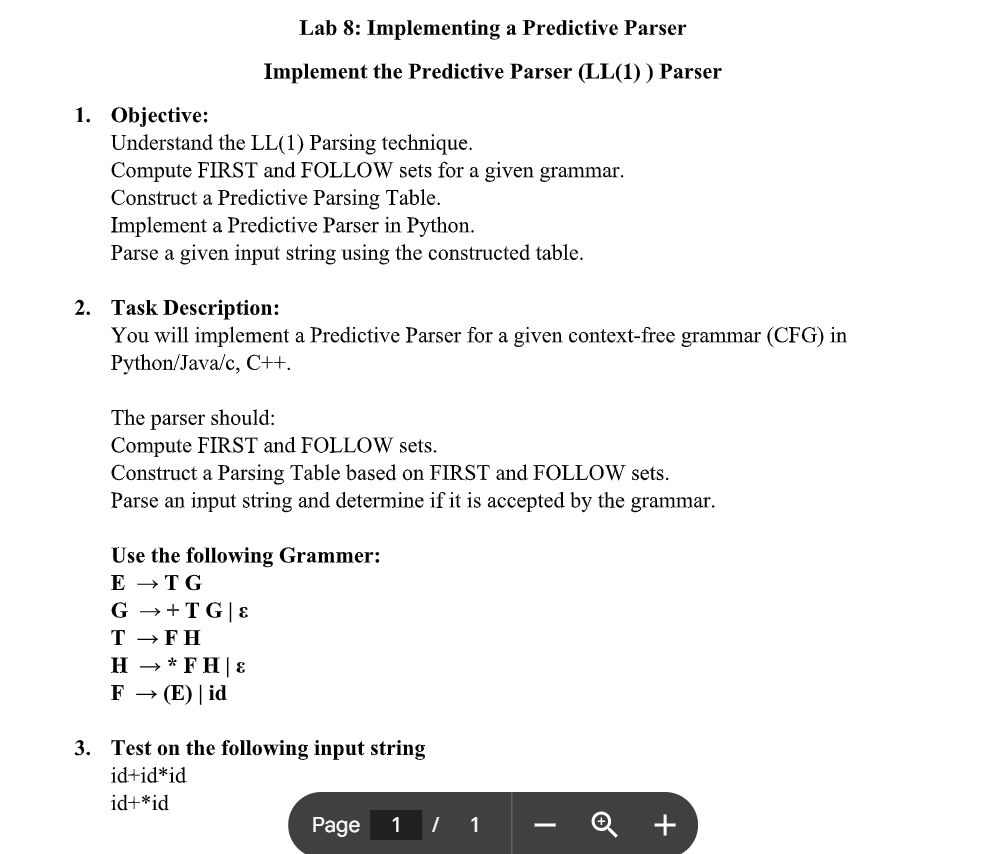
COMPILER DESGIN CODE :



*def* first(*rule*):

    global rules, nonterm\_userdef, term\_userdef, diction, firsts

    if len(*rule*) != 0 and (*rule* is not None):

        if *rule*[0] in term\_userdef:

            return *rule*[0]

        elif *rule*[0] == '#':

            return '#'

    if len(*rule*) != 0:

        if *rule*[0] in list(diction.keys()):

            fres = []

            rhs\_rules = diction[*rule*[0]]

            for itr in rhs\_rules:

                indivRes = first(itr)

                if type(indivRes) is list:

                    for i in indivRes:

                        fres.append(i)

                else:

                    fres.append(indivRes)

            if '#' not in fres:

                return fres

            else:

                newList = []

                fres.remove('#')

                if len(*rule*) > 1:

                    ansNew = first(*rule*[1:])

                    if ansNew != None:

                        if type(ansNew) is list:

                            newList = fres + ansNew

                        else:

                            newList = fres + [ansNew]

                    else:

                        newList = fres

                    return newList

                fres.append('#')

                return fres

*def* follow(*nt*):

    global start\_symbol, rules, nonterm\_userdef, term\_userdef, diction, firsts, follows

    solset = set()

    if *nt* == start\_symbol:

        solset.add('$')

    for curNT in diction:

        rhs = diction[curNT]

        for subrule in rhs:

            if *nt* in subrule:

                while *nt* in subrule:

                    index\_nt = subrule.index(*nt*)

                    subrule = subrule[index\_nt + 1:]

                    if len(subrule) != 0:

                        res = first(subrule)

                        if '#' in res:

                            newList = []

                            res.remove('#')

                            ansNew = follow(curNT)

                            if ansNew != None:

                                if type(ansNew) is list:

                                    newList = res + ansNew

                                else:

                                    newList = res + [ansNew]

                            else:

                                newList = res

                            res = newList

                    else:

                        if *nt* != curNT:

                            res = follow(curNT)

                    if res is not None:

                        if type(res) is list:

                            for g in res:

                                solset.add(g)

                        else:

                            solset.add(res)

    return list(solset)

*def* computeAllFirsts():

    global rules, nonterm\_userdef, term\_userdef, diction, firsts

    for rule in rules:

        k = rule.split("->")

        k[0] = k[0].strip()

        k[1] = k[1].strip()

        rhs = k[1]

        multirhs = rhs.split('|')

        for i in range(len(multirhs)):

            multirhs[i] = multirhs[i].strip()

            multirhs[i] = multirhs[i].split()

        diction[k[0]] = multirhs

    for y in diction:

        print(*f*"{y}->{diction[y]}")

    for y in list(diction.keys()):

        t = set()

        for sub in diction.get(y):

            res = first(sub)

            if res != None:

                if type(res) is list:

                    for u in res:

                        t.add(u)

                else:

                    t.add(res)

        firsts[y] = t

*def* computeAllFollows():

    global start\_symbol, rules, nonterm\_userdef, term\_userdef, diction, firsts, follows

    for NT in diction:

        solset = set()

        sol = follow(NT)

        if sol is not None:

            for g in sol:

                solset.add(g)

        follows[NT] = solset

*def* print\_first\_follow():

        global diction, firsts, follows, term\_userdef

        print("\nFirsts and Follow Result table\n")

        mx\_len\_first = 10

        mx\_len\_fol = 10

        print(*f*"{{:<{10}}} {{:<{mx\_len\_first + 5}}} {{:<{mx\_len\_fol + 5}}}".format("Non-T", "FIRST", "FOLLOW"))

        for u in diction:

            print(*f*"{{:<{10}}} {{:<{mx\_len\_first + 5}}} {{:<{mx\_len\_fol + 5}}}".format(u, str(firsts[u]), str(follows[u])))

*def* createParseTable():

    import copy

    global diction, firsts, follows, term\_userdef

    ntlist = list(diction.keys())

    terminals = copy.deepcopy(term\_userdef)

    terminals.append('$')

    mat = []

    for x in diction:

        row = []

        for y in terminals:

            row.append('')

        mat.append(row)

    grammar\_is\_LL = True

    for lhs in diction:

        rhs = diction[lhs]

        for y in rhs:

            res = first(y)

            if '#' in res:

                if type(res) == str:

                    firstFollow = []

                    fol\_op = follows[lhs]

                    if fol\_op is str:

                        firstFollow.append(fol\_op)

                    else:

                        for u in fol\_op:

                            firstFollow.append(u)

                    res = firstFollow

                else:

                    res.remove('#')

                    res = list(res) + list(follows[lhs])

            ttemp = []

            if type(res) is str:

                ttemp.append(res)

                res = copy.deepcopy(ttemp)

            for c in res:

                xnt = ntlist.index(lhs)

                yt = terminals.index(c)

                if mat[xnt][yt] == '':

                    mat[xnt][yt] = mat[xnt][yt] + *f*"{lhs}->{' '.join(y)}"

                else:

                    if *f*"{lhs}->{y}" in mat[xnt][yt]:

                        continue

                    else:

                        grammar\_is\_LL = False

                        mat[xnt][yt] = mat[xnt][yt] + *f*",{lhs}->{' '.join(y)}"

    print("\nGenerated parsing table:\n")

    frmt = "{*:>12*}" \* len(terminals)

    print(frmt.format(\*terminals))

    j = 0

    for y in mat:

        frmt1 = "{*:>12*}" \* len(y)

        print(*f*"{ntlist[j]} {frmt1.format(\*y)}")

        j += 1

    return (mat, grammar\_is\_LL, terminals)

*def* validateStringUsingStackBuffer(*parsing\_table*, *grammarll1*, *table\_term\_list*, *input\_string*, *term\_userdef*, *start\_symbol*):

    print(*f*"\nValidate String => {*input\_string*}\n")

    if *grammarll1* == False:

        return *f*"\nInput String = \"{*input\_string*}\"\nGrammar is not LL(1)"

    stack = [*start\_symbol*, '$']

    buffer = *input\_string*.split()

    buffer.reverse()

    buffer = ['$'] + buffer

    print("{*:>20*} {*:>20*} {*:>20*}".format("Buffer", "Stack", "Action"))

    while True:

        if stack == ['$'] and buffer == ['$']:

            print("{*:>20*} {*:>20*} {*:>20*}".format(' '.join(buffer), ' '.join(stack), "Valid"))

            return "\nValid String!"

        elif stack[0] not in *term\_userdef*:

            x = list(diction.keys()).index(stack[0])

            y = *table\_term\_list*.index(buffer[-1])

            if *parsing\_table*[x][y] != '':

                entry = *parsing\_table*[x][y]

                print("{*:>20*} {*:>20*} {*:>25*}".format(' '.join(buffer), ' '.join(stack), *f*"T[{stack[0]}][{buffer[-1]}] = {entry}"))

                lhs\_rhs = entry.split("->")

                lhs\_rhs[1] = lhs\_rhs[1].replace('#', '').strip()

                entryrhs = lhs\_rhs[1].split()

                stack = entryrhs + stack[1:]

            else:

                return *f*"\nInvalid String! No rule at Table[{stack[0]}][{buffer[-1]}]."

        else:

            if stack[0] == buffer[-1]:

                print("{*:>20*} {*:>20*} {*:>20*}".format(' '.join(buffer), ' '.join(stack), *f*"Matched:{stack[0]}"))

                buffer = buffer[:-1]

                stack = stack[1:]

            else:

                return "\nInvalid String! Unmatched terminal symbols"

# DRIVER CODE - MAIN

if \_\_name\_\_=="\_\_main\_\_":

    rules = [

                "E -> T G",

                "G -> + T G | #",

                "T -> F H",

                "H -> \* F H | #",

                "F -> ( E ) | id"

            ]

    nonterm\_userdef = ['E', 'G', 'F', 'T', 'H']

    term\_userdef = ['id', '+', '\*', '(', ')']

    sample\_input\_string = "id + id \* id"

    diction = {}

    firsts = {}

    follows = {}

    computeAllFirsts()

    start\_symbol = list(diction.keys())[0]

    computeAllFollows()

    print\_first\_follow()

    (parsing\_table, result, tabTerm) = createParseTable()

    if sample\_input\_string != None:

        validity = validateStringUsingStackBuffer(parsing\_table, result, tabTerm, sample\_input\_string, term\_userdef, start\_symbol)

        print(validity)

    else:

        print("\nNo input String detected")

OUTPUT :