Christian Wendlandt

Dr. Erik Krohn

CS 431

8 May 2018

Assignment 6

* + Splitting a compiler into many parts gives it its modularity. Each part can be upgraded and changed without affecting other pieces so long as the part performs the same function. If a designer wants to improve a single stage of the compiler, they won’t need to understand every aspect of the compiler, just that of their module.
  + This also allows the compiler to be built in testable stages. Since each piece takes a tangible input and gives a tangible output, testing becomes simple. Validation becomes iterative and so it morphs a waterfall”development methodology into an agile development methodology.
  + Dividing compilation into a set of stages allows the compiler to fail fast. If errors can be caught during the fast and inexpensive stages, then time and processing power won’t be wasted on later stages that would have eventually errored out.
  + Modularity makes stages swappable. For example, an assembler for one operating system could be swapped out to target a new operating system, or a parser might be swapped with a new parser that finds keywords in a different spoken language.

The visitor pattern’s main purpose is to separate the tasks of node traversal and node implementation. On the implementation side of the pattern, the visitor performs operations on the applied children of the current node and returns it to the parent. The visitor here knows the abstract type of a given child, but doesn’t need to know about the child private data. This allows each set of node operations to be implemented without needing to know anything more about the graph’s structure. When applied, the visitor traverses the child, identifies it, and only then will it visit the child’s graph node to repeat the implementation process. This technique separates each node with a barrier of abstract method calls, and is useful when wanting to carry out node operations throughout a tree with a vast and unknown topology, where each node has its own properties and can carry out its own set of operations.

By isolating each node’s implementation with the visitor pattern, we’re able to further modularize the compiler’s design. Each set of node operations is allowed to be swapped, updated, and modified independently of all but their own individual neighbors. If, for example, a node needs to accept a new class of child nodes, then so long as the child nodes extend the same abstract class and tosses up the same expected return value, nothing in the original node needs to be updated to accommodate the new child types. Perhaps the best use for this is for language math expressions, where exponential, logarithmic, and trigonometric expressions can be easily appended to the list of basic arithmetic expressions with complete autonomy.

3.

Another design pattern that is useful for a compiler to use is the chain of responsibility pattern. If data that has been passed up to a node needs to be processed, but the node don’t know in advanced which processing method to call on the data, then you may want the node to call several processes and to just stick with the first method that accepts your data. Since node neighbors are hidden from each other by the visitor pattern, the parent node will have to pass the data along until it’s accepted and processed by a method. It may even be that this parent node is just a middleman, and doesn’t know what kind of objects it’s meant to receive or what it should do with them. In that case, deduction is impossible and deferring to a chain of responsibility is the only way to properly deal with the unknown object.

This, of course, occurs with implicit casting. While there are a countable number of primitive types that can be tested for in each node, there are an unlimited number of foreign object types that can be received from applying a child node. By using the chain of responsibility, all return data testing can be pushed onto a set of modular and mutable processing methods, and carried out by unique operations for unique objects. In this way, new data types can be added and handled without ever needing to modify the receiving parent nodes.