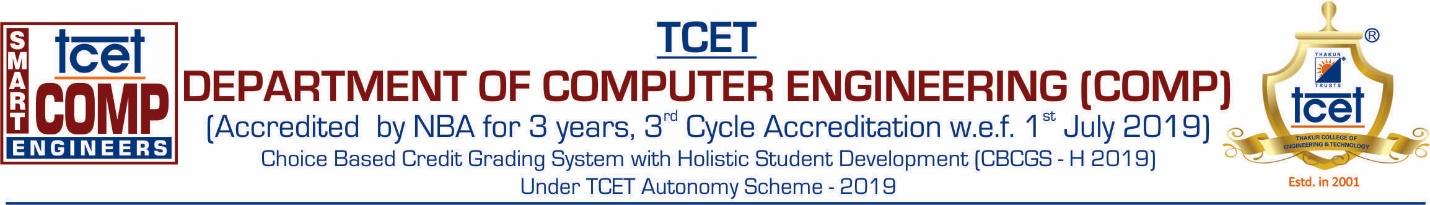
****

**Experiment 04 : Single Pass MacroProcessor**

**Learning Objective**: Student should be able to Apply single pass Macro Processor.

**Tools:** Jdk1.8,Turbo C/C++, Python, Notepad++

**Theory:**

IMPLEMENTATION A SINGLE PASS ALGORITHM Definition of a macro within other macro is possible in case of one pass macro processor. Here the inner macro is defined only after the outer one has been called: in order to provide for any use of the inner macro, we would have to repeat the both the macro definition and the macro call passes. This can be assumed by considering that the macros are never called before they are defined.

Here we make use of additional data structures like macro definition indicator (MDI) and macro definition level counter (MDLC). The MDI and MDLC are the switches used to keep track of macro calls and macro definition.

The MDI has status “ON” during the expansion of macro call and the value “OFF” all the other times. When its value is “ON” the cards are read from the MDT and when it is “OFF” the cards are read from the input source card. The use of MDLC is used keep track of the level of macros while defining the macros. Initially it is zero and it is incremented each time a MACRO code is found within a macro. The reverse process happens in case of MEND i.e. the valued of MDLC is decremented by one each time it encounters a MEND and the process continues till the MDLC is zero i.e. the completion of macro definition.

**ALGORITHM**

The process of one pass macro process can be clearly understood with the help of a MAIN algorithm that make use of a sub algorithm named READ.

**READ: (Macro call expansion or read a next instruction form the source input card)**

1. If MDI =”OFF”, then

a. Read next source card from input file.

b. Return to MAIN algorithm.

2. Else increment MDT pointer to next entry MDTP<-MDTP+1.

3. Get next card from MDT.

4. Substitute arguments from macro call.

5. If MEND pseudo code

a. Then if MDLC=0, i. Then MDI<-“OFF”. ii. GOTO 1.a.

b. Else return to MAIN algorithm.

6. Else if AIF or AGO present

a. Then process AIF or AGO and update MDTP.

b. Return to MAIN algorithm. 7. Else return to MAIN algorithm.

**MAIN: (One pass macro processor)**

1. Initialize MDTC and MNTC to 1, MDI to “OFF” and MDLC to 0.

2. READ

3. Search MNT for match with operation code.

4. If macro name found

a. MDI<-“ON”.

b. MDTP<-MDT index from MNT entry.

c. Setup macro call ALA.

d. GOTO 2.

5. Else if MACRO pseudo code

a. Then READ. //macro name line.

b. Enter macro name and current value of MDTC in MNT entry number MNTC.

c. Increment MNTC <- MNTC+1.

d. Prepare macro definition ALA.

e. Enter macro name card into MDT.

f. MDTC<-MDTC+1.

g. MDLC<-MDLC+1.

h. READ.

i. Substitute index notation for arguments in definition.

j. Enter line into MDT.

k. MDTC<-MDTC+1

l. If MACRO pseudo code

i. MDLC<-MDLC+1

ii. GOTO 5.h.

m. Else If MEND pseudo code i. Then MDLC<-MDLC-1

1. If MDLC=0 the a. Then GOTO 2.

2. Else GOTO 5.h. n. Else GOTO 5.h.

6. Write into expanded source card file.

7. If END pseudo code

a. Then Supply expanded source file to assembler processing.

8. Else GOTO 2.

**Application:** To design Single pass macroprocessor for X86 processor.

**Design:**

import re

f\_input = open("macro\_input.txt")

inputcode = list(line.strip() for line in f\_input)

MDT = []

MNT = {}

ALA\_list = {}

input\_for\_pass\_2 = []

iterator = iter(inputcode)

while True:

try:

line = next(iterator)

if line == "MACRO":

nameline = next(iterator)

nameline = re.split('[,\s]', nameline)

macro\_name = ""

for token in nameline:

if "&" not in token:

macro\_name = token

break

MNT[macro\_name] = len(MDT)

ALA = {}

arg\_counter = 0

for token in nameline:

if token is not macro\_name:

arg\_counter += 1

ALA[token] = "#" + str(arg\_counter)

nameline[nameline.index(token)] = ALA[token]

ALA\_list[macro\_name] = ALA

MDT.append(nameline)

while True:

macroline = next(iterator)

for argument in ALA.keys():

if argument in macroline:

macroline = macroline.replace(argument, ALA[argument])

MDT.append(macroline)

if macroline == "MEND":

break

else:

input\_for\_pass\_2.append(line)

except StopIteration:

break

print("\nMNT is ")

for line in MNT.items():

print(line)

print("\nMDT is ")

for line in MDT:

print(line)

print("\nALAs are ")

for line in ALA\_list.items():

print(line)

iterator = iter(input\_for\_pass\_2)

print("\nFinal Output is ")

while True:

try:

line = next(iterator)

line = re.split('[,\s]', line)

if any(word in line for word in MNT.keys()):

macroname = ""

if line[0] in MNT.keys():

macroname = line[0]

else:

macroname = line[1]

label = line[0]

actual\_args = []

for token in line:

if not token == macroname:

actual\_args.append(token)

ALA = ALA\_list[macroname]

ALA = {val: key for key, val in ALA.items()}

formal\_args = sorted(list(ALA.keys()))

for i in range(len(formal\_args)):

ALA[formal\_args[i]] = actual\_args[i]

MDTP = MNT[macroname] + 1

while "MEND" not in MDT[MDTP]:

line = MDT[MDTP]

for formal\_arg, actual\_arg in ALA.items():

line = line.replace(formal\_arg, actual\_arg)

print(line)

MDTP += 1

else:

print(" ".join(line))

except StopIteration:

break

macro\_input.txt

MACRO

INCR &ARG1

L AX,&ARG1

A AX,1

MEND

MACRO

FOOBAR &ARG1,&ARG2

L AX,&ARG1

L BX,&ARG2

ST AX,BX

MEND

MACRO

&LAB HARAMBE &ARG1

&LAB SR &ARG1,1

RR &ARG1,2

MEND

START 0

INCR 69

FOOBAR 69,96

LOOP HARAMBE 69

DC F'69'

END

**Result and Discussion:**

MNT is

('INCR', 0)

('FOOBAR', 4)

('HARAMBE', 9)

MDT is

['INCR', '#1']

L AX,#1

A AX,1

MEND

['FOOBAR', '#1', '#2']

L AX,#1

L BX,#2

ST AX,BX

MEND

['#1', 'HARAMBE', '#2']

#1 SR #2,1

RR #2,2

MEND

ALAs are

('INCR', {'&ARG1': '#1'})

('FOOBAR', {'&ARG1': '#1', '&ARG2': '#2'})

('HARAMBE', {'&LAB': '#1', '&ARG1': '#2'})

input pass 2 ['START 0', 'INCR 69', 'FOOBAR 69,96', 'LOOP HARAMBE 69', "DC F'69'", 'END']

Final Output is

START 0

L AX,69

A AX,1

L AX,69

L BX,96

ST AX,BX

LOOP SR 69,1

RR 69,2

DC F'69'

END

**Learning Outcomes:** The student should have the ability to

LO1: ***Describe*** the different database formats of Single pass Macro processor with the help of examples.

LO2: **Design** Single pass Macro processor for X86 machine.

LO3: ***Develop*** Single Pass Macro processor for X86 machine.

LO4: Illustrate the working of Single Pass Macro-processor.

**Course Outcomes**: Upon completion of the course students will be able to Use of macros in modular programming design

**Conclusion:**

The code is implementing a macro processor using two-pass algorithm in Python. It reads macro definitions, stores them in MDT and MNT, and replaces macro calls with their expanded version using the ALA.

For Faculty Use

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** |  |
| **Marks Obtained** |  |  |  |