

Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester VI

16 April 2021

UCS1602 - Compiler Design

Exercise 7: Generation of Intermediate Code using Lex and Yacc

Objective:

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:

Lex:

```

1  %{
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      #include "y.tab.h"
6  %}
7  %option yylineno
8
9  num [0-9]+
10 real {num}\.{num}
11
12 if if
13 else else
14 then then
15 begin begin
16 end end
17
18 rel_op ("<"|"<="|">"|">="|"=="|"!=")
19 add_op ("+"|" -")
20 mul_op ("*"|"/"|"%")
21 assn_op ("+="|" -="|" *="|" /="|" =")
22
23 id [a-z][a-z]*
24 spl (";"|", "|" "{" |"}"|"(" |")"|"="|"&"|"|"|"!"|" ":"")
25
26 %%
27 {num} {yyval.int_val = atoi(yytext);return INT_CONST;}
28 {real} {yyval.float_val = atof(yytext);return REAL_CONST;}
29 ['].['] {yyval.char_val = yytext[1];return CHAR_CONST;}
30
31 "integer" {return INT;}
32 "real" {return REAL;}
33 "char" {return CHAR;}
34
35
36 "(" {return POPEN;}
37 ")" {return PCLOSE;}
38
39 {if} {return IF;}
40 {else} {return ELSE;}
41 {then} {return THEN;}
42 {begin} {return BGN;}
43 {end} {return END;}
44
45 {rel_op} {yyval.str = strdup(yytext); return REL_OP;}

```

```

46 {mul_op} {yyval.str = strdup(yytext); return MUL_OP;}
47 {add_op} {yyval.str = strdup(yytext); return ADD_OP;}
48
49 {id} {yyval.str = strdup(yytext);return ID;}
50 {spl} {return *yytext;}
51 [\t\n]+  {};
52 " "  {};
53 . {
54     char errmsg[100];
55     sprintf(errmsg, "Invalid Character: %s at line %d",
yytext, yylineno);
56     strcat(errmsg, "\n");
57     yyerror(errmsg);
58 }
59 %%

```

Yacc:

```

1  %{
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      #include <math.h>
6
7      int yylex(void);
8      int yyerror(char *);
9      int yywrap();
10
11     int tmp = 0;
12     int jump = 0;
13
14     struct info{
15         char *var;
16         char *code;
17         int int_val;
18         float float_val;
19         char char_val;
20     };
21
22     typedef struct info node;
23
24     node *makeNode(){

```

```

25         node *n = (node*)calloc(1, sizeof(node));
26         n->int_val = 0;
27         n->float_val = 0;
28         n->char_val = 0;
29         n->var = (char*)calloc(50, sizeof(char));
30         n->code = (char*)calloc(5000, sizeof(char));
31         return n;
32     }
33 %}
34
35 %token BGN END
36 %token INT REAL CHAR
37 %token INT_CONST REAL_CONST CHAR_CONST
38 %token ID
39 %token IF ELSE THEN REL_OP
40 %token POPEN PCLOSE
41 %token MUL_OP ADD_OP
42
43 %right MUL_OP
44 %left ADD_OP
45
46 %union{
47     int int_val;
48     float float_val;
49     char char_val;
50     char *str;
51     struct info *Node;
52 }
53
54 /*Declaring types for the tokens*/
55 %type<str> ID REL_OP ADD_OP MUL_OP
56 %type<int_val> INT_CONST
57 %type<float_val> REAL_CONST
58 %type<char_val> CHAR_CONST
59 %type<Node> program structure decl_stmts stmts
60 %type<Node> decl_stmt type value stmt
61 %type<Node> assn_stmt cond_stmt 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 : decl_stmts BGN stmts END{
72     sprintf($$->code, "%s%10s\n%s", $1->code,
73         "|", $3->code);
74 }
75 ;
76 decl_stmts : decl_stmt decl_stmts{
77     $$ = makeNode();
78     sprintf($$->code, "%s%s", $1->code, $2->code)
79 ;
80     }
81     | decl_stmt{
82         $$ = $1;
83     }
84 ;
85
86 decl_stmt : ID ':' type ';' {
87     $$ = makeNode();
88     sprintf($$->code, "%10s %-5s := %s\n", "|",
89         $1, $3->var);
90     }
91     | ID ':' type '=' value ';' {
92         $$ = makeNode();
93         sprintf($$->code, "%10s %-5s := %s\n", "|",
94             $1, $5->var);
95     }
96 ;
97 type : INT{
98     $$ = makeNode();
99     $$->int_val = 0;
100     sprintf($$->var, "%d", 0);
101     sprintf($$->code, "");
102 }
103
104 | REAL{
105     $$ = makeNode();
106     $$->float_val = 0.0;
107     sprintf($$->var, "%.2f", 0.0);
108     sprintf($$->code, "");
109 }
110

```

```

111 | CHAR{
112     $$ = makeNode();
113     $$->char_val = 0;
114     sprintf($$->var, "%s", "NULL");
115     sprintf($$->code, "");
116 }
117 ;
118
119 value : INT_CONST{
120     $$ = makeNode();
121     $$->int_val = $1;
122     sprintf($$->var, "%d", $1);
123     sprintf($$->code, "");
124 }
125 | REAL_CONST{
126     $$ = makeNode();
127     $$->float_val = $1;
128     sprintf($$->var, "%.2f", $1);
129     sprintf($$->code, "");
130 }
131 | CHAR_CONST{
132     $$ = makeNode();
133     $$->int_val = $1;
134     sprintf($$->var, "%c", $1);
135     sprintf($$->code, "");
136 }
137 ;
138
139 stmts : stmt stmts{
140     $$ = makeNode();
141     sprintf($$->code, "%s%s", $1->code, $2->code);
142 }
143 | stmt{
144     $$ = $1;
145 }
146 ;
147
148 stmt : assn_stmt {
149     $$ = $1;
150 }
151 | cond_stmt{
152     $$ = $1;
153 }
154 ;
155

```

```

156 assn_stmt : ID '=' expr ';' {
157     $$ = makeNode();
158     char tac[100];
159     sprintf($$->var, "%s", $1);
160     sprintf(tac, "%10s %-5s := %s\n", "|", $$->
    var, $3->var);
161     sprintf($$->code, "%s%s", $3->code, tac);
162     }
163 ;
164
165 expr : E{
166     $$ = $1;
167     }
168 ;
169
170 E : T MUL_OP E{
171     $$ = makeNode();
172     char tac[100];
173     sprintf($$->var, "x%d", ++tmp);
174     sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var,
    $1->var, $2, $3->var);
175     sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);
176     }
177 | T{
178     $$ = $1;
179     }
180 | F{
181     $$ = $1;
182     }
183 ;
184
185 T : T ADD_OP F{
186     $$ = makeNode();
187     char tac[100];
188     sprintf($$->var, "x%d", ++tmp);
189     sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var,
    $1->var, $2, $3->var);
190     sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);
191     }
192 | F{
193     $$ = $1;
194     }
195 ;
196
197 F : ID{

```

```

198         $$ = makeNode();
199         sprintf($$->var, "%s", $1);
200         sprintf($$->code, "");
201     }
202     | INT_CONST{
203         $$ = makeNode();
204         $$->int_val = $1;
205         sprintf($$->var, "%d", $1);
206         sprintf($$->code, "");
207     }
208     | REAL_CONST{
209         $$ = makeNode();
210         $$->float_val = $1;
211         sprintf($$->var, "%.2f", $1);
212         sprintf($$->code, "");
213     }
214     | CHAR_CONST{
215         $$ = makeNode();
216         $$->char_val = $1;
217         sprintf($$->var, "'%c'", $1);
218         sprintf($$->code, "");
219     }
220 ;
221
222 cond_stmt : IF POPEN condition PCLOSE THEN stmts ELSE stmts
223           END IF{
224             $$ = makeNode();
225             int condBlock = ++jump;
226             int endBlock = ++jump;
227             sprintf($$->code, "%s%10s if %s then goto L%d\n%s%10s goto L%d\n%10s\nL%-5d - |\n%s%10s\nL%-5d - |\n",
228                 $3->code, "|", $3->var, condBlock, $8->code, "|", endBlock
229                 , "|", condBlock, $6->code, "|", endBlock);
230         }
231 ;
232
233 condition : expr REL_OP expr{
234         $$ = makeNode();
235         char tac[100];
236         sprintf($$->var, "%s%s%s", $1->var, $2, $3->var);
237         sprintf($$->code, "%s%s", $1->code, $3->code);
238     }
239 ;
240 %%

```



```

239 int yyerror(char* str){
240     printf("\n%s", str);
241     return 0;
242 }
243
244 int yywrap(){
245     return 1;
246 }
247
248 int main(){
249     printf("\nGiven code\n");
250     system("cat file.txt");
251     printf("\n
-----
n");
252     printf("\nThree Address Code\n");
253
254     yyparse();
255     return 0;
256 }

```

Input:

```
Given code
i:integer=1;
a:integer=4;
b:integer=3;
c:integer=6;
d:integer=2;
x:integer;
begin
if (i>0) then
  x=a+b*c/d;
else
  x=a*b*c-d;
end if
end
```

Output:

Three Address Code

```
L0      - |
          | i      := 1
          | a      := 4
          | b      := 3
          | c      := 6
          | d      := 2
          | x      := 0
          |
          | if i>0 then goto L1
          | x4     := c - d
          | x5     := b * x4
          | x6     := a * x5
          | x      := x6
          | goto L2
L1      - |
          | x1     := a + b
          | x2     := c / d
          | x3     := x1 * x2
          | x      := x3
L2      - |
```

Learning Outcomes:

- Understood the basic idea of Three Address Code.
 - Learnt how to identify control structures and write TAC for them.
 - Learnt to use yacc efficiently for string concatenation, and hence generate code.
-