Web Browser in C++ using flex and bison

Project Proposal

Submitted to:
Department of Electronics and Computer

Submitted By:
WebBrowser Team:
1. Nootan Ghimire (BCT/069/21)
2. Nishan Bajracharya (BCT/069/20)
3. Shashi Khanal (BCT/069/38)

1. Introduction

1.1. Lexical Analysis and Lexer

In computer science, **lexical analysis** is the process of converting a sequence of characters into a sequence of tokens. A program or function that performs lexical analysis is called a **lexical analyzer** or **lexer**.

1.1.1. The Lex Theory

During the first phase of parsing the parser reads the input and converts strings in the source to **tokens**. We can specify patterns to a lexer with regular expressions, so that the lexer can generate code that will allow it to scan and match strings from the input. Typically the lexer is used to identify and return the tokens to the parser.

The following represents a simple pattern, composed of a regular expression, that scans for identifiers.

```
letter(letter|digit)*
```

This pattern matches a string of characters that begins with a single letter followed by zero or more letters or digits. This example nicely illustrates operations allowed in regular expressions:

- The ***** operator
- The **|** operator
- Concatenation Operation
-

Any regular expression expressions can be expressed as a Finite State Automaton (FSA). We can represent an FSA using states, and transitions between states. There is one start state and one or more final or accepting states.

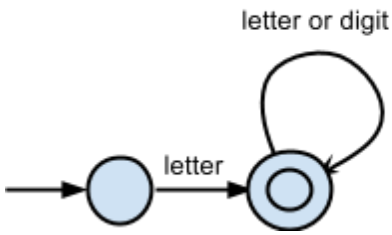

Figure 3: Finite State Automaton

In Figure 3 state 0 is the start state and state 1 is the accepting state. As characters are read we make a transition from one state to another. When the first letter is read we transition to state 1. We remain in state 1 as more letters or digits are read until the end of input.
Any FSA may be expressed as a computer program. For example, our 2-state machine is easily programmed:

```
start:  goto state0

state0: read c
        if c = letter goto state1

state1: read c
        if c = letter goto state1
        if c = digit goto state1
```

This is the technique used by lexer programs. Regular expressions are translated by the lexer to a computer program that mimics an FSA. Using the next *input* character and *current state* the next state is easily determined by indexing into a computer-generated state table.

1.2. Parser

Parsing or **syntactic analysis** is the process of analysing a string of symbols, in computer languages, according to the rules of a formal grammar. The term *parsing* comes from Latin *pars* (*orationis*), meaning part (of speech).

A parser is the program that helps in parsing the string, or in the typical case: a programming or an scripting language. Parser reads the token from the lexer and analyzes it according to the grammar rules that we provide.

1.3. Flex and Bison

Flex and Bison are lexer and parsers. They are free and open source alternative to their counterpart: lex and yacc. They were actually designed for C, so we need to define some magic to make it work on C++ too.

1.3.1 Flex

Flex is a tool for generating scanners: programs which recognized lexical patterns in text. Flex reads 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. Flex generates as output a C source file, ***lex.yy.c***, which defines a routine ***yylex()***. This file is compiled and linked with the ***-lfl*** library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

A typical flex input file looks like:

```
/*The definition part*/
%{
#include <iostream>
using namespace std;
#define YY_DECL extern "C" int yylex()
%}
```

```
/*The rules part*/
%%
stop    cout<<"Stop command received\n";
start   cout<<"Start command received\n";
%%

/* The source code part -- generally not required */
main() {
      // lex through the input:
      yylex();
}
```

We are going to name this file **`example.l`** . We should now use the flex program and feed this as it's input which will generate a file called **`lex.yy.cc`** . Now we compile **`lex.yy.cc`** and then run the executable which matches different inputs.

Example:

```
$ flex example.l
$ g++ lex.yy.cc -lfl -o myexecutable
$ ./myexecutable
```

1.3.2 Bison

Computer program input generally has some structure; in fact, every computer program that does input can be thought of as defining an ***input language*** which it accepts. An input language may be as complex as a programming language, or as simple as a sequence of numbers. Unfortunately, usual input facilities are limited, difficult to use, and often are lax about checking their inputs for validity.

Bison is a general-purpose parser generator that converts a grammar description for an context-free grammar into a C program to parse that grammar. Once you are proficient with Bison, you may use it to develop a wide range of language parsers, from those used in simple desk calculators to complex programming languages. It is based on the Yacc.
Yacc provides a general tool for describing the input to a computer program. The Yacc user specifies the structures of his input, together with code to be invoked as each such structure is recognized. Yacc turns such a specification into a subroutine that handles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine.

A typical Bison file looks like:

```
/*The declaration part*/
%{
```

```
#include <iostream>
using namespace std;

extern "C" int yylex();
extern "C" int yyparse();
extern "C" FILE *yyin;

void yyerror(const char *str) {
      cout<<"Error: "<<str<<"\n";
}
int yywrap(){ return 0; }
main() {
      yyparse();
}
%}

%token NUMBER ALPHA   //Defining tokens available (from lexer)
/* The grammar section */

%%
commands: /* empty */
      | commands command
command:
      isAlpha
      |
      isNumber
      ;
isAlpha:
      ALPHA
      {
            cout << “\nfound alphabet :D ”;
      }
      ;
isNumber:
      NUMBER
      {
            cout<<"\nfound Number :D”;
      }
      ;
%%
```

If looked closely into the bison file, one can notice that this is based on Context Free Grammars as learnt on the lectures on **Theory of Computation**. Basically in bison file, we provide set of grammar so that it can match the required text in the input file.

1.4. Graphics Library

GTK+(Gimp Toolkit) library will be used for the graphical programming. GTK+ is an object-oriented widget toolkit written in the C programming language; it uses the GLib object system for the object orientation. While GTK+ is primarily targeted at the X Window System, it works on other platforms, including Microsoft Windows (interfaced with the Windows API), and Mac OS X (interfaced with Quartz).

Typically, GTK+ requires the executables from official GTK+ website and uses the gtk.h header file located inside include/gtk/gtk.h to work.

//Possibly, we'll use Glade GUI Builder for GTK+.

1.5. The Internet

The **Internet** is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to serve several billion users worldwide. It is a *network of networks* that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless, and optical networking technologies. The Internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (WWW), the infrastructure to support email, and peer-to-peer networks.

It started as a networking project in ARPANET but is a very popular among current generation people.

1.6. HTML and Web Browser

HTML or **HyperText Markup Language** is the main markup language for creating web pages and other information that can be displayed in a web browser.
HTML is written in the form of HTML elements consisting of *tags* enclosed in angle brackets (like <html>), within the web page content. HTML tags most commonly come in pairs like <h1> and </h1>, although some tags represent *empty elements* and so are unpaired, for example <img>. The first tag in a pair is the *start tag*, and the second tag is the *end tag* (they are also called *opening tags* and *closing tags*). In between these tags web designers can add text, further tags, comments and other types of text-based content

A Web Browser is the software that helps to render the html codes into beautiful page. A Web browser analyses the tokens i.e., parses the html and show and render it to user.

1.6 Graphics Libraries For C++

There are a lot of graphics libraries for C++, but we are using **GTK+** so that we can display the contents inside the window. Since GTK+ is an cross-platform graphics library, we do not have to worry about system-specific windows manager.

2. Objectives

We have some of our objectives that we seek in order to complete the project.

- To be able to make a simple browser using c plus plus.
- To understand the working of flex/bison.
- To be able to gain the practical knowledge of subjects like OOP and ToC

3. Literature Review

Web Browsers have always existed on the past. The first web browser was Nexus created by Tim Berners Lee himself. Following the evolution, there were two major browser rendering engine in the industry, the most popular ones being **webkit** (used by Google Chrome/Safari) and **gecko**(used by Mozilla firefox).

Both the browser engines were coded in C++ and have standard set of API available.

Gecko is a free and open source layout engine used in many applications developed by Mozilla Foundation and the Mozilla Corporation (notably the Firefox web browser including its mobile version and their e-mail client Thunderbird), as well as in many other open source software projects.

It is designed to support open Internet standards, and is used by different applications to display web pages and, in some cases, an application's user interface itself (by rendering XUL). Gecko offers a rich programming API that makes it suitable for a wide variety of roles in Internet-enabled applications, such as web browsers, content presentation, and client/server.

Gecko is written in C++ and is cross-platform, and runs on various operating systems including BSDs, Linux, OS X, Solaris, OS/2, AIX, OpenVMS, and Microsoft Windows. Its development is now overseen by the Mozilla Foundation and is licensed under version 2 of the Mozilla Public License.

WebKit is a layout engine software component designed to allow web browsers to render web pages. It powers Apple's Safari web browser application and previously powered Google's Chrome web browser application.

It is also used as the basis for the experimental browser included with the Amazon Kindle e-book reader, as well as the default browser in the Apple iOS, Android, BlackBerry 10, and Tizen mobile operating systems. WebKit's C++ API provides a set of classes to display web content in windows, and implements browser features such as following links when clicked by the user, managing a back-forward list, and managing a history of pages recently visited.

But instead of using these rendering engines and making our work easier, we are doing it from scratch so that we can know the basics about parsing/lexing.

4. Technical Details

The program will be coded in **C++** and the **gcc** compiler will be used for compiling the source code. The program is expected to run on linux platform and it may as well run in any other platform.

All the testing-for-bugs will be done on Ubuntu operating system. We will use **git** source control and use **github** to host our codes. Also, the issue tracking will be easier when doing a project with github.

The program will be released on a open source **creative commons license.**
Open Source: CC-BY-SA 4.0 International

5. Methodology

We are going to use collaborative editing using github. Besides, our program will follow an iterative process where we will reach to full functionality from basic functionality with some iterations.

A rough algorithm for the program is listed below.

5.1. Algorithm

Step 1: Get the URL from user
Step 2: Parse the URL
Step 3: Get the HTML
Step 4: Parse the HTML
Step 5: Render the HTML to user

8. Bibliography

Aho, Alfred V., Ravi Sethi and Jeffrey D. Ullman [2006]. Compilers, Principles, Techniques and Tools (2nd edition). Addison-Wesley, Reading, Massachusetts.

Gardner, Jim, Chris Retterath and Eric Gisin [1988]. MKS Lex & Yacc. Mortice Kern Systems Inc., Waterloo, Ontario, Canada.

Johnson, Stephen C. [1975]. Yacc: Yet Another Compiler Compiler. Computing Science Technical Report No. 32, Bell Laboratories, Murray hill, New Jersey.

Lesk, M. E. and E. Schmidt [1975]. Lex - A Lexical Analyzer Generator. Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey.

Levine, John R., Tony Mason and Doug Brown [1992]. Lex & Yacc. O'Reilly & Associates, Inc. Sebastopol, California.

E. Shepherd *et al.*. (2013, July 15) *Embedding Mozilla* [Online]. Available: http://www.mozilla.org/projects/embedding/

Wikipedia. (2014, Jan 28) *Gecko* [Online] Available: http://en.wikipedia.org/wiki/Gecko_(layout_engine)

Wikipedia. (2014, Jan 26) *Webkit* [Online] Available: http://en.wikipedia.org/wiki/WebKit

B.Hubert. (2004, September 20) *Lex and YACC primer/HOWTO (v0.8)* [Online]. Available: http://ds9a.nl/lex-yacc/cvs/output/lexyacc.html