Assignment 4 – Generate intermediate code using Lex and Yacc Tools

Date: 05/05/2023

Aim:

To develop an intermediate code generator to generate three address code for the following statements by writing suitable syntax directed translation rules:

1. Assignment statements
2. Boolean expressions
3. Flow of control statements

Code:

```
/*ex4.l*/

%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
void yyerror(char*);
extern YYSTYPE yylval;
%}
digit [0-9]
letter [a-zA-Z]
identifier (_|{letter})(_|{digit}|{letter})*
relop "<"|">"|"<="|">="|"=="|"!="
space " "
%%
{space} { return *yytext; }
"="|"+"|"*"|"-" { return *yytext; }
"\n" { return *yytext; }
{relop} { yylval.string=strdup(yytext); return relop; }
"and" { return and; }
"or" { return or; }
"not" { return not; }
"true" { return truth; }
"false" { return falsity; }
{identifier} { yylval.string=strdup(yytext); return identifier;
}
%%
```

/*ex4.y*/

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "y.tab.h"
int yylex(void);
void yyerror(char*);
int yywrap(void);
int tempCount = 1;
int address = 100;
%}
%union {
char *string;
int num;
};
%token <string> identifier
%token and or not truth falsity
%token <string> relop
%type <string> E
%type <string> C
%left '+' '-'
%left '/' '*'
%nonassoc relop
%%
S: A | B ;
A:
identifier '=' E '\n' {
printf("%s = %s\n\n", $1, $3); return 0;
}
E:
E '+' E {
printf("%s%d = %s %s %s\n", "temp", tempCount, $1,
"+", $3);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
tempCount++;
}

| E '*' E {
```

```
printf("%s%d = %s %s %s\n", "temp", tempCount, $1,
"*", $3);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
tempCount++;
}

| '-' E {
printf("%s%d = %s %s\n", "temp", tempCount, "-",
$2);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
tempCount++;
}
| identifier {
$$ = strdup($1);
}
;
B: identifier '=' C '\n'{
printf("%d: %s = %s\n\n", address, $1, $3);
address++; return 0;
}
C:
C ' ' or ' ' C {
printf("%d: temp%d = %s or %s\n",
address, tempCount, $1, $5);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
address++;
tempCount++;
}
| C ' ' and ' ' C {
printf("%d: temp%d = %s and %s\n",
address, tempCount, $1, $5);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
address++;
tempCount++;
}
```

```
| not ' ' C {
printf("%d: temp%d = not %s\n", address,
tempCount, $3);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
address++;
tempCount++;
}
| identifier relop identifier {
printf("%d: if %s %s %s, goto %d\n",
address, $1, $2, $3, address + 3);
address++;
printf("%d: temp%d = 0\n", address,
tempCount);
address++;
printf("%d: goto %d\n", address, address
+ 2);
address++;
printf("%d: temp%d = 1\n", address,
tempCount);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
address++;
tempCount++;
}
| truth {
printf("%d: temp%d = 1\n", address,
tempCount);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
tempCount++;
address++;
}
| falsity {
printf("%d: temp%d = 0\n", address,
tempCount);
char temp[10];
sprintf(temp, "temp%d", tempCount);
$$ = strdup(temp);
tempCount++;
```

```
address++;
}
%%
int yywrap(){
return 1;
}
void yyerror(char* msg){
fprintf(stderr, "%s", msg);
return;
}
int main(){
printf("Enter expression: ");
yyparse();
return 0;
}

//with flow control
/*ex4.l*/
%{
   #include<stdio.h>
   #include<stdlib.h>
   #include<string.h>
   #include "y.tab.h"

   int debug=0;
%}
term ([a-zA-Z_][a-zA-Z0-9_]*|-?[0-9]+)
relop ("<="|"<"|">="|">"|"=="|"!=")
op ("+"|"-"|"*"|"/"|"%")
bool_op ("!"|"&&"|"||")
%%
";" {return EOS;}
"if" {return IF;}
"else" {return ELSE;}
"while" { return WHILE; }
"do" { return DO; }
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"break" { return BREAK; }
{bool_op} {yylval.str = strdup(yytext);return BOOL_OP;}
{term} { yylval.str = strdup(yytext); return TERM; }
{relop} { yylval.str = strdup(yytext); return REL_OP; }
```

```
"=" {yylval.str = strdup(yytext); return ASSIGN_OP;}
"(" {return ROUND_LEFT;}
")" {return ROUND_RIGHT;}
"-" {yylval.str = strdup(yytext); return MINUS_OP;}
{op} { yylval.str = strdup(yytext); return ARITH_OP; }
[ \t\n]+ { }
. { return *yytext; }
%%

/*ex4.y*/
%{
   #include <stdlib.h>
   #include <stdio.h>
   int yylex(void);
   extern FILE* yyin;
   #include "y.tab.h"
   int error = 0;
   /*extern int debug;*/
   int termCount = 1, controlCount = 1, switchCount = 0, numberCase = 1;
   int cc = 1, tc = 1, nc = 1, sc = 0;
%}
%token TERM ASSIGN_OP ARITH_OP REL_OP ID BOOL_OP EOS IF ELSE WHILE SWITCH
CASE DEFAULT BREAK DO MINUS_OP ROUND_LEFT ROUND_RIGHT
%union
{
   int intval;
   float floatval;
   char *str;
}
%type<str> TERM REL_OP ARITH_OP ASSIGN_OP BOOL_OP MINUS_OP
%%
line: /* empty */
   | TERM ASSIGN_OP TERM ARITH_OP TERM EOS { printf("t%d := %s %s %s\n%s :=
t%d\n", termCount, $3, $4, $5, $1, termCount); termCount++; } line

   | TERM ASSIGN_OP TERM MINUS_OP TERM EOS { printf("t%d := %s %s %s\n%s :=
t%d\n", termCount, $3, $4, $5, $1, termCount); termCount++; } line

   | TERM ASSIGN_OP TERM REL_OP TERM EOS { printf("t%d := %s %s %s\n%s := t%d\n",
termCount, $3, $4, $5, $1, termCount); termCount++; } line

   | TERM ASSIGN_OP TERM EOS { printf("%s := %s\n", $1, $3); } line
```

```
   | TERM ASSIGN_OP MINUS_OP TERM EOS {printf("t%d := -%s\n%s := t%d\n", termCount, $4, $1, termCount); termCount++; } line

   | TERM ASSIGN_OP TERM BOOL_OP TERM EOS { printf("t%d := %s %s %s\n%s := t%d\n", termCount, $3, $4, $5, $1, termCount); termCount++; } line

   | while_block
   | switch_block

while_block: WHILE ROUND_LEFT TERM REL_OP TERM ROUND_RIGHT '{' { printf("LABEL_%d: if not %s %s %s then goto FALSE_%d\nTRUE_%d: ", controlCount, $3, $4, $5, controlCount, controlCount); } line '}' { printf("FALSE_%d: ", controlCount); controlCount++; } line

   | WHILE ROUND_LEFT TERM ARITH_OP TERM ROUND_RIGHT '{' { printf("LABEL_%d: if not %s %s %s then goto FALSE_%d\nTRUE_%d: ", controlCount, $3, $4, $5, controlCount, controlCount); } line '}' { printf("FALSE_%d: ", controlCount); controlCount++; } line

   | WHILE ROUND_LEFT TERM BOOL_OP TERM ROUND_RIGHT '{' { printf("LABEL_%d: if not %s %s %s then goto FALSE_%d\nTRUE_%d: ", controlCount, $3, $4, $5, controlCount, controlCount); } line '}' { printf("FALSE_%d: ", controlCount); controlCount++; } line

   | WHILE ROUND_LEFT TERM ROUND_RIGHT DO '{' { printf("LABEL_%d: if not %s then goto FALSE_%d\nTRUE_%d: ", controlCount, $3, controlCount, controlCount); } line '}' { printf("FALSE_%d: ", controlCount); cc++; } line

switch_block: SWITCH ROUND_LEFT TERM REL_OP TERM ROUND_RIGHT '{' { printf("t%d := %s %s %s\n", termCount, $3, $4, $5); switchCount = termCount; termCount++; } cases_block '}' { printf("NEXT_%d: ", controlCount); controlCount++; } line

   | SWITCH ROUND_LEFT TERM ARITH_OP TERM ROUND_RIGHT '{' { printf("t%d := %s %s %s\n", termCount, $3, $4, $5); switchCount = termCount; termCount++; } cases_block '}' { printf("NEXT_%d: ", controlCount); controlCount++; } line

   | SWITCH ROUND_LEFT TERM MINUS_OP TERM ROUND_RIGHT '{' { printf("t%d := %s %s %s\n", termCount, $3, $4, $5); switchCount = termCount; termCount++; } cases_block '}' { printf("NEXT_%d: ", controlCount); controlCount++; } line

   | SWITCH ROUND_LEFT TERM BOOL_OP TERM ROUND_RIGHT '{' { printf("t%d := %s %s %s\n", termCount, $3, $4, $5); switchCount = termCount; termCount++; } cases_block '}' { printf("NEXT_%d: ", controlCount); controlCount++; } line
```

```
   | SWITCH ROUND_LEFT TERM ROUND_RIGHT '{' { printf("t%d := %s\n", termCount, $3); switchCount = termCount; termCount++; } cases_block '}' { printf("NEXT_%d: ", controlCount); controlCount++; } line

   | BREAK EOS line { printf("goto NEXT_%d\n", controlCount); }

cases_block: /* empty */
    | CASE TERM ':' { printf("CASE_%d: if not t%d == %s goto CASE_%d\n", numberCase, switchCount, $2, numberCase + 1); numberCase++; } line cases_block

    | DEFAULT { printf("CASE_%d: ", numberCase); numberCase++; } ':' line { printf("goto NEXT_%d\n", controlCount); } cases_block

%%
int yyerror(char* s)
{
  fprintf(stderr, "%s\n", s);
  return 0;
}
int yywrap(){
   return 1;
}

int main(int argc, char **argv){
   /*yydebug = 1;*/
   if(argc != 2){
      fprintf(stderr, "Enter file name as argument!\n");
      return 1;
   }
   yyin = fopen(argv[1], "rt");
   if (!yyin){
      fprintf(stderr, "File not found!\n");
      return 2;
   }
   yyparse();
   printf("\n");
   return 0;
}
```

Output:

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$ lex ex4.l
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$ yacc -d ex4.y
ex4.y: warning: 3 shift/reduce conflicts [-Wconflicts-sr]
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$ gcc -w y.tab.c lex.yy.c
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$ ./a.out
Enter expression: a=b*-c+b*-c
temp1 = - c
temp2 = b * temp1
temp3 = - c
temp4 = b * temp3
temp5 = temp2 + temp4
a = temp5

kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$ ./a.out
Enter expression: x=a<b or c<d or e<f
100: if a < b, goto 103
101: temp1 = 0
102: goto 104
103: temp1 = 1
104: if c < d, goto 107
105: temp2 = 0
106: goto 108
107: temp2 = 1
108: if e < f, goto 111
109: temp3 = 0
110: goto 112
111: temp3 = 1
112: temp4 = temp2 or temp3
113: temp5 = temp1 or temp4
114: x = temp5

kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4$
```

```
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4/ex4g$ ./a.out in.txt
LABEL_1: if not i < 10 then goto FALSE_1
TRUE_1: a := 0
t1 := i + 1
i := t1
FALSE_1: t2 := i + j
CASE_1: if not t2 == 1 goto CASE_2
t3 := y + z
x := t3
goto NEXT_2
CASE_2: if not t2 == 2 goto CASE_3
t4 := v + w
u := t4
goto NEXT_2
CASE_3: t5 := q + r
p := t5
goto NEXT_2
NEXT_2:
kri@Krithika-PC-Win11:/mnt/e/ssn/sem 6/compiler design/lab/assignments/ex4/ex4g$
```

Learning outcomes:

- The internal working of a compiler was analysed and understood.
- The concept of tokens and parsing for tokens in Java was understood and implemented.
- A syntax analyser was implemented for a Java program using the lex and yacc tools.
- Intermediate code was generated for the given sample code using the lex and yacc tools.
