Notes for Dragon Book

Declarative vs. Imperative

Broadly speaking, computer programming languages have been divided into two categories — imperative languages and declarative languages. Examples of imperative languages are Pascal, C, Java, etc. Examples of declarative languages are ML, pure Lisp and pure Prolog.

The programming model in imperative languages is based on a statement-at-a-time paradigm where each statement has some effect on a memory store. Imperative programming is centered around the assignment statement, which allows one to change the content of cells in the memory store. Programs written in imperative languages are generally harder to write, debug, and maintain compared to those written in declarative languages. Imperative programming lays more stress on ”how” a solution procedure is specified. Programs written in imperative languages are generally larger in terms of code size and run faster compared to programs written in declarative languages. Imperative languages do not have a solid mathematical basis.

The programming model in declarative languages is based on stating the relationship between inputs and outputs. The actual computation procedure adopted is left to the runtime system. A declarative program can be viewed as a high level specification. Declarative programs are shorter, supposedly easier to write, debug, and maintain. Declarative programs are generally slower than imperative programs in execution speed. There are two major programming paradigms that are declarative: functional programming (with its formal basis in mathematical functions and the lambda calculus) and logic programming (with its formal basis in first order logic).

Static vs. Dynamic scoping

Dynamic scoping

```c
int x = 10;

// Called by g()
int f()
{
  return x;
}

// g() has its own variable
// named as x and calls f()
int g()
{
  int x = 20;
  return f();
}

int main()
{
  printf(g());
}
```