Program Abstractions, Language Paradigms

CS152

Chris Pollett

Aug. 27, 2008.

Outline

- Abstractions for telling a computer how to do things
- Computational Paradigms
- Language Definition, Translation

Recall

- Last day, we said: “ A programming language is a notational system for describing computation in a machine readable and in a human readable form.”
- We explored what we meant by computation and what we meant by machine readable.
- Today, we begin by looking at some of the more common abstraction used by humans to tell machines what to do

Abstractions for Programming Languages

- The ways we describe what we'd like a machine to do can be characterized in a couple different ways:
  - **Data abstractions versus control abstraction**: the former is how we describe the data such as strings, numbers, etc we want the computer to manipulate, the later is how we describe the sequence of operation the machine should take, like loops, if-statements etc.
  - **Levels of abstraction such as Basic, Structured, or Unit Abstraction**: these go from local machine information, to more global structure of the program, to entire components of the program.
- Let's look at some example of Data abstractions and control abstractions at each of these levels.

Data Abstractions

- Basic Abstraction Examples:
  - Rather than refer to how an integer is actually stored as bits in memory we might have a **data type** which acts as an integer and hides these details. For example, int x; in Java, C, etc.
  - Rather than have to refer to memory location to talk about a particular integer, we instead have **variables.**
- Structured Abstraction Examples:
  - A **data structure** is a tool for collecting related datatypes together. The most common example of such a structure is the notion of array that many languages support. Such an array might be declared with a line like: int a[10]; //in Java or C or INTEGER a(10) in Fortran.
  - You might also have **type declarations** like

    typedef int Intarray[10];
- Unit Abstractions Examples:
  - Here we might want to collect all the code related to a data structure in one place. For instance, we might want to **data encapsulate** the code for a List in a class, and we might want to hides it internals from users of this code (**information hiding**).

Control Abstractions

- **Basic Abstraction Examples:**
  - Typically a basic control abstraction combines a few machine code operations into one statement. For example, variable assignments such as: x =x +3;
  - Another example, might be the GOTO assignment which abstracts the jump operations of the machine code. Ex: goto label; label: //do something.
- **Structure Abstraction Examples:**
  - Break the code into groups of instructions together with tests that govern their execution. For example, if-statements, or case-statements (generally called selection statements).
  - While, for, do, repeat.
  - Function calls.
  - These structure abstraction can often be **nested** such as have nested if's or a while outside an if, etc.
- **Unit Abstraction:**
  - A stand-alone **unit** such a C library which collects together related procedures.
  - Threads, processes, tasks.

Computational Paradigms

- Sometimes the architecture of having a single processor executing instruction one at a time from memory using only a fixed number of other memory locations is called the **von Neumann Architecture**.
- Many common programming languages today such as C, abstract away the low-level messiness but otherwise map directly to this architecture. That is, one has a list of commands one executes in sequence. These languages are generally called **imperative languages**. The term **procedural** is also used.
- This is not the only framework (**paradigm**) for doing things. In some situation, such as in retrieving data from databases, parallel programming, etc other paradigms provide a more natural fit.
- We will now look at a few of these…

Object-Oriented Programming

- This paradigm is based on the notion of **object**.
- An object collects together a collections related properties, data structure, and the functions used to manipulate them.

Functional Programming

- A functional programming language is based on the notion of function and applications of functions to known values. (Sometimes also called an applicative language).
- Scheme is an example of such a language. The definition of the gcd function in scheme might look like:

  (define (gcd u v) (if (= v 0) u (gcd v modulo(u v))))

  We could then execute this with (gcd 6 3)

Logic Programming

- Logic programming is based on using symbolic logic to declare what things are known. (hence, also called **declarative programming**)
- And then using unification and resolution to figure out wha the person wants.
- For example, gcd might be defined as:

```
gcd(U, V, U) :- V = 0.
gcd(U,V, X) :- not (V =0), Y is U mod V, gcd (V,Y,X).
```

Language Definition

- There are several different ways one can define a programming language. One way is to write a compiler and say that the language is specified by how the compiler outputs your code. This is called creating a **reference implementation**. Some languages like PHP, Perl only do this.
- Another way to create a standard specification for the language is some English (or other human language) **reference manual**. To do this one needs to specify two things:
  - **Language Syntax**: this describes what strings constitute programs in the given languages. A spec for the syntax might be to give a context free grammar. Such a grammar might have rules like:

    <if-statement> ::== if(<expression>) <statement> [else <statement>]

    String like "if' and 'else' in the above are called **tokens**. Their description form the **lexical structure** of the language.
  - **Language Semantics**: a specification of this must describe how a given syntactic structure should be implemented on the computer. This can be English, or using one of a few common formal semantics such as: **operational semantics, denotational semantics, or axiomatic semantics.**

Language Translation

- To be useful a programming language needs to have a **translator.**
- This is a program which either translates instances of the language into machine code, which can then be executes **(compiler);** or which takes instances of the languages and executes them as it read them **(interpreter).**