INDEX

**INTRODUCTION**

**Lex**

In computer science, lex is a program that generates lexical analyzers ”scanners” or ”lexers”. It is a tool for scanners or programs which recognizes lexical patterns in text . Lex, originally written by Eric Schmidt and Mike Lesk, is the standard lexical analyzer generator on many Unix systems, and a tool exhibiting its behavior as specified as a part of the POSIX standard.

Lex reads an input stream .ie. the given input files or its standard input if no file names are given ;for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code,called rules specifying the lexical analyzer .It then generates as output a C source file, lex.yy.c, which defines a routine yylex(). This file is compiled and linked with -ll library to produce an executable. When the executable is run, it analyzes its input for occurences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

It is commonly used with the yacc parser generator.

**Yacc**

The computer program yacc is a parser generator developed by Stephen C. Johnson at AT&T for the Unix operating system. The name is an acronym for ”Yet Another Compiler’s Compiler.” It generates a parser .ie. the part of a compiler that tries to make syntactic sense of the source code, based on an analytic grammar written in a notation similar to BNF. Yacc generates the code for the parser in the C programming language.

In general it provides a general tool for imposing structure on the input to a computer program.

The yacc part includes the following:

1. rules to describe the elements of the input;

2. code to be invoked when a rule is recognized;

3. either a definition or declaration of a low-level scanner to examine the input.

Yacc then turns the specification into a C-language function that examines the input stream. This function, called a parser, works by calling the low-level scanner.

The scanner, called a lexical analyzer, picks up items from the input stream. The selected items are known as tokens. Tokens are compared to the input construct rules, called grammar rules.

When one of the rules is recognized, the code you have supplied for the rule is invoked. This code is called an action. Actions are fragments of C language code. They can return values and use values returned by other actions.

The heart of the yacc specification is the collection of grammar rules.Each rule describes a construct and gives it a name.

**How lex and yacc work together:**

Lex and Yacc (a parser generator) are commonly used together. Yacc uses a formal grammar to parse an input stream, something which Lex cannot do using simple regular expressions .However, Yacc cannot read from a simple input stream - it requires a series of tokens. Lex is often used to provide Yacc with these tokens.

The tokens defined by the lexwhich are returned by the lexer is specified as the input for the yacc .In the main program of yacc, a call to yyparse(), first calls the routine yylex() which returns the tokens. These tokens are then received by the yacc and actions are taken on them according to the rules specified in the yacc grammar.

**Scanf and printf:**

Any C program contains various types of statements. To check for the syntax of a C program you have to check for the individual syntax of each statement. The statements ***scanf*** and ***printf*** are used numerous times as they are used for formatting the output as well as to obtain input from the user.

Area of the project application is in designing of a compiler to check for the syntax of scanf and printf statements which are a part of C programs.Most of the C compilers till date use a specific grammar to evaluate the syntax of various functions. This mini project can be viewed a s a part of such grammar.

The domain of this mini project can be viewed as a subse t of the domain of a conventional C compiler.By this it is meant that only printf and scanf grammar is analysed and checked.

As it does the job of syntax checking and single error reporting in succession , it can be viewd as an interpretor .

The mini application to be developed should be able to identify or check for the syntax of ***printf*** and ***scanf*** statements as well as identify the error type along with the line number in which the error occurs.

The domain of the project application is as a part of a C compiler to check the syntax of ***scanf-printf*** statements of a C program. The mini project would take in files containing only ***printf*** and ***scanf*** statements and would check for its syntax and would report one error at a time. Once that error is corrected then the next error in the syntax is reported. The various types of errors are informed to the user through notification boxes.

**OBJECTIVES**

The objectives of the mini project can be stated as follows:

* To SYNTACTICALLY ANALYZE AND validate printf and scanf statements from the file which is input to the application,
* To identify and notify the user with the errors in the syntax of the pritnf and scanf statements as and when generated one at a time.

**2. Requirements**

2.1 Input requirements

Input is in the form of a file that contains printf and scanf statements. The statements within the file can be valid or invalid.ie.the statements may or may not contain the tokens rewuires for ht evalidity of the statement.Various examples of printf and scanf statements in the file are:

printf(“VALID”);

printf(“VALID %d %f %e ”,a,b,c);

scanf(“%d”,&a);

printf(“INVALID %d %f %e ”,a,b,c)

printf(;

printf(“INVALID”)

scanf(“%d”);

printf(INVALID);

* 1. Output requirements

Validity of the statements has to be notified as the output along with the errors ,if invalid.

Example outputs for some random statements are as follows:

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| printf(“VALID”); | Valid |
| printf(“VALID %d %f %e ”,a,b,c); | Valid |
| scanf(“%d”,&a); | Valid |
| printf(“INVALID %d %f %e ”,a,b,c) | Invalid  Semi colon missing |
| printf(“”; | Invalid  Closing Bracket missing |
| printf(“INVALID”) | Invalid  Semi colon missing |
| scanf(“%d”); | Invalid  Variable to be assigned missing |
| printf(INVALID); | Invalid  Opening Quote missing |

* 1. Functional Requirements
* The basic functionality of the program is to validate printf and scanf statements.
* The software must provide the user with the facility to specify the input file that contains the statements to be validated.
* A method to invoke the grammar.
* A clearly specified output showing the result of the validation should also be provided.
  1. External Interface Requirements

2.4.1 User Interface

An unambiguous and easy to use graphical User Interface is provided which serves the following purposes:

* Specifying the input file.
* Specifying the operation to be performed(Validation).
* Notifying the validity and extending the invalidity notification, if any.

2.4.2 Hardware Requirements

For the development phase taken in to consideration ,the following can be listed out as the hardware requirements:

✔ Intel processor

✔ 256MB of RAM or higher

✔ 40GB hard disc

2.4.3 Software Requirement

Keeping in mind the development phase ,the following software requirements can be listed out:

-A linux platform

-for the grammar generation a unix platform with lex and yacc tools

-for the front end design qt4/zenity support on unix platform

-a C compiler to compile the code generated y the yacc to link the back end with the front end

2.5 Non functional Requirements:

\* As the application can be run on any platform,it is highly compatible.Thus it is higly portable.

\* The application can be accessed by program developers(C program developers) to check the syntax of the scanf and printf statements used in their programs.Thus making it accessible.

\* As the code can be used as a module for the conventional C compiler generation. it is adaptive in nature. Thus it is adaptable.

**Design and Implementation:**

Algorithm:

➢ In the YACC file, we write our own main() function, which calls yyparse() at one point. The function yyparse() is created by YACC, and ends up in y.tab.c.

-yyparse() reads a stream of token/value pairs from yylex(), which needs to be supplied.

-The yylex() as written by Lex reads characters from a FILE \* file pointer called yyin. If we do not set yyin, it defaults to standard input. It outputs to yyout, which if unset defaults to stdout.

-Each call to yylex() returns an integer value which represents a token type. This tells YACC what kind of token it has read.

-In present situation, the parser goes through the statements given in the input file. It identifies the tokens as declared in the lex part and executes the rules given in the grammar section of yacc. If the syntax is not correct ,then it displays the line no. as well as the error type. If the syntax is right, it displays that the the statements have the right syntax.

pns.l -------------->LEX------------->lex.yy.c

pns.y -------------->YACC -d--------------> y.tab.c y.tab.h

lex.yy.c y.tab.c-------------> C compiler-------------> --------------> a.out

**Testing and Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SR.NO | Features | Expected result | Result Obtained | Remark |
| 1 | File input containing valid printf statement | Scan the entire line and validate | Scans the given line and validates | Success. |
| 2 | File input containing more then one invalid statement | Scans the entire file and displays all the errors occurred | Only the first error is displayed | Should modify the grammar to display multiple error . |
| 3 | File input containing more then one valid statement | No errors will be displayed | Valid Statement is displayed | Success. |
| 4 | File input containing both printf and scanf statement | Both the statements are syntactically analyzed | Both the statements are validated one | Succecc. |
| 5 | File input containing invalid printf and scanf statements together | Reports both the errors | Only the first error is displayed | Have to check for multiple errors. |
| 6 | File input containing error in conversion specifications | Reports all the related errors | Fails to report the error | Have to write the advanced grammar to match those. |