Ex-no: 6 M. Rohith
08-11-2024 3122 21 5001 085

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Compiler Lab – UCS 2702
Programming Assignment-6 - Generate intermediate code using Lex and Yacc Tools

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

INPUT

a := b * -c + b* -c
x:= a<b or c<d and e<f

OUTPUT

t1 = -c
t2 =b*t1
t3 = -c
t4 = b* t3
t5=t2 + t4
a=t5
100: if a<b goto 103
101: t1 := 0
102: goto 104
103: t1 := 1
104: if c<d goto 107
105: t2 := 0
106: goto 108
107: t2 := 1
108: if e<f goto 111
109: t3 := 0
110: goto 112
111: t3 := 1
112: t4 := t2 and t3113: t5 := t1 or t4
114: x=t5

Program code:

intermediate.l

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

%%
"and"     { return AND; }
"or"      { return OR; }
":="      { return ASSIGN; }
"+"       { return PLUS; }
"-"       { return MINUS; }
"*"       { return MULT; }
"/"       { return DIV; }
"<"       { return LT; }
"("       { return LPAREN; }
")"       { return RPAREN; }
";"       { return SEMICOLON; }
[0-9]+              { yylval.num = atoi(yytext); return NUM; }
[a-zA-Z][a-zA-Z0-9]* { yylval.id = strdup(yytext); return ID; }
[ \t\n]+  { /* ignore whitespace */ }
.         { printf("Unknown character: %s\n", yytext); }




%%

int yywrap() {
   return 1;
}
```

intermediate.y

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

int temp_count = 0;   // Counter for temporary variables
int label_count = 100; // Counter for labels

int new_temp() { return temp_count++; }
int new_label() { return label_count++; }

void emit(const char* op, const char* arg1, const char* arg2, const char* result) {
   if (strcmp(op, "=") == 0) {
```

```
        printf("%s = %s\n", result, arg1);          // For assignments
    } else if (arg2) {
        printf("%s = %s %s %s\n", result, arg1, op, arg2);      // For binary operations
    } else {
        printf("%s = %s %s\n", result, op, arg1);       // For unary operations
    }
}

void emit_if_goto(const char* op, const char* arg1, const char* arg2, const char* label) {
    printf("if %s %s %s goto %s\n", arg1, op, arg2, label);
}

void emit_label(const char* label) { printf("%s:\n", label); }
void emit_goto(const char* label) { printf("goto %s\n", label); }

void yyerror(const char* s) {  }
int yylex();
%}

%union { int num; char* id; }
%token <id> ID
%token <num> NUM
%token PLUS MINUS MULT DIV ASSIGN LT AND OR
%token LPAREN RPAREN SEMICOLON
%type <id> expr assignment boolean_expr
%left OR
%left AND
%nonassoc LT
%left PLUS MINUS
%left MULT DIV
%right ASSIGN UMINUS

%%

stmt_list:
    stmt_list stmt SEMICOLON | stmt SEMICOLON ;

stmt:
    assignment | boolean_expr ;

assignment:
    ID ASSIGN expr { emit("=", $3, NULL, $1); } ;

expr:
    expr PLUS expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("+", $1, $3,
temp); $$ = strdup(temp); }
    | expr MINUS expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("-", $1, $3,
temp); $$ = strdup(temp); }
    | expr MULT expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("*", $1, $3,
temp); $$ = strdup(temp); }
    | expr DIV expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("/", $1, $3,
temp); $$ = strdup(temp); }
```

```
    | MINUS expr %prec UMINUS { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t);
emit("-", $2, NULL, temp); $$ = strdup(temp); }
    | ID { $$ = strdup($1); }
    | NUM { char temp[10]; sprintf(temp, "%d", $1); $$ = strdup(temp); }
    | LPAREN expr RPAREN { $$ = $2; }
    ;


boolean_expr:
    expr LT expr { char label_true[10], label_end[10]; sprintf(label_true, "L%d", new_label());
sprintf(label_end, "L%d", new_label()); emit_if_goto("<", $1, $3, label_true);
emit_goto(label_end); emit_label(label_true); printf("1\n"); emit_label(label_end); }
    | expr OR expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("or", $1, $3,
temp); $$ = strdup(temp); }
    | expr AND expr { int t = new_temp(); char temp[10]; sprintf(temp, "t%d", t); emit("and", $1, $3,
temp); $$ = strdup(temp); }
    ;

%%

int main() {
    return yyparse();
}
```

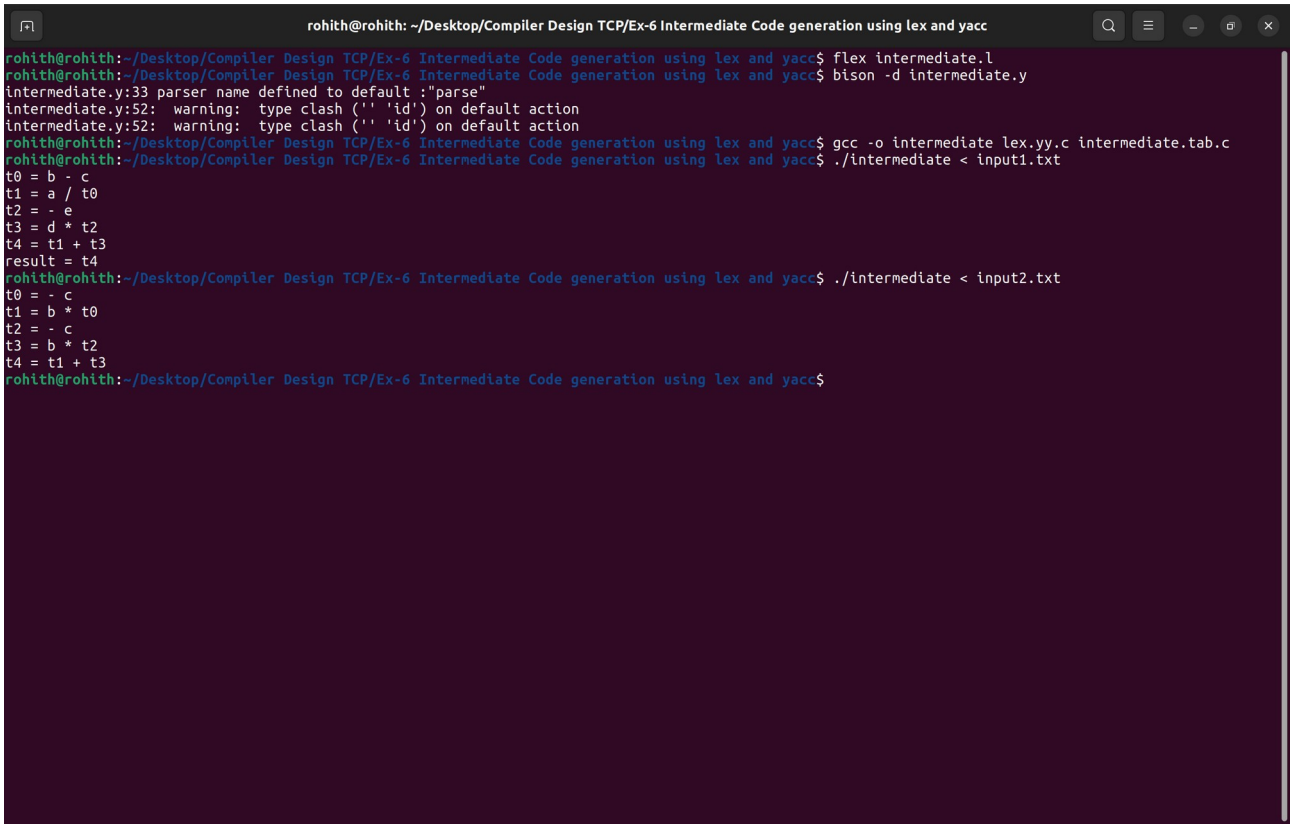
input1.txt

```
result := a / (b - c) + d * -e
```

input2.txt

```
a := b * -c + b* -c
```

Output


rohith@rohith: ~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$ flex intermediate.l
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$ bison -d intermediate.y
intermediate.y:33 parser name defined to default :"parse"
intermediate.y:52: warning: type clash ('' 'id') on default action
intermediate.y:52: warning: type clash ('' 'id') on default action
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$ gcc -o intermediate lex.yy.c intermediate.tab.c
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$ ./intermediate < input1.txt
t0 = b - c
t1 = a / t0
t2 = - e
t3 = d * t2
t4 = t1 + t3
result = t4
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$ ./intermediate < input2.txt
t0 = - c
t1 = b * t0
t2 = - c
t3 = b * t2
t4 = t1 + t3
rohith@rohith:~/Desktop/Compiler Design TCP/Ex-6 Intermediate Code generation using lex and yacc$