Heap, Variables, References, and Garbage

CS152

Chris Pollett

Oct. 13, 2008.

Outline

- Dynamic Allocation
- Variables and Constants
- Aliases and Problems
- Garbage

Introduction

- On Wednesday, we were talking about environments
- These were mappings of names to environments.
- We had just discussed how this worked for local variables in block structured languages:
- When we make a function call, we push locations onto the stack as needed for the local variables; when the call completes these locations are popped.
- Parameter variables that don't fit into the available registers are handled similarly.
- It is usually arranged that after the call returns the top of the stack will have the return value.
- Usually one has a couple of offset pointers into the stack, such as a pointer to the top and a base pointer for the start of the frame. How long a variable lives on the stack is its **lifetime**.

The Heap

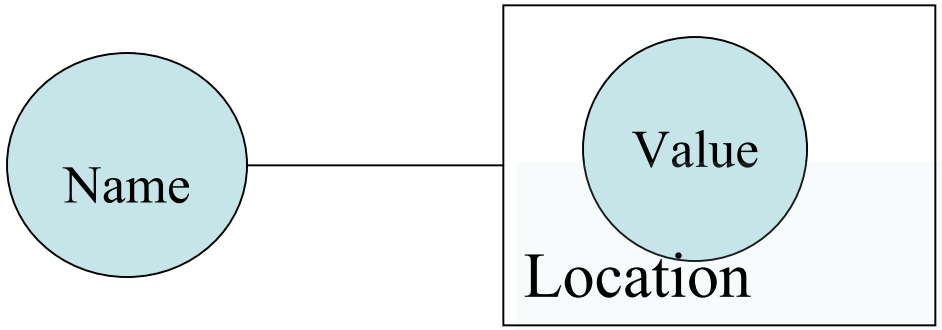
- It is often the case that one has a programming task for which we don't know how much memory will be needed until runtime.
- For example, we store user input and we don't know in advance how much there will be.
- It also might be convenient to allow the lifetime of this data to exactly match that of a given block.
- To facilitate this kind of allocation (called **dynamic allocation**), most modern programming languages have a runtime **heap**.
- In C, when we make a call to malloc, we are allocating memory from this heap; a call to free deletes from the heap.
- In C++, or Java when we use new we are allocating from the heap.

More on the Heap

- Typically, the memory for the heap is allocated starting from the bottom of the program environment growing towards the top; the stack on the other hand, grows downward to the bottom; and the static area is separate from either.
- C/C++ use pointers to refer to locations within the heap. A NULL or 0 address is used as the address for pointers which are not yet assigned to allocated memory.
- Other languages such as Java use the name **references**.
- The sections of heap memory currently allocated may become holey as memory becomes deallocated. So the heap structure needs mechanisms both to remember what is currently free/allocated; as well as occasionally to defragment this memory.
- Some languages such as Java have a garbage collector which is used to deallocate memory that is no longer referenced.
- Note some languages allow us to set which of the heap, stack, or global storage should be used for storing a variable. For example, the keyword static in C.

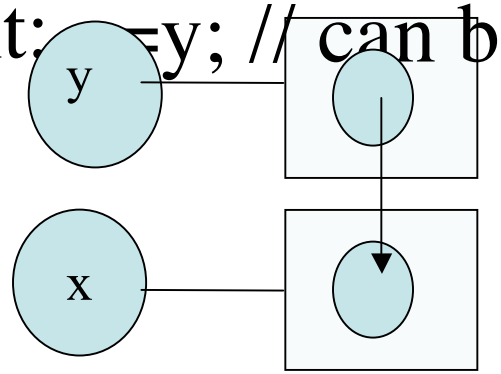
Variables

- A **variable** is an object whose stored value can change during execution.
- So a variable can be completely specified by: Name, Other Attributes (such as type and size), Location, and Value.
- Sometimes people use **box and circle diagrams** to represent the main aspects of this:



Assignment

- The principle way a variable changes value is through assignment: x= e; where e is an expression.
- The semantics of this are that e is evaluated to a value, which is then copied into the location of x.
- If e is another variable y, then the assignment: x=y; // can be viewed as



More on Assignment

- It is important to distinguish between the location of a variable and the value stored there.
- The value stored at a location is often called an **r-value** (because this is what a variable often means when it is on the right hand side of an assignment); the location of a variable is often called the **l-value**.
- Some languages like ML make this distinction explicit. For instance, x would refer to the location, !x to the value. So to increment x one would write: x := !x+1;
- C automatically dereferences l-values to r-values, and to do things explicitly uses & to get the l-value of an r-value and * to get r-value of an l-value.

Assignment Types

- In some languages an assignment x=y actually binds the location of x to y. This is called **assignment by sharing**.
- An alternative to this is to allocate a new location, copy the value of y to this new location, and bind x to the new location. This might be called **assignment by cloning**.
- These two kinds of semantics are often called **pointer semantics** to distinguish them from the usual storage semantics.
- For example, Java uses assignment for sharing for objects, but not for simple data.

Constants

- A **constant** is a language entity that has a fixed value for the duration of its existence in a program.
- A constant is like a variable except that it has no location attribute only a value attribute.
- One sometimes says it has **value semantics**.
- The notion of a constant is **symbolic**. A constant is essentially a name for a value.
- Entities like "hello there", 42 and the like are **literals** not constants.
- Constants come in two main flavors: **compile-time**, whose value can be determine at the time of translation; and **static**, whose value can only be determined after the program loads.(For instance, the location of a global variable).
- Only the latter actually need to be stored in memory.
- One can also have function constants, function variables, and function literals.

Aliases

- An **alias** occurs when the same object is bound to two different names at the same time.
- For example, two pointers to the same object.
- Aliases can cause **side-effects**, where a side-effect of a statement to be any change in the value of a variable that persists beyond the execution of the statement.
- For example, if x and y point to the same thing. Then *y=2; has two side-effects changing the value of y and changing the value of x -- the latter may not have been intended.
- Another example is x and y point to the same memory. We call free(y); //Now x points to memory which has been deallocated -- this is called a **dangling reference.**

Garbage Collection

- One way to reduce the problem of dangling references is to not allow explicit deallocation of memory.
- Instead, a runtime program called a **garbage collector** is used which determines which locations on the heap are no longer referenced, and deallocate them.
- Java, Smalltalk, and many functional languages use garbage collectors.