# Imperative Sprachen

## Strukturiertes Programmieren

PS-Hilfsmittel dazu:

1. Strukturierte Programmierung √
2. Abstraktion
3. Module

# Imperative Sprachen

2. PS-Hilfsmittel: Abstraktion
   - Grundsatz: Verstecken von Implementierungsdetails
   - dafür Interface: Schnittstelle zwischen Client und Implementierung
   - 2 Abstraktionsarten:

   Prozedurale Abstraktion:
   - Client: Programmteil, welcher Funktion/Prozedur aufruft
   - Implementierung: Funktions-/Prozedurrumpf
   - Algorithmus ist versteckt (information hiding)
   - Versch. aktuelle Parameter: Abstraktion von speziellen Werten

# Imperative Sprachen

Datenabstraktion:

- Abstrakte Datentypen: Typ + Menge an Operationen
- Abstract data type-Deklarationen in PS
- Gute PS: type checking prüft, ob wirklich nur über Interface-Operationen auf ADT zugegriffen wird

Beispiel: priority-queue (sortierte Reihe),

Interface-Operationen: empty, insert, delmax

interne Datenstruktur (z.B. Array, Suchbaum,…) gekapselt

# Imperative Sprachen

Beispiel: Ada:

```
-- Datei Priority-queue.ads mit ADT-Spezifikation
package Priority-queues is
    type Priority-queue is private;
    function Empty return Priority-queue;
    ...
    private
    ...  -- Deklaration des Datentyps
end Priority-queues;

-------------------------------------------------------
-- Datei Priority-queues.adb mit ADT-Implementierung
package body Priority-queues is
    function Empty return Priority-queue is
        begin
        return null;
    end Empty;
    ...
end Priority-queues;
```

# Imperative Sprachen

## 3. PS-Hilfsmittel: Module

- Modul: Zusammenfassung von Deklarationen
- gutes Modulkonzept: Sichtbarkeit der einzelnen Bestandteile vom Programmierer festlegbar
- Pascal: kein Modulkonzept → Modula (Wirth, Ende 70ger)
- Moduldefinition: legt Konstanten, Typen, Funktionen fest, die andere Module sehen
- Modul-Implementation: implementiert obiges
- Beispiel

```
definition module Stack
    type stack;
    procedure create:stack;
    procedure push (x:integer, var s:stack);
    procedure pop…..
end Stack

implementation module Stack
    type stack = array[1..1000] of integer;
    …
end Stack
```