**1) Only 16 first char counted**

Token \*readIdentKeyword(void)

{

Token \*token = makeToken(TK\_NONE, lineNo, colNo);

int count = 1;

token->string[0] = (char)currentChar;

readChar();

while ((currentChar != EOF) &&

((charCodes[currentChar] == CHAR\_LETTER) || (charCodes[currentChar] == CHAR\_DIGIT)))

{

// token in KPL has a max length of 15

// everytime we read a char, check if it's over MAX\_INDENT\_LEN and notify error

if (count <= MAX\_IDENT\_LEN)

token->string[count++] = (char)currentChar; // store currentChar to string of token

// currentChar is a number --> must be parse to char

readChar();

}

if (count > MAX\_IDENT\_LEN)

{

error(ERR\_IDENTTOOLONG, token->lineNo, token->colNo);

return token;

}

// count++ above --> string[count] should be end of string

token->string[count] = '\0';

token->tokenType = checkKeyword(token->string);

// if TK\_NONE (NOT A KEYWORD), convert to TK\_INDENT

if (token->tokenType == TK\_NONE)

token->tokenType = TK\_IDENT;

return token;

}

**2) 4 quotes**

Token \*readConstChar(void)

{

Token \*token = makeToken(TK\_CHAR, lineNo, colNo);

int numberOfSingleQuote = 1;

readChar();

if (currentChar == EOF)

{

token->tokenType = TK\_NONE;

error(ERR\_INVALIDCHARCONSTANT, token->lineNo, token->colNo);

return token;

}

if (charCodes[currentChar] == CHAR\_SINGLEQUOTE)

// begin to check if is a single quote 4 - ''''

{

numberOfSingleQuote++; // = 2 now

readChar();

if (charCodes[currentChar] == CHAR\_SINGLEQUOTE)

{

numberOfSingleQuote++; // = 3 now

readChar();

if (charCodes[currentChar] == CHAR\_SINGLEQUOTE)

{

numberOfSingleQuote++; // = 4 now

token = makeToken(SB\_SINGLEQUOTE, lineNo, colNo);

readChar();

return token;

}

else

{

token->tokenType = TK\_NONE;

error(ERR\_INVALIDCHARCONSTANT, token->lineNo, token->colNo);

return token;

}

}

else

{

token->tokenType = TK\_NONE;

error(ERR\_INVALIDCHARCONSTANT, token->lineNo, token->colNo);

return token;

}

}

else

{

token->string[0] = currentChar;

token->string[1] = '\0';

readChar();

if (currentChar == EOF)

{

token->tokenType = TK\_NONE;

error(ERR\_INVALIDCHARCONSTANT, token->lineNo, token->colNo);

return token;

}

if (charCodes[currentChar] == CHAR\_SINGLEQUOTE)

{

readChar();

return token;

}

else

{

token->tokenType = TK\_NONE;

error(ERR\_INVALIDCHARCONSTANT, token->lineNo, token->colNo);

return token;

}

}

}

**3) Exclaimation not equal (!=)**

case CHAR\_EXCLAIMATION:

ln = lineNo;

cn = colNo;

readChar();

if ((currentChar == EOF)) {

error(ERR\_INVALIDSYMBOL, ln, cn);

return token;

} else {

error(ERR\_INVALIDSYMBOL, ln, cn);

if(charCodes[currentChar] == CHAR\_EQ) {

token = makeToken(SB\_EQ, lineNo, colNo);

readChar();

return token;

}

}

**4) Ident - letter - underscore**

Token\* readIdentKeyword(int ln, int cn) {

// TODO

Token \*token = makeToken(TK\_IDENT, ln, cn);

int sz = 0;

while (charCodes[currentChar] == CHAR\_LETTER || charCodes[currentChar] == CHAR\_DIGIT || charCodes[currentChar] == CHAR\_UNDERSCORE) {

if (sz >= MAX\_IDENT\_LEN) {

token->tokenType = TK\_NONE;

}

else

token->string[sz++] = currentChar;

readChar();

}

if (token->tokenType == TK\_NONE) {

//error(ERR\_IDENTTOOLONG, ln, cn);

return token;

}

token->string[sz] = 0;

token->tokenType = checkKeyword(token->string);

if (token->tokenType == TK\_NONE) {

token->tokenType = TK\_IDENT;

}

return token;

}