Concepts of programing Languages

Lecture 3 :
Names, Binding, Type checking & Scopes.

Sudan University of Science and Technology

Dr. Aghabi Nabil Abosaif

20/10/2021

Lecture Contents

- **Names.**
- **Name Forms.**
- **Special Words.**
- **Variables(**name, address, type, and value**).**
- **The Concept of Binding**.
  - **Static Type Binding.**
  - **Dynamic Type Binding.**
- **Scope**

1.Names

- A name is a **string of characters** used to identify some **entity** in a program.
- Names **one of the fundamental attributes** of variables, subprograms, formal parameters, and other program constructs.
- The term **identifier** is often used *interchangeably* with **name**.
- **When Designing the name some** issues must be considers:
  - Are names case sensitive?
  - Are they refer to the **special words** of the language **reserved words** or **keywords**?

History Note

- The earliest programming languages used **single-character names. WHY?**
- This notation was natural because early programming was primarily mathematical.
- ***FortranI*** broke the tradition of the single-character name, **allowing up to six characters in its names**.

Name Forms

- It differentiate from one langue to another :
  - **C99** has **no length limitation** on its internal names, but only *the first 63 are significant*.

  External names in C99 (those defined outside functions, which must be handled **by the linker**) are restricted *to 31 characters*.

  - Names in **Java and** C# have **no length limit**, and *all characters in them are significant.*

  - **C++** does **not specify a length limit** on names, although implementers sometimes do.

Name Forms(Cons.)

- Names in **most** programming languages have **the same form**:
  - A *letter followed by a string consisting* of letters, digits, and underscore characters (_).
- Although the use of underscore characters to form names was widely used in the 1970s and 1980s, that practice *is* ***now far less popular***.
- In the **C-based** languages, all of the words of a multiple-word name *except the first are capitalized*, as in myStack ,Which is called ***camel notation***.
- All variable names in **PHP** must begin with a *dollar sign.*

Name Forms(Cons.)

- In **Perl**, the special character at the beginning of a variable’s name, $, @, or %, specifies its type (although in a different sense than in other languages).
- In **Ruby**, special characters at the beginning of a variable’s name, @ or @@, indicate that the variable is an instance or a class variable, respectively.
- ***In many languages***, uppercase and lowercase letters in names are distinct; that is, names in these languages are case sensitive.
  - For example rose, ROSE, and Rose.
- To some programmer, this is a serious **detriment** to **readability**, ***because*** *names that look very similar in fact denote different entities.*

Name Forms(Cons.)

- Despite this, **<u>not everyone agrees that case sensitivity is bad for names</u>**.

- In C, the problems of case sensitivity are avoided by the convention that variable names do not include uppercase letters.

- For example, the Java method for converting a string to an integer value is ***parseInt***, and spellings such as ***ParseInt*** and ***parseint*** are not recognized.

- This is a problem of ***writability rather than readability,*** because the need to remember specific case usage makes it more difficult to write correct programs.

- It is a kind of ***intolerance on the part of the language designer***, which is enforced by the compiler.

2.Special Words

- Special words in programming languages are used **to make programs more readable** by **naming actions to be performed**.
- They also are used to separate the syntactic parts of statements and programs.
- It are defined as **reserved words**, which mean they cannot be redefined by programmers.
- In some, such as **Fortran**, they are **only keywords**, which means they can be **redefined**.
- There is one **potential problem** with reserved words: If the language includes a **large number of reserved words**, the user may have difficulty making up names that are not reserved.
- Example of this is; COBOL, which has 300 reserved words. Unfortunately, some of the most commonly chosen names by programmers—for example, LENGTH, BOTTOM, DESTINATION, and COUNT.

Special Words (cons.)

- In most languages, names that are **defined in other program units**, such as Java packages and C and C++ libraries, can be made visible to a program.

- These names <u>are predefined</u>, <u>but visible only if explicitly imported.</u> Once imported, they <u>cannot be redefined</u>.

3.Variables

- variable is an **abstraction** of a computer **memory cell or collection of cells.**
- Programmers often think of variables **as names for memory locations,** but there is much more to a variable than just a name.
- The move from *machine languages* to *assembly languages* was largely one of **replacing** absolute **numeric memory** addresses for data with **names**, making programs far more **readable and easier to write and maintain.**
- A variable can be **characterized** as a one of following attributes: (name, address, value, type, lifetime, and scope).

3.1 Variable Name

- Variable names are the most common names in programs.
- Most variables have names. The ones that do not are discussed later.

3.2Variable Address

- The address of a variable is the **machine memory address** with which it is associated.
- In many languages, it is possible for the **same variable** to be associated with **different addresses** at <u>different times during the execution of the program.</u>
- For example, if a **subprogram** has a local variable that is allocated from the run-time stack when the subprogram is called; different calls may **result in that variable having different addresses.**
- The address of a variable is sometimes called its **l- value**, because the address is what is required when the name of a variable appears in **the left side** of an assignment.
- It is possible to have **multiple variables** that have the **same address**. When more than one variable name can be used to access the same memory location, the variables are called **aliases**.

3.3 Aliasing

- **Aliasing** is a **hindrance to readability** because it **allows a variable to have its value changed** by an assignment to a different variable.
- For example, if variables named **total** and **sum** are aliases, any change to the value of total also changes the value of sum and vice versa.
- A reader of the program must **always remember that total and sum are different names for the same memory cell**.
- Aliasing also makes program verification more difficult.
- A common way to create it such as in C and C++ is with their union types.
- **Two pointer variables are aliases** when they point to the same memory location.
- The same is true for **reference variables**.
- Aliasing can be created in many languages through **subprogram parameters**.

3.4 Variable Type

- The type of a variable determines :
  1. **The range of values** the variable can store.
  2. The **set of operations** that are defined for values of the type.

- For example, the **int** type in Java specifies
  - A **value range** of −2147483648 to 2147483647
  - **Arithmetic operations** for addition, subtraction, multiplication, division, and modulus.

3.5 Variable Value

- The value of a variable is **the contents of the memory cell** or cells associated with the variable.

- It is **convenient** *to think of* **computer memory** in terms of **abstract cells**, rather than **physical cells**.

- The **physical cells**, or individually addressable units, of most computer memories are **byte size**, with a byte having eight bits.

- For example
  - Although **floating- point** values may occupy four physical bytes. It is thought of as occupying a **single abstract memory cell**.

- The value of each simple **non-structured type** is considered to *occupy* **a single abstract cell**. Henceforth, the term memory cell will mean abstract memory cell.

Variable Value(Cons.)

- A variable's value is sometimes called its **r- value** because it is what is required when the name of the variable ***appears in the right side*** of an assignment statement.

- To **access** the r- value, the l- value must be determined first. Such determinations **are not always simple**.

- For example, **scoping** rules can greatly complicate matters.

4. The Concept of Binding

- A binding is an **association** between an **attribute** and an **entity**, such as between a **variable** and its **type or value**, or between an **operation** and a **symbol**.

- The time at which a binding takes place is called **binding time**.

- Bindings can **take place at** language **design** time, language **implementation** time, **compile** time, **load** time, **link** time, or **run** time.

4.1 Binding Time

1. The *asterisk symbol* (*) is usually bound to the *multiplication* operation at language **design time**.
2. A *data type*, such as **int** in C, is bound to *a range of possible values* at language **implementation time**.
3. **At compile time**, a *variable* in a Java program is bound to a particular *data type*.
4. A *variable* may be bound *to a storage cell* when the program is loaded into memory. That same binding does not happen until **run time**.
5. A call to a *library* subprogram is bound to the *program* code **at link time**.

C++ Binding Time Example

- In the following statement:
  - count = count + 5;
- The **type** of count is bound at **compile** time.
- The **set of possible values** of count is bound at compiler **design** time.
- The **meaning of the operator** symbol + is bound at **compile** time, when the types of its operands have been determined.
- The **internal representation** of the literal 5 is bound at **compiler design time.**
- The **value of count** is bound at **execution** time with this statement.

Binding Time

- The two important aspects of binding are :
  - how the type is specified?
  - when the binding takes place?

- For example, to understand **what a subprogram does**, one must understand how the actual parameters in a call are bound to the formal parameters in its definition.

- **To determine the current value of a variable**, it may be necessary to know when the variable was bound to storage and with which statement or statements.

4.2 Binding of Attributes to Variables

- A binding is **static**, if it first occurs before run time begins and remains **unchanged** throughout program execution.
- A binding is **dynamic**, If the binding first occurs during run time or **can change** in the course of program execution.
- The **physical binding** of a variable to a **storage cell in** a virtual memory environment is **complex**.

Binding of Attributes to Variables(Cons.)

- This because the segment of the address space in which the cell resides **may be moved** in and out of memory **many times during** program execution.

- In a sense, such variables **are bound and unbound** repeatedly.

- These type of bindings, are **maintained** by computer **hardware**, and the **changes are invisible** to the program and the user.

Static Type Binding

- **An explicit declaration** is a statement in a program that lists variable names and specifies that they are a particular **type**.
- **An implicit declaration** is a means of **associating** variables with types through **default conventions**, *rather* than **declaration statements.**
- **Both** explicit and implicit declarations **create static bindings to types.**
- Most widely used programming languages that **use static** type binding exclusively require explicit declarations of all variables (Visual Basic and ML are two exceptions).
- Implicit variable type binding is **done** by the language processor, either a compiler or an interpreter.

Implicit Declaration

- Some of the problems with **implicit declarations can be avoided** by requiring names **for specific types to begin with particular special characters.**
- For example, in **Perl** **any name** that begins with **$** is a scalar, which can store either a **string or a numeric value**.
- If a name begins with **@**, it is an **array**; if it begins **with a %**, it is **a hash structure.**
- Its creates different namespaces for different type variables.
- In this scenario, the names @apple and %apple are **unrelated, because each is from a different namespace**.

Implicit Declaration(cons.) type inference

- Consider the following declarationsin C#:
  - var sum = 0; var total = 0.0; var name = "Fred";
- The types of sum, total, and name are **int**, **float**, and **string**, respectively.
- *Keep in mind* that these are **statically typed variables**—their types are **fixed for the lifetime of the unit in which they are declared**.
- Visual Basic and the functional languages ML, Haskell, OCaml, and F# also use **type inference**.
- In these functional languages, **the context of the appearance of a name is the basis for determining its type**.

Dynamic Type Binding

- In dynamic type binding, the **type of a variable is not specified by a declaration statement**, ***nor*** can **it be determined by the spelling of its name.'**

- Instead, the **variable is bound** to a type when it is **assigned a value in an assignment statement**.

- When the assignment statement is **executed**, the variable being **assigned is bound** to the type of the value of the expression on the **right side of the assignment**.

- This assignment also **bind the variable to an address** and a **memory cell**, because different type values may require different amounts of storage.

Dynamic Type Binding(Cons.)

- A *variable's type* **can change** any number of times during *program execution*.
- It is important to realize that the type of a variable whose type is dynamically bound may be **temporary**.
- The primary **advantage** of dynamic binding of variables to types is that it provides more **programming flexibility**.

Dynamic Type Binding(Cons.)

- Before the **mid-1990s**,the most commonly used programming languages **used static type binding**, the primary *exceptions being some functional languages such as Lisp*.

- However, **since then there has been a significant shift to languages that use dynamic** type binding. In Python, Ruby, JavaScript, and PHP, type binding is dynamic.

Dynamic Type Binding(Cons.)

- In pure object-oriented languages—
  - For example, **Ruby—all variables are references and do not have types**; all data are objects and any variable can reference any object.
  - unlike the references in Java, **which are restricted to referencing one specific type of value**, variables in Ruby can reference any object.

Dynamic Type Binding Disadvantages

- There are two **disadvantages** to dynamic type binding.
- First, it **causes programs to be less reliable**,
  - because the **error-Detection capability** of the compiler is diminished relative to a compiler for a language with static type bindings.
  - Dynamic type binding allows any variable to be **assigned a value of any type**.
- Second, dynamic **type binding is cost**.
  - The cost of implementing dynamic attribute binding is considerable, **particularly in execution time.**
  - Type checking **must be done at run time.**

Question

- **Dynamic Type Binding**
  - Interpreter or compiler ? Why?

The End