پروژه کامپایلر

آرمان وزیری بزرگ

شماره دانشجویی : 9632425

امین حسن زاده مقدم

شماره دانشجویی : 9632413

این پروژه شامل دو بخش lexical analyzer و یک بخش syntax analyzer است. در قسمت syntax analyzer باید توکن ورودی را با توجه به گرامر تعریف شده که در ادامه به طور کامل آن را شرح خواهیم داد parse کنیم و همچنین همزمان با عمل parse کردن ، ترجمه ی رشته ی ورودی را به رشته ی حرفی مشخص شده در تعریف پروژه انجام دهیم. این ترجمه توسط semantic rules و attribute های synthesized انجام میشود که در ادامه ی مسیر این روند ترجمه را به صورت کامل شرح خواهیم داد. نمونه ای از ترجمه خواسته شده را در شکل زیر مشاهده میکنید.

ورودی کامپایلر	خروجی کامپایلر
12925	Assign (OneTen_Two)Tou_NinHun_TwoTen_Fiv to t1 Print t1
2840*(106+5)	Assign OneHun_ZerTen_Six Plu Fiv to t1 Assign (Two)Tou_EigHun_FouTen_Zer Mul t1 to t2 Print t2

وظیفه ی بخش lexical آن است که توکن ورودی را هر وقت که syntax analyzer بخواهد (آن را صدا بزند) در اختیار آن قرار دهد. در این پروژه فقط نوع فقط یک نوع توکن وجود دارد که regular expression متناظر با این توکن را در ادامه بررسی خواهیم کرد و همچنین در قسمت lexical analyzer ، توکن ورودی را که عدد است (int) به صورت رشته حرفی که در جدول زیر مشاهده میکنید باید تبدیل شود (نه در قسمت syntax analyzer) و به syntax analyzer پاس داده شود.

Zer	0		Sev	7	
One	1		Eig	8	
Two	2		Nin	9	
Thr	3		Ten	10	
Fou	4		Hun	100	
Fiv	5		Tou	1000	
Six	6				

syntax analyzer

رشته ورودی ما به صورت دنباله ای از عملوند ها و عملگر های اولویت دار است که گرامر متناظر با آن را میتوان هم به صورت مبهم و هم به صورت غیر مبهم جهت عملیات parse به دلیل امکاناتی که ابزار yacc در اختیار ما قرار میدهد تعریف کرد. ما از گرامر غیر مبهم تولید کننده ی این رشته از اعداد استفاده کرده ایم. به دلیل نیاز به ترجمه ی رشته ی ورودی و دانستن این که چه زمانی به ریشه ی parse tree رسیدیم ، ما باید گرامر موجود را augmented کنیم.

expr_p -> expr
expr -> expr + term | expr – term | term
term -> term * fact | term / fact | fact
fact -> (expr) | NUMBER

گرامر بالا گرامر ذکر شده در کتاب جهت عمل جمع ، تفریق ، ضرب و تقسیم اعداد است. جهت عمل ترجمه همزمان با parse کردن نیاز داریم که گرامر فوق را به صورت syntax definition translation بازنویسی کنیم.

expr_p -> expr	{ if(count==1){printf("Assign %s to t%d\n", expr.val ,count);printf("Print t%d\n",count);}else printf ("Print %s\n", expr.val); }
expr -> expr1 + term	{ printf ("Assign %s Plu %s to t%d\n" , expr1.val , term.val , count);char buffer[6]; sprintf(buffer,"t%d",count);count++; expr.val = buffer ;}
expr -> expr1 - term	{ printf ("Assign %s Min %s to t%d\n" , expr1.val , term.val , count);char buffer[6]; sprintf(buffer,"t%d",count);count++; expr.val = buffer ;}
expr -> term	{ expr.val = term.val; }
term -> term1 '*' fact	{ printf ("Assign %s Mul %s to t%d\n" , term1.val , fact.val , count);char buffer[6]; sprintf(buffer,"t%d",count);count++; term.val = buffer ;}
term -> term1 '/' fact	{ printf ("Assign %s Div %s to t%d\n" , term1.val , fact.val , count);char buffer[6]; sprintf(buffer,"t%d",count);count++; term.val = buffer ;}
term -> fact	{ term.val = fact.val; }
fact -> '(' expr ')'	{ fact.val = expr.val; }
fact -> NUMBER	{ fact.val = NUMBER.lexeme; }

در production هایی که هیچ عملیات محاسباتی اتفاق نمی افتد تنها کافیست attribute پسر را به پدر انتقال دهیم اما در production هایی که عملیات محاسباتی در آن اتفاق می افتد نیاز به آن است که متغیر جدیدی به نام t_{count} ایجاد کنیم که count نشانگر تعداد assignment های اعمال شده تا به این لحظه میباشد و آن را به attribute پدر انتساب میدهیم و همچنین assignment حاصل را چاپ میکنیم. یادآور میشویم که NUMBER.lexeme مقدار عددی ندارد و حاوی رشته ی حرفی است که در بخش lexical تولید شده است.

Lexical analyzer

همانطور که در گرامر بالا مشاهده کردید تنها توکن موجود NUMBER است که regular expression آن به شرح زیر است. توجه داشته باشید به هنگام match شدن رشته ی ورودی با توکن NUMBER نیاز داریم که آن را به رشته ی حرفی که در صورت پروژه تعریف شده است تبدیل کنیم که جزییات آن را در قسمت پیاده سازی پروژه به طور کامل شرح میدهیم.

$$digit \rightarrow [0\text{-}9]$$
$$digit\_nz \rightarrow [1\text{-}9]$$
$$digits \rightarrow digit\_nz\ digit^*$$

پیاده سازی فایل lex.l

جهت استفاده از توابعی مانند printf و sprintf نیاز به کتابخانه ی stdio داریم که آن را در بخش declaration فایل lex.l include میکنیم. همچنین هدر y.tab.h که از اجرای دستور yacc –d yacc.y ایجاد شده است را در همین قسمت include میکنیم.

```
%{
#include <stdio.h>
#include "y.tab.h"

void buildNumber(char* yytext, int yyleng);
char _buffer[42];
%}
```

سپس باید regular expression ای که در بالا تعریف کردیم را در فایل lex.l تعریف کنیم.

```
Number [1-9][0-9]*
```

اکنون باید translation rule هایی را تعریف کنیم تا به هنگام match شدن رشته ی ورودی با عملگرها و توکن NUMBER ، عملیاتی که مورد نظر است را اعمال کنیم.

```
%%
`
"+"     {return '+';}
"-"     {return '-';}
"*"     {return '*';}
"/"     {return '/';}
"("     {return '(';}
")"     {return ')';}

{Number}        { buildNumber(yytext,yyleng); yylval.sval = strdup(_buffer); return NUMBER;}


%%
```

به هنگام match شدن با عملگرها کافیست خودش را جهت استفاده در parser بازگردانیم اما به هنگام match شدن رشته ورودی با توکن NUMBER ، نیاز داریم که علاوه بر برگرداندن نام توکن باید attribute value (lexeme) که رشته ی حرفی متناظر با عدد است را نیز برگردانیم. این رشته ی حرفی توسط تابع builldNumber تولید میشود و yytext و yyleng که ظول رشته ی match شده است را به عنوان ورودی به این تابع میدهیم و این تابع خروجی مورد نظر که تبدیل شده ی رشته ی ورودی به رشته ی حرفی مدنظر است را در متغیر buffer_ که در بخش declaration این فایل تعریف شده است ، قرار میدهد. حداکثر طول رشته ی تولید شده توسط تابع buildNumber برابر 41 است که نیاز داریم یک 0 در انتهای رشته ی خروجی تولید شده قرار دهیم. در نتیجه نیاز داریم که طول متغیر buffer_ که از جنس * char است را برابر 42 در نظر بگیریم. سپس باید مقدار buffer_ را در yylval.sval که همان attribute value (lexeme) قرار دهیم.

```
%%

void numberString (char number,char * buffer){
    switch (number) {
        case '0' : sprintf(buffer,"Zer");
            break;
        case '1' : sprintf(buffer,"One");
            break;
        case '2' : sprintf(buffer,"Two");
            break;
        case '3' : sprintf(buffer,"Thr");
            break;
        case '4' : sprintf(buffer,"Fou");
            break;
        case '5' : sprintf(buffer,"Fiv");
            break;
        case '6' : sprintf(buffer,"Six");
            break;
        case '7' : sprintf(buffer,"Sev");
            break;
        case '8' : sprintf(buffer,"Eig");
            break;
        case '9' : sprintf(buffer,"Nin");
            break;
    }
}
```

تابع buildNumber به شرح زیر است.

```
    char buffer_0[4];

    for (int i = 0; i < yyleng; i++) {

        switch (yyleng - i - 1) {

            case 0:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_0,"%s",buffer_temp);
                if (yyleng==3){
                    sprintf(_buffer,"%s_%s_%s",buffer_2,buffer_1,buffer_0);
                }
                else if (yyleng==2){
                    sprintf(_buffer,"%s_%s",buffer_1,buffer_0);
                }
                else if (yyleng==1) {
                    sprintf(_buffer,"%s",buffer_0);
                }
                else{
                    sprintf(_buffer,"%s_%s_%s_%s",buffer_sub1,buffer_2,buffer_1,buffer_0);
                }

                break;

            case 1:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_1,"%sTen",buffer_temp);
                break;

            case 2:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_2,"%sHun",buffer_temp);
                break;

            case 3:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_3,"%s",buffer_temp);
                if (yyleng==6){
                    sprintf(buffer_sub1,"(%s_%s_%s)Tou",buffer_5,buffer_4,buffer_3);
                }
                else if (yyleng==5){
                    sprintf(buffer_sub1,"(%s_%s)Tou",buffer_4,buffer_3);
                }
                else if (yyleng==4){
                    sprintf(buffer_sub1,"(%s)Tou",buffer_3);
                }
                break;

            case 4:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_4,"%sTen",buffer_temp);
                break;

            case 5:
                numberString(yytext[i],buffer_temp);
                sprintf(buffer_5,"%sHun",buffer_temp);
                break;
        }
    }
}
```

پیاده سازی yacc.y

همانند قسمت قبل کتابخانه ی stdio در قسمت declaration ، include میکنیم. همچنین تابع yylex() را جهت استفاده ی parser به صورت implicit در همین قسمت تعریف میکنیم.

همانطور که گفته شد متغیر count که تعداد assignment های اعمال شده را در خود ذخیره میکند را نیز در این قسمت تعریف میکنیم.

```
%{
#include <stdio.h>
void yyerror(char *);

int yylex();
int count=1;
%}
```

اکنون توکن NUMBER ، start symbol و جنس non terminal ها را تعریف میکنیم. توجه داشته باشید که اگر گرامر ما مبهم باشد ، باید اولویت و شرکت پذیری non terminal ها را در همین قسمت مشخص کنیم.

```
%union {
        char* sval;
}
%start expr_p
%token <sval> NUMBER
%left '+' '-'
%left '*' '/'
%type<sval> expr_p expr term fact
```

اکنون sdt تعریف شده را باید به فایل yacc.y اضافه کنیم. همچنین باید تابع yyparse() جهت شروع کردن عملیات parse در تابع main() صدا بزنیم.

```
%%

expr_p : expr           { if(count==1){printf("Assign %s to t%d\n",$1,count);printf("Print t%d\n",count);}else printf ( "Print %s\n",$1 ); }


expr :  expr '+' term   { printf  ( "Assign %s Plu %s to t%d\n" , $1 , $3 , count );char buffer[6]; sprintf(buffer,"t%d",count);count++; $$ = buffer ;}
|       expr '-' term   { printf  ( "Assign %s Min %s to t%d\n" , $1 , $3 , count );char buffer[6]; sprintf(buffer,"t%d",count);count++; $$ = buffer ;}
|       term            { $$ = $1; }
;
term :  term '*' fact   { printf ( "Assign %s Mul %s to t%d\n" , $1 , $3 , count );char buffer[6]; sprintf(buffer,"t%d",count);count++; $$ = buffer ;}
|       term '/' fact   { printf ( "Assign %s Div %s to t%d\n" , $1 , $3 , count );char buffer[6]; sprintf(buffer,"t%d",count);count++; $$ = buffer ;}
|       fact            { $$ = $1; }
;
fact :  '('  expr  ')'  { $$ = $2; }
|       NUMBER          { $$ = $1; }
;

%%

void yyerror(char * s){
        fprintf(stderr, "%s\n" , s);
}

int main(void){
        yyparse();
        return 0;
```

تست یک نمونه

فایل های اولیه به صورت زیر است که در فایل input.txt مقدار ورودی جهت ترجمه نوشته شده است.

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ ls
input.txt       lex.l           yacc.y
```

ابتدا دستور yacc –d yacc.y را اجرا میکنیم.

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ yacc -d yacc.y
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ ls
input.txt       lex.l           y.tab.c         y.tab.h         yacc.y
```

سپس دستور lex lex.l را اجرا میکنیم.

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ lex lex.l
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ ls
input.txt       lex.l           lex.yy.c        y.tab.c         y.tab.h         yacc.y
```

سپس فایل های به زبان C ایجاد شده را با استفاده از کامپایلر C کامپایل میکنیم.

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ gcc lex.yy.c y.tab.c -ll
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ ls
a.out           input.txt       lex.l           lex.yy.c        y.tab.c         y.tab.h         yacc.y
```

و در نهایت input.txt را به فایل خروجی تولید شده ی a.out به عنوان ورودی میدهیم.

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ cat input.txt
(123456*(((375+99)/(1100-2))))/22256
```

```
Amins-MacBook-Pro:test aminhasanzadehmoghadam$ ./a.out < input.txt
Assign ThrHun_SevTen_Fiv Plu NinTen_Nin to t1
Assign (One)Tou_OneHun_ZerTen_Zer Min Two to t2
Assign t1 Div t2 to t3
Assign (OneHun_TwoTen_Thr)Tou_FouHun_FivTen_Six Mul t3 to t4
Assign t4 Div (TwoTen_Two)Tou_TwoHun_FivTen_Six to t5

Print t5
```