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:

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

num [0-9]+
real {num}\.{num}

if if
else else
then then
begin begin
end end

rel_op ("<"|"<="|">"|">="|"=="|"!=")
add_op ("+"|"-")
mul_op ("*"|"/"|"%")
assn_op ("+="|"-="|"*="|"/="|"=")

id [a-z][a-z]*
spl (";"|","|"{"|"}"|"("|")"|"="|"&"|"|"|"!"|":")

%%
{num} {yylval.int_val = atoi(yytext);return INT_CONST;}
{real} {yylval.float_val = atof(yytext);return REAL_CONST;}
['].['] {yylval.char_val = yytext[1];return CHAR_CONST;}

"integer" {return INT;}
"real" {return REAL;}
"char" {return CHAR;}


"(" {return POPEN;}
")" {return PCLOSE;}

{if} {return IF;}
{else} {return ELSE;}
{then} {return THEN;}
{begin} {return BGN;}
{end} {return END;}

{rel_op} {yylval.str = strdup(yytext); return REL_OP;}
```

```
{mul_op} {yylval.str = strdup(yytext); return MUL_OP;}
{add_op} {yylval.str = strdup(yytext); return ADD_OP;}

{id} {yylval.str = strdup(yytext);return ID;}
{spl} {return *yytext;}
[\t\n]+  {;}
" " {;}
. {
    char errmsg[100];
    sprintf(errmsg, "Invalid Character: %s at line %d",
   yytext, yylineno);
    strcat(errmsg, "\n");
    yyerror(errmsg);
  }
%%
```

Yacc:

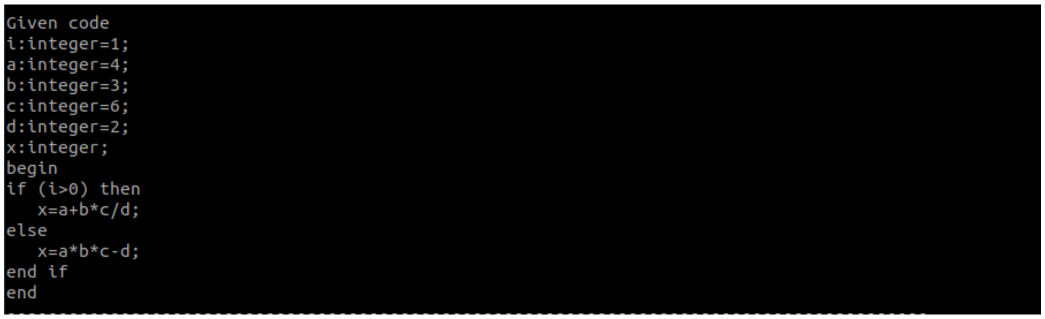
```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <math.h>

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

    int tmp = 0;
    int jump = 0;

    struct info{
        char *var;
        char *code;
        int int_val;
        float float_val;
        char char_val;
    };

    typedef struct info node;

    node *makeNode(){
        node *n = (node*)calloc(1, sizeof(node));
        n->int_val = 0;
        n->float_val = 0;
        n->char_val = 0;
        n->var = (char*)calloc(50, sizeof(char));
        n->code = (char*)calloc(5000, sizeof(char));
        return n;
    }
%}

%token BGN END
%token INT REAL CHAR
%token INT_CONST REAL_CONST CHAR_CONST
%token ID
%token IF ELSE THEN REL_OP
%token POPEN PCLOSE
%token MUL_OP ADD_OP

%right MUL_OP
%left ADD_OP

%union{
    int int_val;
    float float_val;
    char char_val;
    char *str;
    struct info *Node;
}

/*Declaring types for the tokens*/
%type<str> ID REL_OP ADD_OP MUL_OP
%type<int_val> INT_CONST
%type<float_val> REAL_CONST
%type<char_val> CHAR_CONST
%type<Node> program structure decl_stmts stmts
%type<Node> decl_stmt type value stmt
%type<Node> assn_stmt cond_stmt condition expr
%type<Node> E T F

%%

program : structure{
            printf("\nL%-5d - |\n%s", 0, $$->code);
          }
;
```

```

structure : decl_stmts BGN stmts END{
                sprintf($$->code, "%s%10s\n%s", $1->code, "|", $3->code);
            }
;

decl_stmts : decl_stmt decl_stmts{
                $$ = makeNode();
                sprintf($$->code, "%s%s", $1->code, $2->code);
             }

           | decl_stmt{
                $$ = $1;
             }
;

decl_stmt : ID ':' type ';' {
                $$ = makeNode();
                sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $3->var);
            }

          | ID ':' type '=' value ';'{
                $$ = makeNode();
                sprintf($$->code, "%10s %-5s := %s\n", "|", $1, $5->var);
            }
;

type : INT{
            $$ = makeNode();
            $$->int_val = 0;
            sprintf($$->var, "%d", 0);
            sprintf($$->code, "");
       }

     | REAL{
            $$ = makeNode();
            $$->float_val = 0.0;
            sprintf($$->var, "%.2f", 0.0);
            sprintf($$->code, "");
       }

     | CHAR{
            $$ = makeNode();
            $$->char_val = 0;
            sprintf($$->var, "%s", "NULL");
            sprintf($$->code, "");
       }
                 ;

value : INT_CONST{
            $$ = makeNode();
            $$->int_val = $1;
            sprintf($$->var, "%d", $1);
            sprintf($$->code, "");
        }
      | REAL_CONST{
            $$ = makeNode();
            $$->float_val = $1;
            sprintf($$->var, "%.2f", $1);
            sprintf($$->code, "");
        }
      | CHAR_CONST{
            $$ = makeNode();
            $$->int_val = $1;
            sprintf($$->var, "%c", $1);
            sprintf($$->code, "");
        }
;

stmts : stmt stmts{
            $$ = makeNode();
            sprintf($$->code, "%s%s", $1->code, $2->code);
        }
      | stmt{
            $$ = $1;
        }
;

stmt : assn_stmt {
        $$ = $1;
       }
     | cond_stmt{
            $$ = $1;
       }
;
```

```
assn_stmt : ID '=' expr ';'{
                $$ = makeNode();
                char tac[100];
                sprintf($$->var, "%s", $1);
                sprintf(tac, "%10s %-5s := %s\n", "|", $$->var, $3->var);
                sprintf($$->code, "%s%s", $3->code, tac);
            }
;

expr : E{
        $$ = $1;
       }
;

E : T MUL_OP E{
        $$ = makeNode();
        char tac[100];
        sprintf($$->var, "x%d", ++tmp);
        sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var, $1->var, $2, $3->var);
        sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);
    }
  | T{
        $$ = $1;
    }
  | F{
        $$ = $1;
    }
;

T : T ADD_OP F{
        $$ = makeNode();
        char tac[100];
        sprintf($$->var, "x%d", ++tmp);
        sprintf(tac, "%10s %-5s := %s %s %s\n", "|", $$->var, $1->var, $2, $3->var);
        sprintf($$->code, "%s%s%s", $1->code, $3->code, tac);
    }
  | F{
        $$ = $1;
    }
;

F : ID{
        $$ = makeNode();
        sprintf($$->var, "%s", $1);
        sprintf($$->code, "");
    }
    | INT_CONST{
            $$ = makeNode();
            $$->int_val = $1;
            sprintf($$->var, "%d", $1);
            sprintf($$->code, "");
      }
    | REAL_CONST{
            $$ = makeNode();
            $$->float_val = $1;
            sprintf($$->var, "%.2f", $1);
            sprintf($$->code, "");
      }
    | CHAR_CONST{
            $$ = makeNode();
            $$->char_val = $1;
            sprintf($$->var, "'%c'", $1);
            sprintf($$->code, "");
      }
;

cond_stmt : IF POPEN condition PCLOSE THEN stmts ELSE stmts
    END IF{
                $$ = makeNode();
                int condBlock = ++jump;
                int endBlock = ++jump;
                sprintf($$->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, condBlock, $8->code, "|", endBlock
    , "|", condBlock, $6->code, "|", endBlock);
            }
;

condition : expr REL_OP expr{
            $$ = makeNode();
            char tac[100];
            sprintf($$->var, "%s%s%s", $1->var, $2, $3->var);
            sprintf($$->code, "%s%s", $1->code, $3->code);
       }
;
%%

```

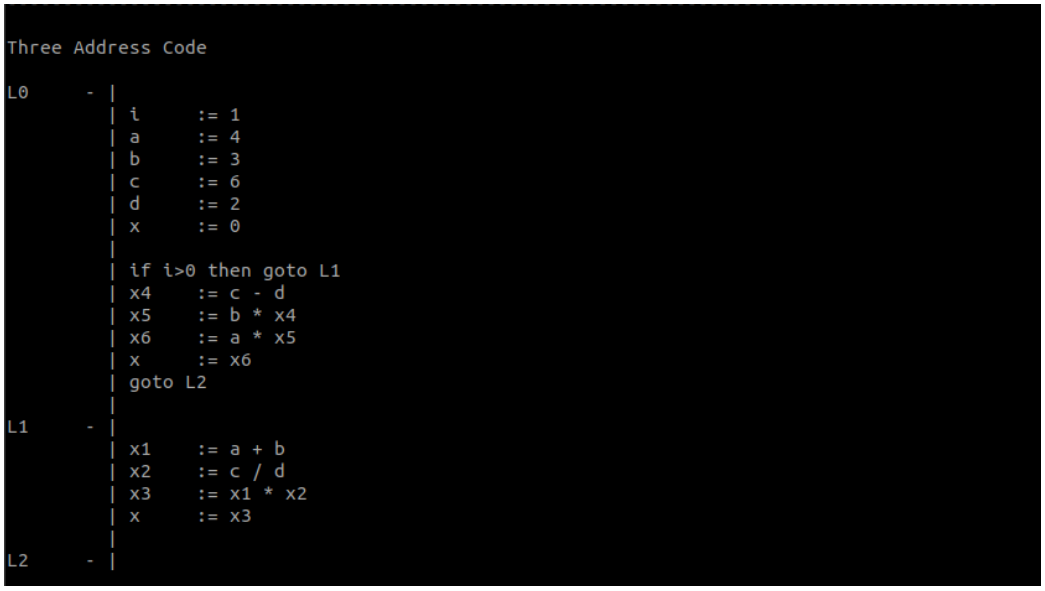
```
int yyerror(char* str){
    printf("\n%s", str);
    return 0;
}

int yywrap(){
    return 1;
}

int main(){
    printf("\nGiven code\n");
    system("cat file.txt");
    printf("\n
   -----------------------------------------------------------------------------
   n");
    printf("\nThree Address Code\n");

    yyparse();
    return 0;
}
```

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.