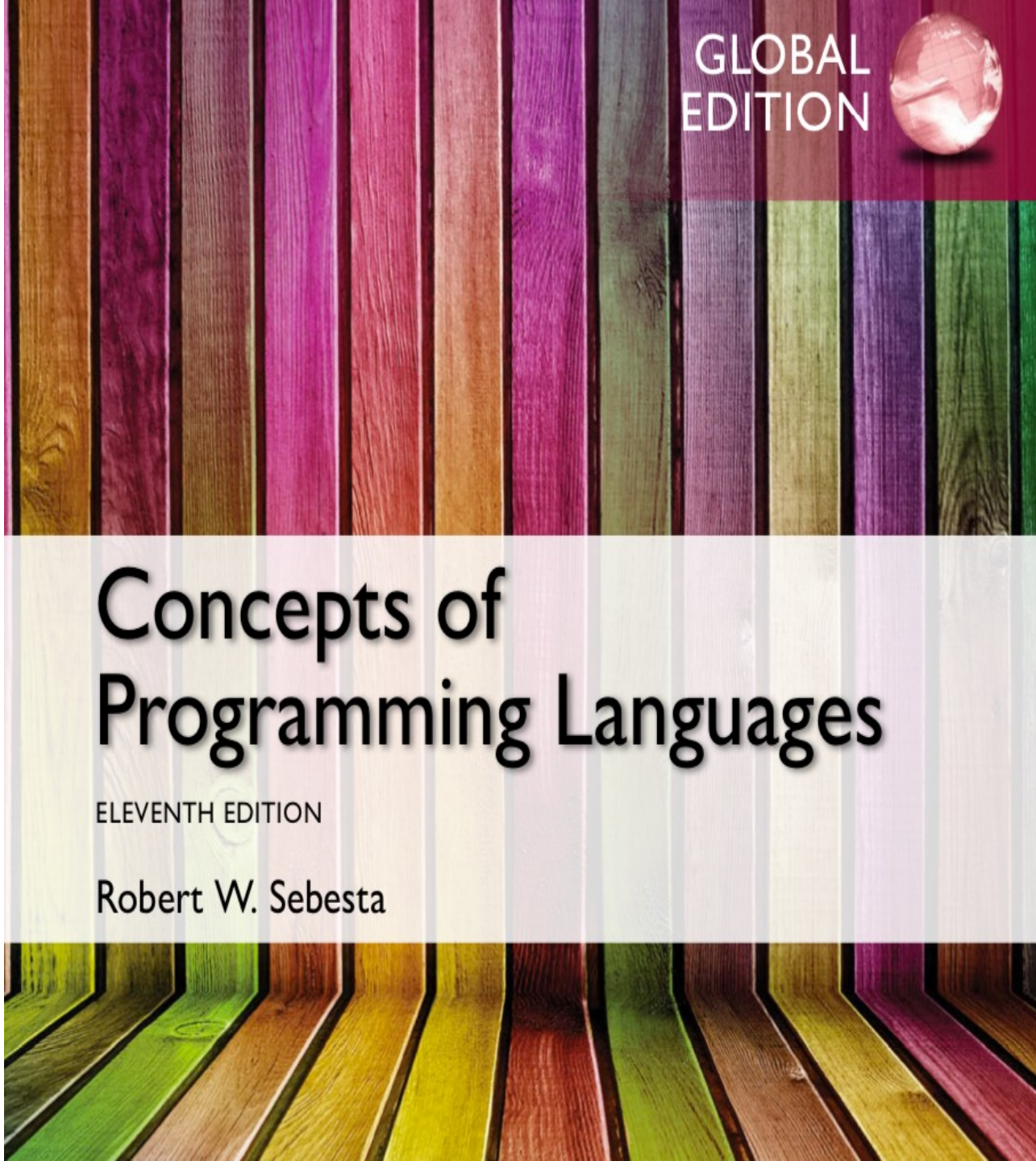
Chapter 5

Names,
Bindings,
Scope



Introduction

- Imperative languages are abstraction of von Neuman Architecture
  - Memory
  - Processor
- Variables are characterized by attributes
  - To design a type, must consider, scope, lifetime, type checking, initialization and type compatibility

Name

- Design issue for names:
  - Are name case sensitive?
  - Are special words reserved words or keywords?

Name

- Length:
  - If too short, they cannot be connotative
  - Language Examples:
    - FORTRAN 95: maximum if 31(only 6 in FORTRAN IV)
    - C#, Ada, Java: No limit and all are significant
    - C++: no limit, but implementers often impose one

Name

- Special Characters:
  - PHP: all variables name must begin with the dollar sign
  - Perl: All variable names begin with special character, which specify the variable's type
  - Ruby: variable names that begin with @ are instance variables , those that begin with @@ are class variable

Name

- Case Sensitivity:
  - Disadvantage: Readability (name that look alike are different)
  - Names in the C-based language are case sensitive
  - Names in the other are not
  - Worse in C++, Java and C# because predefined names are mixed case (e.g IndexOutOfBoundsException)

Name

- Special Words
  - An aid to readability, used to delimit or sperate statement clauses.
    - A *keyword* is a word that is special only in certain context e.g in FORTRAN
      - Real VarName (Real is a datatype followed with a name therefore REAL is a keyword)
      - Real = 3.4 (It is a variable here)
  - A reserved word is a special word that cannot be used as a user-defined name.
  - Potential problem with reserved words: If there are two many ,many collisions occur(e.g COBOL has 300 reserved words!)

Variables

- A variable is an abstraction of memory cell
- Variable can be characterized are 6 attributes:
  - Name
  - Address
  - Value
  - Type
  - Life Time
  - Scope

Variables

- Name: not all variables have them
- Address: The memory address with which it is associated.
  - A variable may have different addresses at different times during execution
  - A variable may have different addresses at different places in the program.

Aliases

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

Value

- The content of the location with which the variable is associated
  - The l-value of a variable is its address
  - The r-value of a variable is its value

The Concept of Binding

- A binding is an association between an entity and an attribute, such as between an operation and a symbol
- Binding time is the time at which a binding takes place.

The Concept of Binding

- A binding is an association between an entity and an attribute, such as 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
- Language implementation time – binding floating point type to a representation
- Compile time – bind a variable to a type in C or Java
- Load Time – bind a C or C++ static variable to a memory cell
- Run time – bind a monostatic local variable to a memory cell

Static Binding and Dynamic Binding

- A binding is static if it first occurs before runtime and remain unchanged throughout program execution
- A binding is dynamic if it first occurs during execution or can change during execution of the program

Type Binding

- How is a specified?
- When does the binding takes place?
- If static, the type may be specifies by either an explicit or an implicit declaration

Explicit/Implicit Declaration

- An Explicit declaration is a program statement used for declaring the types of the variables.
- An implicit declaration is a default mechanism for specifying types of the variables through the default conventions, rather than declaration statement.
- FORTRAN, BASIC, Perl, Ruby , JavaScript and PHP provides implicit declarations
  - Advantage : writability(a minor convenience)
  - Disadvantage : reliability

Explicit/Implicit Declaration

- An Explicit declaration is a program statement used for declaring the types of the variables.
- An implicit declaration is a default mechanism for specifying types of the variables through the default conventions, rather than declaration statement.
- FORTRAN, BASIC, Perl, Ruby , JavaScript and PHP provides implicit declarations
  - Advantage : writability(a minor convenience)
  - Disadvantage : reliability

Dynamic Type Binding

- Dynamic type Binding (JavaScript, Python Ruby, PHP and C#)
- Specified through an assignment statemet e.g
  - JavaScript
    - list = [2,3,4,5,6];
    - list = 17.3
  - Advantage: flexibility (generic program units)
  - Disadvantage:
    - High Cost (dynamic type checking and interpretation)
    - Type error detection by the compiler is difficult.

Storage Bindings and Lifetime

- Allocation – getting a cell from the same pool of available cells
- Deallocation – putting a cell back into the pool
- The life time of a variable is the time during which it is bound to a particular memory cell

Categories of Variable by Lifetimes

- Static – bound to memory cells before execution begins and remains bound to the same memory cell throughout the execution, e,g C and C++ static variables in the function
  - Advantage : efficiency (direct addressing), history-sensitive subprogram support
  - Disadvantage : lack of flexibility (no recursion)

Categories of Variable by Lifetimes

- Stack dynamic – Storage bindings are created for variables when their declarations statements are elaborated.
  - A declaration is elaborated when the executable code associated with its executed
  - Advantage: allow recursion, conserve storage
  - Disadvantage: Over head of allocation deallocation, Subprograms cannot be history sensitive , Inefficient reference

Scope

- The scope of a variable is the range of statements in which the variable is visible.
- A variable is visible in a statement if it can be referenced or assigned in that statement.
- A variable is local in a program unit or block if it is declared there. The nonlocal variables of a program unit or block are those that are visible within the program unit or block but are not declared there.
- Global variables are a special category of nonlocal variables

Static Scope

- Method of binding names to nonlocal variables called static scoping
- Static scoping is so named because the scope of a variable can be statically determined—that is, prior to execution.
- Two categories of static scoping
  - Nested static scope: those in which sub- programs can be nested
  - Non nested static scoping: those in which sub- programs can be nested
- Ada, JavaScript, Common Lisp, Scheme, Fortran 2003+, F#, and Python allow nested subprograms, but the C-based languages do not.

Static Scope

```
function big() {
  function sub1() {
    var x = 7;
    sub2();
  }
  function sub2() {
    var y = x;
  }
  var x = 3;
  sub1();
}
```

Blocks

- A section of code to have its own local variables whose scope is minimized
- Such variables are typically stack dynamic, so their storage is allocated when the section is entered and deallocated when the section is exited

```
if (list[i] < list[j]) {
  int temp;
  temp = list[i];
  list[i] = list[j];
  list[j] = temp;
}
```

Global Scope

- Definitions outside funtions in a file create ***Global variables***, which potentially can be visible to those functions.
- C and C++ have both declarations and definitions of global data.

```
$day = "Monday";
$month = "January";

function calendar() {
  $day = "Tuesday";
  global $month;
  print "local day is $day  ";
  $gday = $GLOBALS['day'];
  print "global day is $gday <br \>";
  print "global month is $month  ";
}

calendar();
```

```
day = "Monday"

def tester():
print "The global day is:", day

tester()
```