COMPILER DESIGN

Ritu Yadav

INTERMEDIATE CODE GENERATION FOR SUBSET C LANGUAGE

OBJECTIVE:

The main objective of the project is to generate the intermediate code for the given **SUBSET C LANGUAGE**. The language may consist **of IF-THEN-ELSE, FOR, WHILE, SWITCH** constructs (even if they are nested).

The programs are written in lex and yacc.

Background:

Lex helps you by taking a set of description of possible tokens and producing a C routine, which we call as lexical analyzer or a lexer, that identify the tokens. As input is divided into tokens, a program often needs to establish the relationship among the tokens. A compiler needs to find the expression, statement, declaration, blocks and procedure in the program. This task is known as parsing and the list of rules that define the relationship that the program is a grammar. Yacc takes a concise description of a grammar and produces a C routine that can parse the grammar, a parser. The yacc parser automatically detects whenever a sequence of input token matches one of the rule of the grammar and also detects a syntax error whenever its input does not match any of the rule. When a task involves dividing the input into units and establishing some relationship among those units then we should use lex and yacc. Hence, using Lex and Yacc, in this project a compiler is designed.

Design:

The program is modularized into three separate levels, the condition part, where in it moves to different levels of code within the condition part as per the different operators. The inside of the ***if-then-else, for, while, switch*** statements move into different intermediate codes as per different assignment statements. The written code generates the ***intermediate code*** in ***two passes***.

Execution Steps:

$lex inter_code_1.l
$yacc -d inter_code_1.y
$cc y.tab.c -ll-ly
$./a.out (filename (test1.c / test2.c /test3.c)

LEX PROGRAM:

Description:

Lex is a tool for building lexical analyzer or lexer. A lexer takes an arbitrary input stream and tokenizes it. In this program, when we create a lex specification, we create a set of pattern which lex matches against the input. Lex program basically has three sections one is definition section another is rule section and last one is user subroutine section which in this case is not written here but in yacc program. The rules section contain the patterns that has to be matched and followed by an action part, which returns the tokens to the yacc program when the input is parsed and the pattern is matched.

/ --- refer to the inter_code_1.l for lex code ------- */*

YACC PROGRAM:

Description:

Yacc takes a grammar that we specify and writes a parser that recognizes valid syntax in that grammar. Grammar is a series of rules that the parser uses to recognize syntactically valid input. Again in same way as lex specification, a yacc grammar has three-part structure. The first section is the definition section, handles control information for the yacc generated by the parser. The second section contains the rule of parser and the third section contains the C code. In the following program, the definition section has the header files that will generate the necessary files needed to run the C

program next comes the declaration of the tokens which are passed from the lex program. The LEX file lex.yy.c is attached to it. Then we define the grammar for the valid input typed by the user. Then it contains the necessary SDT's (Syntax Directed Translations) which move to a specific function when that particular input is encountered by the parser. The conditional code moves as per the Boolean expression defined into various levels. The number of labels generated is equal to the number of the Boolean comparisons in the conditional code.

/ --- refer to the inter_code_1.y for yacc code ------- */*

Experiments:

Test Experiment 1:

Input File :- test3.c

```
switch(a)
{
        case 1: for(i=0;i<p;i+=1)
                {
                        for(h=0;h=o;h+=1)
                        {
                                a=a+b*s/d;
                        }
                        a=a+b;
                }
                break;
        case 2: while(k=c/s)
                {
                                for(i=0;i>p;i-=1)
                                {
                                        b=n*o+(d/r);
                                }
                        k=1;
                }
                break;
        case 3:if(i+1)
                then s=1;
                else s=2;
                break;
        default:c=c+d;
}
```

Output:-

```
ubuntu@ubuntu:~$ ./a.out test3.c
r0:
t0 = not 1
if t0 goto r1
i = 0
L0:
t1 = i < p
t2 = not t1
if t2 goto N8
goto L2
L3:
t3 = i + 1
i = t3
goto L0
L2:
h = 0
L3:
h = o
t4 = not h
r1:
t10 = not 2
if t10 goto r2
L7:
t11 = c / s
c = t11
t12 = not c
if t12
goto N8
L9:
i = 0
L9:
t13 = i > p
t14 = not t13
if t14 goto L10
goto L11
L12:
t15 = i - 1
i = t15
goto L9
L11:
t16 = n * o
t17 = d / r
t18 = d / t17
d = t18
goto L7
k = 1
goto L0
r2:
t19 = not 3
if t19 goto r3
t20 = i + 1
t21 = not t20
if t21 goto L13
s = 1
goto N8
L13:
s = 2
t22 = c + d
c = t22
goto L0
```

Test Experiment 2:

Input File :- test2.c

```
switch(a)
{
        case 1:for(i=0;i<p;i+=1)
                {
                        a=a+b;
                }
               break;
        case 2:while(k=c/s)
                {
                                for(r=2;r>6;r-=1)
                                {
                                w=w*e*f;
                                }
                                k=1;
                }
               break;
        case 3:if(i+1)
               then s=1;
               else s=2;
               break;
        default:c=c+d;
}
for(j=1;j=r;j+=1)
{
        b=c*d/e;
}
```

Output :

```
ubuntu@ubuntu:~$ ./a.out test2.c
r0:
t0 = not 1
if t0 goto r1
i = 0
L0:
t1 = i < p
t2 = not t1
if t2 goto N8
goto L2
L3:
t3 = i + 1
i = t3
goto L0
L2:
t4 = a + b
a = t4
goto L3
r1:
t5 = not 2
if t5 goto r2
L4:
t6 = c / s
c = t6
t7 = not c
if t7
goto N8
L6:
r = 2
L6:
t8 = r > 6
t9 = not t8
if t9 goto L7
goto L8
L9:
t10 = r - 1
r = t10
goto L6
L8:
t11 = w * e
t12 = t11 * f
t11 = t12
goto L4
k = 1
goto L0
r2:
t13 = not 3
if t13 goto r3
t14 = i + 1
t15 = not t14
if t15 goto L10
s = 1
goto N8
L10:
s = 2
t16 = c + d
c = t16
goto L0
j = 1
L11:
j = r
t17 = not j
if t17 goto L12
goto L13
L14:
t18 = j + 1
j = t18
goto L11
L13:
t19 = c * d
t20 = t19 / e
t19 = t20
goto L0
```

Test Experiment 3:.

Input:-
test1.c

```
switch(a)
{
        case 1:for(i=0;i<p;i+=1)
                {
                        a=a+b;
```

```
            }
        break;
    case 2:while(k=c/s)
            {
                k=1;
            }
        break;
    case 3:if(i+1)
        then s=1;
        else s=2;
        break;
    default:c=c+d;
}
```

Output:-

```
r0:
t0 = not 1
if t0 goto r1
i = 0
L0:
t1 = i < p
t2 = not t1
if t2 goto N8
goto L2
L3:
t3 = i + 1
i = t3
goto L0
L2:
t4 = a + b
a = t4
goto L3
r1:
t5 = not 2
if t5 goto r2
L4:
t6 = c / s
c = t6
t7 = not c
if t7
goto N8
L6:
k = 1
goto L4
r2:
t8 = not 3
if t8 goto r3
t9 = i + 1
t10 = not t9
if t10 goto L7
s = 1
goto N8
L7:
s = 2
t11 = c + d
c = t11
```

Attached Files:- inter_code_1.l (lex file) , inter_code_1.y(yacc file), Input Files:- test1.c ,test2.c , test3.c