Objektorientierte Programmierung
(Winter 2006/2007)

Kapitel 4: Syntaxdiagramme und Grammatikregeln

- Syntaxdiagramme
- Grammatikregeln (kontextfrei)
- Beispiele: Lexikalische Syntax

Syntax-Formalismen (1)

- Ein C++-Programm ist eine Folge von Zeichen, aber nicht jede Folge von Zeichen ist ein C++-Programm.
- Ein Alphabet ist eine endliche, nicht-leere Menge, deren Elemente Zeichen heißen.
- Ein Wort über einem Alphabet ist eine endliche Folge von Zeichen des Alphabets.
- Eine formale Sprache (über einem Alphabet) ist eine (meist unendliche) Menge von Worten (über dem Alphabet).

Syntax-Formalismen (2)

- Es gibt verschiedene Formalismen, um klar und eindeutig eine formale Sprache (Menge von Zeichenfolgen) zu definieren, z.B.
  - ◇ Syntax-Diagramme
  - ◇ (Kontextfreie) Grammatik-Regeln

    BNF (Backus-Naur-Form) ist eine spezielle Syntax für kontextfreie Grammatikregeln (recht verbreitet).
- Syntaxdiagramme und kontextfreie Grammatiken haben die gleiche Ausdruckskraft, d.h. können die gleichen Mengen von Zeichenfolgen beschreiben.

Syntax-Formalismen (3)

- Als professioneller Programmierer sollte man die Definition seiner Sprache lesen können.
- Die Grammatik ist eine Referenz in unklaren Fällen, hilft aber auch zum Verständnis der Sprache.

  Die syntaktischen Kategorien sind oft nützliche Konzepte.
- Es könnte ja sein, daß der Compiler einen Fehler enthält.

  Man sollte sich nicht zum "Sklaven" eines einzelnen Compilers machen. Der Compiler hat nicht immer recht (aber meistens schon). Es gibt ein unabhängiges Gesetz (den ISO-Standard). Zumindest hilft die Aufklärung eines Fehlers, ihn zukünftig zu vermeiden. Herumprobieren ("wie hätte er es denn gern?") ist alleine keine richtige Lösung.

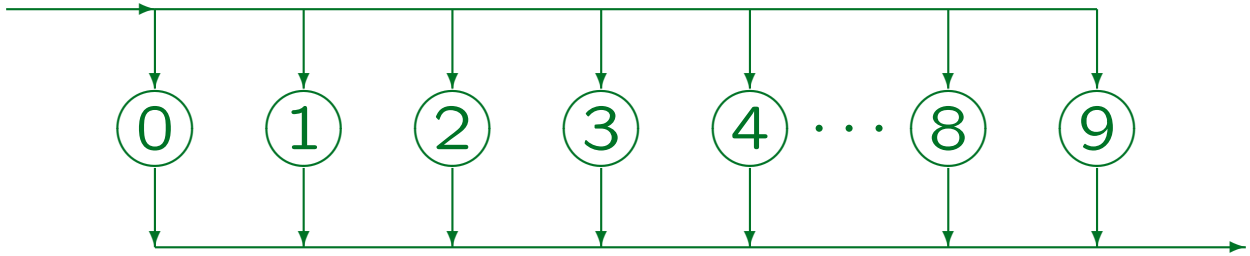
Syntax-Formalismen (4)

- Mit kontextfreien Grammatiken definiert man nur eine Obermenge der C++-Programme, und stellt dann weitere einschränkende Forderungen, z.B.
  - ◇ jede Variable muß vor ihrer Verwendung deklariert sein.
- Kontextfreie Grammatiken (und amit auch Syntaxdiagramme) können solche Zusatzforderungen nicht ausdrücken.

  Die Trennung dieser Aufgaben in die Compiler-Phasen Parser und semantische Analyse ist aber auch für die Modularisierung nützlich.

Syntaxdiagramme (1)

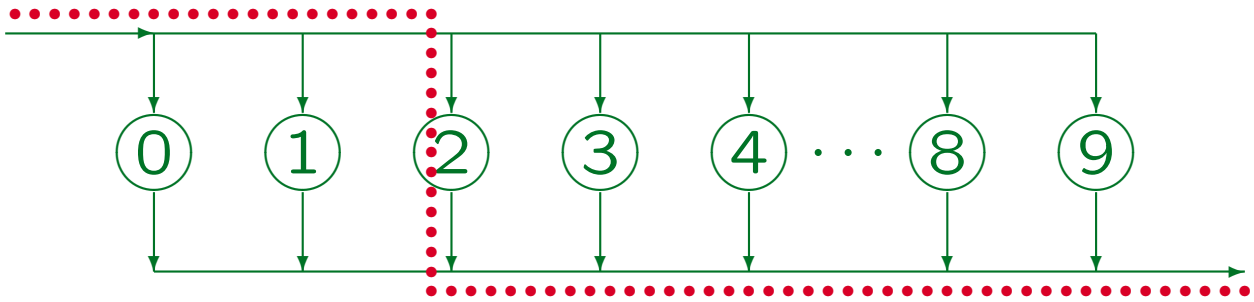
- Digit:



- Ein Syntaxdiagramm besteht aus:
  - ◇ Namen, hier "Digit".
  - ◇ Startknoten: Pfeil von außen in das Diagramm.
  - ◇ Zielknoten: Pfeil, der das Diagramm verlässt.
  - ◇ Kreise/Ovale und Rechtecke, die mit gerichteten Kanten verbunden sind.

Syntaxdiagramme (2)

- Die formale Sprache, die durch dieses Diagramm definiert wird, ist die Menge der Dezimalziffern $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Um zu prüfen, ob eine Eingabe, z.B. "2", zu der Sprache "Digit" gehört, die durch das Digramm definiert wird, muß man einen Weg durch das Diagramm finden, der der Eingabe entspricht:



Syntaxdiagramme (3)

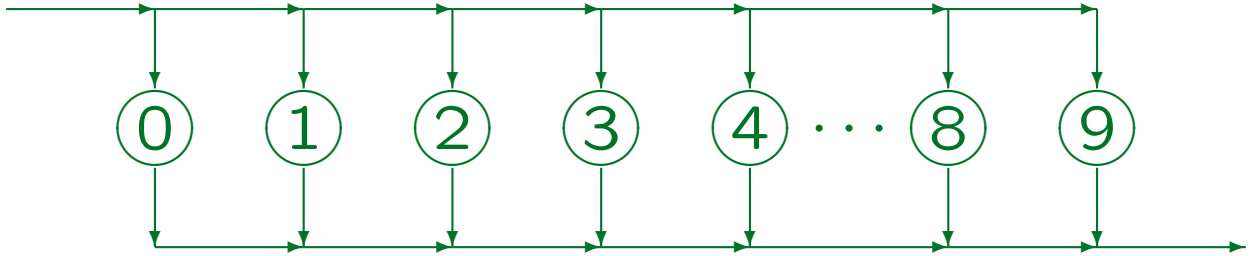
- Solche Diagramme können auf zwei Arten verwendet werden:
  - ◇ Um ein gültiges Wort der Sprache zu erzeugen, folgt man einem Pfad durch das Diagramm vom Start zum Ziel und druckt jedes Symbol in einem Kreis/Oval aus, den man durchläuft.
  - ◇ Um festzustellen, ob eine gegebene Eingabe syntaktisch korrekt ist, muß man einen Pfad durch das Diagramm finden, so daß beim Durchlaufen eines Kreises/Ovals das Zeichen darin das nächste Eingabezeichen ist.

Syntaxdiagramme (4)

- Jede Kante hat nur eine mögliche Richtung.

  Manchmal ist es für den Anfänger etwas schwierig, die Richtung einer Kante zu erkennen.

- Wenn alle Richtungen explizit gemacht sind, sieht das Diagramm so aus:



- Normalerweise wird der Pfeilkopf aber nicht in jedem Segment einer Kante wiederholt.

Syntaxdiagramme (5)

- Es gibt Verzweigungsknoten, an denen man verschiedene ausgehende Kanten benutzen kann.

  Z.B., nach der Eingangskante könnte man nach unten gehen, und die Ziffer 0 ausgeben/einlesen, oder man kann nach rechts gehen, um eine der Ziffern 1 bis 9 zu erhalten.

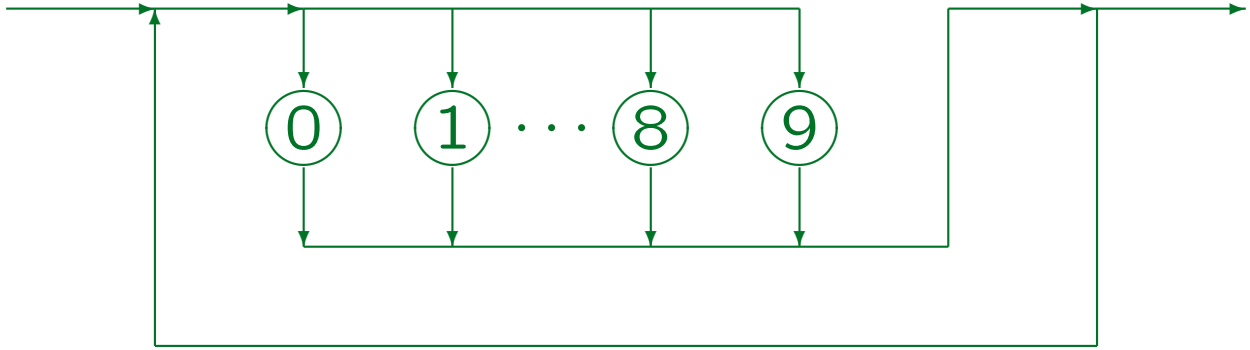
- Um eine gegebene Eingabe zu prüfen, hilft es meist, das nächste Eingabesymbol anzuschauen, um den richtigen Pfad auszuwählen.

  Dies ist, was der Compiler auch macht. Man versucht sicherzustellen, daß an jeder Verzweigung, die Kreise/Ovale, die man in verschiedenen Richtungen erreichen kann, disjunkt sind.

- Gibt es keinen passenden Pfad: Syntaxfehler.

Syntaxdiagramme (6)

- Syntaxdiagramme können Zyklen enthalten (es gibt ja unendlich viele gültige C++-Programme).
- Digit Sequence:



- Aufgabe: Finden Sie einen Pfad durch das Diagramm, um zu zeigen, daß "81" korrekt ist.

  Man darf die gleiche Kante mehrere Male durchlaufen.

Syntaxdiagramme (7)

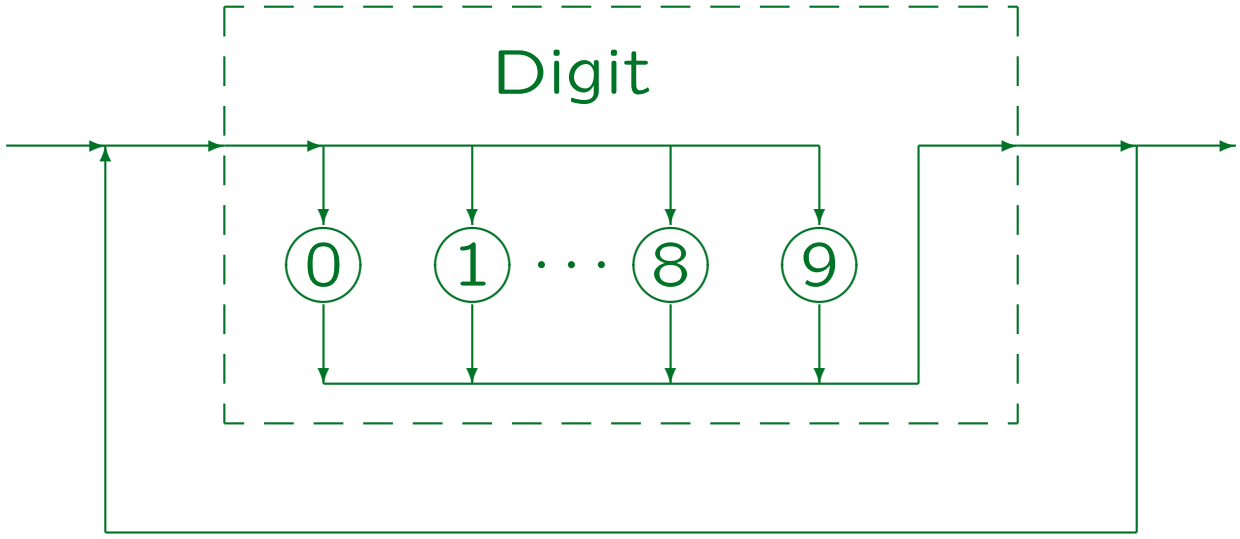
- Man kann an anderer Stelle definierte Diagramme als Module benutzen.
- Dies wird als Rechteck dargestellt, das den Namen des anderen Diagramms enthält:

Digit Sequence:

- Ein Rechteckt steht also für eine syntaktische Kategorie (wie “Verb” und “Objekt”).

Syntaxdiagramme (8)

- Ein Kasten kann durch das Diagramm ersetzt werden, für das er steht (er hat wie das Diagramm eine eingehende und eine ausgehende Kante).
- Digit Sequence:



Syntax Diagrams (9)

- One can view the boxes also like procedures:
  - ◇ When a box is entered, one go to the corresponding diagram,
  - ◇ finds a path through it,
  - ◇ and then returns back to the original diagram where the arrow leaves the box.
- When syntax diagrams are used for analyzing input, each such procedure reads part of the input.

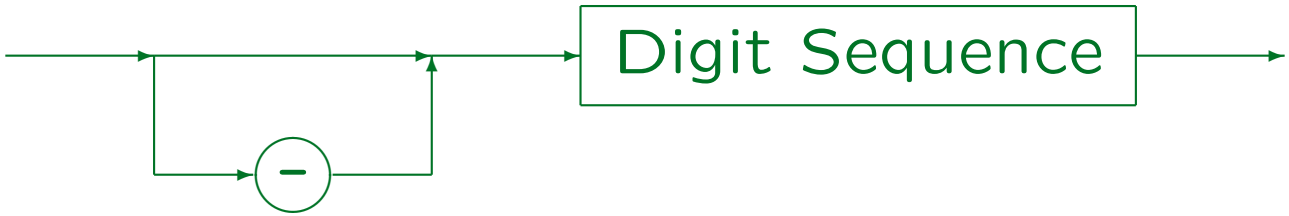
  E.g. when "digit" is called, it expects to see in the input a digit, and it reads this away. Whatever comes after it remains in the input to be analyzed by other procedures.

Syntax Diagrams (10)

- Of course, after some time, one knows what e.g. "Digit" stands for, and does not have to look up the syntax diagram explicitly.
- Each diagram defines a formal language (set of character strings).
- When passing through a box, one can read/print any element of the language defined by the corresponding syntax diagram.

Syntax Diagrams (11)

- Ovals and boxes can be freely mixed in one diagram.
- Integer:



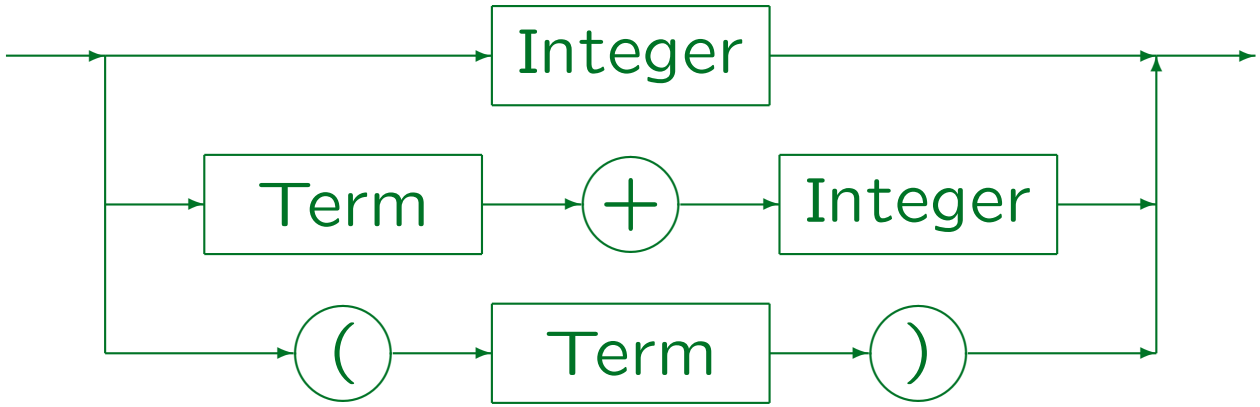
- The diagram "Integer" describes a language containing, e.g., `123`, `-45`, `007`.
- It does not contain, e.g., `+89`, `--5`, `23-42`, `0.56`.

Syntax Diagrams (12)

- Syntax diagrams can be defined recursively, e.g. the diagram for X can contain the a box labelled X.

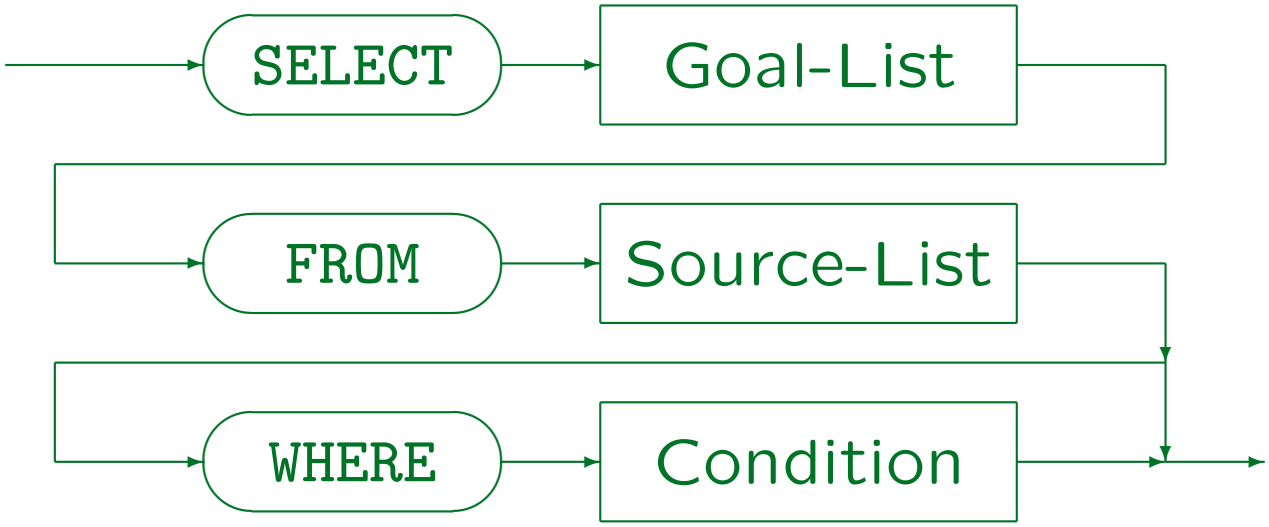
  Mutual recursion is also allowed. With recursion, one cannot expand all boxes beforehand, but one can expand them "on demand" (whenever the box is entered).

- Term:



Syntax Diagrams (13)

- When defining the lexical syntax (possible words or tokens), single characters appear in the ovals.
- Later, words/tokens like "SELECT" are used as the basic building blocks:



Exercise

- Define syntax diagrams for the command language of a text advanture game.
- Typical commands consist of a verb and an object, e.g. "`take lamp`".

  Verbs are e.g. "`take`", "`drop`", "`examine`", "`use`".
  Objects are e.g. "`lamp`", "`sword`", "`rope`".
- One can optionally use an article: "`take the lamp`".
- A verb can be used with multiple objects, separated by "and": "`take the lamp and the rope`".

  This stands for "`take lamp`", "`take rope`".