Homework #1: Growing a Language

20210356 안치현

At first, the lecturer defined the words that he would use in his lecture. He first defined the 'man' and 'woman', 'machine', class 'noun', 'other', class 'number', and various names of people, etc. Then he introduced various programming languages like Java, Fortran APL, Pascal, and PL/I.

With these examples of programming languages, he asked us 'Should a programming language be small or large?'. However, there are pros and cons to each language size. For small-size languages, it will take a short time to learn but we must keep adding new features in that language. On the other hand, it will take a very long time to learn the big language. But we can use all these features more easily without making or adding them ourselves. When we write real computer programs, we may think some words are primitive. But many of these words may not a primitive. The lecturer gives us a good example of it. The 'max' yields the larger of two numbers, and many might think this is primitive but only a few programming languages have it as primitive. In this point of view, a computer looks like a person who is but four years old.

The main thing that the lecturer wants to talk about is 'Which programming languages should we use to write programs? Small programming languages? Or Big programming languages?' The answer to this question from the lecturer is not both languages. The answer is 'language that can grow'. With some programming languages that I mentioned above, Fortran was a small language designed for tasks with numbers. And it can be grown and grown, many new words and new rules have been added so it served well. However, PL/I was a big language at first but now, we think PL/I is too small now because of the fixed size. Also, Pascal was designed as a small but whole language with no plan to add to it later. After this, he told us about the two kinds of growth in a language. One for changing the vocabulary, and the other for changing the rules that say what a string of words means. A library is a vocabulary designed to be added to a programming language to make the vocabulary of the programming language larger. The key point is that the new words defined by a library should look just like primitives of the language. And those that aren't are harder to grow with the help of users.

After this, he suggests two ways to do the growing. One way is for one person or a small group to be in charge and to take in, test, judge, and add the work done by other persons. The other way is to just put all the source code out there, once things start to work and let each person do as he wants. But the best way is to do both. Put the source code out there and let all people play with it. Have a person in charge who is a quick judge of good work and who will take it in and shove it back out fast. When I hear this thing from the lecturer, I can predict the example of this way. The 'Linux'. Linux is a very well-known operating system, and many people try to add new features and divide it into new versions. And he exactly mentioned the same thing with my thought. Exactly the 'Linux'.

The lecture introduced two ideas ‘cathedral’ and ‘bazaar’ to us. The ‘cathedral’ is a huge church. But the key thought is if there is one design plan, it may take a long time to make it real. Many people are needed to build it, but there is just one designer. Another idea is a ‘bazaar’ which is a place with many small shops or stalls, where we can buy things from many people who are there to sell their wares. There is no one plan, and each seller or buyer may change his mind at a whim Eric Raymond wrote a short work called ‘The Cathedral and the Bazaar’. Linux rose its fame and wide use in much the same way as Bazaar. The key point is that in the bazaar style of building a program or designing a language or what you will, the plan can change in real time to meet the needs of those who work on it and use it.

After this introduction of two styles of designing programming languages, the lecturer showed us some examples that can explain the importance of using the bazaar style. He introduced various examples of sum and multiplication between two pairs of numbers like complex numbers, rational numbers, ‘intervals’, and vectors. Some may think all these methods of sum and multiplication are very needed, but almost everyone thinks these are not often used. For these cases, we should not make the Java programming language a cathedral, but we need it more like a shopping mall, where there are not quite as many choices but most of the goods are well designed, and sellers stand up and back what they sell.

The very point of his lecture is language design must be a pattern (a pattern for growth). A pattern for growing the pattern of defining the patterns that programmers can use for their real work and their main goal. In other words, a good programmer does language design, though not from scratch, but by building on the frame of a base language.

Finally, he thinks that programming languages need to be more like the languages we speak, but it might be good too.

This is the end of the lecture, and I really agree with his opinion. While I take classes in ‘Programming Languages’, if we plan to make languages very big first time, I think it must be very difficult to make and there must be many bugs and problems for our programming languages. But following the course of ‘Programming Languages’, I can make my language bigger and bigger from the basic language. Also, if we make a new language, I think we should make the languages able to grow and grow for the convenience of using our language for users. With this lecture, I can finally know the purpose of the course ‘Programming Languages’. The process of making our languages gives us the reasons why we make programming languages like this. And even after this course, I will keep this in my mind while I keep programming.