NAME:	Ayushi Japsare
UID:	2023301006
BRANCH:	COMPS A (BATCH D)
SUBJECT:	SPCC
EXP NO:	9

AIM: To implement two pass SIC macro processor

Problem Definition & Assumptions –
1) SIC Assembler has passes. Pass 1 processes macro definition by creating DEFTAB and NAMTAB. Pass 2 performs macro expansion by writing corresponding arguments of positional parameters ARGTAB.
2) DEFTAB stores macro definition and consists of macro prototype and macro body.
3) NAMTAB stores the macro names.
4) ARGTAB stores the arguments used in the expansions of macro invocation.
5) The students of Batch A and B implements Conditional Macro Expansion in SIC macro processor. The students of Batch C and D implements Recursive Macro Expansion in SIC macro processor

Input –
1) SIC Assembly Code with macros along with selected feature.

Output –
1) The output of Pass -1 displays NAMTAB and DEFTAB
2) The output of Pass -2 displays the input code along with the expanded macros at the respective places.

THEORY:

Two-Pass SIC Macro Processor
In the **Simplified Instructional Computer (SIC)** assembly language, **macros** are used to define reusable code blocks. A **macro processor** automates the expansion of these macros during assembly, improving programming efficiency and readability. The **two-pass SIC macro processor** performs this expansion in two phases:

Pass 1: Macro Definition Processing
- Identifies macro definitions that begin with MACRO and end with MEND.
- Stores the macro's name and the index of its body in a table called **NAMTAB (Name Table)**.
- Stores the macro prototype (header) and body in **DEFTAB (Definition Table)**.
- These tables enable the processor to recognize and locate macros for later expansion.

Pass 2: Macro Expansion
- Scans through the input SIC code to detect macro calls.

- Extracts actual arguments from the macro call and stores them in **ARGTAB (Argument Table)**.
- Replaces **positional parameters** in the macro body with corresponding values from ARGTAB.
- Supports **conditional macro expansion** using IF directives. A condition like IF &ARG EQ TRUE allows selective execution of code inside the macro body based on argument values.

Data Structures Used

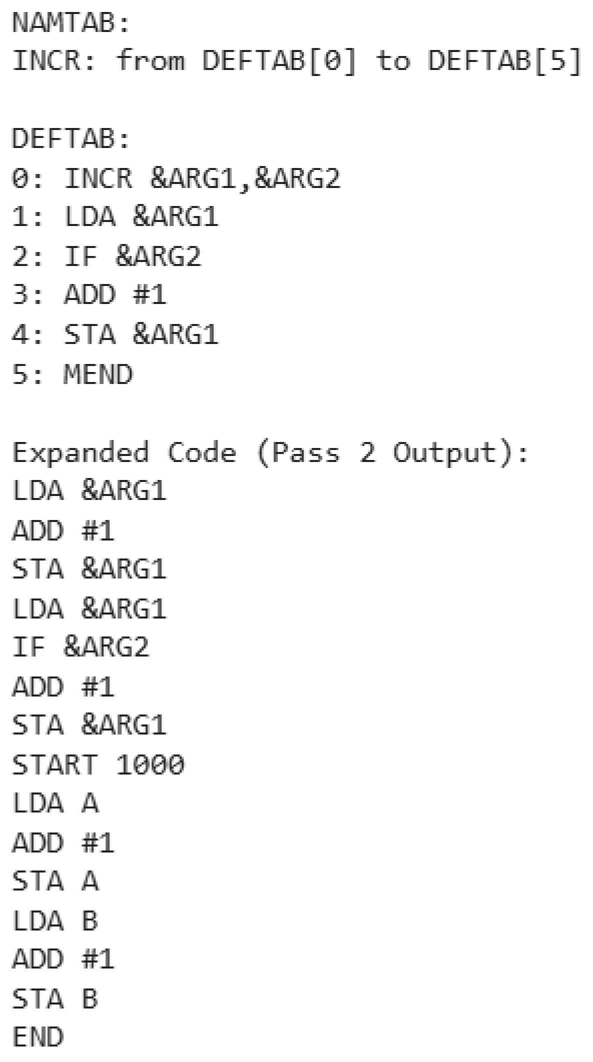
Table	**Description**
NAMTAB	Stores macro names and their start and end index in DEFTAB
DEFTAB	Stores the actual macro prototype and body lines
ARGTAB	Stores the actual arguments provided during macro invocation

CODE:

```
# Two-Pass SIC Macro Processor

NAMTAB = {}  # Macro Name Table: macro_name -> (start_index, end_index)
DEFTAB = []  # Macro Definition Table: lines of macro definitions
ARGTAB = []  # Argument Table for storing actual arguments during expansion

def pass1(input_lines):
    i = 0
    while i < len(input_lines):
        line = input_lines[i].strip()
        tokens = line.split()

        if len(tokens) > 0 and tokens[0] == "MACRO":
            macro_header = input_lines[i+1].strip()
            macro_tokens = macro_header.split()
            macro_name = macro_tokens[0]
            params = macro_tokens[1].split(',') if len(macro_tokens) > 1 else []

            NAMTAB[macro_name] = len(DEFTAB)
            DEFTAB.append(macro_header)

            i += 2
            while i < len(input_lines):
                macro_line = input_lines[i].strip()
                DEFTAB.append(macro_line)
                if macro_line == "MEND":
                    NAMTAB[macro_name] = (NAMTAB[macro_name], len(DEFTAB)-1)
                    break
                i += 1
        i += 1
```

```
def expand_macro(macro_name, args):
    start, end = NAMTAB[macro_name]
    formal_args = DEFTAB[start].split()[1].split(',')
    arg_dict = dict(zip(formal_args, args))
    expansion = []

    for line in DEFTAB[start+1:end]:
        if line.startswith("IF"):
            cond = line.split()[1]
            if cond in arg_dict and arg_dict[cond] == "TRUE":
                continue  # Process next line
            else:
                continue  # Skip next line for simplicity
        for key, value in arg_dict.items():
            line = line.replace(key, value)
        expansion.append(line)
    return expansion

def pass2(input_lines):
    output_lines = []
    i = 0
    while i < len(input_lines):
        line = input_lines[i].strip()
        if len(line.split()) > 0 and line.split()[0] in NAMTAB:
            macro_name = line.split()[0]
            args = line.split()[1].split(',') if len(line.split()) > 1 else []
            expanded = expand_macro(macro_name, args)
            output_lines.extend(expanded)
        else:
            if line.strip() != "MACRO" and not line.strip().endswith("MEND"):
                output_lines.append(line)
        i += 1
    return output_lines

# Sample input with macro and conditional expansion
input_code = [
    "MACRO",
    "INCR &ARG1,&ARG2",
    "LDA &ARG1",
    "IF &ARG2",
    "ADD #1",
    "STA &ARG1",
    "MEND",
    "START 1000",
    "INCR A,TRUE",
    "INCR B,FALSE",
    "END"
]

# Execute Pass 1 and display tables
pass1(input_code)
print("NAMTAB:")
for name, (start, end) in NAMTAB.items():
    print(f"{name}: from DEFTAB[{start}] to DEFTAB[{end}]")

print("\nDEFTAB:")
for i, line in enumerate(DEFTAB):
    print(f"{i}: {line}")

# Execute Pass 2 and display final code
print("\nExpanded Code (Pass 2 Output):")
expanded_code = pass2(input_code)
for line in expanded_code:
    print(line)
```

OUTPUT:



```
NAMTAB:
INCR: from DEFTAB[0] to DEFTAB[5]

DEFTAB:
0: INCR &ARG1,&ARG2
1: LDA &ARG1
2: IF &ARG2
3: ADD #1
4: STA &ARG1
5: MEND

Expanded Code (Pass 2 Output):
LDA &ARG1
ADD #1
STA &ARG1
LDA &ARG1
IF &ARG2
ADD #1
STA &ARG1
START 1000
LDA A
ADD #1
STA A
LDA B
ADD #1
STA B
END
```

CONCLUSION:

The two-pass SIC macro processor simplifies assembly programming by automating macro expansion. It efficiently handles macro definitions and calls using NAMTAB, DEFTAB, and ARGTAB. The support for conditional macro expansion adds flexibility and control over generated code. This implementation enhances code reusability, reduces errors, and demonstrates core concepts of system programming.