Grundlagen der Informatik für Ingenieure I

18 Programmieren und Programmiersprachen

- Sicher werden Sie oft in Ihrem Leben danach gefragt werden, ob Sie die Sprache xy kennen oder schon einmal mit dem System z gearbeitet haben. Diese Frage ist fast so absurd, als ob man Sie als Führerscheininhaber fragen würde, ob Sie Opel Astra fahren können.
- Eine selbstbewusste Antwort könnte lauten:
  “Ich kann Programmieren! Ich kenne die Grundlagen der objektorientierten Programmierung. Ich habe bereits Programme in Java, ... geschrieben. Es ist für mich aber überhaupt kein Problem auch Programme in anderen Sprachen zu formulieren!”
- Dazu einige Unterscheidungsmerkmale der verschiedenen (techn./wiss.) Sprachen:

18.1 Java vs. C

❑ Stellen Sie sich vor, Sie würden den gesamten Code einer Applikation in der Methode *main()* unterbringen (wobei natürlich Einzelaufgaben in Unterprogrammen abgehandelt werden) - dann haben Sie fast ein C-Programm!

Ein furchtbarer Gedanke? Sie haben OO-Programmierung verstanden!

- C ist eine Systemprogrammiersprache, Effizienz hat sich anderen Anforderungen unterzuordnen. Es ist eine “hardwarenahe” Sprache; beim Programmentwurf werden häufig Kenntnisse von Hardware- und Betriebssystemmechanismen genutzt. Der Abstraktionsgrad des APIs ist im Vergleich zu Java gering.
- Alles was sie über Klassen und Objekte gehört haben, gibt es in C nicht.
- Die “Dekompositionseinheit” ist das Unterprogramm *(subroutine, function).*
- C kennt ein *Preprocessing (#define; #include...);* ein auf Sourcecodeebene vorhandener Mechanismus, große Programmsysteme zu strukturieren.

18.1 Java vs. C

- C kennt: *typedef, enum, struct, union*
  - Bis auf *union* strukturierte Datentypen. Die sind in Java entbehrlich, da hierfür Objekte zur Verfügung stehen.
  - *unions* sind ohnehin überflüssig, sie wurden erfunden, um Speicherplatz zu sparen.

18.1 Java vs. C

- Der Programmstrukturierung dienen *functions* (Unterprogramme mit einem Returnparameter (und Typ)) und *procedures* (Unterprogramme ohne Returnparameter).
- C kennt globale Variable.
- C kennt als primitiven Datentyp zusätzlich noch *unsigned.*
- Kontrollstrukturen
  - C kennt ein *goto*, Anwendung ist aber auch dort “verpönt”!
  - C kennt keine *labelled break* und *labelled continue.*
- Ein Basiskonzept in C sind (Zeiger) Pointer, ein elegantes und effizientes, aber auch fehlerträchtiges Verfahren, auf Daten und Funktionen zuzugreifen. Dieses Zugriffsverfahren entzieht sich weitgehend der Kontrolle eines Compilers!
  - In Java gibt es nur die Anwendung des dereferenzierten Pointers, den wir als Referenz bezeichnet haben. Ein direkter Zugriff auf den Zeiger selbst ist in Java nicht möglich. (Siehe auch Kapitel 14)

18.1 Java vs. C

18.1 Java vs. C

◆ Was macht das Arbeiten mit *Pointern* (Zeigern) so schwierig?

- Wenn man mit Zeigern arbeitet, nutzt man Kenntnisse darüber, wie Compiler Daten im Speicher organisieren.
- Zeiger sind also Variable, deren Inhalte Speicheradressen sind.
- Zeiger haben einen Typ!!!; dieser Typ bestimmt die Basiseinheit der numerischen Operationen auf den Zeiger.
- Da man bei Operationen unterscheiden können muss, ob man den Zeiger selbst oder das Element meint, auf das der Zeiger zeigt, brauch man dafür ein Ausdrucksmittel. Der Zugriff auf eine Variable über einen Zeiger nennt man: *Dereferenzieren.*
- Zeiger können selbst wieder Referenzen auf Zeigervariablen enthalten, so dass *Dereferenzierungen* mehrstufig sein können.
- Zeiger können auch “Einsprungadressen” von *functions* enthalten!

18.2 Java vs. C++

- C ist eine echte Untermenge von C++; insoweit gilt das im Vorkapitel gesagte.
- Zusätzlich bietet C++ OO-Mechanismen an, ähnlich wie Sie sie kennen, jedoch kann der Compiler und das Laufzeitsystem nicht die Einhaltung der Grundprinzipien der OO-Programmierung erzwingen. Ein wesentliches Problem sind dabei Zeiger, durch deren Benutzung keine Kapselung durchsetzbar ist.
- C++ kennt *multiple inheritance*.

18.3 Java vs. Fortran95

- Fortran (FormulaTranslation) ist die älteste “höhere” Programmiersprache, Fortran95 die aktuellste Version.
- Fortran ist eine “Numerik” - Programmiersprache, numerische Effizienz hat sich anderen Anforderungen unterzuordnen.
- Die “Dekompositionseinheit” ist das Unterprogramm; eine weitere statische Strukturierungsmöglichkeit (mit Kapselung) ist der Modul.
- Fortran hat die am besten codeoptimierenden, vektorisierenden und autoparallelisierenden Compiler. Alle “Sprachkonstruktionen”, die geeignet sind, diese Möglichkeiten zu behindern, haben kaum eine Chance in Fortran realisiert zu werden.
- Standardisierte Compileranweisungen unterstützen die Codeoptimierung des Compilers und erhalten die Source-Code-Kompatibilität des Programms.
- Fortran verfügt nur über einen eingeschränkten Zeichensatz, Klein- und Großbuchstaben werden nicht unterschieden.
- Keywords sind nicht geschützt.

18.3 Java vs. Fortran95

- Statementende ist Zeilenende (Fortsetzungszeilen).
- “Block”-Klammerung sind Schlüsselwortpaare wie

  ```
  DO ... ENDDO; IF ... ENDIF; (anstatt {})
  ```
- Statement-Ende ist das zeilenende(72 Zeichen / Lochkarte!) Es gibt die Möglichkeit Fortsetzungszeilen zu spezifizieren.
- Fortran kennt den Datentyp *complex.*
- "()" werden sowohl bei Feldern als auch bei Unterprogrammdefinition bzw. -aufrufen verwendet. Deshalb benötigt man Schlüsselworte wie SUBROUTINE, FUNCTION und CALL.
- Felder können in beliebigen Intervallen angelegt werden; auch negativer Index ist möglich; default ist “1” der 1. Index.
- Fortran kennt spezielle Feldanweisungen, mit denen Feldbereiche - fast beliebiger Konfiguration - miteinander verknüpft werden können. Diese Ausdrucksmöglichkeiten unterstützen den Compiler bei der Autovektorisierung und Autoparallelisierung (SPMD = Single Programm Multiple Data).
- Das E/A-System ist integraler Bestandteil der Sprache.