Ayush Raghuvanshi
101850 Comps B2

SPCC
Praes

→ **Aim:** To design & implement a single pass macro processor to handle recursive & nested macro calls.

→ **CO: CO2:** Construct different databases of single pass macro processor.

→ **Theory:** A macro processor is a program that copies a stream of text from one place to another making a systematic set of replacements as it does so. Macroprocessors are often embedded in other programs such as assemblers & compilers.

The basic steps that any macroprocessor should perform are as:

1) Recognize macro definitions
Proper match of all the nested & recursive macro calls should be done & macroprocessor should identify macro definitions given by op-code MACRO & MEND.

2. Recognize calls:
The processor should recognise all the macro calls that appear as mnemonic op-codes.

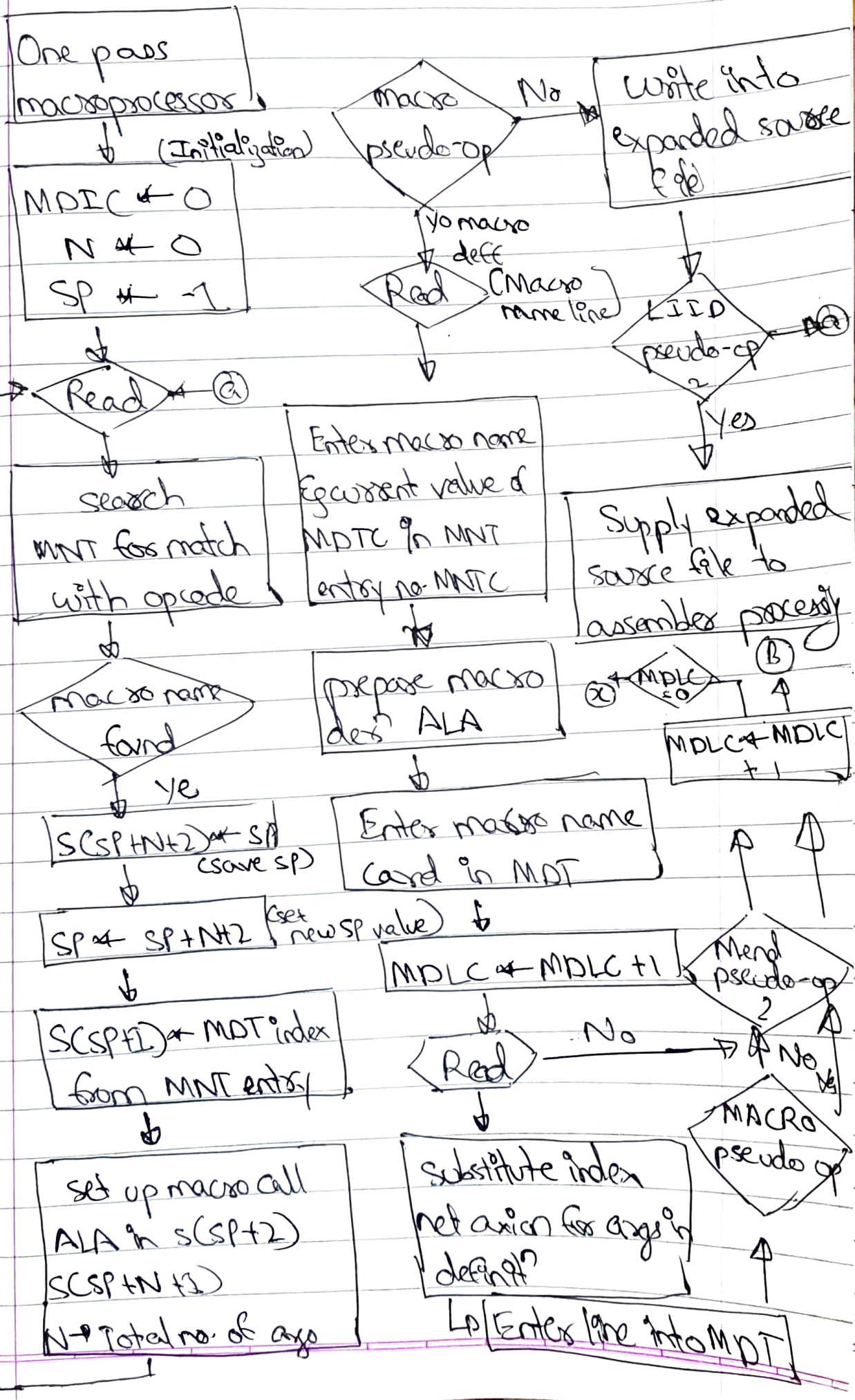
3. Expand calls & substitute arguments:
Macro definition arguments are replaced by the corresponding macro calls.

~~* Concept~~

* Algorithms
1) Define procedure
2) Enter macro name into MNT & enter macro prototype into MDT
3) set level = 1
4) while level > 0 do: getline
5) If not comment line then begin: substitute positional notation for each parameter.
6) IF OPCODE = 'MACRO' then
level = level + 1
else if opcode = 'MEND' then
level = level - 1
7) END all conditions
8) END

* Conclusion: Thus, we have studied & implemented the one pass macro processor to handle recursive & nested macro calls.

Flowchart:


One pass macroprocessor
(Initialization)
MDTC ← 0
N ← 0
SP ← -1
Read
@
search MNT for match with opcode
macro name found
ye
S(SP+N+2) ← SP
(save SP)
SP ← SP+N+2
(set new SP value)
S(SP+1) ← MDT index from MNT entry
set up macro call
ALA in S(SP+2)
S(SP+N+1)
N → Total no. of args
macro pseudo-op
No
Write into expanded source file
Yo macro defn
Read
(Macro name line)
Enter macro name & current value of MDTC in MNT entry no. MNTC
prepare macro def ALA
Enter macro name card in MDT
MDLC ← MDLC + 1
Read
No
Substitute index not arion for args in defn
Enter line into MDT
END pseudo-op
2
Yes
Supply expanded source file to assembler process
MDLC = 0
B
MDLC ← MDLC + 1
Mend pseudo-op
2
No
MACRO pseudo op

Ayush Raghuwanshi
101850

Practical no. 2:

Design and Implement single pass macro processor to handle recursive and nested calls

Program:

```
inp_file  = open('src_code.txt')

macro_start_flag = 0
macro_name_flag = 0
MDT = dict()
MNT = dict()
ALA = dict()
macro_name = ''
mdt_index = 0
mnt_index = 0
ala_index = 0
macro_def = []
macro_args = []
call_count = 0

# reading through input and creating MDT table
for line in inp_file:
    if len(line.split()) > 0:
        line = line.replace('\n', '')
        if line.split()[0] == 'MACRO':
            macro_name_flag = 0
            macro_start_flag = 1
            pass
        elif line.split()[0] == 'MEND':
            mdt_index += 1
            macro_line = (mdt_index, line)
            macro_def.append(macro_line)
            mdt_entry = {
                macro_name: macro_def
            }
            MDT.update(mdt_entry)
            macro_def = []
            macro_args = []
            macro_start_flag = 0
        else:
            if macro_start_flag != 0:
                if macro_name_flag == 0:
                    macro_name = line.split()[0]
                    mdt_ent = {
                        macro_name: None
                    }
                    MDT.update(mdt_ent)
                    macro_name_flag = 1
                mdt_index += 1
                for code in line.split():
                    # sustituting '#indx' for arguments
                    if '&' in code:
                        if code not in [arg[0] for arg in
macro_args]:
                            arg_sub =
'#'+str(len(macro_args)+1)
                            macro_args.append((code,
arg_sub))
                        else:
                            for args in macro_args:
                                if args[0] == code:
                                    arg_sub = args[1]
                        line = line.replace(code, arg_sub)
                    macro_line = (mdt_index, line)
                    macro_def.append(macro_line)
                else:
                    # initializing ALA
                    macro_ala = []
                    macro_call = line.split()[0]
                    if macro_call in list(MDT.keys()):
                        call_count += 1
                        arguments = line.split()[1:]
                        for arg in arguments:
                            arg = arg.upper()
                            if len(arg) < 8:
                                additional_b = 8 - len(arg)
                                for b in range(additional_b):
                                    arg += 'b'
                            macro_ala.append(arg)
                        ala_entry = {
                            macro_call+'_'+str(call_count):
macro_ala
                        }
                        ALA.update(ala_entry)

# creating MNT table
for entry in MDT:
    mnt_index += 1
    mnt_entry = {
        entry: (mnt_index, MDT[entry][0][0])
    }
    MNT.update(mnt_entry)

# adding values in ALA
ALA_final = ALA.copy()
for calls in ALA:
    macro_ala = []
    macro_call = calls.split('_')[0]
    given = ALA[calls]
    for line in MDT[macro_call]:
        if macro_call not in line[1] and 'MEND' not in
line[1]:
            for code in line[1].split():
                if '#' in code:
                    index = int(code.replace('#', '')) - 1
                    macro_ala.append(given[index])
                next_call = line[1].split()[0]
    call_count += 1
    next_call += '_' + str(call_count)
    ala_entry = {
        next_call: macro_ala
    }
    ALA_final.update(ala_entry)
```

```
# printing output
print('\nMacro Definition Table (MDT)')
print('Index\tContents')
for entry in MDT:
    for lines in MDT[entry]:
        print(lines[0], '\t', lines[1])

print('\nMacro Name Table (MNT)')
print('Index\tMacro Name\tMDT Index')
for entry in MNT:
    print(MNT[entry][0], '\t', entry,'\t\t', MNT[entry][1])

print('\nArgument List Array (ALA)')
print('Index\tArgument')
for entry in ALA_final:
    for arg in ALA_final[entry]:
        ala_index += 1
        print(ala_index, '\t', arg)
```

Input source Program:

```
MACRO
ADD1 &arg
L 1 &arg
A 1 =F'1'
ST 1 &arg
MEND

MACRO
ADDS &arg1 &arg2 &arg3
ADD1 &arg1
ADD1 &arg2
ADD1 &arg3
MEND

ADDS data1 data2 data3
```

Output:

```
Macro Definition Table (MDT)
Index   Contents
1        ADD1 #1
2        L 1 #1
3        A 1 =F'1'
4        ST 1 #1
5        MEND
6        ADDS #1 #2 #3
7        ADD1 #1
8        ADD1 #2
9        ADD1 #3
10       MEND

Macro Name Table (MNT)
Index   Macro Name      MDT Index
1        ADD1            1
2        ADDS            6

Argument List Array (ALA)
Index   Argument
1        DATA1bbb
2        DATA2bbb
3        DATA3bbb
4        DATA1bbb
5        DATA2bbb
6        DATA3bbb
```