



## **Vorlesung an der Berufsakademie Oldenburg**

### **Unterrichtseinheit 8**

### **Boolesche Funktionen (Hamster-Modell)**

**Dr. Dietrich Boles**

- Motivation
- return-Anweisung
- Test-Funktionen
  - Definition
  - Aufruf
  - Beispiele
- Seiteneffekte
- Codekonventionen
- Aufgaben

**Prozeduren:**

Definition neuer Befehle

**Test-Funktionen:**

Definition neuer Testbefehle;

boolesche Funktionen;

Funktionen, die einen Wahrheitswert liefern

`mauerDa( )`

`rechtsFrei( )`

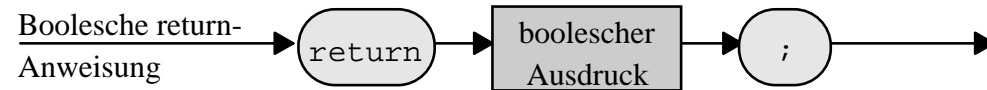
`fuenfKoernerDa( )`

`linksFrei( )`

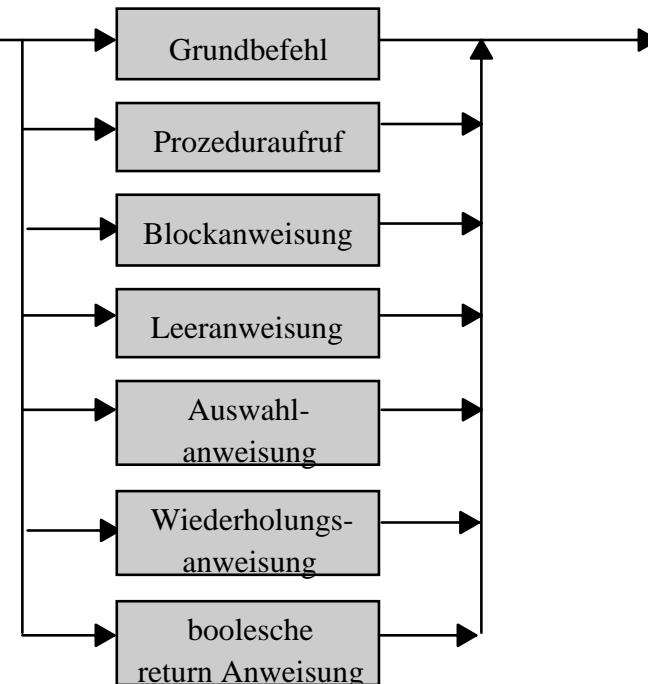
`...`

```
if (linksFrei()) {  
    linksUm();  
    vor();  
}
```

## Syntax:



Anweisung (7)

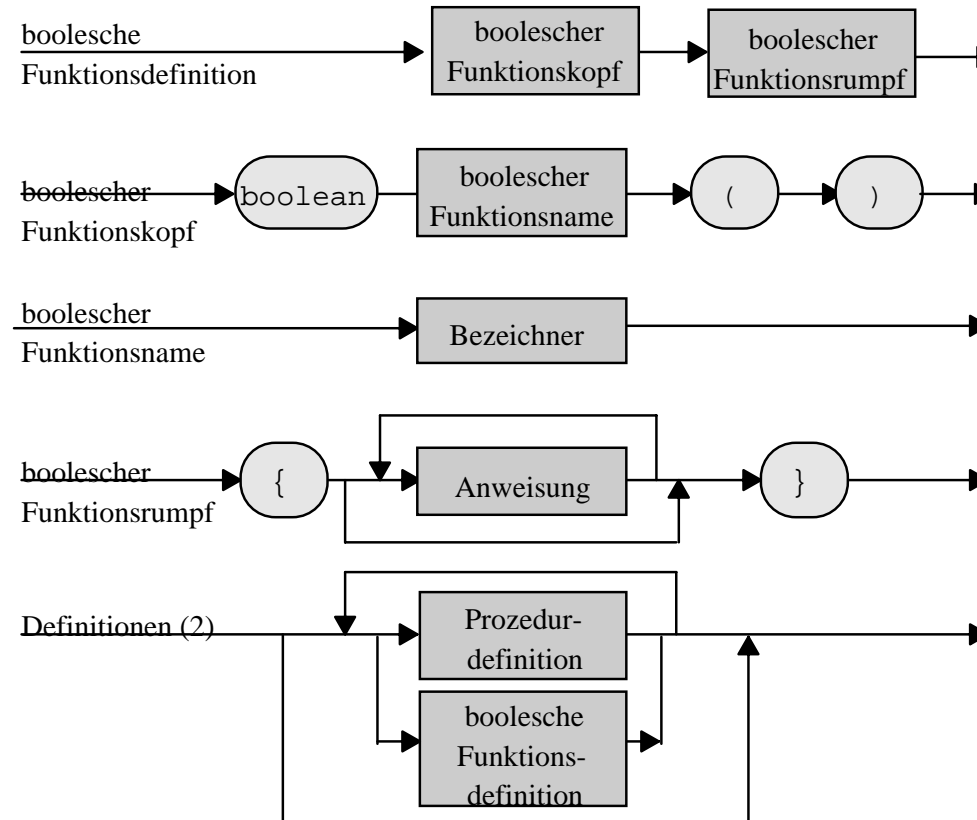


## Zusatzbedingung:

Darf nur in einer booleschen Funktion verwendet werden.

**Semantik:** Der Ausdruck wird ausgewertet und die Funktion unmittelbar verlassen. Geliefert wird der Wert des Ausdrucks.

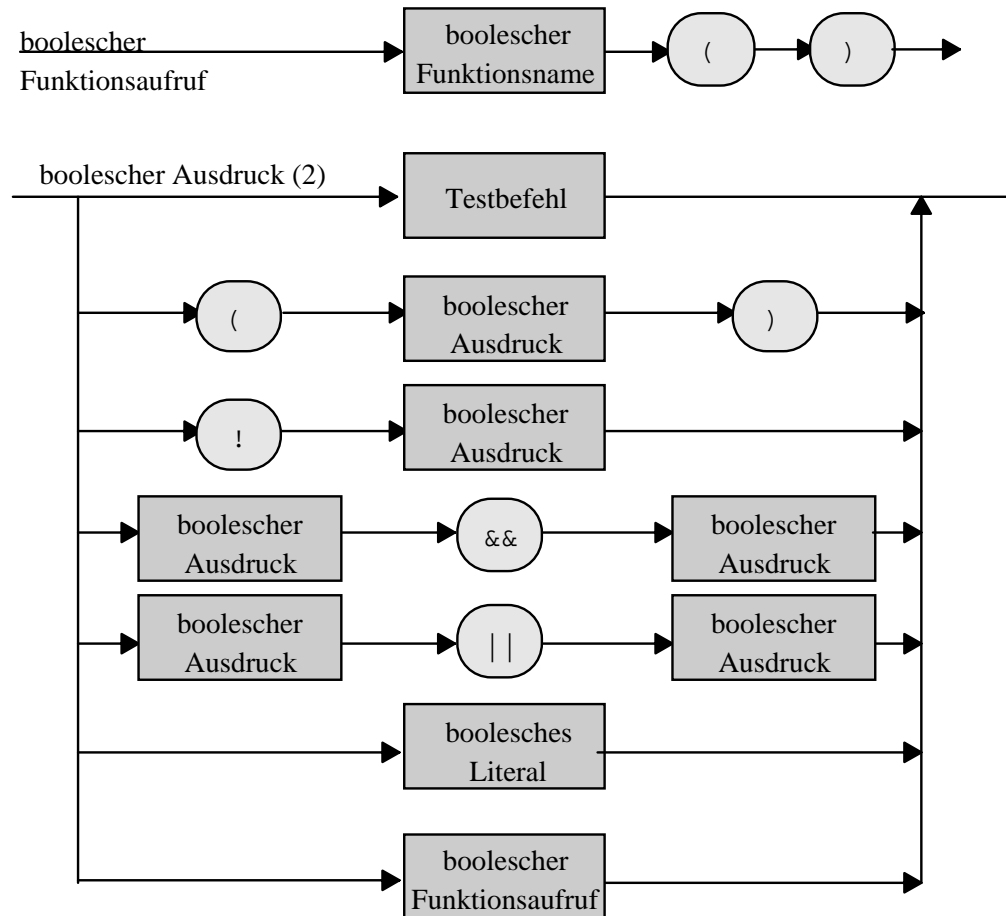
## Syntax:



**Zusatzbedingung:** In jedem möglichen Weg durch den Rumpf muss eine boolesche return-Anweisung stehen!

**Semantik:** Es wird eine neue Funktion eingeführt.

## Syntax:



## Semantik:

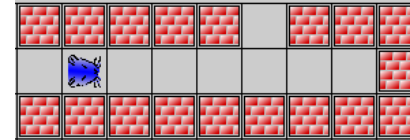
Der Funktionsrumpf wird ausgeführt. Der Wert ergibt sich aus dem von der Funktion gelieferten Wert.

## Aufgabe:

Links Nische suchen und  
hineinbegeben.

```
boolean mauerDa() {  
    return !vornFrei();  
}  
  
boolean linksFrei() {  
    linksUm();  
    if (vornFrei()) {  
        rechtsUm();  
        return true;  
    } else {  
        rechtsUm();  
        return false;  
    }  
}
```

## Typische Landschaft:



```
void main() {  
    while (!mauerDa() &&  
           !linksFrei()) {  
        vor();  
    }  
    if (linksFrei()) {  
        linksUm();  
        vor();  
    }  
}  
  
void rechtsUm() {  
    linksUm();  
    linksUm();  
    linksUm();  
}
```

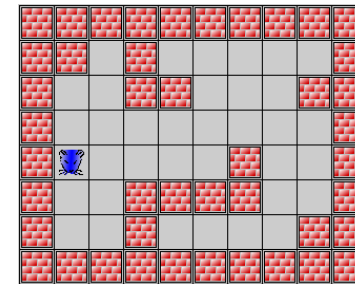
## Aufgabe:

Der Hamster (mit mindestens einem Korn im Maul)  
soll entlang der Wand laufen, bis er zur Ausgangsposition  
zurückgekehrt ist.

## Programm:

```
void main() {
    gib(); // Markierung
    vor();
    while (!kornDa()) {
        if (rechtsFrei()) {
            rechtsUm(); vor();
        } else if (vornFrei()) {
            vor();
        } else if (linksFrei()) {
            linksUm(); vor();
        } else {
            kehrt(); vor();
        }
    }
}
```

## Typische Landschaft:



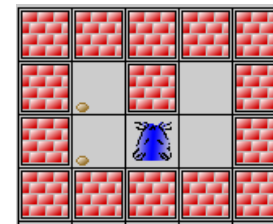
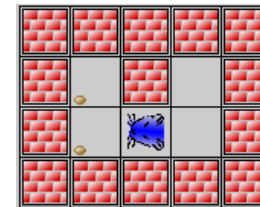
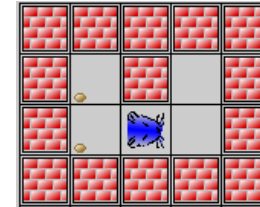
```
boolean linksFrei() {
    linksUm();
    if (vornFrei()) {
        rechtsUm(); return true;
    } else {
        rechtsUm(); return false;
    }
}
boolean rechtsFrei() {
    rechtsUm();
    if (vornFrei()) {
        linksUm(); return true;
    } else {
        linksUm(); return false;
    }
}
```



```
boolean linksFrei() {  
    linksUm();  
    return vornFrei();  
}
```

```
void main() {  
    if (linksFrei())  
        ;  
    if (!linksFrei())  
        ;  
}
```

```
void main() {  
    if (linksFrei())  
        ;  
    else  
        ;  
}
```



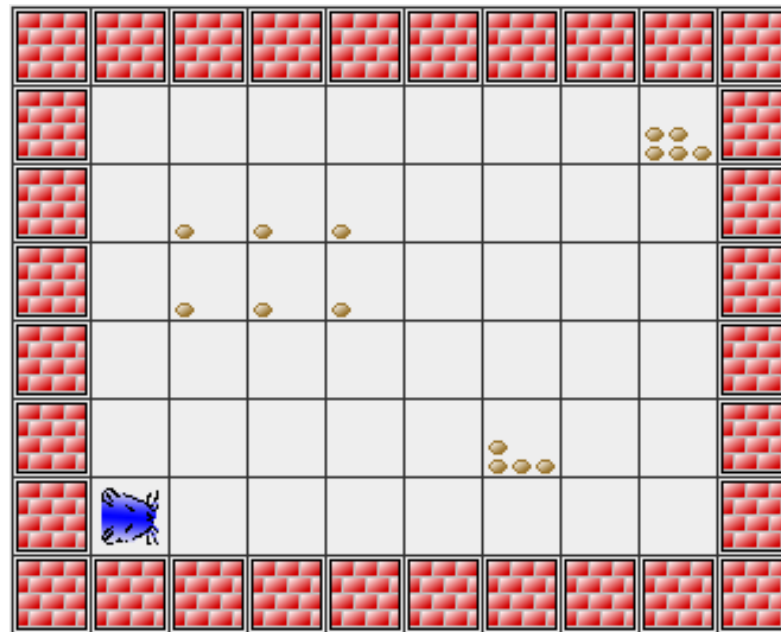
- Funktionsname: Anfangsbuchstabe klein; Anfangsbuchstaben neuer Wortbestandteile groß
- Funktionskopf und { in eine Zeile
- } unterhalb des b von `boolean`
- innere Anweisungen um 4 Spalten einrücken
- Hinter Funktionsnamen kein Leerzeichen
- Funktionsdefinition und -aufruf: Zwischen ( ) kein Leerzeichen
- Funktionsdefinition: Hinter ( ) ein Leerzeichen
- mehrere Prozedur- und Funktionsdefinitionen jeweils durch Leerzeile trennen

# Aufgabe 1

## Aufgabe:

Gegeben ein beliebiges Territorium ohne innere Mauern. Der Hamster sitzt in der linken unteren Ecke. Er soll alle Körner fressen. Er darf kein Feld zweimal betreten.

## Beispiellandschaft:



## Aufgabe 2

### Aufgabe:

Der Hamster sitzt vor einer unregelmäßigen Mulde (ohne Überhänge), die mit Körnern gefüllt ist. Er soll alle Körner einsammeln.

### Beispiellandschaft:

