Program #4
Due: Thursday March 22nd, 2018 at 11:59:00 pm

Instructor	Dr. Stephen Perkins
Office Location	ECSS 4.702
Office Phone	(972) 883-3891
Email Address	stephen.perkins@utdallas.edu
Office Hours	Tuesday and Thursday 1:30pm – 4:00pm and by appointment

TAs

CS/SE 3377.002 - Vatsalkumar Patel - drp140230@utdallas.edu

CS/SE 3377.501 - Alexander Lee Hayes - alh170730@utdallas.edu

CS/SE 3377.502 – Subrahmanyam Oruganti – Oruganti.Subrahmanyam@utdallas.edu

Purpose

Demonstrate the ability to utilize the UNIX ***make*** build system with implicit rules. Demonstrate the ability use the flex lexical analyzer package to analyze an input file. Demonstrate the ability to use the bison parser package to make sense of the contents of a file and generate useful output. Demonstrate the ability to have your program offer different services based on the name by which it was called.

Assignment

Overview:

The process of converting an input file (stream of characters) into an understandable format is called syntax analysis. It is usually divided into two parts. The first, *lexical analysis*, tokenizes the input via a lexical scanner. The second, *parsing*, generates an abstract model of the tokens and then generates useful content from that model. You are to write a syntax analyzer using the *flex* tool for lexical analysis and the *bison* tool for parsing.

The goal of the program is to parse an input file that contains postal addresses and then to output a representation of those addresses in XML.

Usage:

scanner < *infilename*	(Lexical Scanning Only)
parser < *infilename*	(Lexical Scanning and Parsing)

Your program may be called with one of two different names (use UNIX symbolic links to create these names).

If your program is called with the name “scanner”, then it should only provide lexical analysis of the *infilename* file. In this case, the output should be a list of tokens along with any meta-information about the tokens that may be needed. This should be sent to ***stdout***.

If your program is called with the name “parser”, then it should provide both lexical analysis and parsing of the *infilename* file. In this case, output will be sent to both stdout and stderr:

> stdout - should be a set of statements indicating whether the *infilename* file was successfully processed and should include information about bad addresses.
>
> stderr - should be an XML representation of the addresses read from the *infilename* file. If errors are encountered during the parsing of an address, you should output the best XML possible and ignore the bad information.

In parse mode, all work should be performed by the parser. Your C program should only call *yyparse*() and provide feedback on its return value.

Postal Address

Here is a Backus-Naur Form (BNF) notation of a U.S. Postal Address. This definition is a modified version lifted from the Backus-Naur Form Wikipedia page:

> http://en.wikipedia.org/wiki/Backus-Naur_Form

```
<postal_addresses>  ::=   <address-block> EOLTOKEN <postal-addresses>
                    |     <address-block>

   <address-block>  ::=   <name-part> <street-address> <location-part>

       <name-part>  ::=   <personal-part> <last-name> <suffix-part> EOLTOKEN
                    |     <personal-part> <last-name> EOLTOKEN
                    |     error

   <personal-part>  ::=   NAMETOKEN
                    |     NAME_INITIAL_TOKEN

       <last-name>  ::=   NAMETOKEN

     <suffix-part>  ::=   SRTOKEN
                    |     JRTOKEN
                    |     ROMANTOKEN

  <street-address>  ::=   <street-number> <street-name> INTTOKEN EOLTOKEN
                    |     <street-number> <street-name> HASHTOKEN INTTOKEN EOLTOKEN
                    |     <street-number> <street-name> EOLTOKEN
                    |     error

   <street-number>  ::=   INTTOKEN
                    |     IDENTIFIERTOKEN

     <street-name>  ::=   NAMETOKEN

   <location-part>  ::=   <town-name> COMMATOKEN <state-code> <zip-code> EOLTOKEN
                    |     error

       <town-name>  ::=   NAMETOKEN

      <state-code>  ::=   NAMETOKEN

        <zip-code>  ::=   INTTOKEN DASHTOKEN INTTOKEN
                    |     INTTOKEN
```

Additional Info:

`NAMETOKEN`	Regular Expression	An identifier that represents something that only contains 2 or more letters
`IDENTIFIERTOKEN`	Regular Expression	An identifier that represents something that starts with a letter or number and contains 1 or more letters or numbers.
`NAME_INITIAL_TOKEN`	Regular Expression	An identifier that represents one letter or one letter followed by a period.
`ROMANTOKEN`	Regular Expression	An identifier that represents roman numerals
`SRTOKEN`	Regular Expression	An identifier that represents "Sr."
`JRTOKEN`	Regular Expression	An identifier that represents "Jr."
`EOLTOKEN`	Regular Expression	An identifier that represents the end of line
`INTTOKEN`	Regular Expression	An identifier that represents an integer
`COMMATOKEN`	Regular Expression	An identifier that represents a comma (,)
`DASHTOKEN`	Regular Expression	An identifier that represents a dash (-)
`HASHTOKEN`	Regular Expression	An identifier that represents a hash (#)

XML Output

The XML notation should match this:

```
<FirstName></FirstName>
<LastName></LastName>
<Suffix></Suffix>
<HouseNumber> </HouseNumber>
<StreetName></StreetName>   (You should assume street names are exactly 1 word long)
<AptNum></AptNum>
<City></City>
<State></State>
<Zip5></Zip5>
<Zip4></Zip4>
```

If an item is empty, you may omit it from the output.

Example Input file

The input file should be a text file containing sets of address blocks. Each address block should be separated from the next by exactly one blank line.

NOTE: the last three addresses have errors. This is so you can test your parser's operation in the presence of errors. The goal is to completely parse the file. You may omit XML output for bad data. You may print error messages and continue parsing the file, but your parser should not stop with syntax error during parsing. yyparse() should return success and not failure.

```
F. Flinstone
5806 PebbleRd #345
Stonecity, TX 75080

W Flinstone IV
5806 PebbleRd 345
Stonecity, TX 75080-4321

Barney Rubble Jr.
106B RockyRoad
Stonecity, TX 75080-9384

Betty Rubble
106B RockyRoad
Stonecity, TX 75080

1234 Bad Name Line
9000 SomeStreet
Dallas, TX 75080-4872

B. Wayne III
BadBatStreetNumber #456
Gotham, TX 88234-3664

B. Wayne III
1000 GothamManor
Gotham, TX ABCD
```

Example Output in Scanner Mode

```
/Program4} ./scanner < input.txt

Operating in scan mode

yylex returned NAME_INITIAL_TOKEN token (F.)
yylex returned NAMETOKEN token (Flinstone)
yylex returned EOLTOKEN token (267)
yylex returned INTTOKEN token (5806)
yylex returned NAMETOKEN token (PebbleRd)
yylex returned HASHTOKEN token (265)
yylex returned INTTOKEN token (345)
yylex returned EOLTOKEN token (267)
yylex returned NAMETOKEN token (Stonecity)
yylex returned COMMATOKEN token (266)
yylex returned NAMETOKEN token (TX)
yylex returned INTTOKEN token (75080)
yylex returned EOLTOKEN token (267)
yylex returned EOLTOKEN token (267)
yylex returned NAME_INITIAL_TOKEN token (W)
yylex returned NAMETOKEN token (Flinstone)
yylex returned ROMANTOKEN token (262)
yylex returned EOLTOKEN token (267)
yylex returned INTTOKEN token (5806)
yylex returned NAMETOKEN token (PebbleRd)
yylex returned INTTOKEN token (345)
yylex returned EOLTOKEN token (267)
yylex returned NAMETOKEN token (Stonecity)
yylex returned COMMATOKEN token (266)
yylex returned NAMETOKEN token (TX)
yylex returned INTTOKEN token (75080)
yylex returned DASHTOKEN token (268)
yylex returned INTTOKEN token (4321)
yylex returned EOLTOKEN token (267)
yylex returned EOLTOKEN token (267)
…
```

Example of stdout in Parser Mode

```
Program4} ./parser < input.txt 2> output.txt

Operating in parse mode

Bad name-part ... skipping to newline
Bad address_line ... skipping to newline
Bad location_line... skipping to newline

Parse Successful!
```

Example of stderr in Parser Mode

```
<FirstName>F.</FirstName>
<LastName>Flinstone</LastName>
<HouseNumber>5806</HouseNumber>
<StreetName>PebbleRd</StreetName>
<AptNum>345</AptNum>
<City>Stonecity</City>
<State>TX</State>
<Zip5>75080</Zip5>

<FirstName>W</FirstName>
<LastName>Flinstone</LastName>
<Suffix>IV</Suffix>
<HouseNumber>5806</HouseNumber>
<StreetName>PebbleRd</StreetName>
<AptNum>345</AptNum>
<City>Stonecity</City>
<State>TX</State>
<Zip5>75080</Zip5>
<Zip4>4321</Zip4>

<FirstName>Barney</FirstName>
<LastName>Rubble</LastName>
<Suffix>Jr.</Suffix>
<HouseNumber>106B</HouseNumber>
<StreetName>RockyRoad</StreetName>
<City>Stonecity</City>
<State>TX</State>
<Zip5>75080</Zip5>
<Zip4>9384</Zip4>
```

```
<FirstName>Betty</FirstName>
<LastName>Rubble</LastName>
<HouseNumber>106B</HouseNumber>
<StreetName>RockyRoad</StreetName>
<City>Stonecity</City>
<State>TX</State>
<Zip5>75080</Zip5>

<HouseNumber>9000</HouseNumber>
<StreetName>SomeStreet</StreetName>
<City>Dallas</City>
<State>TX</State>
<Zip5>75080</Zip5>
<Zip4>4872</Zip4>

<FirstName>B.</FirstName>
<LastName>Wayne</LastName>
<Suffix>III</Suffix>
<City>Gotham</City>
<State>TX</State>
<Zip5>88234</Zip5>
<Zip4>3664</Zip4>

<FirstName>B.</FirstName>
<LastName>Wayne</LastName>
<Suffix>III</Suffix>
<HouseNumber>1000</HouseNumber>
<StreetName>GothamManor</StreetName>
<City>Gotham</City>
<State>TX</State>
```

Deliverables

You must submit your homework through ELearning. You must include a tarball that contains your Makefile, .h, .c, .l, .y, and any other important files. All source files and Makefiles need to have your name, email, and course number commented at the top. You also need to capture and submit the output of your program by redirecting it to appropriate files.

Additional Info

You may assume that people do not have full names that are one letter long.
You may assume that people do not have names that are spelled like roman numerals.
You may assume that all streets are exactly one word long.

Notes

Your code must be compiled using the **--Wall** compiler flag and must product no errors/warnings.

No late homework is accepted.

Must compile and run on cs1.utdallas.edu