Imperative Programmierung

Wie können in C zusammengesetzte Datentypen (z.B. Kunde, Stack) und Datenstrukturen erzeugt werden?

- Mit Hilfe von `struct` und einer Typdefinition `typedef`
- Strukturen werden in C als Werte (call by value) an Funktionen übergeben
- Möchte man Strukturen als Referenz übergeben, muss man Zeiger und Pointer verwenden

Anwendungsbeispiel einer Struktur

```c
// Definition
struct point {
    int x;
    int y;
};

// Anwendung
struct point p;
p.x = 100;
p.y = 200;
printf(" p = (%i, %i)", p.x, p.y);
```

Anwendungsbeispiel einer Typdefinition

```c
typedef int people_count;       // create own data type
people_count attendees = 55;    // use own data type
```

Wieso werden Typdefinitionen verwendet?

- Jedes System kann die Datentypen auf seine Art und Weise interpretieren
- Auf gewissen Systemen ist ein int 64Bit und auf anderen nur 32Bit
- Mit eigenen Datentypen lässt sich die Länge fix definieren

Wann kann man `struct` weglassen?

- Wenn man eine Struktur mit einem eigenen Datentyp kombiniert,
- kann man das Schlüsselwort `struct` bei der Deklaration einer Struktur weglassen

```
typedef struct {
    int x;
    int y;
} point_type;

point_type pt; // Hier kommt kein struct mehr
pt.x = 100; pt.y = 200;
```

Für was werden abstrakte Datentypen (ADT) verwenden?

- Um Daten und Operationen miteinander zu verwenden $\rightarrow$ in C
- beschreiben, was die Operationen tun (Semantik)
- **nicht** wie sie es tun sollen (Implementation)
-
- typisches Beispiel: *Stack*

```
// Signatur der Stack-Operationen z.B. im *.h
stack init();                       // returns new empty stack
stack push(element e, stack s);     // adds element to stack
element top(stack s);               // returns top element
stack pop(stack s);                 // removes top element
void print(stack s);                // prints all elements

// Semantik ist aber unabhängig von der Implementation
top(init()) = ERROR
top(push(e, init()) = e
pop(init()) = init()
pop(push(e, s)) = s
```

Wo liegt das Problem bei abstrakten Datentypen?

- ADTs bieten zwar Datenstrukturen und darauf definierte Operationen
- allerdings ist die Implementierung der Datenstruktur getrennt von der Implementierung der dazu gehörigen Funktionen

Welche Art von Programmierung löst das Problem von ADTs?

- Objektorientierte Programmierung, indem sie die Daten und Operationen zusammenbringt (Klassen haben Zustand und Verhalten)
- sozusagen eine Weiterentwicklung von ADTs

Java erweitert C um welche Konzepte in Bezug zu ADTs?

- **Klassen und Instanzen**: `class`, `extends`, `instanceof`, `new`, `this`, `super`
- **Interfaces**: `interface`, `implements`
- **Packages**: `package`, `import`
- **Sichtbarkeit**: `private`, `protected`, `public`
- **Exception-Handling**: `try`, `catch`, `finally`, `throw(s)`