Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VI

14 April 2021

UCS 1602 - Compiler Design

Exercise 7: Generation of Intermediate Code Using Lex and Yacc

Aim:

The new Language Pascal-2021 is introduced with the following programming constructs:
Datatypes

- Integer
- Real
- Char

Declaration Statement

- var: type;
- var: type = constant;

Conditional Statement

- if condition then —– else —– endif

Generate Intermediate code in the form of Three Address Code sequence for the sample input program written using declaration, conditional and assignment statements in new language Pascal-2021.

Code - Yacc Parser File:

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <math.h>

    int yylex(void);
    int yyerror(char *);
    int yywrap();

    int vars = 0, labels = 0;

    struct info{
        char *var;
        char *code;
        int intval;
        float floatval;
        char charval;
    };

    typedef struct info node;

    node *makeNode(){
        //creating a new node to store intermediate code

        node *n = (node *)malloc(sizeof(node));
        n->intval = 0;
        n->floatval = 0;
        n->charval = 0;
        n->var = (char *)malloc(50 * sizeof(char));
        n->code = (char *)malloc(5000 * sizeof(char));

        return n;
    }
%}

/*Declaration of tokens and precedence*/
%token BGN END IF THEN ELSE INT CHAR
%token REAL CHCONST VAR NUM RELOP ADDOP MULOP

/*Increasing precedence*/
%right MULOP
%left ADDOP

/*Declaration of the types that YYSTYPE can take with the union*/
%union{
    int intval;
```

```
48      float floatval;
49      char ch;
50      char *str;
51      struct info *Node;
52  }
53
54  /*Declaring types for the tokens*/
55  %type<str>      VAR RELOP ADDOP MULOP
56  %type<intval>   NUM
57  %type<floatval> REAL
58  %type<ch>       CHCONST
59  %type<Node>     Program Structure Declarations Statements
60  %type<Node>     Declaration Type Value Statement
61  %type<Node>     Assignment Conditional Condition Expr
62  %type<Node>     E T F
63
64  %%
65
66  Program         :   Structure{
67                          printf("\nL%-5d - |\n%s", 0, $$->code);
68                      }
69                  ;
70
71  Structure       :   Declarations BGN Statements END{
72                          sprintf($$->code, "%s%10s\n%s", $1->code, "|", $3
        ->code);
73                      }
74                  ;
75
76  Declarations    :   Declaration Declarations{
77                          $$ = makeNode();
78                          sprintf($$->code, "%s%s", $1->code, $2->code);
79                      }
80
81                  |   Declaration{
82                          $$ = $1;
83                      }
84                  ;
85
86  Declaration     :   VAR ':' Type ';' {
87                          $$ = makeNode();
88                          sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $3
        ->var);
89                      }
90
91                  |   VAR ':' Type '=' Value ';'{
92                          $$ = makeNode();
93                          sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $5
        ->var);
94                      }
95                  ;
```

```
96
97  Type            :   INT{
98                          $$ = makeNode();
99                          $$->intval = 0;
100                         sprintf($$->var, "%d", 0);
101                         sprintf($$->code, "");
102                     }
103
104                 |   REAL{
105                         $$ = makeNode();
106                         $$->floatval = 0.0;
107                         sprintf($$->var, "%.2f", 0.0);
108                         sprintf($$->code, "");
109                     }
110
111                 |   CHAR{
112                         $$ = makeNode();
113                         $$->charval = 0;
114                         sprintf($$->var, "%s", "NULL");
115                         sprintf($$->code, "");
116                     }
117                 ;
118
119 Value           :   NUM{
120                         $$ = makeNode();
121                         $$->intval = $1;
122                         sprintf($$->var, "%d", $1);
123                         sprintf($$->code, "");
124                     }
125
126                 |   REAL{
127                         $$ = makeNode();
128                         $$->floatval = $1;
129                         sprintf($$->var, "%.2f", $1);
130                         sprintf($$->code, "");
131                     }
132
133                 |   CHCONST{
134                         $$ = makeNode();
135                         $$->intval = $1;
136                         sprintf($$->var, "%c", $1);
137                         sprintf($$->code, "");
138                     }
139                 ;
140
141 Statements      :   Statement Statements{
142                         $$ = makeNode();
143                         sprintf($$->code, "%s%s", $1->code, $2->code);
144                     }
145
146                 |   Statement{
```

```
147                         $$ = $1;
148                     }
149                 ;
150
151 Statement       :   Assignment {
152                         $$ = $1;
153                     }
154
155                 |   Conditional{
156                         $$ = $1;
157                     }
158                 ;
159
160 Assignment      :   VAR '=' Expr ';'{
161                         $$ = makeNode();
162                         char tac[100];
163                         sprintf($$->var, "%s", $1);
164                         sprintf(tac, "%10s %-5s := %s\n", "|", $$->var, $3
    ->var);
165                         sprintf($$->code, "%s%s", $3->code, tac);
166                     }
167                 ;
168
169 Expr            :   E{
170                         $$ = $1;
171                     }
172                 ;
173
174 E               :   T MULOP E{
175                         $$ = makeNode();
176                         char tac[100];
177                         sprintf($$->var, "x%d", ++vars);
178                         sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->
    var, $1->var, $2, $3->var);
179                         sprintf($$->code, "%s%s%s", $1->code, $3->code,
    tac);
180                    }
181
182                 |   T{
183                         $$ = $1;
184                     }
185
186                 |   F{
187                         $$ = $1;
188                     }
189                 ;
190
191 T               :   T ADDOP F{
192                         $$ = makeNode();
193                         char tac[100];
194                         sprintf($$->var, "x%d", ++vars);
```

```
195                         sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->
    var, $1->var, $2, $3->var);
196                         sprintf($$->code, "%s%s%s", $1->code, $3->code,
    tac);
197                     }
198
199                 |   F{
200                         $$ = $1;
201                     }
202                 ;
203
204 F               :   VAR{
205                         $$ = makeNode();
206                         sprintf($$->var, "%s", $1);
207                         sprintf($$->code, "");
208                     }
209
210                 |   NUM{
211                         $$ = makeNode();
212                         $$->intval = $1;
213                         sprintf($$->var, "%d", $1);
214                         sprintf($$->code, "");
215                     }
216
217                 |   REAL{
218                         $$ = makeNode();
219                         $$->floatval = $1;
220                         sprintf($$->var, "%.2f", $1);
221                         sprintf($$->code, "");
222                     }
223
224                 |   CHCONST{
225                         $$ = makeNode();
226                         $$->charval = $1;
227                         sprintf($$->var, "'%c'", $1);
228                         sprintf($$->code, "");
229                     }
230                 ;
231
232 Conditional     :   IF '(' Condition ')' THEN Statements ELSE Statements
    END IF{
233                         $$ = makeNode();
234                         int condnBlock = ++labels;
235                         int endBlock = ++labels;
236                         sprintf($$->code, "%s%10s if %s then goto L%d\n%s
    %10s goto L%d\n%10s\nL%-5d - |\n%s%10s\nL%-5d - |\n", $3->code, "|", $3
    ->var, condnBlock, $8->code, "|", endBlock, "|", condnBlock, $6->code,
    "|", endBlock);
237                     }
238                 ;
239
```

```
Condition       :   Expr RELOP Expr{
                        $$ = makeNode();
                        char tac[100];
                        sprintf($$->var, "%s%s%s", $1->var, $2, $3->var);
                        sprintf($$->code, "%s%s", $1->code, $3->code);
                    }
                ;
%%

int yyerror(char* str){
    printf("\n%s", str);
    return 0;
}

int yywrap(){
    return 1;
}

int main(){
    printf("\n\t\tIntermediate Code Generation\n");
    printf("\nYour Code:\n\n");
    system("cat Code.txt");
    printf("\n\nThree Address Code:\n");

    yyparse();
    return 0;
}

/*
Usage:

yacc -d -Wnone TAC.y
lex TAC.l
gcc y.tab.c lex.yy.c -w
./a.out < Code.txt

*/
```

Code - Lex Grammar File:

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

term    ([a-zA-Z\_][a-zA-Z\_0-9]*)
num     ([0-9]+)
real    {num}\.{num}
relop   ("<"|"<="|">"|">="|"=="|"!=")
addop   ("+"|"-")
mulop   ("*"|"/"|"%")
spl     (";"|","|"{"|"}"|"("|")"|"="|"&"|"|"|"!"|":")

%%
"begin"     {return BGN;}
"end"       {return END;}
"if"        {return IF;}
"then"      {return THEN;}
"else"      {return ELSE;}
"integer"   {return INT;}
"char"      {return CHAR;}
"real"      {return REAL;}
['].['] {yylval.ch = yytext[1]; return CHCONST;}
{term}      {yylval.str = strdup(yytext); return VAR;}
{real}      {yylval.floatval = atof(yytext); return REAL;}
{num}       {yylval.intval = atoi(yytext); return NUM;}
{relop}     {yylval.str = strdup(yytext); return RELOP;}
{mulop}     {yylval.str = strdup(yytext); return MULOP;}
{addop}     {yylval.str = strdup(yytext); return ADDOP;}
{spl}       {return *yytext;}
[ \t\n]+    {;}

.           {char errmsg[100];
             strcpy(errmsg, "Invalid Character: ");
             strcat(errmsg, yytext);
             strcat(errmsg, "\n");
             yyerror(errmsg);}

%%
```

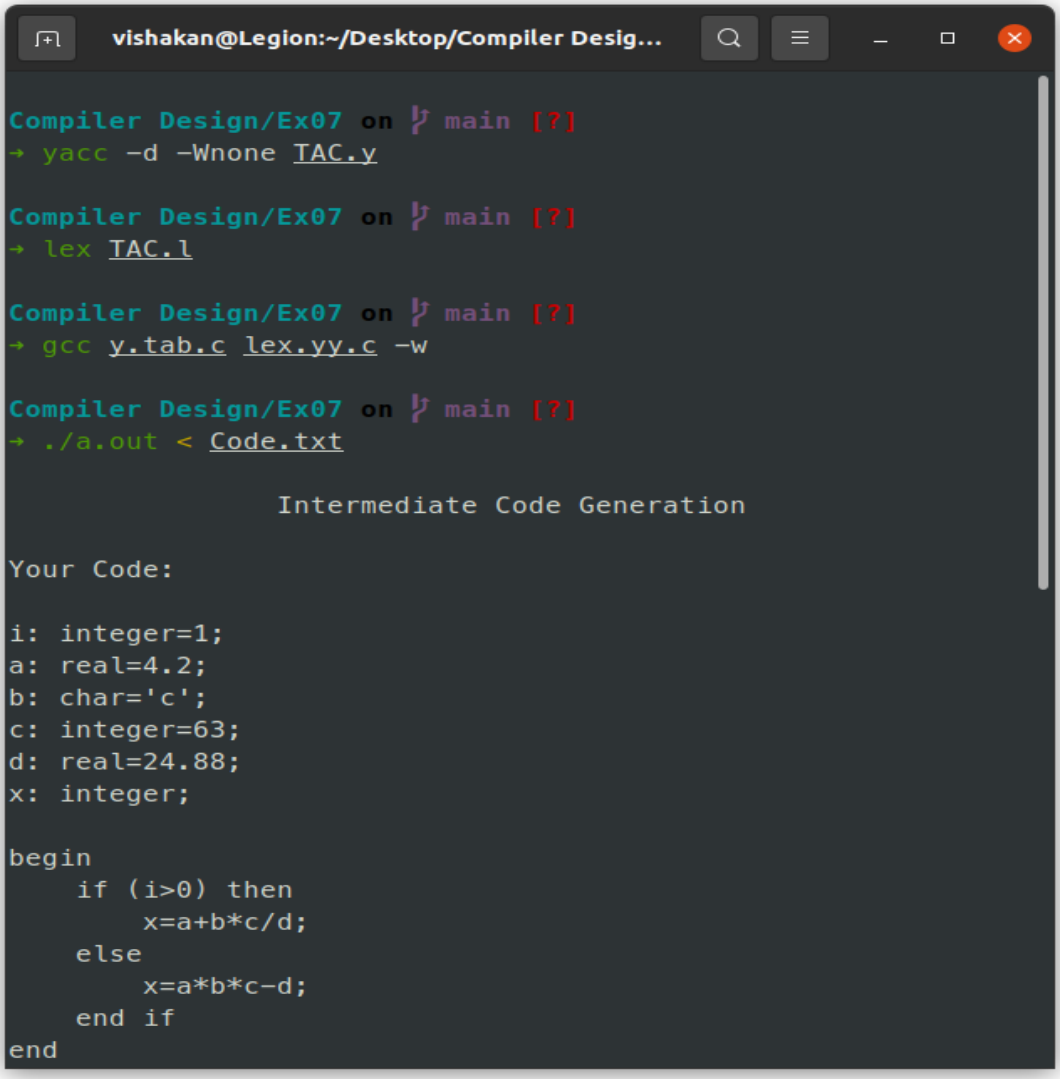
Sample - Parsed Pascal-2021 Code:

```
i: integer=1;
a: real=4.2;
b: char='c';
c: integer=63;
d: real=24.88;
x: integer;

begin
    if (i>0) then
        x=a+b*c/d;
    else
        x=a*b*c-d;
    end if
end
```

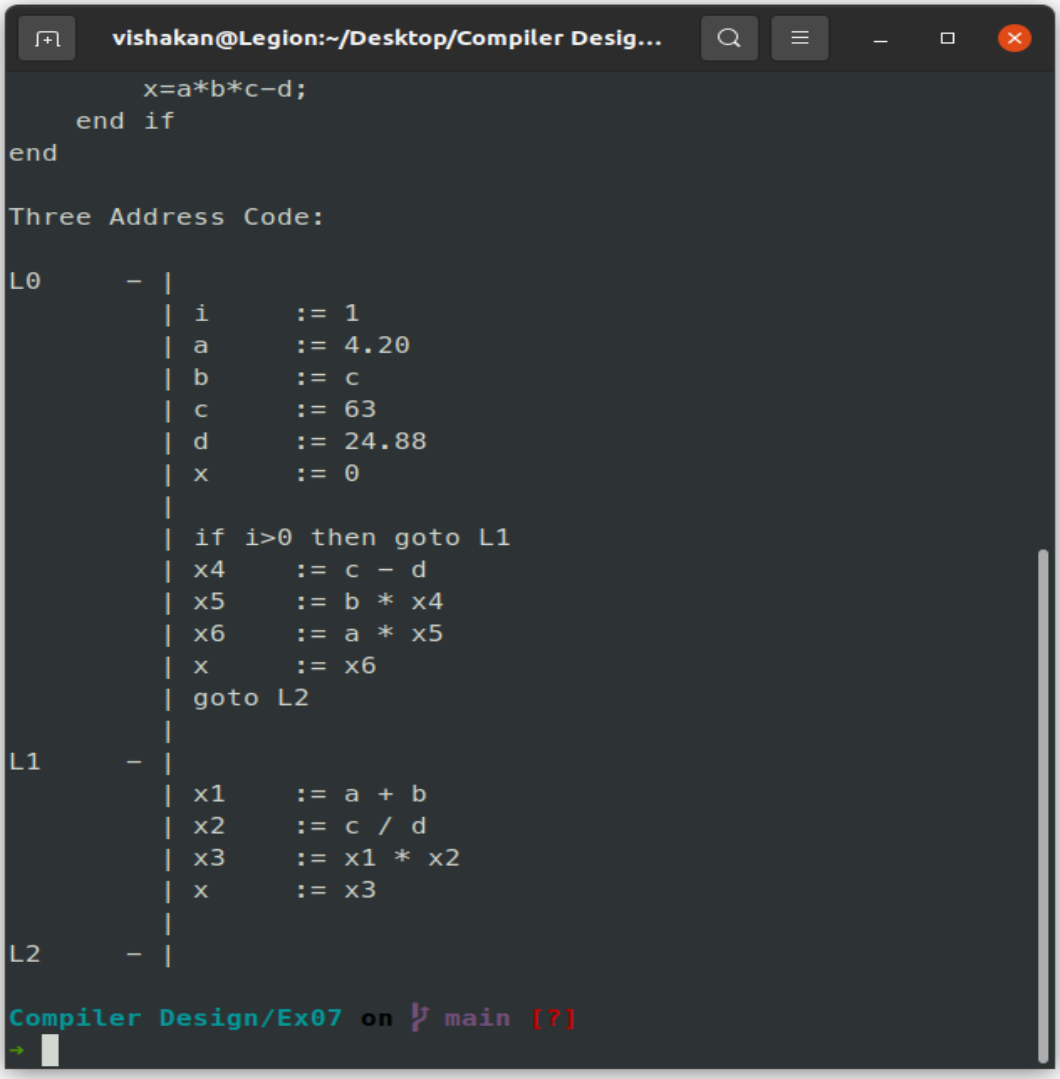
Output 1 - Compilation & Code:

Figure 1: Console Output - Compilation & Code.



Output 2 - Intermediate Code:

Figure 2: Console Output - Intermediate Code.



Learning Outcome:

- I learnt more theory behind **Yacc Parser Generator**.
- I understood how to construct a grammar for a basic syntax checker.
- I learnt that grammar can be built upon layer by layer, each one adding more detail and complexity.
- I was able to implement the required token recognition with Lex tool.
- I was able to implement the required intermediate code generator with the Yacc tool and Lex tool.
- I understood the use of the %union declaration for **yylval**'s types for passing different values from Lex to Yacc.
- I declared a custom structure to store intermediate code and variables/values and assigned them values while parsing the respective grammar using the $$ operator of Yacc.
- I made use of the **sprintf()** function to create intermediate code conveniently.
- I understood that precedences can only be given to tokens in Yacc, and not for grammar symbols.
- I was able to construct intermediate code for conditional block with appropriate grammar definition.
- I came to know that there was no need to return the structures I created inside the parsing of a lower grammar to pass it up to the higher grammar, as it gets implicitly passed up and can be called with the $ operator.
- I understood that subtle grammar differences need to be made in the Yacc grammar definition to work for right and left associativity & precedences.
- I learnt to call yyerror() with a custom error message within the Lex code.