**Computer Science Department**

**Senior Team Portfolio**

Compilers

CSCI 468

Spring Semester of 2019

by

Spencer Debuf, Gregor Smirnow, & Kenneth Taylor

Montana State University

Bozeman, Montana

**Section 1: Program**

*Attach the source listing of the program that you wrote for your capstone course (CSCI 468 or CSCI 483). Include the specifications for the program.*

**Section 2: Teamwork**

The team for this capstone project consisted of three people. While group projects are notoriously frustrating things that make some swear to never take classes that include this type of class work, it was quite the opposite for this project. All members started each of the assignments early on and met, if only briefly, at least once a week to see where each of the other members were with their part and was needed before the next meeting. One of the group members was more experienced with documentation writing and therefore took over the vast majority of writing each of the reports and this portfolio. This same member was a little slower with the coding aspect of the project but helped out design ideas, trouble-shooting, and debugging. The other two members of the group then pretty much cut the coding portion of the project in half and worked together to complete each section. As a group we used GitHub to manage the contributions from each of the members and merge the project into a single working venture. The member that did most of the documentation was also more experienced with using git and was able to help out in managing branching and merging bits and pieces of the project and helped teach this tool to the other members. Overall, the feeling was that the project was pretty much split into thirds and shared equally.

**Section 3: Design Pattern**

*Identify one design pattern that was used in your capstone project and describe exactly where in the code it is located. Highlight the design pattern in yellow. Explain why you used the pattern and didn’t just code directly.*

Define the factory pattern.

In [class-based programming](https://en.wikipedia.org/wiki/Class-based_programming), the **factory method pattern** is a [creational pattern](https://en.wikipedia.org/wiki/Creational_pattern) that uses factory methods to deal with the problem of [creating objects](https://en.wikipedia.org/wiki/Object_creation) without having to specify the exact [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a [constructor](https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)).

**Section 4: Technical Writing**

*Include the technical document that accompanied your capstone project.*

**Section 5: UML**

The size of this project didn’t really lend itself to creating a UML diagram. While diagramming can be an important part of a project that will span several classes and be an undertaking by many developers, we were able to discuss our project and stick to a design without the use of this tool.

**Section 6: Design Trade-offs**

Due to time constraints, a few of the classes started to follow the God Class Anti-Pattern. This wasn’t exactly a conscious decision to write code this way but happened as quick fixes and trial runs of testing. These coding snippets were slotted for cleanup at a later date that never seemed to come along while the snippets grew in numbers. Suddenly, it seemed, this technical debt was too large to deal with before the end of the project.

**Section 7: Software Development Life Cycle Model**

We decided to use an agile approach to the project since we consisted of a small working group. This scrum approach worked well for us. Since class met three times weekly we would take just a few minutes after class to hold brief sprints to see what each of the others had accomplished and where we needed to go as a group for lab each week. This approach kept the lines of communication open and a sense of progress from week to week. No large contributions ever needed to be made since several small one were accomplished each sprint cycle.