Jesse Uchytil

12-7-18

CIS 161 – SQL Interpreter Project

Final Writeup

When creating a software program capable of handling any situation, one must either plan for constant expansion over time or possess the ability to see into the future. As the latter is impossible for now, we are left with the less ideal former. This is a difficult situation, to say the least. Anyone with a basic understanding of how to code can create a singular program that does one thing, but to design a program that can do many things is quite difficult especially when it comes to maintenance and adding on to it in the future. This is the problem I sought to solve with this project; to implement a system that was scalable to the point that it could be used from start to finish on a program as large as a language interpreter.

Initially, I had wanted to create a simple SQL interpreter as a proof of my abilities as a programmer. I would create something that would parse some basic SQL code and use it as commands for a theoretical database program; something complex enough to warrant the time put into it yet simple enough for others to understand at a glance. However, as time wore on and I began doing more research into the viability of the project I started to become a little apprehensive towards the project. Was this really challenging enough? Sure, figuring out how to search for key phrases and parse information from a string is difficult, especially so in C++, but once you’ve figured it out once you’re good to go. Realizing this, I sought to change direction on the project.

Rather than creating a singular program for parsing a bit of SQL code, I began work on designing a scalable program that could be worked on by not just myself, but anyone who wished to work on it. Although, I am very much aware that no one will likely touch this project after I am done with it, the thought of designing with this design philosophy in mind intrigued me greatly. I had never done something like this before, and I was eager to challenge myself with it. So, after some initial research and thought, I began work on the project.

First things first, I needed to confirm what knowledge I could carry over into C++ from other languages. I’m a decently experienced Java programmer so I already had a general idea on how object-oriented languages worked, but I didn’t know the specifics of how C++ worked or even if it had the same features. Thankfully, C++ has had a few recent updates to the language that brought over some of the things in other, more modern languages. However, one thing that was a hurdle for the first half of the project’s development was getting inheritance and polymorphism to work properly. They worked mostly the same as they do in Java, but getting the syntax down correctly was difficult to say the least. On top of this, the initial program I was using to develop the project, Dev C++, didn’t support some of the designs I wanted quite as well as I liked, so I had to switch development over to Visual Studio for the last half of development.

After confirming what I knew was correct, I set about starting development. Namely, I began on the ‘storage’ section of the program. To help shorten compilation time, reduce errors, and help with keeping the structure of the program simple, I separated the program into two main sections: a storage section and a processing section. I’ll leave the processing section for later as it is the more complex of the two. For the storage section, I relied on the use of polymorphism and inheritance to get the job done. Each different SQL command would have its own object assigned to it, all inheriting from a generic SQLCommand class. Then, after they had been fully processed, each command would be assigned to a Vector variable of the type SQLCommand for returning to the calling program where they would be taken and used to do whatever the calling program wanted to do with them.

Due to the storage section of the program being the first to be completed, it is most certainly the one that needs the most reworks done to it in a theoretical update to the program. I’m not entirely happy with it as is, but it does the job well enough to not warrant any immediate tampering. Were I to redo it, I would definitely organize the header files better, grouping similar functions together in a consistent manner with other header files.

The processing section of the program is the ‘meat’ of the project and is what was originally planned to be the extent of the project. Had I gone with just the processing part of the project, this would’ve been a straightforward problem. As is, this part of the project ended up being basically the same as my original vision except for a few quirks. Due to the nature of how strings are treated in C++, what with their somewhat lackluster support in comparison to other languages, I had to accommodate for quite a few things that can be handled easily in other languages, namely the parsing of strings.

In java, I can easily prepare and parse an SQL statement with just a few calls to a scanner object. There is no such luxury in C++, so creative solutions must be used. Due to strings in C++ essentially being arrays, I used this to my advantage in two ways. Firstly, I used this aspect of strings to iterate through them in order to clean them up, namely by ensuring every non-escaped character was in lower case. Secondly, I essentially treated the input string as a queue of sorts. Rather than keeping track of an index and carrying that throughout the program, creating plenty of room for error, I created a new string that subtracted off what I had just processed each time I processed a part of it. This way, I can ensure that what I’m working with is always starting at index 0, reducing chances of error.

Were I to redo the processing portion of the project, I would likely change how strings are handled. Rather than simply using strings at all times, I would likely create a custom storage object that stores each character in its own node on a doubly linked list. This custom object would have its own iterator stored within it that would act as cursor for more efficient parsing of the commands. By doing this, I would eliminate the processing problem of creating a new string each time I process something and I would make it much easier to delete and alter things as necessary.

In the end, I overcame many hurdles throughout the process of developing this project; hurdles that are better overcame now than later when I am creating something that would be used outside of an academic environment. Getting some design experience rather than just straight programming experience was also a major positive of this project. As stated in the intro: It is easy to create a program to do something, not so much to create a program that does many things. This is still true even at the end of this project, although now I have much more evidence for it than I did going in. All in all, I had fun creating this, and I hope the lessons learned are useful for years to come.