Aufgabe 1 (Operator-Vorrang-Analyse) ***20 Punkte***

Gegeben sei die Grammatik $G = (\{S\}, \{+, \mathbf{f(}, \mathbf{)}, \mathbf{1/}, \mathbf{x}\}, P, S)$ mit

$P = \{\ S \rightarrow S + S,$

$S \rightarrow \mathbf{f(}S\mathbf{)},$

$S \rightarrow \mathbf{1/}S,$

$S \rightarrow \mathbf{x}\ \}.$

(a) Die Wörter aus $L(G)$ sind geschachtelte Funktionsausdrücke. Der Operator „**1/**“ ist hier rechtsassoziativ. Konstruieren Sie die Tabelle für eine Operator-Vorrang-Analyse zu G. *12 Punkte*

(b) Parsen Sie mit Hilfe der in (a) definierten Tabelle die Folge *8 Punkte*

f(1/f(x)) + f(x).

Orientieren Sie sich bei der Darstellung an den Beispielen im Kurs.

Aufgabe 2 (LR(1)-Analyse) ***30 Punkte***

Gegeben sei die Grammatik $G = (N, \Sigma, P, S)$ mit

$N = \{S, X, Y, Z\}$,

$\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$,

$P = \{\ S \rightarrow X\mathbf{a}$

$X \rightarrow X\mathbf{b} \mid \mathbf{b}Y\mathbf{a}$

$Y \rightarrow \mathbf{a}X\mathbf{c} \mid \mathbf{a} \mid \mathbf{a}XZ$

$Z \rightarrow \mathbf{cc} \mid \varepsilon\}$.

(a) Geben Sie die erweiterte Grammatik G’ zu G an und berechnen Sie die kanonische LR(1)-Kollektion für G’. Nummerieren Sie die Produktionen von G’. *15 Punkte*

(b) Berechnen Sie die Steuertabelle für G’. *8 Punkte*

(c) Analysieren Sie das Wort **babaaaba**. Geben Sie dazu für jeden Schritt den Inhalt des Stacks, die noch nicht verarbeitete Eingabe und die durchgeführte Aktion an. *7 Punkte*

50 Punkte

Aufgabe 3 (myPS - Teil 3)

Nachdem wir bereits in der Lage sind, *myPS*-Programme syntaktisch zu analysieren, werden nun Übersetzungsregeln benötigt, die Anweisungen unserer Sprache in Anweisungsfolgen der Sprache Postscript übersetzen. Dazu benötigt man natürlich zunächst einige Grundkenntnisse über die Zielsprache.

Postscript wurde Anfang der 80er Jahre von der Firma Adobe-Systems entwickelt. Es ist eine Sprache, die Grafiken und Texte unabhängig von den technischen Gegebenheiten eines Ausgabegerätes beschreibt. Außerdem ist es damit möglich, Zeichnungen und Schriften frei zu skalieren. Postscript-Code wird in Postfix-Notation angegeben und ein Postscript-Interpreter führt eine Stackverarbeitung der eingegebenen Operationen durch. Postscript wird häufig als Druckformat verwendet, es gibt jedoch auch viele Anwendungsprogramme, die Postscript-Dateien direkt verarbeiten bzw. erzeugen. Eine simple Postscript-Datei sieht z.B. folgendermaßen aus:

```
%!PS-Adobe
%%Creator: Kursbetreuer 1810
%%Title: Simple Example
%%EndComments

10 10 moveto
100 100 lineto 10 100 lineto 10 10 lineto stroke
showpage
```

Die Kopfzeilen sind Konvention; es gehört zum guten Stil, diese einzuhalten. Mit der Operation `moveto` wird ein Anfangspunkt und mittels der Operation `lineto` wird eine Verbindungslinie zwischen Punkten definiert. Eine Folge von solchen Punkt- und Verbindungsdefinitionen wird als Pfad bezeichnet. Weiterhin gibt es Befehle, die auf Pfaden operieren, z.B. zeichnet `stroke` eine Verbindungslinie zwischen den Punkten. Schließlich beendet der Befehl `showpage` die Definition der Seite und führt alle zuvor beschriebenen Zeichnungen aus. Der Punkt (0, 0) definiert die linke, untere Ecke der Zeichenfläche und die Einheit des Koordinatensystems ist ein Point (pt), es gilt: 72 pt = 1 inch = 25.4 mm.

Statt der Definition über einzelne Punkte können auch kurvige Pfade, z.B. Kreisbögen, definiert werden, beispielsweise zeichnen die Anweisungen

```
1 0 0 setrgbcolor
100 100 50 0 360 arc fill
```

eine rote Kreisfläche. Der Operator `setrgbcolor` ändert die aktuelle Zeichenfarbe und der Befehl `arc` definiert einen Pfad, der einen Kreisbogen beschreibt. Schließlich wird mittels `fill` die durch den Pfad eingeschlossene Fläche ausgefüllt. Detaillierte Informationen über alle Operationen und Sprachkonzepte findet man im

Buch „Postscript Language Reference, Third Edition“, dies ist auch als PDF-Datei verfügbar unter der URL www.adobe.com/products/postscript/pdfs/PLRM.pdf.

Nun sollten Sie zunächst mit Postscript experimentieren, um Übersetzungen für die in Teil 1 unseres Projektes geforderten Sprachelemente zu finden. Um dies zu tun, benötigt man einen Postscript-Interpreter und ein Anzeigeprogramm (Viewer). Glücklicherweise gibt es eine freie Implementierung namens Ghostscript, z.B. unter http://www.cs.wisc.edu/~ghost.

Aufgabenstellung

Erweitern Sie die Implementierung des Übersetzers durch Übersetzungsregeln. Die Regeln für `union`, `concat`, `plot` und `scaletobox` sind recht kompliziert und sollen zunächst nicht implementiert werden. Generieren Sie für diese Regeln eine entsprechende Fehlermeldung und beenden Sie die Übersetzung, falls versucht wird, diese Operatoren zu verwenden. Variablen sollen bereits definiert und benutzt werden können. Es findet jedoch noch keine Typprüfung (typkonforme Wertzuweisung) bzw. Konsistenzprüfung (undefinierte Variablen) statt.

Hinweise

Es wird eine Hilfsdatenstruktur zur Verwaltung der in einem Knoten des Ableitungsbaumes gespeicherten Informationen benötigt. Diese kann als C++-Struktur (wir nennen sie *ComplexNode*), die vorerst lediglich ein Attribut *code* vom C++-Typ *string* besitzt, implementiert werden. Damit sowohl Parser als auch Scanner diese Struktur verwenden können, muss deren Deklaration in eine externe Datei ausgelagert werden.

Im Scanner müssen manchen Token nun Attributwerte zugeordnet werden. Dies geschieht über die Variable `yylval`, deren Typ in der Parserspezifikation (also in `myps.y`) folgendermaßen festgelegt wurde:

```
%union {
  struct ComplexNode* content;
}
```

Momentan besitzt die Vereinigung aller möglichen Attributwerte - ein Union ist ein Record, in dem sich die verschiedenen Komponenten gegenseitig ausschließen - nur ein Attribut. Wird nun zum Beispiel ein gültiges Fließkommamuster erkannt, so wird im Programmcode des Scanners dem Token `ZZREAL` über die folgenden Codezeilen ein Wert zugeordnet.

```
{num}           { LINEINFO;
                  yylval.content = new ComplexNode(yytext);
```

```
        return (ZZREAL);
      }
```

`LINEINFO` ist hierbei ein Macro, das wie folgt definiert wurde:

```
#define LINEINFO {cols += yylength; lineStr += yytext; }
```

Die dort geänderten Werte `cols` und `lineStr` werden bei einer evtl. Fehlerausgabe verwendet.

Für die Token bzw. Nichtterminale, die Attributwerte benötigen, kann man nun in der Parserspezifikation gleich den passenden Attributnamen und damit auch den Typ festlegen, z.B.:

```
%token <content> ZZREAL
%type <content> numexpr
```

Dies hat den Vorteil, dass beim Programmieren von Übersetzungsregeln die vom Parser bereitgestellten Markierungssymbole $$ bzw. $1 ... $*n* wie Variablen vom entsprechenden Typ behandelt werden können, z.B.:

```
numexpr : numexpr '-' numterm {
        $$ = new ComplexNode($1->code+" "+$3->code+" sub ");
        delete $1; delete $3;
}
```

Zum Abschluss erläutern wir noch ein Beispiel zum Zeichnen einer Ellipse:

```
/savematrix matrix currentmatrix def
100 300 translate
30 50 scale
0 0 1 0 360 arc
savematrix setmatrix
stroke
```

Zunächst wird die aktuelle Transformationsmatrix des Koordinatensystems unter dem Namen `savematrix` zwischengespeichert. Danach wird der Ursprung in den Punkt (100, 300) verschoben und die Koordinatenachsen werden unterschiedlich skaliert. Als nächstes wird ein Kreis um den Nullpunkt mit Radius $r = 1$ definiert. Vor dem Zeichenbefehl wird die Transformationsmatrix wieder zurückgesetzt, damit die zu zeichnende Linie nicht verzerrt wird.