```
// Automatische Syntaxanalyse
// (C) Andreas Harder 2004

**********************************************/

#include "stdio.h"
#include <string.h>
#include <process.h>
#include <iostream.h>

/* Konstanten
**********************************************/

#define SUBSTANTIV 1
#define VERB 2
#define ADJEKTIV 3
#define ADVERB 4
#define ARTIKEL 5
#define PRAEPOSITION 6
#define BEGRENZER 7
int Begrenzer();


/* Metasyntax
**********************************************/

class SYNTAX
{
      public:
            char LEXKLASSE;
            void gibZeichen() ;
            int findeLEXKLASSE ( char* klasse) ;
            virtual bool parse(char *s) = 0 ;
};

/* Grammatik
**********************************************/
```

```
/* Lexikon DEUTSCHLEX
**************************************/

class Lexikon : public SYNTAX
{
      public:
            Lexikon() {
            }

            char LEMMA[40];
            void init () ;

            bool parse (char *s)
            {
                  // hier zur morphologischen Analyse
                  überladen

                  return 0 ;
            } ;

            int inLEXIKON(char* w, int LEXKLASSE) ;
};

/* Struktur von SATZ
**************************************/

class SATZ : public SYNTAX
{
public:

      SATZ() {
      }

      char * SatzEingabe ;
      bool istKongruent ( int np, int vp, int bg ) ;
      bool istNominalPhrase (char* zKette) ;
      bool istVerbalPhrase (char* zKette) ;
      int istPraepositionalPhrase (char* zKette) ;
```

```
        void gibZeichen() ;
        void segmentiereKette (char *s) ;
        int findeLEXKLASSE (char* w) ;
        bool parse(char *SatzEingabe) ;

        int existBegrenzer() ;
};

/* Struktur der Nominalphrase
*****************************************/

class NP : public SATZ
{
        public:
                int istDET_NP ( char *zKette ) ;
                int istSUBSTANTIV ( char *zKette ) ;
                int istERWEITERTES_SUBSTANTIV( char *zKette ) ;
                int istPRAEPOSITION ( char *zKette ) ;
};

/* Struktur der Verbalphrase
*****************************************/

class VP : public SATZ
{
        public:
                int VALENZ ;
                int istVerbalPhrase (char* zKette) ;
                int istVerbFinit (char *s);
                int existVERBKOMPLEMENT(char *s) ;
};

/* Struktur der Praepositionalphrase
*********************************/

class PP : public NP
{
        public:
                int istPRAEPOSITION( char *zKette ) ;
```

```
            int istPRAEP_NP ( char *zKette ) ;
};

class VF : public VP
{
      public:
            int istERWEITERTES_VERB ( char * zKette ) ;
            int istVERB ( char * zKette ) ;
};

/* Globale
*****************************************/

SATZ einSatz ;
Lexikon DEUTSCHLEX[40];
Lexikon LEXEM ;
VP vp ;
VF vf ;
NP np ;
PP pp ;
int db_pos = 0;
char zKette[200];
char *positionImZeichen=0;
char *CpositionImZeichen=0;
char *startPositionImZeichen= 0 ;
char Zeichen[80];
char CZeichen[80];
char PRAEDIKAT[30] ;
char npstr[2][200] ;
char vstr[200] ;
char praepo_backup[30] = "" ;
int i = 0 ;

/* Finde die LEXKLASSE des gegebenen LEXEMs
*****************************************/

int SATZ :: findeLEXKLASSE ( char* zKette )
{
      int posImLEXIKON;
```

```
        for ( posImLEXIKON=0; posImLEXIKON <
        db_pos; posImLEXIKON++ )
        {
              if ( strcmp(zKette,
              DEUTSCHLEX[posImLEXIKON].LEMMA ) == 0)

              {
                    return
                    DEUTSCHLEX[posImLEXIKON].LEXKLASSE;
              }
        }
        return 0;
}

/* Segmentiere Zeichen im Ausgabestrom
*****************************************/

void SATZ :: gibZeichen()
{
        char *posZeiger;
        posZeiger = Zeichen;

        /* Übergehe Leerstelle(n) */

        while ( *positionImZeichen == ' ')
        {
              positionImZeichen++;
        }

        if( *positionImZeichen == '.' )
        {
               *posZeiger++ =    '.' ;
               *posZeiger =      NULL ;
        }

        /* Lies LEXEM bis Leerstelle oder Satzende
        ****************************************/

        while(*positionImZeichen!=' ' &&
```

```
        *positionImZeichen != '.' )

        {
                *posZeiger = *positionImZeichen++;
                posZeiger++;
        }

        *posZeiger = NULL ;
}

/* Segmentiere Kette
*****************************************/

void SATZ :: segmentiereKette (char *s)
{
        gibZeichen();
        strcat (s, Zeichen) ;
        strcat (s, " ") ;
}

/* Lies eine NominalPhrase aus dem Eingabestrom
*****************************************/

bool SATZ :: istNominalPhrase ( char* s )
{
        char temp[200] ;

         if ( np.istDET_NP (s) == 5 )
         {
                strcpy (temp, s) ;
                segmentiereKette (temp) ;

                if ( np.istERWEITERTES_SUBSTANTIV
                     (temp) == 1 )
                {
                        i++ ;
                        return 1 ;
                }
```

```
                if ( np.istSUBSTANTIV (s) == 1 )
                {
                        i++ ;
                        return 1 ;
                }
        }

        return 0 ;
}

/* Lies einen DETERMINATOR
*****************************************/

int NP :: istDET_NP( char *s )
{
        LEXKLASSE = findeLEXKLASSE ( Zeichen ) ;

        if (LEXKLASSE == ARTIKEL)
        {
                        strcat ( npstr[i], Zeichen ) ;
                        return 5 ;
        }

        return 0 ;
}

/* Lies einen PRAEP-DETERMINATOR
*****************************************/

int PP :: istPRAEPOSITION( char *s )
{
        LEXKLASSE = findeLEXKLASSE ( Zeichen ) ;

        if ( LEXKLASSE == PRAEPOSITION )
        {
                strcpy (praepo_backup, Zeichen ) ;
                return 6;
        }
```

```
        return 0 ;
}

/* Lies eine PraepositionalPhrase
*****************************************/

int SATZ :: istPraepositionalPhrase ( char* s )
{
        LEXKLASSE = pp.istPRAEPOSITION (Zeichen) ;

        if ( LEXKLASSE == PRAEPOSITION )
        {
                segmentiereKette (s);
                if ( np.istNominalPhrase (s) == 1 )
                {
                        return 1 ;
                }
        }

        return 0 ;
}

/* Lies ein SUBSTANTIV
*****************************************/

int NP :: istSUBSTANTIV( char *s )
{
        LEXKLASSE = findeLEXKLASSE(Zeichen);

        if ( LEXKLASSE == SUBSTANTIV )
        {
                strcat ( npstr[i], " " ) ;
                strcat ( npstr[i], Zeichen ) ;
                return 1;
        }

        return 0 ;
}
```

```
/* Lies ein ERWEITERTES_SUBSTANTIV
**************************************/

int NP :: istERWEITERTES_SUBSTANTIV( char *s )
{
      LEXKLASSE = findeLEXKLASSE(Zeichen);

      if (LEXKLASSE == ADJEKTIV )
      {
            strcat ( npstr[i], " " ) ;
            strcat ( npstr[i], Zeichen ) ;
            segmentiereKette (s);

            if ( np.istSUBSTANTIV (s) == 1 )
            {
                  return 1;
            }

            return 0 ;
      }

      return 0 ;
}

/* Lies eine VERBALPHRASE
****************************************/

bool SATZ :: istVerbalPhrase (char *zKette)
{
       char *pos, temp[200] ;
       char temp2[200] ;
       strcpy( temp, zKette );
       pos = positionImZeichen;

       /* Identifiziere das PRAEDIKAT
      ***********************************/

       segmentiereKette ( temp );
```

```
        if( vp.istVerbFinit ( temp ) != 0 )
        {
                vp.VALENZ = 1 ;
        }

       /* Prüfe VERBERGAENZUNG
       ************************************/

        strcpy ( temp2, temp ) ;
        segmentiereKette (temp2) ;

        if ( vp.existVERBKOMPLEMENT (temp2) == 1)
        {
             return 1 ;
        }

       return 0 ;
}

/*****************************************/

int VP :: existVERBKOMPLEMENT (char *s)
{

 if ( vp.VALENZ == 1)
 {
       /* VERB + PP */
       /*************************************/

        if ( pp.istPraepositionalPhrase ( s ) == 1)
        {
             return 1;
        }

       /* VERB + NP
       /*************************************/

        if( np.istNominalPhrase ( s ) == 1)
        {
```

```
            return 1;
        }

        /* Prüfe Satzende
        ***************************************/

            if( existBegrenzer() == 1 )
            {
                return 1;
            }
            return 0 ;
        }

        return 0 ;
}

/* Lies ein VERB
*******************************************/

int VP :: istVerbFinit (char *temp)
{
        if ( vf.istVERB ( Zeichen ) == 1 )
        {
            vp.VALENZ = 1 ;
        }

        char temp_backup[200] ;
        strcpy (temp_backup, temp) ;
        segmentiereKette (temp) ;

        if ( vf.istERWEITERTES_VERB (temp) == 1 )
        {
            return 1 ;
        }

        return 0 ;
        }

        ***************************************/
```

```
int VF :: istVERB( char *s )
{
	LEXKLASSE = findeLEXKLASSE( Zeichen );

	if ( LEXKLASSE == VERB )
	{
		strcat ( vstr, Zeichen ) ;
		return 1;
	}

	return 0 ;
}

/* Lies ein ERWEITERTES_VERB
*****************************************/

int VF :: istERWEITERTES_VERB( char *s )
{
	LEXKLASSE = findeLEXKLASSE(Zeichen);

	if ( LEXKLASSE == ADVERB )
	{
		strcat ( vstr, " " ) ;
		strcat ( vstr, Zeichen ) ;
		return 1;
	}

	return 0 ;
}

/* Finde BEGRENZER
*****************************************/

int SATZ :: existBegrenzer()
{
	gibZeichen () ;
	return ( findeLEXKLASSE ( Zeichen ) == BEGRENZER );
}
```

```
/* Kontextxtfreies RTN (Recursive Transition Network) -
Parser
**********************************************/

bool SATZ :: parse ( char *s )
{
      segmentiereKette( s ) ;

      if ( einSatz.istNominalPhrase ( s) == 1 )
      {
            if( einSatz.istVerbalPhrase( s ) == 1)
            {
                  if( einSatz.existBegrenzer() )
                  {
                              return 1 ;
                  }

                  return 0 ;
            }

            return 0;
      }

      return 0;
}

/* LEMMATA ins LEXIKON eintragen
**********************************************/

int Lexikon :: inLEXIKON ( char* w, int LEXKLASSE )
{
       if(db_pos<40)
       {
            strcpy(DEUTSCHLEX[db_pos].LEMMA, w);
            DEUTSCHLEX[db_pos].LEXKLASSE=LEXKLASSE;
            db_pos++;
       }
```

```
        return 0 ;
}

/* inLEXIKON
*****************************************/

void Lexikon :: init()
{
        inLEXIKON ("Tuer",SUBSTANTIV);
        inLEXIKON ("Fenster",   SUBSTANTIV);
        inLEXIKON ("Haus",      SUBSTANTIV);
        inLEXIKON ("Junge",     SUBSTANTIV);
        inLEXIKON ("Hof",       SUBSTANTIV);
        inLEXIKON ("Ball",  SUBSTANTIV);
        inLEXIKON ("hat",       VERB);
        inLEXIKON ("geht",VERB);
        inLEXIKON ("spielt",    VERB);
        inLEXIKON ("sieht",     VERB);
        inLEXIKON ("grosses",   ADJEKTIV);
        inLEXIKON ("grossen",   ADJEKTIV);
        inLEXIKON ("grosse",    ADJEKTIV);
        inLEXIKON ("neue",      ADJEKTIV);
        inLEXIKON ("neues",     ADJEKTIV);
        inLEXIKON ("neuen",     ADJEKTIV);
        inLEXIKON ("gern",ADVERB);
        inLEXIKON ("oft",       ADVERB);
        inLEXIKON ("viel",  ADVERB);
        inLEXIKON ("das",       ARTIKEL);
        inLEXIKON ("der",       ARTIKEL);
        inLEXIKON ("dem",       ARTIKEL);
        inLEXIKON ("ein",       ARTIKEL);
        inLEXIKON ("eine", ARTIKEL);
        inLEXIKON ("einen",     ARTIKEL);
        inLEXIKON ("einem",     ARTIKEL);
        inLEXIKON ("mit",       PRAEPOSITION);
        inLEXIKON ("in",        PRAEPOSITION);
        inLEXIKON ("auf",       PRAEPOSITION);
        inLEXIKON ("zu",        PRAEPOSITION);
        inLEXIKON ("nach",PRAEPOSITION);
```

```
        inLEXIKON (".",          BEGRENZER);
        printf ( "Lexikon initialisiert\n" ) ;

        printf ( "Bilden Sie einen deutschen Satz aus den
      Woertern: \n\nTuer, Fenster, Haus, Junge, Hof,
      Ball, hat, geht, spielt, sieht, \ngrosses, grosse,
      grossen, neue, neues, neuen, gern, oft, viel,
      \ndas, der, dem, ein, eine, einem, einen, mit, in,
      auf, zu, nach !\n\n") ;
}

/*****************************************/

void main()
{
      char se [200] ;

      // Deutsches Wörterbuch initialisieren

      DEUTSCHLEX->init();

      // Dialog

      printf ( "Bitte einen Satz eingeben und mit Punkt (.)
      beenden:\n" ) ;
      gets ( se ) ;

      // Initialisiserung

      positionImZeichen = se ;

      // Syntaxanalyse
      /*************************************/

      if( einSatz.parse (se) == 1 )
      {
            printf ("\n-------------------------------\n") ;

            printf ("S[ NP[%s] + ", npstr[0]) ;
```

```
                printf ("\n\tVP[VERB[%s]", vstr ) ;

                if (!(strcmp(praepo_backup, "" ) == 0) )
                {
                      printf("+ \n\t\tPP[ PAEPOSITION[%s] ",
                      praepo_backup ) ;
                }

                printf ( "+ NP [%s]", npstr[1]) ;

                if (!(strcmp(praepo_backup, "" ) == 0) )
                {
                      printf("\n\t\t]\n\n") ;
                }

                printf ("\t]\n") ;
                printf ("]\n------------------------------\n") ;
                printf("Syntax OK\n");
        }

        else
        {
                printf("Satz ist syntaktisch falsch\n");
        }
}
```