GHARDA FOUNDATION'S

GHARDA INSTITUTE OF TECHNOLOGY, LAVEL

COMPUTER ENGINEERING DEPARTMENT

A/P: Lavel, Tal.Khed Dist. Ratnagiri

Evaluation Sheet

Class: TE-Computer

Semester: VI

Subject: **System Programming & Compiler Construction**

Experiment No: 2

Title of Experiment: Implementation of Two passes Macro Processor.

Sr. No.	Evaluation Criteria	Max Marks	Marks Obtained
1	Practical Performance	10	
2	Oral	5	
	Total	15	

Signature of Subject Teacher

Theory:

Two Pass Macroprocessor

A two-pass macroprocessor is a type of macroprocessor that performs two passes over the input source code to expand macros.

In the first pass, the macro definitions are expanded and stored in a table. This pass is called the definition pass. During this pass, the macroprocessor reads the input source code and identifies any macro definitions. It then stores these definitions in a table for later use in the second pass.

In the second pass, the macro invocations are expanded using the macro definitions stored in the table. This pass is called the expansion pass. During this pass, the macroprocessor reads the input source code again and identifies any macro invocations. It then replaces each macro invocation with the corresponding macro definition from the table.

The advantage of using a two-pass macroprocessor is that it allows for more complex macro definitions and invocations than a single-pass macroprocessor. Since the macro definitions are expanded and stored in a table before any invocations are expanded, the macroprocessor can handle recursive macro invocations and macro definitions that depend on previous macro invocations.

However, a two-pass macroprocessor can be slower and use more memory than a single-pass macroprocessor, since it needs to store the macro definitions in a table and perform two passes over the input source code.

In addition to being able to handle more complex macro definitions and invocations, a two-pass macroprocessor can also perform more advanced error checking during the definition pass. For example, it can detect and report errors such as undefined macro names, duplicate macro definitions, and circular macro references.

The two-pass macroprocessor also allows for more efficient handling of macro definitions that are used multiple times in the source code. Since the macro definitions are expanded and stored in a table during the definition pass, they can be quickly retrieved and used during the expansion pass, without the need to re-parse and expand the macro definition each time it is invoked.

One potential drawback of a two-pass macroprocessor is that it requires more memory to store the macro definition table. This can be a concern for large source code files or systems with limited memory resources. Additionally, the two-pass approach can be slower than a single-pass approach for small source code files, since it requires two passes over the input instead of one.

Key points of a Two Pass Macroprocessor

1. A two-pass macroprocessor performs two passes over the input source code to expand macros.
2. The first pass is called the definition pass, where the macro definitions are expanded and stored in a table for later use.
3. The second pass is called the expansion pass, where the macro invocations are expanded using the macro definitions stored in the table.
4. A two-pass macroprocessor can handle more complex macro definitions and invocations than a single-pass macroprocessor.
5. It allows for more efficient handling of macro definitions that are used multiple times in the source code.
6. It can perform more advanced error checking during the definition pass.
7. It requires more memory to store the macro definition table than a single-pass macroprocessor.
8. The two-pass approach can be slower than a single-pass approach for small source code files.
9. The choice between a single-pass or two-pass macroprocessor depends on the specific requirements of the system or application being developed.

Program: (main.py)

```
fp = open('program3.txt', 'r')
program = fp.read().split('\n')

print('\nGiven Assembly Code\n')
for line in program:
    print(line)

fp.close()

fp = open('program3.txt', 'r')
program = fp.read().split('MEND\n')
fp.close()

mnt = [] #empty list
mdt = {} # empty dict

for line in program:
    line.strip() # remove begining and trailing white spaces
    a = line.split('\n')
    if a[0] == 'MACRO':
        #print ('Macro name is - ', a[1])
        mnt.append(a[1])
        #print('Macro Instructions are - ', a[2:])
        mdt[a[1]] = a[2:len(a)-1]
    else:
        prog = a

print('\nContent of MNT\n-')
for each_mn in mnt:
    print(each_mn)

print('\nContent of MDT-\n')
for k, v in mdt.items():
    for command in v:
        print(command)

print('\nAfter Macro Expansion\n')



for line in prog:
    identify_mc = line.split()
    for word in identify_mc:
        if word  in  mnt:
            value = mdt[word]
            for i in value:
                print(i)
        else:
            print(word, '', end = '')
```

(program3.txt):-

```
MACRO

INCR

mover areg one

mover breg two

add areg breg

MEND


MACRO

DNL

mover creg one

mover dreg two

div creg dreg

MEND


MACRO

INCR1

mover areg one

mover breg two

mult areg breg

MEND



START 200

MOVER AREG FIRST

ADD AREG SECOND

MOVEM AREG RESULT

INCR

PRINT RESULT
```

RESULT DS 1

FIRST DC 5

INCR1

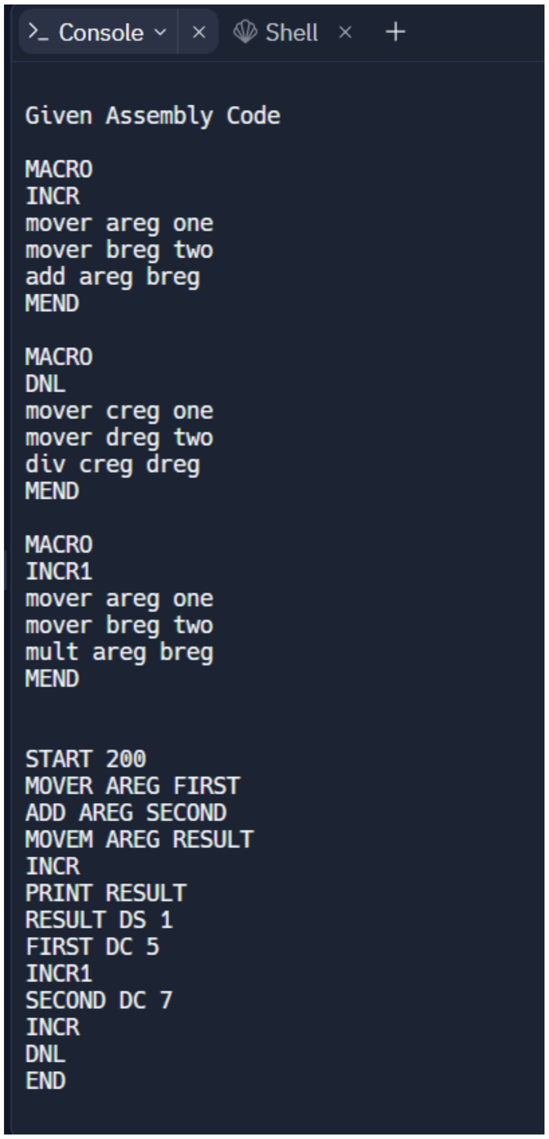
SECOND DC 7

INCR

DNL

END

Output:-



Console Shell

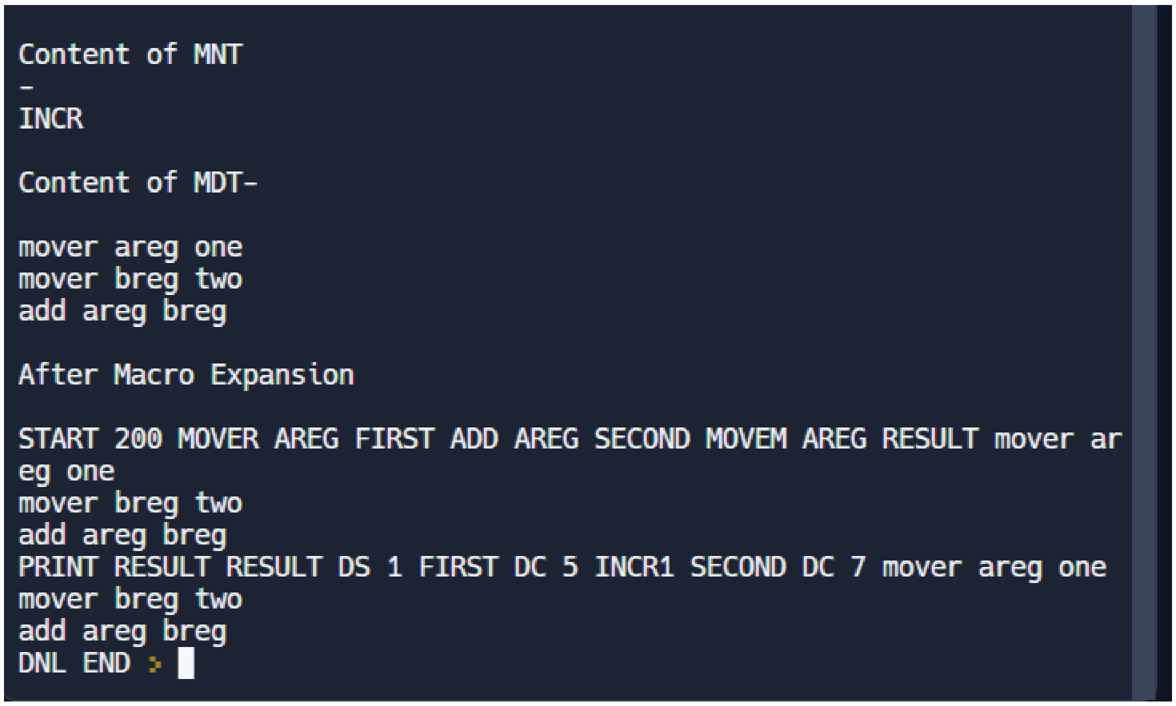
```
Given Assembly Code

MACRO
INCR
mover areg one
mover breg two
add areg breg
MEND

MACRO
DNL
mover creg one
mover dreg two
div creg dreg
MEND

MACRO
INCR1
mover areg one
mover breg two
mult areg breg
MEND


START 200
MOVER AREG FIRST
ADD AREG SECOND
MOVEM AREG RESULT
INCR
PRINT RESULT
RESULT DS 1
FIRST DC 5
INCR1
SECOND DC 7
INCR
DNL
END
```



```
Content of MNT
-
INCR

Content of MDT-

mover areg one
mover breg two
add areg breg

After Macro Expansion

START 200 MOVER AREG FIRST ADD AREG SECOND MOVEM AREG RESULT mover ar
eg one
mover breg two
add areg breg
PRINT RESULT RESULT DS 1 FIRST DC 5 INCR1 SECOND DC 7 mover areg one
mover breg two
add areg breg
DNL END >
```