January 11 – February 22, 2021

BHALACHANDRA GAJANANA BHAT

Eigenständigkeitserklärung

Mit Einreichen des Portfolios versichere ich, dass ich das von mir vorgelegte Portfolio selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Mir ist bekannt, dass Plagiate einen Täuschungsversuch darstellen, der dem Prüfungsausschuss gemeldet wird und im wiederholten Fall zum Ausschluss von dieser und anderen Prüfungen führen kann.

Declaration of Academic Honesty

By submitting the portfolio, I confirm that the submitted portfolio is my own work, that I have fully indicated the sources and tools used, and that I have identified and referenced the passages in the work - including tables and figures - that are taken from other works or the Internet in terms of wording or meaning. I am aware of the fact that plagiarism is an attempt to deceit which will be reported to the examination board and, if repeated, can result in exclusion from this and other examinations.

Reflection

1. **What was the most interesting thing that you learned while working on the portfolio? What aspects did you find interesting or surprising?**

   I had hard time in designing an optimal name table with the minimal data structure which doesn't consumes too much memory when program grows. This part made me go through the collections of Java language and helped me to gain deeper insights on them. I enjoyed working on the name analysis phase and also came across some of my funny implementations in the beginning days. It was hilarious and interesting when I found out that the Stack data structure can be traversed with iterator in Java which actually contradicts its purpose.

   In the early days of implementation, it was difficult to check the future references of methods and classes which are used before hand in the source file. The idea behind two pass assemblers helped me in reading the definition of the methods and classes in first pass. In the second pass these references were used to validate the statement and declaration of class, methods and variables.

   Debugging through the code is always fruitful as it provides more insights on the concepts along with data structures used. In one of the incidents, I was loading all the methods of superclass to child class with putAll() function of map which forcefully updates key value pair. So, every time the signature of the overridden method was matching with the parent class. This always masked the error which needs to be thrown. Later I found out that the overridden methods need to be ignored while loading the methods from parent class using foreach method of map data structure.

   The idea behind semantic analysis where the type and declarations are assigned beforehand fascinated me. The switch from programming in C++ to python made my life easier as developer. But after going through detailed procedure of compilation, it is evident that the type checking process and declaration made in this phase is one time action which saves lots of overhead during run time. This is one of the reasons which makes strongly typed programming languages like c and c++ more efficient.

2. **Which part of the portfolio are you (most) proud of? Why?**

I am happy with the implementation of return statement check and cyclic dependency of classes. Check for cyclic dependency is very important in large programs which unnecessarily could consumes lot of memory. I could have used Graph data structure for cyclic dependency check as there are already available algorithms which could easily find cycles in graph and it could be efficient for large programs.
But the graph nodes can have multiple edges which contradicts with inheritance in java. So, I stick with maps where the classes are pushed only if it inherits from other class. The algorithm which finds the inheritance cycles traverses through the map by updating the key with the parent class name which are stored as values. If the key value matches with the passed class name argument, there is a cyclic dependency in the program provided. The time complexity will be O(n) in the worst-case scenario for multilevel inheritance. This method is used while implementing overriding of methods as well. This makes use of a separate map data structure where only the classes which inherits from superclass are stored in the beginning. Hence there is no need to traverse through other classes.

Writing a recursive function is always fun. The algorithm for the return statement checks required lots of brainstorm session for me. This recursive function checks all the paths of if and else, while loop. It returns true if return statement exists. The first time when it was written there were plenty of duplicate code and many bugs. These used to miss return statements which exists in the middle of source file. But with this function it checks all the return statements in the program including the ones inside the blocks.

3. **What adjustments to your design and implementation were necessary during the implementation phase? What would you change or do different if you had to do the portfolio task a second time?**

Symbol table was designed wrongly for the name analysis part during the initial days of implementation. Every time few of the details were missed while implementing the name analysis. Every time while implementing a new feature the new design was required which would make few codes and operation redundant. For example, in the early stage of implementation separate data structure was maintained to check the uniqueness of class names but later with the usage of hash maps early check was made on this instead of verifying at the end. Later at some point I could able to design name table which covers almost all aspects of name analysis for NQJ. The design of name table gave deeper insights through the data structures and it's usages. It also helped in developing the programming skills in Java and analytical thinking. I explored many inbuilt functionalities of collections in java.

Revisiting the features like generic types in java and understanding of few of the semantic analysis concepts along with translation phase were important in the middle of the implementation phase were also necessary.

If given a second chance, I would be happy to redesign virtual method table and class structure of *Translation of Object-Oriented Structure* part. May be with the wrong design, I couldn't complete this task.

Apart from translation part, I would like to redesign the implementation of method overriding in name analysis part. The signature matching technique used here consumes lot of execution cycles because nested for loops are used to in the design which makes it slower.

4. From the lecture/course, what topic did excite you most? Why? What would you like to learn more about and why?

The concept of Intermediate representation excited me the most. It clearly explains what is going behind the scene in the compilation. Most of the times as a developer you don't bother about the compilation processes. But this part of course helps me understanding the concept of converting high level language into native machine code. As embedded systems is one of my specialization, I was able to relate few of the concepts like memory allocation and understand concepts. As I was having C++ background it helped me much better to understand the concepts behind dynamic dispatch and the way objects are internally handled at run time.

I would like to learn more about the implementation of overloading methods, Generic methods, Template classes in C++ and nested classes. As NQJ is not supporting overloading, it was easy to store functions in a hash map which maintains the uniqueness of methods. But when overloading is allowed, it's always challenging to design the data structure which could perform these operations. I would like to know how template classes are handled at compile time. It would give clear picture on the usage of same.