morbius@cs.rutgers.edu

CS 211: Computer Architecture

6/26

Data Types & Constant:

a) Numeric Integers
- → Decimal
- → Hexadecimal (dec. val. = 24)
- → octal (dec. val. = 14)
- → Binary

b) Char & String
- → char c = char
- → char* c = String

* c) size_t
- Unsigned integer type that is the result of the size of operator
- = 'long' type
- allows portability among diff. platforms

✡ d) const. qualifier: main purpose = data isolation

study this!
- Case 1: *p is const., p could change
- Case 2: p is const., *p could change
- case 3: p is " , *p is const.

e) Structure // memory = ~~sum of member~~ all of its members occupy

← Purpose: express abstract data obj. via encapsulation

```
struct TokenizerT_
{
    member list
};
```

f) Union // memory = largest member occupy

g) Enumeration type
- purpose: refer to a countable set (eg. set of events)

h) Functions
- Also a DATA!
- called by name
- Recursive

] argc value = 1 + number of command line arguments

- multiple outputs? **p
  ↳ p still points to the object with an intermediate pointer

$$\begin{tikzcd} p \arrow[r] & \boxed{\text{P\_intermediate}} \arrow[r] & \boxed{\text{object}} \end{tikzcd}$$

i) type def
  ↳ Purpose: create an alias name for data types
  - typedef vs #define

Pointers
a) Pointer to function
  ↳ Purpose: trigger mechanism via callback
  - No diff. between &function and function

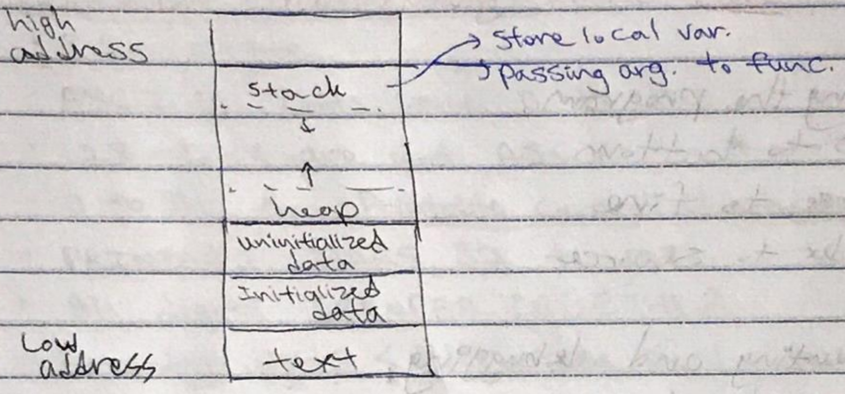
b) Regular pointer
  ↳ Diff. between variable and pointer is that the value (content) of a pointer (eg Zp) represents an address)
  ↳ & is the reference operator: get address
  ↳ * = dereference " : get the target pointed by a pointer

c) void* pointer
  ↳ Purpose: generic programming (can hold the address value of any data pointer type)

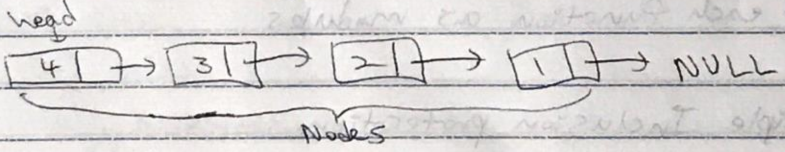
Dynamic Memory:
1) Allocation and free
  - void* malloc (size_t size)
  - void* calloc (size_t num, size_t size)
  - void* realloc (void* ptr, size_t size)
  - void free (void ptr)

Stack overflow: Stack runs out of memory → crash
fragmentation: heap being stored as noncontiguous / disconnected blocks
# malloc() = # free() to avoid memory leak

2) Memory Organization



3) Memory management via linked list
↳ Provide a generic dynamic data container



node_t* head = malloc (sizeof (node_t)); //assign head
i) Traverse
ii) Add to last
iii) Remove first item
iv) Remove specific item

Preprocessor:
Two phases { 1) Preprocessing
2) Compilation
* Use macro and conditional compilation together for debugging

Program Structure:

Source files - function definitions, global var, Static var.
(.c file) ↳ Scope

Header files - function declarations and macros that
(.h file) need to be exposed can be included in header
→ Don't define global var. here!

When designing the program,
- from top to bottom
- from coarse to fine
- from header to source

When implementing and debugging,
- from bottom to top

* Always define the abstract object first
* Treat each function as modules

a) Multiple Inclusion protection

$$\begin{tikzcd} \boxed{h1} \arrow[r] \arrow[rr, bend right] & \boxed{h2} \arrow[r] & \boxed{S1} \end{tikzcd}$$

b) 'extern' and 'static' → can only use that var. only in that Source file
↳ share variable among source files

- Static for internal variable
* Initialization will only occur once

Build Process and ☆ makefile ☆know this!
1. Compile each .c file into individual object file
2. Link all object files to generate an executable file.
3. Execute