Name, Scope, and Binding

In Text: Chapter 3

Outline [1]

• Variable
• Binding
  – Storage bindings and lifetime
  – Type bindings
• Type Checking
• Scope
• Lifetime vs. Scope
• Referencing Environments
Variable

- **A program variable** is an abstraction of a memory cell or a collection of cells
- It has several attributes
  - Name: A mnemonic character string
  - Address
    - Points to location memory
    - May vary dynamically
  - Type
    - Range of values + legal operations
    - E.g., int type in Java specifies a value range of -2147483648 to 21473647, and arithmetic operations for +, -, *, /, %

Variable

- **Scope**
  - Range over which the variable is visible
  - Static/dynamic
- **Lifetime**
  - Time during which the variable is bound to a specific location
Binding

- A binding is an association between two things, such as a name and the thing it names
- Binding time is the time at which a binding takes place

Possible Binding Time

- Language design time
  - Bind operator symbols to operations
- Language implementation time
  - Bind floating point type to a representation
- Compile time
  - Bind a variable to a type in C or Java
- Load time
  - Bind a variable to a memory cell (C static variable)
- Runtime
  - Bind a nonstatic local variable to a memory cell (method variables)
An Example

count = count + 5

- count is a local variable
- When is the type of count bound?
- When is + bound to addition?
- When the value of count is bound?

Static and Dynamic Binding

- A binding is static if it occurs before run time and remains unchanged throughout program execution
- A binding is dynamic if it occurs during execution or can change during execution of the program
An Example of Dynamic Binding

• In JavaScript and PHP,
  list = [10.2, 3.5];

  ... ...
  list = 47;

Static and Dynamic Binding

• As binding time gets earlier:
  – execution efficiency goes up
  – safety goes up
  – flexibility goes down

• Compiled languages tend to have early binding times

• Interpreted languages tend to have later bindings
ONE CANNOT OVERSTATE THE IMPORTANCE OF BINDING TIMES IN PROGRAMMING LANGUAGES

Storage Bindings and Lifetime

- Allocation
  - Getting a memory cell from a pool of available memory to bind to a variable
- Deallocation
  - Putting a memory cell that has been unbound from a variable back into the pool
- Lifetime
  - The lifetime of a variable is the time during which it is bound to a particular memory cell
Lifetime

• If an object’s memory binding outlives its access binding, we get garbage
• If an object’s access binding outlives its memory binding, we get a dangling reference

Storage Allocation Mechanism

• Static allocation
• Stack-based allocation
• Heap allocation
• Variable lifetime begins at allocation, and ends at deallocation either by the program or garbage collector
Static Allocation

- Static memory allocation is the allocation of memory at compile time before the associated program is executed.
- When the program is loaded into memory, static variables are stored in the data segment of the program's address space.
- The lifetime of static variables exists throughout program execution.
  - E.g., static int a;

Stack-based Allocation

- Arguments to called routines
- Temporaries
- Local variables
- Miscellaneous bookkeeping
- Return address

Direction of stack growth (usually lower addresses)

$p$ (when subroutine C is running)

procedure C
  $D$, $E$
procedure $B$
  if ..., then $B$ else $C$
procedure $A$
  $B$
  $A$

-- main program

N. Meng, S. Arthur
A More General Representation

<table>
<thead>
<tr>
<th>Local variables</th>
<th>Parameters</th>
<th>Dynamic link</th>
<th>Return address</th>
</tr>
</thead>
</table>

Stack-based Allocation

- The location of local variables and parameters can be defined as negative offsets relative to the base of the frame (fp), or positive offsets relative to sp.
- The displacement addressing mechanism allows such addition to be specified implicitly as part of an ordinary load or store instruction.
- Variable lifetime exists through the declared method.
Heap-based Allocation

• Heap
  – A region of storage in which subblocks can be allocated and deallocated at arbitrary time

• Heap space management
  – Different strategies achieve different trade-offs between speed and space

Garbage Collection Algorithms

• Reference Counting
  – Keep a count of how many times you are referencing a resource (e.g., an object in memory), and reclaim the space when the count is zero
  – It cannot handle cyclic structures
  – It causes very high overhead to maintain counters
**Garbage Collection Algorithms**

- **Mark-Sweep**
  - Periodically marks all live objects transitively, and sweeps over all memory and disposes of garbage
  - Entire heap has to be iterated over
  - Many long-lived objects are iterated over and over again, which is time-consuming

- **Mark-Compact**
  - Mark live objects, and move all live objects into free space to make live space compact
  - It takes even longer time than mark-sweep due to object movement
Garbage Collection Algorithms

• **Copying**
  – It uses two memory spaces, and each time only uses one space to allocate memory, when the space is used up, copy all live objects to the other space
  – Each time only half space is used

Garbage Collection Algorithms

• **Generational Garbage Collection**
  – Studies show that
    • most objects live for very short time
    • the older an object is, the more likely it is to live quite long
  • Concentrate on collections of young objects, and move surviving objects to older generations, which are collected less frequently
Space Concern

• Fragmentation
  – The phenomenon in which storage space is used inefficiently
  – E.g., although in total 6K memory is available, there is not a 4K contiguous block available, which can cause allocation to fail

• Internal fragmentation
  – Allocates a block that is larger than required to hold a given object
  – E.g., Since memory can be provided in chunks divisible by 4, 8, or 16, when a program requests 23 bytes, it will actually gets 32 bytes

• External fragmentation
  – Free memory is separated into small blocks, and the ability to meet allocation requests degrades over time