VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Aditya .

Reg No: 23BCE1344 .

Subject: Compiler Design .

Lab Task: ASSESSMENT-2

(PART-2)

~ Dr. Sureshkumar WI

Experiment-6:

1. Implement LEX code to count the frequency of the given word in a file
2. Implement LEX code to replace a word with another taking input from a file.
3. Implement LEX code to find the length of the longest word
4. Construct a lexical analyser using LEX tool

Algorithm (1)

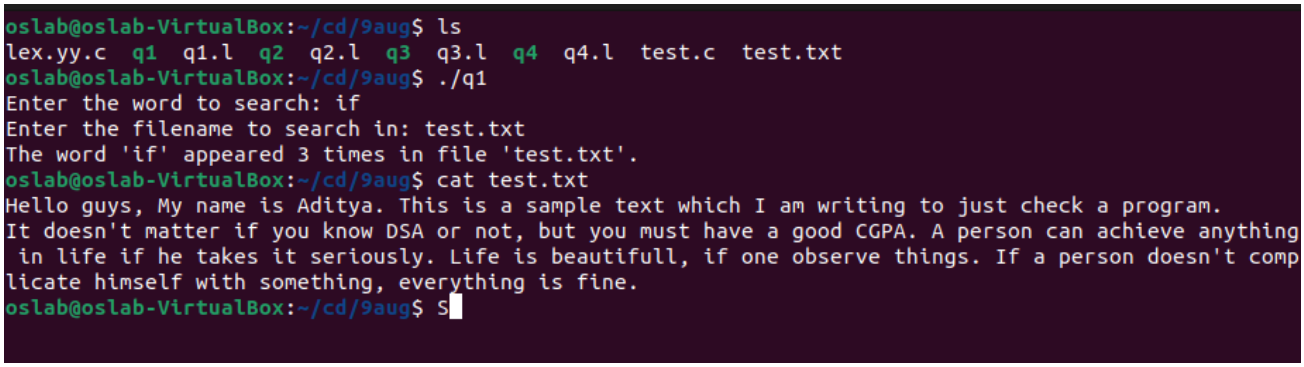
1. Start and read the target word from the user.
2. Take the filename from the user and open it for reading.
3. Use Lex to scan each token matching the pattern `[a-zA-Z]+`.
4. Compare each matched token with the target word and increment a counter if it matches.
5. After scanning the file, display the total count and close the file.

Source Code:

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

char word[100];
int count = 0;
FILE *yyin;
%}

%%
[a-zA-Z]+ {
  if(strcmp(yytext, word) == 0)
    count++;
}
.|\n  ; // ignore other characters
%%

int main() {
  char filename[100];

  printf("Enter the word to search: ");
  scanf("%99s", word);

  printf("Enter the filename to search in: ");
  scanf("%99s", filename);

  yyin = fopen(filename, "r");
  if(yyin == NULL) {
    perror("Error opening file");
    return 1;
  }

  yylex();

  printf("The word '%s' appeared %d times in file '%s'.\n", word, count, filename);
```

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

Input and Output:



Algorithm(2) :

1. Take the original word (`oldword`) and replacement word (`newword`) from the user.
2. Accept the filename, open it, and set it as input for Lex.
3. Use the pattern `[a-zA-Z]+` to match words from the file.
4. If a matched word equals `oldword`, print `newword`; otherwise, print the original word.
5. For non-word characters, print them unchanged, then close the file.

SourceCode:

```
%{

#include <stdio.h>

#include <string.h>


char oldword[100];

char newword[100];

FILE *yyin;

%}


%%

[a-zA-Z]+ {

  if(strcmp(yytext, oldword) == 0)

    printf("%s", newword);

  else

    printf("%s", yytext);

}

.|\n {

  // Print punctuation, whitespace, newline as is

  printf("%s", yytext);

}

%%


int main() {

  char filename[100];


  printf("Enter the word to replace: ");

  scanf("%99s", oldword);
```

```
  printf("Enter the replacement word: ");
  scanf("%99s", newword);

  printf("Enter the filename: ");
  scanf("%99s", filename);

  yyin = fopen(filename, "r");
  if(yyin == NULL) {
    perror("Error opening file");
    return 1;
  }

  yylex();

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

Input/Output:

```
oslab@oslab-VirtualBox:~/cd/9aug$ ls
lex.yy.c  q1  q1.l  q2  q2.l  q3  q3.l  q4  q4.l  test.c  test.txt
oslab@oslab-VirtualBox:~/cd/9aug$ ./q2
Enter the word to replace: if
Enter the replacement word: why
Enter the filename: test.txt
Hello guys, My name is Aditya. This is a sample text which I am writing to just check a program.
It doesn't matter why you know DSA or not, but you must have a good CGPA. A person can achieve anythin
g in life why he takes it seriously. Life is beautifull, why one observe things. If a person doesn't c
omplicate himself with something, everything is fine.
oslab@oslab-VirtualBox:~/cd/9aug$ S
```

Algorithm (3):

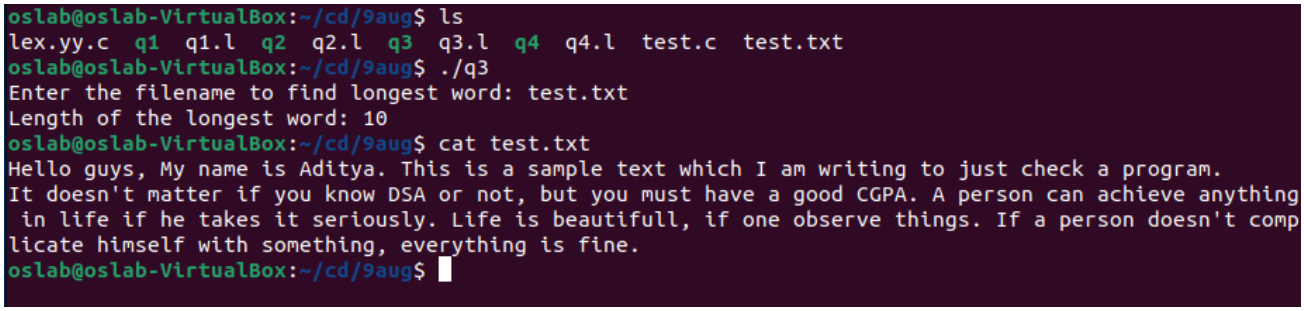
1. Prompt the user for the filename and open it for reading.
2. Use Lex to scan tokens matching the pattern `[a-zA-Z]+`.
3. For each matched token, check if its length (`yyleng`) is greater than `max_len`.
4. If yes, update `max_len` with the current word's length.
5. After scanning, print the value of `max_len` and close the file.

Source Code:

```
%{

#include <stdio.h>


int max_len = 0;

FILE *yyin;

%}


%%

[a-zA-Z]+ {

  if (yyleng > max_len) {

    max_len = yyleng;

  }

}

.|\n  ; // Ignore other characters

%%

int main() {

  char filename[100];

  printf("Enter the filename to find longest word: ");

  scanf("%99s", filename);

  yyin = fopen(filename, "r");

  if(yyin == NULL) {

    perror("Error opening file");

    return 1;
  }

  yylex();
```

```
  printf("Length of the longest word: %d\n", max_len);

  fclose(yyin);
  return 0;

}
```

Input/OUTPUT:



Algorithm (4) :

1. Prompt the user for an input filename and open it for reading.
2. Define patterns in Lex for keywords, identifiers, numbers, operators, and punctuation.
3. Scan the file token by token, matching each against the patterns.
4. For each match, print the token type and the actual text found.
5. Ignore whitespace and classify any unmatched text as UNKNOWN.

Source Code:

```
%{

#include <stdio.h>

#include <stdlib.h>


FILE *yyin;

%}


digit  [0-9]

letter [a-zA-Z]

id   {letter}({letter}|{digit})*

number {digit}+


%%


"if"      { printf("KEYWORD\t%s\n", yytext); }

"else"    { printf("KEYWORD\t%s\n", yytext); }

"while"   { printf("KEYWORD\t%s\n", yytext); }

"return"  { printf("KEYWORD\t%s\n", yytext); }


{id}      { printf("IDENTIFIER\t%s\n", yytext); }

{number}  { printf("NUMBER\t%s\n", yytext); }


"+"|"-"|"*"|"/"|"=" { printf("OPERATOR\t%s\n", yytext); }


";"|","|"("|")"|"{"|"}" { printf("PUNCTUATION\t%s\n", yytext); }
```

```
[ \t\n]+    ; // Ignore whitespace

.       { printf("UNKNOWN\t%s\n", yytext); }

%%

int main() {
  char filename[256];

  printf("Enter the input filename: ");
  if (scanf("%255s", filename) != 1) {
    fprintf(stderr, "Failed to read filename\n");
    return 1;
  }

  yyin = fopen(filename, "r");
  if (!yyin) {
    perror("Error opening file");
    return 1;
  }

  yylex();

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

Input/Output:

```
oslab@oslab-VirtualBox:~/cd/9aug$ ls
lex.yy.c  q1  q1.l  q2  q2.l  q3  q3.l  q4  q4.l  test.c  test.txt
oslab@oslab-VirtualBox:~/cd/9aug$ ./q4
Enter the input filename: test.c
UNKNOWN #
IDENTIFIER      include
UNKNOWN <
IDENTIFIER      stdio
UNKNOWN .
IDENTIFIER      h
UNKNOWN >
IDENTIFIER      int
IDENTIFIER      main
PUNCTUATION     (
PUNCTUATION     )
PUNCTUATION     {
IDENTIFIER      int
IDENTIFIER      a
OPERATOR        =
NUMBER  5
PUNCTUATION     ;
IDENTIFIER      int
IDENTIFIER      b
OPERATOR        =
NUMBER  9
PUNCTUATION     ;
IDENTIFIER      int
IDENTIFIER      sum
OPERATOR        =
IDENTIFIER      a
OPERATOR        +
IDENTIFIER      b
PUNCTUATION     ;
IDENTIFIER      printf
PUNCTUATION     (
UNKNOWN "
IDENTIFIER      The
IDENTIFIER      sum
IDENTIFIER      of
UNKNOWN %
IDENTIFIER      d
IDENTIFIER      and
UNKNOWN %
IDENTIFIER      d
IDENTIFIER      is
UNKNOWN %
IDENTIFIER      d
UNKNOWN "
PUNCTUATION     ,
IDENTIFIER      a
PUNCTUATION     ,
IDENTIFIER      b
PUNCTUATION     ,
IDENTIFIER      sum
PUNCTUATION     )
PUNCTUATION     ;
KEYWORD return
NUMBER  0
PUNCTUATION     ;
PUNCTUATION     }
oslab@oslab-VirtualBox:~/cd/9aug$ cat test.c
#include <stdio.h>

int main(){
        int a =5;
        int b =9;
        int sum = a+b;
        printf("The sum of %d and %d is %d",a,b,sum);
        return 0;
}
oslab@oslab-VirtualBox:~/cd/9aug$
```

Conclusion:

- The set of LEX programs successfully demonstrates how lexical analysis can be applied to various text-processing tasks, including word frequency counting, word replacement, longest word detection, and token classification.
- By leveraging pattern matching, each program efficiently processes input files while ignoring irrelevant characters such as whitespace and punctuation.
- The implementations highlight the flexibility of LEX in building both problem-specific text utilities and general-purpose compiler components.
- Overall, the assessment showcases a practical understanding of tokenization and the first phase of compilation, proving LEX as a powerful tool for language processing.

b)

Algorithm:

1. Write the Lex code (dfa_b.l) to simulate DFA that accepts strings containing "bca".
2. Open terminal, navigate to file location.
3. Compile with:
   flex dfa_b.l
   gcc lex.yy.c -lfl -o dfa_b

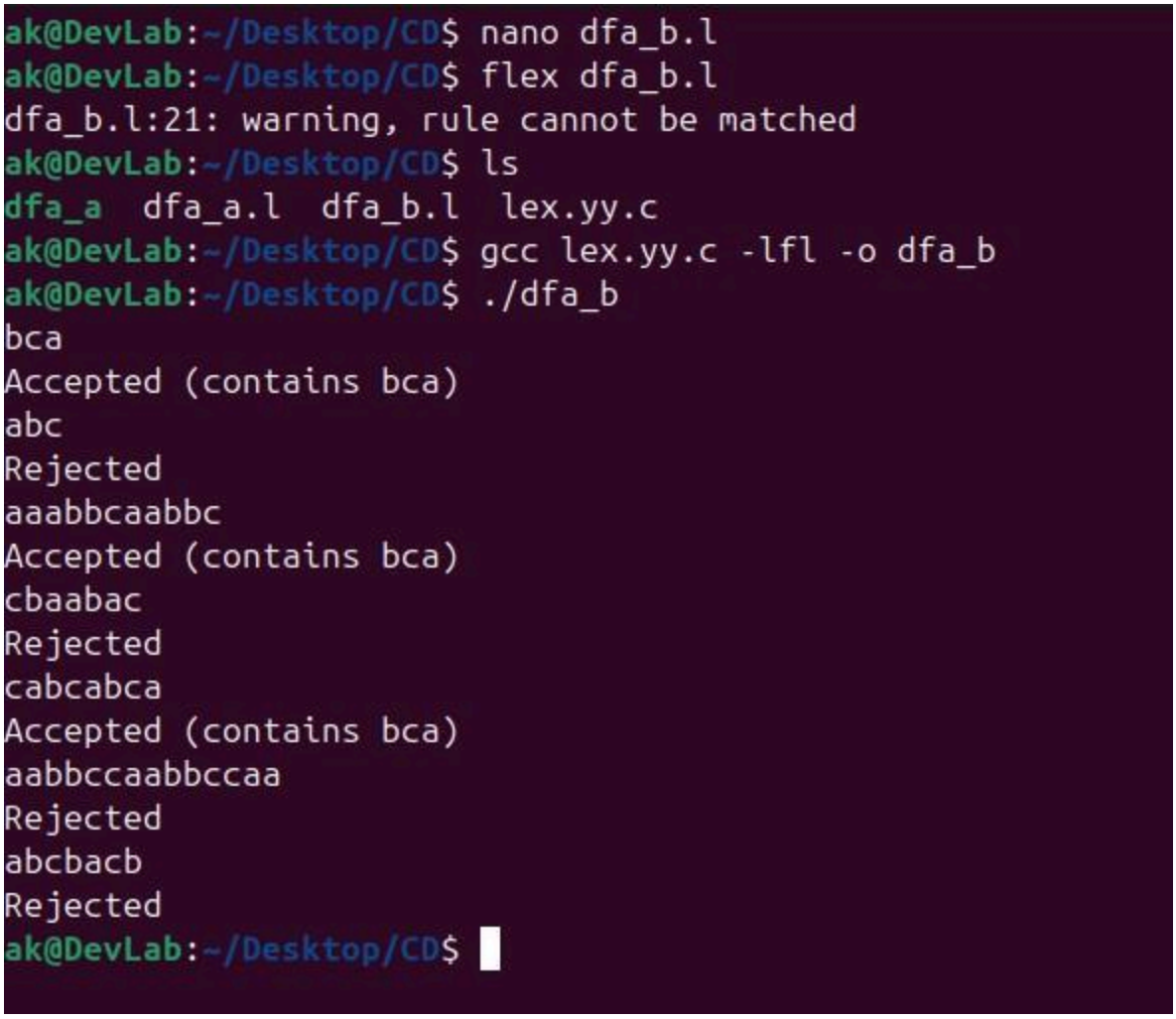
4. Run with:
   ./dfa_b

5. Enter strings (only using a, b, c) one per line. The program will accept if bca is present. Use Ctrl+D to end input.

Source Code:

```
%{
int state = 0;
%}

%%
b {
  if (state == 0) state = 1;
  else if (state == 1) state =
  1; else if (state == 2) state
  = 1; else if (state == 3)
  state = 3;
}
c {
  if (state == 1) state = 2;
  else if (state == 2) state = 0;
  else if (state == 3) state = 3;
}
a {
  if (state == 2) state = 3;
  else if (state == 3) state = 3;
}
[abc] {
  if (state != 3) state = 0;
}
\n {
  if (state == 3) printf("Accepted (contains bca)\n");
  else printf("Rejected\n");
  state = 0;
}
%%

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

Input/Output:



Conclusion:

- DFA successfully identifies the required substring using sequential state transitions.
- Lex provides efficient substring matching logic without manual character indexing.
- This method is ideal for compilers or parsers needing pattern recognition in token streams.