CS 350: Programming Language Design

Lecture 8

Imperative Languages are based on the structure of our computer, namely the Von Neumann architecture.

* Processor: execute programs modifying the contents of memory
* Memory: store data and instructions
  + Variables are vital

Variables are abstractions of memory cells or groupings of memory cells

This provides significant readability over machine and assembly languages.

Variables are more than just a name to an address; it has several properties.

* Name
  + This is just the string of characters that we and the compiler use to identify that variable
  + We are concerned with the legality of a name (does it suit the syntax, are they case sensitive, are special words reserved or key?)
* Address
  + The memory address with which a variable is related
  + Sometimes called it’s *l-value*
* Value
  + The contents of the location with which the variable name is associated
  + The address of a variable -> the *l-value* of the variable
  + The value of a variable -> *r-value* of the variable
  + To access r-value, the l-value must be evaluated first
  + 0x00054BC = 4
* Type
  + Determines the range of values of a variable
  + Allows operations
  + Required storage space
* Lifetime
* Scope

Design Issues for Names

* Most languages have the same standards for variables
  + A letter followed by a string consisting of letters, digits, and underscores
  + A multiple-word naming convention
    - Camel notation, all of the first letters for each word, except for the first, will be uppercase.
  + The use of special characters
    - PHP’s use of $
    - Perl’s use of various symbols at the beginning of an identifier to indicate type
* Case sensitivity
  + Names in C-based languages are case sensitive
    - Names in others are not
  + Stick to a convention to avoid confusion
  + In C variables, names are lowercase
  + In C++, Java, and C# predefined names are mixed case
  + This offers poor readability
    - Names that look alike are actually quite different
  + Poor writability
    - The need to remember specific case usage makes it harder to write correct programs
* Special words
  + Used to separate syntactic parts of statements and programs
* Two types
  + Reserved words
    - Can’t be redefined
    - If there are too many, collisions will occur
    - We should only make them visible after they have been imported
  + Keywords
    - Have special meaning only in certain context, can be redefined
* Length
  + If too short, they aren’t descriptive
    - Fortran has a limit of 31 length identifiers
    - C99: no limit, but only first 63 are significant
    - C++: no limit, but implementers often impose a limit on name length to simplify symbol table
    - C# and Java have no limit, all are significant
* Not all variables will have names (anonymous variables)
* Design Issues for addresses
  + A variable may have different addresses at different times during execution
  + Variables may have different names
    - When two variables point to the same address, they are aliases for that address
      * Aliases can be created via pointers, reference variables
      * Aliases are harmful to readability
    - A variable may have different addresses at different places in a program
      * Different instantiations.