review for Ex2

Names, bindings, Scope — ch 5

- static type binding (at compile time)
  - explicit declaration — using decl. statement eg) int x
  - implicit declaration — eg) perl ($--) scalar type, (@--) array type
    - eg) C# — var sum = 0; var total = 0.0; var name = "Fred"; (name: string)

vs.

- aliases — name1, name2, ⋮ → same mem. location
- dynamic type binding — for interpreter based lang. only (Python, Ruby, javascript)..
  - (at run time via assign-st.
  - eg) list = [10.2, 3.5]; list = 47
  - adv. — flexible
  - disadv. — less reliable
    - error detection at compile time is difficult.

Storage binding

- static var. — no recursion
- Stack dynamic var — alloc. in run time stack (A.R.) — recursion possible.
- explicit heap dynamic vars — nameless mem. cells — new/delete
- implicit heap dynamic vars
  - bound to heap only when assigned values
  - eg) javascript: highs = [74, 84, 86, --]

— Scope (of vars)

— static scope rule

- — nested scope — subprograms can be nested — e.g. python
- — non-nested scope — C/C++/Java

for non-locals in a subprog,
search
- static parent — static scope
- caller's environment — dynamic scope.

blocks { ≡ } — nested static scope

— declaration order

e.g.
- C — all decl. at the begin of a func.
- C++/Java, .. — decl. at anywhere

C

'{' <decl_list> <state_list> '}' — terminal

C++

'{' { <decl_list> | <state_list> } '}' — terminal

terminal; 0 or more

— scope of local var

- C/C++/Java — from decl. to the end of block
- C# — whole block

e.g.

```
{ int x;
  { int x; } ✗ — no!
  }
}
```

whole block

e.g.

```
{
  { int x; } ✗ no!
  }
  int x;
}
```

whole block

— global scope

extern int x; , C++ ::x=5 , PHP way of accessing globals.,

- referencing environment of a statement
  a collection of all vars that are visible in the statement.
  (dynamic/static scope lang's) different
  examples

= Data Types — Ch6

- primitive types
- char. string types — seq. of char's
- type conversion
  - coercion — auto type conversion
  - typecasting — explicit type conversion
- short-circuit eval. of Boolean expression.
  ||, &&. — if (○ || ○ ...)
  if (○ && ○ ...)

— Array types

- static array — storage alloc. is static (at compile time)
- fixed stack-dynamic array — storage alloc. during exec. (run time stack)
- fixed heap-dynamic array — eg) new/delete
- heap-dynamic array — storage alloc. — dynamic and can change any # of times during execution.
  eg) C#.
  range and storage (size)

- slices — substructures of an array as a unit
  eg) python
  vector = [2, 4, 6, 8, 10, 12, 14] [0] [1]
  vector [3:6] = [8, 10, 12] — [3, 6)
  1st index not included

— Array implementation

1-D array layout

lower bound = 0 case:

$$\boxed{\text{addr. of } A[i] = i * w + \text{base}}$$

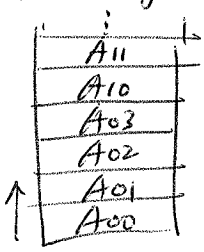
w: size of ele (in bytes)

2-D array layout

ex) int A[3][4]

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}

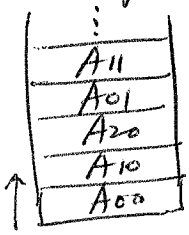
— row-major layout



addr. of $A[i_1][i_2]$

$$= [(i_1 * \underbrace{H_2}_{\#\text{cols}}) + i_2] * \underbrace{w}_{\text{ele. size}} + \text{base}$$

— col-major layout



addr. of $A[i_1][i_2]$

$$= [i_1 + (i_2 * \underbrace{H_1}_{\#\text{rows}})] * \underbrace{w}_{\text{ele. size}} + \text{base}$$

— array bound evaluation

(at compile time (static) — ex). int A[10];

(at run time (dynamic) — ex) --- new int[10];

Java int[] A = new int[10];

— Associative array

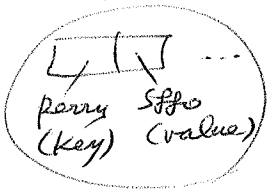
Perl — hash var starts with %

ex) % salaries = ("Gary" ⇒ 7500, "Perry" ⇒ 5700, ...);

⋮

$salaries{"Perry"} = 5880;

scalar var



python — dictionary — {...}

C++ — unordered_map — <P, P>

key value (can be structure)

— Heap memory management

- explicit — new/delete — C++
- implicit deallocation — (Java garbage collector, LISP)

— pointers/reference types

dangling pointers/memory leak

array (index) vs. pointers in C — class example

→ Lazy approaches

— semantic of q = new node;

- mark/sweep approach
  - mark phase: (trace and mark all reachable cells:
  - sweep phase: start from lowest addr., collect all unmarked cells to the head of free-list (LIFO).
- copy collector

eager approach

— reference counting method — as soon as a cell's ref-counter = ∅, collect it.

adv/disadv.

Imperative Lang features — ch 7, 8,

- expressions
- overloaded operations
- type conversions
  explicit/implicit
- short-circuit eval. of Boolean expr.

✓ already covered

- multiple assignment
  - ex) perl: (a, a, a) = (a, a, a);
  - swap operation: in 1 statement (A, B) = ((B), (A));

- syntax-directed control structures
  - loop constructs — 1 entry/1 exit
  - handling special case in the loop — using "break".
    (breaks 1 entry/1 exit rule of a loop construct.

Subprograms — ch 9.

- parameter passing methods
  - call by value
  - call by reference
  - call by value-result
  - ~~Ca~~ macro-expansion
  - call by name — keep uniq names

  use swap operation example

- multi-dimensional array as parameter
  C++, Java examples
- parameters that are subprograms
  JavaScript example.

parameter passing

1. Call by value

(1) formals ← r-value of arguments
(2) body execution

2. Call by reference

(1) formals:
l-value of formals ← l-value of arguments
(2) body execution

3. Call by value result

(1) copy-in phase:
formals ← r-value of arguments
save l-value of arguments for formals
(2) body execution
(3) copy-out phase:
final values of formals → saved l-values of arguments

4. Macro-expansion

(1) text of arguments → formals/body
(2) text of callee's body → call statement
(Substitutes)
(3) body execution

5. Call by name

(1) text of arguments → formals/body
if name conflicts, rename locals in the callee.
(2) text of callee's body → call statement
if name conflict (locals in caller ↔ non-locals in callee),
rename locals in caller.
(3) body execution.

§9.6 — parameters that are subprograms — in nested subprograms lang. ⇒ Javascript

```
function Sub1()
{ var x;
  function Sub2()
  { alert(x);
  };
  function Sub3()
  { var x;
    x=3;
    Sub4(Sub2);
  };
  function Sub4(SubX)
  { var x;
    x=4;
    SubX();
  };
  x=1;
  Sub3();
};
```

Sub2

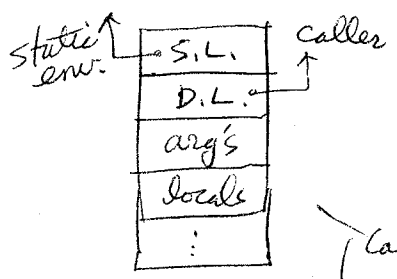
actually, Sub2 is called

— output under
1. — 4
2. — 1
3. — 3

— 3 choices of binding non locals
1. Shallow binding — caller's env.
2. deep binding — static env.
3. Ad hoc binding — bound to env. in which the call statement passed the subprog. as a parameter.

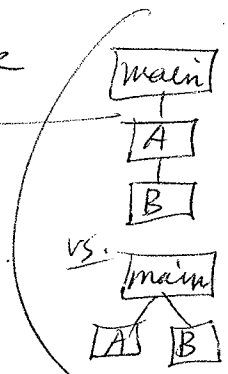
Ch10 — Implementing Subprograms

— A.R. and runtime stack



— snapshots of ~~A.R.~~ runtime stack

— activation tree



Caller sets S.L., D.L., parameters, return addr.
Callee sets remaing fields (locals)

— run-time stack (ARs)

with parameter (pass by value / vs pass by reference) — class example.

(→ binary search program (homework)

Activation tree / run time stack for recursive func.

— (static link / dynamic link) for ((nested) static scope / dynamic scope)

— nested static scope (PASCAL, .. python)

How to set S.L. of callee

1. Compute static distance

d = nesting level of caller − nesting level of callee + 1

2. Start from caller's AR (S.L.), trace S.L. (d) times.