Class 06: A Hand-Coded Lexical Analyzer

Held: Wednesday, 7 September 2011

Summary: We explore how one might write a lexical analyzer by hand.

Related Pages:
- EBoard.

Notes:
- Remember: The Exploring Compilation lab is due at 11 p.m. tonight.
- No reading for Friday.
- Today’s Pascal code should be in Examples/HW1/Solutions.
- Today’s lexing code should be in Examples/LA.
- EC for TK on Alice this Thursday.
- I have started to grade Assignment 1 and hope to return it on Friday. We will talk about it a bit in class today.

Overview:
- Detour: The Pascal Homework.
- Hand-Coding a Lexer.
- Lessons.

Detour: The Pascal Homework

Correctness, Issue One

- An unacceptable comment: "I’m sorry that I’m turning in crap, but I had problems with the assignment."
- An almost as unacceptable comment: "I’m sorry that this is late, but I’d rather turn in correct code late than incorrect code on time."
- Your goal should be to get correct code in on time, which means that you should start sufficiently early to allow yourself to get correct code in on time.

Tarballs

- I don’t need compiled executables or .o files in your tarballs. Don’t include them.
- I gather all of your tarballs in one place. Please give them names that let me distinguish them.
- I untar your tarballs in that same place. Try to make sure that they untar to something the identifies you.
- For those who are confused about
Create the source files and make sure that they are working.
Create a new directory
mkdir \texttt{username.HW\textunderscore i}
Copy the appropriate files into that directory.
cp filename\_1 ... filename\_n username.HW\textunderscore i
Tar it all together.
tar cvzf \texttt{username.HW\textunderscore i.tar username.HW\textunderscore i.tar}

\textit{Style}

\begin{itemize}
\item You know how to write good programs. That knowledge should translate from language to language.
\item Name variables clearly.
\item Include comments that document the purpose of variables and procedures.
\item Include comments that document design decisions.
\item When you find yourself rewriting common code, factor it out into a separate procedure.
\item Good indentation helps the reader. Bad indentation hinders the reader.
\item $\leq 80$ Columns!
\end{itemize}

\textit{Correctness, Revisited}

\begin{itemize}
\item The first four Quicksort algorithms I saw were all buggy. We’ll take a look at one.
\end{itemize}

\textbf{Writing Lexical Analyzers by Hand}

\begin{itemize}
\item Even though I prefer lexical-analyzer generators, it may be useful to take a quick look at how we might hand-code a lexical analyzer.
\item Let’s think about how we might write a C program to do lexical analysis for a simple language.
  \begin{itemize}
  \item Numbers are sequences of digits.
  \item Words are sequences of characters and digits.
  \item Whitespace between tokens ignored (but you cannot have whitespace in the middle of a token).
  \item Comments are in the standard C slash-star-slash format.
  \item The operators $+,-,\ast,/,$ are permitted.
  \item Everything else is invalid.
  \end{itemize}
\item The goal of the program is to be as efficient as possible. Hence, we don’t want to reread a character if we don’t have to.
\item Let’s start by designing the interface (which we’ll store in a \texttt{.h} file)
  \begin{itemize}
  \item What does a token look like?
  \item What does next\_token look like?
  \item What other operations do we provide?
  \end{itemize}
\item Here’s a start of something in pseudocode
\end{itemize}
token readToken
    skip over whitespace and comments
    if the next character is a digit, read and return a number
    if the next character is a character, read and return a word
    if the next character is an operator, return it
    otherwise, crash and burn

- We’ll think about the various parts in more detail

**Lessons**

- What did you learn?