1. Introduction

For the last few decades developers have created a variety of tools to help with data management and analysis. SQL and R language, for example, are two of the most prominent tools in this field, being widely utilized by business analysts, social scientists, statisticians and many more. Popular as they are, there still remain certain scenarios where both tools are imperfect. For instance, social scientists today often need to incorporate data of distinct formations from different sources to conduct their statistical analysis. SQL may work well for this job, yet it is designed for relational databases, a system not exactly suitable for most statistical tasks. R language, on the contrary, is fully optimized for statistical analysis with tons of functions and libraries, yet it still lacks the simplicity and usability of SQL in terms of managing datasets.

Data Analytics Visualization with Ease (DAVE) is a programming language optimized for data retrieval, manipulation, analysis and visualization. DAVE is designed with cross-dataset operations in mind, which is fairly common yet complicated to achieve with current toolsets in today's data-intensive tasks. The primary goal of DAVE is to allow the programmers to validate datasets, incorporate (parts of) datasets from different sources, split up oversized datasets, conduct statistical analysis and visualize critical data.

2. Tutorial

2.1 Environment Setup

DAVE language needs **the GNU Compiler Collection 5** (GCC5) or later to function, as it compiles to C++. The additional helper scripts included execute only in an environment with **Bash** and the **make** command could greatly help with the process of compiling the DAVE source code files to DAVE compiler. However, both **Bash** and **make** command is not required for DAVE itself.

Also, you may want to:

- Put the DAVE Standard Library files (*dave.h*, *dave.hpp, dave_io.hpp*) under the same path of your DAVE program
- Add the DAVE binaries to your **PATH** environment variable

2.2 Compiling and Running DAVE Programs

The Standard DAVE Package comes with the following source code files (under */src* folder, in alphabetic order):

ast.ml	astcompile.ml
compile.ml	dave_io.hpp

dave.h	dave.hpp
dave.ml	Makefile
Makefile.ocaml	parser.mly
sast_to_cpp.ml	sast.ml
scanner.mll	semantic_expr.ml
semantic.ml	semanticAnalysis.ml
semanticExceptions.ml	

You need to compile the source code files to DAVE compiler first before starting to compile and run DAVE programs. Run the following commands:

```
make
```

Suppose the source code file of your DAVE program is named *my_program.dave*, the following commands would help you compile your DAVE program to executable:

```
./dave -c <my_program.dave

g++ dave.cc
```

Two helper scripts are packed into the Standard DAVE Package, allowing easier compilation and running of DAVE programs. *ezcompile.sh* will prompt an input of the name of your DAVE source code file and offer options to compile it to C++ source code or executable. *ezrun.sh* will prompt an input of the name of your DAVE source code file, compile it to executable and run it immediately.

```
./ezcompile.sh
```

```
./ezrun.sh
```

3. Language Reference Manual

<Insert Language Reference Manual Here>

4. Project Plan

4.1 Overview of Development Process

Planning

The Planning of DAVE language starts at the very beginning of this semester. The ideas and expectations come from our own experiences of processing and analyzing data, from which we feel that a tool with both the clarity of SQL language and the power of general programming languages would be most helpful. With the help of Ms. Prachi Shula and Prof. Edwards, our planning was further refined throughout the development process, focusing more on DAVE's capability of implementing complex algorithms.

Specification

Most tasks in the specification phase of DAVE language starts after the submission of language proposal. Similar to the planning phase, our specification of DAVE keeps developing throughout the process. DAVE started out as a language akin to C and compiles to python, however in the end we took the suggestions from Ms. Prachi Shula and let DAVE compile to C++. The concept of table, columns and rows, which is the core of DAVE language, also evolved considerably from our initial perspective.

Development

Scanner and Parser begin earliest in the development phase. A minimally working semantics analyzer and C++ code generator are implemented afterwards. We keep expanding and refactoring their codes as the requirement grows from simple arithmetics to control flows and then to self-defined functions and much more. The standard library of DAVE is also developed along with the process, in which we keep adding new functions and variables when they become necessary. A review session is added at the end, making sure that the codes remain consistent and all the components are functioning properly together.

Testing

Basic tests are added when they become possible. More general and comprehensive tests are implemented and completed after most of the development tasks are finished. A testing script is available in the package for verifying the code.

4.2 Programming Style Guide[1]

> *Despite the best effort we have made, due to the limited resources and time we have as a team, some of our codes may not be fully compliant to the style guide.*

- Indentations comprises of four spaces, and are implemented with spaces instead of tabs.

[1] We referenced a JAVA Programming Style Guide (http://www.javaranch.com/style.jsp) for our own DAVE Programming Style Guide.

- Method names are followed immediately by a left parenthesis.
- Array deferences are followed immediately by a left sure bracket.
- Binary operators should have a space on either side.
- Unary operators are followed/preceded immediately by their operand.
- *if, while, for* and *switch* are followed by a space.
- Avoid making a line too long.
- The braces should locate as the following:

```
type t_expr = {

(* Some Code Here *)

}
```

Instead of

```
type t_expr =

{

(* Some Code Here *)

}
```

- Adopt the *lowerCamelCase*.
  - The first letter of each word in the name will be uppercase, except for the first letter of the name. All other letters will be in lowercase.
- Start the comment in a new line
- Self-documenting code.
- Avoid long names in general
- Avoid long parameter lists in general

4.3 Project Timeline

Date and Time	Milestone
Sep. 30th	Language Proposal Completed
Oct. 26th	Language Reference Manual Completed
Oct. 28th - Nov. 9th	Scanner and Parser Completed
Nov. 10th - Nov. 15th	Minimally Working Build of DAVE Compiler Completed
Nov. 16th	Hello_World Presentation
Nov. 17th - Dec. 14th	Expanding Features
Dec. 15th - Dec. 20th	Review and General Testing
Dec. 22nd	Final Report Completed

4.4 Roles and Responsibilities

Name	UNI	Roles	Responsibilities
Hyun Seung Hong	hh2473	Manager	Coordinates within the team
Min Woo Kim	mk3351	Language Guru	Implementing the DAVE language
Fan Yang	fy2207	System Architect	Define the Architecture of the DAVE language
Chen Yu	cy2415	Tester	Testing the Components
	Some of the responsibilities are shared among members due to contingencies		

4.5 Software Development Environment

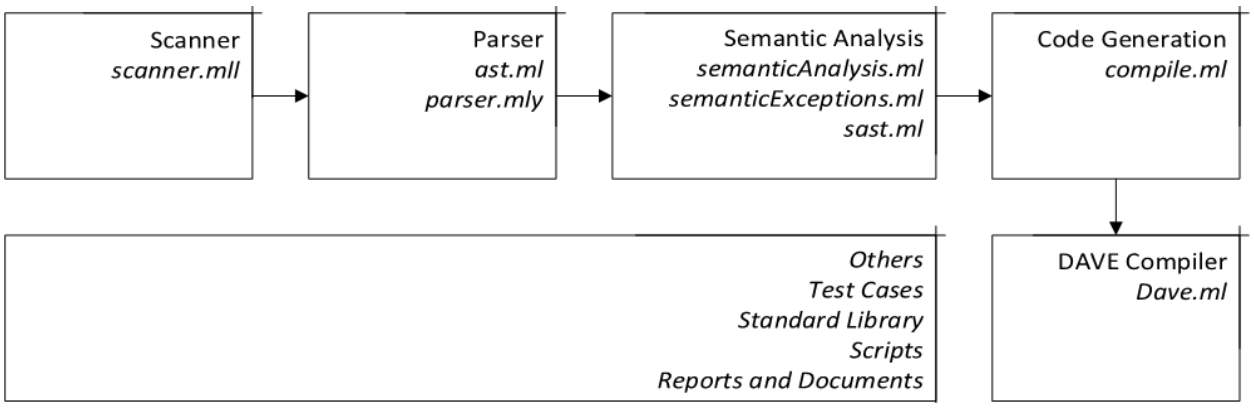
The DAVE project is developed on Mac OS X 10.10 (Yosemite) and Mac OS X 10.11 (El Capitan). Source codes and documents are controlled using Git and hosted on github.com. The compiler and related test cases is created in OCaml and completed with Sublime Text 3. Helper scripts are coded with TextEdit.

4.6 Project Log

<Insert Project Log Here>

5. Architecture

5.1 Architecture Overview



Scanner created by the whole team

Scans the source code and generates a series of tokens. Whitespace and comments are removed; invalid characters/words are reported.

Parser created by the whole team

Parses the sequence of tokens generated by the scanner. Outputs an **AST** (**Abstract Syntax Tree**). Syntax errors are examined and reported.

Semantic Analysis created by Hong, Kim and Yang

Parses the **AST** and conduct the semantics check. Outputs a **SAST** (**Semantically-Checked Abstract Syntax Tree**)**.** Semantic errors and type inconsistencies are reported.

Code Generation created by Hong, Kim and Yang

Generate the C++ code based on **SAST**.

Others Standard Library created by Kim, Test Cases + Scripts + Reports created by Yu

Standard Library includes functions and variables that could be referenced in DAVE programs.
Test Cases include all the tests.
Scripts help with the automatization of DAVE programming.
Reports and Documents provides necessary references.

6. Test Plan

<Insert Test Plan Here>

7. Lessons Learned

Trying to create a programming language of our own is an eye-opening experience, not only because of all the new ideas and theories involved in the process, but also for the fact that it grants us an unique opportunity to reexamine a lot of tools we take for granted in everyday software development. With today's convenient integrated development environments, we tend to forget about all the delicacies under the hood, and building something from bottom to the top does have the magic to make people recollect these hidden little wonders.

DAVE is by no means easy to build. All of us have other tasks to worry about throughout the semester and it is really difficult to coordinate efforts. Worst of all, contingencies happen, and we are forced to figure out a way to fill things up. Had we granted another chance, we might be able to plan things better ahead and finish more than we are doing right now. However, it is exactly from those kinds of experiences that we begin to learn and grow, for which we are deeply thankful for.

Appendix

<Include Appendix Here>