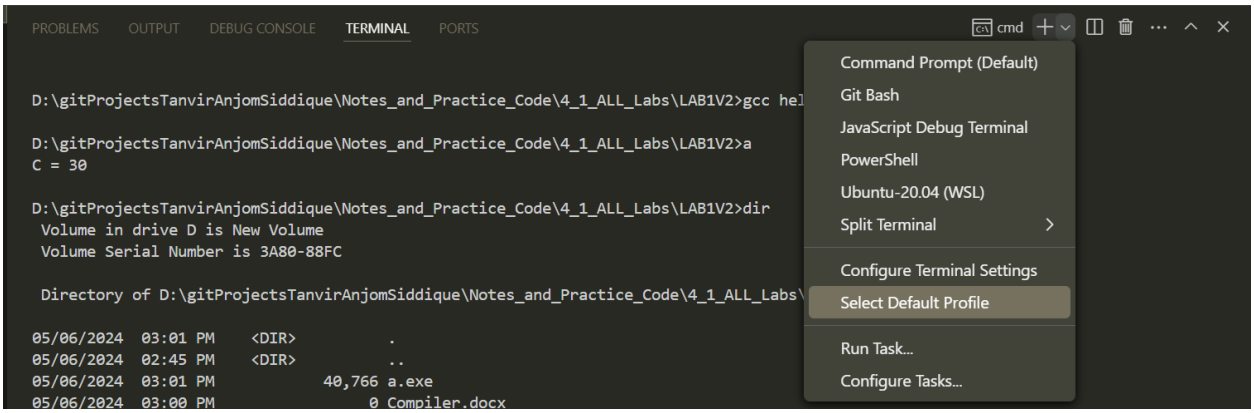
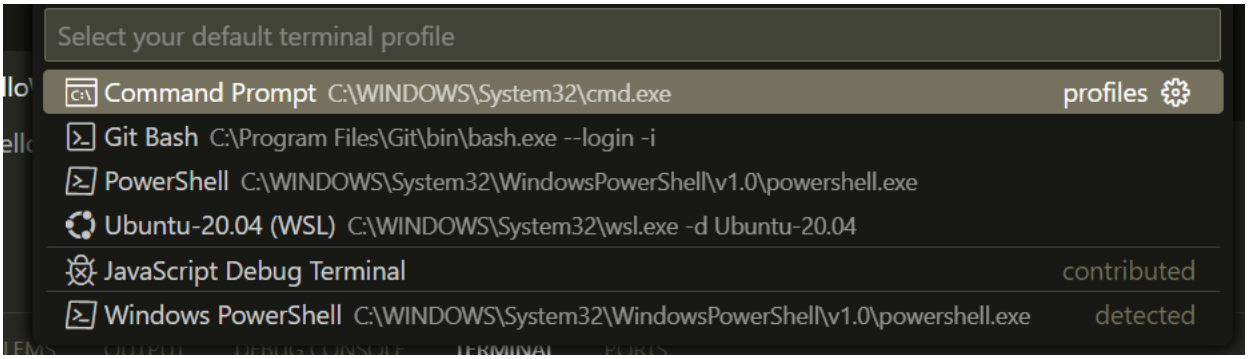
Table of Contents

Tanvir Anjom Siddique

Lab 1:

Setup Terminal:





Compile Code:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                cmd

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc helloWorld.c

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>a
C = 30

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2

05/06/2024  03:01 PM    <DIR>          .
05/06/2024  02:45 PM    <DIR>          ..
05/06/2024  03:01 PM            40,766 a.exe
05/06/2024  03:00 PM                 0 Compiler.docx
05/06/2024  03:00 PM               137 helloWorld.c
               3 File(s)         40,903 bytes
               2 Dir(s)  19,710,251,008 bytes free

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc helloWorld.c -o helloWorld.exe

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>helloWorld.exe
C = 30
```

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc -o helloWorld helloWorld.c

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>helloWorld.exe
C = 30
```

Preprocess:

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc -E helloWorld.c > helloWorld.i

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2

05/06/2024  03:06 PM    <DIR>          .
05/06/2024  02:45 PM    <DIR>          ..
05/06/2024  03:01 PM            40,766 a.exe
05/06/2024  03:03 PM           719,799 Compiler.docx
05/06/2024  03:00 PM               137 helloWorld.c
05/06/2024  03:05 PM            40,766 helloWorld.exe
05/06/2024  03:06 PM            21,364 helloWorld.i
               5 File(s)        822,832 bytes
               2 Dir(s)  19,709,460,480 bytes free

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

.i file → Code of header files are pasted before your main .c code

Create Assembly file (.i to .s):

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc -S helloWorld.i

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2

05/06/2024  03:09 PM    <DIR>          .
05/06/2024  02:45 PM    <DIR>          ..
05/06/2024  03:01 PM            40,766 a.exe
05/06/2024  03:08 PM           737,221 Compiler.docx
05/06/2024  03:00 PM               137 helloWorld.c
05/06/2024  03:05 PM            40,766 helloWorld.exe
05/06/2024  03:06 PM            21,364 helloWorld.i
05/06/2024  03:09 PM               743 helloWorld.s
               6 File(s)        840,997 bytes
               2 Dir(s)  19,709,440,000 bytes free

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

Assembly language for masm Intel:

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>gcc -S -masm=intel helloWorld.i
```

Create .s to .o:

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>as -o helloWorld.o helloWorld.s

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2

05/06/2024  03:23 PM    <DIR>          .
05/06/2024  02:45 PM    <DIR>          ..
05/06/2024  03:01 PM            40,766 a.exe
05/06/2024  03:22 PM           827,443 Compiler.docx
05/06/2024  03:00 PM               137 helloWorld.c
05/06/2024  03:05 PM            40,766 helloWorld.exe
05/06/2024  03:06 PM            21,364 helloWorld.i
05/06/2024  03:23 PM               842 helloWorld.o
05/06/2024  03:15 PM               826 helloWorld.s
               7 File(s)        932,144 bytes
               2 Dir(s)  19,709,341,696 bytes free

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

Create .dump file (Machine code that you can read):

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>objdump -M intel -d helloWorld.o > helloWorld.dump

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2

05/06/2024  03:25 PM    <DIR>          .
05/06/2024  02:45 PM    <DIR>          ..
05/06/2024  03:01 PM            40,766 a.exe
05/06/2024  03:23 PM           912,128 Compiler.docx
05/06/2024  03:00 PM               137 helloWorld.c
05/06/2024  03:25 PM             1,079 helloWorld.dump
05/06/2024  03:05 PM            40,766 helloWorld.exe
05/06/2024  03:06 PM            21,364 helloWorld.i
05/06/2024  03:23 PM               842 helloWorld.o
05/06/2024  03:15 PM               826 helloWorld.s
               8 File(s)      1,017,908 bytes
               2 Dir(s)  19,709,255,680 bytes free

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

```
helloWorld.o:     file format pe-i386


Disassembly of section .text:

00000000 <_main>:
   0:   55                      push   ebp
   1:   89 e5                   mov    ebp,esp
   3:   83 e4 f0                and    esp,0xfffffff0
   6:   83 ec 20                sub    esp,0x20
   9:   e8 00 00 00 00          call   e <_main+0xe>
   e:   c7 44 24 1c 0a 00 00    mov    DWORD PTR [esp+0x1c],0xa
  15:   00
  16:   c7 44 24 18 14 00 00    mov    DWORD PTR [esp+0x18],0x14
  1d:   00
  1e:   8b 54 24 1c             mov    edx,DWORD PTR [esp+0x1c]
  22:   8b 44 24 18             mov    eax,DWORD PTR [esp+0x18]
  26:   01 d0                   add    eax,edx
  28:   89 44 24 14             mov    DWORD PTR [esp+0x14],eax
  2c:   8b 44 24 14             mov    eax,DWORD PTR [esp+0x14]
  30:   89 44 24 04             mov    DWORD PTR [esp+0x4],eax
  34:   c7 04 24 00 00 00 00    mov    DWORD PTR [esp],0x0
  3b:   e8 00 00 00 00          call   40 <_main+0x40>
  40:   b8 00 00 00 00          mov    eax,0x0
  45:   c9                      leave
  46:   c3                      ret
  47:   90                      nop
```

Here size of each line different → CISC

Add Linker/ Loader:

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>objdump -M intel -d a.exe > helloWorld2.dump
```

Size of Hellowrold2.dump is greater than helloworld.dump as library functions are linked here.

*** All Steps Of Compiling C Program ***

Makefile

```
main:
    gcc -E helloWorld.c > helloWorld.i
    gcc -S helloWorld.i
    gcc -S -masm=intel helloWorld.i
    as -o helloWorld.o helloWorld.s
    objdump -M intel -d helloWorld.o > helloWorld.dump
    gcc -o helloWorld.exe helloWorld.o
    helloWorld.exe
    objdump -M intel -d helloWorld.exe > helloWorld.txt
```

Explanation

1. **Target**:
    - `main:` is the target in this Makefile. When you run `make main`, all the commands under this target will be executed sequentially.
2. **Preprocessing**:
    - `gcc -E helloWorld.c > helloWorld.i`
        - The `gcc -E` command preprocesses the `helloWorld.c` file. The preprocessing step includes handling directives like `#include` and `#define`. The output is saved in `helloWorld.i`.
3. **Compilation to Assembly (AT&T syntax)**:
    - `gcc -S helloWorld.i`
        - The `gcc -S` command compiles the preprocessed file `helloWorld.i` into assembly language code in AT&T syntax. The output is `helloWorld.s`.

4. **Compilation to Assembly (Intel syntax)**:
    - `gcc -S -masm=intel helloWorld.i`
        - This command is similar to the previous one, but it generates assembly code in Intel syntax instead of AT&T syntax. It will overwrite the `helloWorld.s` file with the Intel syntax version.
5. **Assembling**:
    - `as -o helloWorld.o helloWorld.s`
        - The `as` command assembles the `helloWorld.s` file (which is in assembly language) into an object file `helloWorld.o`.
6. **Disassembling the Object File**:
    - `objdump -M intel -d helloWorld.o > helloWorld.dump`
        - The `objdump` command disassembles the `helloWorld.o` object file. The `-M intel` option specifies that the disassembly output should be in Intel syntax. The output is saved to `helloWorld.dump`.
7. **Linking**:
    - `gcc -o helloWorld.exe helloWorld.o`
        - This command links the object file `helloWorld.o` to create the executable `helloWorld.exe`.
8. **Running the Executable**:
    - `helloWorld.exe`
        - This runs the compiled executable `helloWorld.exe`.
9. **Disassembling the Executable**:
    - `objdump -M intel -d helloWorld.exe > helloWorld.txt`
        - The `objdump` command disassembles the `helloWorld.exe` executable file. The `-M intel` option specifies that the disassembly should be in Intel syntax. The output is saved to `helloWorld.txt`.

Summary

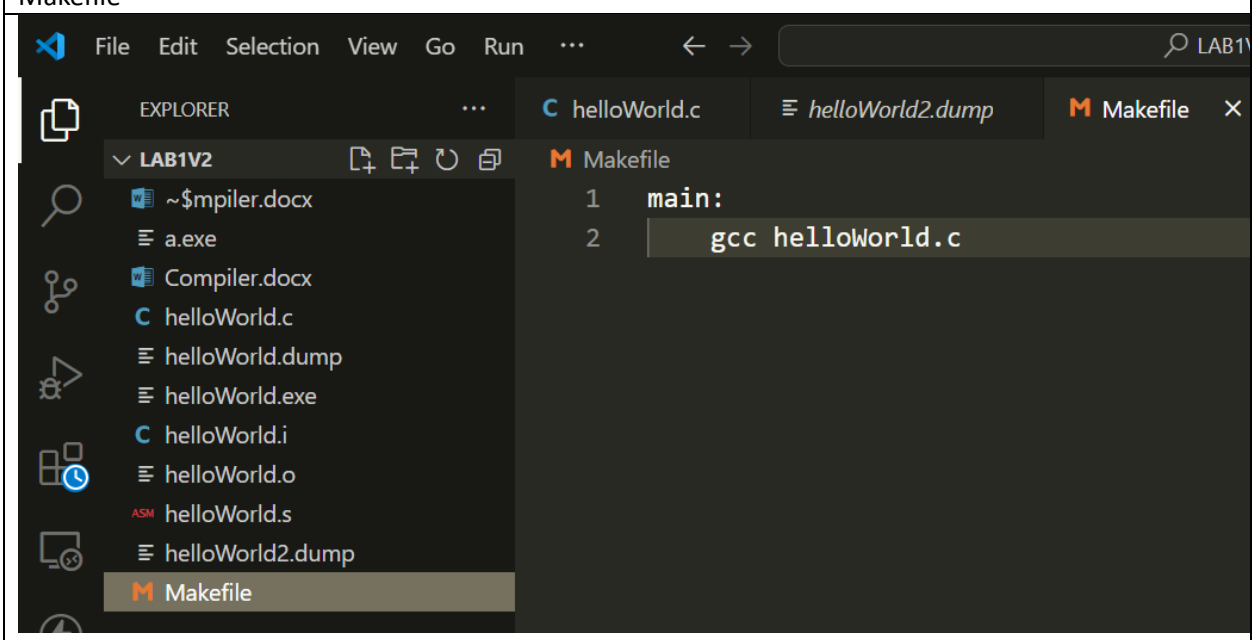
- The Makefile automates the process of compiling a C program (**`helloWorld.c`**) into an executable (**`helloWorld.exe`**).
- It includes steps to preprocess the C code, compile it to assembly, assemble it into an object file, link the object file into an executable, and disassemble both the object file and the executable for inspection.
- The output files at each step allow you to see the intermediate stages: preprocessed code (**`helloWorld.i`**), assembly code (**`helloWorld.s`**), object file (**`helloWorld.o`**), and disassembled output (**`helloWorld.dump`** and **`helloWorld.txt`**).

Build System:

Write 1 command compile many files.

Create a file: (this name is fixed)

Makefile



We can execute multiple commands here

Now in terminal:

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>make
gcc helloWorld.c

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2>
```

**Makefile (compile multiple file):

HelloFunc.h

```c
void HelloPrint();//declare function
```

HelloFunc.c

```c
#include<stdio.h>

void HelloPrint()
{
    printf("Hello World\n");
}
```

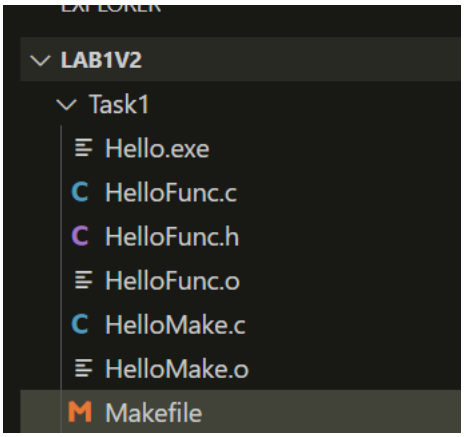
HelloMake.c

```c
#include "HelloFunc.h"

int main()
{
    HelloPrint();
    return 0;
}
```

Makefile

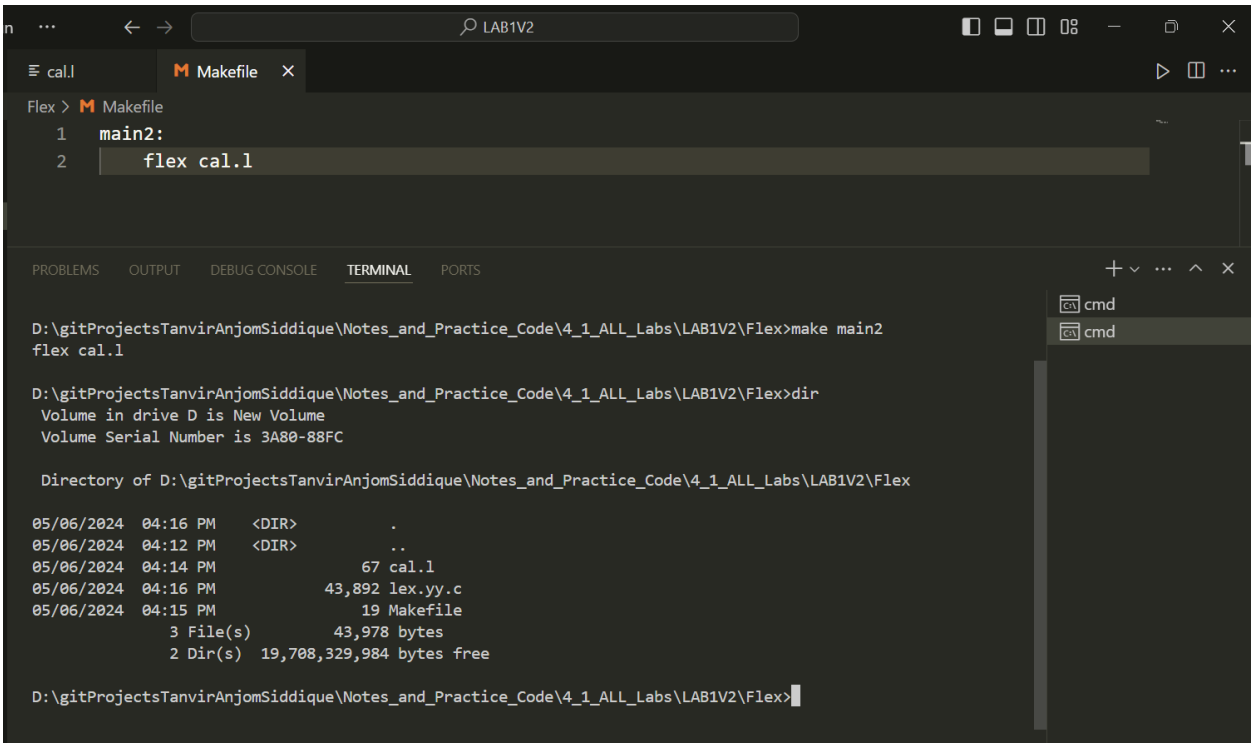
```
main:
    gcc -c -o HelloFunc.o HelloFunc.c
    gcc -c -o HelloMake.o HelloMake.c
    gcc -o Hello HelloFunc.o HelloMake.o
```

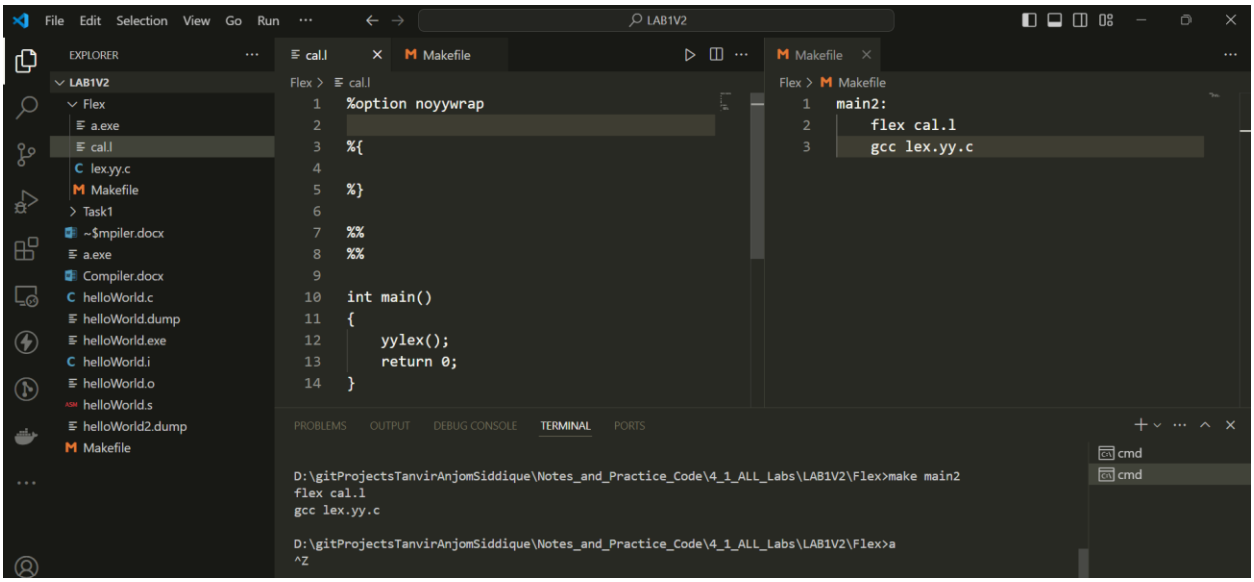
```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1>make
gcc -c -o HelloFunc.o HelloFunc.c
gcc -c -o HelloMake.o HelloMake.c
gcc -o Hello HelloFunc.o HelloMake.o

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1>hello
Hello World

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1>dir
 Volume in drive D is New Volume
 Volume Serial Number is 3A80-88FC

 Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1

05/06/2024  03:55 PM    <DIR>          .
05/06/2024  03:55 PM    <DIR>          ..
05/06/2024  03:55 PM            41,028 Hello.exe
05/06/2024  03:44 PM                74 HelloFunc.c
05/06/2024  03:44 PM                36 HelloFunc.h
05/06/2024  03:55 PM               782 HelloFunc.o
05/06/2024  03:43 PM                76 HelloMake.c
05/06/2024  03:55 PM               712 HelloMake.o
05/06/2024  03:48 PM               117 Makefile
               7 File(s)         42,825 bytes
```



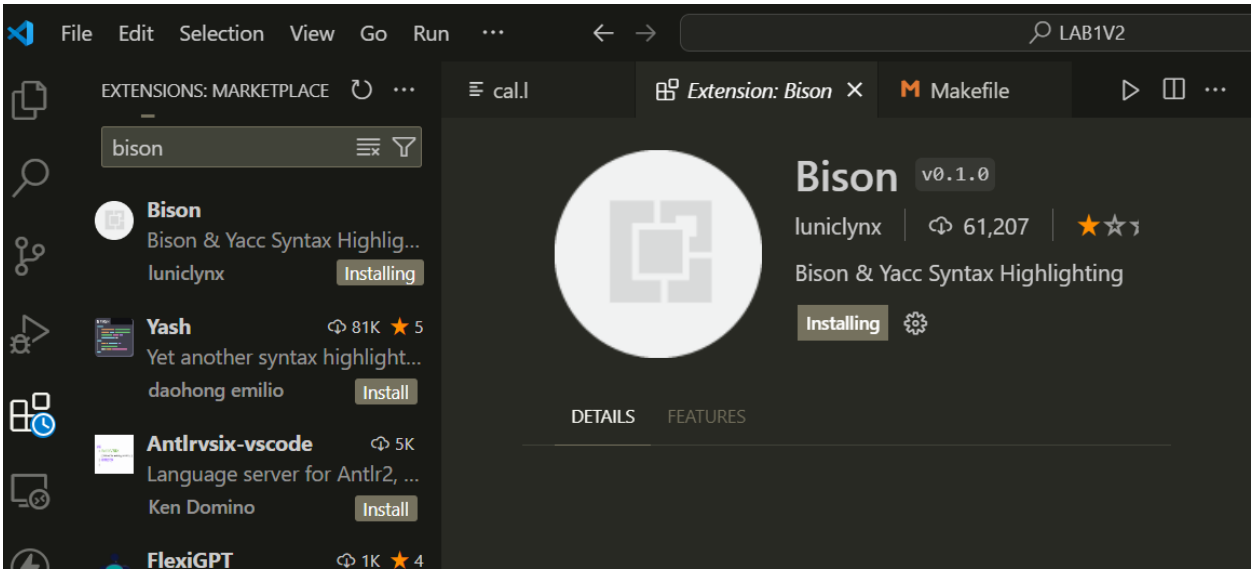
Files in different folders

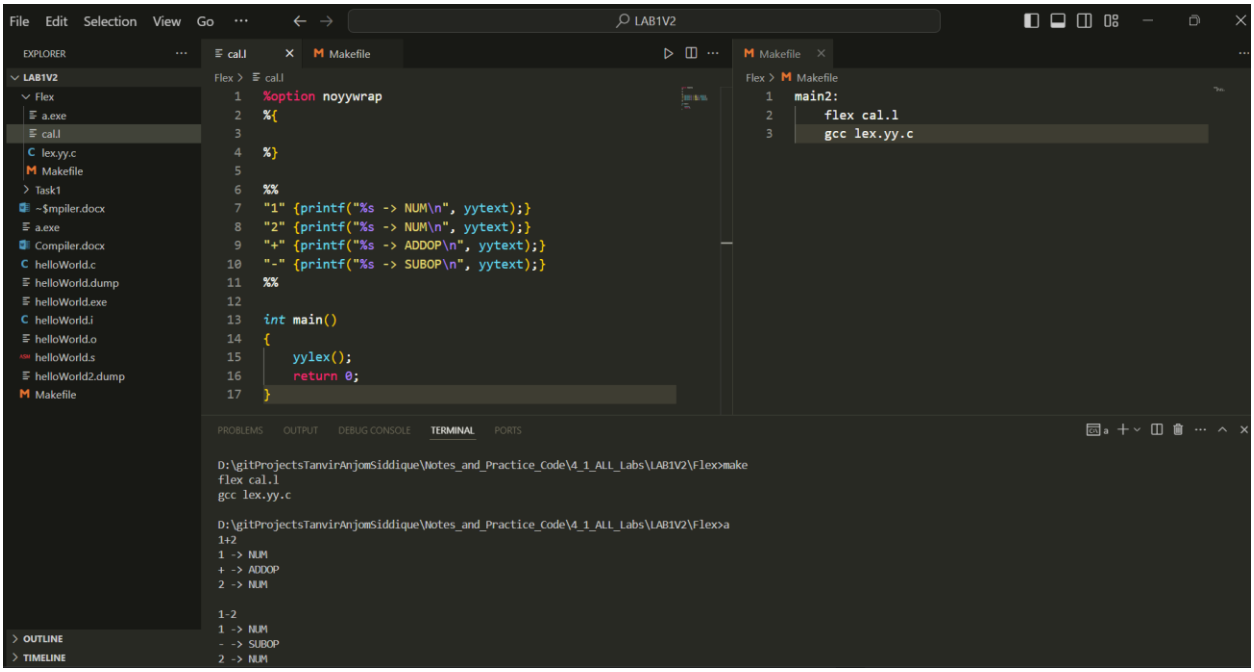
Flex (lexical analysis): [Identify ID , VAR, STRING, OP]

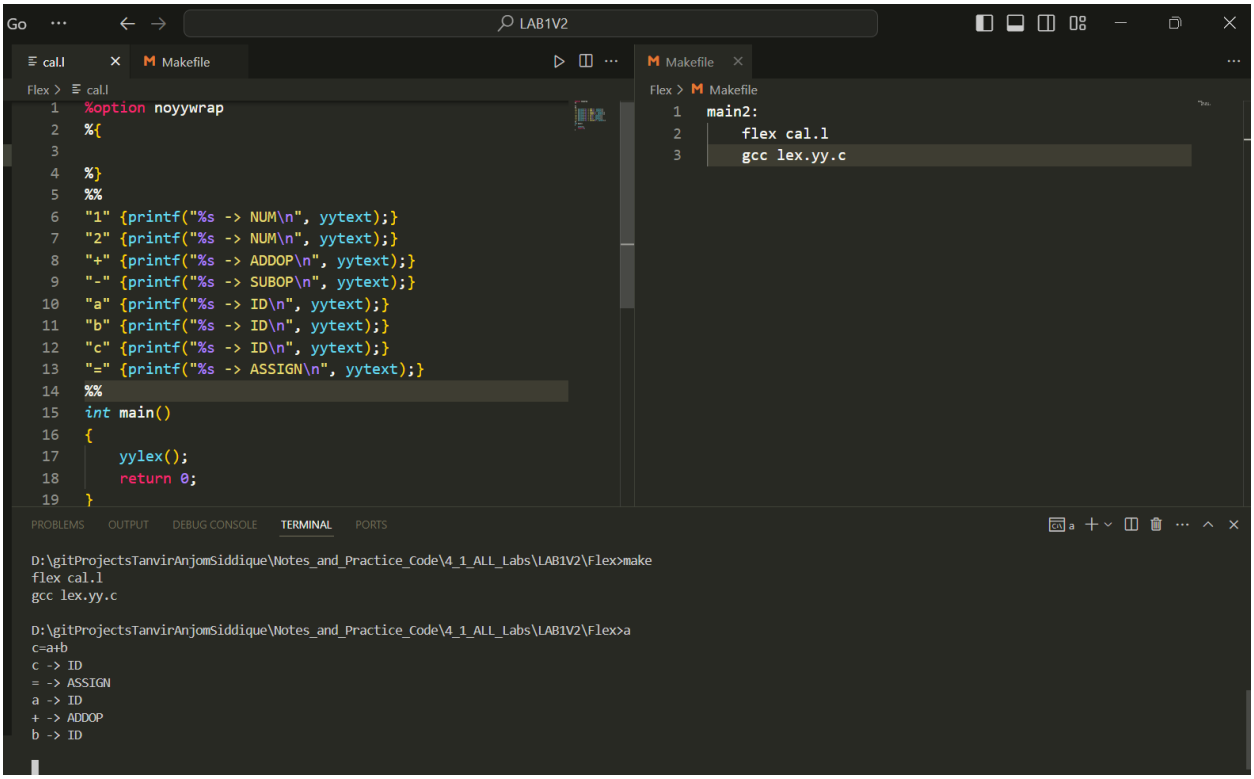




*** Highlight cal.l > bison & lex (extension install)



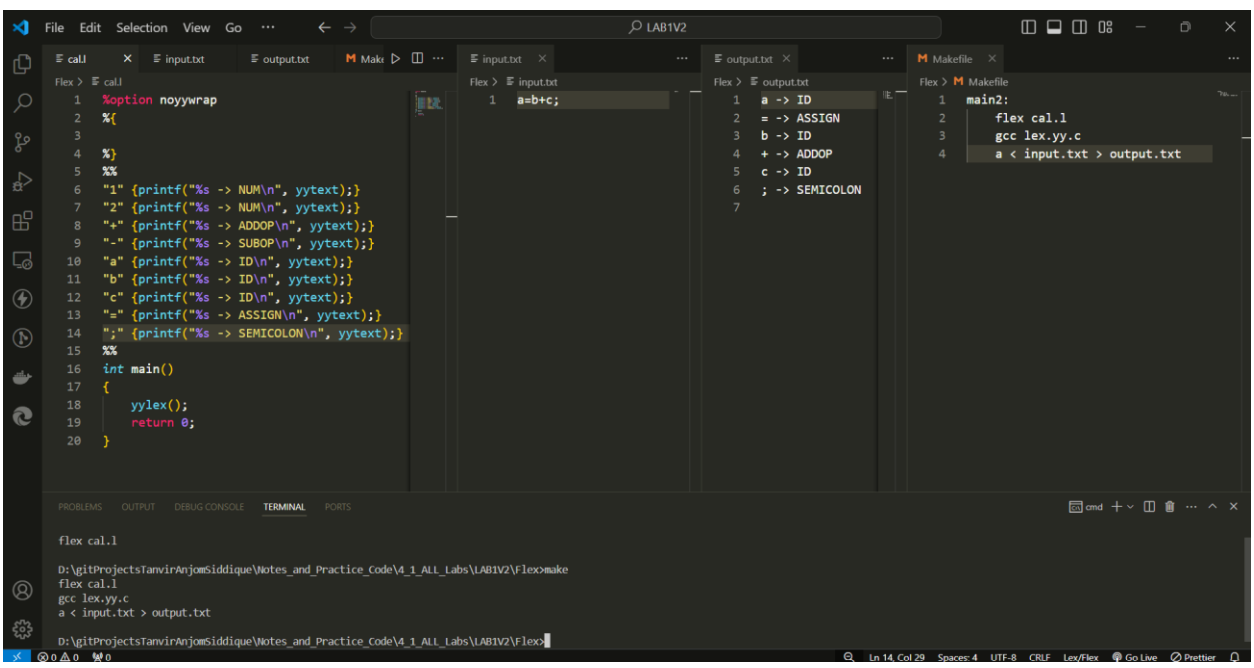

cal.l
%option noyywrap
%{
%}
%%
"1" {printf("%s -> NUM\n", yytext);}
"2" {printf("%s -> NUM\n", yytext);}
"+" {printf("%s -> ADDOP\n", yytext);}
"-" {printf("%s -> SUBOP\n", yytext);}
%%
int main()
{
yylex();
return 0;
}
Makefile
main2:
flex cal.l
gcc lex.yy.c
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>make
flex cal.l
gcc lex.yy.c
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>a
1+2
1 -> NUM
+ -> ADDOP
2 -> NUM
1-2
1 -> NUM
- -> SUBOP
2 -> NUM


cal.l
%option noyywrap
%{
%}
%%
"1" {printf("%s -> NUM\n", yytext);}
"2" {printf("%s -> NUM\n", yytext);}
"+" {printf("%s -> ADDOP\n", yytext);}
"-" {printf("%s -> SUBOP\n", yytext);}
"a" {printf("%s -> ID\n", yytext);}
"b" {printf("%s -> ID\n", yytext);}
"c" {printf("%s -> ID\n", yytext);}
"=" {printf("%s -> ASSIGN\n", yytext);}
%%
int main()
{
yylex();
return 0;
}
Makefile
main2:
flex cal.l
gcc lex.yy.c
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>make
flex cal.l
gcc lex.yy.c
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>a
c=a+b
c -> ID
= -> ASSIGN
a -> ID
+ -> ADDOP
b -> ID

cal.l

```
%option noyywrap
%{

%}
%%
"1" {printf("%s -> NUM\n", yytext);}
"2" {printf("%s -> NUM\n", yytext);}
"+" {printf("%s -> ADDOP\n", yytext);}
"-" {printf("%s -> SUBOP\n", yytext);}
"a" {printf("%s -> ID\n", yytext);}
"b" {printf("%s -> ID\n", yytext);}
"c" {printf("%s -> ID\n", yytext);}
"=" {printf("%s -> ASSIGN\n", yytext);}
%%
int main()
{
    yylex();
    return 0;
}
```

Take input from “input.txt” show ouput in “output.txt”



Lab 02: Lexical & Syntax Analysis

Identify Token

cal.l

```
%option noyywrap
%{

%}
%%
"1" {printf("%s -> NUM\n", yytext);}
"2" {printf("%s -> NUM\n", yytext);}
"+" {printf("%s -> ADDOP\n", yytext);}
"-" {printf("%s -> SUBOP\n", yytext);}
"a" {printf("%s -> ID\n", yytext);}
"b" {printf("%s -> ID\n", yytext);}
"c" {printf("%s -> ID\n", yytext);}
"=" {printf("%s -> ASSIGN\n", yytext);}
";" {printf("%s -> SEMICOLON\n", yytext);}
%%
int main()
{
    yylex();
    return 0;
}
```

Makefile

```
main:
    flex cal.l
    gcc lex.yy.c
    a
```

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make
flex cal.l
gcc lex.yy.c
```

```
a
1+2
1 -> NUM
+ -> ADDOP
2 -> NUM
```

lex.yy.c will be created then, we will compile it with gcc and run it a.exe

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                   make

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make
flex cal.l
gcc lex.yy.c
a
1+2
1 -> NUM
+ -> ADDOP
2 -> NUM
```

Read input from a file

Makefile

```
main:
    flex cal.l
    gcc lex.yy.c
    ./a <input.txt
```

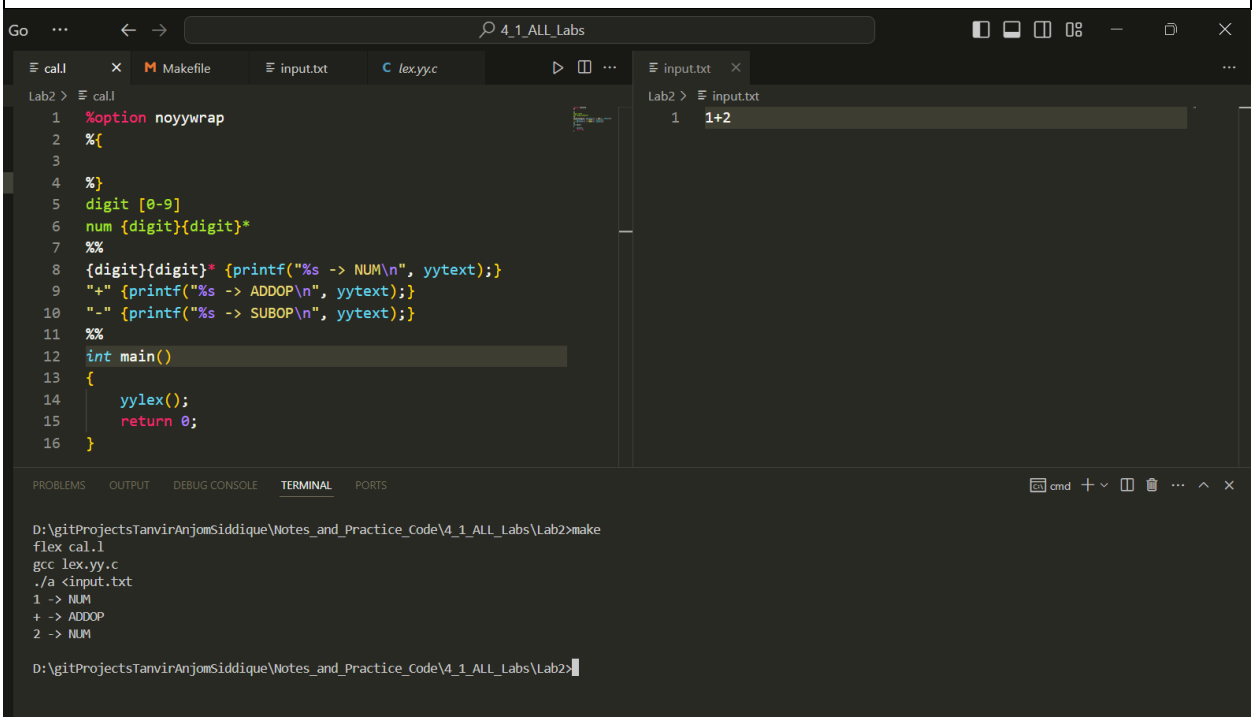
```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make
flex cal.l
gcc lex.yy.c
./a <input.txt
1 -> NUM
+ -> ADDOP
2 -> NUM
```

Use RE rules:

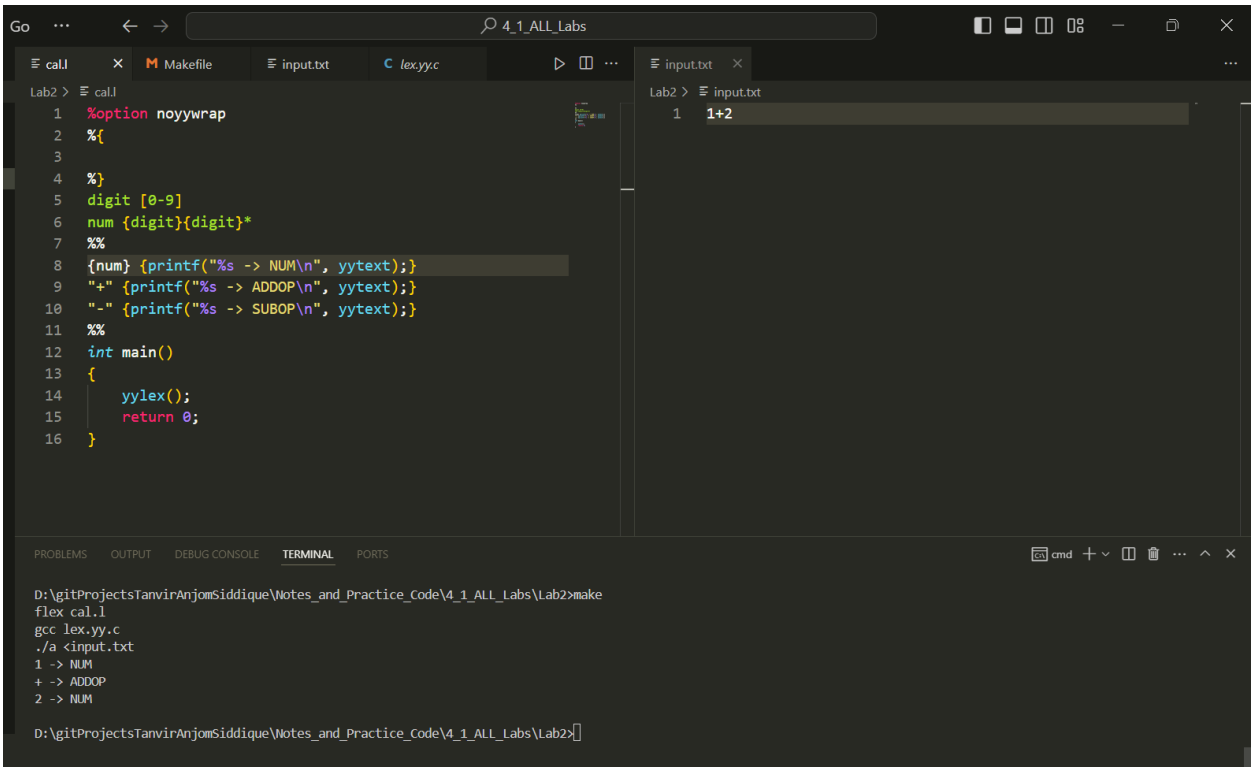
cal.l

```
%option noyywrap
%{

%}
digit [0-9]
num {digit}{digit}*
%%
{digit}{digit}* {printf("%s -> NUM\n", yytext);}
"+" {printf("%s -> ADDOP\n", yytext);}
"-" {printf("%s -> SUBOP\n", yytext);}
```

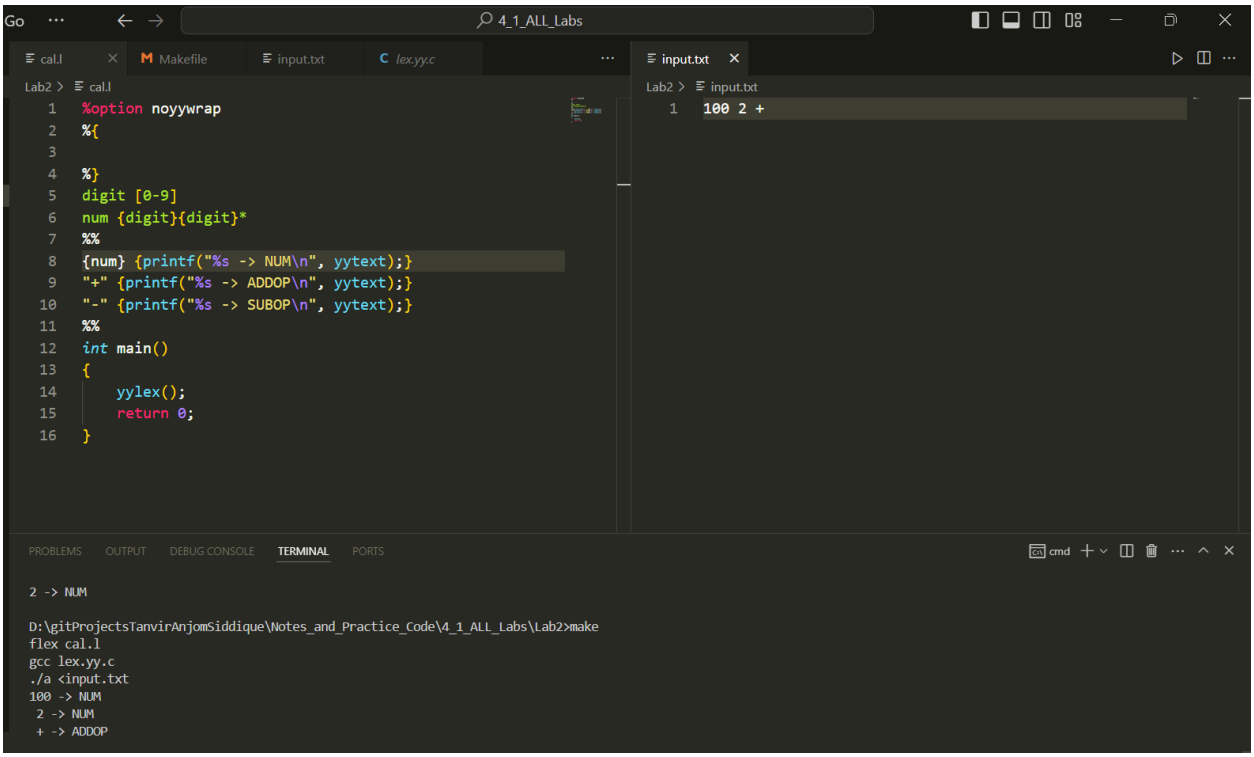
```
%%
int main()
{
    yylex();
    return 0;
}
```



Or,



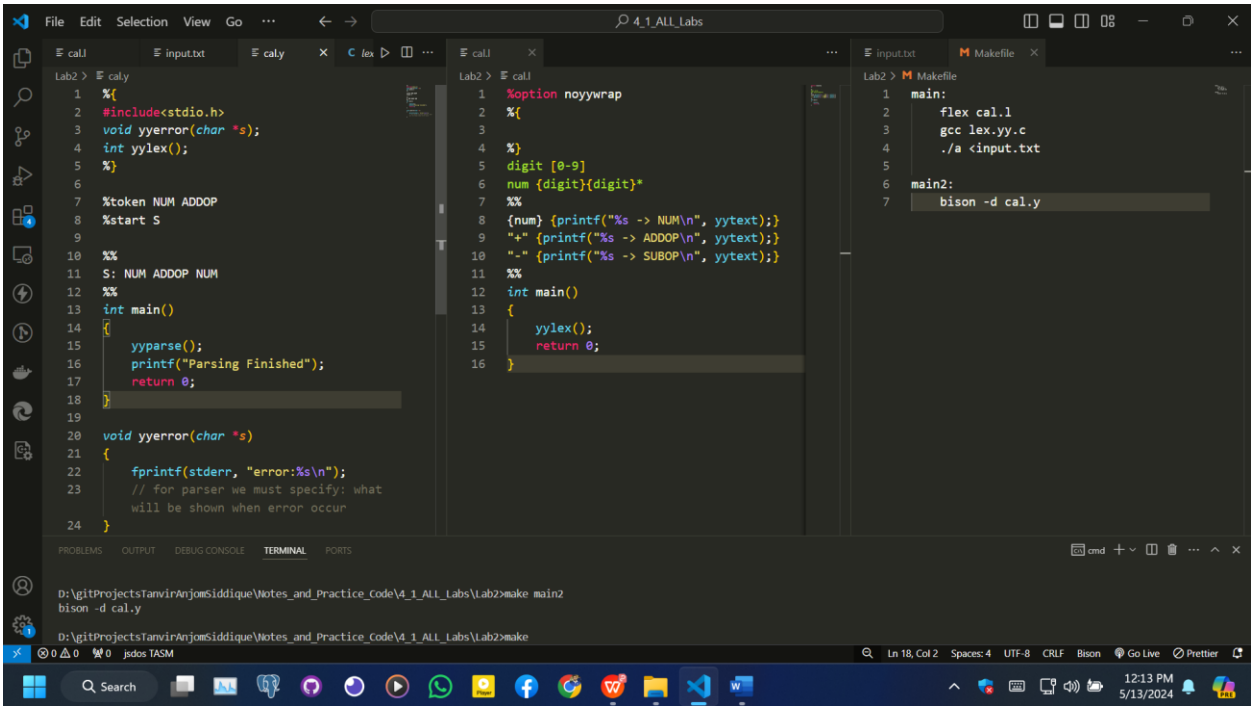
This don't check syntax but token only



BISON:

cal.y

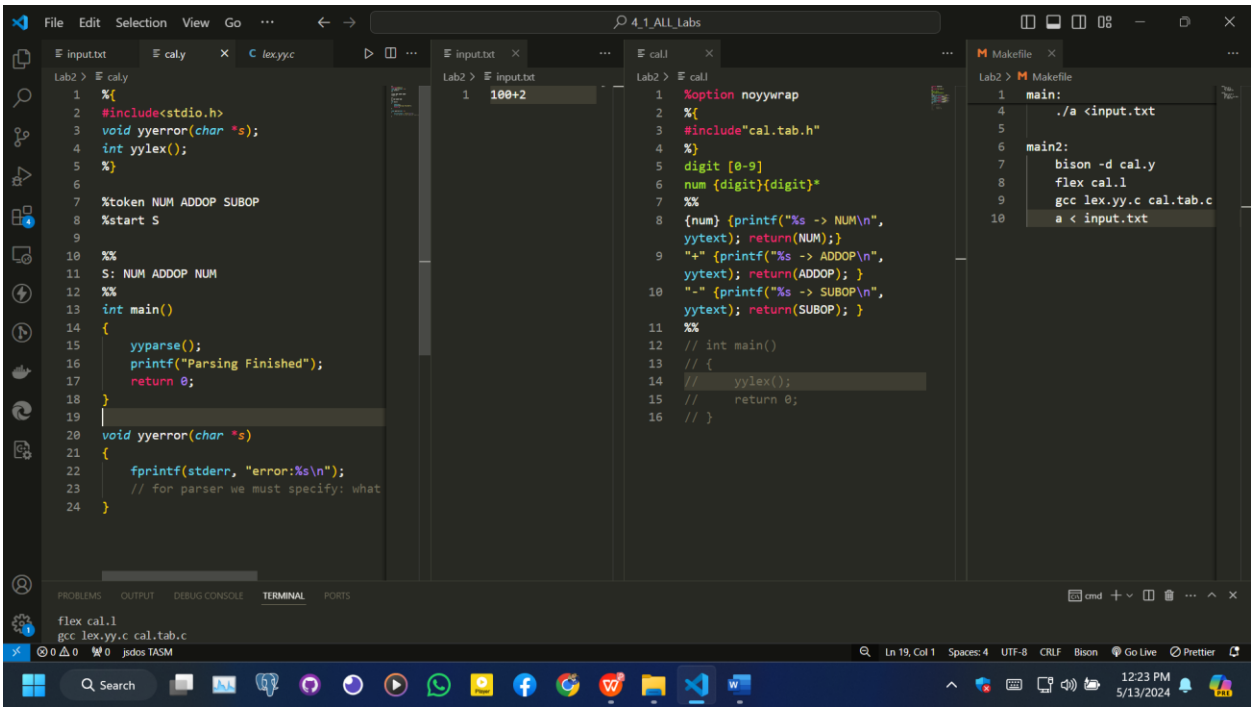
```
%{
#include<stdio.h>
void yyerror(char *s);
int yylex();
%}

%token NUM ADDOP
%start S

%%
S: NUM ADDOP NUM
%%
int main()
{
    yyparse();
    printf("Parsing Finished");
    return 0;
}

void yyerror(char *s)
{
    fprintf(stderr, "error:%s\n");
    // for parser we must specify: what will be shown when error
occur
}
```



cal.tab.c

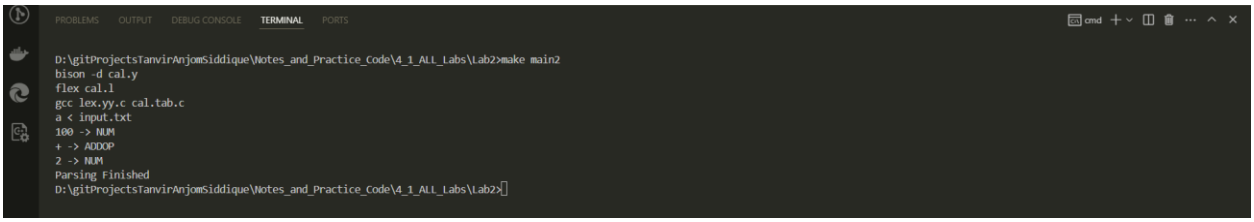
and

cal.tab.h will be created

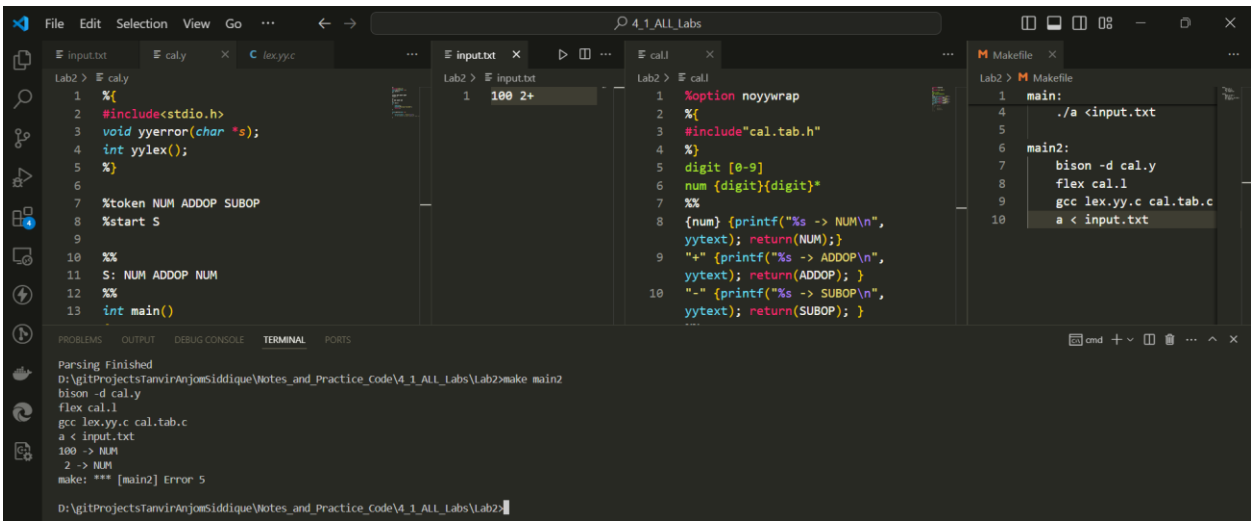


Terminal:

```
make main2
```



If there is error in input's syntax in file:



Parse Program line: int a=10;

Copy cal.l to prog1.l

Copy cal.y to prog1.y

prog1.l

```
%option noyywrap
%{
#include"prog1.tab.h"
%}
letter [a-zA-Z]
digit [0-9]
num {digit}{digit}*
id ({letter}|"_")({letter}|"_"|{digit})*
%%
"int" {printf("%s -> INT_TYPE\n", yytext); return(INT_TYPE);}
{id} {printf("%s -> ID\n", yytext); return(ID);}
```

```
{num} {printf("%s -> NUM\n", yytext); return(NUM);}
"=" {printf("%s -> ASSIGN\n", yytext); return(ASSIGN); }
";" {printf("%s -> SEMI\n", yytext); return(SEMI); }
%%
// int main()
// {
//     yylex();
//     return 0;
// }
```

prog1.y

```
%{
#include<stdio.h>
void yyerror(char *s);
int yylex();
%}

%token INT_TYPE ID NUM ASSIGN SEMI
%start S

%%
S: INT_TYPE ID ASSIGN NUM SEMI
%%
int main()
{
    yyparse();
    printf("Parsing Finished");
    return 0;
}

void yyerror(char *s)
{
    fprintf(stderr, "error:%s\n");
    // for parser we must specify: what will be shown when error
occur
}
```

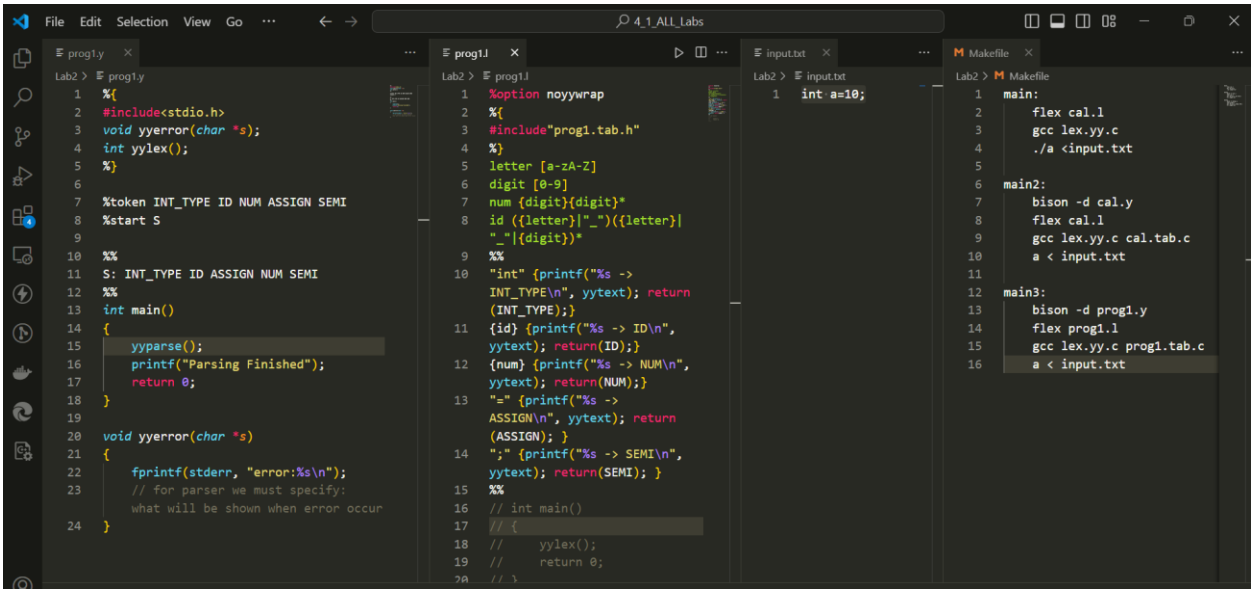
input.txt
int a=10;

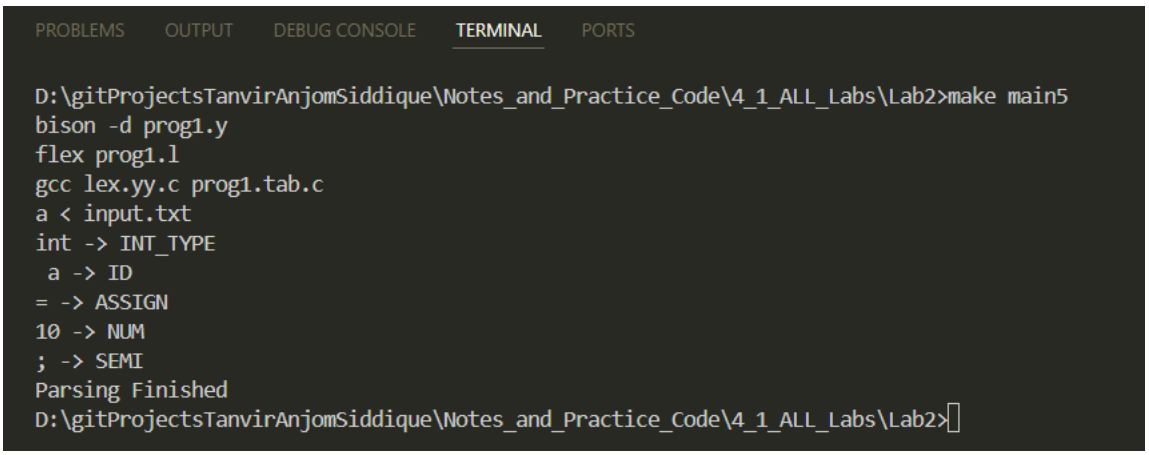
Makefile

```
main:
    flex cal.l
    gcc lex.yy.c
    ./a <input.txt

main2:
    bison -d cal.y
    flex cal.l
    gcc lex.yy.c cal.tab.c
    a < input.txt

main3:
    bison -d prog1.y
    flex prog1.l
    gcc lex.yy.c prog1.tab.c
    a < input.txt
```

Terminal

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make main5
bison -d prog1.y
flex prog1.l
gcc lex.yy.c prog1.tab.c
a < input.txt
int -> INT_TYPE
 a -> ID
= -> ASSIGN
10 -> NUM
; -> SEMI
Parsing Finished
```





Parse While loop:

Parser: prog1.y → prog1.tab.c , prog1.tab.h

prog1.y

```
%{
#include<stdio.h>
void yyerror(char *s);
int yylex();
%}

%token INT_TYPE ID NUM ASSIGN SEMI WHILE LP LT RP LB INCREMENT ADDOP
RB
%start S
```

```
%%
S: declr while_loop;
declr: INT_TYPE ID ASSIGN NUM SEMI
while_loop: WHILE LP ID LT NUM RP LB ID INCREMENT SEMI RB
%%
int main()
{
    yyparse();
    printf("Parsing Finished");
    return 0;
}

void yyerror(char *s)
{
    fprintf(stderr, "error:%s\n");
    // for parser we must specify: what will be shown when error
occur
}
```

Lexical Analyzer: prog1.l

prog1.l

```
%option noyywrap
%{
#include"prog1.tab.h"
%}
letter [a-zA-Z]
digit [0-9]
num {digit}{digit}*
id ({letter}|"_")({letter}|"_"|{digit})*
delim [ \n]
%%
{delim} {}
"int" {printf("%s -> INT_TYPE\n", yytext); return(INT_TYPE);}
"while" {printf("%s -> WHILE\n", yytext); return(WHILE); }
{id} {printf("%s -> ID\n", yytext); yylval=atoi(yytext); return(ID);}
{num} {printf("%s -> NUM\n", yytext); return(NUM);}
"=" {printf("%s -> ASSIGN\n", yytext); return(ASSIGN); }
```

```
";" {printf("%s -> SEMI\n", yytext); return(SEMI); }
"(" {printf("%s -> LP\n", yytext); return(LP); }
"<" {printf("%s -> LT\n", yytext); return(LT); }
")" {printf("%s -> RP\n", yytext); return(RP); }
"{" {printf("%s -> LB\n", yytext); return(LB); }
"++" {printf("%s -> INCREMENT\n", yytext); return(INCREMENT); }
"+" {printf("%s -> ADDOP\n", yytext); return(ADDOP); }
"}" {printf("%s -> RB\n", yytext); return(RB); }
%%
// int main()
// {
//     // while and int --> match with id --> so write their rule
before id
//     // {delim} {}  --> space or \n (newline)  no operation {}
//     yylex();
//     return 0;
// }
```

Input: input.txt

input.txt

```
int i=0;
while ( i<5 ){
   i++;
}
```

Makefile

Makefile

```
main:
    flex cal.l
    gcc lex.yy.c
    ./a <input.txt

main2:
    bison -d cal.y
    flex cal.l
    gcc lex.yy.c cal.tab.c
    a < input.txt
```

```
main3:
    bison -d prog1.y
    flex prog1.l
    gcc lex.yy.c prog1.tab.c
    a < input.txt
```

Terminal:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                   cmd

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make main3
bison -d prog1.y
flex prog1.l
gcc lex.yy.c prog1.tab.c
a < input.txt
int -> INT_TYPE
i -> ID
= -> ASSIGN
0 -> NUM
; -> SEMI
while -> WHILE
( -> LP
i -> ID
< -> LT
5 -> NUM
) -> RP
{ -> LB
i -> ID
++ -> INCREMENT
; -> SEMI
} -> RB
Parsing Finished
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>
```

*** Parser (cal.y->cal.tab.c, cal.tab.h) , Lexical Analyzer (cal.l-> lex.yy.c) all commands Makefile***

Makefile

```
main3:
    bison -d prog1.y
    flex prog1.l
    gcc lex.yy.c prog1.tab.c
    a < input.txt
```

1. **Generate Parser**:
   - `bison -d prog1.y`
     - This command runs `bison` on the `prog1.y` file, generating `prog1.tab.c` and `prog1.tab.h`.

2. **Generate Lexical Analyzer**:
   - `flex prog1.l`
     - This command runs `flex` on the `prog1.l` file, generating `lex.yy.c`.
3. **Compile Lexer and Parser**:
   - `gcc lex.yy.c prog1.tab.c`
     - This compiles both the lexical analyzer (`lex.yy.c`) and the parser (`prog1.tab.c`) using `gcc`, producing an executable (default name `a.out` or `a`).
4. **Run the Executable**:
   - `a < input.txt`
     - This runs the compiled program with `input.txt` as its input.

*** Parse & Lexically Analyze "While loop" in short: ***

Parser code: prog1.y

```
%{
#include<stdio.h>
void yyerror(char *s);
int yylex();
%}

%token INT_TYPE ID NUM ASSIGN SEMI WHILE LP LT RP LB INCREMENT ADDOP
RB
%start S

%%
S: declr while_loop;
declr: INT_TYPE ID ASSIGN NUM SEMI
while_loop: WHILE LP ID LT NUM RP LB ID INCREMENT SEMI RB
%%
int main()
{
    yyparse();
    printf("Parsing Finished");
    return 0;
}
```

```
void yyerror(char *s)
{
    fprintf(stderr, "error:%s\n");
    // for parser we must specify: what will be shown when error
occur
}
```

```
Terminal >
    bison -d prog1.y
```

Lexical Analyzer Code: prog1.l

```
%option noyywrap
%{
#include"prog1.tab.h"
%}
letter [a-zA-Z]
digit [0-9]
num {digit}{digit}*
id ({letter}|"_")({letter}|"_"|{digit})*
delim [ \n]
%%
{delim} {}
"int" {printf("%s -> INT_TYPE\n", yytext); return(INT_TYPE);}
"while" {printf("%s -> WHILE\n", yytext); return(WHILE); }
{id} {printf("%s -> ID\n", yytext); yylval=atoi(yytext); return(ID);}
{num} {printf("%s -> NUM\n", yytext); return(NUM);}
"=" {printf("%s -> ASSIGN\n", yytext); return(ASSIGN); }
";" {printf("%s -> SEMI\n", yytext); return(SEMI); }
"(" {printf("%s -> LP\n", yytext); return(LP); }
"<" {printf("%s -> LT\n", yytext); return(LT); }
")" {printf("%s -> RP\n", yytext); return(RP); }
"{" {printf("%s -> LB\n", yytext); return(LB); }
"++" {printf("%s -> INCREMENT\n", yytext); return(INCREMENT); }
"+" {printf("%s -> ADDOP\n", yytext); return(ADDOP); }
```

```
"}" {printf("%s -> RB\n", yytext); return(RB); }
%%
// int main()
// {
//     // while and int --> match with id --> so write their rule
before id
//     // {delim} {}  --> space or \n (newline)  no operation {}
//     yylex();
//     return 0;
// }
```

```
Terminal>
    flex prog1.l
    gcc lex.yy.c prog1.tab.c
    a < input.txt
```