Universidad Nacional Autónoma de México
Facultad de Estudios Superiores Aragón

Ingeniería en Computación

COMPILADORES

Grupo: 2608

Profesor: Pérez Medel Marcelo

TAREA 7

Generación de código intermedio y ensamblador

Alumna: Cruz Cervantes Guadalupe Sugeily

Generación de código intermedio y ensamblador

- **Lean y expliquen cada función.**

genera Codigo.py > ...

```python
class Pila: #Se crea la clase Pila(nos ayuda a generar la parte del codigo intermedio)
    arreglo = []
    def meter(self, dato):#Agrega datos a la Pila
        self.arreglo.append(dato)
    def sacar(self):#Saca datos de la Pila
        if (len(self.arreglo)==0):
            print("Pila vacia")
        else:
            return self.arreglo.pop()
```

```python
def esOperador(v): #Funcion que trabaja con la parte del codigo intermedio
    return (v in "*/+-")

def esSeparador(caracter): #Funcion que trabaja en la Funcion tokeniza.
    return caracter in " \n\t"

def esSimboloEsp(caracter): #Funcion que trabaja en la Funcion tokeniza.
    return caracter in "+-*;,.:!=%&/()[]{}<><=>=:="

def obtenerPrioridadOperador(o): # Función que trabaja con convertirInfijaA**.
    return {'(':1, ')':2, '+': 3, '-': 3, '*': 4, '/':4, '^':5}.get(o)
```

La Función obtenerListaInfija() recibe una cadena_infija la cual es una lista, que al entrar al if se convierte en str, después entra a un for y va agregando dependiendo de las condiciones que tenemos nuestro valor en la variable “i” en nuestra lista nueva “infija”, al final nos regresa una lista en notación infija.

```python
def obtenerListaInfija(cadena_infija):
    if(type(cadena_infija) == list):
        return obtenerListaInfija("".join(cadena_infija))
    '''Devuelve una cadena en notación infija dividida por sus elementos.'''
    infija = []
    cad = ''
    for i in cadena_infija:
        if i in['+', '-', '*', '/', '(', ')', '^', '=']:
            if cad != '':
                infija.append(cad)
                cad = ''
            infija.append(i)
        elif i == chr(32): # Si es un espacio.
            cad = cad
        else:
            cad += i
    if cad != '':
        infija.append(cad)
    return infija
```

La Función infija2Posfija() recibe una expresión infija y la convierte a posfija con ayuda de un for que ocupa la funcion obtenerPrioridadOperador() (para poder comparar la prioridad del operador de e, con el que tiene la pila en la posición pila-1) y sentencias condicionales; if y while para poder ordenar nuestra lista “salida” en notación posfija.

```
43  def infija2Posfija(expresion_infija):
44      '''Convierte una expresión infija a una posfija, devolviendo una lista.'''
45      infija = obtenerListaInfija(expresion_infija)
46      pila = []
47      salida = []
48      for e in infija:
49          if e == '(':
50              pila.append(e)
51          elif e == ')':
52              while pila[len(pila) - 1 ] != '(':
53                  salida.append(pila.pop())
54              pila.pop()
55          elif e in ['+', '-', '*', '/', '^']:
56              while (len(pila) != 0) and (obtenerPrioridadOperador(e)) <= obtenerPrioridadOperador(pila[len(pila) - 1]):
57                  salida.append(pila.pop())
58              pila.append(e)
59          else:
60              salida.append(e)
61      while len(pila) != 0:
62          salida.append(pila.pop())
63      return salida
```

La función tokeniza como ya se vio antes, nos ayuda a separar nuestra cadena en tokens con ayuda de ciertas reglas, para esto ocupamos las funciones; esSeparador() y esSimboloEsp().

```
65  def tokeniza(cad):
66      tokens = []
67      dentro = False
68      token = ""
69      for c in cad:
70          if dentro: #esta dentro del token
71              if esSeparador(c):
72                  tokens.append(token)
73                  token = ""
74                  dentro = False
75              elif esSimboloEsp(c):
76                  tokens.append(token)
77                  tokens.append(c)
78                  token = ""
79                  dentro = False
80              else:
81                  token = token + c
82          else: #esta fuera del token
83              if esSimboloEsp(c):
84                  tokens.append(c)
85              elif esSeparador(c):
86                  a=0
87              else:
88                  dentro = True
89                  token = c
90      if token != '':
91          tokens.append(token)
92      return tokens
```

Pasamos a la parte de crear nuestro código intermedio y ensamblador, definiendo las variables a ocupar y el método por el cual se generará cada paso.

```python
prog = "a*b*c+d+e+f*g*a"
tokens = tokeniza(prog)
posfija = infija2Posfija(tokens)

ct = 1
pila1 = Pila()
intermedio = []
codigo = []
for e in posfija:
    if esOperador(e):
        op2 = pila1.sacar() #saca el ultimo valor dentro de nuestra pila, recordando que su estructura es LIFO.
        op1 = pila1.sacar() # Para ir en orden lo colocamos dentro de op2 y despues en op1
        cad = "t"+str(ct)+"="+op1+e+op2+";" # crea el codigo intermedio. op1 y op2 seran operandos mientras que e el operador
        intermedio.append(cad) #Cada codigo intermedio se va agregando a una lista "intermedio".
        print("LDA "+op1+";") #Comenzamos a generar el codigo ensamblador, dependiendo de e a que operador es igual.
        if e=="+":
            print("ADD "+op2+";")
        elif e=="-":
            print("SUB "+op2+";")
        elif e=="*":
            print("MUL "+op2+";")
        elif e=="/":
            print("DIV "+op2+";")
        pila1.meter("t"+str(ct)) # agregamos dentro de pila1 el orden de t dependiendo el contador.
        print("STA "+ "t"+str(ct)+";")
        ct = ct + 1
    else:
        pila1.meter(e) #agregamos los operandos.
```

```
gpy\launcher' '58476' '--' 'c:\Users\55gus\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\6 semestre\Compiladores\genera Codigo.py'
LDA a;
MUL b;
STA t1;
LDA t1;
MUL c;
STA t2;
LDA t2;
ADD d;
STA t3;
LDA t3;
ADD e;
STA t4;
LDA f;
MUL g;
STA t5;
LDA t5;
MUL a;
STA t6;
LDA t4;
ADD t6;
STA t7;
PS C:\Users\55gus\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\6 semestre\Compiladores>
```

- Agregue los operadores de resta y división.

```python
        print("LDA "+op1+";")#CARGA
        if e=="+":
            print("ADD "+op2+";")#SUMA
        elif e=="-":
            print("SUB "+op2+";")#RESTA
        elif e=="*":
            print("MUL "+op2+";")#MULTIPLICA
        elif e=="/":
            print("DIV "+op2+";")#DIVIDE
        pila1.meter("t"+str(ct))
        print("STA "+ "t"+str(ct)+";")#TOMA
```

- Arregle la función (Tokeniza) para que no requiera un espacio al final.

```
65  def tokeniza(cad):
66      tokens = []
67      dentro = False
68      token = ""
69      for c in cad:
70          if dentro: #esta dentro del token
71              if esSeparador(c):
72                  tokens.append(token)
73                  token = ""
74                  dentro = False
75              elif esSimboloEsp(c):
76                  tokens.append(token)
77                  tokens.append(c)
78                  token = ""
79                  dentro = False
80              else:
81                  token = token + c
82          else: #esta fuera del token
83              if esSimboloEsp(c):
84                  tokens.append(c)
85              elif esSeparador(c):
86                  a=0
87              else:
88                  dentro = True
89                  token = c
90      if token != '':
91          tokens.append(token)
92      return tokens
```