



COMPILER DESIGN LABORATORY

6th SEMESTER 2018

Submitted by:
Arindum Roy
Roll Number- 115CS0233

Department of Computer Science and Engineering
National Institute of Technology, Rourkela

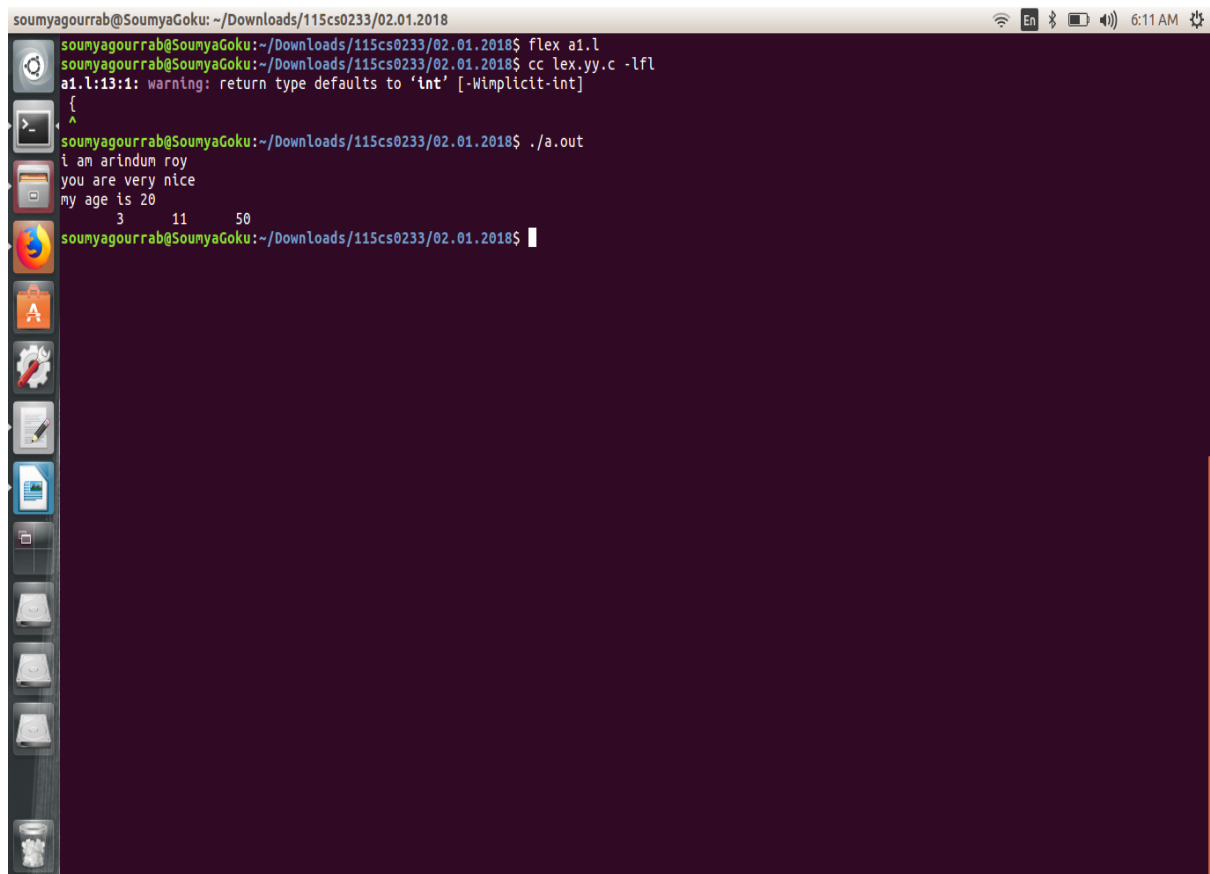
ASSIGNMENT-1

1. Write a flex program which reads through a file and reports the number of lines, words, and characters in the file.

Solution:

```
%{  
int chars = 0;  
int words = 0;  
int lines = 0;  
%}  
%%  
[a-zA-Z]+ { words++; chars += strlen(yytext); }  
\n { chars++; lines++; }  
. { chars++; }  
  
%%  
main(int argc, char **argv)  
{  
yylex();  
printf("%8d%8d%8d\n", lines, words, chars);  
}
```

Output:



```
soumyagourrab@SoumyaGoku: ~/Downloads/115cs0233/02.01.2018
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/02.01.2018$ flex a1.l
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/02.01.2018$ cc lex.yy.c -lfl
a1.l:13:1: warning: return type defaults to 'int' [-Wimplicit-int]
{
^
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/02.01.2018$ ./a.out
i am arindum roy
you are very nice
my age is 20
      3      11      50
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/02.01.2018$
```

The image shows a terminal window with a dark purple background. The window title is 'soumyagourrab@SoumyaGoku: ~/Downloads/115cs0233/02.01.2018'. The terminal displays the following commands and output:

- `flex a1.l`
- `cc lex.yy.c -lfl`
- A warning message: `a1.l:13:1: warning: return type defaults to 'int' [-Wimplicit-int]`
- The execution of `./a.out` produces the output:
`i am arindum roy`
`you are very nice`
`my age is 20`
 `3 11 50`

The terminal window has a sidebar on the left with various application icons, including a file manager, a web browser, and a terminal. The top of the window shows system status icons for network, battery, and time (6:11 AM).

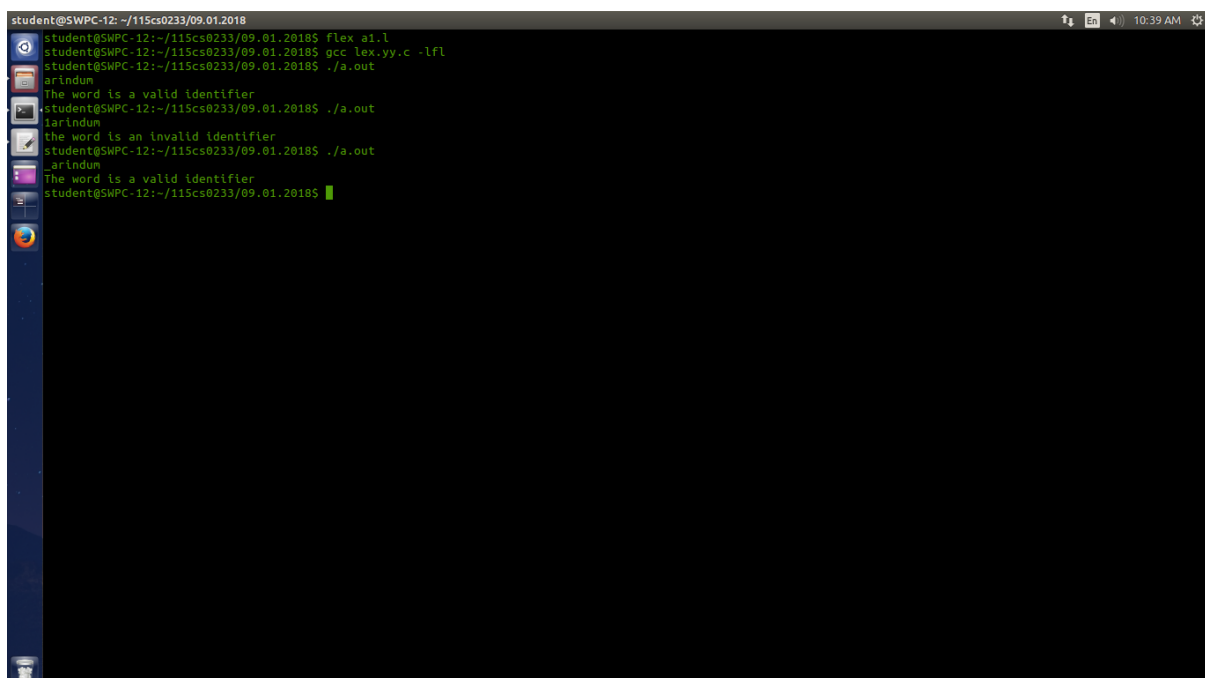
ASSIGNMENT-2

1. Write a flex program that will recognize a valid identifier having symbols a-z, A-Z, 0-9 and underscore. Each identifier should not start with a symbol like decimal or full stop, comma, semi colon, colon. (+, -, *, |, @, &, \$, #) are considered as invalid symbols.

Solution:

```
%{  
%}  
%%  
([a-zA-Z_][a-zA-Z0-9_]*) { printf("The word is a valid identifier\n");return 0;}  
. {printf("the word is an invalid identifier\n");return 0;}  
%%  
main(int argc, char **argv)  
{  
yylex();  
}
```

Output:



```
student@SWPC-12: ~/115cs0233/09.01.2018
student@SWPC-12:~/115cs0233/09.01.2018$ flex a1.l
student@SWPC-12:~/115cs0233/09.01.2018$ gcc lex.yy.c -lfl
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
arindum
The word is a valid identifier
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
iarindum
the word is an invalid identifier
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
.arindum
The word is a valid identifier
student@SWPC-12:~/115cs0233/09.01.2018$
```

2. Write a flex application that will recognize strings of odd 0's and even 1's.

Solution:

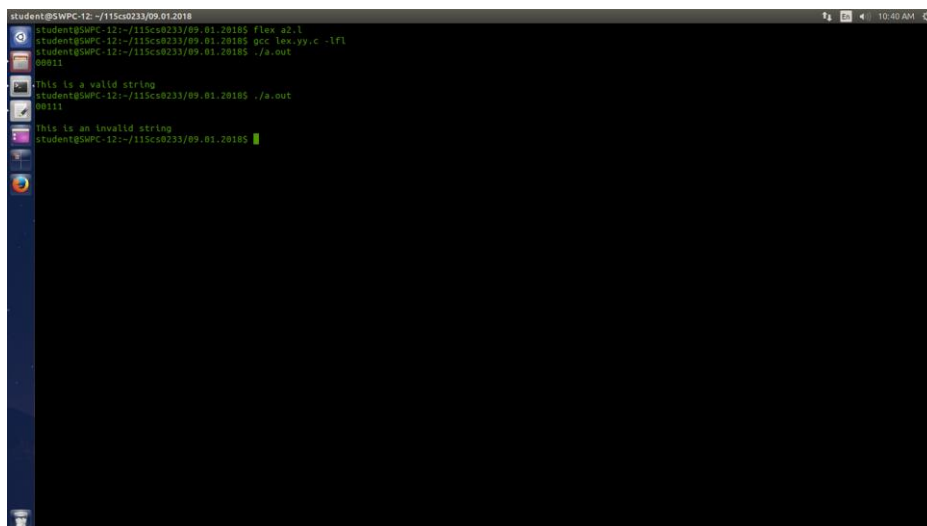
```
%{
int noz=0;
int noo=0;
%}
%%

[0] {noz++;}
[1] {noo++;}
. {printf("invalid");}

%%

main(int argc, char **argv) /** *BEGINNING OF THE MAIN FUNCTION */
{
yylex();
if(noz%2 !=0 && noo%2==0)
printf("This is a valid string\n");
else
printf("This is an invalid string\n");
}
```

Output:



```
student@SWPC-12: ~/115cs0233/09.01.2018
student@SWPC-12:~/115cs0233/09.01.2018$ flex a2.l
student@SWPC-12:~/115cs0233/09.01.2018$ gcc flex.yy.c -If1
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
00011
This is a valid string
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
00111
This is an invalid string
student@SWPC-12:~/115cs0233/09.01.2018$
```

3. Write a flex application that will identify signed or unsigned integers and long integer constant in decimal, hexadecimal, binary and octal representation in C.

Solution:

```
%{
%}

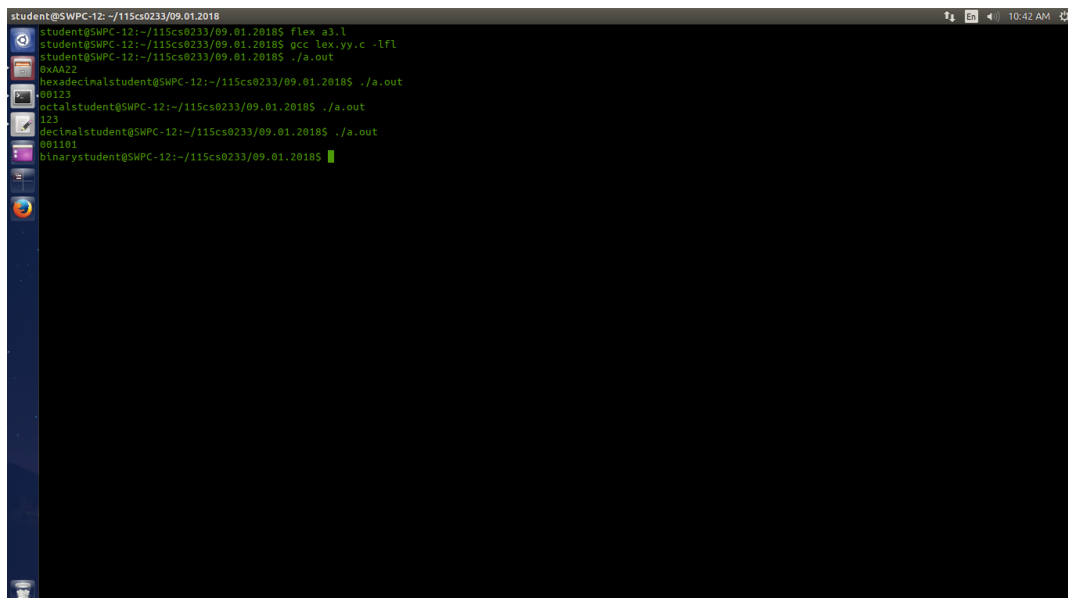
%%

[+]                { printf("unsigned");}
[-]                {printf("signed");}
0[xX][0-9a-fA-F]+ { printf("hexadecimal");return 0; }
[0-1]+             { printf("binary");return 0; }
0[0-7]+           { printf("octal");return 0; }
[0-9]+            { printf("decimal");return 0; }
.                  {printf("The word is invalid");return 0;}

%%

main(int argc, char **argv)
{
yylex();
}
```

Output:



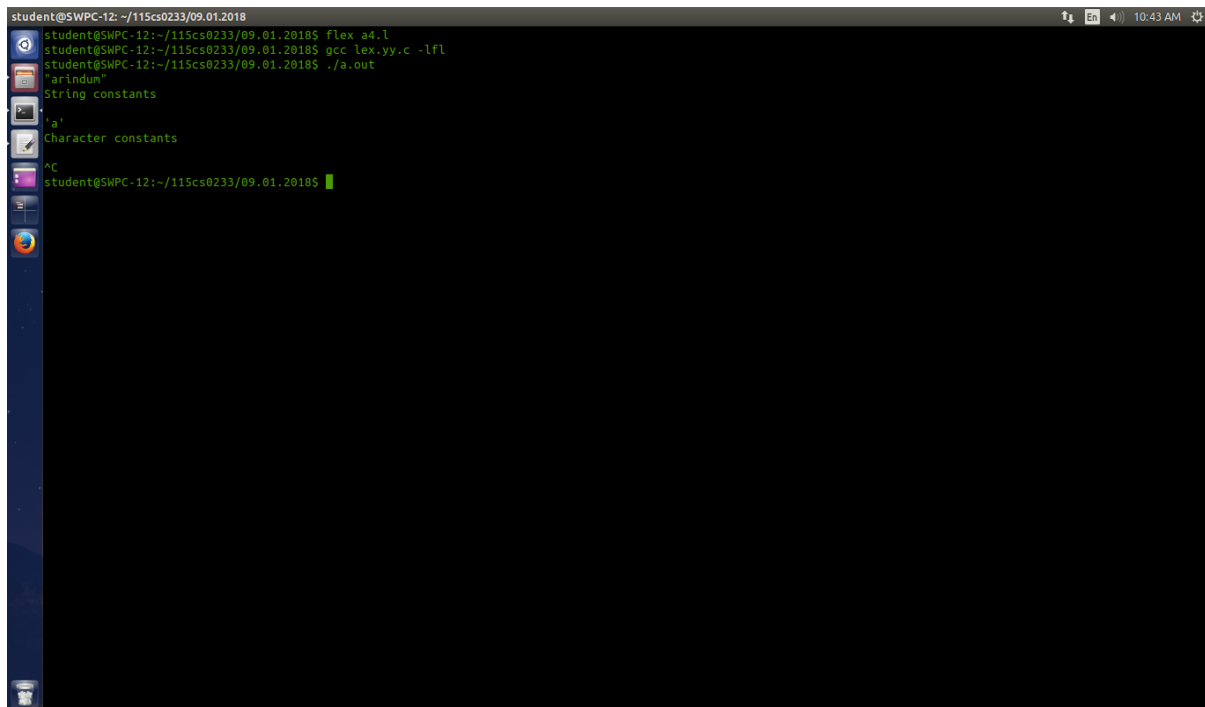
```
student@SWPC-12: ~/115cs0233/09.01.2018
student@SWPC-12:~/115cs0233/09.01.2018$ flex a3.l
student@SWPC-12:~/115cs0233/09.01.2018$ gcc lex.yy.c -lfl
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
0xAAA22
hexadecimalstudent@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
00123
octalstudent@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
123
decimalstudent@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
001101
binarystudent@SWPC-12:~/115cs0233/09.01.2018$
```

4. Write a flex program that will identify characters and string constants as identified in C language.

Solution:

```
%{  
%}  
%%  
\"(\\.|[^\"]\\|)*\" {printf(\"String constants\\n\");} /** *FOR STRING CONSTANTS */  
\\(\\.|[^\"]\\|)\" {printf(\"Character constants\\n\");} /** *FOR CHARACTER CONSTANTS */  
%%  
  
main(int argc, char **argv) /** *BEGINNING OF THE MAIN FUNCTION */  
{  
yylex();  
}
```

Output:



```
student@SWPC-12: ~/115cs0233/09.01.2018
student@SWPC-12:~/115cs0233/09.01.2018$ flex a4.l
student@SWPC-12:~/115cs0233/09.01.2018$ gcc lex.yy.c -lfl
student@SWPC-12:~/115cs0233/09.01.2018$ ./a.out
"arindum"
String constants
'a'
Character constants
^C
student@SWPC-12:~/115cs0233/09.01.2018$
```

ASSIGNMENT-3

1. Design a machine that will recognize numbers between 0-999. For example: "Two hundred thirty-nine" is a correct form of 239 in words. It should also detect an incorrect input "Twelve hundred twenty-two."

Solution:

```
%{  
    int sum = 0;  
%}  
%%  
"zero"          {sum+=0;}  
"one"           {sum+=1;}  
"two"           {sum+=2;}  
"three"         {sum+=3;}  
"four"          {sum+=4;}  
"five"          {sum+=5;}  
"six"           {sum+=6;}  
"seven"         {sum+=7;}  
"eight"         {sum+=8;}  
"nine"          {sum+=9;}  
"hundred"       {sum*=100;}  
"ten"           {sum+=10;}  
"eleven"        {sum+=11;}  
"twelve"        {sum+=12;}  
"thirteen"     {sum+=13;}  
"fourteen"      {sum+=14;}  
"fifteen"       {sum+=15;}  
"sixteen"       {sum+=16;}  
"seventeen"     {sum+=17;}  
"eighteen"      {sum+=18;}
```



```
"nineteen"      {sum+=19;}
"twenty"        {sum+=20;}
"thirty"        {sum+=30;}
"forty"         {sum+=40;}
"fifty"         {sum+=50;}
"sixty"         {sum+=60;}
"seventy"       {sum+=70;}
"eighty"        {sum+=80;}
"ninety"        {sum+=90;}
%%
```

```
main(int argc, char **argv)
{
    yylex();
    if((sum>=0) && (sum<=999))
        printf("correct%8d\n", sum);
    else
        printf("incorrect%8d\n", sum);
}
```

Output:

```
soumyagourrab@SoumyaGoku: ~/Downloads/115cs0233/16.01.2018
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/16.01.2018$ flex a1.l
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/16.01.2018$ cc lex.yy.c -lfl
a1.l:40:1: warning: return type defaults to 'int' [-Wimplicit-int]
{
^
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/16.01.2018$ ./a.out
two hundred fifty four
correct      254
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/16.01.2018$ ./a.out
eleven hundred fifty four
incorrect    1154
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/16.01.2018$
```

ASSIGNMENT-4

1. Consider the following tokens:

	code	value
begin	1	-
end	2	-
if	3	-
else	4	-
then	5	-
identifier	6	(6,sym_tab_ptr)
constants	7	(7,sym_tab_ptr)
"=="	8	(8,1)
!=	8	(8,2)
>=	8	(8,3)
<=	8	(8,4)
<	8	(8,5)
>	8	(8,6)

Design a DFA that will identify and display the code and values for valid tokens, manage symbol table for identifiers and constants, and find errors.

Solution:

```
#include <stdbool.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
// Returns 'true' if the character is a DELIMITER.
```

```
bool isDelimiter(char ch)
```

```
{
```

```
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
```

```
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
```

```

    ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
    ch == '[' || ch == ']' || ch == '{' || ch == '}')
    return (true);
return (false);
}

```

// Returns 'true' if the character is an OPERATOR.

```

bool isOperator(char ch)
{
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=')
        return (true);
    return (false);
}

```

// Returns 'true' if the string is a VALID IDENTIFIER.

```

bool validIdentifier(char* str)
{
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isDelimiter(str[0]) == true)
        return (false);
    return (true);
}

```

// Returns 'true' if the string is a KEYWORD.

```

bool isKeyword(char* str)

```

```

{
    if (!strcmp(str, "if") || !strcmp(str, "else") ||
        !strcmp(str, "while") || !strcmp(str, "do") ||
        !strcmp(str, "break") ||
        !strcmp(str, "continue") || !strcmp(str, "int")
        || !strcmp(str, "double") || !strcmp(str, "float")
        || !strcmp(str, "return") || !strcmp(str, "char")
        || !strcmp(str, "case") || !strcmp(str, "char")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "short") || !strcmp(str, "typedef")
        || !strcmp(str, "switch") || !strcmp(str, "unsigned")
        || !strcmp(str, "void") || !strcmp(str, "static")
        || !strcmp(str, "struct") || !strcmp(str, "goto"))
        return (true);
    return (false);
}

```

// Returns 'true' if the string is an INTEGER.

```

bool isInteger(char* str)
{
    int i, len = strlen(str);

    if (len == 0)
        return (false);

    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' || (str[i] == '-' && i > 0))

```

```

        return (false);
    }
    return (true);
}

// Returns 'true' if the string is a REAL NUMBER.
bool isRealNumber(char* str)
{
    int i, len = strlen(str);
    bool hasDecimal = false;

    if (len == 0)
        return (false);
    for (i = 0; i < len; i++) {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' && str[i] != '.' ||
            (str[i] == '-' && i > 0))
            return (false);
        if (str[i] == '.')
            hasDecimal = true;
    }
    return (hasDecimal);
}

```

```

// Extracts the SUBSTRING.
char* subString(char* str, int left, int right)
{

```

```

int i;

char* subStr = (char*)malloc(
    sizeof(char) * (right - left + 2));

for (i = left; i <= right; i++)
    subStr[i - left] = str[i];
subStr[right - left + 1] = '\0';
return (subStr);
}

// Parsing the input STRING.
void parse(char* str)
{
    int left = 0, right = 0;
    int len = strlen(str);

    while (right <= len && left <= right) {
        if (isDelimiter(str[right]) == false)
            right++;

        if (isDelimiter(str[right]) == true && left == right) {
            if (isOperator(str[right]) == true)
                printf("'%' IS AN OPERATOR\n", str[right]);

            right++;
            left = right;
        } else if (isDelimiter(str[right]) == true && left != right
            || (right == len && left != right)) {
            char* subStr = subString(str, left, right - 1);

```

```

    if (isKeyword(subStr) == true)

        printf("%s' IS A KEYWORD\n", subStr);

    else if (isInteger(subStr) == true)

        printf("%s' IS AN INTEGER\n", subStr);

    else if (isRealNumber(subStr) == true)

        printf("%s' IS A REAL NUMBER\n", subStr);

    else if (validIdentifier(subStr) == true
        && isDelimiter(str[right - 1]) == false)

        printf("%s' IS A VALID IDENTIFIER\n", subStr);

    else if (validIdentifier(subStr) == false
        && isDelimiter(str[right - 1]) == false)

        printf("%s' IS NOT A VALID IDENTIFIER\n", subStr);

    left = right;
}
}
return;
}

```

// DRIVER FUNCTION

```

int main()
{
    // maximum length of string is 100 here
    char str[100] = "int a = b + 1c; ";

```

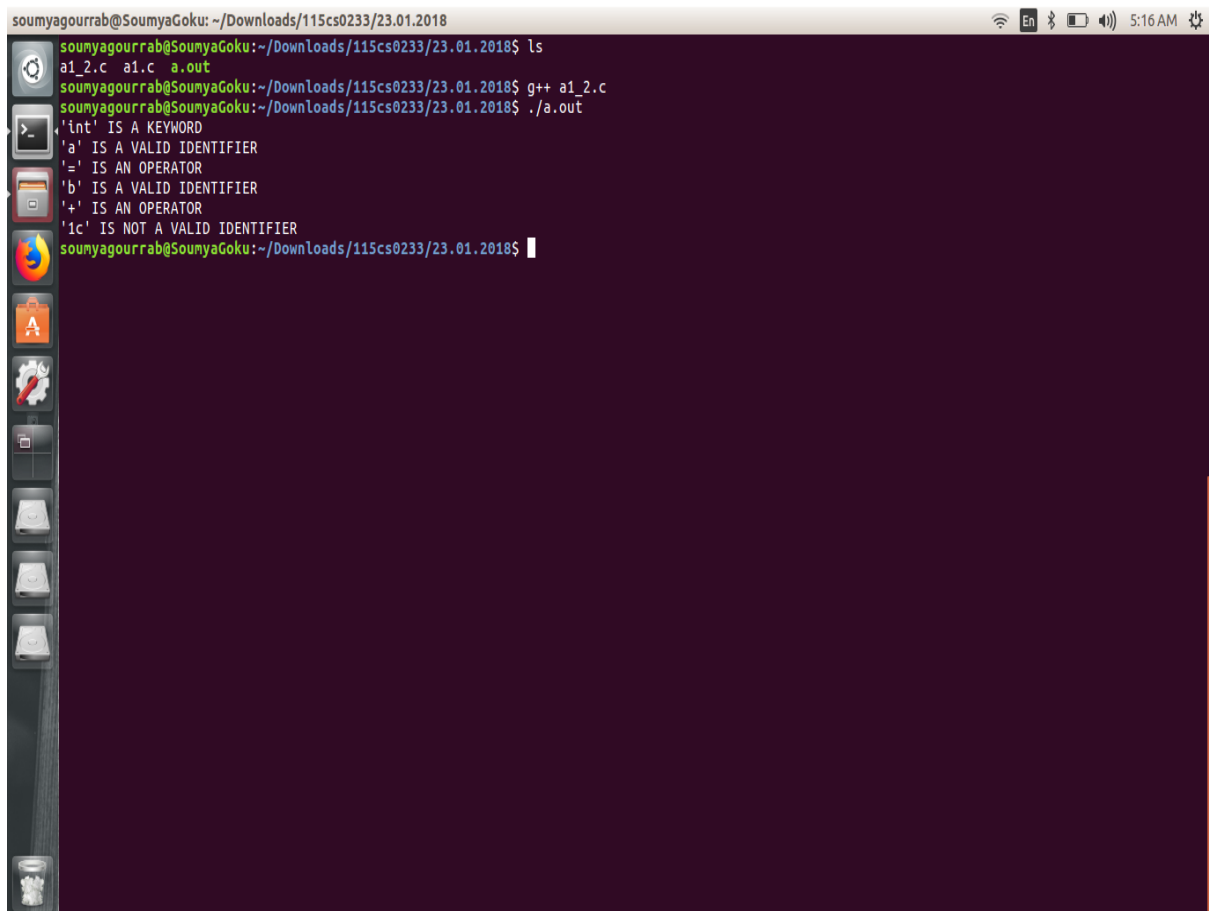


```
parse(str); // calling the parse function
```

```
return (0);
```

```
}
```

Output:



The image shows a terminal window on a Linux desktop. The terminal title is 'soumyagourrab@SoumyaGoku: ~/Downloads/115cs0233/23.01.2018'. The user has run the following commands and received the following output:

```
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/23.01.2018$ ls
a1_2.c  a1.c  a.out
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/23.01.2018$ g++ a1_2.c
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/23.01.2018$ ./a.out
'int' IS A KEYWORD
'a' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
'b' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'1c' IS NOT A VALID IDENTIFIER
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/23.01.2018$
```

The desktop environment includes a sidebar with icons for applications like a file manager, web browser, and terminal, and a top status bar showing system icons and the time 5:16 AM.

ASSIGNMENT-5

1. Design a flex-bison application that will check the correctness of the declaration of variables and functions in C language.

Example:

type id1, id2;

type func1(type param1, type param2);

type= {void, int, char, float} (keywords)

id1, id2, func1, param1, param2 are identifiers.

Token separators or punctuations are , ; ()

Solution:

Flex code:

```
%{  
#include "a1.tab.h"  
#include <string.h>  
extern int yylex(void);  
  
int yylval;  
int count = 0;  
  
char cc[100][100];  
int n = 0;  
int c = 0;  
int i;  
int l;  
int flag=0;  
%}
```

%%

("int")|("float")|("char")|("void") { return D;}

[][a-zA-Z_][a-zA-Z0-9]* {

 c=1;

 for(i=0;i<n;i++){

 if(strcmp(cc[i],yytext)==0){

 printf("Identifier Already Declared\n");

 c=0;

 flag=1;

 }

 }

 if(c==1){

 strcpy(cc[n],yytext);

 n=n+1;

 count++;

 }

 return ID;

}

"(" {return X;}

")" {return Y;}

;" { return E;}

"," { return CO;}

[\n] { return EOL;}

[\t] { }

. { printf("Invalid Identifier\n");

 return I;}

%%

Bison code:

```
%{  
#include <stdio.h>  
int yylex();  
  
void yyerror(const char *s);  
  
extern int flag;  
%}  
  
%token D ID E CO EOL I X Y  
%%  
S:  
| D ID X Z Y E { printf(" The given function declaration is correct \n"); return 0;}  
| D F E EOL { if(flag==0)printf(" The given variable declaration is correct \n"); else printf("The  
given variable declaration is incorrect \n"); return 0;}  
| D EOL {printf("Incorrect\n Incomplete Statement \n"); return 0;}  
| D F EOL {printf("Incorrect\n Incomplete Statement \n" ); return 0;}  
| D IN F IN E EOL  
;  
Z: D ID CO Z { }  
| D ID { }  
| { }  
;  
F: ID CO F { }  
| ID { }  
;  
IN: I { }
```

```

;

%%

main(int argc, char **argv) {

    yyparse();

}

void yyerror(const char *s) {

    printf("%s \n", s);

}

```

Output:

```

soumyagourrab@SoumyaGoku: ~/Downloads/115cs0233/30.01.2018
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ ls
a1.l  a1.tab.c  a1.tab.h  a1.y  a.out  lex.yy.c
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ bison -d a1.y
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ flex a1.l
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ gcc a1.tab.c lex.yy.c -lfl
a1.y:31:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(int argc, char **argv) {
^
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ ./a.out
int a = 5 ;
=5 The given variable declaration is correct
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ ./a.out
void main ( int a ) ;
The given function declaration is correct
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ ./a.out
int main ( ) ;
The given function declaration is correct
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$ ./a.out
int main (void abc , , ) ;
syntax error
soumyagourrab@SoumyaGoku:~/Downloads/115cs0233/30.01.2018$

```