Tree Editor DISP

Kenneth Micklas

May 13, 2016

Introduction

Today, we program by writing and editing text. A variety of tools try to extract meaning from this text, both to assist with our editing (e.g. syntax highlighting, automatic indenting), and to transform into some final product (e.g. evaluation, compilation). Many have tried to integrate all these tools into one system, creating an Integrated Development Environment. In doing so, they attempt to allow more cooperation between the tools (e.g. allowing compiler errors to influence syntax highlighting), and de-duplicate common infrastructure such as parsers needed by multiple tools. But there is still a curious redundancy in the itself use of almost all tools, whether they are integrated or not. The programmer thinks about the types of constructions the language allows, which form a tree, serializes it to to a string of characters, and then all these tools parse the string into a very similar tree. We have a useless round-trip to text and back.

With this in mind, I and John Ericson decided to create an editor which works directly on the underlying tree structure of program code. I completed the majority of the work in the fall semester with some assistance and advising from him and we switched roles as he continued working for his capstone in the spring.

Implementation

Like every project, we wish our tree editor to be easily and widely accessible. For us, this is even more important to balance how foreign the tool itself

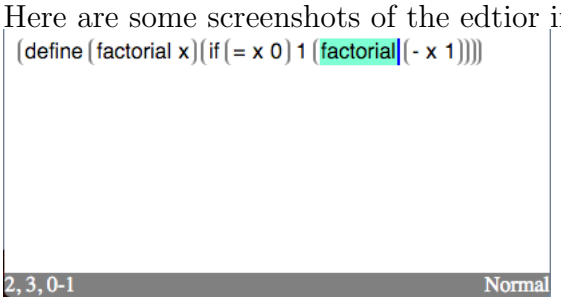
will be to most users. "Native" GUI libraries are often baroque, and, by definition, platform-specific. The web has won the GUI war with respect to these goals, so we decided to make a web-app. Since it is hard to write large Javascript applications directly, we used GHCJS to write our editor in Haskell and compile it to Javascript to run in the browser. This allowed us to use a powerful and expressive language which we are familiar in while still achieving portability and convenience benefits of the web.

Features

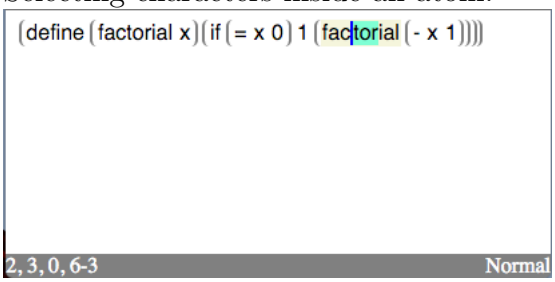
- Fully structured edits at all times
- An insert mode for efficient input
- A normal mode for efficiently performing complex edits
- Automatic indentation and line breaking to keep complex trees readable
- Compact visual representation inspired by LISP's S-expressions
- Almost every imaginable navigation command
- A diverse set of editing commands such as delete, wrap, unwrap, reverse, split, join, etc.
- Standard cut/copy/paste clipboard support
- A status line displaying the location of the selection in the tree, and the current mode

Display

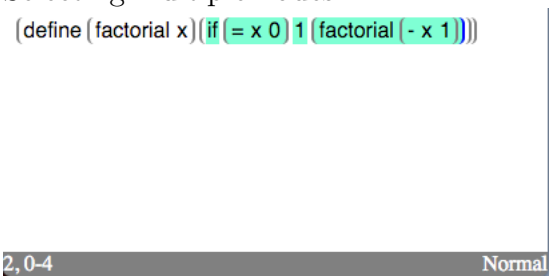
Here are some screenshots of the edtior in action:



Selecting characters inside an atom:



Selecting multiple nodes:



Smart indentation and line breaking when lines get long enough: