Variables and Bindings

Programming Languages

William Killian

Millersville University

Background

- Variables are necessary with ***imperative languages***
- Imperative Languages are abstractions of the von Neumann Architecture
  - Variables abstract **Memory**
  - Operations/Instructions abstract the **CPU**
- Variables/Bindings are defined by a set of attributes
  - These attributes can affect the behavior of the program
  - These attributes are common across **all** languages

Naming

- Naming things is **hard**

  "There are two hard problems in Computer Science: naming things, cache invalidation, and off-by-one errors"

- Things to consider:
  - Should capitalization matter?
  - Variable length (maximum?) (minimum?!)
  - What happens if I name a function/variable "**for**"?
  - Acceptable characters to include?
  - Naming conventions (e.g. FORTRAN)

Naming

- Case Sensitivity
    - coolFunction
    - CoolFunction
    - COOLFUNCTION

- Length
    - Should a minimum length be imposed?
    - Should a maximum length be imposed?
    - Languages:
        - C99: no limit, first 63 are significant
        - Java/C#: no limit, all significant
        - C++: no limit, implementation specific behavior (lol)

Naming

- Naming Conventions
  - PHP: All variables must start with a $
  - Perl (older): first character determines type
  - FORTRAN (older): first character determines type
  - Ruby: @ - Instance variables, @@ - Class variables
  - OCaml: Capital first letter (Module or Discriminator)

- Keywords / Reserved Words
  - *Keyword* – special only in certain context
    - **async** in C#, **override** in C++
  - *Reserved* – cannot be used as a user-defined name
    - **this** in Java/C++, **list** in OCaml

Variables

A **variable** is an abstraction of a memory cell

Composed of six attributes:

1. Name
2. Address
3. Value
4. Type
5. Lifetime
6. Scope

Variables

- Name
  - An identifier for the variable
  - Not all variables have names! (how?)
- Address
  - The location in memory associated with the variable
  - A variable may have different addresses at **different times**
  - A variable may have different addresses at **different places**
  - When two variables have the same address, they **alias** one another. Aliasing may be considered harmful

Variables

- Type
  - Determines the **range of possible values**
  - Determines the **set of possible operations**

    `*p` for C++ pointers / iterators

    `+ - * /` for integer/floating-point operations (C/C++/Java)

    `+. -. *. /.` for floating-point operations (OCaml)
- Value
  - **Contents** of the location in memory associated
  - Two types of values: l-values and r-values
    - **l-value**: the address of the variable (necessary for references)
    - **r-value**: the value of the variable (defined by type)

Bindings

The concept of a binding is to form an **association** between an **entity** and its **attribute**

Examples

- Variable and its type
- Variable and its value
- Operation and symbol

Binding Time: ***when*** the association is formed

Possible Binding Times

- Language Design Time

  Binding operator symbols to operation

- Language Implementation Time

  Bind floating-point type to representation (e.g. IEEE 754)

- Compile Time

  Bind a variable to a type

- Load Time

  Bind a C/C++ static variable to a memory cell

- Run Time

  Bind a non-static variable to a memory cell

Static and Dynamic Binding

Static

First occurs **before** runtime and remains unchanged

Dynamic

First occurs **during** execution **or** can change

Type Binding

- How is a type specified?
  - **Explicit:** specify the type – most compiled languages
  - **Inferred:** omit the type – Scripting Languages, OCaml
- When does the binding take place?
  - Compile time?
  - Run time?
- Can the type change during a program?
  - **Yes:** JavaScript, Python, Ruby, Perl
  - **No:** Java, C, C++, Swift, OCaml

Bindings

To Be Continued…