این قسمت پروژه برای تحلیل لغتی زبان است که به این معناست که برای مثال برنامه ای که به زبان cool نوشنه شده است باید به صورتی دربیاید که برای parser قابل تحلیل باشد پس باید برنامه را lex کنیم. که به این معناست که کامنت ها را در نظر نگیریم و کلاس ها و استرینگ ها و متغیر ها را که در هر خطی وجود دارد مشخص کنیم و ارور های مربوط به ان ها را هم مشخص کنیم.

در ابندا تعریف هایی برای حرف ها و رقم ها و خط جدید و کامنتی که با - - شروع شود و متغییر ها و شی ها و عدد ها را تعریف میکنیم.

سپس برای مثال CLASS (?i:class) این را تعریف میکنیم که یعنی CLASS همه کلمات با حروف بزرگ و کوچک class است .

ولی برای true , false این متفاوت است زیرا حرف اول باید کوچک باشد پس حرف اول را کوچک میگذاریم و بقیه حروف هم میتوانند کوچک باشند و هم بزرگ.

در این قسمت در صورت شروع نشدن کامنت(در واقع نبودن در استیت کامنت) باید در صورت خواندن "\*)" خطا دهد.

"\*)" {

yylval.error\_msg ="Unmatched \*)";

return ERROR;

}

در صورتی که "(\*" را خواند مشخص میکند که کامنت شروع شده پس از اینجا به بعد دراستیت کامنت است و یکی به کامنت ها اضافه میکند که در صورت وجود کامنت های تو در تو مشکلی پیش نیاد و تا زمانی که کامنت ها برابر صفر نیست هنوز در استیت کامنت هستیم.

"(\*" {

BEGIN(comment);

commentha++;

}

در صورتی که در استیت کامنت باشیم برای رفع مشکل کامنت های تو در تو اگر "(\*" بیاید تعداد کامنت ها را یکی اضافه میکنیم.

<comment>"(\*" {

commentha++;

}

در صورتی که در قسمت کامنت ها خطای EOF رخ دهد در واقع به اخر فایل برسیم و هنوز در استیت کامنت باشیم پس باید ان را تمام میکردیم که این خطاست و وارد استیت شروع میشویم.

<comment><<EOF>> {

BEGIN(INITIAL);

if(commentha>0){

commentha=0;

yylval.error\_msg ="EOF in comment";

return ERROR;

}

}

در صورتی که \n را دیدیم در قسمت کامنت باید یکی به خطی که در ان هستیم اضافه کنیم.

<comment>\n {

curr\_lineno++; }

در صورتی که "\*)" در ادامه کامنت بیاید باید یکی از تعداد کامنت ها کم کنیم و اگر صفر شد BEGIN(INITIAL) استفاده میکنیم که وارد استیت شروع شویم و اگر کوچکتر از صفر شد باید خطا دهد و از ابتدا شروع کند.

<comment>"\*)" {

commentha--;

if(commentha==0)

BEGIN(INITIAL);

else if(commentha<0){

yylval.error\_msg = "Unmatched \*)";

commentha=0;

BEGIN(INITIAL);

return ERROR;

} }

اگر هر چیزی غیر این بود پس توضیحات در کامنت است که میتوانیم ان را در نظر نگیریم یعنی ان را با هیچی جایگزین میکنیم.

<comment>. {}

اگر – کامنت بود که ان را تعریف کرده ایم فقط یکی به تعداد خط ها اضافه میکنیم.

{DASHCOMMENT} {

curr\_lineno++;}

در صورت خواندن هر کدام از این کلمات(بعضی را خودمان تعریف کرده ایم) کلمه مناسب را برمیگردانیم که در parser استفاده میشود.

"=>" { return DARROW;}

"=<" { return LE;}

"<-" { return ASSIGN;}

"<" { return '<';}

"@" { return '@';}

"~" { return '~';}

"=" { return '=';}

"." { return '.';}

"-" { return '-';}

"," { return ',';}

"+" { return '+';}

"\*" { return '\*';}

"/" { return '/';}

"}" { return '}';}

"{" { return '{';}

"(" { return '(';}

")" { return ')';}

":" { return ':';}

";" { return ';';}

{CLASS} { return CLASS;}

{ELSE} { return ELSE;}

{FI} { return FI;}

{IF} { return IF;}

{IN} { return IN;}

{INHERITS} { return INHERITS;}

{LET} { return LET;}

{LOOP} { return LOOP;}

{POOL} { return POOL;}

{THEN} { return THEN;}

{WHILE} { return WHILE;}

{CASE} { return CASE;}

{ESAC} { return ESAC;}

{NEW} { return NEW;}

{OF} { return OF;}

{NOT} { return NOT;}

در صورتی که یک Object(تعریف شده است) یا self را خواندیم ان را در idtable قرار میدهیم و OBJECTID را برمیگردانیم که نشان میدهد از چه نوعی است.

{OBJECTID}|(self) {

yylval.symbol = idtable.add\_string(yytext);

return OBJECTID;

}

در صورتی که یک typeid (تعریف شده است) را خواندیم ان را در idtable قرار میدهیم و TYPEID را برمیگردانیم.

{TYPEID} {

yylval.symbol = idtable.add\_string(yytext);

return TYPEID;

}

در صورتی که یک عدد خواندیم ان را در inttable قرار میدهیم.

{INT\_CONST} {

yylval.symbol = inttable.add\_string(yytext);

return INT\_CONST;

}

در صورتی که \n را بخوانیم باید به خط بعد برویم.

\n {

curr\_lineno++;

}

و کلا میتوانیم WHITESPACE (که قبلا تعریف شده است ) را در نظر نگیریم.

{WHITESPACE} {}

در صورت دیدن TRUE , FALSE که ان ها را تعریف کرده ایم yylval.boolean را تعیین کنیم و BOOL\_CONST را برگردانیم.

{FALSE} {

yylval.boolean = false;

return BOOL\_CONST;

}

{TRUE} {

yylval.boolean = true;

return BOOL\_CONST;

}

در صورت خواندن “ که نشان شروع یک string است باید مکان ابتدای ان را ذخیره کنیم و در استیت string کار خود را ادامه دهیم تا زمانی که وارد استیت اغازین شویم یا به خطا بخوریم.

"\"" {

BEGIN(string);

string\_buf\_ptr = string\_buf;

}

حال در صورتی که بعد از استرینگ “ را خواندیم باید چک کنیم که طول استرینگ از 1025 بزرگتر نباشد در غیر این صورت باید خطا دهیم و اگر درست بود باید از استرینگ خارج شده و استرینگ بدست امده را در stringtable قرار دهیم و STR\_CONST را برگردانیم.

<string>"\"" {

if (string\_buf\_ptr - string\_buf > MAX\_STR\_CONST-1) {

\*string\_buf = '\0';

yylval.error\_msg = "String constant too long";

BEGIN(escape);

return ERROR;

}

BEGIN(INITIAL);

\*string\_buf\_ptr = '\0';

yylval.symbol = stringtable.add\_string(string\_buf);

return STR\_CONST;

}

در صورتی که به انتهای فایل رسیدیم و هنوز استرینگ ادامه داشت خطا میدهیم و از استیت استریگ خارج میشویم و وارد استیت شروع میشویم.

<string><<EOF>> {

yylval.error\_msg = "EOF in string constant";

BEGIN(INITIAL);

return ERROR;

}

در صورتی که در استرینگ وارد خط بعدی شویم باید خطا دهیم و string\_buf را خالی میکنیم و وارد استیت شروع میشویم.

<string>\n {

\*string\_buf = '\0';

yylval.error\_msg = "Unterminated string constant";

BEGIN(INITIAL);

return ERROR;

}

اگر استرینگ خالی بود باید خطا دهیم و وارد استیت escape شویم که در صورتی که “ یا \n خوانده شود دوباره وارد استیت شروع و در غیر این صورت ادامه نمیدهیم.

<string>\0 {

BEGIN(escape);

\*string\_buf = '\0';

yylval.error\_msg = "String contains null character";

return ERROR;

}

حال اگر در استیت استرینگ باشیم و یکی از \n یا \t یا \b یا \f را در استرینگ بخوانیم ان را به استریگی که میخواهیم ذخیره کنیم اضافه میکنیم و درصورتی که \c را خواندیم که c هیچکدام از حروف n,t,b,f نبود فقط c را به استرینگمان اضافه میکنیم و در غیر این صورت هر چیزی که میخوانیم را به استرینگ اضافه میکنیم.

<string>"\\n" \*string\_buf\_ptr++ = '\n';

<string>"\\t" \*string\_buf\_ptr++ = '\t';

<string>"\\b" \*string\_buf\_ptr++ = '\b';

<string>"\\f" \*string\_buf\_ptr++ = '\f';

<string>"\\"[^ntbf] \*string\_buf\_ptr++ = yytext[1];

<string>. \*string\_buf\_ptr++ = \*yytext;

<escape>[\n|"] BEGIN(INITIAL);

<escape>[^\n|"]

حال هر چیزی بخوانیم که در قسمت های بالا نباشد ان را خطا به حساب میاوریم و ان را برمیگردانیم.

. {

yylval.error\_msg = yytext;

return ERROR;

}

در test.cl یک کد ارایه شده که همان کد test.cl با کمی تغییر است به صورتی که بیشتر خطا ها را داشته باشد و تحلیل ان هم در فایل lex test.cl امده است.