Assignment No.: A6

Roll No.

- **Title :**
  To write an application for parsing input text file concurrently and compare the result the result against serial parsing.

- **Problem Definition :**

  Write an application to parse input text file concurrently and compare the result of concurrent parsing with serial parsing (Use Concurrent YACC parser).

- **Learning Objective :**

  To study YACC parser and concurrent programming.

- **Learning Outcome :**

  Successfully implemented parsing of a text file using concurrent YACC.

- **Software and Hardware Requirement:**

  – Latest version of 64 Bit Operating Systems Open Source Ubuntu 14.04
  – Multicore CPU equivalent to Intel i5/7 4^{th} generation
  – Lex
  – YACC/BISON
  – gedit(any text editor)
  – GNU C compiler

- **Theory :**

  1. Introduction

  The program has been written in lex, the lexical analyzer and is further parsed by using a yacc parser. The program shall use only standard library functions. As the program is utilizing the OpenMP compiler directives, the program will run on uniprocessor system but to observe parallelism multi-cores processor system is needed. Along with the c compiler the lexer lex or flex and the parser yacc or bison should be installed on your system. Except these standard software the program shall not require any particular hardware or software.

  The program is designed to act like a basic tokenizer and parser accepting a very wide range of file extensions which stores text in ASCII format. There is significant improvement in performance using parallel pro-gramming in OpenMP which introduces shared memory parallelism utilizing multiple processors of advanced architecture.

  When lex fnds a match, yytext points to the first character of the match in the input builder. The string itself is part of the input builder, and is NOT allocated separately. The value of yytext will be overwritten the next time yylex() is called. In short, the value of yytext is only valid from within the matched rule's

action. Often, you want the value of yytext to persist for later processing, i.e., by a parser with non-zero lookahead. In order to preserve yytext, you will have to copy it with strdup() or a similar function. But this introduces some headache because your parser is now responsible for freeing the copy of yytext. If you use a yacc or bison parser, (commonly used with ex), you will discover that the error recovery mechanisms can cause memory to be leaked.
yytext points to the first character of the match in the input builder if the word retuened by yytext is a number i.e. a string composed of only decimal numbers(0-9) the the word is classified under the integer constant token. In the case that yytext returns one of the words from the set of fvoid, int, char,include, main, printf, if, for, break, whileg then the given word is clas- sited under the keyword token. If the word is composed of a combination of letters, numbers and underscore then the word is an identifier. Else if the word is constant it is literal.

2. Related Concepts for Concurrent Yacc

- Writing tokeniser using Lex.
- Writing a parser using Yacc.
- Shared memory concurrent programming using OpenMP.

3. Concept of programming language use

- **Lex**: Lex is a computer program that generates lexical analyzers (”scanners”).Lex is commonly used with the yacc parser generator. Lex, originally written by Mike Lesk and Eric Schmidt and described in 1975,is the standard lexical analyzer generator on many Unix systems, and an equivalent tool is specified as part of the POSIX standard.
- **Yacc**:Yacc is a computer program for the Unix operating system.The name is an acronym for ”Yet Another Compiler Compiler”.It is a LALR parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to BNF.It was originally developed in the early 1970s by Stephen C. Johnson at AT&T Corporation and written in the B programming language, but soon rewritten in C. It appeared as part of Version 3 Unix, and a full description of Yacc was published in 1975.
- **OpenMP**:OpenMP (Open Multi-Processing) is an API that supports multiplatform shared memory multiprocessing programming in C, C++, and Fortran on most processor architectures and operating systems, including Solaris, AIX, HP-UX, Linux, Mac OS X, and Windows platforms.It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread (series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.

4. Algorithm:

Openmp code:

1. Start
2. Accept the file to be parsed from user
3. Use threads for parallel execution
4. Copy the output in an character array
5. Display the result
6. End

On Lex Code:

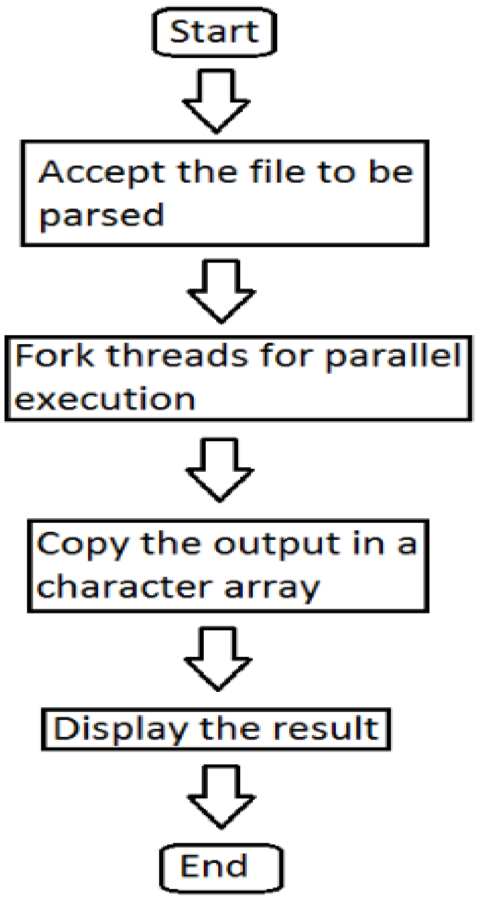
1. Start
2. Open file

```
3. Read the file to be parsed
4. Define rules for identifiers , literals , keywords , binary operators and integer constants
5. if literal
6.     then print literal
7. else if keyword
8.     then print keyword
9. else if identifier
10.     then print identifier
11. else if binary operator
12.     then print binary operator
13. else if integer constant
14.     then print integer constant
15. Close file
16. End
```

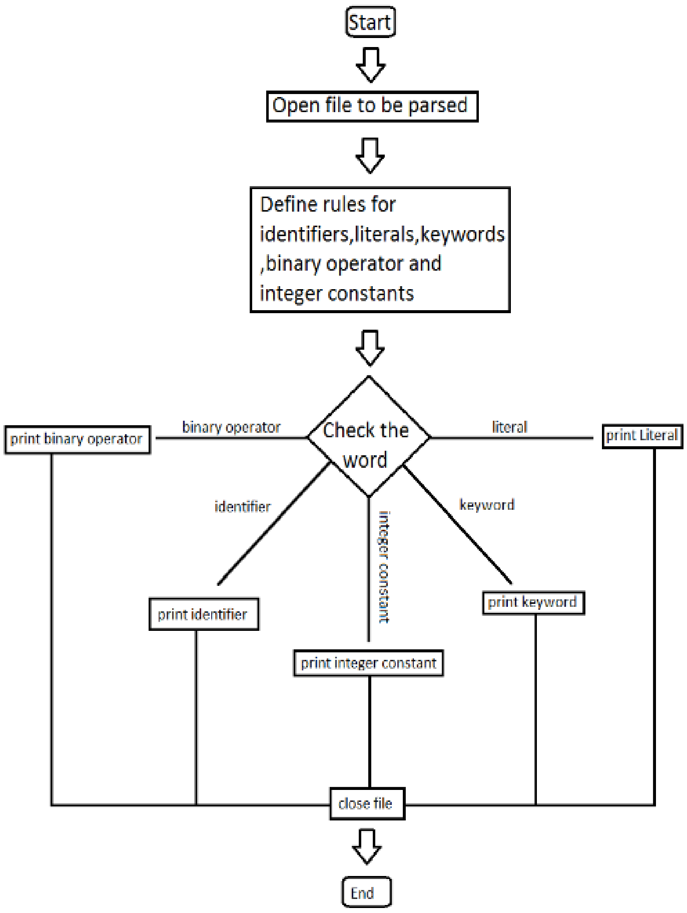
Commands for execution

```
lex prog_name.l
cc lex.yy.c y.tab.h -ll
gcc -fopenmp prog.c
./a.out filename
```

5. Flowcharts:


Start
Accept the file to be parsed
Fork threads for parallel execution
Copy the output in a character array
Display the result
End

(a) OpenMP Flowchart



(b) Lex Flowchart

Figure 1: Flowcharts

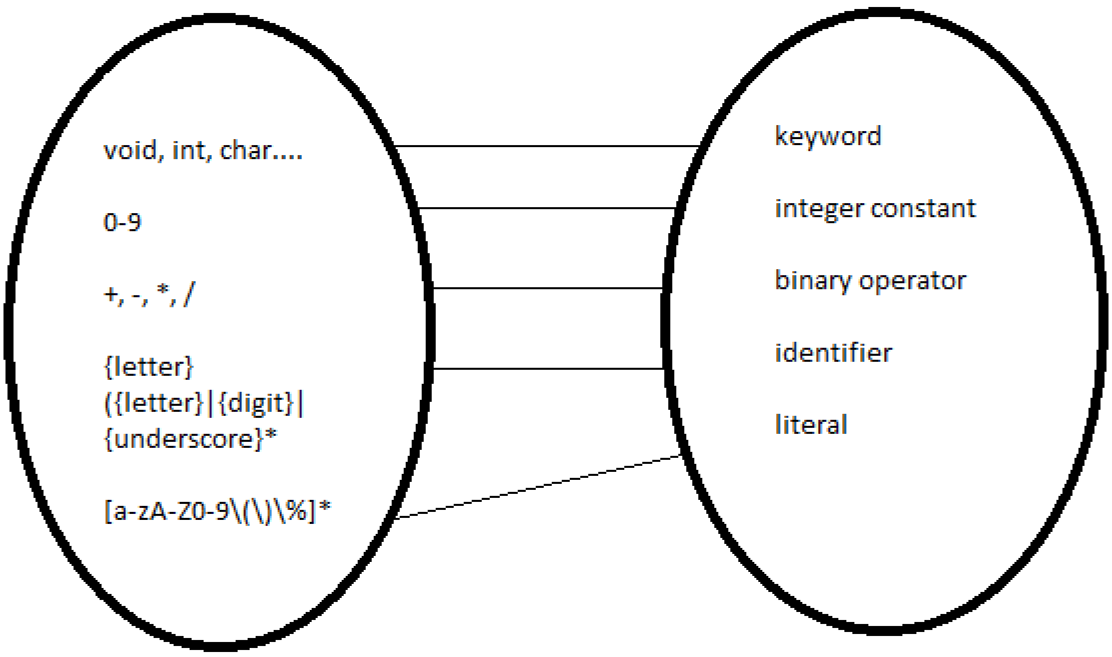
6. Mathematical model:

Let U = { s, e, f, S, F, I, O, DD, ND } be a universal set
where,

s = start
e = end
f = set of functions ; f = { f1,f2,f3,f4,f5,f6 }
I = Input set ; I = { I1,I2 }
O = Output set
DD = Deterministic Data
ND = Non-Deterministic Data
S = Cases of Success
F = Cases of Failure

In the case of our program,
s = Unformatted raw text document.
e = Tokenized and Parsed document.
f1 = if $(yytext == (void|int|char|include|main|printf|if|for|break|while))$ then given token is a keyword.
f2 = if $(yytetext == ``+'' |``-'' |``*'' |``/'')$ then given token is a binary operator.
f3 = if $(yytext == (0{-}9))$ then the given token is an integer constant. f4 = if $(yytext == letter(letter|digit|underscore*$
then the given token is a identifier.
f5 = if $(yytext == (a-z)|(A-Z)|(0-9))$ then the given token is a literal.

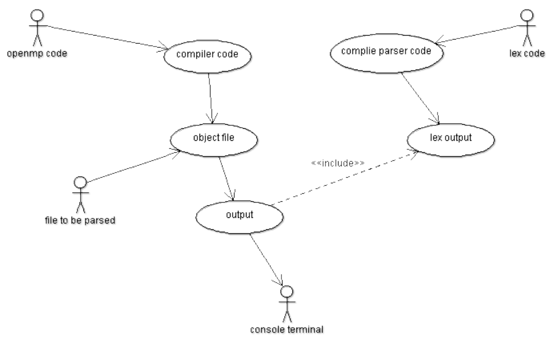
Venn diagram: 7. SRS:



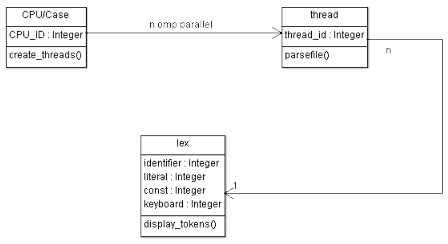
Specification	Description
Project Scope	To understand Parsing using concurrent Yacc
Functional Requirements	Whenever any file with ASCII encoding is passed to the lexer i.e. the lex.l file, a tokenized output should be generated
Non-functional Requirement	The program should be executed in parallel making use of the multiple cores of the computing device and should be tolerant to any input file and must be robust.
Design And Implementation	The input file should be parsed a sraw text and should be classified into the tokens as per specified rules until the end of the file.
Machine Specification	32/64 bit computing device with two or more cores, atleast 128MB RAM.

8. UML Diagrams:

Use case Diagram:



Class Diagram:



- **Input:**

  Input to this program is a file to be parsed.

- **Output:**

  Output of this program are tokens.

- **Conclusion :**

Hence we have studied Concurrent Yacc to parse input text file concurrently.OpenMP is used for parallel execution of the program

Course Outcomes	Achieved Outcome
CO I : Ability to perform multi-core, Concurrent and Distributed Programming.	√
CO II : Ability to perform Embedded Operating Systems Programming using Beaglebone	
CO III :Ability to write Software Engineering Document.	√
CO IV :Ability to perform Concurrent Programming using GPU.	

FAQ:

- State and Explain the phases of Compiler?
- Explain the structure of a lex file with example.
- How can concurrency be achieved using Yacc?
- How do lex and yacc work internally ?
- Explain different functions in yacc library .
- What is Openmp ? Explain how parallelism can be achieved using openmp .