Name : Kshitij Narkhede

Roll No. : 322047

PRN No. : 22110941

Batch : B2

LPCC Assignment 3

Aim : WAP to Generate Symbol table

Objectives :

- To develop a program to generate a symbol table.
- To manage variable names and their corresponding attributes or values in a given programming language or context.

Theory :

- Symbol tables serve as fundamental data structures utilized by compilers to store and manage information pertaining to various identifiers encountered during the compilation process.
- Each entry in the symbol table typically comprises crucial details such as the identifier's name, its type, scope, and any associated attributes or values, enabling compilers to accurately resolve identifiers within the program.
- Symbol tables play a pivotal role in facilitating essential compiler tasks such as type checking, scope resolution, and code generation.
- By organizing identifiers systematically and associating them with relevant information, symbol tables enable compilers to perform these tasks efficiently and accurately.
- Structurally, symbol tables can be implemented using a variety of data structures, including hash tables, binary search trees, or linked lists, each offering distinct advantages in terms of lookup and manipulation efficiency.
- Throughout the compilation process, symbol tables are continuously updated as identifiers are encountered and processed, ensuring that the most current information is available for analysis and resolution.

Code :

```
#Symbol table generation
def getClass(operator):
    clas = 0
    for i in EMOT:
        if(i[0] == operator):
            clas = i[1]
    return clas
file = open("ASSEMBLY CODE.txt", "r")
lines = file.readlines()
tokens = []
for line in lines:
    tokens.append(line.split())
EMOT = [['STOP', 1, 0], ['ADD', 1, 1], ['SUB', 1, 2], ['MULT', 1, 3], ['MOVER', 1, 4], ['MOVEM', 1, 5], ['COMP', 1, 6], ['BC', 1, 7],
lc = int(tokens[0][-1])
lcList = []
for i in tokens:
    lcList.append(lc)
    length = len(tokens)
    if(length == 4):
        operator = i[1]
        clas = getClas(operator)
        if(clas == 1):
            lc+=1
        elif(clas == 2):
            if(operator == 'DS'):
                incr = int(i[-1])
                lc+=incr
```

```
                else:
                    lc+=1
        else:
            if('DS' in i or 'DC' in i):
                operator = i[1]
            else:
                operator = i[0]
            clas = getClass(operator)
            if(clas == 1):
                lc+=1
            elif(clas == 2):
                if(operator == 'DS'):
                    incr = int(i[-1])
                    lc+=incr
                else:
                    lc+=1
            elif(clas == 3):
                pass

n = len(tokens)
st = []
entered_symbol = []
entered_label = []
for i in tokens:
    length = len(i)
    if(length == 4):
        label = i[0]
```

```
        label = i[0]
        if(label not in entered_label):
            index = tokens.index(i)
            lc = lcList[index]
            st.append([label, lc])
            entered_label.append(label)
        symbol = i[-1]
        if(symbol[0] != '='):
            if(symbol not in entered_symbol):
                st.append([symbol, 0])
                entered_symbol.append(symbol)
            if('DS' in i or 'DC' in i):
                operator = i[2]
            else:
                operator = i[1]
            if(operator == 'DS'):
                incr = int(i[-1])
                for s in st:
                    index = tokens.index(i)
                    lc = lcList[index]
                    if s[0] == symbol:
                        s[1] = lc
            else:
                for m in EMOT:
                    if (m[0] == operator):
                        clas = m[1]
```

```
            if(clas == 1):
                pass
    else:
        if('DS' in i or 'DC' in i):
            operator = i[1]
        else:
            operator = i[0]
        for m in EMOT:
            if(m[0] == operator):
                clas = m[1]
        if(clas == 5):
            pass
        elif(clas == 1):
            symbol = i[-1]
            if(symbol not in entered_symbol and symbol not in entered_label and symbol[0] != '='):
                st.append([symbol, 0])
                entered_symbol.append(symbol)
        elif(clas == 2):
            #EQU and DS
            symbol  = i[0]
            index = tokens.index(i)
            lc = lcList[index]
            for s in st:
                if(s[0] == symbol and symbol not in entered_label and symbol[0] != '='):
                    s[1] = lc
                    if(operator == 'DS'):
                        pass
```

```
                    else:
                        pass

for i in st:
    print(i)
```

Sample output :

```
['L1', 200]
['X', 202]
['Y', 203]
```

Conclusion :

- In conclusion, the program to generate a symbol table successfully fulfills its objective of managing variable names and their attributes.
- The program effectively organizes identifiers and associated information, enabling compilers to perform essential tasks such as type checking, scope resolution, and error detection