CAREER (Comprehensive Automated Resume & Employment Experience Renderer) Language Specification

Harry Albert

May 5, 2024

Introduction

Creating a beautiful resume is often tedious and time consuming, but this annoying task is unfortunately extremely important. Making your resume look professional and easily readable can be the difference between a recruiter considering you for a job and throwing your application away. Formatting a resume well entails taking the time to organize your experience, skills, and interests in an aesthetically pleasing way using an often uncooperative word editing software. This means that making a resume can be both time consumign and painful, while still mandatory for career goals.

CAREER solves this issue by automatically formatting your information for you, taking as input some simple text and outputting a well designed resume. CAREER will always output a visually pleasing and readable product, without the fuss and frustration of actually formatting everything yourself. This also allows for the easy modification of one's resume without the need to go back and reformat everything. In summary, CAREER saves time by making resume creation a breeze, democratizing quality resume creation by putting a well formatted resume at anyone's fingertips.

Design Principles

The main goal of CAREER's design is to be easily readable and modifiable by the user. Any user who has skimmed over CAREER's documentation should be able to easily read and understand a resume written in CAREER. Users should be able to go back to their resume document after not looking at it for a year and quickly find and update whatever has changed in that time. This means that all of CAREER's commands and functions should be easily readable and understandable, ideally with little reference to the documentation. The other main design principle guiding CAREER is conciceness. One of the problems CAREER is solving is that resume writing is tedious and time consuming. Requiring the user to write lots of boiler plate code to create a CAREER resume would not fix this problem.

Examples

Example 1:

This example program shows how one could create a complete, basic resume using CAREER. This resume includes a title, contact information, and multiple unique sections. Each of these sections has multipe titles with different alignments, as well as bullet point items. While this resume is slightly long, it is easily readable and modifiable. It is easy to imagine how a resulting resume would look based on this code.

```
*HEADER Harry Albert
*SUBHEADER (111)111-1111
*SUBHEADER hsa1@williams.edu

*EDUCATION
*TITLE_LEFT Williamstown, MA
*TITLE_CENTER Williams College
*TITLE_RIGHT Fall 2021 - May 2025
```

*ITEM Teaching Assistant for introductory and upper-level CS courses
*ITEM Activities include PAC Comedy Club, Opinions Columnist for the Williams Record Newspaper

*EMPLOYMENT
*TITLE_LEFT Software Engineer, Intern
*TITLE_CENTER Bubble
*TITLE_RIGHT Summer 2023
*ITEM Bubble is a no-code web development platform. I worked on the Growth Team
*ITEM Contributed to multiple successful experiments, including improving the editor UI and user journey

---

Example 2:

This example program is similar to the previous program. However, it adds on the idea of generic sections. Having built in commands for common sections like employment and education is great, but there should still be room for the user to create some of their own generic sections. This idea of generic types can and should be extended to other aspects of the CAREER language.

*HEADER Harry Albert
*SUBHEADER (111)111-1111
*SUBHEADER hsa1@williams.edu

*SECTION Languages & Technologies
*ITEM Javascript/Typescript, Python, HTML + CSS, F#, Java, C
*ITEM React, Scikit-Learn, PyTorch, SolidJS, REST APIs

Example 3:

This example program introduces the idea of brackets as limiters on the skoper of modifiers. Only the words within the brackets are bolded under the projects section.

*HEADER Harry Albert
*SUBHEADER (111)111-1111
*SUBHEADER hsa1@williams.edu
\\ TODO: add in website link here
*SECTION Languages & Technologies
*ITEM Javascript/Typescript, Python, HTML + CSS, Java, C \\ remember to add on F# when I'm done with PL
*ITEM React, Scikit-Learn, PyTorch, SolidJS, REST APIs

*SECTION Projects
*ITEM *BOLD [ML Lung Cancer Prediction Tool]: Created a neural network to predict lung cancer based on several easily self-diagnosable conditions. Published website where users can interact with this model to democratize access and allow users to play with the model. \\ Remember to bring this up in job interviews as an example of ML experience
*ITEM *BOLD [Booktrak]: Independently designed and created Booktrak for WSO (student run website for Williams College). Booktrak is a platform where Williams students can buy and sell used books, and link their book listings to specific classes in order to better facilitate used book sales.

Language Concepts

The core concept of this language is that there are modifiers and text. Text is a primitive, and it is defined as any string that is not a modifier. For example, in the line "HEADER Harry Albert," "Harry Albert" is the text. Modifiers are (conventionally all caps) functions, indiciated by *, that change how text is displayed. For example, the SECTION modifier bolds the text that follows it and adds a horizontal line above it to indicate a new section. Ideally, users will be able to create their own modifiers to make the language more groweable. The ITEM function is a special modifier function that can only be applied to the start of a line. Finally, modifiers can be stacked. For example, ITEM creates a bullet pointed text, and BOLD (used within an item) creates a bolded piece of text. When brackets are included after a modifier, the modifier will apply only to the text included in the brackets. If brackets are not included, the modifier will apply to all text until a new line.

Formal Syntax

<expression> ::= <line>*
<line> ::= <line_content> \n
<line_content> ::= *ITEM␣<formatted_text>*
| *SUBSECTION_TITLE ␣<subsection_content>*
| <formatted_text>*
<subsection_content> ::= <string> — [<string>] | [<string>][<string>] | [<string>][<string>][<string>]
<formatted_text> ::= <modifier> | <string>
<modifier> ::= *<mod_fun>␣[<formatted_text>*] | <mod_fun>␣<formatted_text>*
<mod_fun> ::= HEADER
| SUBHEADER
| SECTION
| BOLD
| UNDERLINE
| LINK

<string> ::= <str_char>*
<str_char> ::= $c \in \{\text{all characters}\} \wedge c \notin \{\backslash n, *, [,]\}$

Semantics

String

Syntax: <string>
Abstract Syntax: String of string
Prec./Assoc.: 6/left
Meaning: A string is, as you might expect, a string. It can contain any characters besides \n, *, [, and]. Each of these characters is used for a specific purpose in CAREER and thus cannot be used in basic strings.

Modifier

Syntax: *<mod_fun>␣[<formatted_text>*]|<mod_fun>␣<formatted_text>*
Abstract Syntax: Modifier of String * (FormattedText list)
Prec./Assoc.: 5/left
Meaning: A modifier stores some modifying command and a list of formatted texts to apply this modifying command to. Each of these formatted texts must eventually evaluate to a string, and each of the strings in the formatted text list will be combined into one final string. This string will have the modifying function applied to it in the corresponding latex code. For example, the modifying function BOLD applies \textbf to all of its inner formatted texts.

Item Modifier

Syntax: *LINE ␣<formatted_text>
Abstract Syntax: Modifier of string * (FormattedText list)
Prec./Assoc.: 3/left
Meaning: Item is a special modifier that puts an entire line into bullet point format. Because of this, Item (unlike other modifiers) cannot be applied anywhere within a line; it can only be applied at the start of a line. Thus, the line modifier has its own, special syntax that is taken into account by the Line combining form.

Subsection Title Modifier

Syntax: *SUBSECTION_TITLE ␣<string> | 1-3 * [<string>]
Abstract Syntax: Modifier of string * (String list)
Prec./Assoc.: 4/left
Meaning: The subsection title modifier is another special modifier. This modifier can also only be applied at the beginning of a line. It takes one to three arguments, all of which must be Strings. These arguments must be surrounded by brackets to deliniate between each argument. If there are no brackets, CAREER will assume the entire string after the modifying function is one argument. Given one argument, CAREER centers the given text, bolded and underlined, on a line. Given two arguments, the modifier places the first argument bolded on the left side of the line and the second argument bolded on the right side. Finally, given three arguments, CAREER places the first and third argument bolded on the left/right side of the line, and the second argument bolded and underlined in the middle.

Formatted Text

Syntax: <modifier> | <string>
Abstract Syntax: FormattedText of Modifier or String
Prec./Assoc.: 2/left
Meaning: FormattedText is essentially a container that can store either Modifiers or Strings. This is a necessary combining form because Modifiers must be able to be applied to other modifiers. Thus, modifiers are applied to lists of formatted texts, and each formatted text can store either a modifier or a string.

Line

Syntax: *ITEM␣<formatted_text>* \n
| *SUBSECTION_TITLE ␣<subsection_content>* \n
| <formatted_text>* \n
Abstract Syntax: Line of FormattedText list
Prec./Assoc.: 1/left
Meaning: One line in CAREER represents one input line in the given text file and one output line in the resulting tex/pdf file. A line is simply a list of formatted texts, meaning a list of modifiers and strings. A line is stopped by the new line character, and outputs a string with a new line character at the end.

Expression

Syntax: <line>*
Abstract Syntax: Expression of Line list
Prec./Assoc.: 1/left
Meaning: An expression is simply a list of lines. Each CAREER ast is composed of one expression. This one expression holds all of the lines that were given as an input and will eventually be outputted as formatted latex.