

Structure of Programming Languages – Lecture 6

CS 636 – 536

May 11, 2009

- 1 Object Creation and Initialization
 - Initialization–Assignment–Binding
 - Lifetime
 - Dangling references

- 2 Assignment
 - Statement , Function, or Not supported?
 - Left and Right Semantics

Assignment vs. Initialization

Definitions:

- **initialization** is the act of storing a value in a variable when it is created.
- **assignment** happens when a value is stored in a variable at any other time. Assignment is also called **destructive assignment** because the former value of the variable is destroyed.
- **binding** is the act of attaching a name to an object, as in call-by-reference.

Birth and Death

Definitions:

- An object is **created** or **born** when space for it is allocated.
- **initialization** is the act of storing a value in a variable when it is created. They are created when functions are called and are deleted when the function returns.
- An object **dies** when it can no longer be used by the program.
- An object is **deallocated** or **deleted** when its space is returned to the system for future reuse.

Static vs. Dynamic

- **Static objects** are created and initialized at load time and live until program termination.
- Parameters in modern languages are **stack objects**. They are created when functions are called and initialized using the argument values.
- Local variables are also **stack objects**. They are created by a function when control reaches a local declaration, which may include an initializing expression. In C++, local variables of class types are initialized by the class constructor.
- Stack objects are deleted when the function returns.
- In C, this create/delete pattern is called **automatic** or **auto**.
- Non-auto dynamic objects are allocated in the **heap**.

Heap Storage

The **heap** is a storage area for dynamic objects, managed by the language run-time system..

- In OO languages, heap objects are initialized by a class constructor. In other languages, assignment, not initialization is used to initialize themn.
- A heap object lives from creation until the last reference to it is lost.
- Some languages provide garbage collectors to pick up and recycle all the dead heap objects. (Costly, unpredictable.)
- Other languages expect the programmer to explicitly delete heap objects when they die. (Risky.)
- A deleted heap object SHOULD go onto a system free-list for future re-use.

Static vs. Dynamic Languages

- FORTRAN: all objects are static – even function parameters.
- FORTH: Most objects are static and are allocated in the Dictionary. A named object in the dictionary may be used to point at an unnamed dynamic object, also allocated in the dictionary.
- Pascal: Stack and heap objects were supported, but not static variables.
- C, C++, and Java: static, stack, and heap are all supported.
- Java: class objects and arrays cannot live in the stack or static areas.
- Python: Question left for the student.
- Ruby: Question left for the student.

Dangling References

- A pointer can be NULL or it can point to something.
- If it points to something, that thing should be alive.
- If not, you have a **dangling pointer**.
- The most common cause of dangling pointers is:
 - Let function A call function B with a reference to a pointer p as a parameter.
 - B allocates a local stack-allocated variable and sets p to point at it.
 - B then finishes and exits; p is a dangling pointer, since its referent is dead.
- Pascal minimizes this problem: it prohibits pointing at a stack object.

Dangling References Revisited

There is another way to create a dangling reference.

- Let function F create a dynamic heap object and put it into a linked list.
- A scanner, S , is used to process this list sequentially, doing searches, insertions, and deletions.
- During a deletion operation, the cell that S points at is freed, but the pointer to the freed cell is not changed prior to the deletion.
- This common error creates a dangling pointer and erroneously turns the tail of the list into dead meat.

How does assignment fit into the language?

- In COBOL, FORTRAN, ALGOL, PL/1, Basic, FORTH, Pascal, and Ada, assignment is a statement.
- Python supports a parallel assignment statement.
- In APL and C, assignment is an operator-function (not a statement).
- In LISP, assignment is a function (setq, replaca, replacd: dirty back doors).
- In Scheme, assignment does not exist.
- In 1971 C, members of a struct had to be separately assigned.
- In COBOL, ANSI C, C++, and Java, a struct can be assigned coherently.

Python assignment

- An assignment creates an object and a reference to it.
- The reference is then bound to the name on the left.
- A name must be assigned before it can be used in a computation.
- The same operation is used to bind function arguments to parameter names.
- You can write a list assignment: `a, b = b, a`
- You can write a tuple assignment: `[a, b] = [3, 10]`
- You can chain assignments: `x = y = 10`

L-values and R-values

- An L-value is a reference to a variable: it can receive an = .
- An R-value is something that can be stored in a variable.
- Generally, L-values are written on the left side of an = operator and R-values are written on the right.
- However, subscripts and some parameters on the left side of an assignment still denote R-values.
- 20% of a C compiler's front-end is devoted to sorting out L- and R-values.
- Examples: p. 146–47.
- In FORTH, wysiwyg. No coerced dereference.

L-values and R-values

- When an R-value is written in a context that requires an L-value, it is always an error.
- When an L-value is written in a context that requires an R-value, most languages will coerce: the L-value will be dereferenced to get an R-value.
- `x = x + 1;`
- `x[y] = 10;`
- `double& mySlot(int k); mySlot(n) = sqrt(n);`
- Which one compiles? `int x,*p; x= ++p++; x= ++*p++;`
- why?