Assignment for the subject

Compiler Construction (UCS802)

Submitted By:

Name: Aditya Vashista
Roll No.: 101703039
Group No.: COE 2

Submitted To:

Dr. Ravneet Kaur

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY, PATIALA
(Date of submission: 24/06/2020)

Code (in python 3 for all valid grammars):

Github: https://github.com/AdityaVashista30/SLR-PARSER

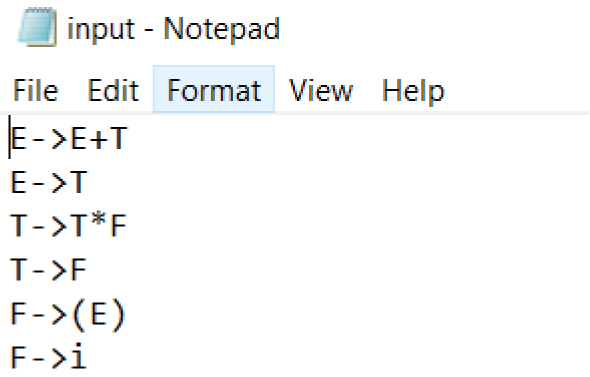
```
import copy
grammar = []
new_grammar = []
terminals = []
non_terminals = []
I_n = {}
shift_list = []
reduction_list = []
action_list = []
rule_dict = {}
SR = []
RR = []

def Conflict():
    global SR, RR, shift_list, reduction_list
    conflict = False
    for S in shift_list:
        for R in reduction_list:
            if S[:2] == R[:2]:
                SR.append([S, R])
                conflict = True

    for R1 in reduction_list:
        for R2 in reduction_list:
            if R1 == R2:
                continue
            if R1[:2] == R2[:2]:
                RR.append(R1)
                conflict = True

    return conflict


def read_grammar():
    global grammar, terminals, non_terminals, rule_dict

    file_name = str(input("Enter Grammar File Name:: "))

    try:
        grammar_file = open(file_name, "r")
    except:
        print("Cannot Find File Named", file_name)
        exit(0)

    for each_grammar in grammar_file:
        grammar.append(each_grammar.strip())

        if each_grammar[0] not in non_terminals:
            non_terminals.append(each_grammar[0])

    for each_grammar in grammar:
        for token in each_grammar.strip().replace(" ", "").replace("->", ""):
            if token not in non_terminals and token not in terminals:
                terminals.append(token)
    for l in range(1, len(grammar)+1):
        rule_dict[l] = grammar[l-1]


def augmented_grammar():
    global grammar, new_grammar
    read_grammar()
    if "'" not in grammar[0]:
        grammar.insert(0, grammar[0][0]+"'"+"->"+grammar[0][0])

    new_grammar = []
    for each_grammar in grammar:
        idx = each_grammar.index(">")
        each_grammar = each_grammar[:idx+1]+"."+each_grammar[idx+1:]
        new_grammar.append(each_grammar)


def compute_I0():
    global new_grammar, non_terminals, I_n
    augmented_grammar()

    grammar2add = []
    grammar2add.append(new_grammar[0])
    i = 0
    for each in grammar2add:
        current_pos = each.index(".")
        current_variable = each[current_pos+1]

        if current_variable in non_terminals:
            for each_grammar in new_grammar:
                if each_grammar[0] == current_variable and each_grammar
not in grammar2add:
                    grammar2add.append(each_grammar)

        I_n[i] = grammar2add



def GOTO():
    global grammar, non_terminals, terminals, I_n, shift_list
    compute_I0()

    variables = non_terminals + terminals
    i = 0
    current_state = 0
    done = False

    while (not done):
        for each_variable in variables:
            grammar2add = []
            try:
                for each_rule in I_n[current_state]:
                    if each_rule[-1] == ".":
                                continue
                    dot_idx = each_rule.index(".")

                    if each_rule[dot_idx+1] == each_variable:

                        rule = copy.deepcopy(each_rule)
                        rule = rule.replace(".", "")
                        rule = rule[:dot_idx+1]+"."+rule[dot_idx+1:]
                        grammar2add.append(rule)

                        for rule in grammar2add:
                            dot_idx = rule.index(".")
                            if rule[-1] == ".":
                                pass
                            else:
                                current_variable = rule[dot_idx+1]

                                if current_variable in
non_terminals:
                                    for each_grammar in
new_grammar:
                                        if each_grammar[0]
== current_variable and each_grammar[1] != "" and each_grammar not in
grammar2add:

    grammar2add.append(each_grammar)

            except:
                done = True
                break

            if grammar2add:
                if grammar2add not in I_n.values():
                    i += 1
                    I_n[i] = grammar2add

                for k,v in I_n.items():
                    if grammar2add == v:
                        idx = k

                shift_list.append([current_state, each_variable, idx])
        current_state += 1


def follow(var):

    global rule_dict, terminals

    value = []
    if var == rule_dict[1][0]:
        value.append("$")

    for rule in rule_dict.values():

        lhs, rhs = rule.split("->")

        if var == rule[-1]:
            for each in follow(rule[0]):
                if each not in value:
                    value.append(each)

        if var in rhs:
            idx = rhs.index(var)

            try:
                if rhs[idx+1] in non_terminals and rhs[idx+1] != var:
                    for each in follow(rhs[idx+1]):
                        value.append(each)
```

```
                                            else:
                                                       value.append(rhs[idx+1])
                                 except:
                                            pass
           return value



def first(nt):
    global non_terminals,rule_dict
    d1={}
    d2={}
    for i in non_terminals:
        d1[i]=[]
        d2[i]=[]
        for j in list(rule_dict.values()):
            if j[0]==i and j[3]!=j[0]:
                if j[3] in non_terminals:
                   d1[i].append(j[3])
                else:
                    d2[i].append(j[3])
    l=d2[nt]
    for i in d1[nt]:
        d1[nt].extend(d1[i])
    for i in d1[nt]:
        l=l+d2[i]

    return l

def reduction():
           global l_n, rule_dict, reduction_list
    reduction_list.append([1, "$", "Accept"])

    for item in I_n.items():
        try:
            for each_production in item[1]:
                lhs, rhs = each_production.split(".")

                for rule in rule_dict.items():

                    if lhs == rule[1]:
                        f = follow(lhs[0])

                        for each_var in f:
                            reduction_list.append([item[0],
each_var, "R"+str(rule[0])])

        except:
            pass



def test(string):
    global action_list, shift_list, reduction_list
    done = False
    stack = []
    stack.append(0)

    print("\n\nSTACK\t\tSTRING\t\tACTION")
    while not done:
        Reduce = False
        Shift = False

        for r in reduction_list:
            if r[0] == int(stack[-1]) and r[1] == string[0]:
                Reduce = True
                print(".join(str(p) for p in stack), "\t\t", string, "\t\t",
"Reduce", r[2])

                if r[2] == 'Accept':
                    return 1
                var = rule_dict[int(r[2][1])]
                lhs, rhs = var.split("->")

                for x in range(len(rhs)):
                    stack.pop()
                    stack.pop()

                var = lhs
                stack.append(var)

                for a in action_list:
                    if a[0] == int(stack[-2]) and a[1] == stack[-1]:
                        stack.append(str(a[2]))
                        break

        for g in shift_list:
            if g[0] == int(stack[-1]) and g[1] == string[0]:
                Shift = True
                print(".join(str(p) for p in stack), "\t\t", string, "\t\t", "Shift",
"S"+str(g[2]))
```

```
                stack.append(string[0])
                stack.append(str(g[2]))
                string = string[1:]




        if not Reduce and not Shift:
            print(".join(str(p) for p in stack), "\t\t", string)
            return 0




def printGOTO_ACTION():
    global terminals,non_terminals,reduction_list,shift_list
    l=[" "]+terminals+['$']+non_terminals
    l2=[]
    for i in range(max(max(shift_list)[0],max(reduction_list)[0])+1):
        l2.append(["  "]*len(l))

    for item in shift_list:
        i=l.index(item[1])
        l2[item[0]][0]='I'+str(item[0])
        if l[i] in non_terminals:
            l2[item[0]][i]=item[2]
        else:
            l2[item[0]][i]='S'+str(item[2])
    for item in reduction_list:
        i=l.index(item[1])
        l2[item[0]][0]='I'+str(item[0])
        l2[item[0]][i]=item[2]
    for i in l:
        print(i,"  ",end=" ")
    print()
    for i in l2:
        for j in i:
            print(j," ",end=" ")
        print()



def main():
    global I_n, shift_list, reduction_list, action_list, SR, RR
    GOTO()
    reduction()

    print("\n--------------------RULES--------------------")
    for item in rule_dict.items():
        print(item[1])
    print("\n------------AUGMENTED RULES------------------")
    for item in new_grammar:
        print(item.replace(".", ""))

    print("\n")
    print("Terminals:", terminals)
    print("NonTerminals:", non_terminals)

    print("\n------------------FOLLOW SET--------------------")
    for item in non_terminals:
        print(item,": ",follow(item))
    print("\n------------------FIRST SET--------------------")
    for item in non_terminals:
        print(item,": ",first(item))

    print("\n--------------------STATES--------------------")
    for item in I_n.items():
        print('S'+str(item[0])+':',item[1])

    print("\n--------------------GOTO OPERATIONS--------------------")
    for item in shift_list:
        print(item)



    print("\n--------------------REDUCTION--------------------")
    for item in reduction_list:
        print(item)

    print("\n-----------ACTION & GOTO TABLE--------------------")
    printGOTO_ACTION()

    if Conflict():
        if SR != []:
            print("SR conflict")
            for item in SR:
                print(item)
            print
        if RR != []:
            print("RR conflict")
            for item in RR:
                print(item)
            print
        exit(0)

    else:
        print("\nNO CONFLICT")

    action_list.extend(shift_list)
    action_list.extend(reduction_list)

    string = str(input("\n\nEnter String:: "))

    try:
        if string[-1] != "$":
            string = string + "$"
    except:
        print("InputError")
        exit(0)

    print("\nTest String:", string)
    result = test(string)

    if result == 1:
        print("---ACCEPTED---")
    elif result == 0:
        print("---NOT ACCEPTED---")
    return 0
```

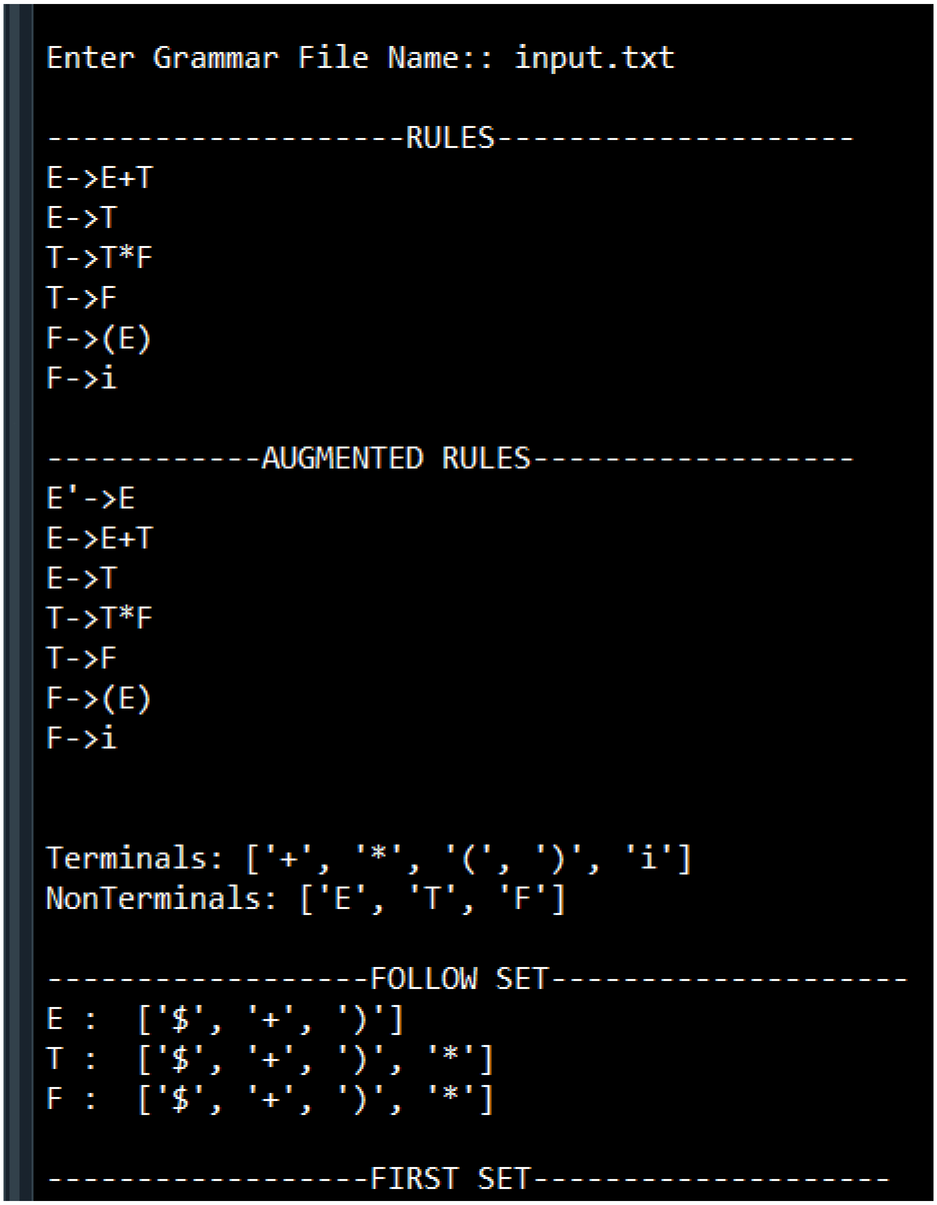
```
if __name__ == '__main__':
	main()
```

INPUT FILE (input.txt) [file containing valid grammar]:

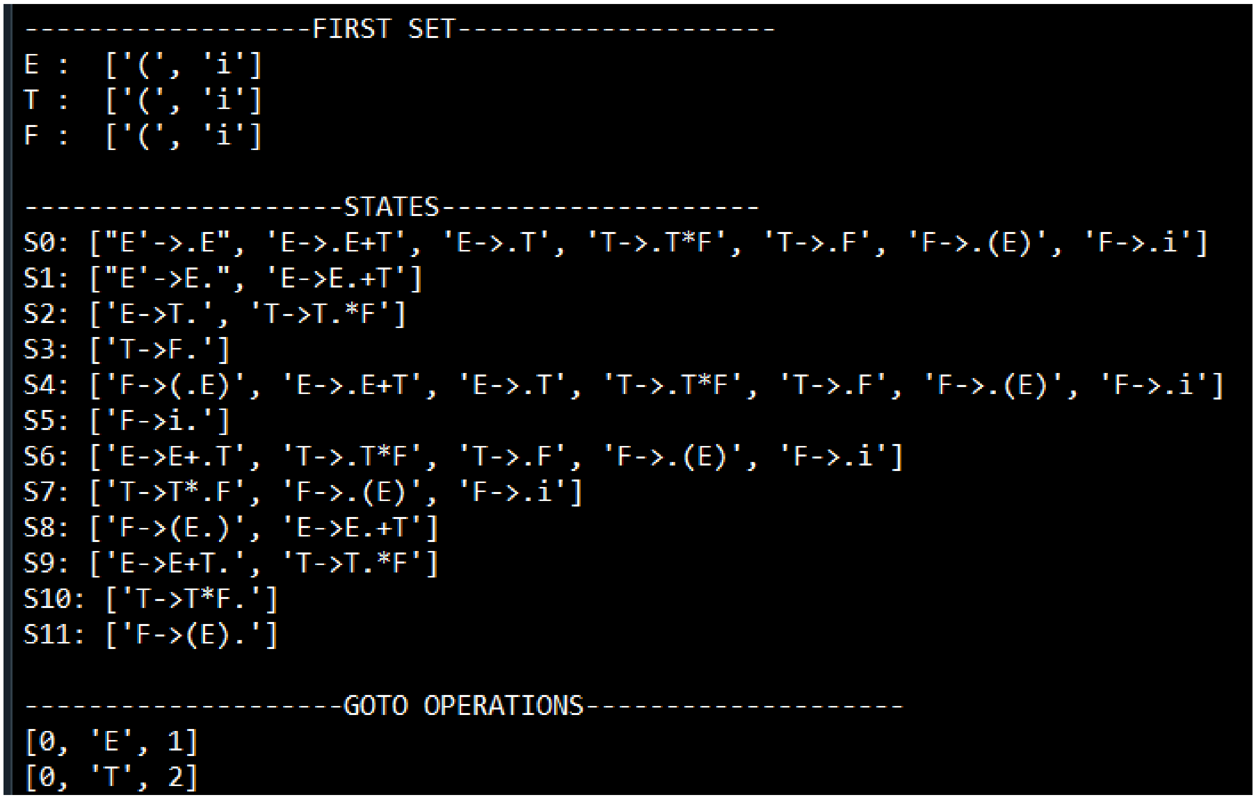


```
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
```

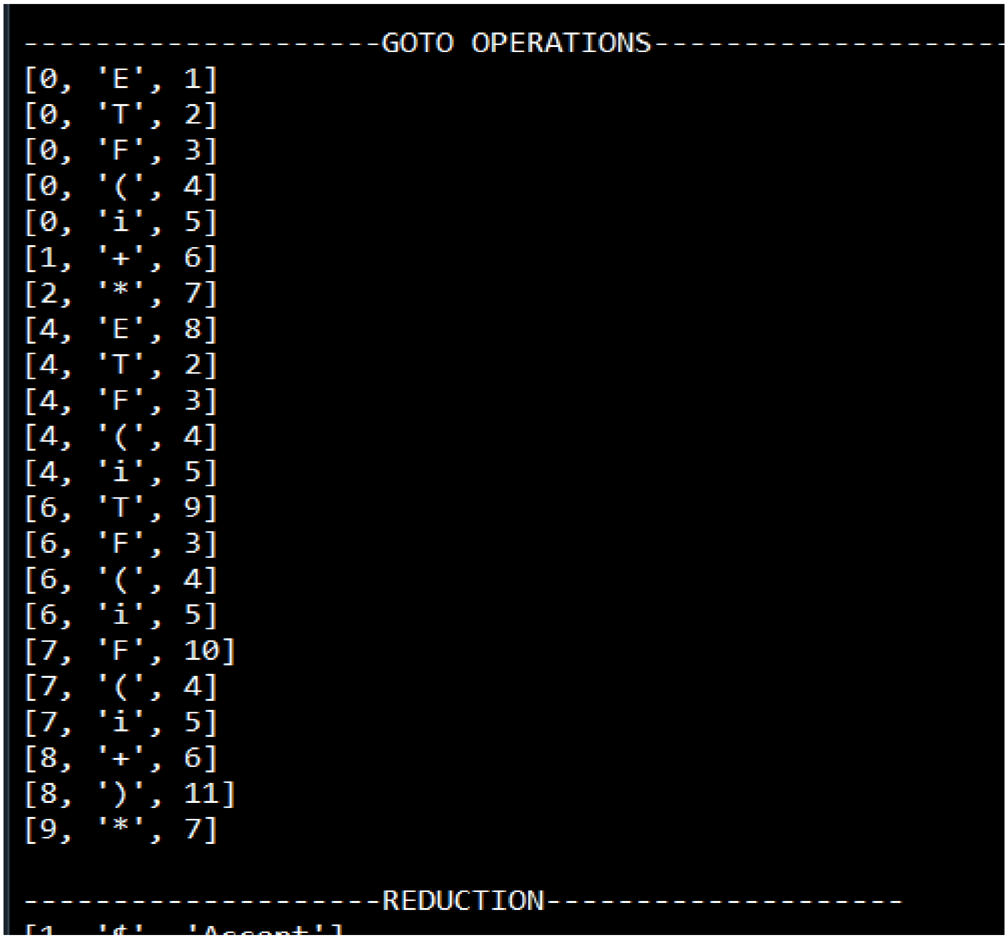
OUTPUT 1 (Acceptable String) :



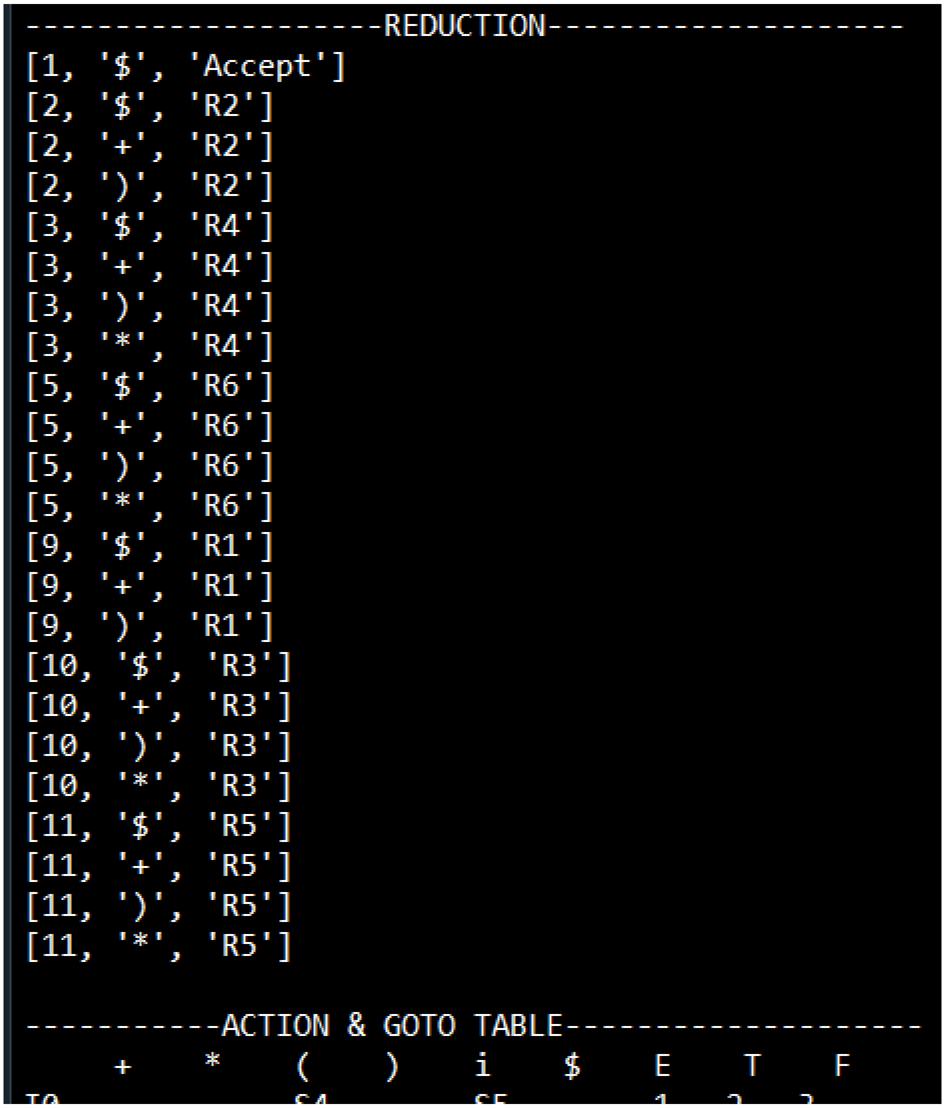
```
Enter Grammar File Name:: input.txt

--------------------RULES--------------------
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i

------------AUGMENTED RULES------------------
E'->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i


Terminals: ['+', '*', '(', ')', 'i']
NonTerminals: ['E', 'T', 'F']

------------------FOLLOW SET--------------------
E :  ['$', '+', ')']
T :  ['$', '+', ')', '*']
F :  ['$', '+', ')', '*']

------------------FIRST SET--------------------
```



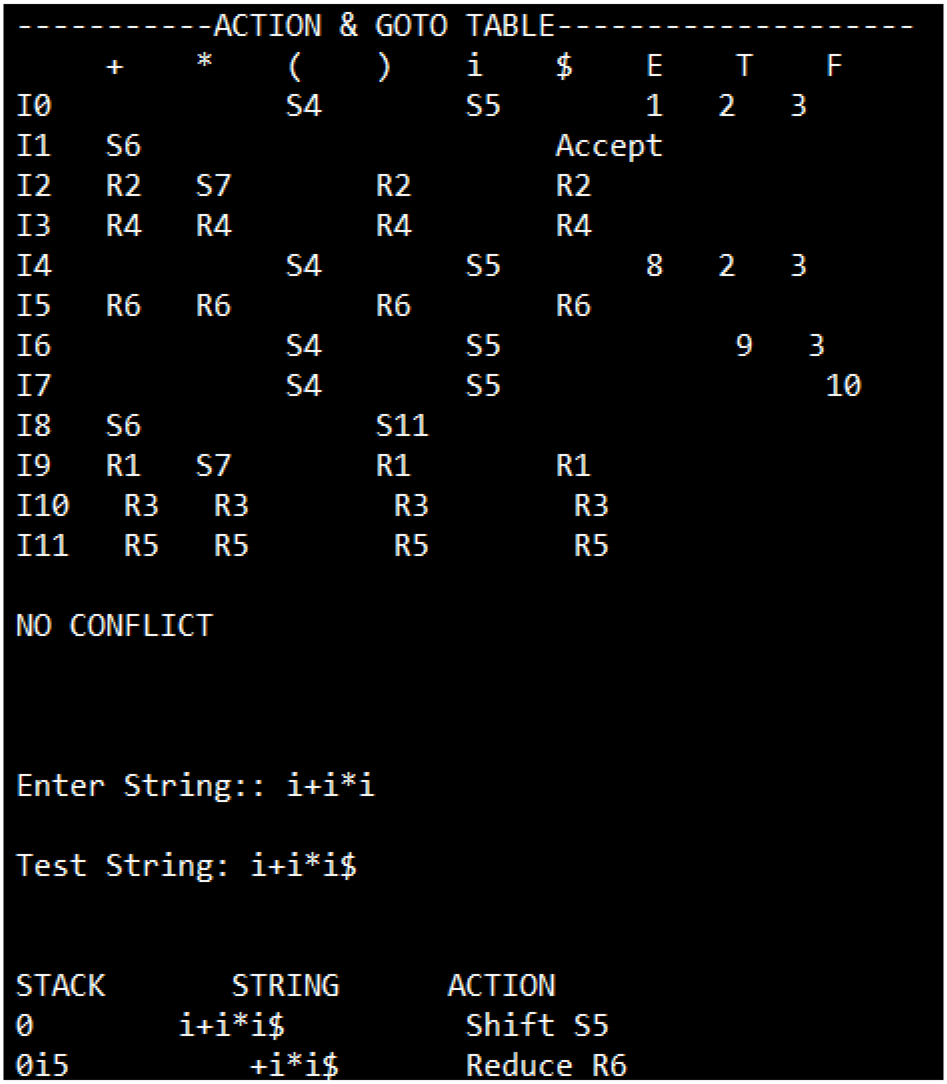
```
------------------FIRST SET--------------------
E :  ['(', 'i']
T :  ['(', 'i']
F :  ['(', 'i']

--------------------STATES-------------------
S0: ["E'->.E", 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i']
S1: ["E'->E.", 'E->E.+T']
S2: ['E->T.', 'T->T.*F']
S3: ['T->F.']
S4: ['F->(.E)', 'E->.E+T', 'E->.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i']
S5: ['F->i.']
S6: ['E->E+.T', 'T->.T*F', 'T->.F', 'F->.(E)', 'F->.i']
S7: ['T->T*.F', 'F->.(E)', 'F->.i']
S8: ['F->(E.)', 'E->E.+T']
S9: ['E->E+T.', 'T->T.*F']
S10: ['T->T*F.']
S11: ['F->(E).']

--------------------GOTO OPERATIONS--------------------
[0, 'E', 1]
[0, 'T', 2]
```



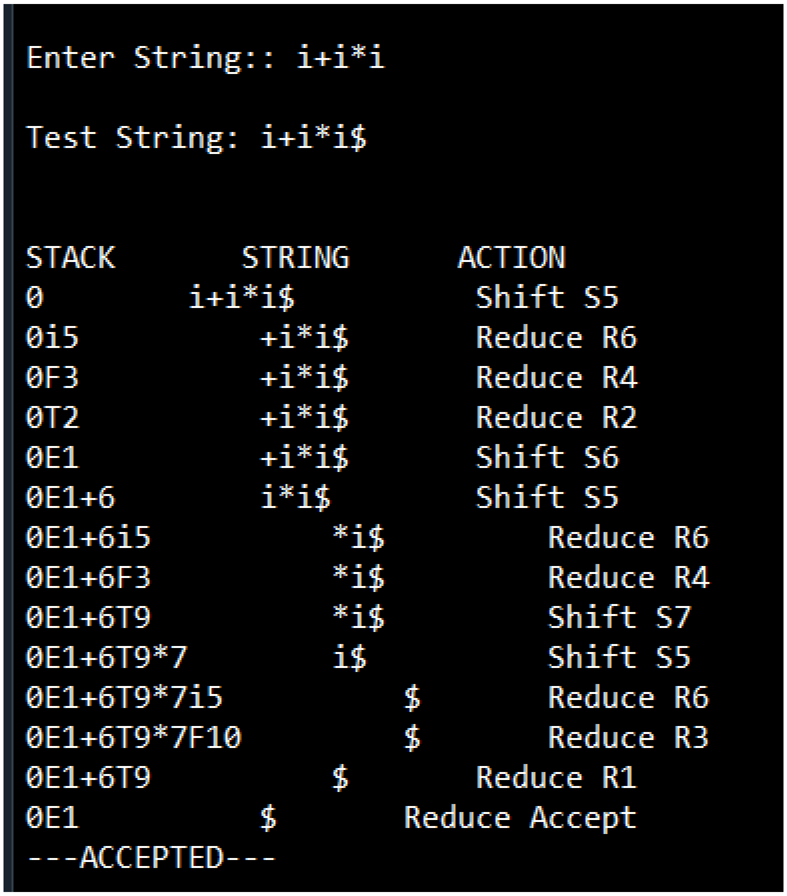
```
--------------------GOTO OPERATIONS--------------------
[0, 'E', 1]
[0, 'T', 2]
[0, 'F', 3]
[0, '(', 4]
[0, 'i', 5]
[1, '+', 6]
[2, '*', 7]
[4, 'E', 8]
[4, 'T', 2]
[4, 'F', 3]
[4, '(', 4]
[4, 'i', 5]
[6, 'T', 9]
[6, 'F', 3]
[6, '(', 4]
[6, 'i', 5]
[7, 'F', 10]
[7, '(', 4]
[7, 'i', 5]
[8, '+', 6]
[8, ')', 11]
[9, '*', 7]

--------------------REDUCTION--------------------
[1, '$', 'Accept']
```



```
--------------------REDUCTION--------------------
[1, '$', 'Accept']
[2, '$', 'R2']
[2, '+', 'R2']
[2, ')', 'R2']
[3, '$', 'R4']
[3, '+', 'R4']
[3, ')', 'R4']
[3, '*', 'R4']
[5, '$', 'R6']
[5, '+', 'R6']
[5, ')', 'R6']
[5, '*', 'R6']
[9, '$', 'R1']
[9, '+', 'R1']
[9, ')', 'R1']
[10, '$', 'R3']
[10, '+', 'R3']
[10, ')', 'R3']
[10, '*', 'R3']
[11, '$', 'R5']
[11, '+', 'R5']
[11, ')', 'R5']
[11, '*', 'R5']

-----------ACTION & GOTO TABLE--------------------
     +    *    (    )    i    $    E    T    F
I0             S4        S5        1   2   3
```

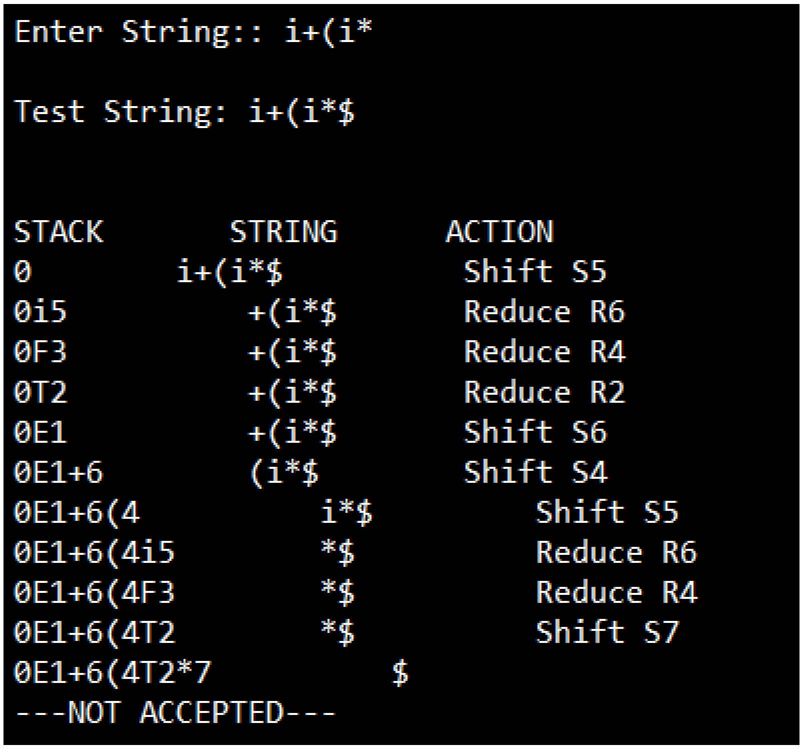


```
-----------ACTION & GOTO TABLE--------------------
     +    *    (    )    i    $    E    T    F
I0             S4        S5        1   2   3
I1   S6                       Accept
I2   R2   S7        R2        R2
I3   R4   R4        R4        R4
I4             S4        S5        8   2   3
I5   R6   R6        R6        R6
I6             S4        S5             9   3
I7             S4        S5                  10
I8   S6             S11
I9   R1   S7        R1        R1
I10   R3   R3         R3         R3
I11   R5   R5         R5         R5

NO CONFLICT



Enter String:: i+i*i

Test String: i+i*i$


STACK       STRING      ACTION
0        i+i*i$          Shift S5
0i5          +i*i$       Reduce R6
```



```
Enter String:: i+i*i

Test String: i+i*i$


STACK       STRING      ACTION
0        i+i*i$          Shift S5
0i5          +i*i$       Reduce R6
0F3          +i*i$       Reduce R4
0T2          +i*i$       Reduce R2
0E1          +i*i$       Shift S6
0E1+6        i*i$        Shift S5
0E1+6i5          *i$         Reduce R6
0E1+6F3          *i$         Reduce R4
0E1+6T9          *i$         Shift S7
0E1+6T9*7        i$          Shift S5
0E1+6T9*7i5          $       Reduce R6
0E1+6T9*7F10         $       Reduce R3
0E1+6T9          $       Reduce R1
0E1          $       Reduce Accept
---ACCEPTED---
```

OUTPUT 2 (INVALID TEST STRING):



```
Enter String:: i+(i*

Test String: i+(i*$


STACK       STRING      ACTION
0        i+(i*$          Shift S5
0i5          +(i*$       Reduce R6
0F3          +(i*$       Reduce R4
0T2          +(i*$       Reduce R2
0E1          +(i*$       Shift S6
0E1+6        (i*$        Shift S4
0E1+6(4          i*$         Shift S5
0E1+6(4i5        *$          Reduce R6
0E1+6(4F3        *$          Reduce R4
0E1+6(4T2        *$          Shift S7
0E1+6(4T2*7          $
---NOT ACCEPTED---
```