

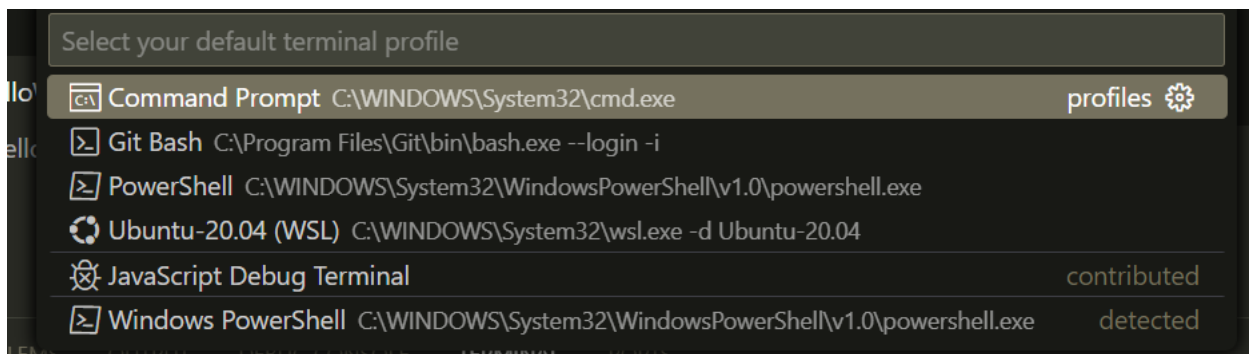
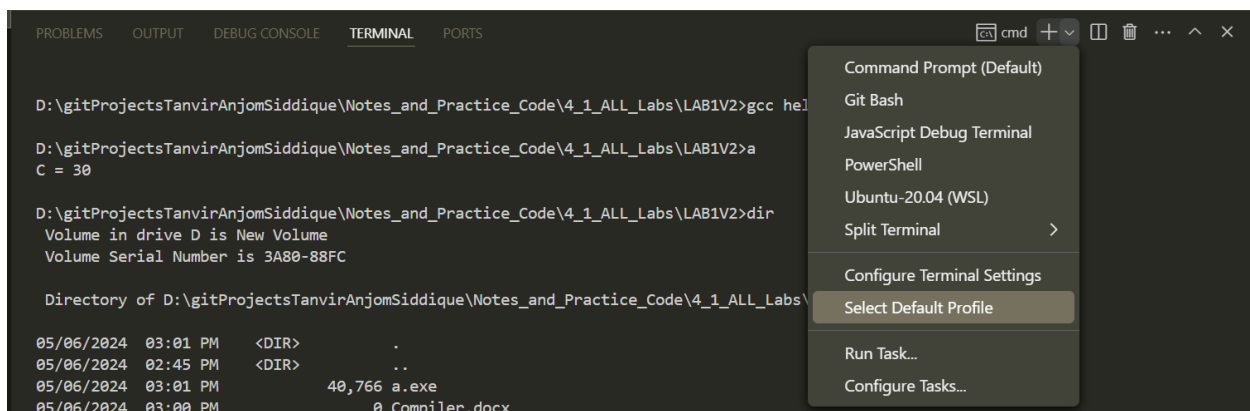
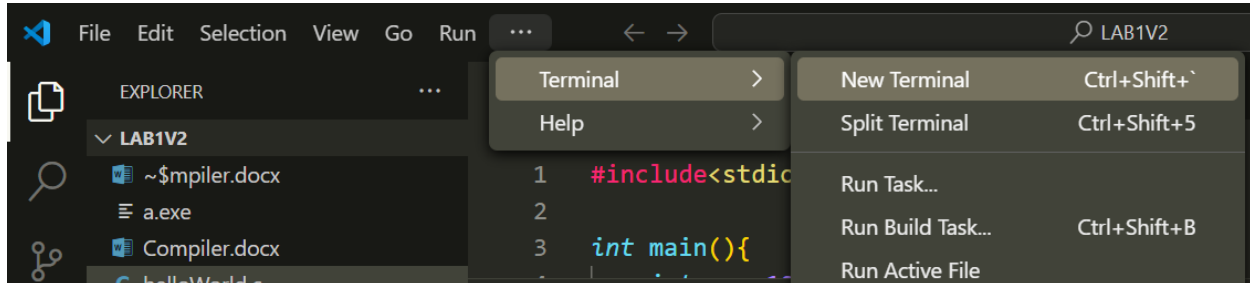
## Table of Contents

Tanvir Anjom Siddique .....	2
Lab 1:.....	2
Setup Terminal: .....	2
Compile Code:.....	3
Preprocess:.....	3
Create Assembly file (.i to .s):.....	4
*** All Steps Of Compiling C Program *** .....	6
Explanation .....	6
Summary .....	8
# Build System:.....	8
**Makefile (compile multiple file): .....	8
Flex (lexical analysis): [Identify ID , VAR, STRING, OP] .....	10
*** Highlight cal.l > bison & lex (extension install).....	11
Lab 02: Lexical & Syntax Analysis .....	14
Identify Token.....	14
Read input from a file.....	15
Use RE rules: .....	15
This don't check syntax but token only .....	17
BISON: .....	18
Parse Program line: int a=10; .....	20
Parse While loop: .....	23
*** Parser (cal.y ->cal.tab.c, cal.tab.h ) , Lexical Analyzer (cal.l -> lex.yy.c) all commands Makefile*** .....	26
1. <b>Generate Parser:</b> .....	26
2. <b>Generate Lexical Analyzer:</b> .....	27
3. <b>Compile Lexer and Parser:</b> .....	27
4. <b>Run the Executable:</b> .....	27
*** Parse & Lexically Analyze "While loop" in short: *** .....	27

## Tanvir Anjom Siddique

### Lab 1:

#### Setup Terminal:



## Compile Code:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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,0xfffff0
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`.

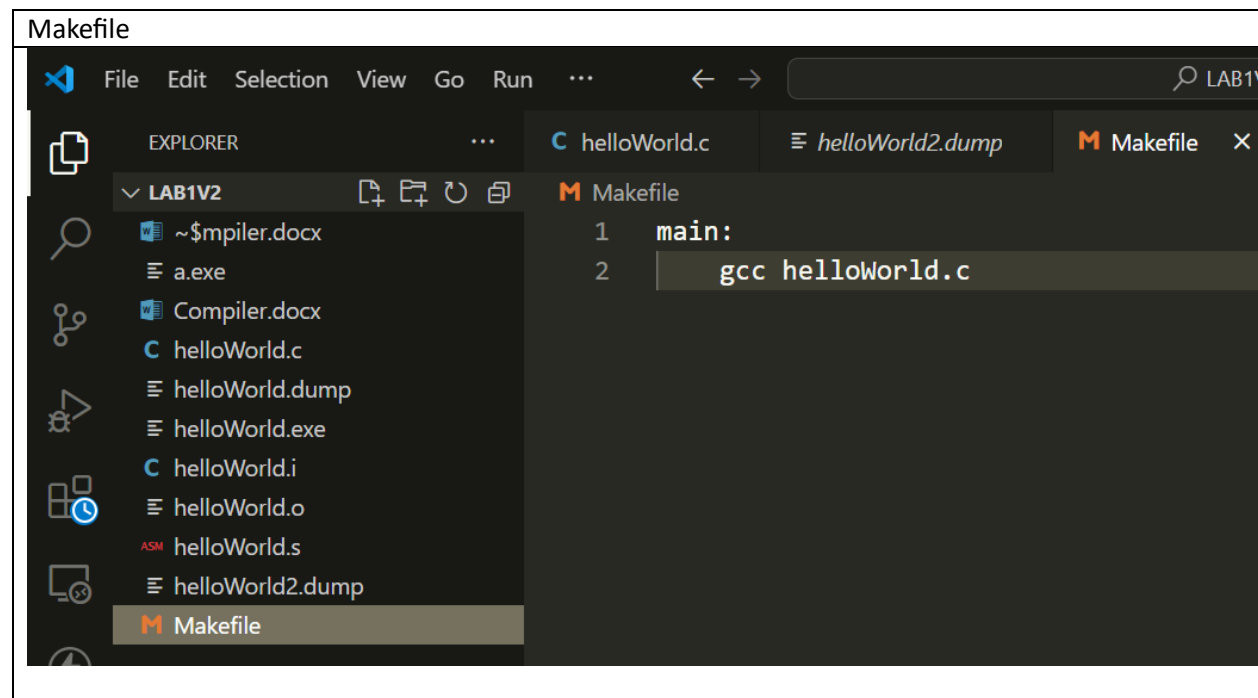
## Summary

- 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)



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):



The screenshot shows two files in an IDE. The left file, `HelloFunc.h`, contains the function declaration: `void HelloPrint(); //declare function`. The right file, `HelloFunc.c`, contains the implementation: `#include <stdio.h>`, `void HelloPrint()`, `{`, `printf("Hello World\n");`, and `}`.

The screenshot shows two files. The left file, `HelloMake.c`, contains the `main` function: `#include "HelloFunc.h"`, `int main()`, `{`, `HelloPrint();`, `return 0;`, and `}`. The right file, `Makefile`, contains the following rules: `main:`, `gcc -c -o HelloFunc.o HelloFunc.c`, `gcc -c -o HelloMake.o HelloMake.c`, and `gcc -o Hello HelloFunc.o HelloMake.o`.

The terminal window shows the following commands and output:

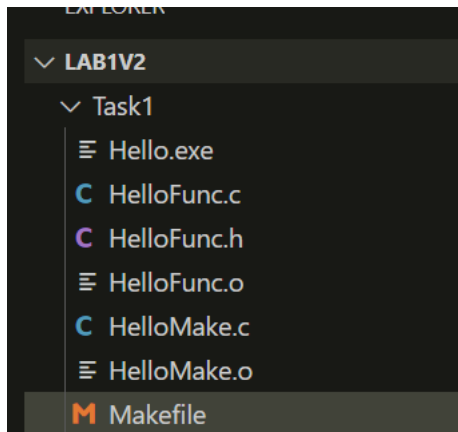
```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1>make
gcc -c -o HelloFunc.o HelloFunc.c
gcc -c -o HelloMake.o HelloMake.c
gcc -o Hello HelloFunc.o HelloMake.o

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

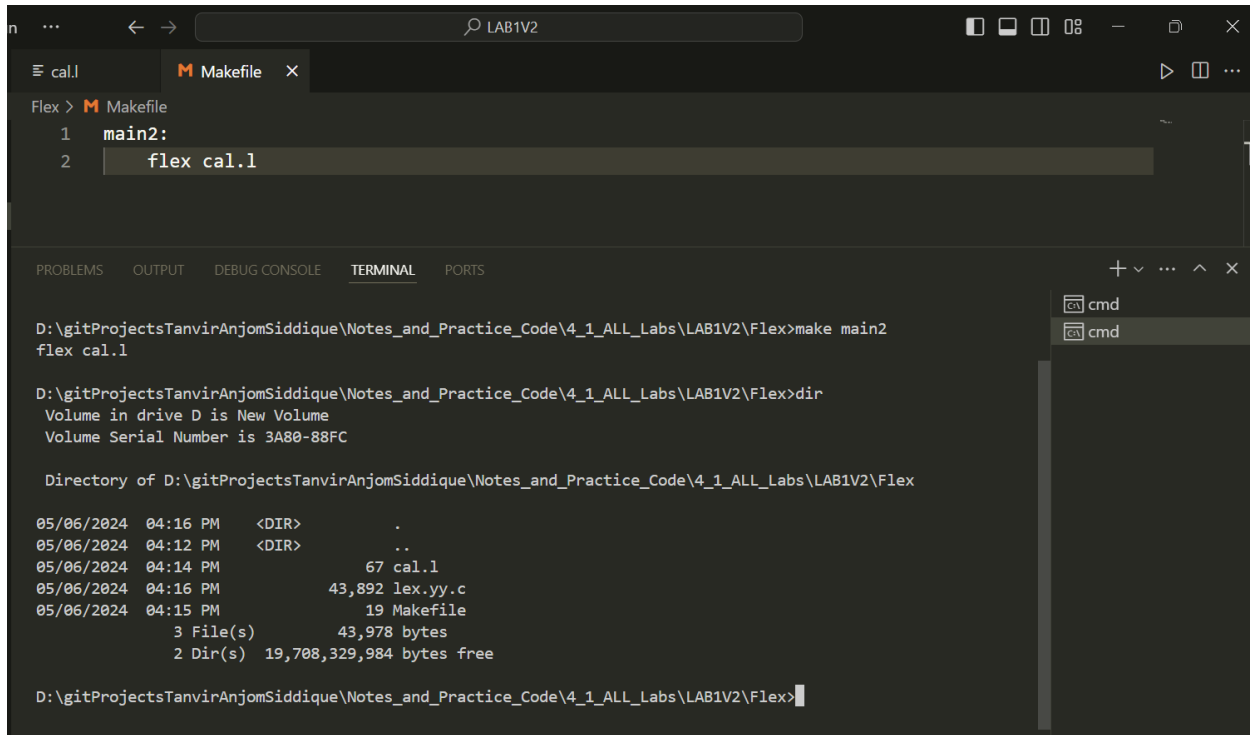
Directory of D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Task1

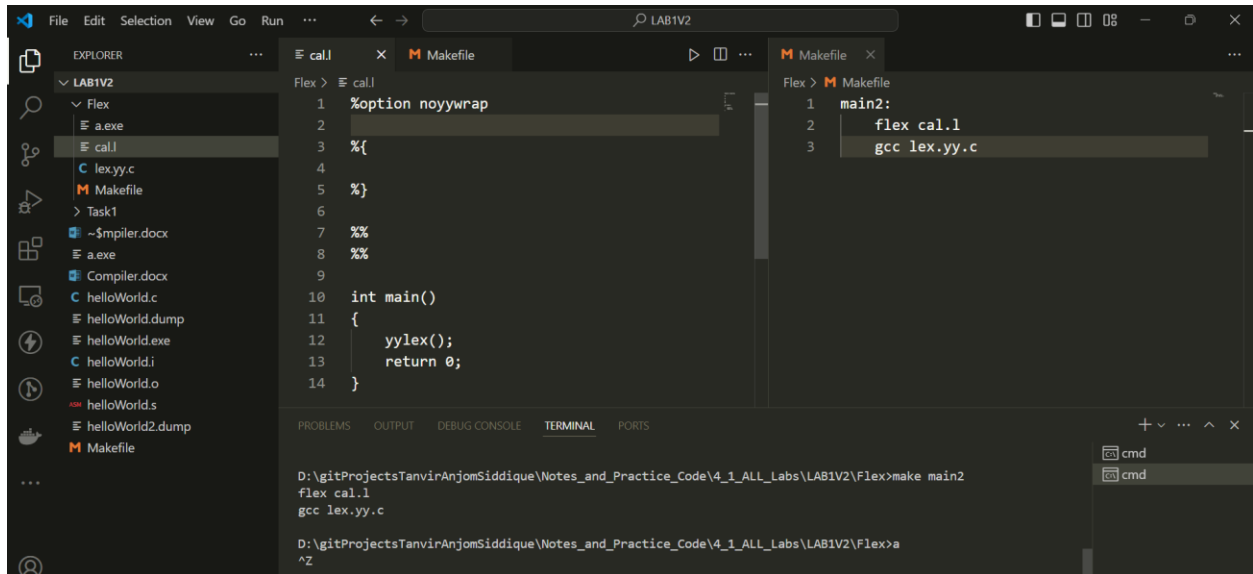
05/06/2024 03:55 PM <DIR> .
05/06/2024 03:55 PM <DIR> ..
05/06/2024 03:55 PM          41,028 Hello.exe
05/06/2024 03:44 PM           74 HelloFunc.c
05/06/2024 03:44 PM           36 HelloFunc.h
05/06/2024 03:55 PM          782 HelloFunc.o
05/06/2024 03:43 PM           76 HelloMake.c
05/06/2024 03:55 PM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



Files in different folders

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





```
File Edit Selection View Go Run ... LAB1V2
```

EXPLORER

- LAB1V2
  - Flex
    - a.exe
    - cal.l
    - lex.yy.c
    - Makefile
  - Task1
  - ~\$mpiler.docx
  - a.exe
  - Compiler.docx
  - helloWorld.c
  - helloWorld.dump
  - helloWorld.exe
  - helloWorld.i
  - helloWorld.o
  - helloWorld.s
  - helloWorld2.dump
  - Makefile

Flex > cal.l

```
1 %option noyywrap
2
3 %{
4
5 %}
6
7 %%
8 %%
9
10 int main()
11 {
12     yylex();
13     return 0;
14 }
```

Flex > Makefile

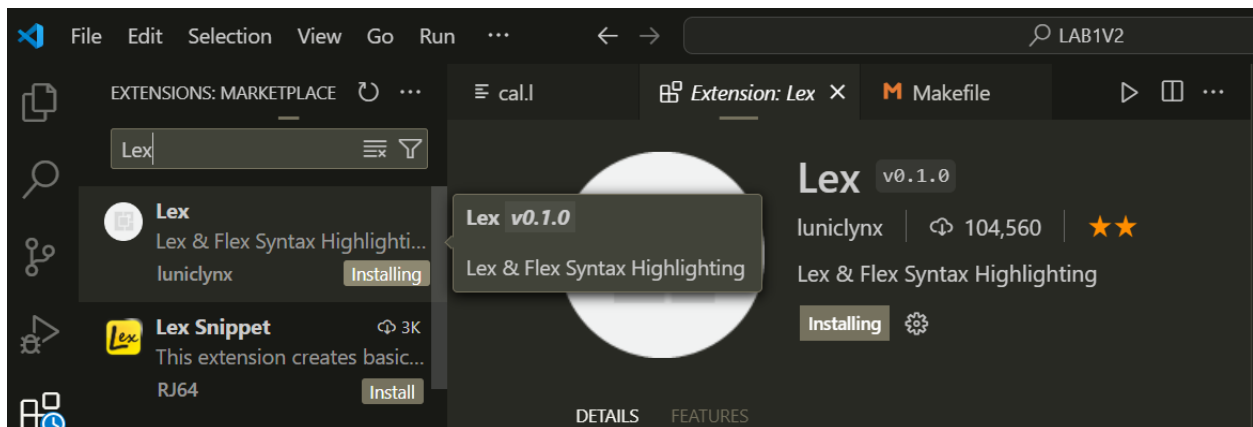
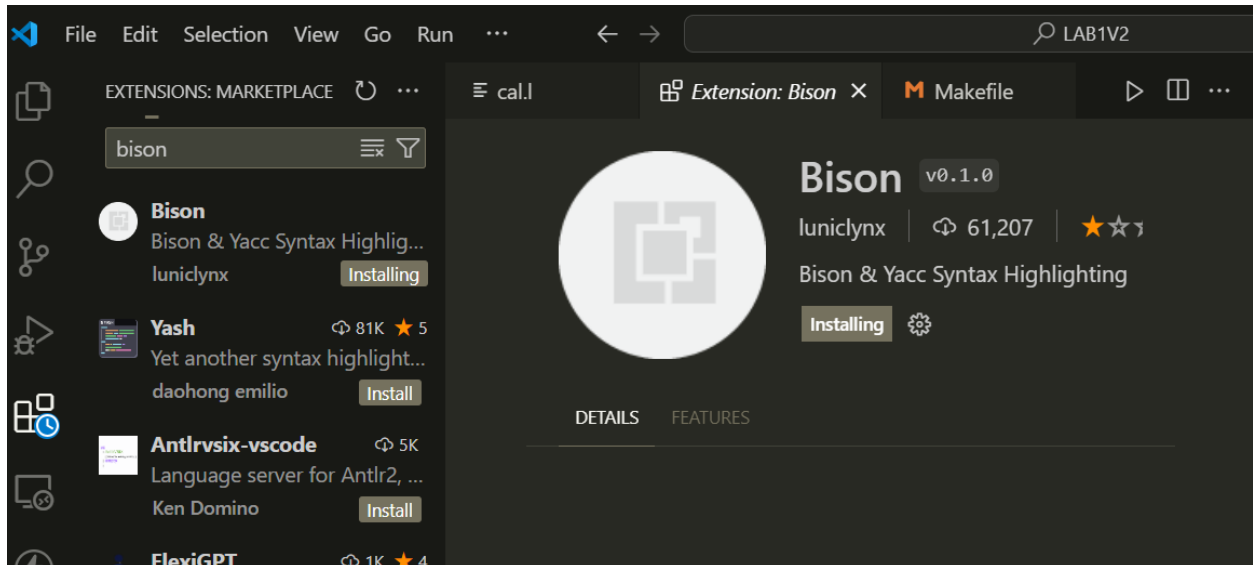
```
1 main2:
2     flex cal.l
3     gcc lex.yy.c
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>make main2
flex cal.l
gcc lex.yy.c

D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>a
^Z
```

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



```
1 %option noyywrap
2 %{
3
4 %}
5 %%
6 "1" {printf("%s -> NUM\n", yytext);}
7 "2" {printf("%s -> NUM\n", yytext);}
8 "+" {printf("%s -> ADDOP\n", yytext);}
9 "-" {printf("%s -> SUBOP\n", yytext);}
10 %%
11
12 int main()
13 {
14     yylex();
15     return 0;
16 }
```

```
1 main2:
2     flex cal.l
3     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
```

```
1 %option noyywrap
2 %{
3
4 %}
5 %%
6 "1" {printf("%s -> NUM\n", yytext);}
7 "2" {printf("%s -> NUM\n", yytext);}
8 "+" {printf("%s -> ADDOP\n", yytext);}
9 "-" {printf("%s -> SUBOP\n", yytext);}
10 "a" {printf("%s -> ID\n", yytext);}
11 "b" {printf("%s -> ID\n", yytext);}
12 "c" {printf("%s -> ID\n", yytext);}
13 "=" {printf("%s -> ASSIGN\n", yytext);}
14 %%
15
16 int main()
17 {
18     yylex();
19     return 0;
20 }
```

```
1 main2:
2     flex cal.l
3     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 output in “output.txt”

The screenshot shows a Visual Studio Code editor with the following content:

- cal1**: Flex source code (as shown in the previous block).
- input.txt**: Contains the input string `a=b+c;`.
- output.txt**: Contains the output of the lexer:
 

```

1 a -> ID
2 = -> ASSIGN
3 b -> ID
4 + -> ADDOP
5 c -> ID
6 ; -> SEMICOLON
7

```
- Makefile**: Contains the following rules:
 

```

1 main2:
2     flex cal.1
3     gcc lex.yy.c
4     a < input.txt > output.txt

```
- Terminal**: Shows the execution of the Makefile:
 

```

flex cal.1
D:\gitProjects\TanvirAnjomsiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>make
flex cal.1
gcc lex.yy.c
a < input.txt > output.txt
D:\gitProjects\TanvirAnjomsiddique\Notes_and_Practice_Code\4_1_ALL_Labs\LAB1V2\Flex>

```

## 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;
}
```

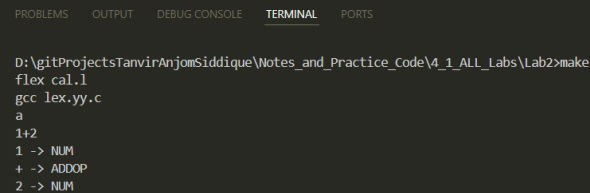
### Makefile

```
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



```
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

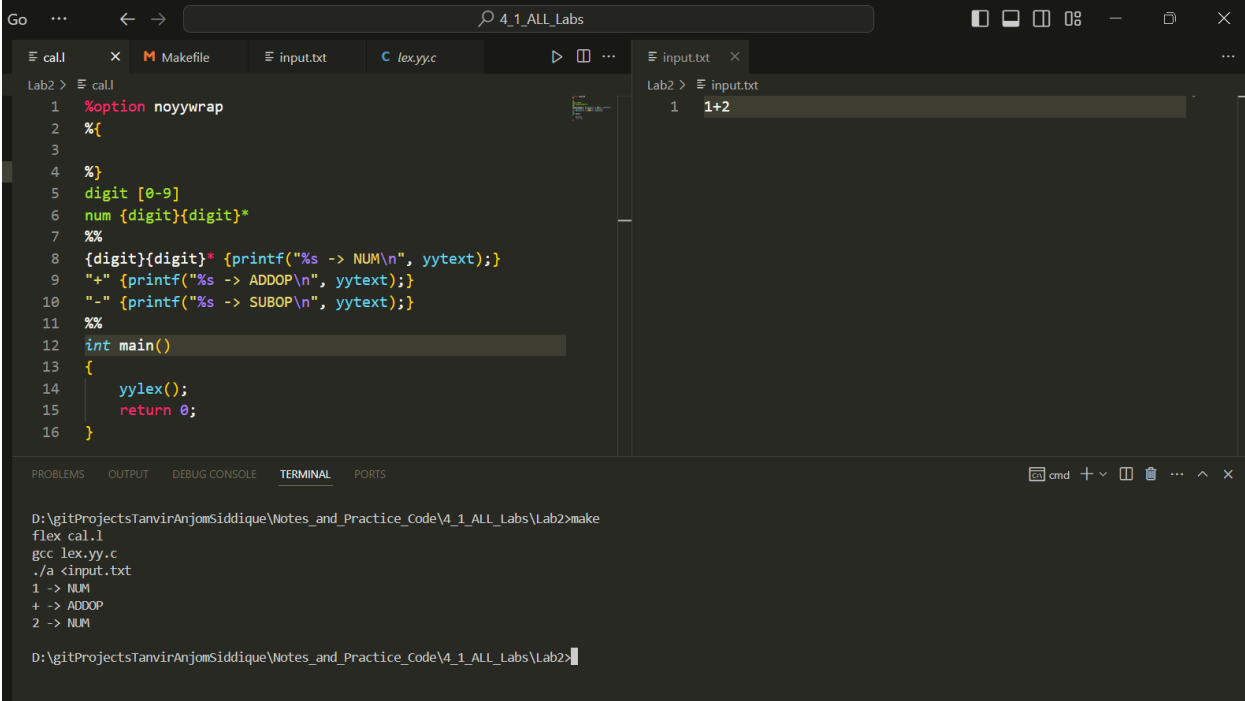
```
main:
    flex cal.l
    gcc lex.yy.c
    ./a <input.txt
```

```
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;
}
```



The screenshot shows a VS Code editor with the following content:

**cal.l**

```
1 %option noyywrap
2 %{
3
4 %}
5 digit [0-9]
6 num {digit}{digit}*
7 %%
8 {digit}{digit}* {printf("%s -> NUM\n", yytext);}
9 "+" {printf("%s -> ADDOP\n", yytext);}
10 "-" {printf("%s -> SUBOP\n", yytext);}
11 %%
12 int main()
13 {
14     yylex();
15     return 0;
16 }
```

**input.txt**

```
1 1+2
```

**Terminal**

```
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
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>
```

Or,



```

1 %option noyywrap
2 %{
3
4 %}
5 digit [0-9]
6 num {digit}{digit}*
7 %%
8 {num} {printf("%s -> NUM\n", yytext);}
9 "+" {printf("%s -> ADDOP\n", yytext);}
10 "-" {printf("%s -> SUBOP\n", yytext);}
11 %%
12 int main()
13 {
14     yylex();
15     return 0;
16 }

```

```

1 1+2

```

```

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
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>

```

This don't check syntax but token only

```

1 %option noyywrap
2 %{
3
4 %}
5 digit [0-9]
6 num {digit}{digit}*
7 %%
8 {num} {printf("%s -> NUM\n", yytext);}
9 "+" {printf("%s -> ADDOP\n", yytext);}
10 "-" {printf("%s -> SUBOP\n", yytext);}
11 %%
12 int main()
13 {
14     yylex();
15     return 0;
16 }

```

```

1 100 2 +

```

```

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

```

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
}
```

```

1  %{
2  #include<stdio.h>
3  void yyerror(char *s);
4  int yylex();
5  %}
6
7  %token NUM ADDOP
8  %start S
9
10 %%%
11 S: NUM ADDOP NUM
12 %%%
13
14 int main()
15 {
16     yyparse();
17     printf("Parsing Finished");
18     return 0;
19 }
20
21 void yyerror(char *s)
22 {
23     fprintf(stderr, "error:%s\n");
24     // for parser we must specify: what
25     // will be shown when error occur
26 }

```

```

1  %option noyywrap
2  %{
3
4  %}
5  digit [0-9]
6  num {digit}{digit}*
7  %%%
8  {num} {printf("%s -> NUM\n", yytext);}
9  "+" {printf("%s -> ADDOP\n", yytext);}
10 "-" {printf("%s -> SUBOP\n", yytext);}
11 %%%
12 int main()
13 {
14     yylex();
15     return 0;
16 }

```

```

1  main:
2
3      flex cal.l
4      gcc lex.yy.c
5      ./a <input.txt
6
7  main2:
8      bison -d cal.y

```

```

D:\gitProjects\tanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>make main2
bison -d cal.y

```

cal.tab.c

and

cal.tab.h will be created

```

1  100+2

```

```

1  %option noyywrap
2  %{
3
4  #include"cal.tab.h"
5  %}
6  digit [0-9]
7  num {digit}{digit}*
8  %%%
9  {num} {printf("%s -> NUM\n", yytext); return(NUM);}
10 "+" {printf("%s -> ADDOP\n", yytext); return(ADDOP);}
11 "-" {printf("%s -> SUBOP\n", yytext); return(SUBOP);}
12 %%%
13 // int main()
14 // {
15 //     yylex();
16 //     return 0;
17 // }

```

```

1  main:
2
3      ./a <input.txt
4
5
6  main2:
7      bison -d cal.y
8      flex cal.l
9      gcc lex.yy.c cal.tab.c
10     a < input.txt

```

```

flex cal.l
gcc lex.yy.c cal.tab.c

```

Terminal:



```

{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
```

```

Lab2 > prog1.y
1 %{}
2 #include<stdio.h>
3 void yyerror(char *s);
4 int yylex();
5 %}
6
7 %token INT_TYPE ID NUM ASSIGN SEMI
8 %start S
9
10 %%
11 S: INT_TYPE ID ASSIGN NUM SEMI
12 %%
13 int main()
14 {
15     yyparse();
16     printf("Parsing Finished");
17     return 0;
18 }
19 void yyerror(char *s)
20 {
21     fprintf(stderr, "error:%s\n");
22     // for parser we must specify:
23     // what will be shown when error occur
24 }

Lab2 > prog1.l
1 %option noyywrap
2 %{}
3 #include "prog1.tab.h"
4 %}
5 letter [a-zA-Z]
6 digit [0-9]
7 num {digit}{digit}*
8 id {(letter)|" "}{letter}
9 _ "{digit}"*
10
11 "int" {printf("%s ->
12 INT_TYPE\n", yytext); return
13 (INT_TYPE);}
14 {id} {printf("%s -> ID\n",
15 yytext); return (ID);}
16 {num} {printf("%s -> NUM\n",
17 yytext); return (NUM);}
18 "=" {printf("%s ->
19 ASSIGN\n", yytext); return
20 (ASSIGN);}
21 ";" {printf("%s -> SEMI\n",
22 yytext); return (SEMI);}
23 %%
24 // int main()
25 // {
26 //     yylex();
27 //     return 0;
28 // }

Lab2 > input.txt
1 int a=10;

Lab2 > Makefile
1 main:
2 flex cal.l
3 gcc lex.yy.c
4 ./a < input.txt
5
6 main2:
7 bison -d cal.y
8 flex cal.l
9 gcc lex.yy.c cal.tab.c
10 a < input.txt
11
12 main3:
13 bison -d prog1.y
14 flex prog1.l
15 gcc lex.yy.c prog1.tab.c
16 a < input.txt

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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
D:\gitProjectsTanvirAnjomSiddique\Notes_and_Practice_Code\4_1_ALL_Labs\Lab2>

```

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
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
```