3 - storage, variables, and commands

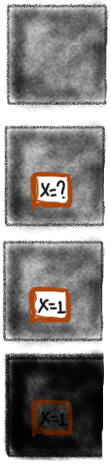
variables

functional language variables
- defined or solved
- remains same

imperative language variables
- has a state and value
- assinged to a different values

operations on variables → inspect (look at)
→ update

memory cells



- initially unallocated
- allocated/ undefined
- storable
- unallocated

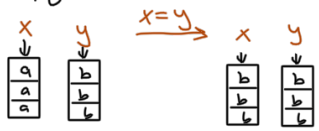
```
void f() {
    int x;
    x = ⊥;
    return;
}
```

implementation of arrays

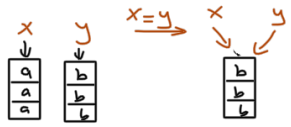
1. static = array size is fixed at compile time
   - `int a[10];`
2. dynamic = array size is defined when variable is allocated, remains constant afterwards.
   - `int f(int n) { int a[n]; .. }`
3. flexible = can exted or shrink at run time, (in python, php..)
   - `a=(1,3,5);` - `a[10] = 27;`

Semantic of assignment in composite variables:

1. copy



2. reference



→ copy semantics is slower
→ reference semantics cause problems from storage sharing

variable lifetime
1. global → while program is running

2. local → while declaring block is active: function
   in C, main variables are also local.

3. [illegible]

③ Heap → arbitrary = allocation and deallocation is not automatic, explicitly requested by function calls: C: malloc() / free() C++: new, delete *p is heap variable

dangling reference: trying to access a variable whose life time is ended, and already deallocated

garbage variables: lifetime still continue but thee is no way to access.

④ persistent → continues after program terminates: files, database, web service objects stored in secondary storage

☆ procedure: user defined commands: in C, function returning void

memory management

stack section: run-time stak

heap section: heap variables (grows / shrinks with allocation/deallocation)

data section: global variables (fixed in run-time)

code section: executable instructions, read only