Projektbeschreibung (Beispiel)
1. Aufgabenbeschreibung des Structure-Generator

- Verbunde mit typisierten Komponenten als **C++-Klassen** implementieren
- Eingabe beschreibt mehrere **Verbunde mit ihren Komponenten**
- Ausgabe ist eine Datei mit einer C++-Klassendeklaration für jeden Verbund
- Datenattribute mit **Schreib- und Lese-Methoden** und initialisierender **Konstruktor** werden generiert

- Einsatz des Generators zur Unterstützung der **Software-Entwicklung**
- Generierte Klassen sollen **manuell weiterentwickelt** werden können (Pretty-Printing)
- **Generator soll erweiterbar** sein, z. B. Lesen und Schreiben von Objekten
- Beschreibungssprache soll erlauben, dass Verbunde aus mehreren Dateien **akkumuliert** werden, Duplikate sind erlaubt

© 2003 bei Prof. Dr. Uwe Kastens

2. Ausgabebeispiele

Import extern definierter Verbunde:

Vorwärtsreferenzen:

Klassendeklaration:

Verbundkomponenten:

initialisierender Konstruktor:

set- und get-Methoden für Komponenten:

weitere Klassendeklarationen

```
#include "util.h"

typedef class Customer_Cl *Customer;
typedef class Address_Cl *Address;

class Customer_Cl {
private:
   Address addr_fld;
   int account_fld;
public:
   Customer_Cl (Address addr, int account)
      {addr_fld=addr; account_fld=account; }
   void set_addr (Address addr)
      {addr_fld=addr;}
   Address get_addr ()
      {return addr_fld;}
   void set_account (int account)
      {account_fld=account;}
   int get_account ()
      {return account_fld;}
};

class Address_Cl {
...
```

© 2003 bei Prof. Dr. Uwe Kastens

3. Eingabebeispiele

geschlossene Form:

Folge von Verbundbeschreibungen
bestehend aus
Folge von Komponentenbeschreibungen

```
Customer(  addr:    Address;
           account: int;
        )
Address (  name:   String;
           zip:    int;
           city:   String;
        )
import String from "util.h"
```

Wiederholungen möglich

```
Address (  zip:   int;
           phone: int;
        )
```

offene Form:

Folge von Komponentenbeschreibungen

```
Customer.addr:  Address;
Address.name: String;
Address.zip: int;
import String from "util.h"
```

Wiederholungen möglich

```
Customer.account: int;
Address.zip: int;
```

© 2003 bei Prof. Dr. Uwe Kastens

4. Analyse und Transformation

Syntax:

Je nach Einsatzgebiet soll eine geeignete Notation in geschlossener oder offener Form gewählt werden.

Namen:

Strukturnamen und Komponentennamen dürfen mehrfach definierend auftreten.

Typnamen müssen definierte Strukturen, vordefiniert oder importiert sein.

Eigenschaften:

Die Typen mehrfach definierter Komponenten müssen gleich sein.

Die Komponenten mehrfach definierter Strukturen werden akkumuliert.

Transformation:

Zu mehrfachen Auftreten derselben Struktur oder Komponente wird nur jeweils eine Deklaration erzeugt.

Die Ausgabe soll mit Pretty-Printing Technik formatiert werden.

© 2003 bei Prof. Dr. Uwe Kastens

Arbeitsschritte zur Entwicklung eines Generators

1. Aufgaben definieren
   a. Beschreibung
   b. Ausgabebeispiele
   c. Eingabebeispiele
   d. Aufgaben der Analyse und Transformation
2. Strukturierungsphase
   a. konkrete Syntax
   b. Notation der Grundsymbole
   c. abstrakte Syntax
   d. umfassende Tests
3. Semantische Analyse
   a. fehlerhafte Eingabe durch Testfälle charakterisieren
   b. Namen binden
   c. Eigenschaften berechnen
   d. umfassende Tests
4. Transformation
   a. Ausgabemuster entwerfen
   b. Zieltext erzeugen
   c. umfassende Tests
5. Präsentation des Generators

© 2003 bei Prof. Dr. Uwe Kastens

Projektthemen 2003

	Thema	Bearbeiter
A	**Automaten**	Hagenkötter, Schwindt
B	**MusiXTeX**	Seifert, Schröder, Walther
C	**CoverGen**	Birkenheuer, Keller
D	**StreckGen**	Soltenborn, Thygs
E	**KlausurErgebnisse**	Böning**,** Hollmann
F	**Graphen in LaTeX**	Lindner, Viergutz
G	**Planeten**	Gebel
H	**PersWeb**	Junklewitz, Rödig

© 2003 bei Prof. Dr. Uwe Kastens