Einführung in die Informatik, Algorithmen und Datenstrukturen

Teil 1 - Thema 3

Sprachübersetzer und Programmiersprachen

Kommunikation mit dem Computer

Benutzungsschnittstellen?

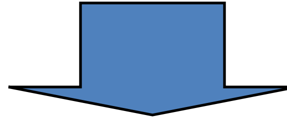
„Die Erkenntnis von der Vergeblichkeit grafischer Benutzungsoberflächen:

Wenn Gott gewollt hätte, dass Computer leicht zu bedienen seien, hätte er den Menschen ein Interface gegeben.“

Murphies Computergesetzte,
Joachim Graf,
Markt&Technik Verlag AG

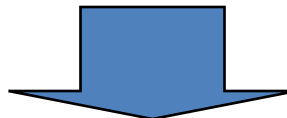
Kriterien zur Auswahl von Programmiersprachen

Welches adäquate Werkzeug kann zur Lösung meines Problems verwendet werden?



Schaffung einer Familie von Programmiersprachen mit klaren semantischen Konzepten für unterschiedliche Anforderungen

Welches Werkzeug muss genommen werden, um alle Probleme zu lösen?



Einbeziehung aller bekannten Konzepte in eine Programmiersprache

Programmiersprachen - Grundbegriffe und Beschreibungsmittel

Begriffe zu Sprachen

SPRACHE:

Eine **Sprache** über einem Alphabet ist eine beliebige Menge von Wörtern über diesem Alphabet. Die Wörter sind dabei Zeichenketten, die durch Aneinanderreihen von Zeichen des Alphabets entstehen.

Die Sprache bezeichnet die wichtigste Kommunikationsform des Menschen. Sie wird akustisch durch Schallwellen oder visuell-räumlich durch Gebärden oder Schrift realisiert. Die Wissenschaft von Sprache als System heißt Allgemeine Sprachwissenschaft. Sprache verfügt über einen Wortschatz, welcher semantische Informationen enthält und eine Grammatik, welche die Wörter in Beziehung zueinander setzt.

/Quelle: www.wikipedia.de/

Begriffe zu Sprachen

LEXIK:

Die **Lexik** bestimmt den Aufbau der Grundelemente der Sprache (Symbole, Worte)

SYNTAX:

Die **Syntax** einer Sprache ist die Menge aller Regeln, nach denen zulässige Sätze in dieser Sprache gebildet werden können. Sie beschreibt die Struktur der Sprache und entspricht damit der Grammatik einer natürlichen Sprache.

Begriffe zu Sprachen

SEMANTIK:

Die **Semantik** ist die Lehre von der Bedeutung einer Sprache. Bei einer Programmiersprache beschreibt die Semantik, was während der Ausführung eines Programms oder eines Programmteils geschieht. Dabei werden auch die Wechselwirkungen berücksichtigt.

PRAGMATIK:

Die **Pragmatik** beschreibt Beziehungen der Sprache zum subjektiven Nutzer. (Lesbarkeit, Erlernbarkeit)

Programmiersprachen

PROGRAMMIERSPRACHE:
Eine **Programmiersprache** ist ein notationelles System zur Beschreibung von Berechnungen in durch Maschinen und Menschen lesbarer Form.

- **Berechnung:** Nachweis der Berechenbarkeit z.B. durch die Turing-Maschine (Churchsche These)
- **Maschinenlesbarkeit:** Es muss eine eindeutige Übersetzung mit einer vertretbaren Komplexität möglich sein.
- **Lesbarkeit durch den Menschen**

Programmiersprachen

Programmiersprachen (Schuldefinition)

Eine Programmiersprache ist eine Sprache zur Formulierung von Algorithmen und Datenstrukturen für die Abarbeitung auf einem Computer.

Je nach dem Grad, mit der die Hardware bei der Programmierung beachtet werden muss, kann man Programmiersprachen in:

- Maschinensprachen,
- Assemblersprachen und
- höhere Programmiersprachen untergliedert.

© Bibliographisches Institut & F.A. Brockhaus AG Mannheim und paetec Gesellschaft für Bildung und Technik mbH Berlin. Alle Rechte vorbehalten. www.schuelerlexikon.de

Informelle Beschreibung einer Anweisung

4.4. ИНСТРУКЦИЯ DO

В примере предыдущего раздела многократное выполнение одной и той же подсекции программы обеспечивается:

1) установкой начального значения некоторого счетчика I = 1;
2) увеличением значения счетчика I = I + 1 каждый раз после выполнения вычисления;
3) использованием инструкции условного перехода для возврата управления в начало подсекции, если вычисления не выполнены указанное число раз.

В Фортране эти три операции могут быть заменены одной инструкцией DO;

DO $n\ i\ = m_1, m_2, m_3$,

где n – метка некоторой исполняемой инструкции, находящейся далее по тексту программы; i – неиндексированная целая переменная, представляющая собой счетчик, получающий приращения (например, I); m_1, m_2 и m_3 –

Informelle Beschreibung von Anweisungen

The form of a DO statement is:

DO <u>s</u> [,] <u>i</u> = $\underline{e}_1$, $\underline{e}_2$ [,$\underline{e}_3$]

where: <u>s</u> is the statement label of an executable statement. The statement identified by <u>s</u>, called the <u>terminal statement</u> of the DO-loop, must follow the DO statement in the sequence of statements within the same program unit as the DO statement.

<u>i</u> is the name of an integer variable, called the <u>DO-variable</u>

$\underline{e}_1$, $\underline{e}_2$, and $\underline{e}_3$ are each an integer constant or integer variable name

The terminal statement of a DO-loop must not be an unconditional GO TO, assigned GO TO, arithmetic IF, block IF, ELSE IF, ELSE, END IF, RETURN, STOP, END, or DO statement. If the terminal statement of a DO-loop is a logical IF statement, it may contain any executable statement except a DO, block IF, ELSE IF, ELSE, END IF, END, or another logical IF statement.

Informelle Beschreibung von Anweisungen

Eine DO-Schleife ist syntaktisch wie folgt definiert:

```
DO_schleife  ::= DO_anweisung
                 DO_bereich

DO_anweisung ::= DO marke laufangabe

laufangabe   ::= laufvariable = anfangsgröße, testgröße,
                               [,schrittweite]

DO_bereich   ::= [ programmteil ]
                 marke { ergibtanweisung |
                 ASSIGN_anweisung
                 CONTINUE_anweisung | CALL_anweisung |
                 logische_IF_anweisung | e_a_anweisung }
```

Grammatiken

„Eine Grammatik $G = (N,T,P,s)$ wird definiert durch

- eine endliche Menge N von Nichtterminalsymbolen oder grammatischen Begriffen,
- eine endliche Menge T von Terminalen, dem Alphabet,
- eine endliche Menge P Produktions- oder Erzeugungsregeln, kurz Regeln, die beschreibt, wie jedes Nichtterminal in Form von Terminalen und Nichtterminalen definiert ist,
- ein ausgewähltes Startsymbol s, wobei $s \in N$."

/HORN03, S. 263/

Kontextfreie Grammatiken

„Eine kontextfreie Grammatik besteht aus einer Menge von grammatischen Regeln: Diese Regeln bestehen aus einer linken Seite, aus dem Metasymbol “::=“ gefolgt von einer rechten Seite, die aus einer Folge von Elementen besteht und Symbole und andere Strukturnamen umfasst“

- Die Namen für Strukturen werden als **Nichtterminalsymbole** bezeichnet, da sie in weitere Strukturen zerlegt werden können.
- Token werden als **Terminalsymbole** bezeichnet.
- Grammatikregeln werden als **Produktionen** bezeichnet.

BNF - Backus - Naur - Form

BNF - Backus - Naur - Form(alismus)

verwendete Zeichen:

::= trennt linke und rechte Seite einer syntaktischen Regel

<> zum Einschluß von Nichtterminalsymbolen

| zur Darstellung von alternativen auf der rechten Seite einer syntaktischen Regel

```
Bsp.: <ziffer> ::= 0|1|2|3|4|5|6|7|8|9
      <buchstabe> ::= a|b|c|...|z|
      <name> ::= <buchstabe>|<name><buchstabe>|
                 <name><ziffer>

      <programm> ::= <kopf> <block> <ende>
      <anweisungsteil> ::= BEGIN <anweisungen>  END
```

EBNF - erweiterte Backus - Naur - Form

EBNF - erweiterte Backus - Naur - Form(alismus)

zusätzlich verwendete Zeichen:

[] zur Kennzeichnung von optional (wahlweise) angebbaren syntaktischen Strukturen

{} zur Kennzeichnung der Wiederholung

() zur Bildung von Teilausdrücken auf der rechten Seite einer syntaktischen Regel

```
<if-anweisung> ::= IF <bedingung> THEN <anweisung>
                   [ ELSE <anweisung> ]
```

Syntaxdiagramm

Ein Syntaxdiagramm ist ein knotenmarkierter, gerichteter Graph.

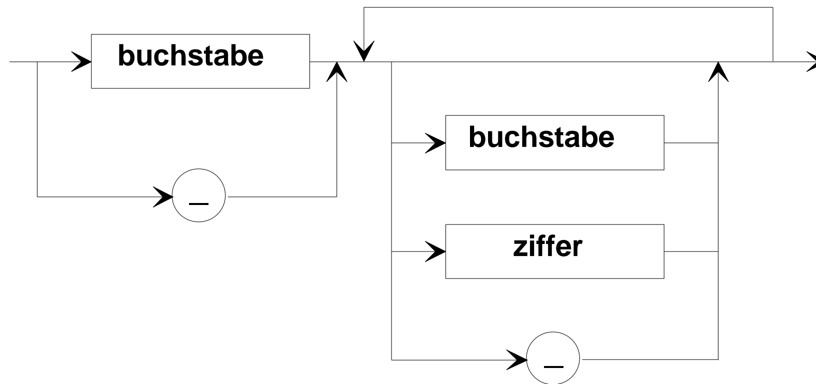
Die Markierungen von Knoten, die durch Rechtecke dargestellt sind, sind Nichtterminalsymbole.

Die Markierungen von Knoten, die durch Ovale oder Kreise dargestellt sind, sind Terminalsymbole.

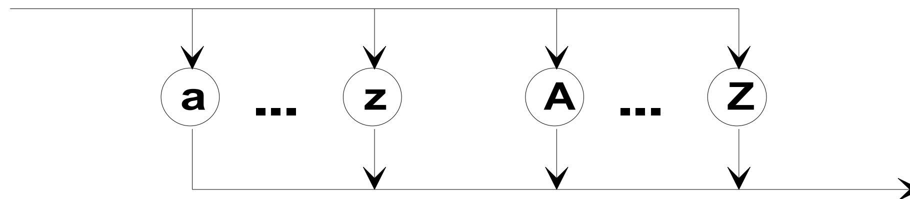
Ein Syntaxdiagramm ist vollständig, wenn jedes Syntaxdiagramm ausschließlich durch Terminalsymbole beschrieben wird.

Beispiele für Syntaxdiagramme

Bezeichner:



Buchstabe:



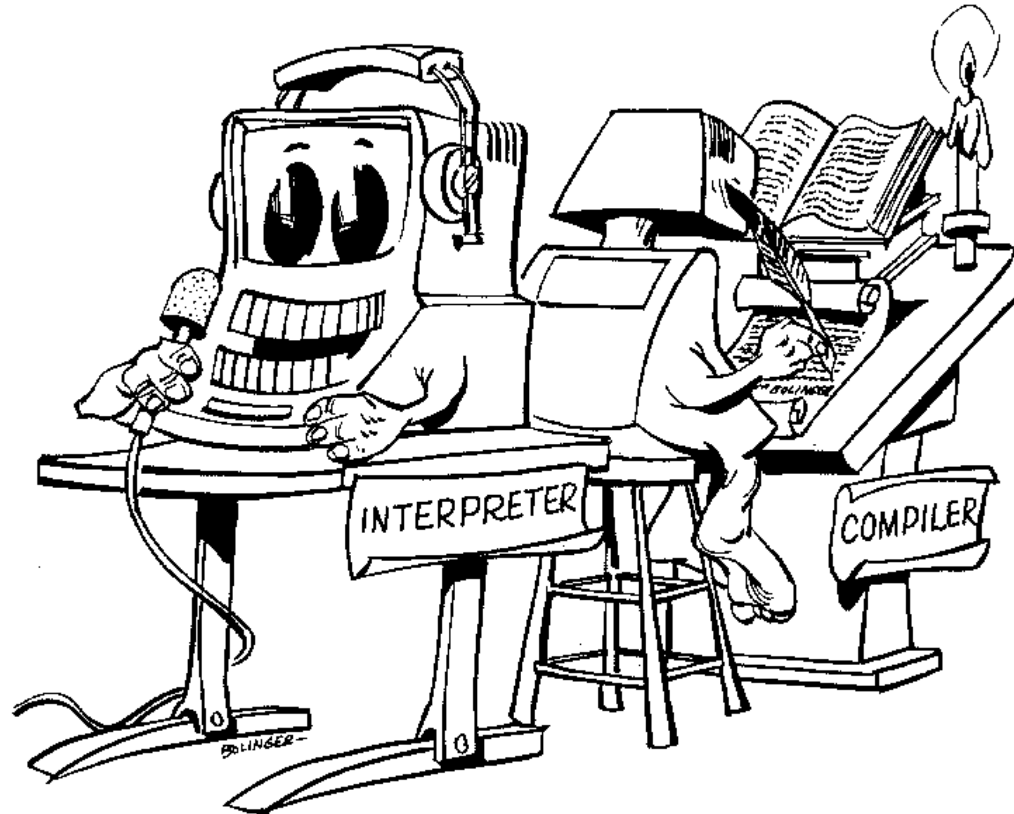
Programmiersprachen – Sprachübersetzung und -verarbeitung

Grundwerkzeuge der Softwareentwicklung

Computer können nur Programme ausführen, die in Maschinensprache vorliegen. Symbolische oder höhere Programmiersprachen müssen durch spezielle Übersetzungsprogramme, Assembler, Interpreter oder Compiler auf das **ausführbare Maschinensprachniveau transformiert** werden.

Der Übersetzungsvorgang ist komplex und kann in zwei Teile, den Analyse- und den Syntheseteil zerlegt werden.

Sprachübersetzung



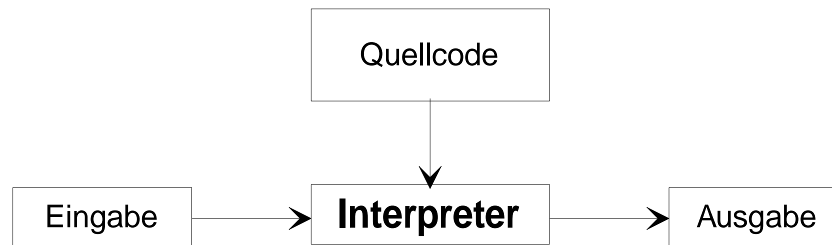
Quelle: Peter Norton, MS-DOS und PC-DOS, 1985

Arbeitsweise eines Interpreters

Algorithmus zur Arbeitsweise eines Interpreters:
Beginne mit dem Anfang des Programms
Wiederhole
    Analysiere die Syntax der nächsten Anweisung,
    um ihren Typ und ihre Operanden zu bestimmen
    Falls kein Syntaxfehler vorliegt
        dann rufe den Modul für diesen
        Anweisungstyp mit den Operanden als
        Parameter auf
        sonst melde einen Fehler
Bis das Ende des Programms erreicht ist oder ein
Syntaxfehler auftritt

Sprachübersetzung durch Interpreter

Die Interpretation ist ein **EINSTUFENPROZESS**, in dem sowohl das Programm als auch die Eingabe an den Interpreter gegeben und die Ausgabe erhalten wird.



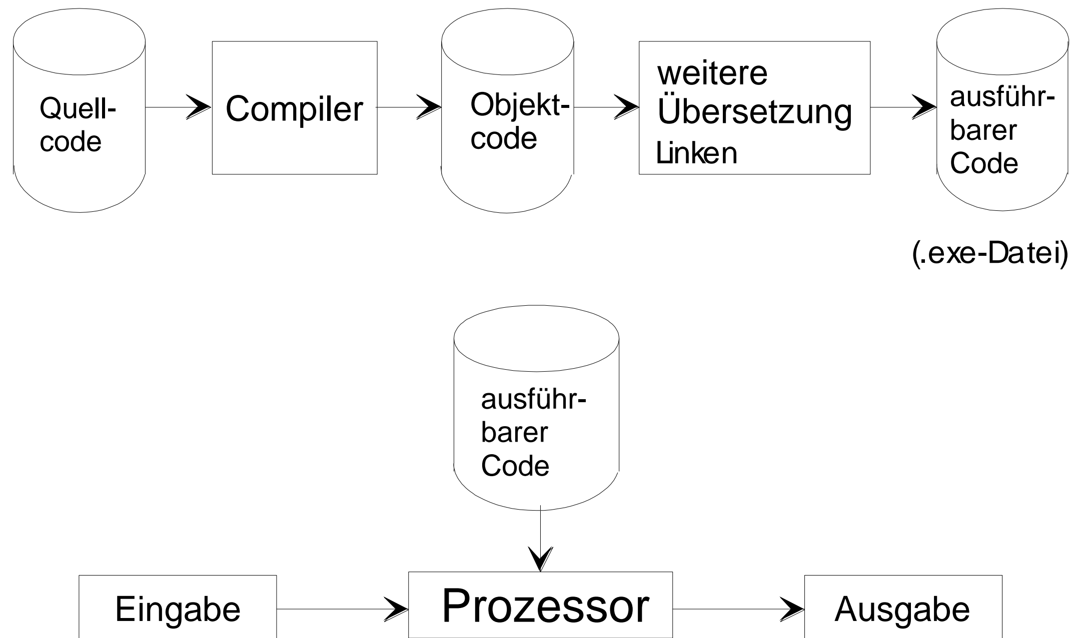
Arbeitsweise eines Compilers

Algorithmus zur Arbeitsweise eines Compilers:

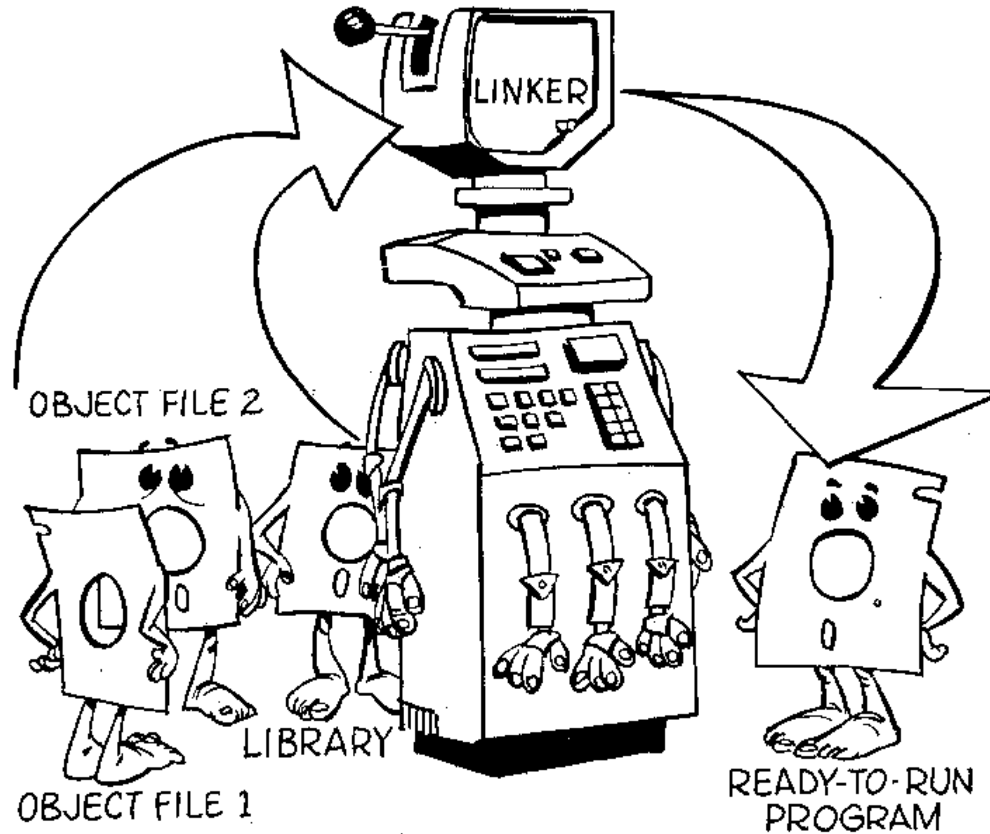
Beginne mit dem Anfang des Programms
Wiederhole
    Übersetze nächste Anweisung
Bis das Ende des Programms erreicht ist
Treffe Vorkehrungen zur Ausführung des übersetzten Programms

Sprachübersetzung durch Compiler

Das Quellprogramm wird in den Compiler eingegeben. Dieser erzeugt daraus ein Zielprogramm. Das Zielprogramm ist ein in Maschinensprache geschriebenes Programm und wird auch als Assemblerprogramm bezeichnet. Dieses Programm wird von einem **Assembler** oder **Assemblierer** in ein **Objektprogramm** übersetzt.



Linken von Programmen



Quelle: Peter Norton, MS-DOS und PC-DOS, 1985

Lexikalische Analyse

Ziel: Analyse, Aufbereitung und Kontrolle des Quellprogramms

1. Es wird für jedes Zeichen bestimmt, ob es ein terminales Symbol ist, oder ob es in Verbindung mit seinen Nachbarsymbolen zu einem terminalen Symbol zusammengesetzt werden kann.
2. Syntaktisch überflüssige Zeichen, wie mehrfache Zwischenräume oder Kommentare werden entfernt.

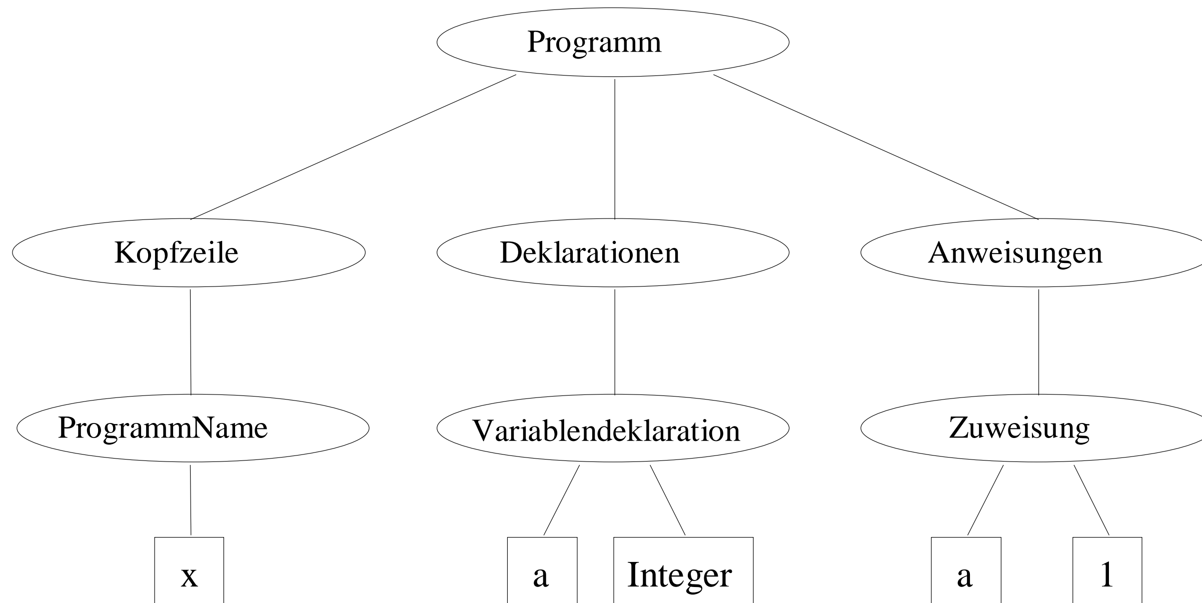
Lexikalische Analyse

3. Nicht vorgesehene Zeichen im Quellprogramm werden als Fehler zurückgewiesen.
4. Terminale Symbole werden zum Syntaxanalysator in Form von Tokens weitergegeben. Diese Tokens bestehen aus den Komponenten Typ und Zahl. Der Typ gibt an, welche Art eines terminalen Symbols der Token enthält, der Wert gibt an, welchen Wert das terminale Symbol enthält. Hat einen Token keinen Wert, so ist dieser undefiniert.

Syntaxanalyse

Ziel: Aufbau des Parsebaumes aus dem Programmtext heraus

```
PROGRAM x;
VAR a : integer;
  BEGIN
    a:= 1;
END.
```



Behandlung von Syntaxfehlern

Ziel:

Auffinden von Syntaxfehlern, diese melden und anschließend die Analyse fortsetzen (Wiederaufsatz).

Gängige Verfahren:

Panik-Modus: Nach Meldung des ersten Fehlers wird die Syntaxanalyse abgebrochen.

Wiederaufsatz mit allgemeinen Fangsymbolen: Tritt ein Syntaxfehler auf, wird der Symbolstrom so lange überlesen, bis ein Symbol auftritt, das an der Fehlerstelle erwartet wird oder das ein Nachfolger eines in Arbeit befindlichen Non-Terminalsymbols ist.

Semantische Analyse

Ziel: Ermittlung der Bedeutung des Quellprogramms und Erkennung von semantischen Fehlern

Erzeugung dreier Strukturen:

- binärer Suchbaum zur Verwaltung der Bezeichner
- Liste von Anweisungsobjekten zur Darstellung der Anweisungen des Anweisungsteils
- jeweils ein Baum zur Darstellung von Ausdrücken (auf diese Bäume wird durch Zeiger verwiesen, die in den Anweisungsobjekten enthalten sind)

Semantische Analyse

Semantische Fehler können entweder statisch sein (z.B. unverträgliche Datentypen oder nicht deklarierte Variable) oder sie können dynamisch sein (z.B. Bereichsüberschreitungen oder Division durch Null). In der Regel werden Syntaxfehler und statische semantische Fehler vom Parser entdeckt.

Codeerzeugung

Ziele:

• den vom Programm zu bearbeitenden Datenelementen wird ein Hauptspeicherplatz zugewiesen und

• für jedes syntaktische Element sind in geeigneter Weise maschinensprachliche Anweisungen zu erzeugen

Codeerzeugung

1. Stufe - Aufbau der Symboltabelle:

ein Eintrag in die Symboltabelle enthält:

- Name
- Typ (lt. Deklaration)
- Hauptspeicheradresse

2. Stufe - Erzeugung des Maschinencodes

Erzeugung der Maschinensprach-Anweisung für jedes syntaktische Element durch den Codegenerator

Kontrollfragen

1. Erklären Sie die Begriffe „Sprache“ und „Programmiersprache“. Gehen Sie dabei auf die Lexik, Syntax und Semantik einer Sprache ein. Nennen Sie zwei Beschreibungsmittel für Programmiersprachen.
2. Erläutern Sie 2 Beschreibungsmittel für Programmiersprachen. Geben Sie mit Hilfe dieser Beschreibungsmittel eine exakte Definition für einen Namen (Bezeichner) in Object-Pascal.
3. Beschreiben Sie die Arbeitsweise eines Interpreters. Nennen sie eine reale Programmiersprache als Beispiel.
4. Erläutern Sie die Phasen der Programmübersetzung durch einen Compiler. Nennen sie eine reale Programmiersprache als Beispiel.
5. Beschreiben Sie die Stufen der lexikalischen Analyse, der Syntaxanalyse und der semantischen Analyse bei der Programmübersetzung. Gehen Sie auf die Fehlerbehandlung ein.

Literatur:

/DUDEN03/	Basiswissen Schule – Informatik Abitur PAETEC Verlag für Bildungsmedien Berlin, 2003 ISBN 3-89818-065-4
/HENNING07/	Henning, Peter A., Holger Vogelsang Taschenbuch Programmiersprachen Fachbuchverlag Leipzig, 2007; ISBN 978-3-446-40744-2
/HORN03/	Horn, Christian; Immo O. Kerner; Forbig, Peter Lehr- und Übungsbuch Informatik Band 1: Grundlagen und Überblick Fachbuchverlag Leipzig; 2003; ISBN 3-446-22543-9
/FORBIG06/	Forbig, Peter; Immo O. Kerner Lehr- und Übungsbuch Informatik Programmierung – Paradigmen und Konzepte Fachbuchverlag Leipzig; 2006; ISBN 3-446-40301-9
/LOUDEN94/	Louden, Kenneth C. Programmiersprachen Thomson Publishing International Bonn, 1994; ISBN 3-929821-03-6