**Practical-1**

1. **Introduction of “VI” Editor with Commands.**

There are many ways to edit files in UNIX and one of the best ways is using screen-oriented text editor vi. It’s usually available on all the flavors of UNIX system. Its implementations are very similar across the board. It requires very few resources. It is more user friendly than any other editors.

Some Commands of VI Editor are as follows.

* vi filename: Creates a new file if it already does not exist, otherwise opens existing file.
* vi -R filename: Opens an existing file in read only mode.
* TOUCH: Use to create empty file or to update timestamp of an existing file.
* LS: List the file is current directory in alphanumeric order. It also display directories’ name in current directory.
* MDIR: Removes directory.
* CLEAR: Clear the contents of the screen.
* CHMOD: Set/Change the permission of one or more file for user.
* WHO:Gives list of all users who are currently logged on to the system.
* CD:Used to change the current working directory.
* MKDIR: Use to create new directories.

1. **Introduction of Lex.**

Lex is a tool that reads a stream of input from somewhere and breaks it up into its component pieces. Each component piece is a token. A token might be a keyword or a number or a string or punctuation. In this context a token is something that cannot be broken down into smaller pieces. Either it's a keyword or it's something else. If it's a string it's a string and we are not concerned with the machine representation of a string. Likewise with numbers and punctuation.

The first section contains general c code declarations and lex directives and is delimited from the second section by a %% line.

The second section contains regular expressions of a lex file. The second section is delimited from the third section by a %% line.

The third section contains raw c code which is copied verbatim to the output file.

And a lex file looks like this

%{

C declarations

%}

lex declarations

%%

Regular expressions

%%

Additional C code

The program Lex generates a so called `Lexer'. This is a function that takes a stream of characters as its input, and whenever it sees a group of characters that match a key, takes a certain action. A very simple example:

%%

/\* match everything except newline \*/

. ECHO;

/\* match newline \*/

\n ECHO;

%%

int yywrap(void) {

return 1;

}

int main(void) {

yylex();

return 0;

}

Two patterns have been specified in the rules section. Each pattern must begin in column one.

This is followed by whitespace (space, tab or newline) and an optional action associated with the

pattern. The action may be a single C statement, or multiple C statements, enclosed in braces.

Anything not starting in column one is copied verbatim to the generated C file. We may take

advantage of this behavior to specify comments in our lex file. In this example there are two

patterns, “.” and “\n”, with an ECHO action associated for each pattern. Several macros and

variables are predefined by lex. ECHO is a macro that writes code matched by the pattern. This is the default action for any unmatched strings.

%{

#include <stdio.h>

%}

%%

stopprintf("Stop command received\n");

startprintf("Start command received\n");

%%

The first section, in between the %{ and %} pair is included directly in the output program. We need this, because we use printf later on, which is defined in stdio.h. Sections are separated using '%%', so the first line of the second section starts with the 'stop' key. Whenever the 'stop' key is encountered in the input, the rest of the line (a printf() call) is executed. Besides 'stop', we've also defined 'start', which otherwise does mostly the same. We terminate the code section with '%%' again.

Matching patterns of text

|  |  |
| --- | --- |
| Expression | Matches |
| . | any single character |
| * | zero or more of the previous expression |
| .* | zero or more arbitrary characters |
| \< | beginning of a word |
| \> | end of a word |
| \ | quote a special character |
| \* | the character ``*'' |
| ^ | beginning of a line |
| $ | end of a line |
| [*set*] | one character from a set of characters |
| [XYZ] | one of the characters ``X'', ``Y'', or ``Z'' |
| [[:upper:]][[:lower:]]\* | one uppercase character followed by any number of lowercase characters |
| [^*set*] | one character not from a set of characters |
| [^XYZ[:digit:]] | any character except ``X'', ``Y'', ``Z'', or a numeric digit |

**Ex:Check given input is digit or character or special character in lex program.**

**Code:**

%{

#include <stdio.h>

%}

%%

[A-Za-z]+ printf("Characters");

[0-9]+ printf("DIGIT");

.\* printf("Special characters");

%%

int yywrap(void)

{

return 1;

}

int main(void)

{

yylex();

**}**

**OutPut:**

