Algorithmen und Datenstrukturen im WS 2005/06

Prozeduren und Funktionen

Prozeduren und Funktionen

- Anwendungen von Prozeduren
  - gemeinsam benutzbare Anweisungsfolge
    - Speicherplatz sparen
      - heute völlig überholt
        - in der Regel ausreichend Speicherplatz
        - Prozeduren bei der Übersetzung häufig expandiert
        - Inline-code
        - Ausführung expliziter Anweisungsfolgen schneller
    - Schreibarbeit im Programm
      - Anweisungsfolge nur einmal notiert
      - durch kurzen Befehl immer wieder ausgeführt
    - gemeinsame Anweisungsfolgen zur Strukturierung

Prozeduren und Funktionen

- Anwendungen von Prozeduren
  - Subroutine in Fortran
    - belegt eigenen Bereich im Speicher
    - konkrete Anweisungsfolgen
    - von jeder anderen Stelle im Programm erreichbar
    - eigener permanenter Speicher
      - Werte über mehrere Aufrufe einer Subroutine festgehalten
    - Heute aktuelle Werte auf Stack (dynamisch)
      - Algol60: own
      - C, Java: static
    - Prozeduren berechnen Wert
      - Funktionen (*function*) oder Funktionsprozeduren
    - Prozedur ohne Ergebnis ändert Zustand des Programms
      - Änderung einer Datei
      - Änderung eines globalen Werts
      - Datum auf anderes Medium ausgeben
      - Seiteneffekte

Prozeduren und Funktionen

- Ausführung von Prozeduren
  - Benötigte (aktuelle) Parameter (a. Argument)
    - gespeichert,
    - geändert oder
    - übertragen
  - aktuellen Parameter
    - explizit angeben
      - Werte beim Aufruf der Prozedur festlegen
    - voreingestellt (default)
    - implizit (implicit)
      - Werte aus Umgebung über Namensmechanismus
      - statische Umgebung
        - Umgebung, in der Prozedur definiert
      - dynamische Umgebung
        - Umgebung, in der Prozedur aufgerufen
      - ***Gültigkeitsbereichs*** eines Objekts

Prozeduren und Funktionen

- Ausführung von Prozeduren
  - Gültigkeitsbereichs eines Objekts
    - Verständnis der verwendeten Mechanismen
- Blöcke und Gültigkeitsbereiche
  - In meisten Programmiersprachen
    - Objekte über Namen referenziert
      - jedem Namen in Programm Objekt zugeordnet
      - Beziehung zwischen Namen und Objekten kann sich innerhalb eines Programms ändern
    - Es ist wichtig zu verstehen, wie diese Beziehung in verschiedenen Sprachen definiert
    - als Bindung (binding) bezeichnet

Prozeduren und Funktionen

- Blöcke und Gültigkeitsbereiche
  - Objekt
    - Speicherplatz und
    - Name für diesen Speicherplatz
    - Objekte bilden Gedächtnis eines Programms
  - zugreifen auf Wert eines Objekts
    - Name oder Bezeichner (*identifier*) für Objekt bekannt
  - Bindung (*binding*) des Namens an das Objekt
    - Relation zwischen Namen und Objekt
  - Attribute
    - (Bezeichnung bedeutet anderes als Attribute eines Modells)
    - die einem Namen zugeordneten Eigenschaften, außer Objekt
      - Typ,
      - Konstante,
      - Prozedur, usw.

Prozeduren und Funktionen

- Blöcke und Gültigkeitsbereiche
  - Namen können sehr verschiedenartige Attribute zugeordnet werden
  - In meisten Programmiersprachen sorgfältig zwischen Namen zugeordneten Attributen unterscheiden
    - gleiche Attribute können verschiedene Namen haben
    - gleiche Namen an verschiedene Attribute gebunden sein
  - auch sorgfältig unterscheiden zwischen
    - Gültigkeitsbereich,
    - Lebensdauer und
    - Sichtbarkeit von Namen und Attributen

Prozeduren und Funktionen

- Blöcke und Gültigkeitsbereiche
  - Bindung zwischen Attributen und Namen
    - während der Übersetzungszeit eines Programms
      - (statische Bindung) oder
    - während der Laufzeit
      - (dynamische Bindung)

Prozeduren und Funktionen

- Blöcke und Gültigkeitsbereiche
  - Aus praktischer Gesichtspunkt
    - Name vor Übersetzung gebunden
    - Man unterscheidet die Bindung von Namen und Attribut
      - zum Zeitpunkt der Sprachdefiniton,
        - beispielsweise Konstante wie true, false oder PI == 3.14159265..;
      - während der Implementierung des Compilers
        - z.B. Werte von MaxInt und MinInt;
      - während der Übersetzung,
        - was der statischen Bindung entspricht;
      - während des Bindens anderer Programme zur Definition anderer Namen;
      - während des Ladens in den Speicher zum Binden an spezielle Speicher-adressen;
        - während der Laufzeit des Programms, was der dynamischen Bindung entspricht.
    - Hier nur statische und dynamische Bindung

Prozeduren und Funktionen

- Blöcke und Umgebungen
  - Blöcke (block)
    - Bereiche eines Programms
    - Bezeichnung seit Algol60
    - Umgebung (environment) des Blocks
      - Menge der in einem Block bekannten Namen
      - Block kann als Anweisungsfolge betrachtet werden
        - wird i.d.R. an spezifizierter Stelle auch ausgeführt
        - keine mehrfach benutzbaren Anweisungsfolgen
    - Schlüsselwörter für Blöcke in
      - Algol60, Simula67, Algol68, Pascal, Modula-2, Ada ...
      - ```
        Begin
           ...
        end
        ```
    - Ausnahme: C, C++, Java
      - { ... }
        ```

Prozeduren und Funktionen

- Blöcke und Umgebungen
  - Blöcke (block)
    - am Beginn eines Blocks
      - neue Namen an neue Attribute binden
        - nicht Pascal
    - neue Variablen oder Prozeduren deklarierbar
      - in C, Java in Blöcken keine neuen Prozeduren definierbar
    - C++ und Java
      - neue Variablen an jeder Stelle deklarierbar
    - in Ada schreibt man
      - ```
        declare
        -- Definitionen und Deklarationen
        begin
        -- Anweisungen
        end;
        ```
    - Hier: Java (oder so ähnlich)
        ```

Prozeduren und Funktionen

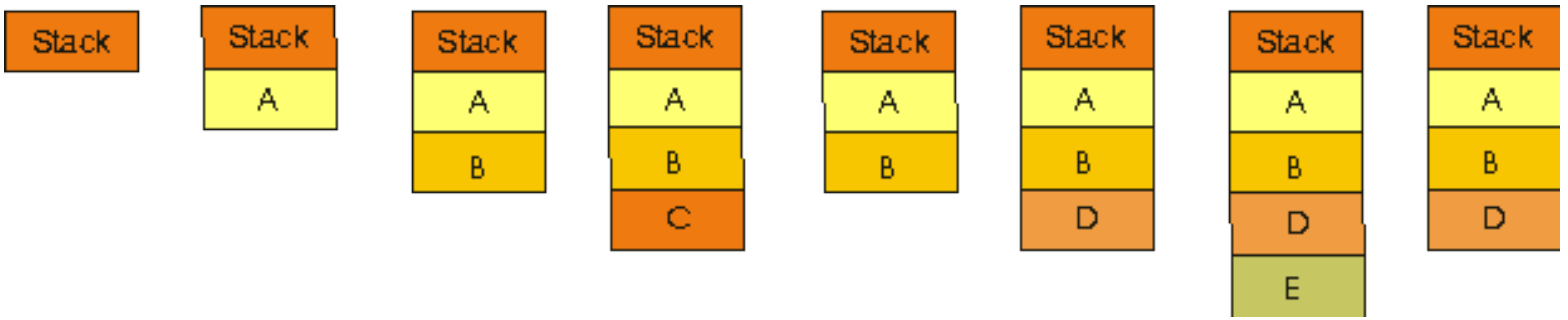
- Blöcke und Umgebungen
  - Umgebung eines Blocks
    - von meisten Programmiersprachen bewahrt
    - im Block sind alle in der Umgebung des Blocks bekannten Namen im Block bekannt
    - im Block definierte Namen
      - Bindung an neue Attribute dynamisch
      - wird bereits bekannter Name neu definiert
        - ändert sich seine Bedeutung
          - überdeklarierte oder
            - neues Objekt deklariert
          - überdefinierte Namen
            - ein anderes wird Attribut definiert

Prozeduren und Funktionen

- Blöcke und Umgebungen
  - Umgebung eines Blocks
    - in allen Programmiersprachen üblich
      - Werte von Objekten nach der Ausführung des Blocks nicht verändert
      - gleiche Werte wie vor Block
        - es sei denn, in Block verändert
        - bei überdeklarierten Namen nicht ohne weiteres möglich
    - frühere Bindungen wiederherzustellen
      - technisch meist durch Aktivierungssegmenten (*activation record*)
        - Menge von Namen und Objekten nicht an fester Stelle im Speicher
        - sondern in dynamischer Struktur,
          - Stapel oder Stack
        - unbegrenzte Anzahl von Datensätzen
          - aufnehmen und
          - wieder abgeben,
          - zuletzt aufgenommener Datensatz jeweils als erstes wieder entfernt

Prozeduren und Funktionen

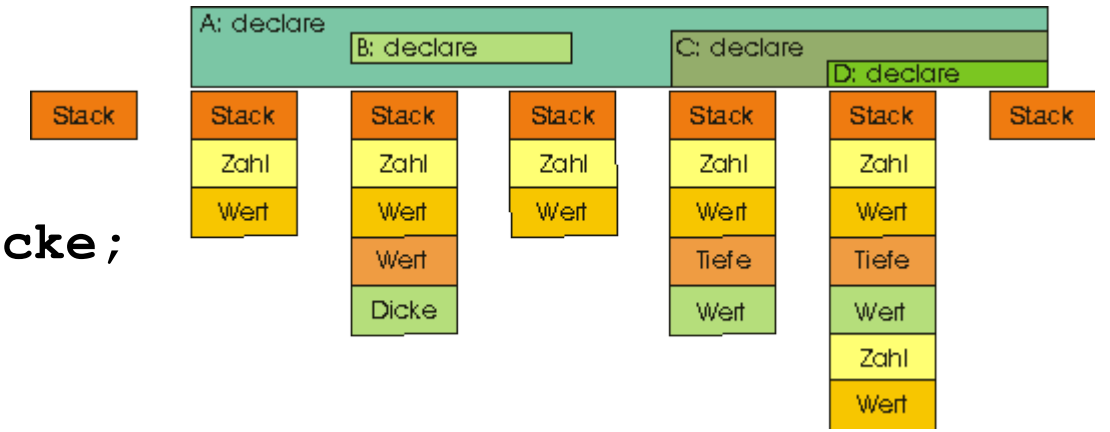
- Blöcke und Umgebungen
  - Darstellungsformen eines Stacks
    - Einfügen der Werte A, B, C in den Stack,
    - entfernen von C aus dem Stack,
    - einfügen von D, E in den Stack,
    - entfernen von E aus dem Stack



Prozeduren und Funktionen

- Blöcke und Umgebungen

```
A: {
      int Zahl, Wert;
        ...
      B: {
          int Wert, Dicke;
          ...
         B: }
      ...
      C: {
           double Tiefe; boolean Wert;
          ...
         D:{
             int Zahl; double Wert;
           ...
         D: }
      C: }
A: };                  // in Java nicht erlaubt! Wert
```



Prozeduren und Funktionen

- Blöcke und Umgebungen in Java
  - Programm soll auf jeweils aktuelle Daten zugreifen
    - muss auf Objekt relativ zum augenblicklichen Zustand des Stacks zugreifen
    - Beispielsweise
      - mit Namen Wert immer erstes Objekt dieses Namens gemeint
      - im Stack, von unten gesehen
    - Zeiger, der immer auf das letzte Element eines Stacks zeigt, wird auch als Stackzeiger (stack pointer) zeichnet.

Prozeduren und Funktionen

- Blöcke und Umgebungen in Java
  - Programm soll auf jeweils aktuelle Daten zugreifen
    - Technik (statisch)
      - im Prinzip über Zeiger und relative Adressierung implementiert
      - Kontrollverbindung
        - Zeiger, verweist in Aktivierungssegment auf vorhergehendes Aktivierungssegment
        - zu Namen kann anhand Kontrollverbindung unterstes Aktivierungssegment gefunden werden, welches diesen Namen enthält.
        - vom Compiler verwendet,
        - muss zu Namen Umgebung der Deklaration dieses Namens

Prozeduren und Funktionen

- Blöcke und Umgebungen in Java
  - Programm soll auf jeweils aktuelle Daten zugreifen
    - Während Programmablaufs meist andere Technik verwendet
      - Feld von Zeigern
        - legen statische Umgebung eines Blocks fest
        - maximale Länge dieses Feldes entspricht maximaler Verschachtelungstiefe der Blöcke in Programm
        - offenbar bereits zur Compilezeit ermittelbar

Prozeduren und Funktionen

- Blöcke und Umgebungen in Java
  - Programm soll auf jeweils aktuelle Daten zugreifen
    - Während Programmablaufs meist andere Technik verwendet
      - wird aktueller Block der Verschachtelungstiefe k betreten
        - Adresse des Aktivierungssegments dieses Blocks als k-tes Element in Feld der statischen Umgebungen geschrieben
        - Compiler hat während Übersetzungszeit Verschachtelungstiefe eines Namens ermittelt
        - einfach über jeweiligen Zeiger auf gesuchtes Aktivierungssegment zugreifen
        - während der Laufzeit wird keine Liste von Aktivierungssegmenten durchsucht
          - wäre zu zeitaufwendig
        - wenn Aktivierungssegmente als Stack organisiert
          - beim Verlassen eines Blocks keine weitere Aktion nötig
          - aktueller Zeiger zeigt automatisch auf direkt umgebenden Block

Prozeduren und Funktionen

- Definition von Prozeduren in Blöcken
  - Prozeduren in innerem Block deklariert
    - meistens nicht im Stack implementiert
    - Bindung des Namens an Prozedur dynamisch
      - muss vom Compiler kontrolliert werden
  - Anweisungsfolgen ohne Objekte an fester Position im Speicher zu ändern
    - Reentrant
      - Anweisungsfolgen verändern keine festen Größen
    - Aktivierungssegmente außer in Blöcken auch in Prozeduren verwendet
      - bei jedem Aufruf einer Prozedur völlig neuer Speicher
      - ohne alten zu verändern oder zu vernichten
      - macht Rekursion (*recursion*) möglich

Prozeduren und Funktionen

- Variable Umgebungen von Blöcken
  - Fortran statische Struktur des Speichers
    - Innerhalb von Subroutinen besitzen Namen eigene Attribute,
    - keine Deklarationen in Blöcken
      - EQUIVALENCE-Anweisung
        - rename in Ada
        - implizit bei der Parameterübergabe
        - schwierig zu verstehen
      - COMMON-Anweisung
        - Umgebung an jede andere Stelle transportieren
        - lokale Umgebung
        - with-Statement in Pascal
        - Präfixklasse in Simula 67

```
simulation
begin
   ...
end
```

Prozeduren und Funktionen

- Lebensdauer und Sichtbarkeit von Namen
  - in Block verdeckt neu deklarierter Name bekannten gleichen Namen
    -
```
declare
  integer Wert := 1;
  do
    real Wert := 2.0;
    -- hier ist Wert == 2.0 vom Typ real:
  done;
  -- hier ist Wert == 1 vom Typ integer:
done;
```

Prozeduren und Funktionen

- Lebensdauer und Sichtbarkeit von Namen
  - ADA
    - ```
      BlockA:
      declare
        Ganz : INTEGER := 1;
        begin
          BlockB:
            declare
            Ganz : INTEGER;
            begin
             Ganz := BlockA . Ganz + 1 ;
            end;
        end;
      ```
  - *Lebensdauer* (*life time*) eines Objekts und *Sichtbarkeit* (*visuality*) eines Objekts müssen nicht identisch sein
      ```

Prozeduren und Funktionen

- Blöcke und Umgebungen in Java
  - Variablen in Blöcken nicht *überschreibbar* (*shadowing*)
    - `int i;`
    - `for(int i; ... ) {}` `// Compiler Error`
    - `{ ... int i; ...}` `// Compiler Error`
  - Variablen in Methoden *überschreibbar*
  - Variablen in inneren Klassen *überschreibbar*
    - `super.Wert // Wert in Basisklasse`
    - `this.Wert  // Wert in dieser Klasse`
    - `Wert       // Wert in Methode`
  - The Java Language Specification, 3rd Edition
    - 14.4.2 Scope of Local Variable Declarations

Prozeduren und Funktionen

- Dynamische Objekte und Garbage
  - Neuere Programmiersprachen
    - Objekte instanzieren
    - über Adressen (Zeiger, *pointer*, *reference*) referenzieren.
      - instanzierte Objekte existieren auch nach Verlassen des Blocks
      - ohne Zeiger Objekt nicht mehr zugreifbar
      - belegt noch Speicherplatz
      - solche Objekte werden als *Garbage* bezeichnet
        - blockiert in der Regel Speicherplatz
        - Speicher muss wieder befreit (*free*, *release*, *dispose*) werden
        - explizite Freigabe durch das Programm
          - fehleranfällig
          - teilweise unmöglich
        - implizite Freigabe durch das Laufzeitsystem
        - automatische Garbageeinsammlung (*automatic garbage collection*)
      - In Java (auch in C#) automatisch

Prozeduren und Funktionen

- Dynamische Objekte und Garbage
  - Neuere Programmiersprachen
    - *Automatic Garbage Gollection*
      - in manchen Programmiersprachen prinzipiell nicht lösbar
      - In C **union**-Struktur

```
union {
   int Zahl;
   int * RefZahl;} Vereinigung;
   Vereinigung.Zahl = 10;
   Vereinigung.RefZahl = new(integer);// kein C
}
```

    - In Sprachen wie C, C++, Pascal, Modula-2 usw. ist automatische Garbage-Collection unmöglich
      - Wird teilweise durch 'raten' gemacht, z.B.
        - kann Zahl überhaupt eine Referenz sein?
        - Liegt Ziel im Heap?
        - Ist Ziel Datensatz?
        - Nicht völlig sicher!