Name: Dhroov Makwana
Gr no.: 22010538
Roll no.: 321042
Batch: A2

Assignment 5

Aim: Write a program to generate three address code for the simple expression.

Theory:

Three address code is a type of intermediate code which is easy to generate and can be easily converted to machine code.It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler. The compiler decides the order of operation given by three address code.

General representation:

a=b op c

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator.

Implementation of Three Address Code –

There are 3 representations of three address code namely

1. Quadruple

2. Triples

3. Indirect Triples

1. Quadruple –

It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.

Advantages –

• Easy to rearrange code for global optimization.

• One can quickly access value of temporary variables using symbol table.

Disadvantage –

• Contain lot of temporaries.

• Temporary variable creation increases time and space complexity.

2. Triples –

This representation doesn't make use of extra temporary variable to represent a single operation instead when a reference to another triple's value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2.

Disadvantage –

• Temporaries are implicit and difficult to rearrange code.

• It is difficult to optimize because optimization involves moving intermediate code. When a triple is moved, any other triple referring to it must be updated also. With help of pointer one can directly access symbol table entry.

3. Indirect Triples –

This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it. Temporaries are implicit and easier to rearrange code.

Code:

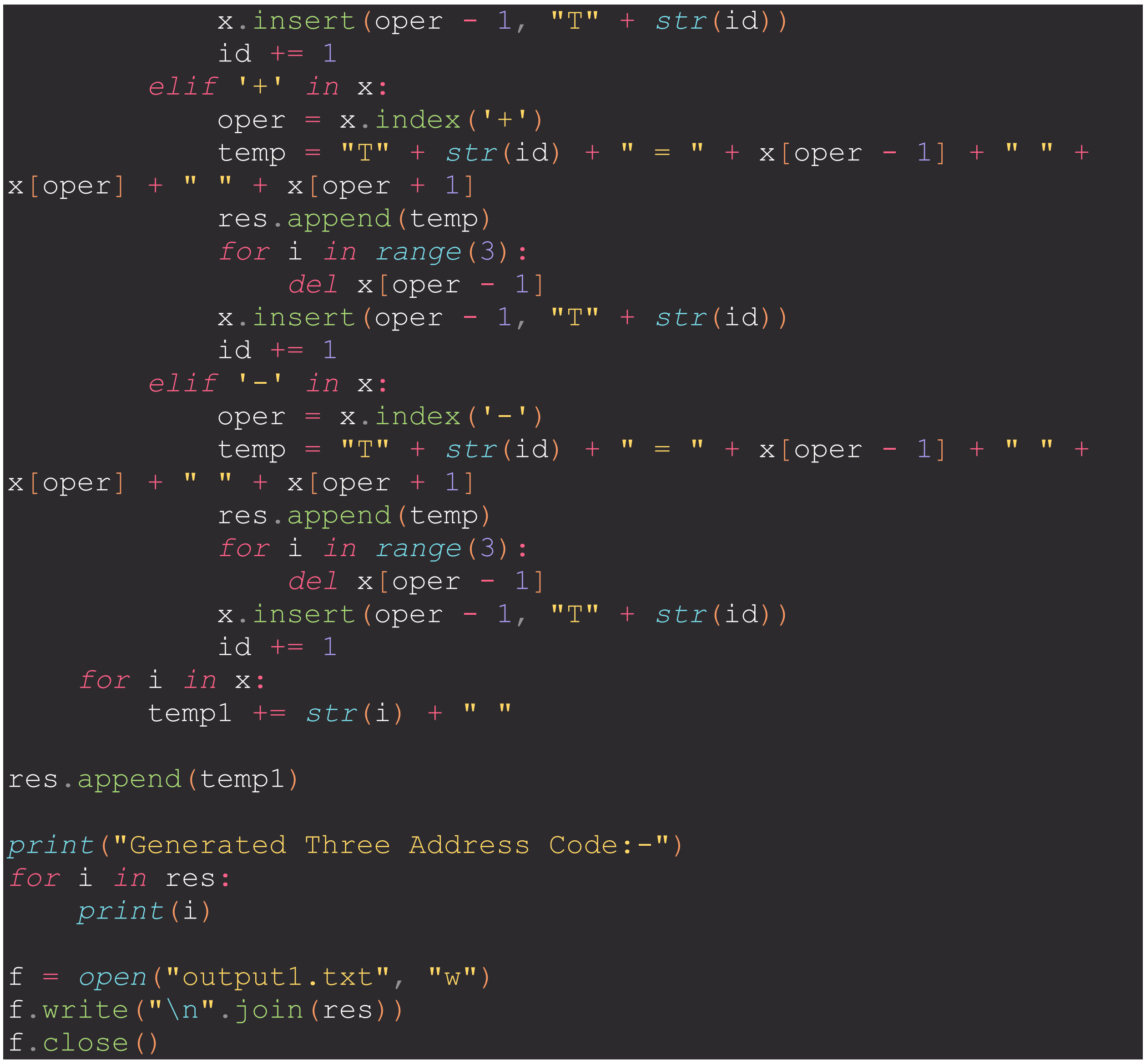
```
input_expression = "a = b * c + d * e + f + g"
print("\nInput Expression:-")
print(input_expression)
print()

arithmetic = ["*", "/", "+", "-", "^"]
extra = ["if", "else"]
assign = ["="]
res = []
```

```python
id = 1
temp1 = ""
expression = input_expression
if expression[0] not in extra:
    x = expression.split()
    temp0 = x[-2]
    x[-2] = "$"
    i = 2
    while x[i] != "$":
        if x[i] in arithmetic:
            if x[i + 1].startswith("-"):
                temp = "T" + str(id) + " = " + x[i + 1]
                res.append(temp)
                del x[i + 1]
                x.insert(i + 1, "T" + str(id))
                id += 1
        i += 1
    x[-2] = temp0

    while len(x) > 3:
        if x[2].startswith("-"):
            temp = "T" + str(id) + " = " + x[2]
            res.append(temp)
            del x[2]
            x.insert(2, "T" + str(id))
            id += 1

        elif '^' in x:
            oper = x.index('^')
            temp = "T" + str(id) + " = " + x[oper - 1] + " " +
x[oper] + " " + x[oper + 1]
            res.append(temp)
            for i in range(3):
                del x[oper - 1]
            x.insert(oper - 1, "T" + str(id))
            id += 1
        elif '/' in x:
            oper = x.index('/')
            temp = "T" + str(id) + " = " + x[oper - 1] + " " +
x[oper] + " " + x[oper + 1]
            res.append(temp)
            for i in range(3):
                del x[oper - 1]
            x.insert(oper - 1, "T" + str(id))
            id += 1
        elif '*' in x:
            oper = x.index('*')
            temp = "T" + str(id) + " = " + x[oper - 1] + " " +
x[oper] + " " + x[oper + 1]
            res.append(temp)
            for i in range(3):
                del x[oper - 1]
```



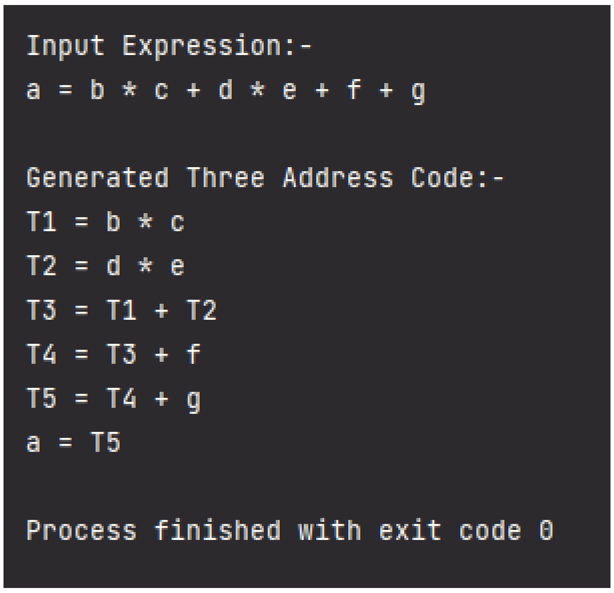
```
            x.insert(oper - 1, "T" + str(id))
            id += 1
        elif '+' in x:
            oper = x.index('+')
            temp = "T" + str(id) + " = " + x[oper - 1] + " " +
x[oper] + " " + x[oper + 1]
            res.append(temp)
            for i in range(3):
                del x[oper - 1]
            x.insert(oper - 1, "T" + str(id))
            id += 1
        elif '-' in x:
            oper = x.index('-')
            temp = "T" + str(id) + " = " + x[oper - 1] + " " +
x[oper] + " " + x[oper + 1]
            res.append(temp)
            for i in range(3):
                del x[oper - 1]
            x.insert(oper - 1, "T" + str(id))
            id += 1
    for i in x:
        temp1 += str(i) + " "

res.append(temp1)

print("Generated Three Address Code:-")
for i in res:
    print(i)

f = open("output1.txt", "w")
f.write("\n".join(res))
f.close()
```

Output:



```
Input Expression:-
a = b * c + d * e + f + g

Generated Three Address Code:-
T1 = b * c
T2 = d * e
T3 = T1 + T2
T4 = T3 + f
T5 = T4 + g
a = T5

Process finished with exit code 0
```