Programmiersprachen-konzepte

Vorlesung im Wintersemester 2012/2013

Dozent: Dr.Jutta Göers
goeers@uos.de

HiWi: Christoph Waßmuth
cwassmut@uos.de

Allgemeines

- Vorlesungen: Mi. 12 -14, 449a || Do. 9 – 9:45, E05
- Übung Do., 8:15 -9 (flexibel), E05
- wöchentliche Übungsblätter/Testate,
  abzugeben bis Mi, 8:00, per Mail an cwassmut@uos.de
- Rechnerraum/Testate: Raum 145, „rechtes Drittel“
  Zeiten: s. Liste an 145

- Skript, Übungsblätter, Musterlösungen:
  http://www-lehre.inf.uos.de/psk

- Klausur: (vorauss.) 31.1.2013, 8-10 Uhr
  Voraussetzung: mind. 50% pro Übungsblatt

Literatur

- J.C. Mitchell: Concepts in Programming Languages Cambridge University Press, 2003, ~70€, (e-Book 44€)
- R.W. Sebesta: Concepts of Programming Languages, 10e, Addison-Wesley, 2012, ~ 130$
- P. Henning, H. Vogelsang: Handbuch Programmier-sprachen, Hanser, 2006 (vergriffen) bzw. Taschenbuch Programmiersprachen, 2e, Hanser, 2007, 30€
- R. Sethi: Programming Languages – Concepts and Constructs 2e, Addison-Wesley, 1996
  (seit Version 2000: statt Pascal und C mehr C++ und Smalltalk, Scheme als funkt. Sprache)
- C. Ghezzi, M. Jazayeri: Programming Language Concepts, 3e, Wiley, 1997, 140€
  (Programming Languages – A Complete View 2004, z.Zt. vergriffen)

Motivation

Warum Beschäftigung mit PSK?

- besseres Verständnis von Sprachkonzepten führt zu besserem Sprachgefühl
- Wissen um Sprachkonzepte allgemein: neue Sprachen leichter erlernbar
- mehr Hintergrundwissen zu Sprachen: passendere Sprache für Problemstellung auswählbar
- Wissen um Sprachkonzepte einer Sprache: intelligentere Lösungen programmieren
- neue Features einer bereits bekannten Sprache lernen

Motivation: Bewertungskriterien

Zweck der Vorlesung:

- Konzepte von Sprachkonstrukten lernen
- in ihrer Wichtigkeit bewerten

Eigenschaften	Lesbarkeit	Schreibbarkeit	Zuverlässigkeit
Einfachheit	X	X	X
Orthogonalität	X	X	X
Kontrollstrukturen	X	X	X
Datentypen/-strukturen	X	X	X
Syntax-Design	X	X	X
Abstraktionsunterst.		X	X
Expressivität		X	X
Typ-Prüfung			X
Ausnahmebehandlung			X
Eingeschr. Aliasing			X

Lesbarkeit

Einfachheit:

- geringe Anzahl an Grundkonstrukten
- möglichst keine Multiplizität bei Features
- möglichst kein Überladen von Operatoren

Orthogonalität:

- Unabhängigkeit der Konstrukte: jedes kann mit jedem kombiniert werden
- Einschränkungen: Ausnahmen, die in Sprachregeln beschrieben sind
- höhere Orthogonalität → höherer Grad in Regelmäßigkeit im Sprachdesign → Sprache leichter zu lernen, lesen, verstehen

Lesbarkeit

Kontrollstrukturen:

- zwischen 1950 und 1970 nur GoTo (BASIC, Fortran,..)
- danach: while, repeat, for, …
- heutiger Standard in strukturierten Sprachen

Datentypen, Datenstrukturen:

- fehlen grundlegende Typen/Strukturen: Simulation durch die vorhandenen
- fehlt boolean: `int gefunden = 1;` //Bedeutung t o. f ??
- fehlt Record-Typ (Fortran 95): Abhilfe z.B. mittels Arrays

```
Character (Len = 30) :: Name(100), Fach(100)
Integer :: MatrNr(100)
```

Lesbarkeit

Syntax-Design:

- Syntax/Form der Sprachelemente wichtig für Lesbarkeit
- Problem Klammerung: zu welcher `{` gehört welche `}` ?
  → lesbarer: `if … end if` oder `loop … end loop`
- Erlaubte Form der Identifier:
  Fortran77: max 6 Zeichen
  ANSI Basic: 1 Buchstabe
  oder 1 Buchstabe gefolgt von 1 Ziffer
- reserviertes Wort mit unterschiedlicher Bedeutung an unterschiedl. Stellen im Programm
- besser „sprechende“ reservierte Wörter (`while`, `class`, `if`, …)
  Gegenbeispiel: `grep`

Schreibbarkeit

Wichtig: immer bezogen auf Anwendungsbereich

Einfachheit:

- zu viele Grundkonstrukte: einige dem Programmierer unbekannt

Orthogonalität:

- zu viel → evtl. unauffindbare Programmierfehler

Kontrollstrukturen, Datentypen u. –strukturen:

- halten Programme einfacher, kürzer

Syntax-Design:

- ermöglicht verständlichere Programme

Schreibbarkeit

Unterstützung von Abstraktion:

- Strukturen/Operationen definierbar, die Details verstecken
- Prozessabstraktion: Unterprogramme; erledigen wiederkehrende Berechnungen
- Datenabstraktion: abstrakte Datentypen durch Grundkonstrukte einfach umsetzbar

Expressivität:

- spezielle Konstrukte um komplexe Dinge einfach auszudrücken
- `count++;  for(i=0;i<10;i++){…}`

Zuverlässigkeit

Alle Kriterien von Les- und Schreibbarkeit:

- schlechte Schreibbarkeit → unnatürliche Umsetzung der geforderten Funktionalität, fehlerhafte Programme
- schlechte Lesbarkeit → Programme schlecht zu korrigieren, warten, erweitern

Typ-Prüfung:

- Suche nach Typfehlern während Übersetzungs-/Laufzeit
- frühe Fehlerentdeckung → billigere Fehlerbeseitigung
- Original-C: keine Typprüfung bei akt. Funktionsparametern → falsche Ergebnisse z.B. bei Funktionsaufruf mit int statt float

Zuverlässigkeit

Ausnahmebehandlung:

- Laufzeitfehler abfangen, kontrolliert reagieren, Programm fortsetzen
- nur wenige Sprachen mit Exception Handling: z.B. C++, Java, Ada

Aliasing:

- auf Speicherzelle kann über mehrere Namen in Progr. zugegriffen werden
- Gefahr für Zuverlässigkeit!

weiteres Bewertungskriterium: Kosten

+ Kosten für Sprachschulungen (Einfachheit, Orthogonalität, Erfahrung der Programmierer)
+ Kosten der Programmerstellung in best. Sprache (Schreibbarkeit, Programmierumgebung)
+ Compilerkosten (Anschaffung, Compilezeit ← Code-Optimierung)
+ Kosten Programmausführung (Laufzeit ← Codeoptimier-ung)
+ Kosten für Instanthaltung/Wartung (Lesbarkeit)
+ evtl. Kosten für Portierung

Sprach-Design

- Berücksichtigung der obigen Kriterien (beeinflusst durch Anwendungsbereich)
- Computer-Architektur Rätsel.jpg
- Programmiermethodik / Sprachparadigma
- Programmierumgebung / Implementationsmethode

Inhaltsverzeichnis

Inhaltsverzeichnis (2)