Übungsaufgaben
Programmierungstechnik I

Aufgabe 11 (Tischrechner)

Das Programm aufg11_0.cpp realisiert einen Tischrechner, der von DOS aus u.a. folgende Eingaben akzeptiert und Berechnungen durchführt:

Beispiel

> **3+2** <enter>
> ***5***
> **3+2*1.4** <enter>
> ***5.8***
> **a=2*6; b=1.3; a+b** <enter>
> ***12***
1.3
13.3

Das Programm akzeptiert somit eine Grammatik der Form

Grammatik

```
<program>     ::=   END
                  |  <expr_list> END

<expr_list>   ::=   <expr> PRINT
                  |  <expr> PRINT <expr_list>

<expr>        ::=   <expr> + <term>
                  |  <expr> - <term>
                  |  <term>

<term>        ::=   <term> / <term>
                  |  <term> * <term>
                  |  <primary>

<primary>     ::=   NUMBER
                  |  NAME
                  |  NAME = <expr>
                  |  - <primary>
                  |  ( <expr> )
```

mit Eingaben:

END:	^Z
PRINT:	\n oder ;
NUMBER:	Zahl
NAME:	Zeichenkette

Das Programm führt die Syntaxanalyse nach dem Prinzip des rekursiven Abstiegs (recursive descent) durch rekursiven Aufruf von Funktionen aus.

Grundstruktur und Funktionen

Die Funktionen expr(), term() und print() entsprechen den jeweiligen grammatikalischen Regeln: in expr() und term() werden die jeweiligen Werteberechnungen durchgeführt.

get_token() liest einen Satz (Zeichen für Zeichen) ein und bestimmt den token_value, d.h. ob eine ZAHL, ein +, ein - usw. eingegeben wurde.

Für die im Eingabesatz definierten Variablen existiert eine Struktur name mit string = Namensstring und value = zugehöriger Wert.

look() überprüft bei Auftreten einer Variablen, ob diese in der Namen-Tabelle bereits vorhanden ist und liefert die Position dieser Variablen. Ist der Variablenname nicht vorhanden und ins=1, dann wird der Name in die Tabelle eingefügt (insert()).

Lösen Sie für das unten stehende Programm folgende Aufgaben:

1. **aufg25_1**: Bringen Sie das Programm zum "Laufen" (Syntaxfehler berichtigen)

2. **aufg25_2**: Ändern Sie das Programm so, daß die nutzerdefinierten Variablennamen und Werte in einer einfach verketteten Liste (dynamische Datenstruktur) abgelegt werden. Der für die Speicherung eines Variablennamens benötigte Speicherplatz soll ebenfalls jeweils dynamisch zugewiesen werden

3. **aufg25_3**: Führen Sie in das Programm zusätzlich einen Term für Potenzen ein:

   term ::= ...
   | term ^ primary

   mit der Semantik $(term)^{(primary)}$. primary muß ganzzahlig (positiv oder negativ) sein. Das soll im Programm geprüft werden (ggf. runtimeerror).

```
#include <string.h>
#include <ctype.h>
#include <stdio.h>
enum token_value
{
        NAME,       NUMBER,         END,
        PLUS = '+', MINUS = '-',    MUL = '*',      DIV = '/',
        PRINT = ';', ASSIGN = '=',  LP = '(',       RP = ')'
};
```

```
token_value    curr_token;

double expr()
{
        double left=term();
        for(;;)
                switch(curr_token)
                {
                case PLUS:
                        get_token();
                        left+=term();
                        break;
                case MINUS:
                        get_token();
                        left-=term();
                        break;
                default:
                        return left;
                }
}
double term()
{
        double left=prim();
        double d;
        for(;;)
                switch(curr_token)
                {
                case MUL:
                        get_token();
                        left*=term();
                        break;
                case DIV:
                        get_token();
                        d=prim();
                        if(!d) return error("Division durch Null");
                        left/=d;
                        break;
                default:
                        return left;
                }
}

struct name
{
        char string[30];
        double value;
};
name table[30];
int max_row=0;
char name_string[256];
```

```
double number_value;

double prim()
{
	double e;
	switch(curr_token)
	{
	case NUMBER:
		get_token();
		return number_value;
	case NAME:
		if(get_token()==ASSIGN)
		{
			int n=insert(name_string);
			get_token();
			table[n].value=expr();
			return table[n].value;
		}
		return table[look(name_string,0)].value;
	case MINUS:
		get_token();
		return -prim();
	case LP:
		get_token();
		e=expr();
		if(curr_token!=RP) return error(") erwartet");
		get_token();
		return e;
	case END:
		return 1;
	default:
		return error("primary erwartet");
	}
}
token_value get_token()
{
	char c;
	do
	{
		if((c=getchar())==EOF) return curr_token=END;
	} while(c!='\n' && isspace(c));
	switch(c)
	{
	case ';': case '\n':
		 return curr_token=PRINT;
	case '*': case '/': case '+': case '-': case '(': case ')': case '=':
		 return curr_token=token_value(c);
	case '0': case '1': case '2': case '3': case '4':
	case '5': case '6': case '7': case '8': case '9':
	case '.':
```

```
                ungetc(c,stdin);
                scanf("%lg",&number_value);
                return curr_token=NUMBER;
        default:
                if(isalpha(c))
                {
                        char* p=name_string;
                        *p++=c;
                        while((c=getchar())!=EOF)
                                if(isalnum(c)) *p++=c;
                                else
                                {
                                        *p='\0';
                                        break;
                                }
                        ungetc(c,stdin);
                        return curr_token=NAME;
                }
                error("bad token");
                return curr_token=PRINT;
        }
}
int look(const char* p, int ins)
{
        int n;
        for(n=0; n<=max_row; n++)
                if(!strcmp(p,table[n].string)) return n;
        if(!ins) error("name not found");
        strcpy(table[n].string,p);
        max_row=n;
        return n;
}
int insert(const char* s)
{
        return look(s,1);
}
int no_of_errors;
double error(const char* s)
{
        printf("ERROR: %s\n",s);
        no_of_errors++;
        return 1;
}
int main()
{
        table[insert("pi")].value=3.1416;
        table[insert("e")].value=2.7183;
        while(1)
        {
                get_token();
```

```
            if(curr_token==END) break;
            if(curr_token==PRINT) continue;
            printf("%lg\n",expr());
        }
        return no_of_errors;
}
```