/\* Charcode \*/

#include "charcode.h"

// Follow ascii table

// example underscore = 95 -> row 12

CharCode charCodes[256] = {

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_SPACE, CHAR\_SPACE, CHAR\_SPACE, CHAR\_SPACE, CHAR\_SPACE, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_SPACE, CHAR\_EXCLAIMATION, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_SINGLEQUOTE,

CHAR\_LPAR, CHAR\_RPAR, CHAR\_TIMES, CHAR\_PLUS, CHAR\_COMMA, CHAR\_MINUS, CHAR\_PERIOD, CHAR\_SLASH,

CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_DIGIT,

CHAR\_DIGIT, CHAR\_DIGIT, CHAR\_COLON, CHAR\_SEMICOLON, CHAR\_LT, CHAR\_EQ, CHAR\_GT, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_UNKNOWN, CHAR\_BACKSLASH, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNDERSCORE,

CHAR\_UNKNOWN, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER,

CHAR\_LETTER, CHAR\_LETTER, CHAR\_LETTER, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN,

CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN, CHAR\_UNKNOWN

};

#ifndef \_\_CHARCODE\_H\_\_

#define \_\_CHARCODE\_H\_\_

typedef enum {

CHAR\_SPACE, // khoang trong

CHAR\_LETTER, // chu cai

CHAR\_DIGIT, // chu so

CHAR\_PLUS, // '+'

CHAR\_MINUS, // '-'

CHAR\_TIMES, // '\*'

CHAR\_SLASH, // '/'

CHAR\_LT, // '<'

CHAR\_GT, // '>'

CHAR\_EXCLAIMATION, // '!'

CHAR\_EQ, // '='

CHAR\_COMMA, // ','

CHAR\_PERIOD, // '.'

CHAR\_COLON, // ':'

CHAR\_SEMICOLON, // ';'

CHAR\_SINGLEQUOTE, // '\'

CHAR\_LPAR, // '('

CHAR\_RPAR, // ')'

CHAR\_UNDERSCORE, // '\_'

CHAR\_BACKSLASH, // '\'

CHAR\_UNKNOWN // Ky tu ngoai bang chu cai

} CharCode;

/\* Error \*/

#include <stdio.h>

#include <stdlib.h>

#include "error.h"

void error(ErrorCode err, int lineNo, int colNo) {

switch (err) {

case ERR\_ENDOFCOMMENT:

printf("%d-%d:%s\n", lineNo, colNo, ERM\_ENDOFCOMMENT);

break;

case ERR\_IDENTTOOLONG:

printf("%d-%d:%s\n", lineNo, colNo, ERM\_IDENTTOOLONG);

break;

case ERR\_INVALIDCHARCONSTANT:

printf("%d-%d:%s\n", lineNo, colNo, ERM\_INVALIDCHARCONSTANT);

break;

case ERR\_INVALIDSYMBOL:

printf("%d-%d:%s\n", lineNo, colNo, ERM\_INVALIDSYMBOL);

break;

case ERR\_NUMBERTOOLONG:

printf("%d-%d:%s\n", lineNo, colNo, ERM\_NUMBERTOOLONG);

break;

}

exit(-1);

}

#ifndef \_\_ERROR\_H\_\_

#define \_\_ERROR\_H\_\_

// Danh sach cac loi trong qua trinh phan tich tu vung

typedef enum {

ERR\_ENDOFCOMMENT, // (\* comment

ERR\_IDENTTOOLONG, // Charrrrrrrrrrrrrrrrrrrrrrrrrrrrrr

ERR\_INVALIDCHARCONSTANT, // ' A'

ERR\_INVALIDSYMBOL, // !!!

ERR\_NUMBERTOOLONG // 999999999999999, max 10 chu so

} ErrorCode;

// Cac thong bao loi

#define ERM\_ENDOFCOMMENT "End of comment expected!"

#define ERM\_IDENTTOOLONG "Identification too long!"

#define ERM\_INVALIDCHARCONSTANT "Invalid const char!"

#define ERM\_INVALIDSYMBOL "Invalid symbol!"

#define ERM\_NUMBERTOOLONG "Number too long!"

// Ham thong bao loi

void error(ErrorCode err, int lineNo, int colNo);

#endif

/\* Reader \*/

#include <stdio.h>

#include "reader.h"

FILE \*inputStream;

int lineNo, colNo;

int currentChar;

int readChar(void) {

currentChar = getc(inputStream);

colNo ++;

if (currentChar == '\n') {

lineNo ++;

colNo = 0;

}

return currentChar;

}

int openInputStream(char \*fileName) {

inputStream = fopen(fileName, "rt");

if (inputStream == NULL)

return IO\_ERROR;

lineNo = 1;

colNo = 0;

readChar();

return IO\_SUCCESS;

}

void closeInputStream() {

fclose(inputStream);

}

#ifndef \_\_READER\_H\_\_

#define \_\_READER\_H\_\_

#define IO\_ERROR 0

#define IO\_SUCCESS 1

int readChar(void);

int openInputStream(char \*fileName);

void closeInputStream(void);

#endif

/\* Token \*/

#include <stdlib.h>

#include <ctype.h>

#include "token.h"

struct

{

char string[MAX\_IDENT\_LEN + 1];

TokenType tokenType;

} keywords[KEYWORDS\_COUNT] = {

{"PROGRAM", KW\_PROGRAM},

{"CONST", KW\_CONST},

{"TYPE", KW\_TYPE},

{"VAR", KW\_VAR},

{"INTEGER", KW\_INTEGER},

{"CHAR", KW\_CHAR},

{"ARRAY", KW\_ARRAY},

{"OF", KW\_OF},

{"FUNCTION", KW\_FUNCTION},

{"PROCEDURE", KW\_PROCEDURE},

{"BEGIN", KW\_BEGIN},

{"END", KW\_END},

{"CALL", KW\_CALL},

{"IF", KW\_IF},

{"THEN", KW\_THEN},

{"ELSE", KW\_ELSE},

{"WHILE", KW\_WHILE},

{"DO", KW\_DO},

{"FOR", KW\_FOR},

{"TO", KW\_TO}};

int keywordEq(char \*kw, char \*string)

{

while ((\*kw != '\0') && (\*string != '\0'))

{

if (\*kw != toupper(\*string))

// if (\*kw != \*string) // distinguish between keyword in upper and not in upper

break;

kw++;

string++;

}

return ((\*kw == '\0') && (\*string == '\0'));

}

TokenType checkKeyword(char \*string)

{

int i;

for (i = 0; i < KEYWORDS\_COUNT; i++) // Keyword count in token.h

if (keywordEq(keywords[i].string, string))

return keywords[i].tokenType;

return TK\_NONE;

}

Token \*makeToken(TokenType tokenType, int lineNo, int colNo)

{

Token \*token = (Token \*)malloc(sizeof(Token));

token->tokenType = tokenType;

token->lineNo = lineNo;

token->colNo = colNo;

return token;

}

#ifndef \_\_TOKEN\_H\_\_

#define \_\_TOKEN\_H\_\_

#define MAX\_IDENT\_LEN 15

#define KEYWORDS\_COUNT 20

#define NUMBER\_LEN 10

typedef enum {

TK\_NONE, // Dai dien cho 1 loi

TK\_IDENT, // Dinh danh

TK\_NUMBER, // So

TK\_CHAR, // Hang ki tu

TK\_EOF, // Ket thuc chuong trinh

// Cac tu khoa

KW\_PROGRAM, KW\_CONST, KW\_TYPE, KW\_VAR,

KW\_INTEGER, KW\_CHAR, KW\_ARRAY, KW\_OF,

KW\_FUNCTION, KW\_PROCEDURE,

KW\_BEGIN, KW\_END, KW\_CALL,

KW\_IF, KW\_THEN, KW\_ELSE,

KW\_WHILE, KW\_DO, KW\_FOR, KW\_TO,

// Cac ki hieu dac biet

SB\_SEMICOLON, SB\_COLON, SB\_PERIOD, SB\_COMMA,

SB\_ASSIGN, SB\_EQ, SB\_NEQ, SB\_LT, SB\_LE, SB\_GT, SB\_GE,

SB\_PLUS, SB\_MINUS, SB\_TIMES, SB\_SLASH,

SB\_LPAR, SB\_RPAR, SB\_LSEL, SB\_RSEL, SB\_BACKSLASH

} TokenType;

// Cau truc luu tru 1 token

typedef struct {

char string[MAX\_IDENT\_LEN + 1];

int lineNo, colNo;

TokenType tokenType;

int value;

} Token;

// Kiem tra 1 xau co phai tu khoa khong

TokenType checkKeyword(char \*string);

// Tao 1 token moi voi kieu va vi tri

Token\* makeToken(TokenType tokenType, int lineNo, int colNo);

#endif