Programming Language Concepts Study Guide

Josepeh Sepich Nov 13

Contents

Scope

Scope is the visibility of variable names in a source program. This visibility can be determined at run-time or statically.

Symbol Table

A symbol table is a table of names and their bindings. A single scope can be seen as a dictionary or hash map, but a nested scope is depicted as a tree of symbol tables. Each scope has one symbol table and each symbol table may have a parent.

Static

Static scoping is also known as lexical scoping. In this scoping scheme the compiler determines the scope by examining the source code. This is used in most languages including C, Java, and Scheme.

- let

```
(let ((x 0) (y 1))
   (+ x y))
```

Scope of x y

- let*

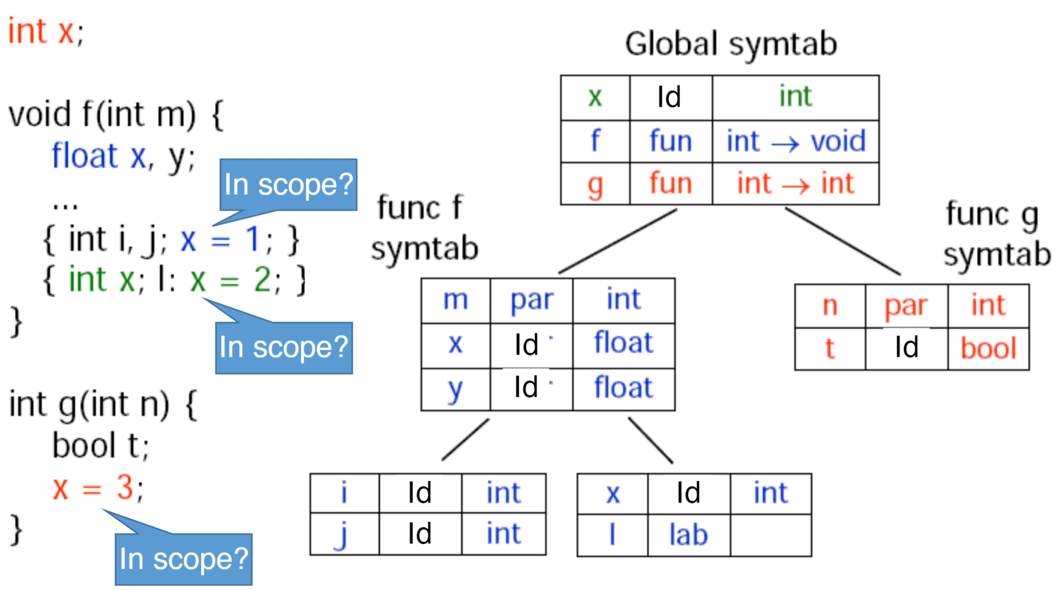
```
(let* ((x 0) (y x))
   (+ x y))
```

Scope of x

- letrec

```
(letrec ((isEven ((lambda n)
                  (if (zero? n) #t
                      (isOdd (- n 1))))))
         ((isOdd  ((lambda n)
                  (if (zero? n) #f
                      (isEven (- n 1))))))
```

Scope of isEven and isOdd

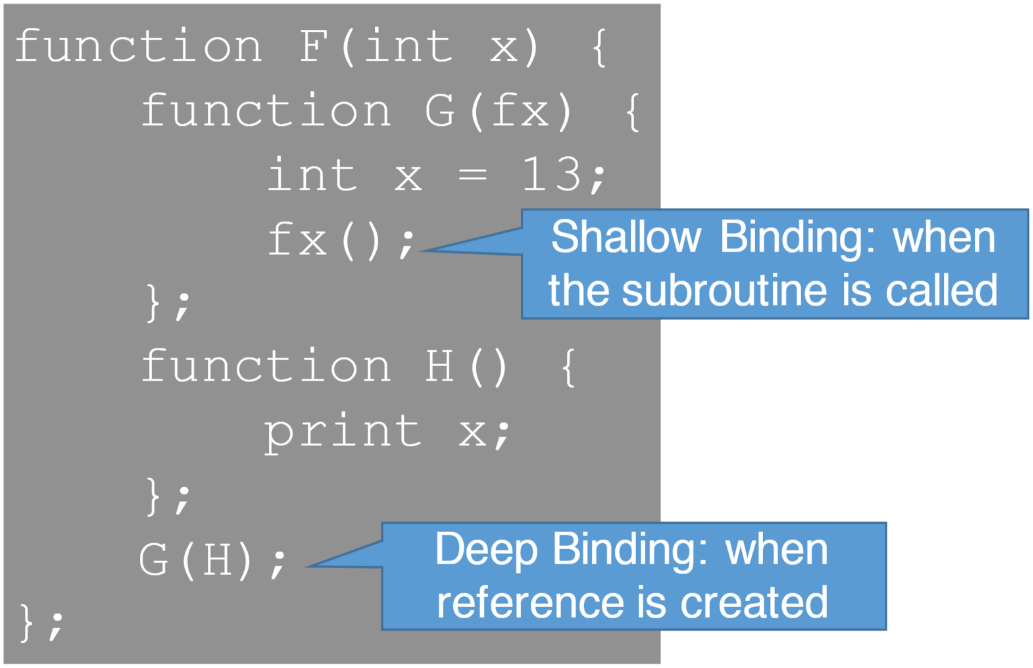


Dynamic

Dynamic scoping has to do with function calls. Dynamic scoping is determined at run time, and the callee symbol table has the caller as a parent. This tree would be a single trunk.

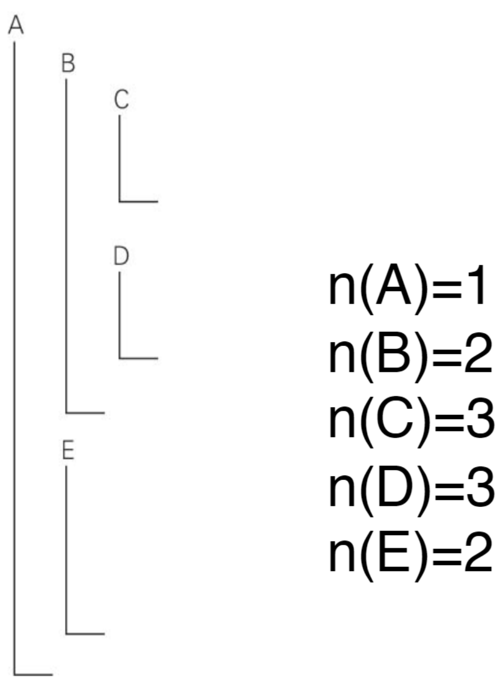
Binding: Shallow vs Deep

A shallow binding creates link when function is called, while deep binding creates link depending on where function is defined.



Links

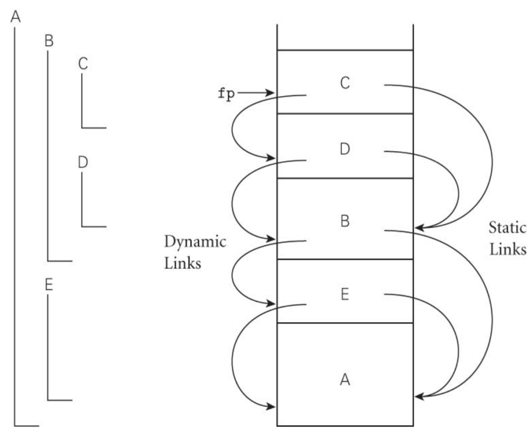
When determining links **nested depth** is very important. n(p), the nested depth of a function, are their distance from a function that is not nested (top-level), which is denoted as 1. A link determines the links of symbol tables when a link from A to B denotes that B is the lexical ancestor of A in static scoping. A dynamic link is easy to find, because it just points to the caller of the callee.



Determining Static Link

Static links are determined using the nested depth as previously mentioned.

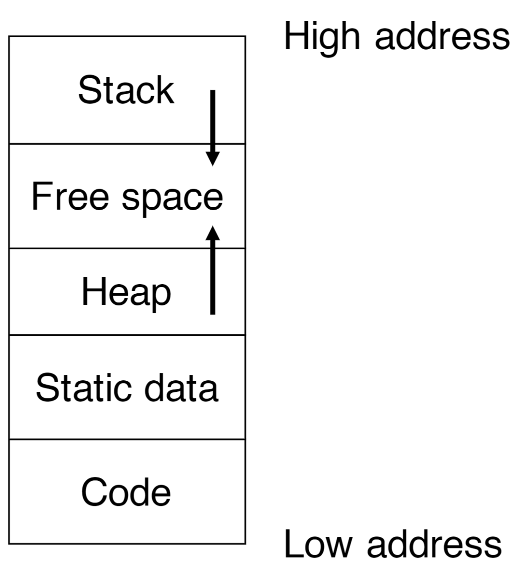
1. B calls D and n(B) < n(D): B is immediate ancestor of D. Static link is dynamic link.
2. B Calls D and n(B) >= n(D): follow B for n(B) - n(D) + 1 hops on static links.



Lifetime

The lifetime of a variable is directly related to the scope. As long as a variable exists in a symbol table it is considered to be alive. The lifetime usually corresponds to the lifetime of the function the variable is declared in.

Memory Storage



Variable Declaration

When variables are declared their default values vary depending on the language implementeation. They can either point to a null value, or like numbers in C, be instantiated to a value. Numbers in Java and C are 0 by default, and strings are empty by default (“”).

Static

Supports storage of any variables that exist for the duration of the program. This includes variables declared in the global scope, because they exist for the duration of the program.

Stack

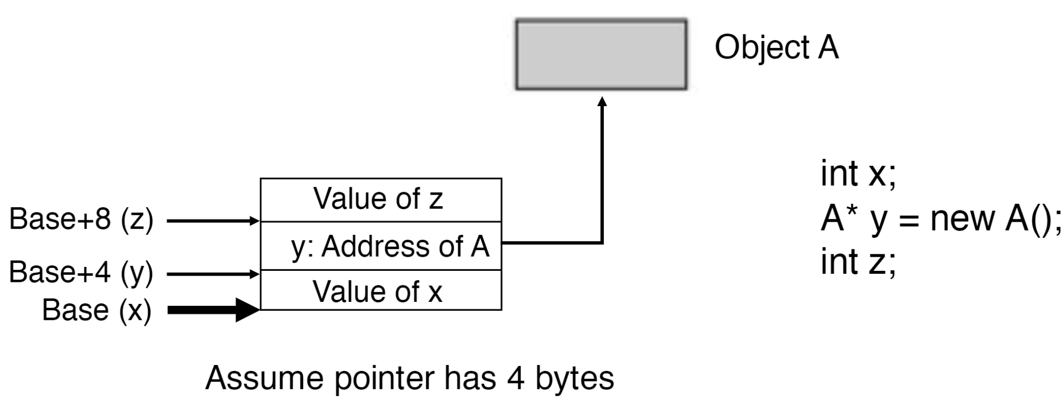
The stack is mostly used to store activation records or frames. A single activation record corresponds to the function call, which is why infinite recursion functions have a stack overflow. The stack also has two pointers to help with storage access. The frame pointer indicates the location of the current activation record, and the stack pointer points to the next available free space.

An activation record contains the following information:

- Local variables
- Temporaries
- Return values
- Bookkeeping info
- Arguments (to the callee)

Heap

The heap contains dynamic variables and objects that the programmer controls. This is where objects are contained when a programmer uses a constructor in Java to create a varaible. Pointers/references to the variable are stored in the stack which point to the actual object held within the heap portion.



Tail Recursive Functions

Definition

A tail recursive function is a function that calls itself as the very last thing: the result of the function is a recursive call. This property makes these types of functions equivalent to a loop.

Optimization

This enables the function to be more efficient both space wise and time wise. Space wise a non-tail recursive function must create a new activation record that has a separate return value and data in it, but with tail-recursion the only value chaging is the return values or variables in scope of the function. This is what a loop does, so you do not need more than a single frame in the stack, which also means no calling of other functions has to occur, which limits call time overhead.

Parameter Passing

Formal parameters are the identifiers used in a method or function to stand for the value passed in by a caller. **Actual** parameters are the actual value passed by the caller.

Call By Value

Call by value uses the values of the actual parameters passed to the function, but changes the actual parameters in no way.

Calling Mechanism

- Arguments evaluated to their values
- Memory or registers allocated for arguments on Activation Record (AR)
- Argument **values** copied to AR
- AR destroyed when callee returns

Characteristics

- Acutal parameters may not be directly changed by callee
- Arguments can be complex expressions
- Simple and intuitive (less error-prone)

Performance

- Primitive types: cost per parameter is small
- Constructed types: copying values is slow
- C: programmer must pass pointers to avoid copying cost
- Java: uses CBV for primitive and CBR for constructed

Call By Value Return (In-Out)

Calling Mechanism

- Arguments are evaluated to their values
- Memory or registers allocated for arguments on AR
- Argument values stored in AR
- Before callee returns, **AR values copied back to actual arguments**
- AR destroyed when callee returns

Characteristics

Mostly identical to Call By Value, but there is an extra step of copying values back to actual parameter.

Performance

Mostly identical to Call By Value, but copying cost happens twice, once on call and once on return.

Call By Reference

Calling Mechanism

- Arguments are evaluated to their values
- Memory or registers allocated for arguments on AR
- Argument **address** stored in AR
- AR destroyed when callee returns

Characteristics

- Formal parameter is merely an alias of actual parameter, since they point to the same memory
- Actual parameters may directly be changed in callee
- Some language disallows complex expression as arguments
- Programs are harder to understand (more error-prone)

Performance

CBR avoids the copying cost of CBV. In C CBR is the default mode, but in Java it is used only for constructued types such as arrays.

Call By Name (Lazy Evaluation)

Calling Mechanism

- Arguments are **not evaluated to their values** (like macros)
- Actual parameters replace all formal parameters in body

Characteristics

```
void swap (int a, int b) {
    int t = a;
    a = b;
    b = t;
}
swap(i, A[i]) // value swapped?
```

Types

Values are type-less in hardware, so operations on wrong values will produce garbage, such as float addition using integers. These errors are called **type errors**. A **type** is an abstraction of a set of values, and legal operations on those values.

- Denotational: a set of values
- Constructive: set of primitive types, type constructors
- Abstraction: an interface (set of operations)

Kind of Types

- Primitive
  - Integer
  - Double
  - Character
  - Boolean
- Constructed
  - Products
  - Unions
  - Arrays
  - Lists
- User-Defined

Type System

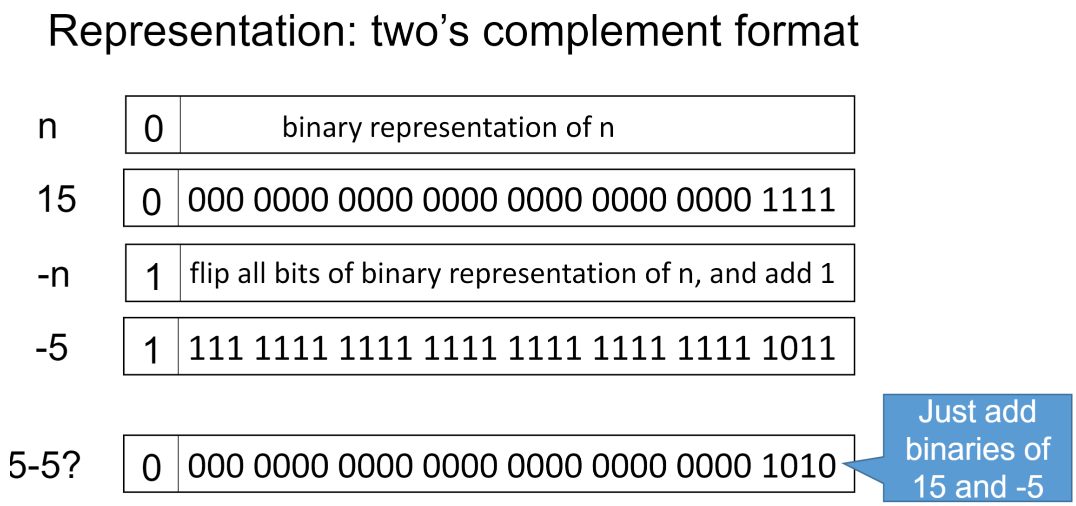
A type system is a method or specification for associating types with variables, expressions, etc. **Type equivalence** happens when two types are equivalent. **Type compatibiliy** determines if two types can be used together. **Type safety** includes the absence of type errors.

Type Checking

- Strongly typed: all type errors caught in checking
- Weakly typed: type checking may miss errors
- Static typing: type checking happens at compile time
- Dynamic typing: type checking happens at run time
- Type inference: type checker can infer variable types

Two's Complement

To find the value of a negative integer merely flip all the bits and add a 1. Odd values would have a 1 on the end and even values would have a zero. The first bit indicates sign.



IEEE 32-bit Floating Point

Due to loss of precision in floats, checking a value is sufficiently bounded is often good enough. An example of floating point rounding causes an error is the patriot missle defense system. Floating point is essentially scientific notation for binary. The mantissa would be the values after the 1.___ in the scientific notation.

Parts:

- 1 bit sign
- 8 bit exponent
- 23 bit mantissa

Boolean

In most languages booleans are either true or false. In C 0 means false, and all other values mean true. A **word** is the basic unit of load and store and is usually how large a boolean value is.

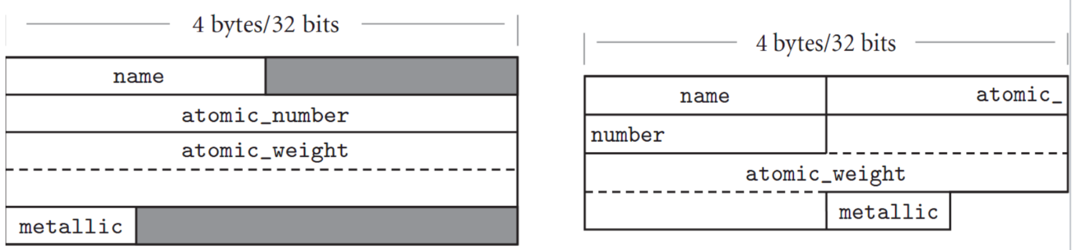
Character

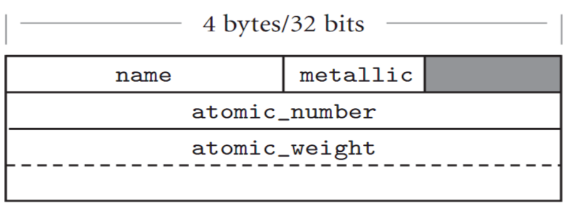
All languages support ASCII encoding, but some mordern languages, like Java, also support Unicode.

Records and Structure Type

- usually laid out contiguously
- possible holes for alignment
- compilers may re-arrange fields to minimize holes

```
struct element {
   char name[2];
   int atomic_number;
   double atomic_weight;
   _Bool metallic;}
```



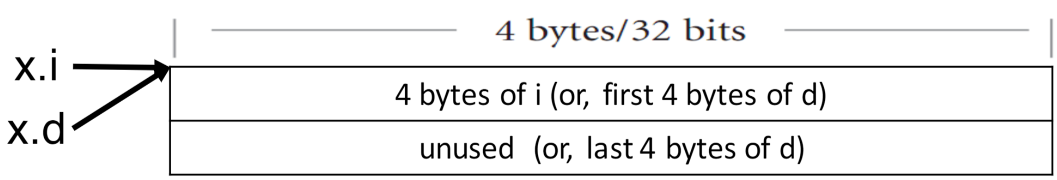


Possible
Memory Layouts

Union Type

In union types all fields share the same piece of memory.

A possible memory layout:

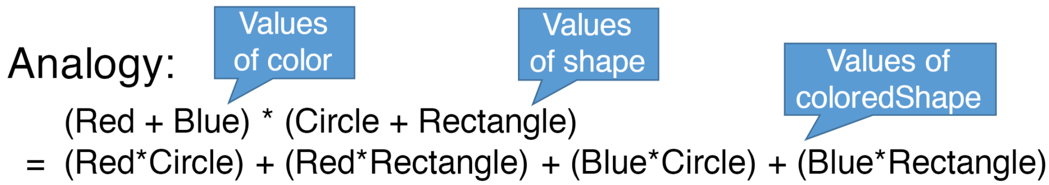


All fields share the same piece of memory

Sum and Product Types

Sum types are alternation of types and are a generalization of enum types. Haskell example below. Product types are concatenation of types such as records and structures.

```
enum Color {Red, Blue}
enum Shape {Circle, Rectangle}
struct ColoredShape {enum color c;enum shape s}
```



Arrays

C Array

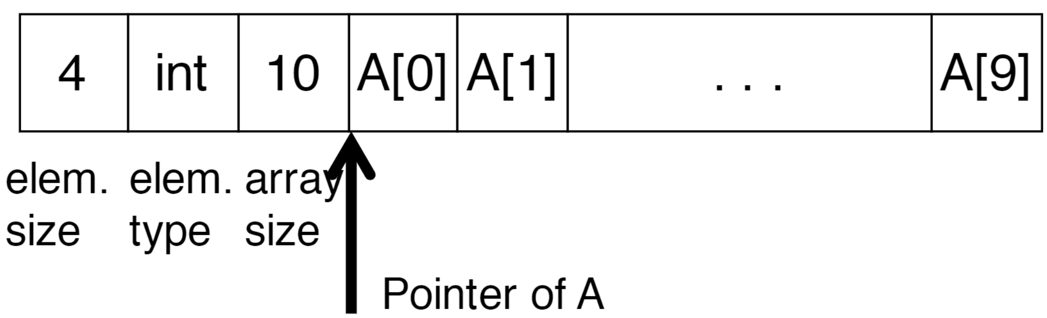
- Static/Stack/Heap allocated
- Size statically/dynamically determined
- Array bounds not checked (buffer overflow)

Java Array

- Heap allocated
- Size dynamically determined
- Array size is part of stored data (dope vector)
- Array bounds checked

Dope Vector

Dope vectors are stored in memories and offer metadata about the array. The beginning of the vector dictates the element size and type along with the array size. The rest of the vector indicated memory storage and acts as indicies.



Pointers and References

Pointers

Pointers are a set of memory addresses and operations on them. This includes legal addresses and the special null value. Some operations that can be performed are assignment, dereferencing, and arithmetic.

Uses:

- Indirect addressing (access arbitrary address)
- Manage dynamic storage (heap)

References

References are restricted pointers and cannot be used as a value or operated in any way. References are used mainly to manage dynamic storage and as aliases for existing variables.

Problems and Solutions

The null pointer problem happens when a pointer points to null, but is tried to be operated on like a type. A dangling reference is when a pointer points to a deallocated address. Tombstone and lock and key algorithms help to prevent this.

Type Systems

Rules needed will be given according to professor. Type systems use typing rules to created proofs by induction. If a proof is able to be made then the code is type safe, but if the proof fails then there exists a type error.

Typing Rules

Typing rules are rules that dictate which terms are well formed terms. The rules are set up as the given symbol table proves that the term has type τ.

Proof Tree

Proof tree starts with the original statement on the bottom. Use typing rules to move up the tree. Form conclusions once the root nodes of the tree are reached.

Constraints

Typing constraints assist with type inference. Constraints are equalities on types. Much like proofs of types constraint carry down through a proof tree.

Unification

Constraints for a given terms can be resolved through unification. Unification is the process that finds substitudes to make two terms equal. There is a set of statements to solve constraints that make an algorithm such as Robinson's algorithm.