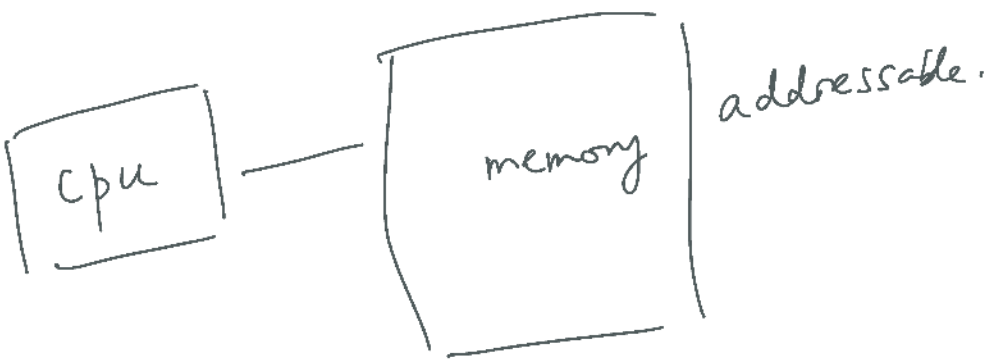
PLC Lecture 2.

Support for functions.

Modularity

Abstraction.

Von Neumann architecture.



modularity.

cpu - circuitry that carries out basic computation.

expects data to be available at fixed location (registers)

Transfer data between memory & cpu.

$x = y + z$ ⇝

```
LOAD y   | transfer from memory
LOAD z   |
ADD
STORE x. | transfer back to memory
```

Most basic abstraction provided by PLs.

Variables. — abstraction of data residing in a memory location.

Need a mapping from variables to memory locations.

Primary job of a compiler

Variables – type (what values can be stored in the variable)

Location – where it is stored

Value – what is stored

Scope, Lifetime.

Mapping of variables to addresses.

– Simplest scenario. assembly languages, Fortran etc.

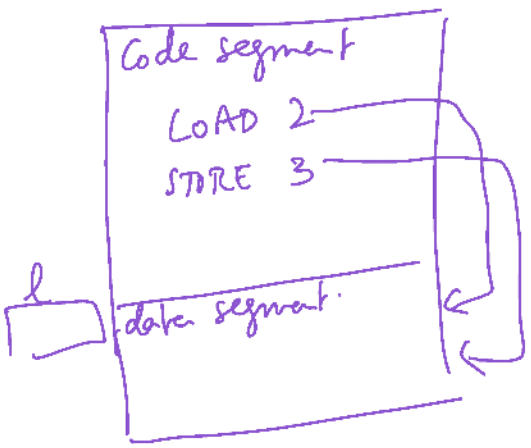
static allocation.

address mapped to a variable does not change in the course of

running the program.

Compiler translates variables to addresses once and for all.

relative addresses: — Specify addresses as offsets from a fixed address.



loaders.

Static allocation.

(works well if there are no functions at all. every variable is global.

Functions – Basic abstraction feature.

Might use the same name (say x) for variables in two different functions.

local.

Static (local) variable.

global variables.
|
static allocation

```
f(..) {
    static int counter = 0;   — count the number of
    counter++;                   times f is invoked.
    ...
}
```

lifetime. — how long is the variable "active"?

for global variables — lifetime is all of the execution time of the program

for local variables — lifetime is the execution time of the function.

↗ typical case, for non-static localvars.

for static local variables — lifetime is the execution time of the program.

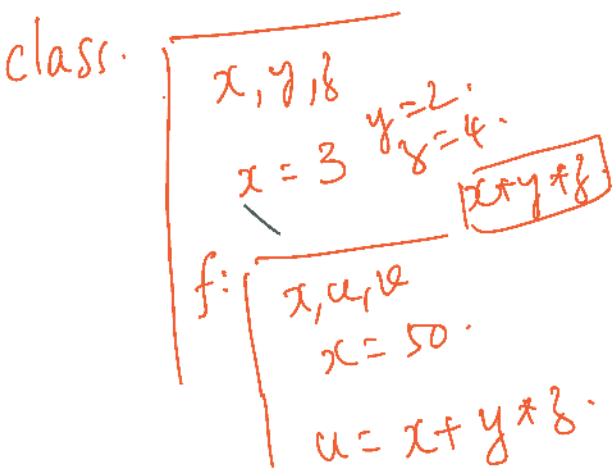
↳ these variables are "present" in between invocations of the function.

May not be accessible.

Scope -

one can use all the variable x, u, v, y, z inside f.

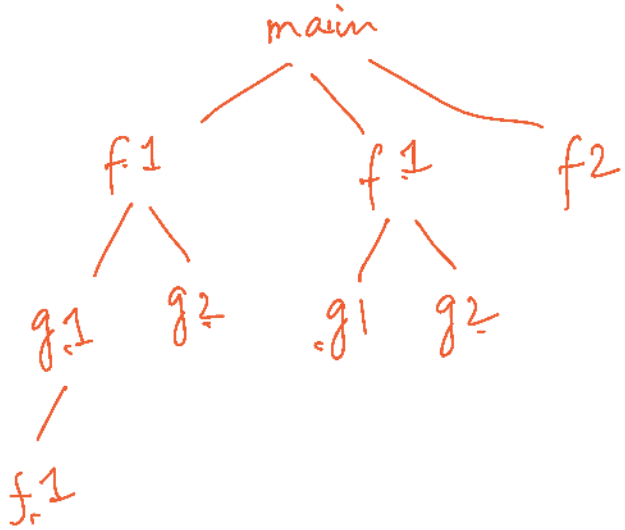
x = 4.



(11).

(58).

Stack allocation.



The function that was invoked last exits first.

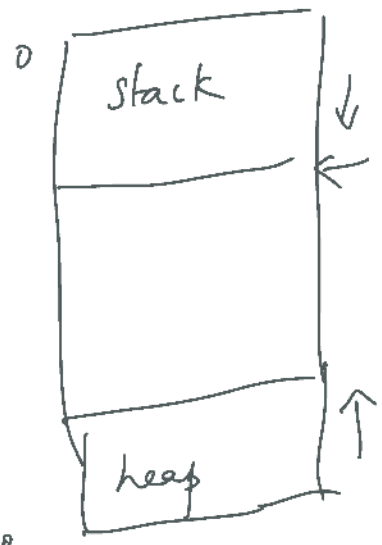
Each time a function is called, you add details about that invocation to the stack.

When the function ends, erase these details from the stack.

activation record.

various pointers.
parameters
local variables.
temporary variables

Control link, return address
return value address.
access link.



Relative addressing.

SP. stack pointer.

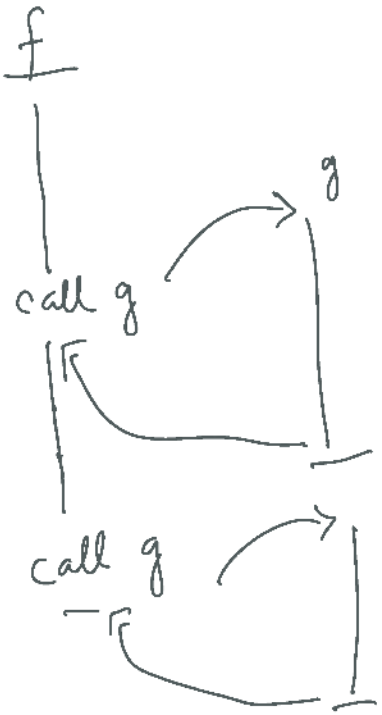
system register.

whenever a function is called,
place AR at location given by SP

activation record.

when a function call ends
— you need to delete the activation record.
you need to know how much to delete

control link — stores the old value of SP.
↳ points to the AR of the caller.



return address
- address of code (in the caller) to which control should return on end of function

return value address.
- where to place the return value in the caller's AR.

---

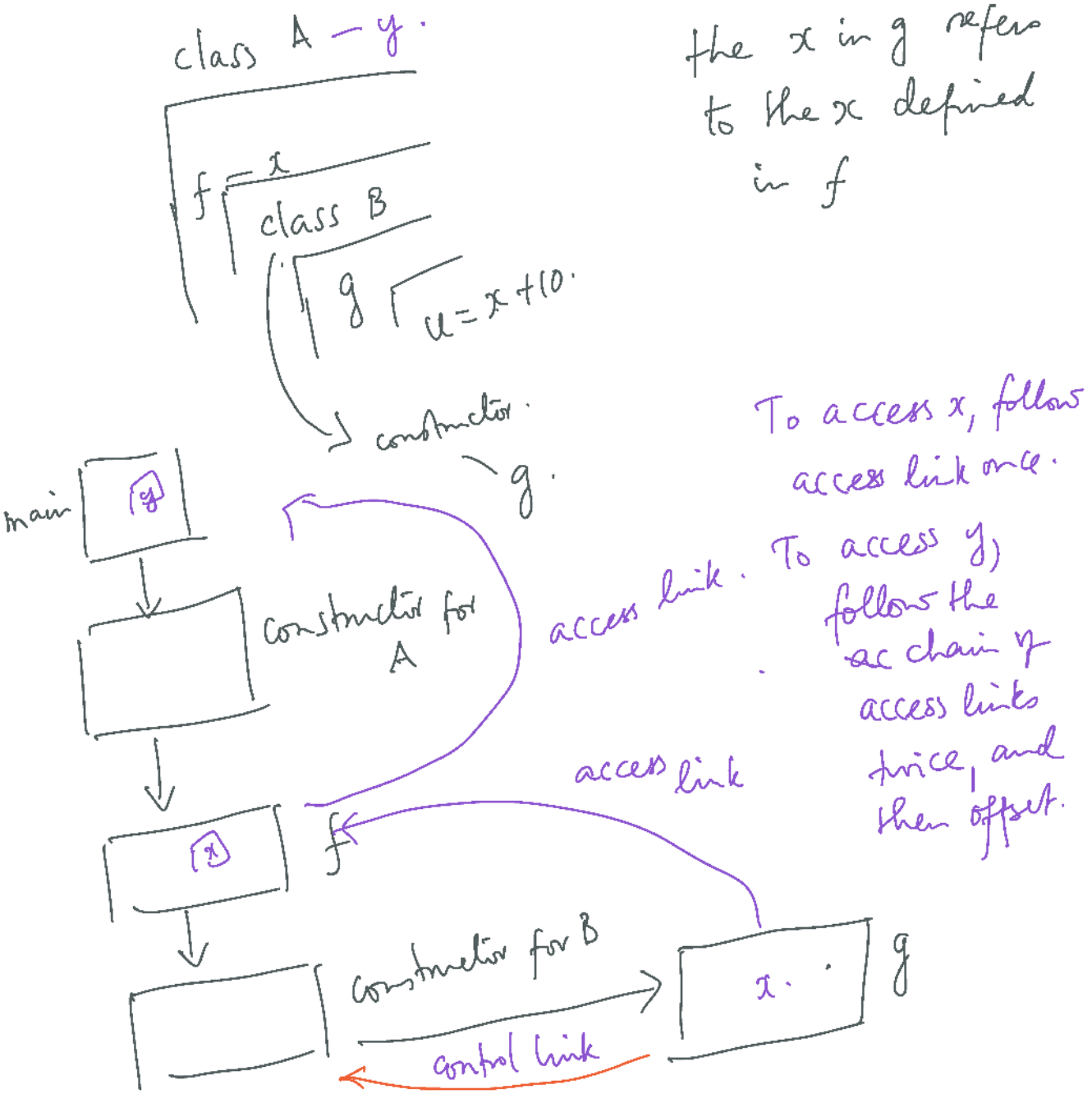
f
int x, y, z

g | a, v, w.
v = x * y

to access x and y, one needs to go to the AR of the caller.
might not always be able to determine it from the control link. Need an access link relative to which addresses of variables are easily computable..

Static scoping.

Any reference to a non-local variable refers to the closest enclosing definition (in the program).



Inside g, you might refer to some variables defined in f, some defined inside class A, some in main.

for varying length data, / part of memory that is not as structured as the stack.
store them on the heap, and store addresses in the stack.

Class B.

constructor

g

g cannot access local variables inside constructor.

Nested function definitions. Pascal.

C / C++ / Java – no nested fn definitions.