COMPILER DESIGN

EXP – 10 Intermediate Code Generation-Infix to Postfix/Prefix

Tamojit Sarkar
RA1811027010034
CSE-BD Sec-I2

Aim: To compute intermediate code generation –infix to prefix and postfix

Language Used: Python

Algorithm:

1. Define the set of operators according to their priortity.
2. Define function for infix to postfix, create a stack and then in the formula check for operators and parentheses and enter the parentheses into the stack and characters directly in output.
3. If there is any operator in the middle of open and close parentheses pop it to the output.
4. Convert Infix to Prefix.

   Step 1: Reverse the infix expression i.e A+B*C will become C*B+A. Note while reversing each '(' will become ')' and each ')' becomes '('.

   Step 2: Obtain the "nearly" postfix expression of the modified expression i.e CB*A+.

   Step 3: Reverse the postfix expression. Hence in our example prefix is +A*BC.

Code:

```
class Stacks:
    def __init__(self):
        self.items = []

    def push(self,data):
        self.items.append(data)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[-1]

    def is_empty(self):
        return self.items == []

    def printStack(self):
        return self.items

# converts infix expression to postfix expression
def infix_to_postfix(st,priority,inf):
    pst = []
    for el in inf:
        if el == ' ':
            continue
        # print("checking: ",el)
        # opening parenthesis
        if el == '(':
            st.push(el)

        elif el == ')':
            while not st.is_empty():
                if st.peek() == '(':
                    st.pop()
                    break
                else:
                    pst.append(st.pop())
        # if it's an operand
        elif el not in priority.keys() and (el != '(' or el != ')'):
            # print("operand:",el)
            pst.append(el)

        # if it's an operator and stack is empty
        elif el in priority.keys() and  st.is_empty():
            # print("operator: ",el)
            st.push(el)
         # if it's an operator of higher priority than the priority of TOS element
        elif el in priority.keys() and not st.is_empty() and st.peek() =='(':
            # print("operator:",el)
            st.push(el)
```

```
        # if it's an operator of higher priority than the priority of TOS element
        elif el in priority.keys() and not st.is_empty() and priority[el] >= priority[st.peek()]:
            # print("operator: ",el)
            st.push(el)
        # if it's an operator of lower priority than the priority of TOS element
        elif el in priority.keys() and not st.is_empty() and priority[el] < priority[st.peek()]:
            # print("operator: ",el)
            while not st.is_empty():
                if st.peek() == '(':
                    break
                elif priority[st.peek()] >= priority[el]:
                    pst.append(st.pop())
                else:
                    break
            st.push(el)
        # print("stack: ",st.printStack())
        # print("postfix:",pst)
    while not st.is_empty():
        pst.append(st.pop())
    return pst

def reverse(exp):
    rev_exp = []
    for i in range(len(exp)-1,-1,-1):
        if exp[i] == '(':
            e = ')'
        elif exp[i] == ')':
            e = '('
        else:
            e = exp[i]
        rev_exp.append(e)
    return rev_exp

def infix_to_prefix(st,priority,inf):
    rev_exp = reverse(inf)
    pst = infix_to_postfix(st,priority,rev_exp)
    return reverse(pst)
if __name__=="__main__":
    priority = {}
    priority['+'] = 1
    priority['-'] = 1
    priority['*'] = 2
    priority['/'] = 2
    priority['^'] = 3
    st = Stacks()
    inf = input("Enter the infix expression:")
    postfix=' '.join(map(str,infix_to_postfix(st,priority,inf) ))
    prefix=' '.join(map(str,infix_to_prefix(st,priority,inf) ))
    print("Postfix expression: ",postfix)
    print("Prefix expression: ",prefix)
```

Output:

```
Enter the infix expression:a+b*c
Postfix expression:  a b c * +
Prefix expression:  + a * b c
```

Result: Converted Infix to Postfix/Prefix Successfully.