

Naming (I)

Overview

- A critical ability of programming languages.
 - variables, functions, and programs need names before defining them.
- A name implies a **binding**, which is the connection between a **definable entity** and an **identifier**.
 - A binding is **static** if it takes place **before run time**.
 - Example: direct C function call. The function referenced by the identifier cannot change at run time
 - A binding is **dynamic** if it takes place **during run time**.
 - Example: non-static methods in Java (methods that can be overridden or inherited). The specific type of a polymorphic object is not known before run time (in general), the executed function is dynamically bound.
- Names (identifiers) are **determined by lexical rules**.
 - Case sensitivity

Case-insensitive	HTML, VHDL
Case-Sensitive	C, Java, Python
Mixed case sensitivity	PHP, MySQL

- In PHP, variables are case sensitive, but functions are not.
- MySQL provides an option for case-sensitive table names, and the query sentences are case-insensitive.
- Use of special characters: some languages allow use special characters

	Allowed	Not Recommend
Underscore (_)	C, Java, Python	SQL (considered as alias)

- Predefined identifiers
 - Reserved: **keywords/reserved** words
 - Most of PLs do not allow redefine the keywords. This is because it can make the parsing more efficiently.
 - For example, when parsing a program, the keyword, if, is always the start of a conditional statement. But if the keyword is redefined, it would need a second or more round to decide the meaning of the keyword.

Variables

- A variable represents the **binding** of an **identifier** to a **memory address**.
- The binding of a variable and address **requires four pieces of information**:

- **Identifier** string
 - **format** of identifier strings is **defined by** the **syntax**
- **Address** (implementation specific)
 - **uniquely identify** the actual memory location
- **Type**
 - Even a language **without explicit variable types**, the compiler or interpreter must, **internally, handle typing**. Otherwise, it does not know how to implement the basic operators on the data stored at the address.
 - Because the **data is just bits**, and it is the **type of manipulation** applied to the bits that **determines the semantic meaning** (if any) of the bits.
- **Value** (memory address or value)

l-value and r-value

- A **variable** provides both a **memory address** and a **value**, a language has to be clear about which to use in an expression.
 - If a variable's **identifier** is on the **left hand side** of an expression, it must provide a **memory address**.
 - If a variable's **identifier** is on the **right hand side** of an expression, it must provide a **value**.
 - **In some cases** the **value is**, in fact, a **memory address**, depending upon the language.
- The **convention** is that the **memory address meaning** of a variable is called **l-value**, and the **value meaning** of a variable is called **r-value**.
- Most languages do not require/allow explicit dereferencing of variables, although some language (C) provide a suite of tools for doing so.
- C Example

```
int x;
int y;
int *z;

// l-value and r-value of x
x = x + 1;

// l-value of y, r-value of x
y = x + 1;

// l-value of z, get l-value (address) of x and convert it to an r-value
z = &x;

// l-value of x, dereferenced z provides a r-value, change value of x
x = *z + 1;

// dereferenced z provides a l-value, r-value of y
*z = y;

// l-value of z, get l-value of y (address) and convert it to an r-value, change value of z
z = &y + 4;

// l-value of x, r-value of z and it's a memory address
x = (int)z
```

Scope vs Lifetime

- The **scope** of an identifier is the **part of program** in which the **identifier can be used to access its binding entity**.
 - C, Java, Python use **static scoping** (lexical scoping). A name is bound to a collection of statements in terms of its **position in the source program**.
 - Good readability and better compile-time checking
 - Perl uses **dynamic scoping**. A name is bound to its most recent declaration in terms of the **program's execution history**.
 - Generally not used, since it makes type checking difficult and is prone to errors