1. **KPL’’s alphabet**

- Letter: a b c … x y z

A B C … X Y Z

- Digit: 0 1 2 3 … 9

- Special charater:

* + - \* /
* < > ! =
* [space] [comma , ] . : ; ‘ \_
* ( )

- Keywords:

|  |  |
| --- | --- |
| * PROGRAM | * INTEGER |
| * CONST | * CHAR |
| * TYPE | * CALL |
| * VAR | * IF |
| * PROCEDURE | * THEN |
| * FUNCTION | * ELSE |
| * BEGIN | * WHILE |
| * END | * DO |
| * ARRAY | * FOR |
| * OF | * TO |

- Operators:

* := assign
* + - \* /
* = != > < <= >=

- Special characters:

* ; (semicolon)
* . (period)
* : (colon)
* , (comma)
* ( (left parenthesis)
* ) (right parenthesis)
* ‘ (singlequote)

1. **Code**
   1. ***CharCode***

typedef enum {  
 CHAR\_SPACE, // space  
 CHAR\_LETTER, // character  
 CHAR\_DIGIT, // digit  
 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\_UNKNOWN // invalid character  
} **CharCode**;

* 1. ***TokenType***

typedef enum {  
 TK\_NONE, // Invalid token - Error  
 TK\_IDENT, // Identifier token  
 TK\_NUMBER, // Number token  
 TK\_CHAR, // Character constant token  
 TK\_EOF, // End of program token  
// keywords  
 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,  
// Special character  
 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  
} **TokenType**;

// Structure of a token

#define MAX\_IDENT\_LEN 15  
typedef struct {  
 char string[MAX\_IDENT\_LEN + 1];  
 int lineNo, colNo;  
 TokenType tokenType;  
 int value;  
} **Token**;

// Check whether a string is a keyword or not  
TokenType checkKeyword(char \*string);  
// Create new token, provided type of token and location  
Token\* makeToken(TokenType tokenType, int lineNo, int  
colNo)

* 1. ***ErrorCode***

// List of error may occur in lexical analysis  
typedef enum {  
 ERR\_ENDOFCOMMENT,  
 ERR\_IDENTTOOLONG,  
 ERR\_INVALIDCHARCONSTANT,  
 ERR\_INVALIDSYMBOL  
} **ErrorCode**;

// Error message  
#define ERM\_ENDOFCOMMENT "End of comment expected!"  
#define ERM\_IDENTTOOLONG "Identification too long!"  
#define ERM\_INVALIDCHARCONSTANT "Invalid const char!"  
#define ERM\_INVALIDSYMBOL "Invalid symbol!"

// Return error message  
void error(ErrorCode err, int lineNo, int colNo);