Compiler Design

Exp–10 Intermediate Code Generation

Name:- K. DUSHYANT REDDY Reg No.:- RA1911033010029

Branch:- CSE-SE

Date:- 01/04/22

AIM: To implement Intermediate Code Generation for Quadruple, Triples and Indirect triples.

CODE:

#ICG- TAC,Quadruple,Triple,Indirect Triple

Symbols = set(['+', '-', '\*', '/', '(', ')'])

Case = {'+': 1, '-': 1, '\*': 2, '/': 2}

*def* infix\_to\_postfix(*cond*):

    stk=[]

    opt=''

    for x in *cond*:

        if x not in Symbols:

            opt +=x

        elif x == '(':

            stk.append('(')

        elif x == ')':

            while stk and stk[-1] != '(':

                opt += stk.pop()

            stk.pop()

        else:

            while stk and stk[-1] != '(' and PRI[x] <= PRI[stk[-1]]:

                opt += stk.pop()

            stk.append(x)

    while stk:

        opt += stk.pop()

    print(*f*'POSTFIX: {opt}')

    return opt

*def* infix\_to\_prefix(*cond*):

    op\_stk=[]

    exp\_stk=[]

    for x in *cond*:

        if not x in Symbols:

            exp\_stk.append(x)

        elif x == '(':

            op\_stk.append(x)

        elif x == ')':

            while op\_stk[-1] != '(':

                op=op\_stk.pop()

                p=exp\_stk.pop()

                q=exp\_stk.pop()

                exp\_stk.append(op+q+p)

            op\_stk.pop()

        else:

            while op\_stk and op\_stk[-1] != '(' and PRI[x] <= PRI[op\_stk[-1]]:

                op=op\_stk.pop()

                p=exp\_stk.pop()

                q=exp\_stk.pop()

                exp\_stk.append(op+q+p)

            op\_stk.append(x)

    while op\_stk:

        op=op\_stk.pop()

        p=exp\_stk.pop()

        q=exp\_stk.pop()

        exp\_stk.append(op+q+p)

    print(*f*'PREFIX: {exp\_stk[-1]}')

    return exp\_stk[-1]

*def* TAC(*pos*):

    print("Three Address Code:")

    exp\_stk=[]

    temp=1

    for i in *pos*:

        if i not in Symbols:

            exp\_stk.append(i)

        else:

            print(*f*'t{temp} := {exp\_stk[-2]} {i} {exp\_stk[-1]}')

            exp\_stk=exp\_stk[:-2]

            exp\_stk.append(*f*'t{temp}')

            temp += 1

exp=input("Enter Expression: ")

pre=infix\_to\_prefix(exp)

pos=infix\_to\_postfix(exp)

TAC(pos)

*def* Quadruple(*pos*):

    stk=[]

    op=[]

    x=1

    for i in *pos*:

        if i not in Symbols:

            stk.append(i)

        elif i == '-':

            op1 = stk.pop()

            stk.append("t(%s)" % x)

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}|{3*:4s*}".format(

                i, op1, "(-)", " t(%s)" % x))

            x=x+1

            if stk!=[]:

                op2=stk.pop()

                op1=stk.pop()

                print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}|{3*:4s*}".format(

                    "+", op1, op2, " t(%s)" % x))

                stk.append("t(%s)" % x)

                x=x+1

        elif i == '=':

            op2=stk.pop()

            op1=stk.pop()

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}|{3*:4s*}".format(i, op2, "(-)", op1))

        else:

            op1=stk.pop()

            op2=stk.pop()

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}|{3*:4s*}".format(

                i, op2, op1, " t(%s)" % x))

            stk.append("t(%s)" % x)

            x=x+1

*def* Triple(*pos*):

    stk=[]

    op=[]

    x=0

    for i in *pos*:

        if i not in Symbols:

            stk.append(i)

        elif i == '-':

            op1=stk.pop()

            stk.append("(%s)" % x)

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}".format(i, op1, "(-)"))

            x=x+1

            if stk != []:

                op2=stk.pop()

                op1=stk.pop()

                print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}".format("+", op1, op2))

                stk.append("(%s)" % x)

                x=x+1

        elif i == '=':

            op2=stk.pop()

            op1=stk.pop()

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}".format(i, op1, op2))

        else:

            op1=stk.pop()

            if stk != []:

                op2=stk.pop()

                print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*}".format(i, op2, op1))

                stk.append("(%s)" % x)

                x=x+1

*def* IndirectTriple(*pos*):

    stk=[]

    op=[]

    x=0

    ch=0

    for i in *pos*:

        if i not in Symbols:

            stk.append(i)

        elif i == '-':

            op1=stk.pop()

            stk.append("(%s)" % x)

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*} | {3*:^5d*}".format(i, op1, "(-)", ch))

            x=x+1

            if stack != []:

                op2=stk.pop()

                op1=stk.pop()

                print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*} | {3*:^5d*}".format(

                    "+", op1, op2, ch))

                stk.append("(%s)" % x)

                x=x+1

                ch=ch+1

        elif i == '=':

            op2=stk.pop()

            op1=stk.pop()

            print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*} | {3*:^5d*}".format(i, op1, op2, ch))

            ch=ch+1

        else:

            op1=stk.pop()

            if stk != []:

                op2=stk.pop()

                print("{0*:^4s*} | {1*:^4s*} | {2*:^4s*} | {3*:^5d*}".format(

                    i, op2, op1, ch))

                stk.append("(%s)" % x)

                x=x+1

                ch=ch+1

print("Quadruple: ")

print("Op   | Src1 | Src2| Res")

Quadruple(pos)

print("Triple: ")

print("Op   | Src1 | Src2")

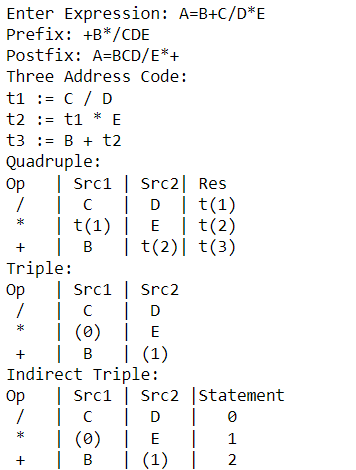
Triple(pos)

print("Indirect Triple: ")

print("Op   | Src1 | Src2 |Statement")

IndirectTriple(pos)

OUTPUT:



RESULT:

The code is successfully implemented in Python language.