

PART B

EXPERIMENT NUMBER 9

Aim: To Design and Implement a single pass Macro Processor.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	Name: AMEY THAKUR
Class: Comps TE B	Batch: B3
Date of Experiment: 07/05/2021	Date of Submission: 07/05/2021
Grade:	

B.1 Software Code written by a student:

(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-9.C**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
FILE *f1,*f2,*f3,*f4,*f5;
int i,len;
char mne[20],opnd[20],la[20],name[20],mne1[20],opnd1[20],arg[20];

f1=fopen("minp2.txt","r");
f2=fopen("ntab2.txt","r");
f3=fopen("dtab2.txt","r");
f4=fopen("atab2.txt","w+");
f5=fopen("op2.txt","w");
fscanf(f1,"%s%s%s",la,mne,opnd);
while(strcmp(mne,"END")!=0)
{
if(strcmp(mne,"MACRO")==0)
{
fscanf(f1,"%s%s%s",la,mne,opnd);
```

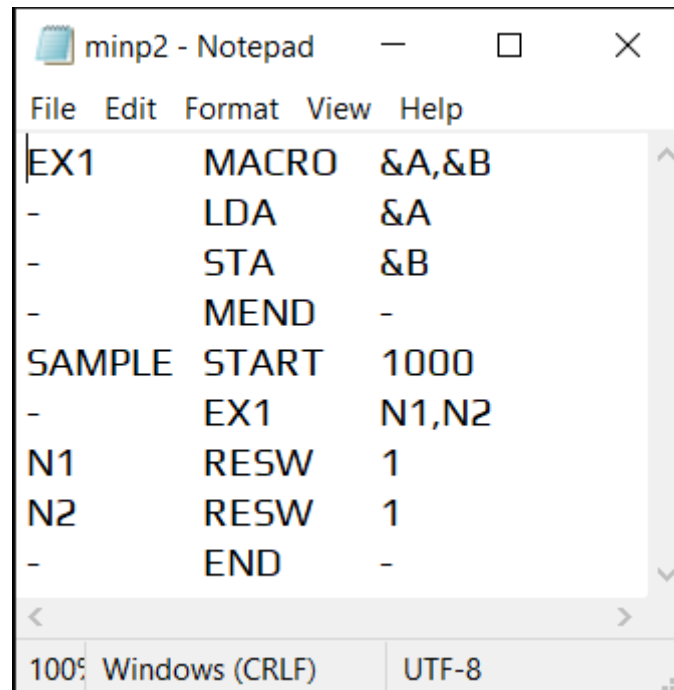
```

while(strcmp(mne,"MEND")!=0)
fscanf(f1,"%s%s%s",la,mne,opnd);
}
else
{
fscanf(f2,"%s",name);
if(strcmp(mne,name)==0)
{
len=strlen(opnd);
for(i=0;i<len;i++)
{
if(opnd[i]!='.')
fprintf(f4,"%c",opnd[i]);
else
fprintf(f4,"\n");
}
fseek(f2,SEEK_SET,0);
fseek(f4,SEEK_SET,0);
fscanf(f3,"%s%s",mne1,opnd1);
fprintf(f5,".\t%s\t%s\n",mne1,opnd);
fscanf(f3,"%s%s",mne1,opnd1);
while(strcmp(mne1,"MEND")!=0)
{
if((opnd1[0]=='&'))
{
fscanf(f4,"%s",arg);
fprintf(f5,"-\t%s\t%s\n",mne1,arg);
}
else
fprintf(f5,"-\t%s\t%s\n",mne1,opnd1);
fscanf(f3,"%s%s",mne1,opnd1);
}
}
else
fprintf(f5,"%s\t%s\t%s\n",la,mne,opnd);
}
fscanf(f1,"%s%s%s",la,mne,opnd);
}
fprintf(f5,"%s\t%s\t%s\n",la,mne,opnd);
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);

```

```
fclose(f5);  
printf("pass2");  
getch();  
}
```

- minp2.txt

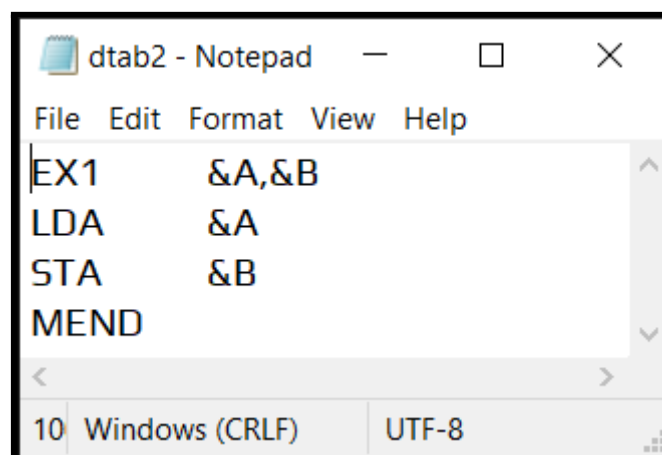


A screenshot of a Notepad window titled "minp2 - Notepad". The window contains assembly code with the following text:

```
EX1      MACRO  &A,&B  
-        LDA    &A  
-        STA    &B  
-        MEND   -  
SAMPLE  START  1000  
-        EX1    N1,N2  
N1       RESW   1  
N2       RESW   1  
-        END    -
```

The status bar at the bottom shows "100%", "Windows (CRLF)", and "UTF-8".

- dtab2.txt

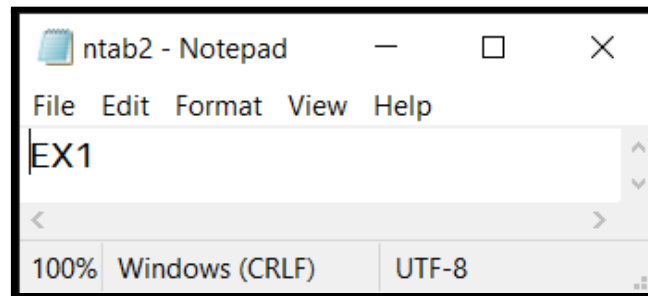


A screenshot of a Notepad window titled "dtab2 - Notepad". The window contains assembly code with the following text:

```
EX1      &A,&B  
LDA      &A  
STA      &B  
MEND
```

The status bar at the bottom shows "10", "Windows (CRLF)", and "UTF-8".

- ntab2.txt



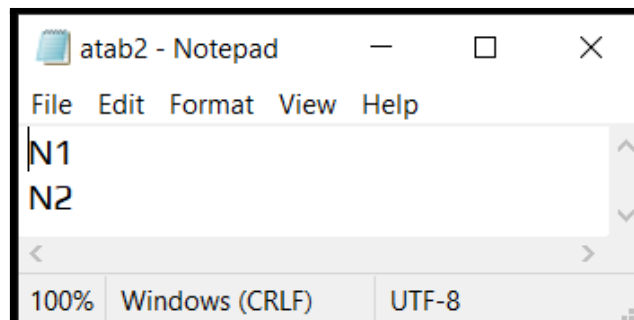
```
ntab2 - Notepad
File Edit Format View Help
EX1
100% Windows (CRLF) UTF-8
```

B.2 Input and Output:



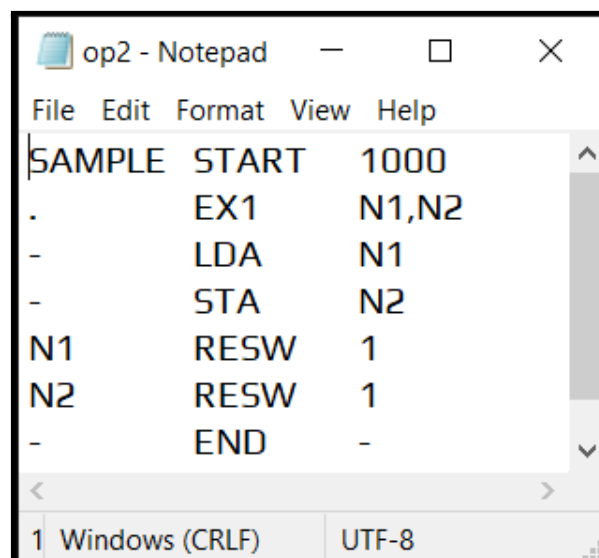
```
Command Prompt - a.exe
C:\Users\ameyt\Desktop>GCC SPCC-9.C
C:\Users\ameyt\Desktop>a.exe
pass2
```

- atab2.txt



```
atab2 - Notepad
File Edit Format View Help
N1
N2
100% Windows (CRLF) UTF-8
```

- op2.txt



```
op2 - Notepad
File Edit Format View Help
SAMPLE START 1000
.      EX1    N1,N2
-      LDA    N1
-      STA    N2
N1     RESW   1
N2     RESW   1
-      END    -
1 Windows (CRLF) UTF-8
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

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. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

We have learnt to design and implement a single pass Macro Processor.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. List the features of Macro

Ans:

- Macro represents a group of commonly used statements in the source programming language.
- Macro Processor replaces each macro-instruction with the corresponding group of source language statements. This is known as the expansion of macros.
- Using Macro-instructions programmers can leave the mechanical details to be handled by the macro processor.
- Macro Processor designs are not directly related to the computer architecture on which it runs.
- Macro Processor involves definition, invocation and expansion.

2. Differentiate Between Macro and Procedure/Function.

Ans:

Macro	Procedure
Macros are useful over Procedures when the number of instructions in the set is less. Therefore, when the subprogram contains less than 10 instructions, Macros are more efficient to use in such cases.	It is better to use Procedures for a set of a large number of instructions. Hence, it is optimal to use Procedures when the number of instructions is more than 10.

The assembler directive- MACRO is used to define a Macro, And to indicate that the body of the procedure has ended, the assembler directive- ENDM is used.	The assembler directive - PROC is used to define a Procedure. And the assembler directive - ENDP is used to indicate that the body of the procedure has ended.
Every time a Macro is called, the assembler of the microprocessor places the entire set of instructions of the Macros in the mainline program form where the call to the macro is made.	Every time a procedure is called, the CALL and RET instructions are required for shifting the control of instruction execution.
The execution of macros is faster as compared to procedures because there is no need to integrate or link the macros with the calling program. It is simply loaded into the main memory every time it is called.	The Procedures execute slower than the Macros because every time a procedure is called, it is necessary to integrate and link it with the calling program and this takes time.
Overhead time is avoided as calling and returning does not take place.	Overhead time occurs while calling the procedure and returning the control to the calling program.
The Macros require a large amount of memory because it is loaded into the main memory every time it is called.	The Procedures require less amount of memory than the Macros because a Procedure is written and loaded into the main memory only once, and is linked to the calling program when called.