Week 4

Memory Areas

Stack **

- local variables
- function arguments
- return addresses
- temporary storage

} managed automatically

Heap ***

- dynamically allocated memory *] manual management
  creates lots of problems -- :(
  ↳ develop has to ~~maintain~~ / keep track of.
  - valid length/size of data
  - valid pointers, etc

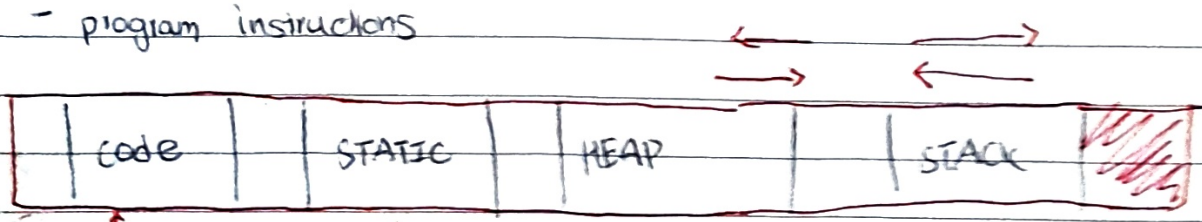
+ free *

Global / Static *

- part of program's binary.) doesn't change really

Code **

- program instructions



fixed size, architecture specific

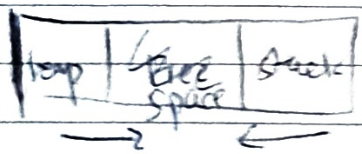
fixed size

changes during runtime

Areas may grow / shrink.

(heap: stack grows in opposite direction toward free space)

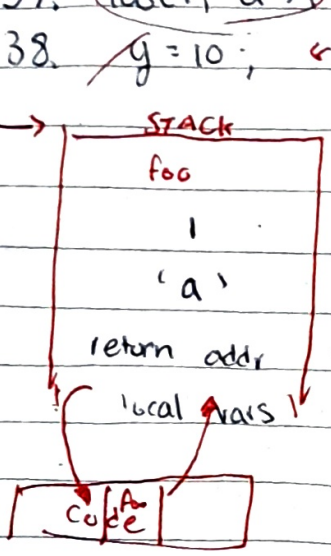
Code manages heap, stack memory regions, where to access / store data, what to do when -- etc.



Function Call

Stack

37. foo(1, 'a'); ← 0x1008
38. y = 10; ← 0x1000



1. First, push arguments onto stack
2. Push return address onto stack
3. Then, jump to function code
4. Increment stack pointer to allocate space for local vars.

After executing the code,

- pop local variables & arguments
- push the return result (one final value remaining) — r.v
- jump to return address

Heap (* dynamic allocation)

(in java) obj = new object();

pointer stored in stack

object stored in heap.

* Needs free space with consecutive space available to allocate the object. (request)

Memory Allocation functions (malloc)

returns a "pointer to void"

∴ needs to be casted ? — add interpretation to retrieved memory.

void * malloc(size_t size) — sizeof(dtype) x num

↳ typedef unsigned int size_t

Calloc

calloc(size_t num, size_t size)

num: # of blocks size: size of each block

- all blocks set to 0.

free(void *ptr)

- deallocate data.

```
int *ptr = (int *) malloc
→ free((void *) ptr);
  ptr = NULL;
```

set back to null to ensure deallocation has happened (ease of debugging)

realloc

if null, behaves the same as malloc

realloc(void *ptr, size_t new_size)

attempts to resize pre-allocated memory.

- may require NEW block of memory
  ∴ *ptr may change careful!

Warnings

- Free the space after use
- Don't ~~double~~ free un-allocated space
- Don't use de-allocated memory
- know the allocation size-requirement (sizeof(ptr) don't do, 10 in sheep)
- check malloc doesn't return NULL