Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Christoph Kuhmünch, Gerald Kühne

Praktische Informatik II
SS 2000

Übungsblatt 10 **Ausgabe: Mi, 18.07.00** **Abgabe: Di, 26.07.00**

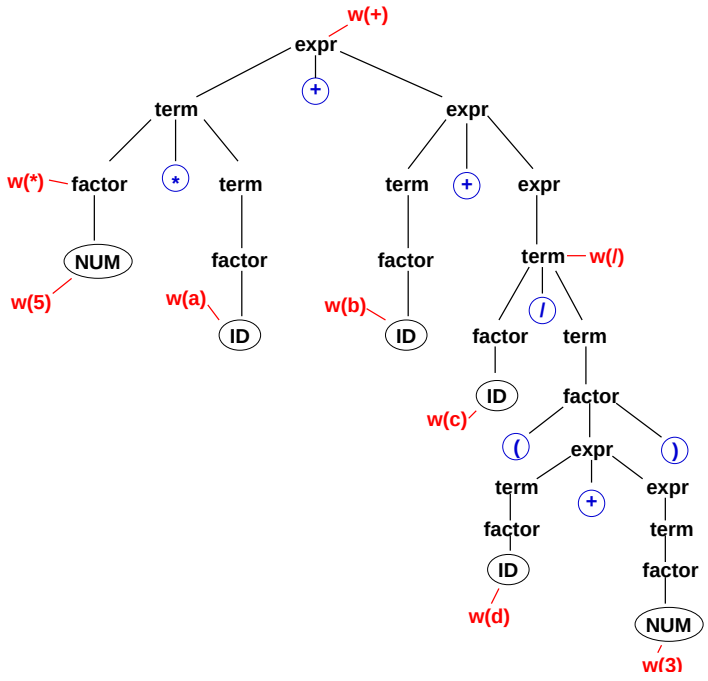
Aufgabe 1: Parsing [10 Punkte]

Gegeben ist folgende Grammatik für einfache arithmetische Ausdrücke in Infixnotation:

$$\begin{aligned} expr &::= term \mid term + expr \mid term - expr \\ term &::= factor \mid factor * term \mid factor / term \\ factor &::= (expr) \mid NUM \mid ID \\ NUM &::= '0' \mid .. \mid '9' \\ ID &::= 'a' \mid .. \mid 'z' \end{aligned}$$

Dabei ist *expr* das Startsymbol. Als Beispiel dient der Ausdruck $5 * a + b + c/(d + 3)$

(a) [4 Punkte] Zeichnen Sie den Parsetree.



(b) [1 Punkt] Ist die Grammatik eindeutig?

Ja, weil eindeutig feststeht, welches Terminal- / Nicht-Terminal-Symbol als "nächstes" zu verwenden ist.

(c) [2 Punkte] Ergänzen Sie den Parsetree um semantische Aktionen zur Erzeugung eines Postfix–Ausdrucks. Wie sieht der von ihrem Parsetree erzeugte Postfix–Ausdruck aus?

Zunächst wird an jedes Terminalzeichen ein "'write"'-Befehl angehängt. Im Baum werden dann zuerst der linke, dann der rechte Zweig und zuletzt der Knoten selbst ausgegeben (siehe rote Markierung in der Abbildung oben).

Postfix–Ausdruck: $5a*bcd3+/++$

(d) [3 Punkte] Zu dieser Grammatik soll nun ein Parser in C entwickelt werden. Da die Symbole nur ein Zeichen lang sind, ist keine lexikalische Analyse notwendig. Als Ausgabe soll ein äquivalenter Ausdruck in Postfix-Form generiert werden.

```
#include<stdio.h>
#include<stdlib.h>

/*
  Uebungsblatt 10, Aufgabe 1d)

  Umwandeln von arithmetischen Ausdruecken (Infixnotation -> Postfixnotation)

  Grammatik siehe Uebungsblatt 10

  HINWEISE: - Die Eingabe erfolgt ueber die Standardeingabe
            - Es sind keine Leerzeichen in der Eingabe erlaubt
*/


/* Vorwaertsdeklarationen */
void error(char* message);
void print(char c);
void expr(FILE* input);
void term(FILE* input);
void factor(FILE* input);
int  NUM(char);
int  ID(char);


void error(char* message)
{
    fprintf(stderr,"\nFehler: %s\n", message);
    exit(1);
}


void print(char c)
{
    putchar(c);
}



/* expr ::= term | term + expr | term - expr */

void expr(FILE* input)
{
    char c;

    term(input);                /* am Anfang steht immer ein term */
    c = getc(input);            /* Folgezeichen einlesen */

    if (c == '+' || c == '-') {
        expr(input);            /* bei + oder - rekursiver Aufruf */
        print(c);             /* Operator (+ oder -) ausgeben */
    } else {
        ungetc(c, input);       /* ansonsten Zeichen zurueck in den Eingabe-
strom */
    }
}
```

```
/* term ::= factor | factor * term | factor / term */

void term(FILE* input)
{
    char c;

    factor(input);               /* am Anfang steht immer ein factor */
    c = getc(input);             /* Folgezeichen einlesen */
    if (c == '*' || c == '/') {
        term(input);             /* bei * oder / rekursiver Aufruf */
        print(c);              /* Operator (* oder /) ausgeben */
    } else {
        ungetc(c, input);        /* ansonsten Zeichen zurueck in den Eingabe-
strom */
    }
}


/* factor ::= (expr) | NUM | ID */

void factor(FILE* input)
{
    char c;

    c = getc(input);              /* Folgezeichen einlesen */
    if (c == '(') {               /* ist es ein Klammerausdruck? */
        expr(input);              /* ja => Ausdruck auswerten */
        if (getc(input) != ')') { /* auf schliessende Klammer pruefen */
            error("')' erwartet");
        }
    } else if (NUM(c) || ID(c)) { /* es muss eine Zahl oder ein Identifier */
        print(c);                 /* angegegeben werden */
    } else {
        error ("Zahl (0..9) oder Identifier (a..z) erwartet");
    }
}


/* NUM ::= '0' | .. | '9' */

int NUM(char c)
{
    return (c >= '0' && c <= '9');
}


/* ID ::= 'a' | .. | 'z' */

int ID(char c)
{
    return (c >= 'a' && c <= 'z');
}

int main()
{
    char c;

    FILE* input = stdin;     /* Standardeingabe verwenden */

    printf ("Infix -> Postfix\n");
```

```
        printf ("Ausdruck eingeben (ohne Leerzeichen): ");

        expr(input);             /* Ausdruck auswerten */

        if(getc(input) != '\n')  /* sind alle Zeichen bearbeitet? */
            error ("Ungueltiges Zeichen in der Eingabe");

        return 0;
    }
```

Aufgabe 2: Codeerzeugung [10 Punkte]

Der Parser aus Aufgabe 2d) soll nun so modifiziert werden, daß er Code für den 68000-Assembler erzeugt. Die arithmetischen Ausdrücke sollen allerdings keine Variablen mehr enthalten (d.h. es sind nur noch Ziffernkonstanten zulässig). Ändern Sie die Ausgaberoutine aus Aufgabe 2d) entsprechend ab.

```
#include<stdio.h>
#include<stdlib.h>

/*
  Uebungsblatt 10, Aufgabe 2)

  Umwandeln von arithmetischen Ausdruecken in 68000-Assembler

  Grammatik siehe Uebungsblatt 10

  ANALOG ZU Aufgabe 1d):
            - Es wird eine andere print-Routine verwendet
            - In der main-Routine wurde ein printf hinzugefuegt

  HINWEISE: - Die Eingabe erfolgt ueber die Standardeingabe
            - Es sind keine Leerzeichen in der Eingabe erlaubt
            - Im Gegensatz zur Aufgabe 1d) sind keine Identifier (a..z) erlaubt
*/


/* Vorwaertsdeklarationen */
void error(char* message);
void print(char c);
void expr(FILE* input);
void term(FILE* input);
void factor(FILE* input);
int  NUM(char);
int  ID(char);


void error(char* message)
{
    fprintf(stderr,"\nFehler: %s\n", message);
    exit(1);
}


void print(char c)
{
    switch (c) {
    case '+':                       /* arithmetische Operationen */
    case '-':
```

```
    case '*':
    case '/':
        printf ("move (SP)+,D1\n");  /* Operanden vom Stack holen */
        printf ("move (SP)+,D0\n");
        switch (c) {                 /* Operation ausfuehren */
        case '+':
            printf ("add D1,D0\n");
            break;
        case '-':
            printf ("sub D1,D0\n");
            break;
        case '*':
            printf ("muls D1,D0\n");
            break;
        case '/':
            printf ("divs D1,D0\n");
            break;
        }
        printf ("move D0,-(SP)\n"); /* Ergebnis auf den Stack legen */
        break;
    default:                        /* Werte */
        if (c >= '0' && c <= '9') {
            printf ("move #%c,-(SP)\n",c);
        } else {
            error ("Ungueltiges Zeichen");
        }
    }
}


/* expr ::= term | term + expr | term - expr */

void expr(FILE* input)
{
    char c;

    term(input);                /* am Anfang steht immer ein term */
    c = getc(input);            /* Folgezeichen einlesen */

    if (c == '+' || c == '-') {
        expr(input);            /* bei + oder - rekursiver Aufruf */
        print(c);              /* Operator (+ oder -) ausgeben */
    } else {
        ungetc(c, input);       /* ansonsten Zeichen zurueck in den Eingabe-
strom */
    }
}


/* term ::= factor | factor * term | factor / term */

void term(FILE* input)
{
    char c;

    factor(input);              /* am Anfang steht immer ein factor */
    c = getc(input);            /* Folgezeichen einlesen */
    if (c == '*' || c == '/') {
        term(input);            /* bei * oder / rekursiver Aufruf */
        print(c);              /* Operator (* oder /) ausgeben */
    } else {
        ungetc(c, input);       /* ansonsten Zeichen zurueck in den Eingabe-
```

```
strom */
    }
}


/* factor ::= (expr) | NUM | ID */

void factor(FILE* input)
{
    char c;

    c = getc(input);              /* Folgezeichen einlesen */
    if (c == '(') {               /* ist es ein Klammerausdruck? */
        expr(input);              /* ja => Ausdruck auswerten */
        if (getc(input) != ')') { /* auf schliessende Klammer pruefen */
            error("')' erwartet");
        }
    } else if (NUM(c) || ID(c)) { /* es muss eine Zahl oder ein Identifier */
        print(c);                 /* angegegeben werden */
    } else {
        error ("Zahl (0..9) oder Identifier (a..z) erwartet");
    }
}


/* NUM ::= '0' | .. | '9' */

int NUM(char c)
{
    return (c >= '0' && c <= '9');
}


/* ID ::= 'a' | .. | 'z' */

int ID(char c)
{
    return (c >= 'a' && c <= 'z');
}

int main()
{
    char c;

    FILE* input = stdin;     /* Standardeingabe verwenden */

    printf ("Infix -> 68000\n");
    printf ("Ausdruck eingeben (ohne Leerzeichen): ");

    expr(input);                /* Ausdruck auswerten */
    printf ("move (SP)+,D0\n"); /* Endergebnis nach D0 laden */

    if(getc(input) != '\n')  /* sind alle Zeichen bearbeitet? */
        error ("Ungueltiges Zeichen in der Eingabe");

    return 0;
}
```