wesentlich effizienter bezüglich des Speicherplatzverbrauchs !

**Idealfall:**

1. Benutze nur call by reference.
2. Verwende den Parameter dennoch nur als Eingabeparameter.

*Aber wer gewährleistet die Einhaltung der zweiten Forderung ?*

**Kompromiss:** Benutze call by value für kleine (einfache) Daten und call by reference für große (zusammengesetzte).

# Modularisierung

## Rückgabe von Werten von der Prozedur an das aufrufende Programm

Wie soll man das bei Befolgung der eben gegebenen Empfehlung bewerkstelligen ?

Antwort:

*entsprechend der Schreibweise der Mathematik:*  $y = f(x_1, x_2, \dots, x_k)$

- Prozeduren erhalten Rückgabewerte
- Wenn der Rückgabewert ein beliebig komplexer Wert sein darf, so können wir uns auf die Forderung nach **einem** Rückgabewert beschränken.
- In Pascal heißen Prozeduren mit Rückgabewert **Funktionen**

# Funktionale Programmierung

## Prozeduren mit Rückgabewert (Funktionen)

```
in := LiesEingabedatum ();  
n := LiesTageszahl ();  
out := BerechneAusgabedatum (in, n);
```

*kein Rückgabewert* -----> **GibAusgabedatumAus** (**out**);

**in der Prozedur (Funktion):**

```
BerechneAusgabedatum (inDate, count);  
begin  
  .  
  .  
  .  
  return outDate;  
end
```

**im aufrufenden Programm:**

```
out := BerechneAusgabedatum (in, n);
```

- Die Prozedur darf in der return-Anweisung einen beliebigen Wertausdruck haben.
- Dieser wird bei der Ausführung der return-Anweisung ausgewertet
- Nach der return-Anweisung wird die Prozedur beendet und der Rückgabewert im Hauptprogramm an der entsprechenden Stelle eingesetzt.

# Funktionale Programmierung

## Verifikation bei Prozeduren

- Die Verifikation innerhalb von Prozeduren unterscheidet sich nicht von der Verifikation allgemeiner Programme.

### Einzigster neuer Punkt der Beachtung: **Parameterübergabe**

- Die aktuellen Parameter müssen die Vorbedingungen der entsprechenden formalen Parameter erfüllen.

*Die meisten Compiler unterstützen das durch Typprüfungen.*

- Der Rückgabewert muss die Vorbedingungen für die zugewiesene Variable des Hauptprogramms erfüllen.

*Analoges gilt bei der Benutzung von Variablenparametern.*

# Funktionale Programmierung

## Funktionale Programmiersprachen

- Alle Ausdrücke sind Funktionen
- Konstante sind Funktionen ohne Parameter *in der Theorie !*
- Im Prinzip kann auf Programmvariable verzichtet werden  
(Parametervariable sind aber erforderlich!)
- Funktionen haben keine Seiteneffekte: *in der Theorie !*  
Sie berechnen nur den Funktionswert in Abhängigkeit der Parameter.

# Funktionale Programmierung

## Beispiel: Die funktionale Ursprache **Lisp** (**L**ist **P**rocessing **L**anguage)

- Als „Datentypen“ gibt es nur Atome (vordefinierte Konstante) und Listen aus Atomen bzw. Listen.
- Alle Anweisungen sind Listen der Form (**Funktionsname param1 param2 ... paramn**).  
Jede Anweisung hat einen Wert. *Also Präfixnotation*  
Jeder Parameter darf ein Ausdruck mit einem beliebigen Wert sein:  
Insbesondere sind Anweisungen als Ausdrücke erlaubt.  
Es wird von innen nach außen ausgewertet.
- Funktionsdeklarationen werden mit der Funktion *defun* eingeleitet:  
(**defun Funktionsname (fParam1 fParam2 ... fParamn) Anw1 ... Anwn**)  
Der Wert der letzten Anweisung, aus der die Funktion besteht,  
wird als Rückgabewert der Funktion interpretiert.
- Funktionsaufrufe sind Listen der Form (**Funktionsname aParam1 aParam2 ... aParamn**).
- Verzweigungen und Schleifen werden durch vordefinierte Funktionen realisiert
- In den meisten Lisp-Dialekten gibt es doch Variablen und Zuweisungen.