Lecture 1

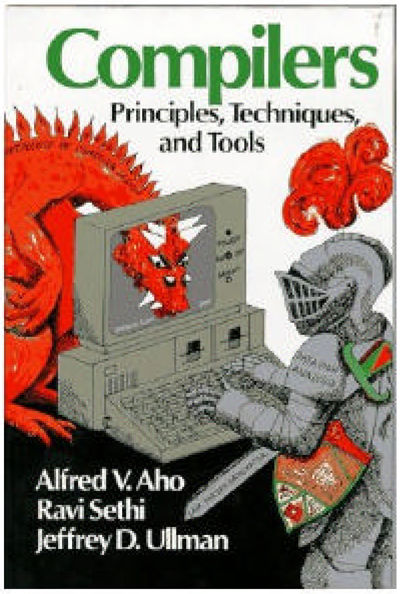
Course Organization

The course is organized around theory and significant amount of practice. The practice will be in the form of home works and a project. The project is the highlight of the course: you will build a full compiler for subset of Java-like language. The implementation will in C++ and you will generate Intel x86 assembly language code. The project will be done in six parts; each will be a programming assignment.

The grade distribution will be

Theory	Homeworks	10%
	Exams	50%
Practice	Project	40%

The primary text for the course is *Compilers – Principles, Techniques and Tools* by Aho, Sethi and Ullman. This is also called the Dragon Book; here is the image on the cover of the book:



Why Take this Course

There are number of reason for why you should take this course. Let's go through a few

Reason #1: understand compilers and languages
We all have used one or computer languages to write programs. We have used compilers to "compile" our code and eventually turn it into an executable. While we worry about data structures, algorithms and all the functionality that our application is supposed to provide, we perhaps overlook the programming language, the structure of the code and the language semantics. In this course, we will attempt to understand the code structure, understand language semantics and understand relation between source code and generated machine code. This will allow you to become a better programmer

Reason #2: nice balance of theory and practice
We have studied a lot of theory in various CS courses related to languages and grammar. We have covered mathematical models: regular expressions, automata, grammars and graph algorithms that use these models. We will now have an opportunity to put this theory into practice by building a real compiler.

Reason #3: programming experience
Creating a compiler entails writing a large computer program which manipulates complex data structures and implement sophisticated algorithm. In the process, we will learn more about C++ and Intel x86 assembly language. The experience, however, will be applicable if we desire to use another programming language, say Java, and generate code for architecture other than Intel.

What are Compilers

Compilers translate information from one representation to another. Thus, a tool that translates, say, Russian into English could be labeled as a compiler. In this course, however, information = program in a computer language. In this context, we will talk of compilers such as VC, VC++, GCC, JavaC FORTRAN, Pascal, VB. Application that convert, for example, a Word file to PDF or PDF to Postscript will be called "translators". In this course we will study typical compilation: from programs written in high-level languages to low-level object code and machine code.

Typical Compilation

Consider the source code of C function

```
int expr( int n )
{
     int d;
     d = 4*n*n*(n+1)*(n+1);
     return d;
}
```

Expressing an algorithm or a task in C is optimized for human readability and comprehension. This presentation matches human notions of grammar of a programming language. The function uses named constructs such as variables and procedures which aid human readability. Now consider the assembly code that the C compiler `gcc` generates for the Intel platform:

```
.globl _expr
_expr:
     pushl %ebp
     movl %esp,%ebp
     subl $24,%esp
     movl 8(%ebp),%eax
     movl %eax,%edx
     leal 0(,%edx,4),%eax
     movl %eax,%edx
     imull 8(%ebp),%edx
     movl 8(%ebp),%eax
     incl %eax
     imull %eax,%edx
     movl 8(%ebp),%eax
     incl %eax
     imull %eax,%edx
     movl %edx,-4(%ebp)
     movl -4(%ebp),%edx
     movl %edx,%eax
     jmp L2
     .align 4
L2:
     leave
     ret
```

The assembly code is optimized for hardware it is to run on. The code consists of machine instructions, uses registers and unnamed memory locations. This version is much harder to understand by humans

Issues in Compilation

The translation of code from some human readable form to machine code must be "correct", i.e., the generated machine code must execute precisely the same computation as the source code. In general, there is no unique translation from source language to a destination language. No algorithm exists for an "ideal translation".

Translation is a complex process. The source language and generated code are very different. To manage this complex process, the translation is carried out in multiple passes.