Hochschule für
Technik und Wirtschaft
des Saarlandes
University of Applied Sciences

Programmiersprachen II

Integration verschiedener Datenstrukturen

Übung 17

Prof. Dr. Reiner Güttler
Fachbereich GIS
HTW

Kapitel 3: Folge Datenstrukturen

Ziel:

- Komplexe Übung als „Mini-Projekt“ mit Einsatz verschiedener bisher behandelter Datenstrukturen
- Besonderes Gewicht auf sauberem objektorientierten Ansatz
- „reales“ Projekt mit konkretem fachlichen Hintergrund (Compilerbau)
- Dabei weglassen von allen realen Details, die das Projekt „aufblähen“, ohne konzeptionell etwas zusätzliches beizutragen

Kapitel 3: Folge Datenstrukturen

Hintergrund:

Von den verschiedenen Modulen eines Compilers sind zwei ausgewählte Gegenstand des Projekts:

- Die sog. Symboltabellenverwaltung: Hierunter versteht man das Verwalten einer Datenstruktur, die in allen Compilerphasen die notwendigen Informationen zu einem im Programm vorkommenden Identifier bereitstellt
- Das Parsen von arithmetischen Ausdrücken mit Erzeugung von auswertbaren „Expression-Trees

Kapitel 3: Folge Datenstrukturen

Symboltabellenverwaltung:

Pro Identifier ein Eintrag mit den zugehörigen Informationen wie

- Der Identifier selbst
- Adresse
- Typ
- Geltungsbereich
- Wert
- ...

Für unsere Aufgabe: nur Identifier und Wert

Kapitel 3: Folge Datenstrukturen

Rahmenbedingung:

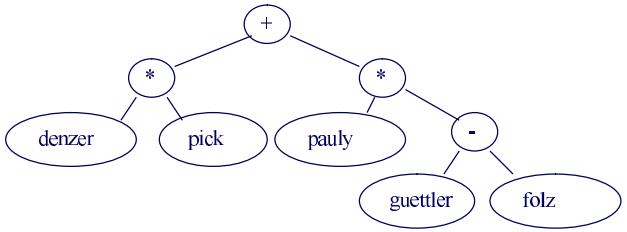
- dynamisch (für grössere und kleinere Quellprogramme)
- Strings als Schlüssel
- benötigte Operationen:
  - ⇨ insert() bei Deklaration
  - ⇨ find() bei Zugriffen

Fazit: Hash-Tabelle ist eine sehr geeignete (und meistens auch gewählte) Datenstruktur

Kapitel 3: Folge Datenstrukturen

Ausdruck: denzer*pick+pauly*(guettler-folz)

Zugehöriger expression-tree (konzeptionell):



Kapitel 3: Folge Datenstrukturen

Parsen:

Gegeben: ein arithmetischer Ausdruck der Form (denzer*pick+pauly*(guettler-folz))

Zur Vereinfachung:

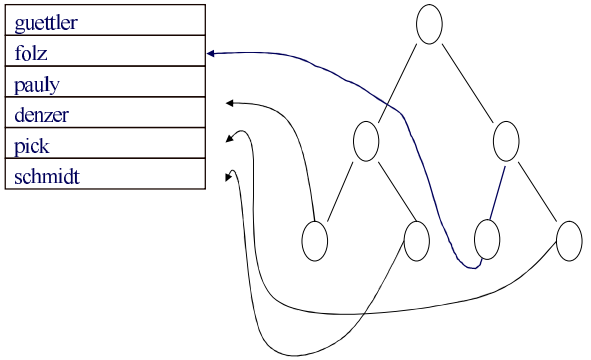
- Nur Identifier als Operanden (keine Konstanten)
- Nur +,- und *,/ (mit Prioritäten!) als Operatoren
- Klammerung beliebig (d.h. auch nicht notwendige Klammerung erlaubt)

Ziel

- Arithmetischen Ausdruck analysieren
- Konstruktion eines auswertbaren expression-tree's, der die Semantik des Ausdrucks genau wiederspiegelt

Kapitel 3: Folge Datenstrukturen

Für unsere Aufgabe (vgl. Übung 14)



Kapitel 3: Folge Datenstrukturen

Auswertung:

Wenn jedem Identifier in der Tabelle ein Wert zugeordnet ist, ist die Auswertung des Baums offensichtlich:

- Bei jedem Knoten wird auf die schon berechneten (oder bei Blättern: aus der Symboltabelle geholten) Werte der beiden Söhne die Knotenoperation angewendet.
- Das Ergebnis der Operation erhält der Knoten als Wert.
- Der Wert der Wurzel entspricht dem Wert des gesamten arithmetischen Ausdrucks.

Kapitel 3: Folge Datenstrukturen

Parsen:

Wir benötigen zwei stacks:

- Einen Operandenstack opndStack
- Einen Operatorstack optStack

Verfahren an Tafel vorführen für:

a+b+c

a*b+c

a+b*c

a+b*(c-d)

(a+b*(c-d))

((a+b)*((c-d)))

Kapitel 3: Folge Datenstrukturen

Regeln: lies den Eingabestring item für item und jeweils folgende Regeln an (pseudo-code)

Eingabe-Item	Aktion
Operand opnd	Kreiere opnd-Knoten, push Knotenpointer auf opndStack
(	push (auf optStack
)	While optStack nonempty repeat das nachfolgende {pop opTop aus optstack if opTop == (exit loop else kreiere opTop-Knoten mit rechter Sohn = pop von opndStack linker Sohn =pop von opndStack push opTop_knotenpointer auf opndStack}

Kapitel 3: Folge Datenstrukturen

Operator opThis	If optStack empty push opThis auf optStack else {while optStack nonempty repeat das nachfolgende {pop opTop aus optStack if opTop == (push it auf optStack else if opTop < opThis push it else kreiere opTop-Knoten wie vorne push opTop_knotenpointer auf opndStack if opTop==(or opTop<opThis exit loop} push opThis}

Kapitel 3: Folge Datenstrukturen

Eingabe leer	while optStack nonempty repeat das nachfolgende {pop opTop aus optStack kreiere opTop-Knoten wie vorne push opTop_knotenpointer auf opndStack}

Noch einzufügen:

- Endbedingungen
- Fehlerfälle und Fehlerbehandlung
- Empfehlung: einfach bis zum Ende parsen und danach die stacks überprüfen (was dürfen die im korrekten Fall noch enthalten??)

Kapitel 3: Folge Datenstrukturen

Projektaufgabe

Entwickeln sie eine Applikation zum

- Initialisieren einer Symboltabelle durch Deklarationen (Form siehe später): insert() der deklarierten Identifier
- Parsen eines arithmetischen Ausdrucks unter Verwendung der Identifier in der initialisierten Symboltabelle (Identifier nicht deklariert => Fehler)
- Generieren des zugehörigen expression-trees (inkl. Ausgabe)
- In einer Schleife
  - ⇨ Zuweisen von Werten an die Identifier
  - ⇨ Auswerten des Trees

Kapitel 3: Folge Datenstrukturen

Dazu benötigen sie:

- Symboltabelle (Hash-Tabelle)
- expression-tree (binärer Baum, wie in Übung 14)
- Parsen (inkl. der benötigten stacks und der Erzeugung des expression-trees)
- Wertzuweisung
- Auswertung
- Jeweils inkl. Ausnahmenbehandlung:
  - ⇨ bzgl. Form und Vollständigkeit der Eingabe
  - ⇨ beim Parsen: zwei Typen von „Parse-Fehler“
    - unkorrekter arithmetischer Ausdruck
    - Verwendung einer nicht deklarierten Variable

Kapitel 3: Folge Datenstrukturen

Form der Eingabe:

- Deklaration:
  - ⇨ Nur die zu benutzenden Identifier, je einen pro Zeile
- Arithmetischer Ausdruck:
  - ⇨ Genau ein arithmetischer Ausdruck (Trennung durch Blanks optional)
- Wertzuweisung: in einer Schleife jeweils ein „Eingabepaket“
  - ⇨ Je ein Integerwert pro Identifier, je einen pro Zeile

Kapitel 3: Folge Datenstrukturen

Eingabebeispiel (korrekt):

folz
denzer
guettler

folz + denzer * guettler

folz = 73
denzer = 22
guettler = 5

folz = 7
guettler = 2
denzer = 1

Kapitel 3: Folge Datenstrukturen

Ausgabe:

- Der expression tree, dabei
  - ⇨ Form wie in PS1 (Wurzel links)
  - ⇨ für jeden Operatorknoten der jeweilige Operator
  - ⇨ für jeden Operandenknoten der entsprechende Identifier aus der Symboltabelle
- Pro „Eingabepaket“ das Ergebnis der Auswertung

Kapitel 3: Folge Datenstrukturen

guettler
*
denzer
+
folz

183

9

Kapitel 3: Folge Datenstrukturen

Objektorientierung?

u.a.

- auf Kapselung achten, z.B.
  - ⇨ Ein Auswechseln der Symboltabellenimplementierung z.B. durch Binärbaum (das machen manche Compiler!) darf keinen Einfluss auf die anderen Klassen haben.
  - ⇨ Ein Auswechseln des Parsealgorithmus z.B. durch einen „Kellerautomaten“ (das machen die meisten Compiler!) darf keinen Einfluss auf die anderen Klassen haben.
- saubere objektorientierte Lösung für folgendes:
  - ⇨ Jeder Knoten im exression-tree kann potentiell zwei verschiedene Typen von Söhnen haben.
  - ⇨ Der Operandenstack kann Einträge verschiedenen Typs haben.