

**Name: Sreejith Sathyadevan**

**Roll No: 23**

## **Design and implementation of Modern Compilers**

### **INDEX**

<b>SR NO</b>	<b>TITLE</b>
1	Write a program to construct NDFA
2	Write a program to convert the given Right linear Grammar to Left Linear Grammar.
3	Write a code to generate DAG for input arithmetic expression.
4	Write a code to for triples.
5	Write a code for postfix evaluation.
6	Write a code to generate 3 address code
7	Write a code to demonstrate loop jamming for the given code sequence containing loop.
8	Write a code to demonstrate loop unrolling for the given code sequence containing loop.

# Practical NO 1

Aim : Write a program to construct NDFA

Install package automata-lib By

using the following command:

pip install automata-lib

```
D:\Python>
D:\Python>pip install automata-lib
Collecting automata-lib
  Downloading automata_lib-5.0.0-py3-none-any.whl (32 kB)
Collecting pydot
  Downloading pydot-1.4.2-py2.py3-none-any.whl (21 kB)
Collecting pyparsing>=2.1.4
  Downloading pyparsing-3.0.7-py3-none-any.whl (98 kB)
----- 98.0/98.0 KB 622.8 kB/s eta 0:00:00
Installing collected packages: pyparsing, pydot, automata-lib
Successfully installed automata-lib-5.0.0 pydot-1.4.2 pyparsing-3.0.7
```

Code: from automata.fa.nfa

import NFA class NDFA: def

\_\_init\_\_(self):

state\_set = set(input("Enter state set>\t"))

input\_symbols = set(input("Enter input symbol set>\t"))

initial\_state = input("Enter the initial state>\t")

final\_states = set(input("Enter the final state(s)>\t"))

rule\_count = int(input("Enter the number of rules you want to add>\t"))

rules = []

for counter in range(rule\_count):

rules.append(input("Enter rule " + str(counter + 1) +

```

">\t").replace(" ", ""))
    rules = self.get_transitions(rules)

    self.nfa = NFA(
states = state_set,
        input_symbols = input_symbols,
transitions = rules,        initial_state
= initial_state,        final_states =
final_states
    )

    del state_set, input_symbols, initial_state, final_states,
rules.

```

```

def get_transitions(self, rules):
rules = [i.split("-") for i in rules]
rules_dict = {}

    for rule in rules:        if
rule[0] not in rules_dict:
        rules_dict[rule[0]] = {rule[1][1]:rule[1][0]}
print("If:", rules_dict)        else:

```

```

        rules_dict[rule[0]][rule[1][0]] = rule[1][1]
print("Else:", rules_dict)        return rules_dict

def print_stats(self):
    print("\n\nSet of states are > ", self.nfa.states)
print("Input symbols are > ", self.nfa.input_symbols)
print("Transitions are > ")    for transition in
self.nfa.transitions:
    print(transition, self.nfa.transitions[transition])
print("Initial state > ", self.nfa.initial_state)    print("Final
states > ", self.nfa.final_states)

def print_transition_table(self):
    input_symbols = list(self.nfa.input_symbols)
transitions = self.nfa.transitions

    print("\n\nTransition table is > ")

#print(f"States\t\t{input_symbols[0]}\t\t{input_symbols[1]}")
print("States\t\t" + str(input_symbols[0]) + "\t\t" +
str(input_symbols[1]))    for transition in transitions:        for
input_symbol in input_symbols:            try:

```

```

        temp = transitions[transition][input_symbol]
del temp            except KeyError:
        transitions[transition][input_symbol] = "-"

#print(f"{transition}\t\t{transitions[transition][input_symbols
[0]]}\t\t{transitions[transition][input_symbols[1]]}")

    print(transition + "\t\t" +
transitions[transition][input_symbols[0]] + "\t\t" +
transitions[transition][input_symbols[1]])    del
input_symbols, transitions if __name__ ==
"__main__":
    ndfa = NDFA()
ndfa.print_stats()
ndfa.print_transition_table()

```

Output:

RESTART: C:\Users\Admin\Downloads\ndia.py

```
Enter state set> WAM
Enter input symbol set> 01
Enter the initial state> W
Enter the final state(s)> M
Enter the number of rules you want to add> 3
Enter rule 1> W - 0A
Enter rule 2> A - 1M
Enter rule 3> M - 0W
If: {'W': {'0': 'A'}}
If: {'W': {'0': 'A'}, 'A': {'1': 'M'}}
If: {'W': {'0': 'A'}, 'A': {'1': 'M'}, 'M': {'0': 'W'}}
Set of states are > {'W', 'A', 'M'}
Input symbols are > {'1', '0'}
Transitions are >
W {'0': 'A'}
A {'1': 'M'}
M {'0': 'W'}
Initial state > W
Final states > {'M'}
Transition table is >
States      1      0
W           -      A
A           M      -
M           -      W
```

## PRACTICAL NO 2

Aim: Write a program to convert the given Right linear grammar to Left Linear Grammar form.

CODE:

```
def get_transitions(rules):
```

```
    my_dict={}
    ld=""
```

```
res=dict()
```

```
    r=""    for i in
```

```
rules:
```

```
        my_dict[i[0]]=i[1][1],i[1][0]    for sub in my_dict:
```

```
if isinstance(my_dict[sub],list):
```

```
res[sub]=ld.join([str(ele) for ele in my_dict[sub]])
```

```
print("Left linear grammar is:")    for item in res:
```

```
r+=item+"-"+str(res[item])+"\n"    print(str(r))
```

```
rule_count=int(input("Enter rule count>\t"))
```

```
rules=[] for i in
```

```
range(rule_count):
```

```
    rules.append(input("Enter right linear grammar"+">\t"))
```

```
rules=[i.split("->") for i in rules] print(rules)
```

```
get_transitions(rules)
```

## OUTPUT:

```
= RESTART: C:\Users\Admin\Desktop\Msc CS\SEM 2\Compiler\Practicals\Practical 2(A
).py
Enter rule count>          2
Enter right linear grammar> S->uP
Enter right linear grammar> T->qW
[['S', 'uP'], ['T', 'qW']]
Left linear grammar is:
Left linear grammar is:
S-Pu
T-Wq
```

## PRACTICAL NO 3

Aim: Write a code to generate DAG for input arithmetic expression.

CODE:

```
def funct1(x):
```

```
    main=[]    for i in
```

```
        range(0,x):
```

```
            y=input()
```

```
            main.append(y)
```



```

    print("Label Operator left Right")
for i in range(0,x):
    q=main[i]    if
q[0] not in res:
res.append(q[0])
if(len(q)>3):
    print(" ",q[0]," ",q[3]," ",q[2]," ",q[4])
else:
    print(" ",q[0]," ",q[1]," ",q[2]," ")
print(main)  print(res)

print("Enter number of 3 address code")
x=input() x=int(x) res=[] funct1(x)

```

Output:

```
= RESTART: C:/Users/Admin/Desktop/Msc CS/
Enter number of 3 address code
4
t=a-b
r=a-c
o=t*r
q=o
Label Operator left Right
      t      -      a      b
      r      -      a      c
      o      *      t      r
      q      =      o
['t=a-b', 'r=a-c', 'o=t*r', 'q=o']
['t', 'r', 'o', 'q']
|
```

## PRACTICAL NO 4

Aim: Write a code for triples.

Code:

```
def funct1(x):
    main=[]
    for i in
    range(0,x):
    y=input()
    main.append(y)
```

```

print("Address operator argument 1 argument2")
for i in range(0,x):
    g=main[i]    if
g[0] not in res:
res.append(g[0])
e=funct2(g[2])
if(len(g)>3):
    r=funct2(g[4])
    print("  (",i,")", "  ",g[3], "  ",e, "  ",r)
else:
    print("  (",i,")", "  ",g[1], "  ",e, "  ")  print(main)
print(res) def funct2(g):
    try:
        z=res.index(g)
    return(z)  except:
    return(g)
print("Enter number of production")
x=input() x=int(x) res=[] funct1(x)

```

Output:

```

y
Enter number of production
4
t=a-b
u=a-c
w=t*u
e=w
Address operator argument 1 argument2
( 0 )          -          a          b
( 1 )          -          a          c
( 2 )          *          0          1
( 3 )          =          2
['t=a-b', 'u=a-c', 'w=t*u', 'e=w']
['t', 'u', 'w', 'e']
|

```

## PRACTICAL NO 5

Aim: Write the code for Postfix Evaluation CODE:

```
def postfix_evaluation(s):
```

```
    s=s.split()  n=len(s)  stack=[]
```

```
    for i in range(n):    if
```

```
        s[i].isdigit():
```

```
            stack.append(int(s[i]))    elif
```

```
        s[i]=="+":    a=stack.pop()
```

```
        b=stack.pop()
```

```
        stack.append(int(a)+int(b))
```

```

elif s[i]=="*":
a=stack.pop()
b=stack.pop()
stack.append(int(a)*int(b))
elif s[i]=="/":
    a=stack.pop()
b=stack.pop()
stack.append(int(a)/int(b))
elif s[i]=="-":
    a=stack.pop()
b=stack.pop()
stack.append(int(a)-int(b))
return stack.pop()

s="8 7 8 * + 4 -"
val=postfix_evaluation(s) print(val)

```

OUTPUT:

```

Y
-60
|

```

## PRACTICAL NO 6

Aim: Write a code to generate 3 address code

```
postfix=input("Enter postfix expression").split()
```

```
operators=['+', '-', '/', '*', '^'] stack=[] result=""
```

```
str1=""
```

```
count=0 print("3
```

```
address code") for i in
```

```
postfix: if i not in
```

```
operators:
```

```
stack.append(i)
```

```
print("Stack-",stack)
```

```
else:
```

```
    op1=stack.pop()
```

```
    op2=stack.pop()
```

```
    result=op2+i+op1
```

```
    str1='T'+str(count)
```

```
    stack.append(str1)
```

```
print("T",count,"=",result)
```

```
count+=1
```

Output:

```
C:\Users\Admin\Desktop\Python\Sem 2\Computer  
Y  
Enter postfix expression a b c + / d *  
3 address code  
Stack- ['a']  
Stack- ['a', 'b']  
Stack- ['a', 'b', 'c']  
T 0 = b+c  
T 1 = a/T0  
Stack- ['T1', 'd']  
T 2 = T1*d  
>
```

## PRACTICAL NO 7

Aim: Write a program to demonstrate loop jamming for given code sequence containing loop. Code: Loop Jamming

```
import time
```

```
from datetime import datetime def
```

```
func1(arr1,arr2,arr3):
```

```
t1=datetime.now()
start=time.time()

print(t1.minute,":",t1.second,":",t1.microsecond)

for i in range (0,10000000):
    sum=0    for j in
range(0,len(arr1)):
        sum=sum+arr1[j]
for k in range(0,len(arr2)):
    sum=sum+arr2[k]
for l in range(0,len(arr3)):
    sum=sum+arr3[l]
if(sum!=210):
print(false)
```

```
tm=datetime.now()
done=time.time()
elapsed=done-start
```



```
print(t1.minute,":",t1.second,":",t1.microsecond)
print("First loop Difference",elapsed)
```

```
start=time.time()  for i in
range(0,10000000):
    sum=0    for j in
range(0,len(arr1)):
        sum=sum+arr1[j]
sum=sum+arr2[j]
sum=sum+arr3[j]
if(sum!=210):
print(false)

tn=datetime.now()
done=time.time()

elapsed=done-start

print(t1.minute,":",t1.second,":",t1.microsecond)
print("second loop Diffrence",elapsed)
```

```
arr1=[10,20,30]
```

```
arr2=[20,10,30]
```

```
arr3=[40,40,10]
```

```
func1(arr1,arr2,arr3)
```

OUTPUT:

```
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more informa
= RESTART: C:/Users/Admin/Desktop/Msc CS/SEM 2/Compiler/Practicals,
)-Loop Jamming.py
= RESTART: C:/Users/Admin/Desktop/Msc CS/SEM 2/Compiler/Practicals,
)-Loop Jamming.py
53 : 14 : 254787
53 : 14 : 254787
First loop Difference 21.988343715667725
53 : 14 : 254787
second loop Difference 10.30445909500122
|
```

## PRACTICAL NO 8

Aim: Write a program to demonstrate loop unrolling for given code sequence containing loop.

Loop Unrolling Code:

```
import time
```

```
from datetime import datetime

def funct1():  arr=[]  arr1=[]

t1=datetime.now()

start=t1.microsecond

print(start)  for i in
range(0,1000):

    arr.insert(0,i)

print(arr)

t2=datetime.now()

end1=t2.microsecond

print(end1)

    for i in range(0,1000,4):

        arr1.insert(0,i)

arr1.insert(0,i+1)

arr1.insert(0,i+2)

arr1.insert(0,i+3)

print(arr1)

t3=datetime.now()
```

```
end2=t3.microsecond
```

```
print(end2)
```

```
print("Before unrolling:",end1-start)
```

```
print("After unrolling:",end2-end1) funct1()
```

OUTPUT:

```
----- RESTART: C:\Users\AQUILA\DO
833747
Squeezed text (54 lines).
112643
Squeezed text (54 lines).
369812
Before unrolling: -721104
After unrolling: 257169
|
```