T. Y. B. Tech Computer Engineering

Student Name	Pranav Dambe (Nikam)
SRN No	202201704
Roll No	68
PRN	2280030506
Division	D(D3)
Subject	System Programming
Year	Third Year

Assignment - 5

QUE 1 :

Design suitable data structures and implement simple Macro expansion for the hypothetical ALP. Generate Actual parameter table, Input file contains multiple macro calls (Minimum 3) . Assume macro definitions are stored in MDT created on Assignment 4.

Output : Submit a single .doc / .pdf file containing input , MNT, MDT, APTAB and expanded code.

OUTPUT :

ALP , MACRO, MACRO CALL :

input.asm

```
MACRO
SAMPLE &X, &N, &REG=AREG
LCL &M
&M SET 0
MOVER &REG, ="0"
.MORE MOVEM &REG, &X + &M
&M SET &M + 1
AIF (&M NE 10) .MORE
MEND

MACRO
DEC &Q, &REG1=BREG
LCL &Z
&Z SET 5
MOVER AREG, &REG1
.NEXT SUB &Q, &Z
&Z SET &Z - 1
AIF (&Z GT 0) .NEXT
MEND

START
MOV AREG, BREG
MUL AREG, NUM
SAMPLE P1, NUM, &REG=CREG
NUM DC 10
DEC ALPHA, &REG1=BEAG
END
```

MNT Table :

MNT:

Index	MACRO	#PP	#KP	#EV	MDTP	KPDTP	SSTP
1:	SAMPLE	2	1	1	1	1	1
2:	DEC	1	1	1	8	2	2

MDT Table :

```
MDT:

Index   MACRO Definition

1:      LCL     (E, 1)
2:      (E,1)   SET     0
3:      MOVER   (P,3)   ="0"
4:      (S, 1)  MOVEM   (P,3)   (P,1)   +       (E,1)
5:      (E,1)   SET     (E,1)   +       1
6:      AIF     (E,1)   NE      10      (S, 1)
7:      MEND
8:      LCL     (E, 2)
9:      (E,2)   SET     5
10:     MOVER   AREG    (P,5)
11:     (S, 2)  SUB     (P,4)   (E,2)
12:     (E,2)   SET     (E,2)   -       1
13:     AIF     (E,2)   GT      0       (S, 2)
14:     MEND
```

PNTAB :

```
PNTAB:

Index   Parameter Name

1:      X
2:      N
3:      REG
4:      Q
5:      REG1
```

SSNTAB :

```
SSNTAB:

Index   SS Name

1:      MORE
2:      NEXT
```

KPDTAB :

```
KPDTAB:

Index   Parameter       Value

1:      REG     AREG
2:      REG1    BREG
```

SSTAB :

```
SSTAB:

Index   MDT_ENTRY

1:      4
2:      11
```

APTAB :

```
APTAB:

Index    Parameter      Value

1:       X      P1
2:       Q      ALPHA
3:       REG1   BEAG
4:       REG    CREG
5:       N      NUM
```

Expanded Code :

Output.txt

```
START
MOV AREG, BREG
MUL AREG, NUM
MOVER CREG ="0"
MOVEM CREG P1 + 0
MOVEM CREG P1 + 1
MOVEM CREG P1 + 2
MOVEM CREG P1 + 3
MOVEM CREG P1 + 4
MOVEM CREG P1 + 5
MOVEM CREG P1 + 6
MOVEM CREG P1 + 7
MOVEM CREG P1 + 8
MOVEM CREG P1 + 9
NUM DC 10
MOVER AREG BEAG
SUB ALPHA 5
SUB ALPHA 4
SUB ALPHA 3
SUB ALPHA 2
SUB ALPHA 1
END
```

CODE :

Class MacroPass :

```
class MacroPass
{
    ArrayList<String> MNT = new ArrayList<>();
    ArrayList<String> MDT = new ArrayList<>();
    ArrayList<String> PNTAB = new ArrayList<>();
    ArrayList<String> EVNTAB = new ArrayList<>();
    ArrayList<String> SSNTAB = new ArrayList<>();
    ArrayList<String> KPDTAB = new ArrayList<>();
    ArrayList<Integer> SSTAB =  new ArrayList<>();
    HashMap<String, String> APTAB = new HashMap<>();

    ArrayList<String> trackSSN = new ArrayList<>();
    ArrayList<String> parameters = new ArrayList<>();
    String trackParameters = null;


    // ---------------------------------------  PASS - I ----------------------------------
    void pass1(String fileName) throws IOException
    {
        String macroName = null;
        Integer PP = 0, KP = 0, EV = 0, tempEV=0;
        Integer MDTP = 1, KPDTP = 0, SSTP = 1;
        boolean flag = false;

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine().trim()) != null) {

                if(line.isEmpty()){
                    continue;
                }

                String[] words = line.split("\\s+");

                if (words.length == 1 && words[0].equalsIgnoreCase("MACRO")) {
                    flag = true;
                    line = br.readLine();
                    words = line.split("\\s+");
                    macroName = words[0];
                    trackParameters = macroName;

                    if (words.length <= 1) {
                        MNT.add(macroName + "\t" + PP + "\t" + KP + "\t" + EV + "\t" + MDTP + "\t"+ (KP == 0 ?
KPDTP : (KPDTP + 1)) + "\t" + SSTP);
                        continue;
                    }
                    for (int i = 1; i < words.length; i++) {
                        words[i] = words[i].replaceAll("[&,]", "");
                        if (words[i].contains("=")) {
                            String param_value[] = words[i].split("=");
                            KP++;
                            PNTAB.add(param_value[0]);
```

```
                KPDTAB.add(String.join("\t",param_value));
                trackParameters += "\t" + param_value[0];
            } else {
                PP++;
                PNTAB.add(words[i]);
                trackParameters += "\t" + words[i];
            }
        }
        parameters.add(trackParameters.trim());
    }
    else if (words[0].equalsIgnoreCase("LCL") || words[0].equalsIgnoreCase("GBL"))
    {
        flag = true;
        ArrayList<String> EVname = new ArrayList<>();
        for (int i = 1; i < words.length; i++) {
            String cleanedWord = words[i].replaceAll("[&,]", "");
            EVNTAB.add(cleanedWord);
            EV++;
            tempEV++;
            EVname.add("(E, " + tempEV + ")");
        }
        String mdtEntry = words[0] + "\t" + String.join("\t", EVname);
        MDT.add(mdtEntry);
        EVname.clear();
    }
    else if (words.length == 1 && words[0].equalsIgnoreCase("MEND"))
    {
        flag = false;
        MDT.add("MEND" );
        createSSTAB();
        SSTP = updateSSTP();
        MNT.add(macroName + "\t" + PP + "\t" + KP + "\t" + EV + "\t" + MDTP + "\t"+ (KP == 0 ?
KPDTP : (KPDTP + 1)) + "\t" + SSTP);
        MDTP = MDT.size() + 1;
        KPDTP += KP;
        PP = KP = EV = 0;
    }
    else if (flag)
    {
        if (words[0].startsWith(".")) {
            String cleanedWord = words[0].replaceAll("[.]", "");
            if (!SSNTAB.contains(cleanedWord)) {
                SSNTAB.add(cleanedWord);
                trackSSN.add(cleanedWord);
            }
        }
        else if (words[words.length - 1].startsWith(".")) {
            String cleanedWord = words[words.length - 1].replaceAll("[.]", "");
            if (!SSNTAB.contains(cleanedWord)) {
                SSNTAB.add(cleanedWord);
                trackSSN.add(cleanedWord);
            }
        }

        ArrayList<String> MDT_parts = new ArrayList<>();
        for (int i = 0; i < words.length; i++)
        {
```

```
                if (words[i].contains("&") || words[i].startsWith("."))
                {
                    words[i] = words[i].replaceAll("[&,.()]", "");
                    if (PNTAB.contains(words[i])) {
                        MDT_parts.add("(P," + (PNTAB.indexOf(words[i]) + 1) + ")");
                    } else if (EVNTAB.contains(words[i])) {
                        MDT_parts.add("(E," + (EVNTAB.indexOf(words[i]) + 1) + ")");
                    } else if (SSNTAB.contains(words[i])) {
                        MDT_parts.add("(S, " + (SSNTAB.indexOf(words[i]) + 1) + ")");
                    }
                }
                else {
                    MDT_parts.add(words[i].replaceAll("[,()]", ""));
                }
            }
            String mdtEntry = String.join("\t", MDT_parts);
            MDT.add(mdtEntry);
            MDT_parts.clear();
        }
        else if (words.length == 1 && words[0].equalsIgnoreCase("START")){
            pass2(br);
            flag = false;
            break;
        }

    }
  } catch (Exception e) {
      System.out.println(e);
  }
}

void createSSTAB(){
    for(int j=0; j<trackSSN.size(); j++){
        String str = trackSSN.get(j);
        Integer indexinSS = SSNTAB.indexOf(str)+1;
        Integer indexinMDT=0;

        for (int i = 0; i < MDT.size(); i++) {
            if (MDT.get(i).startsWith("(S, "+indexinSS +")")) {
                indexinMDT = i + 1;
                SSTAB.add(indexinMDT);
                break;
            }
        }
    }
}

Integer updateSSTP() {
    Integer temp = 0;
    String str = trackSSN.get(0);
    Integer indexinSS = SSNTAB.indexOf(str)+1;
    Integer indexinMDT=0;

    for (int i = 0; i < MDT.size(); i++) {
        if (MDT.get(i).startsWith("(S, "+indexinSS +")")) {
            indexinMDT = i + 1;
            temp = SSTAB.indexOf(indexinMDT)+1;
```

```
            break;
        }
    }
    trackSSN.clear();
    return temp;
}
// ------------------------------------  PASS - I  END ------------------------------


// ------------------------------------  PASS - II  ---------------------------------
void pass2(BufferedReader br)
{
    String line;
    Integer MDTP = -1, SSTP = -1;
    boolean isMacro = false;
    HashMap<String, String> paramValues = new HashMap<>();
    HashMap<String, Integer> evNames = new HashMap<>();

    HashMap<String, String[]> macroParametersMap = new HashMap<>();
    HashMap<String, Integer[]> macroInfoMap = new HashMap<>();

    for (String entry : MNT) {
        String[] parts = entry.split("\\s+");
        String macroName = parts[0];
        macroInfoMap.put(macroName, new Integer[]{
            Integer.parseInt(parts[4]) - 1,
            Integer.parseInt(parts[6]) - 1
        });
        for (String paramSet : parameters) {
            if (paramSet.startsWith(macroName)) {
                macroParametersMap.put(macroName, paramSet.split("\t"));
            }
        }
    }

    try (FileWriter output = new FileWriter("Output.txt")) {
        output.write("START\n");
        while ((line = br.readLine()) != null)
        {
            if (line.isEmpty()) {
                continue;
            }
            String[] words = line.split("\\s+");

            if (macroInfoMap.containsKey(words[0])) {
                isMacro = true;
                String macroName = words[0];
                MDTP = macroInfoMap.get(macroName)[0];
                SSTP = macroInfoMap.get(macroName)[1];
                paramValues.clear();
                evNames.clear();

                String[] macroParams = macroParametersMap.get(macroName);
                for (int j = 1; j < macroParams.length; j++) {
                    String param = macroParams[j];
                    if (j < words.length && words[j].contains("=")) {
```

```
                    paramValues.put(param, words[j].split("=")[1].replace(",", ""));
                } else if (j < words.length) {
                    paramValues.put(param, words[j].replace(",", ""));
                }
            }
        }

        if (isMacro)
        {
            while (!MDT.get(MDTP).trim().equalsIgnoreCase("MEND"))
            {
                Integer evIndex;
                String[] mdtParts = MDT.get(MDTP).split("\t");

                if (mdtParts[0].equalsIgnoreCase("LCL"))
                {
                    for (int i = 1; i < mdtParts.length; i++) {
                        evIndex = Integer.parseInt(mdtParts[i].replaceAll("[(E,)]", "").trim()) - 1;
                        evNames.put(EVNTAB.get(evIndex), -1);
                    }
                }
                else if (mdtParts[1].equalsIgnoreCase("SET"))
                {
                    handleSetStatement(mdtParts, evNames);
                }
                else if (mdtParts[0].equalsIgnoreCase("AIF"))
                {
                    if (evaluateAIF(mdtParts, evNames)) {
                        MDTP = SSTAB.get(SSTP) - 2;
                    }
                }
                else
                {
                    output.write(expandValues(mdtParts, paramValues, evNames) + "\n");
                }
                MDTP++;
            }
            APTAB.putAll(paramValues);
            isMacro = false;
        }
        else
        {
            if ((words.length == 1) && words[0].equalsIgnoreCase("END")){
                output.write("END");
                output.close();
                return;
            }
            output.write(line + "\n");
        }
    }
}catch (Exception e) {
    System.out.println(e);
}
}
```

```java
private void handleSetStatement(String[] mdtParts, HashMap<String, Integer> evNames)
{
    Integer evIndex = Integer.parseInt(mdtParts[0].replaceAll("[(E,)]", "")) - 1;
    if (evNames.containsKey(EVNTAB.get(evIndex))) {
        try {
            evNames.put(EVNTAB.get(evIndex), Integer.parseInt(mdtParts[2]));
        } catch (Exception e) {
            Integer oldVal = evNames.getOrDefault(EVNTAB.get(evIndex), 0);
            Integer newVal;
            if (mdtParts[3].trim().equalsIgnoreCase("+")) {
                newVal = oldVal + Integer.parseInt(mdtParts[4].trim());
                evNames.put(EVNTAB.get(evIndex), newVal);
            } else if (mdtParts[3].trim().equalsIgnoreCase("-")) {
                newVal = oldVal - Integer.parseInt(mdtParts[4].trim());
                evNames.put(EVNTAB.get(evIndex), newVal);
            }
        }
    }
}

private boolean evaluateAIF(String[] mdtParts, HashMap<String, Integer> evNames)
{
    Integer evIndex = Integer.parseInt(mdtParts[1].replaceAll("[(E,)]", "").trim()) - 1;
    Integer op1 = evNames.get(EVNTAB.get(evIndex));
    Integer op2 = Integer.parseInt(mdtParts[3].replaceAll("[(),]", "").trim());
    String operator = mdtParts[2].trim();

    switch (operator)
    {
        case "NE":
            return op1 != op2;
        case "EQ":
            return op1.equals(op2);
        case "GT":
            return op1 > op2;
        case "LT":
            return op1 < op2;
        default:
            return false;
    }
}



private String expandValues(String[] mdtParts, HashMap<String, String> paramValues,
HashMap<String, Integer> evNames){

    StringBuilder expandedLine = new StringBuilder();
    for(String part : mdtParts)
    {
        if(part.startsWith("(S,")){
            continue;
        }
        else if(part.startsWith("(P,")){
            int paramIndex = Integer.parseInt(part.replaceAll("[(),]", "").split("P")[1]) - 1;
            expandedLine.append(paramValues.get(PNTAB.get(paramIndex))).append(" ");
```

```
        }
        else if (part.startsWith("(E,")) {
            int evIndex = Integer.parseInt(part.replaceAll("[(),]", "").split("E")[1]) - 1;
            expandedLine.append(evNames.get(EVNTAB.get(evIndex))).append(" ");
        }
        else {
            expandedLine.append(part).append(" ");
        }

    }

    return expandedLine.toString().trim();
}

// ------------------------------------  PASS - II  END ------------------------------
```