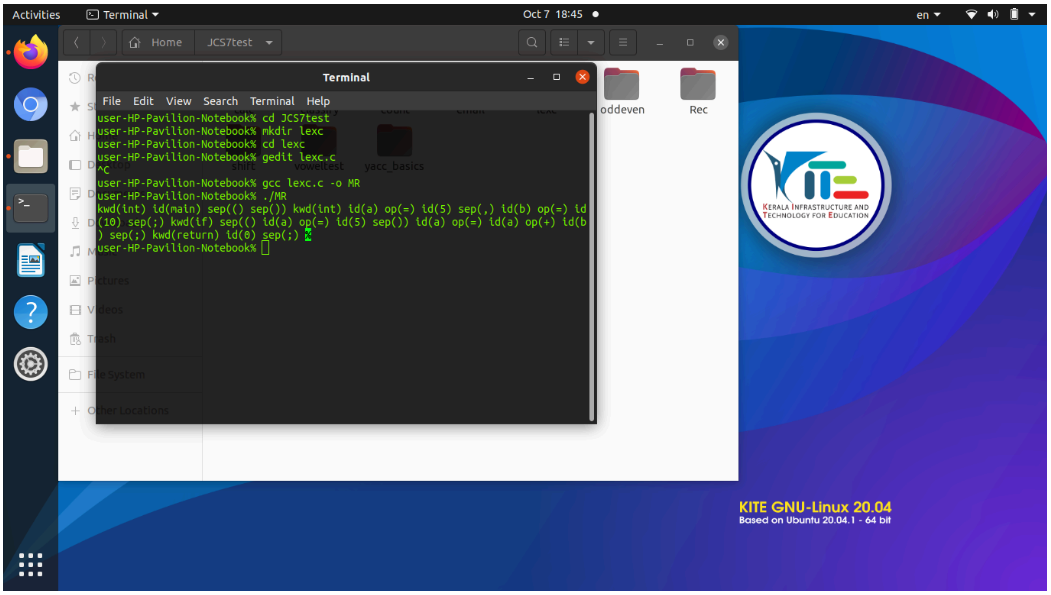
Experiment 1 Lexical Analyzer using C

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Check if a word is a keyword
int isKeyword(char word[]) {
        char *keywords[] = {"int", "if", "while", "return", "else", "for"};
        for (int i = 0; i < 6; i++) {
        if (strcmp(word, keywords[i]) == 0)
        return 1;
        }return 0;
}

int main() {
        FILE *fp;
        char ch, word[20];
        int i = 0;

        fp = fopen("input.txt", "r");
        if (fp == NULL) {
        printf("Could not open input.txt\n");
        return 1;
        }

        while ((ch = fgetc(fp)) != EOF) {
        if (isalnum(ch)) {
        word[i++] = ch;  // build word
        } else {
        if (i > 0) {
                word[i] = '\0';  // end string
                if (isKeyword(word))
                printf("kwd(%s) ", word);
                else
                printf("id(%s) ", word);
                i = 0;
        }

        // Check single-character tokens
        if (ch == '+' || ch == '-' || ch == '=')
                printf("op(%c) ", ch);
```

```
            else if (ch == ';' || ch == ',' || ch == '(' || ch == ')')
                printf("sep(%c) ", ch);
            }
            }

            fclose(fp);
            return 0;
}
```



Activities
Terminal
Oct 7 18:45
en
Home
JCS7test
Terminal
File Edit View Search Terminal Help
oddeven
Rec
KERALA INFRASTRUCTURE AND TECHNOLOGY FOR EDUCATION
KITE GNU-Linux 20.04
Based on Ubuntu 20.04.1 - 64 bit

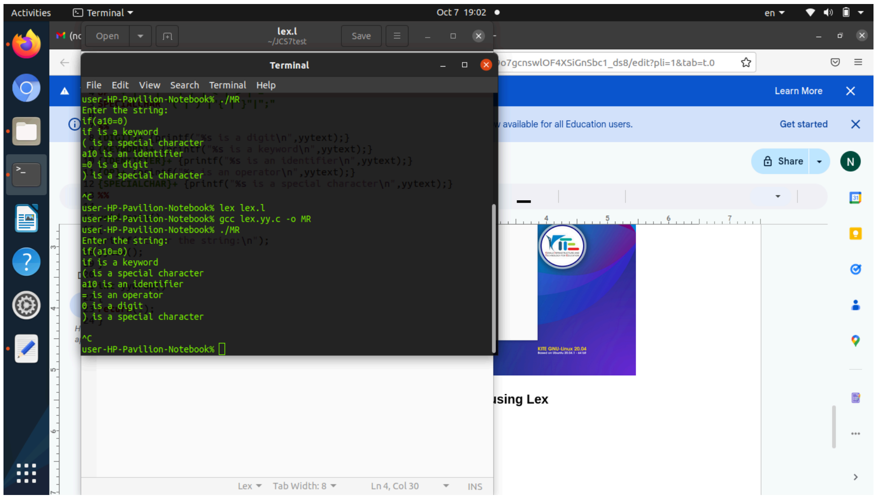
```
user-HP-Pavilion-Notebook% cd JCS7test
user-HP-Pavilion-Notebook% mkdir lexc
user-HP-Pavilion-Notebook% cd lexc
user-HP-Pavilion-Notebook% gedit lexc.c
^C
user-HP-Pavilion-Notebook% gcc lexc.c -o MR
user-HP-Pavilion-Notebook% ./MR
kwd(int) id(main) sep(() sep()) kwd(int) id(a) op(=) id(5) sep(,) id(b) op(=) id
(10) sep(;) kwd(if) sep(() id(a) op(=) id(5) sep()) id(a) op(=) id(a) op(+) id(b
) sep(;) kwd(return) id(0) sep(;) %
user-HP-Pavilion-Notebook% 
```

Experiment 2 Lexical Analyzer using Lex

Code:

```
DIGIT[0-9]
IDENTIFIER[a-zA-Z][a-zA-Z0-9]*
KEYWORD "if"|"else"|"int"|"while"|"main"
OP "+"|"-"|"*"|"/"|"<"|">"|"="
SPECIALCHAR "("|")"|"{"|"}"|";"

%%
{DIGIT}+ {printf("%s is a digit\n",yytext);}
{KEYWORD}+ {printf("%s is a keyword\n",yytext);}
{IDENTIFIER}+ {printf("%s is an identifier\n",yytext);}
{OP}+ {printf("%s is an operator\n",yytext);}
{SPECIALCHAR}+ {printf("%s is a special character\n",yytext);}
%%
```

```
void main()
{
printf("Enter the string:\n");
yylex();
}

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



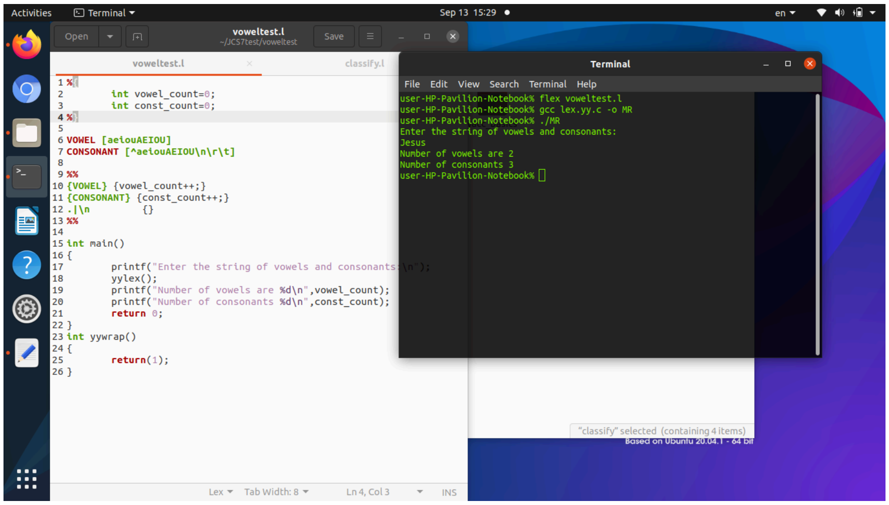
```
user-HP-Pavilion-Notebook% ./MR
Enter the string:
if(a10=0)
if is a keyword
( is a special character
a10 is an identifier
=0 is a digit
) is a special character

^C
user-HP-Pavilion-Notebook% lex lex.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter the string:
if(a10=0)
if is a keyword
( is a special character
a10 is an identifier
= is an operator
0 is a digit
) is a special character

^C
user-HP-Pavilion-Notebook% 
```

Experiment 3(a) Count number of vowels and consonants using Lex

Code:

```
%{
    int vowel_count=0;
    int const_count=0;
%}

VOWEL [aeiouAEIOU]
CONSONANT [^aeiouAEIOU\n\r\t]

%%
{VOWEL} {vowel_count++;}
{CONSONANT} {const_count++;}
.|\n        {}
```

```
%%

int main()
{
   printf("Enter the string of vowels and consonants:\n");
   yylex();
   printf("Number of vowels are %d\n",vowel_count);
   printf("Number of consonants %d\n",const_count);
   return 0;
}
int yywrap()
{
   return(1);
}
```



```
%{
        int vowel_count=0;
        int const_count=0;
%}

VOWEL [aeiouAEIOU]
CONSONANT [^aeiouAEIOU\n\r\t]

%%
{VOWEL} {vowel_count++;}
{CONSONANT} {const_count++;}
.|\n           {}
%%

int main()
{
        printf("Enter the string of vowels and consonants:\n");
        yylex();
        printf("Number of vowels are %d\n",vowel_count);
        printf("Number of consonants %d\n",const_count);
        return 0;
}
int yywrap()
{
        return(1);
}
```

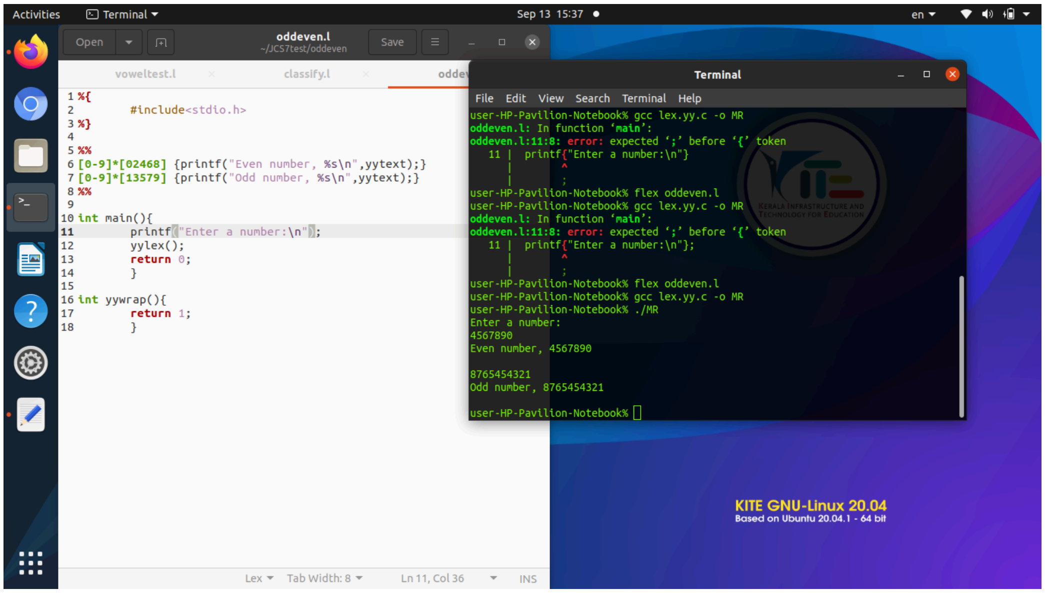
```
user-HP-Pavilion-Notebook% flex voweltest.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter the string of vowels and consonants:
Jesus
Number of vowels are 2
Number of consonants 3
user-HP-Pavilion-Notebook% 
```

Experiment 3(b) Check if a number is odd or even using Lex

Code:

```
%{
   #include<stdio.h>
%}

%%
[0-9]*[02468] {printf("Even number, %s\n",yytext);}
[0-9]*[13579] {printf("Odd number, %s\n",yytext);}
%%

int main(){
   printf("Enter a number:\n");
   yylex();
```

```
    return 0;
    }

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



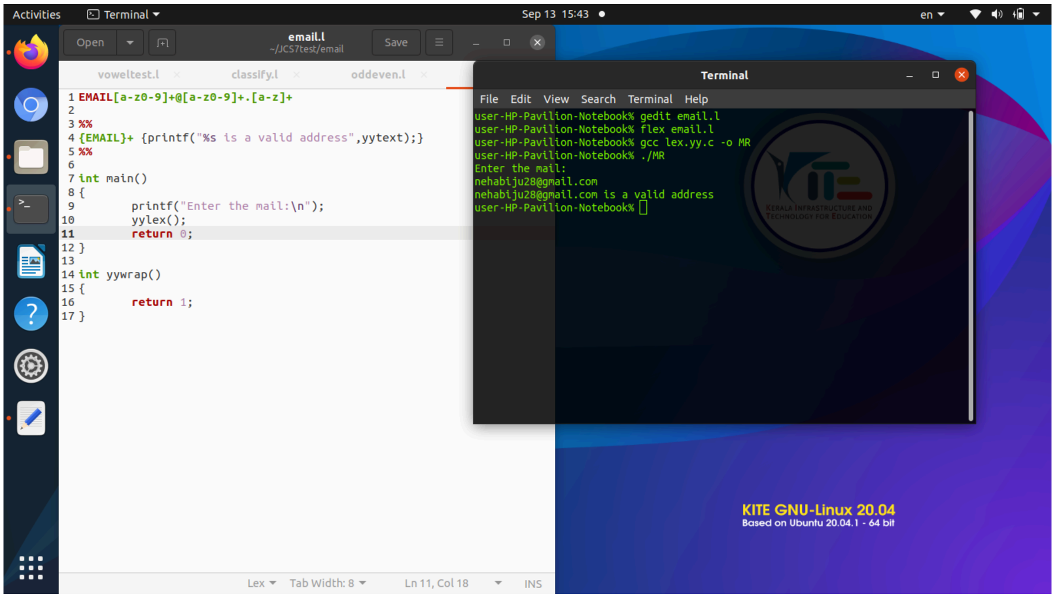
```
%{
        #include<stdio.h>
%}

%%
[0-9]*[02468] {printf("Even number, %s\n",yytext);}
[0-9]*[13579] {printf("Odd number, %s\n",yytext);}
%%

int main(){
        printf("Enter a number:\n");
        yylex();
        return 0;
        }

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

```
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
oddeven.l: In function ‘main’:
oddeven.l:11:8: error: expected ‘;’ before ‘{’ token
   11 |  printf{"Enter a number:\n"}
      |        ^
      |        ;
user-HP-Pavilion-Notebook% flex oddeven.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
oddeven.l: In function ‘main’:
oddeven.l:11:8: error: expected ‘;’ before ‘{’ token
   11 |  printf{"Enter a number:\n"};
      |        ^
      |        ;
user-HP-Pavilion-Notebook% flex oddeven.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter a number:
4567890
Even number, 4567890

8765454321
Odd number, 8765454321

user-HP-Pavilion-Notebook% 
```

Experiment 3(c) Check if an email is valid using Lex

Code:

```
EMAIL[a-z0-9]+@[a-z0-9]+.[a-z]+

%%
{EMAIL}+ {printf("%s is a valid address",yytext);}
%%

int main()
{
    printf("Enter the mail:\n");
    yylex();
    return 0;
}

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



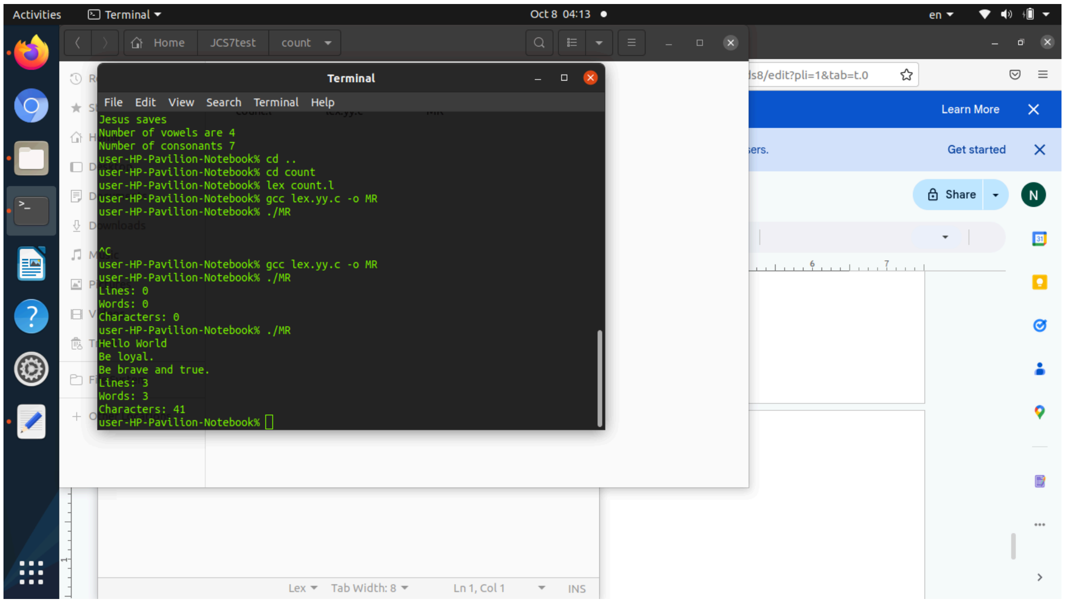
```
EMAIL[a-z0-9]+@[a-z0-9]+.[a-z]+

%%
{EMAIL}+ {printf("%s is a valid address",yytext);}
%%

int main()
{
        printf("Enter the mail:\n");
        yylex();
        return 0;
}

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

```
user-HP-Pavilion-Notebook% gedit email.l
user-HP-Pavilion-Notebook% flex email.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter the mail:
nehabiju28@gmail.com
nehabiju28@gmail.com is a valid address
user-HP-Pavilion-Notebook%
```

Experiment 4(a) Count number of lines,words,characters using Lex

Code:

```
%{
   #include<stdio.h>
   int lines=0;
   int words=0;
   int characters=0;
%}

%%
\n    {lines++;characters++;}
[\t]+   {characters+=yyleng;}
[^\t\n]+ {words++;characters+=yyleng;}
%%

int main(int argc, char **argv)
{
   yylex();
   printf("Lines: %d\n",lines);
   printf("Words: %d\n",words);
   printf("Characters: %d\n",characters);
   return 0;
}

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



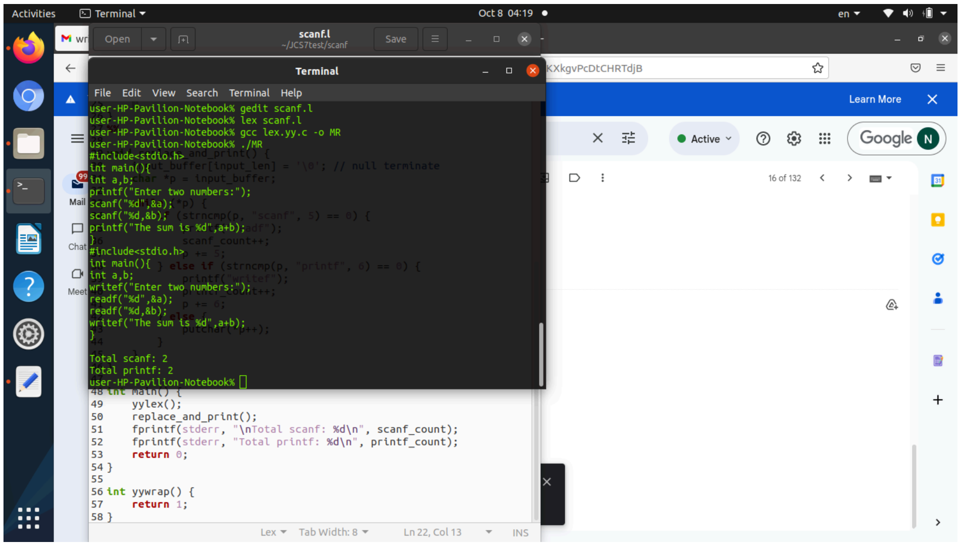
```
Jesus saves
Number of vowels are 4
Number of consonants 7
user-HP-Pavilion-Notebook% cd ..
user-HP-Pavilion-Notebook% cd count
user-HP-Pavilion-Notebook% lex count.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR


^C
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
Lines: 0
Words: 0
Characters: 0
user-HP-Pavilion-Notebook% ./MR
Hello World
Be loyal.
Be brave and true.
Lines: 3
Words: 3
Characters: 41
user-HP-Pavilion-Notebook% 
```

Experiment 4(b) Count and replace scanf and printf using Lex

Code:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_INPUT_SIZE 1000000  // 1 MB max input size for simplicity

char input_buffer[MAX_INPUT_SIZE];
int input_len = 0;

int scanf_count = 0;
int printf_count = 0;

void replace_and_print();
%}

%%
.|\n      {
          if (input_len + yyleng < MAX_INPUT_SIZE) {
```

```
    memcpy(input_buffer + input_len, yytext, yyleng);
    input_len += yyleng;
    } else {
    fprintf(stderr, "Input too large!\n");
    exit(1);
    }
}
%%

void replace_and_print() {
    input_buffer[input_len] = '\0'; // null terminate
    char *p = input_buffer;

    while (*p) {
    if (strncmp(p, "scanf", 5) == 0) {
    printf("readf");
    scanf_count++;
    p += 5;
    } else if (strncmp(p, "printf", 6) == 0) {
    printf("writef");
    printf_count++;
    p += 6;
    } else {
    putchar(*p++);
    }
    }
}

int main() {
    yylex();
    replace_and_print();
    fprintf(stderr, "\nTotal scanf: %d\n", scanf_count);
    fprintf(stderr, "Total printf: %d\n", printf_count);
    return 0;
}

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



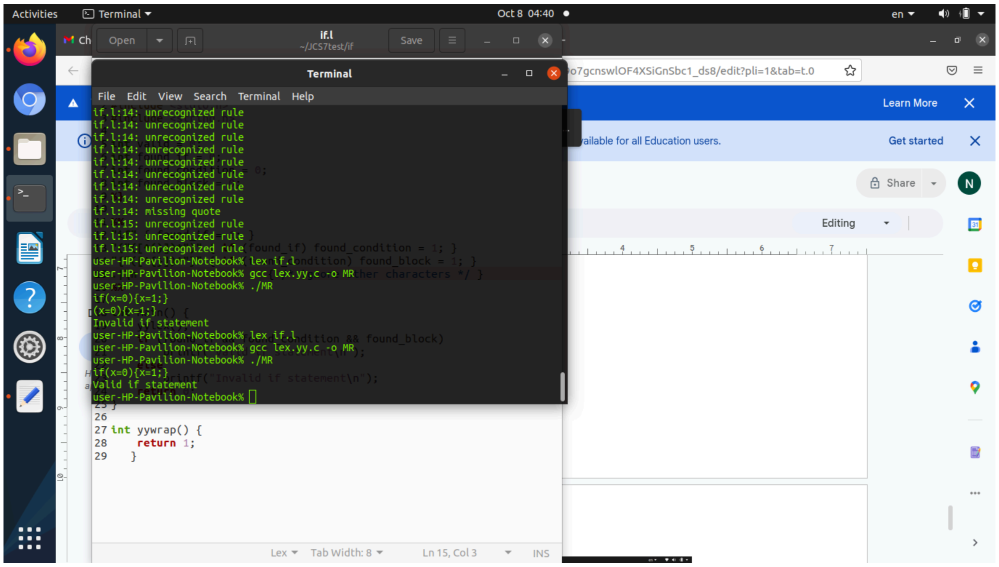
```
user-HP-Pavilion-Notebook% gedit scanf.l
user-HP-Pavilion-Notebook% lex scanf.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
#include<stdio.h>
int main(){
int a,b;
printf("Enter two numbers:");
scanf("%d",&a);
scanf("%d,&b);
printf("The sum is %d",a+b);
}
#include<stdio.h>
int main(){
int a,b;
writef("Enter two numbers:");
readf("%d",&a);
readf("%d,&b);
writef("The sum is %d",a+b);
}

Total scanf: 2
Total printf: 2
user-HP-Pavilion-Notebook% 
```

```
int main() {
    yylex();
    replace_and_print();
    fprintf(stderr, "\nTotal scanf: %d\n", scanf_count);
    fprintf(stderr, "Total printf: %d\n", printf_count);
    return 0;
}

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

Experiment 4(c) Validity of if using Lex

Code:

```
%{
#include <stdio.h>
#include <string.h>

int valid = 0;
int found_if = 0;
int found_condition = 0;
int found_block = 0;
%}

%%
"if" { found_if = 1; }
"("([^()])*")" { if (found_if) found_condition = 1; }
"{"([^{}])*"}" { if (found_condition) found_block = 1; }
.|\n                    { /* ignore other characters */ }
%%

int main() {
        yylex();
        if (found_if && found_condition && found_block)
        printf("Valid if statement\n");
        else
        printf("Invalid if statement\n");
        return 0;
}

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



```
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: unrecognized rule
if.l:14: missing quote
if.l:15: unrecognized rule
if.l:15: unrecognized rule
if.l:15: unrecognized rule
user-HP-Pavilion-Notebook% lex if.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
if(x=0){x=1;}
(x=0){x=1;}
Invalid if statement
user-HP-Pavilion-Notebook% lex if.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
if(x=0){x=1;}
Valid if statement
user-HP-Pavilion-Notebook% 
```

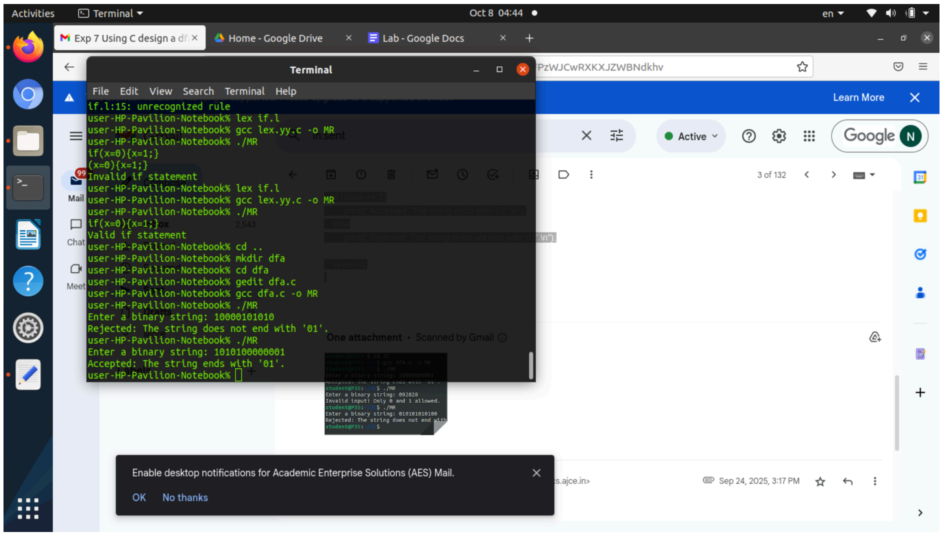
```
int yywrap() {
    return 1;
   }
```

Experiment 5 DFA accepting {0,1} with ending 01

Code:

```
#include <stdio.h>

int main() {
        char input[100];
        int state = 0;  // 0 = q0, 1 = q1, 2 = q2 (accept)
        int i = 0;

        printf("Enter a binary string: ");
        scanf("%s", input);

        while (input[i] != '\0') {
        if (input[i] != '0' && input[i] != '1') {
        printf("Invalid input! Only 0 and 1 allowed.\n");
        return 1;
        }

        switch (state) {
        case 0:
```

```
        if (input[i] == '0')
        state = 1;
        else
        state = 0;
        break;

    case 1:
        if (input[i] == '0')
        state = 1;
        else
        state = 2;
        break;

    case 2:
        if (input[i] == '0')
        state = 1;
        else
        state = 0;
        break;
    }

    i++;
    }

    if (state == 2)
    printf("Accepted: The string ends with '01'.\n");
    else
    printf("Rejected: The string does not end with '01'.\n");

    return 0;
}
```



```
if.l:15: unrecognized rule
user-HP-Pavilion-Notebook% lex if.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
if(x=0){x=1;}
(x=0){x=1;}
Invalid if statement
user-HP-Pavilion-Notebook% lex if.l
user-HP-Pavilion-Notebook% gcc lex.yy.c -o MR
user-HP-Pavilion-Notebook% ./MR
if(x=0){x=1;}
Valid if statement
user-HP-Pavilion-Notebook% cd ..
user-HP-Pavilion-Notebook% mkdir dfa
user-HP-Pavilion-Notebook% cd dfa
user-HP-Pavilion-Notebook% gedit dfa.c
user-HP-Pavilion-Notebook% gcc dfa.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter a binary string: 10000101010
Rejected: The string does not end with '01'.
user-HP-Pavilion-Notebook% ./MR
Enter a binary string: 1010100000001
Accepted: The string ends with '01'.
user-HP-Pavilion-Notebook% 
```

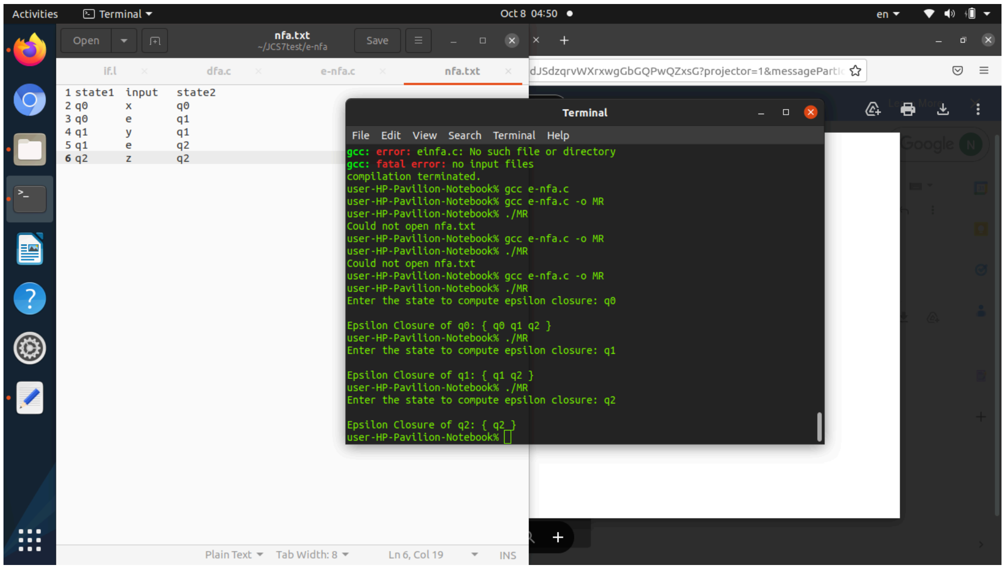
Enable desktop notifications for Academic Enterprise Solutions (AES) Mail.

Experiment 6 e-closure of an NFA

Code:

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

#define MAX 100

char transitions[MAX][3][20]; // transitions[i][0]=state1, [1]=input, [2]=state2
int t_count = 0;
char closure[20][20]; // store epsilon closure states
int closure_count = 0;

// Check if state already in closure
int is_in_closure(const char *state) {
        for (int i = 0; i < closure_count; i++) {
        if (strcmp(closure[i], state) == 0)
        return 1;
        }
        return 0;
}

// Add state to closure if not already present
void add_to_closure(const char *state) {
        if (!is_in_closure(state)) {
        strcpy(closure[closure_count++], state);
        }
}

// Recursively find epsilon closure
void find_epsilon_closure(const char *state) {
        add_to_closure(state);
        for (int i = 0; i < t_count; i++) {
        if (strcmp(transitions[i][0], state) == 0 && strcmp(transitions[i][1], "e") == 0) {
        if (!is_in_closure(transitions[i][2])) {
            find_epsilon_closure(transitions[i][2]);
        }
        }
        }
}

int main() {
        FILE *fp = fopen("nfa.txt", "r");
        if (fp == NULL) {
        printf("Could not open nfa.txt\n");
        return 1;
        }

        // Read transitions from file
        while (fscanf(fp, "%s %s %s", transitions[t_count][0], transitions[t_count][1],
transitions[t_count][2]) == 3) {
        t_count++;
        }

        fclose(fp);

        char start_state[20];
        printf("Enter the state to compute epsilon closure: ");
        scanf("%s", start_state);

        closure_count = 0; // reset global closure count
        find_epsilon_closure(start_state);

        printf("\nEpsilon Closure of %s: { ", start_state);
        for (int i = 0; i < closure_count; i++) {
        printf("%s ", closure[i]);
        }
        printf("}\n");
```

```
        return 0;
}
```



```
state1  input   state2
q0      x       q0
q0      e       q1
q1      y       q1
q1      e       q2
q2      z       q2
```

```
gcc: error: einfa.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
user-HP-Pavilion-Notebook% gcc e-nfa.c
user-HP-Pavilion-Notebook% gcc e-nfa.c -o MR
user-HP-Pavilion-Notebook% ./MR
Could not open nfa.txt
user-HP-Pavilion-Notebook% gcc e-nfa.c -o MR
user-HP-Pavilion-Notebook% ./MR
Could not open nfa.txt
user-HP-Pavilion-Notebook% gcc e-nfa.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter the state to compute epsilon closure: q0

Epsilon Closure of q0: { q0 q1 q2 }
user-HP-Pavilion-Notebook% ./MR
Enter the state to compute epsilon closure: q1

Epsilon Closure of q1: { q1 q2 }
user-HP-Pavilion-Notebook% ./MR
Enter the state to compute epsilon closure: q2

Epsilon Closure of q2: { q2 }
user-HP-Pavilion-Notebook% 
```

Experiment 7 Valid Identifier using YACC

Code:

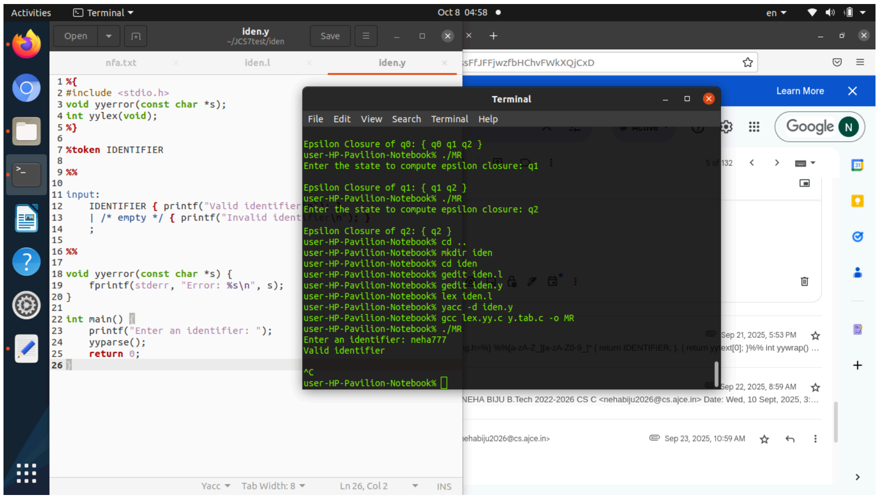
Iden.l

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

%%
[a-zA-Z_][a-zA-Z0-9_]*          { return IDENTIFIER; }
.                        { return yytext[0]; }
%%

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

Iden.y

```
%{
```

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

%token IDENTIFIER

%%

input:
        IDENTIFIER { printf("Valid identifier\n"); }
        | /* empty */ { printf("Invalid identifier\n"); }
        ;

%%

void yyerror(const char *s) {
        fprintf(stderr, "Error: %s\n", s);
}

int main() {
        printf("Enter an identifier: ");
        yyparse();
        return 0;
}
```



```
Epsilon Closure of q0: { q0 q1 q2 }
user-HP-Pavilion-Notebook% ./MR
Enter the state to compute epsilon closure: q1

Epsilon Closure of q1: { q1 q2 }
user-HP-Pavilion-Notebook% ./MR
Enter the state to compute epsilon closure: q2

Epsilon Closure of q2: { q2 }
user-HP-Pavilion-Notebook% cd ..
user-HP-Pavilion-Notebook% mkdir iden
user-HP-Pavilion-Notebook% cd iden
user-HP-Pavilion-Notebook% gedit iden.l
user-HP-Pavilion-Notebook% gedit iden.y
user-HP-Pavilion-Notebook% lex iden.l
user-HP-Pavilion-Notebook% yacc -d iden.y
user-HP-Pavilion-Notebook% gcc lex.yy.c y.tab.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter an identifier: neha777
Valid identifier

^C
user-HP-Pavilion-Notebook% 
```

Experiment 8 Calculator using YACC

Code:

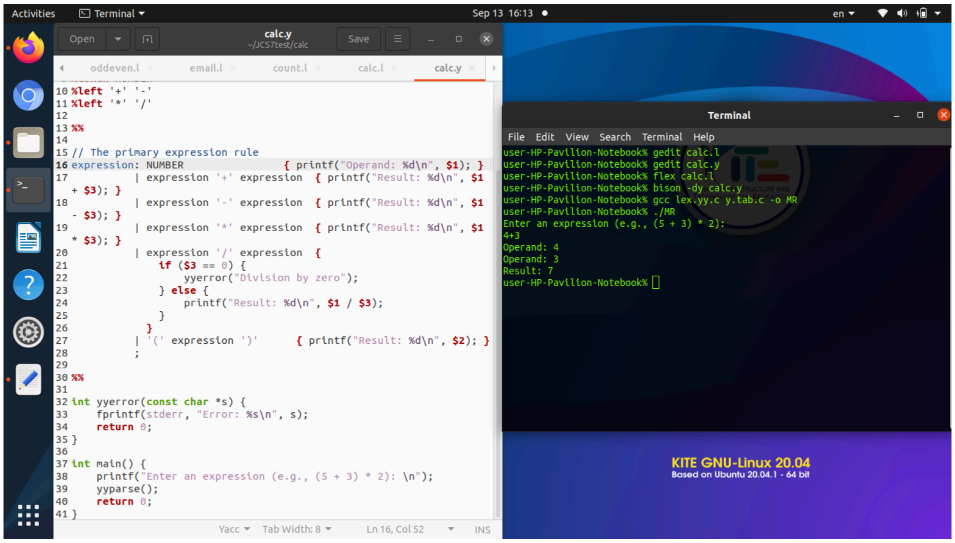
Calc.l

```
%{
#include "y.tab.h"
%}

%%

[0-9]+          { yylval = atoi(yytext); return NUMBER; }
[ \t]      ;
\n         { return 0; }
.          { return yytext[0]; }

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

Calc.y

```
%{
#include <stdio.h>
#include <stdlib.h>

extern int yylex();
extern int yyerror(const char *s);
%}

%token NUMBER
%left '+' '-'
%left '*' '/'

%%

// The primary expression rule
expression: NUMBER          { printf("Operand: %d\n", $1); }
          | expression '+' expression  { printf("Result: %d\n", $1 + $3); }
          | expression '-' expression  { printf("Result: %d\n", $1 - $3); }
          | expression '*' expression  { printf("Result: %d\n", $1 * $3); }
          | expression '/' expression  {
          if ($3 == 0) {
                    yyerror("Division by zero");
          } else {
```

```
                        printf("Result: %d\n", $1 / $3);
                }
            }
            | '(' expression ')'        { printf("Result: %d\n", $2); }
            ;

%%

int yyerror(const char *s) {
            fprintf(stderr, "Error: %s\n", s);
            return 0;
}

int main() {
            printf("Enter an expression (e.g., (5 + 3) * 2): \n");
            yyparse();
            return 0;
}
```



```
%left '+' '-'
%left '*' '/'

%%

// The primary expression rule
expression: NUMBER                          { printf("Operand: %d\n", $1); }
          | expression '+' expression  { printf("Result: %d\n", $1 + $3); }
          | expression '-' expression  { printf("Result: %d\n", $1 - $3); }
          | expression '*' expression  { printf("Result: %d\n", $1 * $3); }
          | expression '/' expression  {
              if ($3 == 0) {
                  yyerror("Division by zero");
              } else {
                  printf("Result: %d\n", $1 / $3);
              }
            }
          | '(' expression ')'          { printf("Result: %d\n", $2); }
          ;

%%

int yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
    return 0;
}

int main() {
    printf("Enter an expression (e.g., (5 + 3) * 2): \n");
    yyparse();
    return 0;
}
```

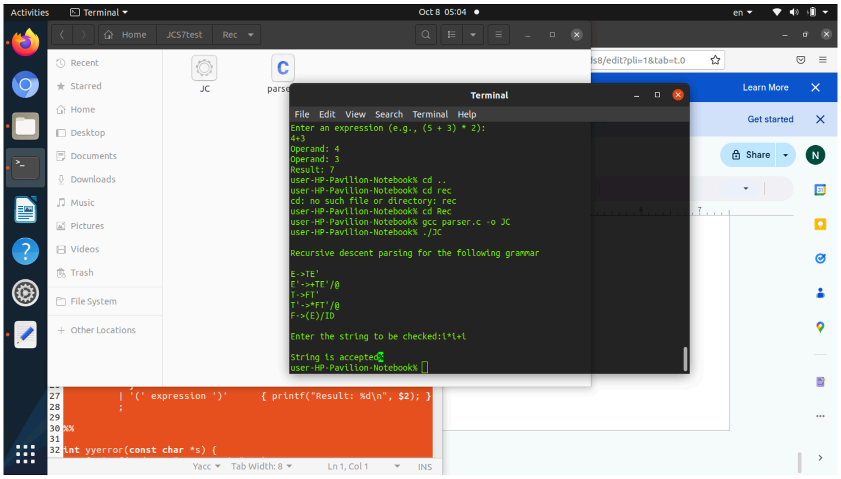
```
user-HP-Pavilion-Notebook% gedit calc.l
user-HP-Pavilion-Notebook% gedit calc.y
user-HP-Pavilion-Notebook% flex calc.l
user-HP-Pavilion-Notebook% bison -dy calc.y
user-HP-Pavilion-Notebook% gcc lex.yy.c y.tab.c -o MR
user-HP-Pavilion-Notebook% ./MR
Enter an expression (e.g., (5 + 3) * 2):
4+3
Operand: 4
Operand: 3
Result: 7
user-HP-Pavilion-Notebook%
```

Experiment 9 Recursive Descent Parser

Code:

```
// C program to Construct of recursive descent parsing for
// the following grammar
// E->TE’
// E’->+TE/@
// T->FT’
// T`->*FT’/@
// F->(E)/id where @ represents null character
```

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
char input[100];
int i, l;

int E();
int EP();
int T();
int TP();
int F();


void main()
{
        printf("\nRecursive descent parsing for the following grammar\n");
        printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n");
        printf("\nEnter the string to be checked:");
        scanf("%s", input);
        if (E())
        {
        if (input[i + 1] == '\0')
        printf("\nString is accepted");
        else
        printf("\nString is not accepted");
        }
        else
        printf("\nString not accepted");
}
int E()
{
        if (T())
        {
        if (EP())
        return (1);
        else
        return (0);
        }
        else
        return (0);
}
int EP()
{
        if (input[i] == '+')
        {
        i++;
        if (T())
        {
        if (EP())
                return (1);
        else
                return (0);
        }
        else
        return (0);
        }
        else
        return (1);
}
int T()
{
        if (F())
        {
        if (TP())
        return (1);
        else
        return (0);
        }
        else
        return (0);
}
int TP()
{
        if (input[i] == '*')
        {
        i++;
        if (F())
        {
        if (TP())
                return (1);
        else
                return (0);
        }
        else
        return (0);
        }
        else
        return (1);
}
int F()
{
        if (input[i] == '(')
        {
        i++;
        if (E())
        {
        if (input[i] == ')')
        {
                i++;
                return (1);
        }
        else
                return (0);
        }
        else
        return (0);
        }
        else if (input[i] >= 'a' && input[i] <= 'z' || input[i] >= 'A' && input[i] <= 'Z')
        {
        i++;
        return (1);
        }
        else
        return (0);
}
```



```
Enter an expression (e.g., (5 + 3) * 2):
4+3
Operand: 4
Operand: 3
Result: 7
user-HP-Pavilion-Notebook% cd ..
user-HP-Pavilion-Notebook% cd rec
cd: no such file or directory: rec
user-HP-Pavilion-Notebook% cd Rec
user-HP-Pavilion-Notebook% gcc parser.c -o JC
user-HP-Pavilion-Notebook% ./JC

Recursive descent parsing for the following grammar

E->TE'
E'->+TE'/@
T->FT'
T'->*FT'/@
F->(E)/ID

Enter the string to be checked:i*i+i

String is accepted%
user-HP-Pavilion-Notebook%
```

```
            | '(' expression ')'        { printf("Result: %d\n", $2); }
            ;

%%

int yyerror(const char *s) {
```

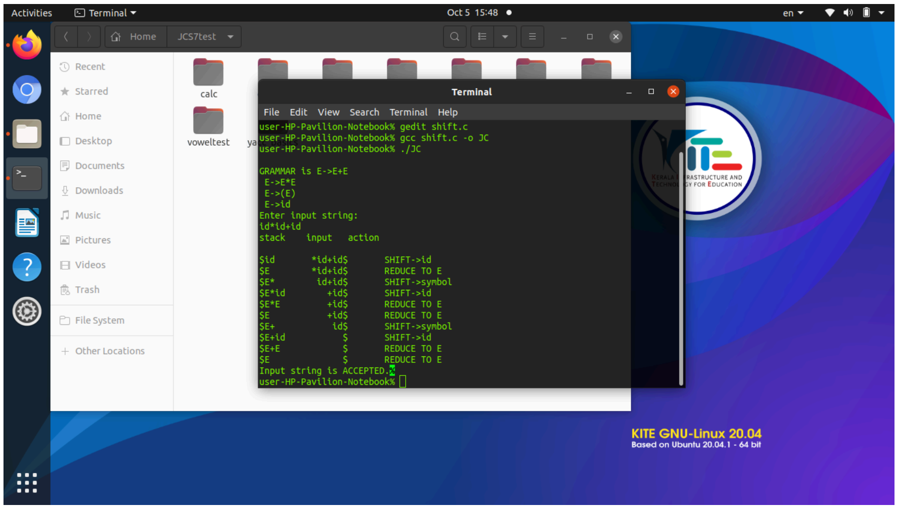
Experiment 10 Shift Reduce Parser

Code:

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

int k = 0, z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];

void check();
void printResult(int accepted);

int main() {
        // Grammar definitions
        puts("\nGRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
        puts("Enter input string: ");

        // Safe input reading using fgets instead of gets
        fgets(a, sizeof(a), stdin);
        a[strcspn(a, "\n")] = '\0';  // Remove newline character if fgets reads it
        c = strlen(a);  // Update the length of the string
        strcpy(act, "SHIFT->");

        puts("stack \t input \t action");

        // Main loop to parse the input
        for (k = 0, i = 0; j < c; k++, i++, j++) {
        if (a[j] == 'i' && a[j + 1] == 'd') {  // Check for 'id'
        stk[i] = a[j];
        stk[i + 1] = a[j + 1];
        stk[i + 2] = '\0';  // Null-terminate the string
        a[j] = ' ';
        a[j + 1] = ' ';
        printf("\n$%s\t%s$\t%sid", stk, a, act);
        check();
        } else {  // For other symbols
        stk[i] = a[j];
        stk[i + 1] = '\0';
        a[j] = ' ';
        printf("\n$%s\t%s$\t%ssymbol", stk, a, act);
        check();
        }
        }

        // Final check if the input is accepted or rejected
```

```
    if (stk[0] == 'E' && stk[1] == '\0' && a[0] == ' ') {
    printResult(1);  // Accepted
    }
    // Rule 5: Check if stack has E and input is fully processed
    else if (stk[0] == 'E' && a[0] == ' ') {
    printResult(1);  // Accepted
    } else {
    printResult(0);  // Rejected
    }

    return 0;
}

// Function to check and apply reduction rules
void check() {
    strcpy(ac, "REDUCE TO E");

    // Rule 1: E -> id
    for (z = 0; z < c; z++) {
    if (stk[z] == 'i' && stk[z + 1] == 'd') {
    stk[z] = 'E';
    stk[z + 1] = '\0';
    printf("\n$%s\t%s$\t%s", stk, a, ac);
    j++;
    }
    }

    // Rule 2: E -> E + E
    for (z = 0; z < c; z++) {
    if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E') {
    stk[z] = 'E';
    stk[z + 1] = '\0';
    stk[z + 2] = '\0';
    printf("\n$%s\t%s$\t%s", stk, a, ac);
    i = i - 2;
    }
    }

    // Rule 3: E -> E * E
    for (z = 0; z < c; z++) {
    if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E') {
    stk[z] = 'E';
    stk[z + 1] = '\0';
    stk[z + 2] = '\0';
```

```
        printf("\n$%s\t%s$\t%s", stk, a, ac);
        i = i - 2;
        }
        }

        // Rule 4: E -> ( E )
        for (z = 0; z < c; z++) {
        if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')') {
        stk[z] = 'E';
        stk[z + 1] = '\0';
        stk[z + 2] = '\0';
        printf("\n$%s\t%s$\t%s", stk, a, ac);
        i = i - 2;
        }
        }
}

// Function to print the result (Accepted or Rejected)
void printResult(int accepted) {
        if (accepted) {
        printf("\nInput string is ACCEPTED.");
        } else {
        printf("\nInput string is REJECTED.");
        }
}
```



```
user-HP-Pavilion-Notebook% gedit shift.c
user-HP-Pavilion-Notebook% gcc shift.c -o JC
user-HP-Pavilion-Notebook% ./JC

GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
Enter input string:
id*id+id
stack    input   action

$id       *id+id$       SHIFT->id
$E        *id+id$       REDUCE TO E
$E*        id+id$       SHIFT->symbol
$E*id        +id$       SHIFT->id
$E*E         +id$       REDUCE TO E
$E           +id$       REDUCE TO E
$E+           id$       SHIFT->symbol
$E+id          $        SHIFT->id
$E+E           $        REDUCE TO E
$E             $        REDUCE TO E
Input string is ACCEPTED.%
user-HP-Pavilion-Notebook% 
```