Module 1.2

Names, Bindings, and Scopes

1.1 Introduction

- Imperative languages are abstractions of von Neumann architecture
  – Memory: stores both instructions and data
  – Processor: provides operations for modifying the contents of memory •

  Variables are characterized by a collection of properties or attributes
  – The most important of which is **type**, a fundamental concept in programming languages
  – To design a type, must consider scope, lifetime, type checking, initialization, and type compatibility

1.2 Names

1.2.1 Design issues

- The following are the primary design issues for names:
  – Maximum length?
  – Are names case sensitive?
  – Are special words reserved words or keywords?

1.2.2 Name Forms

- A **name** is a string of characters used to identify some entity in a program.
- Length
  – If too short, they cannot be connotative –
  Language examples:
  - FORTRAN I: maximum 6 •

    COBOL: maximum 30
  - C99: no limit but only the first 63 are significant; also, external names are limited to a maximum of 31
  - C# and Java: **no limit**, and all characters are significant
  - C++: **no limit**, but implementers often impose a length limitation because they do not want the **symbol table** in which identifiers are stored during compilation to be too large and also to simplify the maintenance of that table.

- Names in most programming languages have the same form: a letter followed by a string consisting of letters, digits, and (_). Although the use of the _ was widely used in the 70s and 80s, that practice is far less popular.
- **C-based** languages (C, Objective-C, C++, Java, and C#), replaced the _ by the "camel" notation, as in myStack.

- Prior to Fortran 90, the following two names are equivalent:

```
Sum Of Salaries SumOfSalaries          // names could
                                       have embedded
                                       spaces // which
                                       were ignored
```

- Special characters
  – PHP: all variable names must begin with dollar signs $
  – Perl: all variable names begin with special characters $, @, or %, which specify the variable's type
    - if a name begins with $ is a scalar, if a name begins with @ it is an array, if it begins with %, it is a hash structure
  – Ruby: variable names that begin with @ are instance variables; those that begin with @@ are class variables
- Case sensitivity
  – Disadvantage: readability (names that look alike are different) –
  Names in the C-based languages are case sensitive
  – Worse in C++, Java, and C# because predefined names are mixed case (e.g. IndexOutOfBoundsException)
  – In C, however, exclusive use of lowercase for names.
    - C, C++, and Java names are case sensitive rose, Rose, ROSE are distinct names "What about Readability"

1.2.3 Special words

- An aid to readability; used to delimit or separate statement clauses
- A **keyword** is a word that is special only in certain contexts.
- Ex: Fortran

```
Real Apple                             // Real is a data type followed
                                       with a name, therefore Real is a
Real = 3.4                             keyword
                                       // Real is a variable name
```

- Disadvantage: poor readability. Compilers and users must recognize the difference. •

  A **reserved** word is a special word that **cannot** be used as a user-defined name.
- Potential problem with reserved words: If there are too many, many collisions occur (e.g., COBOL has **300** reserved words!)
- As a language design choice, reserved words are **better** than keywords.

- Ex: In Fortran, they are **only** keywords, which means they can be redefined. One could have the statements:

```
Integer Real          Real Integer                  // keyword
                                                     "Integer"
                                                           and
                                                     variable
                                                     "Real" //
                                                      keyword
                                                   "Real" and
                                                     variable
                                                     "Integer"
```

1.3 Variables

- A variable is an abstraction of a memory cell.
- Variables can be characterized as a sextuple of attributes:
  – Name
  – Address –
  Value
  – Type
  – Lifetime –
  Scope

- Name

  – Not all variables have names: Anonymous, heap-dynamic variables •

Address
  – The memory address with which it is associated
  – A variable may have **different** addresses at **different** times during execution. If a subprogram has a local var that is allocated from the run time stack when the subprogram is called, different calls may result in that var having different addresses.
  – The address of a variable is sometimes called its ***l*-value** because that is what is required when a variable appears in the **left** side of an assignment statement.

- Aliases
  – If two variable names can be used to access **the same** memory location, they are called aliases
  – Aliases are created via **pointers**, **reference variables**, C and C++ **unions.**
  – Aliases are harmful to readability (program readers must remember all of them) •

  Type
  – Determines the **range** of values of variables and the set of **operations** that are defined for values of that type; in the case of floating point, type also determines the precision.
  – For example, the int type in Java specifies a value range of -2147483648 to 2147483647, and arithmetic operations for addition, subtraction, multiplication, division, and modulus.

- Value
  – The value of a variable is the **contents** of the memory cell or cells associated with the variable.
  – Abstract memory cell - the physical cell or collection of cells associated with a variable.
  – A variable's value is sometimes called its ***r*-value** because that is what is required when a variable appears in the **right** side of an assignment statement.

    ▪ The ***l*-value** of a variable is its address. ▪

    The ***r*-value** of a variable is its value.

1.4 The Concept of Binding

- A **binding** is an association, such as between an attribute and an entity, or between an operation and a symbol.
- **Binding time** is the time at which a binding takes place.
- Possible binding times:
  – Language design time: bind operator symbols to operations.
    - For example, the asterisk symbol (*) is bound to the multiplication operation. –
  Language implementation time:
    - A data type such as **int** in C is bound to a **range** of possible values.
  – Compile time: bind a variable to a **particular data type** at compile time. –
  Load time: bind a variable to a **memory cell** (ex. C **static** variables)
  – Runtime: bind a **nonstatic** local variable to a memory cell.

1.4.1 Binding of Attributes to Variables

- A binding is **static** if it first occurs **before** run time and remains unchanged throughout program execution.
- A binding is **dynamic** if it first occurs **during** execution or can change during execution of the program.