satooooshi / **compiler**

Branch: **dev** compiler / lab2 / **tiger.lex** Find file Copy path

Cannot retrieve latest commit at this time.

0 contributors

230 lines (180 sloc) 4.93 KB

```
%{
#include <string.h>
#include "util.h"
#include "tokens.h"
#include "errormsg.h"

int charPos=1;

int yywrap(void)
{
 charPos=1;
 return 1;
}

void adjust(void)
{
 EM_tokPos=charPos;
 charPos+=yyleng;
}

/*
* Please don't modify the lines above.
* You can add C declarations of your own below.
*/

#define MAX_STR_LEN 1024

int commentLevel=0; /* for nested comment */

char string_buf[MAX_STR_LEN + 1];
char *string_buf_ptr;

void adjuststr(void)
{
 charPos+=yyleng;
}


/* @function: getstr
 * @input: a string literal
 * @output: the string value for the input which has all the escape sequences
 * translated into their meaning.
 */
char *getstr(const char *str)
{
  return NULL;
}

int chtodec (char c)
 {
   if( c>='@' && c<='Z' ){
    c -= 64;//convert\^A(\^065)to\^001
   }else{
    EM_error(charPos, "Unknown Controll Character.");
   }
   return c;
 }


unsigned long charCount = 0, wordCount = 0, lineCount = 0;

#undef yywrap  /* sometimes a macro by default */

%}
  /* You can add lex definitions here. */

%x str comment

INTregExp [0-9]+
IDregExp [a-zA-Z][a-zA-Z0-9_]*


%%
  /*
  * Below are some examples, which you can wipe out
  * and write reguler expressions and actions of your own.

  */


  /* string */
<str>{

  \"  {
    adjuststr();
    *string_buf_ptr='\0';
    if (string_buf[0] != '\0')
      yylval.sval=String(string_buf);
    else
      yylval.sval=String("(null)"); /* Compatible with test case */
    BEGIN(INITIAL);
    return STRING;
  } /* comment starting */

  \\([0-9]{3}) {
    adjuststr();
    int result = atoi(yytext + 1);
    char c = (char)result;

    if (result > 0xff) {
      EM_error(EM_tokPos, "illegal character");
      continue;
    }
    *string_buf_ptr++ = c;
  }

  \\n     {adjuststr(); *string_buf_ptr++ = '\n';}
  \\t     {adjuststr(); *string_buf_ptr++ = '\t';}
  \\\"    {adjuststr(); *string_buf_ptr++ = '\"';}
  \\\\    {adjuststr(); *string_buf_ptr++ = '\\';}
  \\\^[\0-\037]   {
    adjuststr();
    *string_buf_ptr++ = yytext[2];
  } /* \\\^ means "\^" OCT:\0-\37 */



  (\\\^)[\x41-\x5A\x61-\x7A] {
    adjuststr();
   *string_buf_ptr++ = chtodec(yytext[2]);
  } /* \\\^ means "\^", OCT:\101-\132\141-\172, Hex:\x41-\x5A\x61-\x7A Dec:65-90,97-122 means A~Za~z Ex. char ch=67,  char *st


  \\[ \t\n\r]+\\ {
    adjuststr();
```

```
      char *yytextptr = yytext;
      while (*yytextptr != '\0')
      {
        if (*yytextptr == '\n')
          EM_newline();
        ++yytextptr;
      }
    }

    \\. {adjuststr(); EM_error(charPos, "illegal escape char");}

    \n  {
      adjuststr();
      EM_newline();
      EM_error(charPos, "string terminated with newline");
      continue;
    }

    [^\\\n\"]+        {
      adjuststr();
      char *yptr = yytext;

      while (*yptr)
        *string_buf_ptr++ = *yptr++;
    }
  }

    /* comment, note that the special start-condition specifier `<*>' matches every start condition.
      Can match where ever < > state is in .
    */

  <*>{
    "/*" {adjust(); ++commentLevel; BEGIN(comment);}
  }

  <comment>{
    \n    {adjust(); EM_newline();}
    "*/"  {adjust(); --commentLevel; if (commentLevel <= 0) BEGIN(INITIAL);}
    .     {adjust();}
  }

  <INITIAL>{

    (" "|"\t")  {adjust();}
    \n     {adjust(); EM_newline();}

    ","    {adjust(); return COMMA;}
    ":"   {adjust(); return COLON;}
    ";"   {adjust(); return SEMICOLON;}

    "("   {adjust(); return LPAREN;}
    ")"   {adjust(); return RPAREN;}
    "["   {adjust(); return LBRACK;}
    "]"   {adjust(); return RBRACK;}
    "{"   {adjust(); return LBRACE;}
    "}"   {adjust(); return RBRACE;}

    "."   {adjust(); return DOT;}
    "+"   {adjust(); return PLUS;}
    "-"   {adjust(); return MINUS;}
    "*"   {adjust(); return TIMES;}
    "/"   {adjust(); return DIVIDE;}

    "="   {adjust(); return EQ;}
    "<>"  {adjust(); return NEQ;}
    "<"   {adjust(); return LT;}
    "<="  {adjust(); return LE;}
    ">"   {adjust(); return GT;}
    ">="  {adjust(); return GE;}
    "&"   {adjust(); return AND;}
    "|"   {adjust(); return OR;}
    ":="  {adjust(); return ASSIGN;}

    array {adjust(); return ARRAY;}
    if    {adjust(); return IF;}
    then  {adjust(); return THEN;}
    else  {adjust(); return ELSE;}
    while {adjust(); return WHILE;}
    for   {adjust(); return FOR;}
    to    {adjust(); return TO;}
    do    {adjust(); return DO;}
    let   {adjust(); return LET;}
    in    {adjust(); return IN;}
    end   {adjust(); return END;}
    of    {adjust(); return OF;}
    break {adjust(); return BREAK;}
    nil   {adjust(); return NIL;}
    function  {adjust(); return FUNCTION;}
    var   {adjust(); return VAR;}
    type  {adjust(); return TYPE;}


    {INTregExp} {adjust(); yylval.ival=atoi(yytext); return INT;}
    {IDregExp} {adjust(); yylval.sval=String(yytext); return ID;}
    \"   {
      adjust();
      string_buf_ptr = string_buf;
      BEGIN(str);
    }
  }

  .       {adjust(); EM_error(EM_tokPos,"illegal token");}

  %%
```