CMPSCI 691ST Systems **Fall 2011**

Lecture 2

Lecturer: Emery Berger *Scribe: Ryan Hurley*

2.1 The FORTRAN Automatic Coding System

2.1.1 Summary

FORTRAN is a programming language developed for the IBM 704. FORTRAN both encompasses the first effort into developing a high level language while also creating the FORTRAN translator, a precursor to the compiler. Since FORTRAN encompasses these two components, the paper is spilt between describing the design choices of the FORTRAN language, and the operation of the translator. The paper finishes by discussing the benefits and possible impacts that FORTRAN may have on future programming efforts.

2.1.2 Discussion

A notable aspect of FORTRAN is how it uses a Monte-Carlo fashion for determining which code branches will be taken during the execution of a program. An author of a FORTRAN program is allowed to specify the frequency at which he/she believes a branch will be taken. Formal branch prediction did not exist at the time this paper was written. Many compilers after FORTRAN, including GCC, also allow a programmer to enter a frequency for a code branch. Being able to accurately predict branches and speculate correctly about which path will be taken, allows for huge improvements in caching and pipelining. After the advent of computer aided branch prediction however, human supplied frequencies are now generally disregarded except for some uses involving code localization.

To perform other optimizations, FORTRAN had to also find important sections within entire programs. To find these "hot spots", FORTRAN used the notion of "basic blocks" which are areas within the code which contain one entry point and one exit point. The blocks with the largest amount of contiguous code were the first blocks FORTRAN would attempt to optimize. FORTRAN also invested heavily in optimizing mathmatical calculations. Besides inventing all of its mathmatical notation, such as the asterix, the developers also spent a considerable amount of time optimizing expensive calculations like exponentiation.

Since the developers were so focused on improving a mathmeticians life, FORTRAN does not include a lot of information about variable scope. A complex and sometimes confusing inheritance of variables would most likely be frowed upon by mathematicians, so the idea of logical scope was never addressed in FORTRAN. There is also a no type checking, and the implementation of parameter passing seems to be largely ignored.

Another problem faced by the FORTRAN engineers was a lack of experience with parsing. Formal language theory had not been explored, and FORTRAN is actually written as a content-sensitive grammar. Most languages since have been constructed as context-free grammars because content-sensitive grammars cannot be parsed easily. FORTRAN also lacked reserved words which allows DO I = 1, 100 and DO I = 1.100 to both compile. The latter being a bug that sent a voyager space craft into the sun. The notable failures resulting from FORTRAN's lack of strong typing also led to a discussion of how certain modern languages were not learning from the mistakes of the past.

FORTRAN also shows no deep thought into collaboration. It does allow a programmer to create APIs and libraries, but actually dividing up a coding effort does not seem easy given the structure of FORTRAN. This has been a problem with many languages, with .NET being the first truly modular system for coding.

The design of the FORTRAN translator can be seen in many modern compilers. It is important to note that not all compilers have followed this method, however the FORTRAN translator clearly was the basis for future compiler work. As an aside, there appears to be no evidence that the engineers ever thought of using an interpreter instead of the translator they developed.

Discussion of this paper ended on the idea of ”ontogeny recapitulates phylogeny”. This disproven theory is based on the development of the human fetus. In summary, this theory postulates that a developing fetus goes through all the different stages of human evolution while maturing in the womb. Though this thory held no ground in actually biology, it has most certaintly happen multiple times throughout the history of computing with a prime example being the personal computer. When the ”PC” was created, large computing clusters with complicated compilers and operating systems already existed, yet the PC lacked many features of those systems. As time progressed, the personal computer incrementally gained the features that the larger computers already had. The PC went through the same evolution as the larger server based systems, and smartphones are poised to repeat the eveolution again.

2.2 The Education of a Computer

2.2.1 Summary

The Education of a Computer describes how a computer can be used to aid in complex tasks like mathematics. Instead of a mathmatician solving the problem directly, he would use his knowledge to build a computer subroutine to perform the given task. The mathmatician starts with easy to write subroutines, and eventually combines them to solve more complex problems with little strain on him. The paper also discusses not only how to combine these routines together, but how a computer would be expected to compile these soubroutines together.

2.2.2 Discussion

This paper was discussed breifly in class.

This is the earliest systems paper you can read, and it was the first to mention the use of subroutines and combining them together into libraries. Not only did the paper present this idea, but also stresses the importance of generality.

The Type B compiler describes the calculation of multiple derivates given one initial function. This sort of operation would involve symbolic execution and is surprising to find in a paper this old.

Mary Hopper popularized the term ”bug” when a moth flew into a computer and prevented a relay from firing.

”Most importantly” the paper already hints at the anamositity that will most likely develop between programmers and managers.