COMPILER CONSTRUCTION

GROUP ASSIGNMENT 2

<u>GROUP MEMBERS</u>

NAME	*REGISTRATION NUMBER*
YVONNE KIMANI	SCT212-0475/2017
MARK MUNENE	SCT 212-0224-2017
GABRIEL WAINAINA MWANGI	SCT212-0480/2017
IAN MWANGI	SCT212-0066/2017
STANLEY NGUGI	SCT212-0065/2017
DENNIS GACHOMO	SCT212-9218/2015

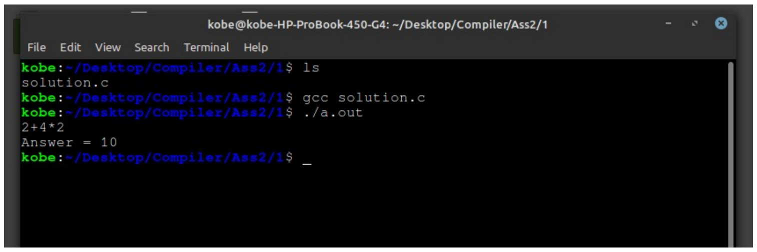
1. Using the Recursive Descent strategy, write a C program for a simple calculator that can be used to perform integer arithmetic involving '+' and '*'. Let your program consist of a set of mutually recursive routines.

Solution: ***(lex file name: solution.l)***

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

/*variable to store the expected*/
char token;

int E(void);
int T(void);
int F(void);

/* report error and die */
void error(void)
{ fprintf(stderr,"Error\n");
  exit(1);
}

/* match input token, read next token */
void match(char expectedToken)
{ if (token==expectedToken)
          token = getchar();
  else
          error();
}

/* process an expression */
int E(void)
{
          int temp = T();
          while (token=='+')
                    switch (token)
                    {
                              case '+':
                                match('+');
                                temp += T();
                                break;
                    }
          return temp;
}

/* process a term */
```

```
int T(void)
{ int temp = F();
  while (token=='*')
  {
          match('*');
          temp *= F();
          }
  return temp;
}


/* process a factor */
int F(void)
{
 int temp = 0;
  if (token=='(')
  {
           match('(');
          temp = E();
          match(')');
}
  else if (isdigit(token))
  { ungetc(token,stdin);
    scanf("%d",&temp);
    token = getchar();
  }
  else
  {
          error();
  }
  return temp;
}

/* calculator driver program */
int main()
{
          int answer;
  token = getchar();
  answer = E();
  if (token=='\n') printf("Answer = %d\n",answer);
  else error();
  return 0;
}
```

<u>Sample run output:</u>



```
kobe@kobe-HP-ProBook-450-G4: ~/Desktop/Compiler/Ass2/1
kobe:~/Desktop/Compiler/Ass2/1$ ls
solution.c
kobe:~/Desktop/Compiler/Ass2/1$ gcc solution.c
kobe:~/Desktop/Compiler/Ass2/1$ ./a.out
2+4*2
Answer = 10
kobe:~/Desktop/Compiler/Ass2/1$ _
```

2. Using the following grammar

S □ a S | b

Create an interpreter using LEX and YACC which will count the number of a's in the input string.

<u>Solution</u> (2 files: lex file, yacc file)

<u>Lex file source code: (name solution.l)</u>

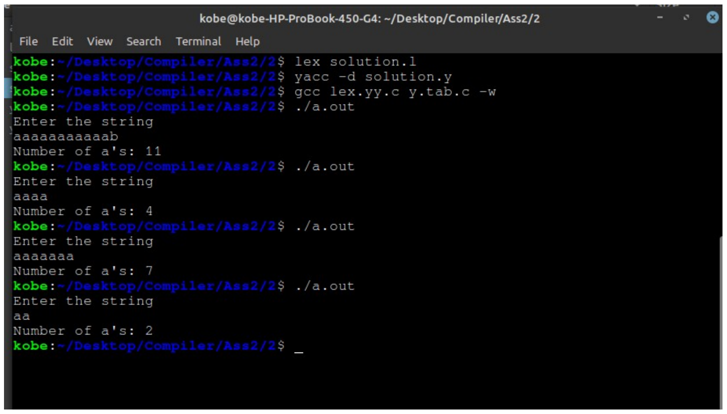
```
%{
#include <stdio.h>
#include "y.tab.h"
%}

%%
a       return *yytext;
b       return *yytext;
[\n]    return NEWLINE;
%%

int yywrap()
{
        return 1;
}
```

<u>Yacc file source code: (name solution.y)</u>

```
%{
#include <stdio.h>
int count = 0;
int yylex();
%}

%token NEWLINE

%%
start : S NEWLINE { return; }
```

```
          ;

S:        'a' S    { count++; }
          | 'b'    {}
          |
          ;
%%

int yyerror(char const *s)
{
          printf("yyerror %s\n", s);
          exit(1);
}

int main()
{
          printf("Enter the string\n");
          yyparse();
          printf("Number of a\'s: %d\n", count);
          return 1;
}
```

Sample run output:


kobe@kobe-HP-ProBook-450-G4: ~/Desktop/Compiler/Ass2/2
File Edit View Search Terminal Help

```
kobe:~/Desktop/Compiler/Ass2/2$ lex solution.l
kobe:~/Desktop/Compiler/Ass2/2$ yacc -d solution.y
kobe:~/Desktop/Compiler/Ass2/2$ gcc lex.yy.c y.tab.c -w
kobe:~/Desktop/Compiler/Ass2/2$ ./a.out
Enter the string
aaaaaaaaaaab
Number of a's: 11
kobe:~/Desktop/Compiler/Ass2/2$ ./a.out
Enter the string
aaaa
Number of a's: 4
kobe:~/Desktop/Compiler/Ass2/2$ ./a.out
Enter the string
aaaaaaa
Number of a's: 7
kobe:~/Desktop/Compiler/Ass2/2$ ./a.out
Enter the string
aa
Number of a's: 2
kobe:~/Desktop/Compiler/Ass2/2$ _
```

3. Write a LEX and YACC specification files for a small calculator that can add and subtract numbers.

Solution:

Lex file source code: (name solution.l)

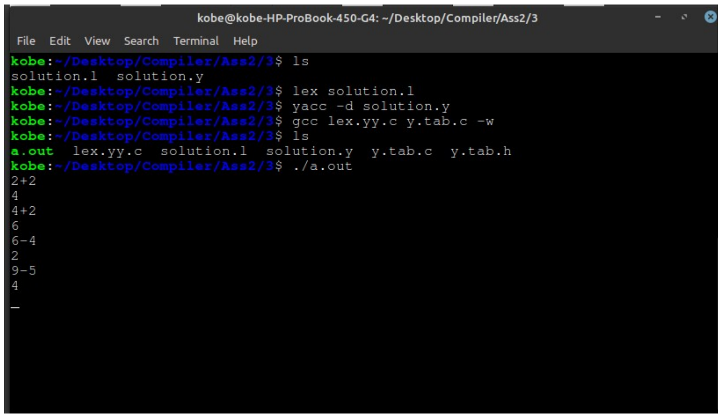
```
%{
        #include <stdlib.h>
        void yyerror(char *);
        #include "y.tab.h"
%}

%%
        /* integers */
[0-9]+ { yylval = atoi(yytext); return INTEGER;}

        /* operators */
[-+()=\n] { return *yytext; }
        /* skip whitespace */
[ \t]   ;

        /* anything else is an error */
.       yyerror("invalid character");

%%

int yywrap(void) {
        return 1;
}
```

Yacc file source code: (name solution.y)

```
%{
//c definitions
        #include<stdio.h>
        #include<stdlib.h>
        void yyerror(char *);
        int yylex(void);

%}
//yacc definitions
%token INTEGER
%left '+' '-'

%%//productions
```

```
program:
        program statement '\n'
        |
        ;
statement:
        expr                { printf("%d\n", $1); }
        ;


expr:
        INTEGER

        | expr '+' expr             { $$ = $1 + $3; }
        | expr '-' expr             { $$ = $1 - $3; }
        | '(' expr ')'          { $$ = $2; }
        ;

%%

void yyerror(char *s) {
        fprintf(stderr, "%s\n", s);
        exit(1);
}
int main(void) {
        yyparse();
        return 0;
}
```

Sample run output:



kobe@kobe-HP-ProBook-450-G4: ~/Desktop/Compiler/Ass2/3
File Edit View Search Terminal Help

```
kobe:~/Desktop/Compiler/Ass2/3$ ls
solution.l  solution.y
kobe:~/Desktop/Compiler/Ass2/3$ lex solution.l
kobe:~/Desktop/Compiler/Ass2/3$ yacc -d solution.y
kobe:~/Desktop/Compiler/Ass2/3$ gcc lex.yy.c y.tab.c -w
kobe:~/Desktop/Compiler/Ass2/3$ ls
a.out  lex.yy.c  solution.l  solution.y  y.tab.c  y.tab.h
kobe:~/Desktop/Compiler/Ass2/3$ ./a.out
2+2
4
4+2
6
6-4
2
9-5
4
_
```