Mobile Order Meal Generator

Maddy Andersen Stella Oh

May 13, 2024

Introduction

What problem does your language solve? What makes you think that this problem should have its own programming language?

While Williams has multiple dining halls, sometimes it can be more convenient to instead order from one of the three mobile ordering options: Lee Snack Bar, '82 Grill, or Fresh n Go. But with this added convenience comes a daunting question: what exactly should you order? Each of the three has an extensive menu, with each of the options on the menu giving the opportunity for further customization. Because there are so many choices, ordering can be overwhelming. Many people — ourselves included — stick to ordering the same meals again and again as a result. Even worse, some people don't mobile order at all and instead go to one of the dining halls, feeling that a mediocre meal is better than having to make the decision about what to mobile order.

In an effort to help out indecisive individuals and those experiencing decision fatigue, our language generates a possible mobile order, following given specifications from the user about what day of the week it is, as well as about what kind of meal they are looking for. Designed for students who are already familiar with the three menus, our language will help users mix up their order and find new options. It will also hopefully re-inspire students to mobile order their meals if they are unhappy with the dining hall options. Because our language is (ideally) easy to use, using our language to generate a meal will be less burdensome — and more fun! — for a user than determining what to order on their own.

Design Principles

Languages can solve problems in many ways. What are the aesthetic or technical ideas that guide its design?

This language is designed with simplicity and convenience in mind. It is tailored for the typical Williams Eph, who is already familiar with the types of mobile ordering meals available (they know '82 Grill serves only lunch, dinner, and late-night, for example), but are looking to switch up what they order. Because this language is intended to be an alternative to the overwhelming task of deciding what to mobile order, it aims to be as straight-forward and stress-free as possible.

Examples

Keeping in mind that your syntax is still informal, sketch out 3 or more sample programs in your language.

```
tuesday lunch any

friday dinner lee

monday dinner fresh n go
tuesday breakfast 82 grill
tuesday late night lee
```

Currently, our interpreter prettyprints the input and generates a random meal category from the given location for the given meal for each order. If the input is invalid — for example, it is not possible to order dinner from lee — our interpreter does not generate a random meal category and gives the user a message to submit a new request that is valid.

Language Concepts

What are the core concepts a user needs to understand in order to write programs? Think in terms of both "primitives" and "combining forms." What are the key ideas and how are they combined?

The user will need to be familiar with the components that go into an order (combining form), which are the day of the week, the type of meal, and the location (primitives). To submit their request (which consists of one or more orders and is another combining form), the user will have to write their order(s) in a text-file, which they can then give to the language. The user will have to know the format of each order (<day> <meal> <location>) to submit a valid request. They will also have to know that each order should begin on a new line. Finally, a user needs to have some familiarity with the three mobile ordering options on campus, as well as a general sense of their menus, to be able to use this language easily.

Formal Syntax

Begin to describe your syntax formally, using BNF.

The following grammar is for a mostly working version of our language.

```
<request>  ::= <order> | <order> '\n' <request>
<order>    ::= <day> <meal> <location> <category>
<day>      ::= monday | tuesday | wednesday | thursday | friday | saturday | sunday
<meal>     ::= breakfast | lunch | mid day | dinner | late night
<location> ::= lee | fng | 82grill | any
<category> ::=
    | <lee_breakfast_categories>
    | <lee_midday_categories>
    | <lee_lunch_categories>
    | <lee_dinner_categories>
    | <fng_lunch_categories>
    | <82grill_lunch_categories>
    | <82grill_dinner_categories>
    | <82grill_latenight_categories>

<lee_breakfast_categories> ::= breakfast entrees | breakfast sandwiches
<lee_lunch_categories> ::= burgers | hot sandwiches | breakfast sandwiches |
    GF burgers | GF hot sandwiches | salads | parfait | specials
<lee_midday_categories> ::= burgers | hot sandwiches | breakfast sandwiches |
    GF burgers | GF hot sandwiches | salads | parfait | specials
<lee_dinner_categories> ::= burgers | hot sandwiches | breakfast sandwiches |
    GF burgers | GF hot sandwiches | salads | parfait | specials

<fng_lunch_categories> ::= build your own | protein rich | GF

<82grill_lunch_categories> ::= create your own | GF | wings | specials
<82grill_dinner_categories> ::= create your own | GF | wings | specials
<82grill_latenight_categories> ::= create your own
```

Semantics

How is your program interpreted? This need not be formal yet, however, you should demonstrate that you've thought about how your program will be represented and evaluated on a computer.

1. What are the primitive kinds of values in your system? For example, a primitive might be a number, a string, a shape, or a sound. Every primitive should be an idea that a user can explicitly state in a program written in your

language.

In our language, the primitive values are the day of the week, meal, and location. The day can be one of seven options: monday, tuesday, wednesday, thursday, friday, saturday, or sunday. The meal can be one of five options: breakfast, lunch, mid day, dinner, or late night. The location can be one of four options: lee, fresh n go, 82 grill, or any location.

2. What are the combining forms in your language? In other words, how are values combined in a program? For example, your system might combine primitive "numbers" using an operation like "plus." Or perhaps a user can arrange primitive "notes" within a "sequence."

   Our language has two combining forms. The first is an order, which is composed of a day, meal, and a location. For example,

   ```
   monday breakfast lee
   ```

   is an order. The second is a request, which consists of one or more orders. So,

   ```
   monday breakfast lee
   ```

   is thus also a request, while

   ```
   monday breakfast lee
   friday late night 82 grill
   ```

   is a request too.

3. How is your program evaluated? In particular,

   (a) Do programs in your language read any input?

   (b) What is the effect (output) of evaluating a program? Does the language produce a file or print something to the screen? Use one of your example programs to illustrate what you expect as output.

   Our program reads in a text file containing the user's request (one or more orders). If the user does not provide a file, the program prints a usage message. If the file does not contain a valid request, the program prints "Invalid program." Otherwise — if the user provides a file with a valid request — our language currently prettyprints the input and generates a random category from the given location for the given meal for each order in the request. In the future, evaluating a program will output a random order (from that category) that takes into account the user's specifications.

Syntax	Abstract Syntax	Prec./Assoc.	Meaning
<meal>	Discriminated Union	n/a	*meal* is a primitive that represents a specific meal of the day (breakfast, lunch, mid day, dinner, late night). To prettyprint a meal, the given meal is turned into a string.
<order>	Record	n/a	*order* is a combining form. It has three elements: the day (of type *day*), the meal (of type *meal*), and the location (of type *location*). This allows us to store all three of these pieces of information together.