EXPERIMENT - XIV
INTERMEDIATE CODE GENERATOR

November 20, 2020

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To write a program to implement intermediate code generation for simple expressions.

THEORY

Intermediate code generator receives input from its predecessor phase, semantic analyzer, in the form of an annotated syntax tree. That syntax tree then can be converted into a linear representation, e.g., postfix notation. Intermediate code tends to be machine independent code. Therefore, code generator assumes to have unlimited number of memory storage (register) to generate code.
For example:

```
a = b + c * d;
```

The intermediate code generator will try to divide this expression into sub-expressions and then generate the corresponding code.

```
r1 = c * d;
r2 = b + r1;
a = r2;
```

r being used as registers in the target program.
A three-address code has at most three address locations to calculate the expression.

ALGORITHM

Algorithm 1 Algorithm to perform constant propagation

```
1: procedure INTERMEDIATE-CODE(rhs)
2:     Start
3:     while len(rhs) ≠ 0 do
4:         if '(' in rhs then
5:             Find the indices of opening and closing parentheses (op and cl).
6:             Call Intermediate-Code(rhs[op+1:cl])
7:         else if '/' in rhs then
8:             op='/';
9:         else if '*' in rhs then
10:            op='*';
11:        else if '+' in rhs then
12:            op='+';
13:        else if '-' in rhs then
14:            op='-';
15:        end if
16:        index=rhs.find(op)
17:        Initialize a new variable (say X)
18:        print variable+'='+rhs[index-1]+op+rhs[index+1]
19:        Replace rhs[index-1:index+2] with the new variable (X).
20:    end while
21:    Stop
22: end procedure
23: Read expression (exp).
24: Let rhs be the right-hand-side of exp.
25: Call Intermediate-Code(rhs).
```

SOURCE CODE

```
def balanced(exp):
        if '(' not in exp and ')' not in exp:
                return True
        else:
                count=0
                for i in range(len(exp)):
                        if exp[i]=='(':
                                count+=1
                        elif exp[i]==')':
                                count-=1
                        if count==-1:
                                return False
        if count==0:
                return True
        return False

def set(exp):
        list=[]
        br={}
        for i in range(len(exp)):
                if exp[i]=='(':
                        list.append(i)
                elif exp[i]==')':
                        br[list.pop()]=i
        return br

def bracket(exp):
        global ch
        br=set(exp)
        index=exp.find('(')
        sub=igc(exp[index+1:br[index]])
        exp=exp[:index]+sub+exp[br[index]+1:]
        return exp
```

```
def div(exp):
        global ch
        index=exp.find('/')
        print('\t'+ch+' = '+exp[index-1]+' / '+exp[index+1])
        str=exp.replace(exp[index-1]+'/'+exp[index+1],ch)
        ch=chr(ord(ch)-1)
        return str

def mul(exp):
        global ch
        index=exp.find('*')
        print('\t'+ch+' = '+exp[index-1]+' * '+exp[index+1])
        str=exp.replace(exp[index-1]+'*'+exp[index+1],ch)
        ch=chr(ord(ch)-1)
        return str

def add(exp):
        global ch
        index=exp.find('+')
        print('\t'+ch+' = '+exp[index-1]+' + '+exp[index+1])
        str=exp.replace(exp[index-1]+'+'+exp[index+1],ch)
        ch=chr(ord(ch)-1)
        return str

def sub(exp):
        global ch
        index=exp.find('-')
        print('\t'+ch+' = '+exp[index-1]+' - '+exp[index+1])
        str=exp.replace(exp[index-1]+'-'+exp[index+1],ch)
        ch=chr(ord(ch)-1)
        return str

def igc(exp):
        rhs=exp
```

```
        while(len(rhs)!=1):
                if '(' in exp:
                        exp=bracket(exp)
                elif '/' in exp:
                        exp=div(exp)
                elif '*' in exp:
                        exp=mul(exp)
                elif '+' in exp:
                        exp=add(exp)
                elif '-' in exp:
                        exp=sub(exp)
                rhs=exp[exp.find('=')+1:]
        return exp

ch='Z'
exp=input("Enter the expression : ")
if '=' not in exp:
        print("Invalid expression!!!")
elif not balanced(exp):
        print("Parentheses not balanced!!!")
else:
        print()
        print("Intermediate Code:")
        str=exp.split('=')
        exp=igc(str[1])
        print('\t'+str[0]+' = '+exp)
        print()
```

SAMPLE OUTPUT

```
user@adithya-d-rajagopal:~/s7/cd$ python3 p14.py
Enter the expression : w=(a*(b+c)/(d-e))/f+g*h

Intermediate Code:
        Z = b + c
        Y = d - e
        X = Z / Y
        W = a * X
        V = W / f
        U = g * h
        T = V + U
        w = T

user@adithya-d-rajagopal:~/s7/cd$
```

RESULT

A program to implement intermediate code generation for simple expressions has been implemented using Python and the outputs were verified.