Section 9
Generic Data

void*. Why'd it have to be void*?

CS 107A, Autumn 2021
Andrew Benson (adbenson@)

Don’t forget to start recording

Unix Tip Spotlight

- Exiting gdb, ssh sessions, terminals
  - <CTRL+D>: send EOF to the current program
  - Causes gdb, ssh, shells, etc to quit

Announcements

- Gradebook updated
  - Please check that the first 1:1 and your attendance is correct
- Midterm Review Session
  - Saturday 10/23 4-6pm in Huang (ideally we'll takeover a room)
  - Problems released Thursday or Friday (likely near identical to "Still More Practice Problems" on the course webpage - might replace a problem)
  - Snacks
- Next week Tuesday: Midterm
- Next week Thursday: Possibly feedback session

Bird's Eye View

Day	Week 5 Monday	Tuesday	Wednesday	Thursday	Friday	Week 6 Monday	Tuesday	Wednesday
CS 107A		Section: Generics		Section: More Generics			Section: Intro x86	
CS 107	Lecture: More Generics		Lab 4: `void*` / Function Pointers		Lecture: Intro x86	Lecture: x86-64 ALU		Lab 5: Assembly
CS 107 assignments			assign3 due, assign4 released				Midterm	assign4 due next Wed

Agenda

- Motivation for Generics
- `void*` in practice
- `void*` Pitfalls
- Code Exercises

Motivation for Generics

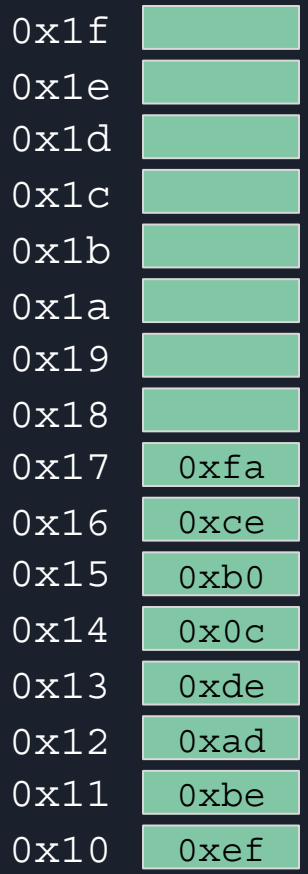
A write function...for `char`

Client code calling our function

```
char box; // Assume box is located at 0x10
char value = 0;
write_char(&box, value);
```

Library function we implement

```
void write_char(char *ptr, char value) {
  *ptr = value;
}
```



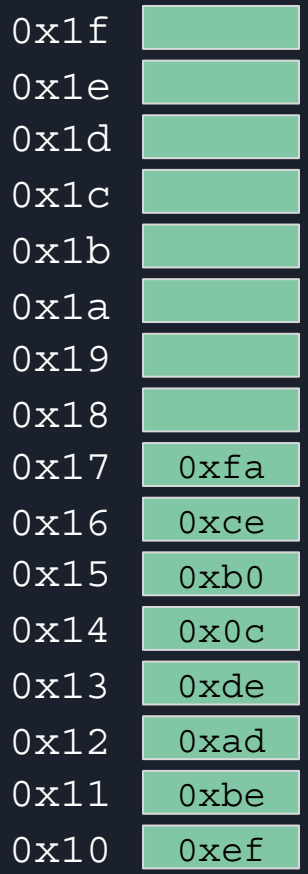
A write function...for `int`

Client code calling our function

```
int box; // Assume box is located at 0x10
int value = 0;
write_int(&box, value);
```

Library function we implement

```
void write_int(int *ptr, int value) {
  *ptr = value;
}
```



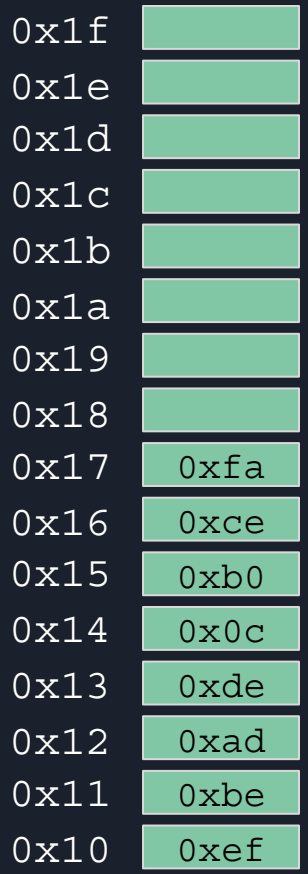
A write function...for `long`

Client code calling our function

```
long box; // Assume box is located at 0x10
long value = 0;
write_long(&box, value);
```

Library function we implement

```
void write_long(long *ptr, long value) {
  *ptr = value;
}
```



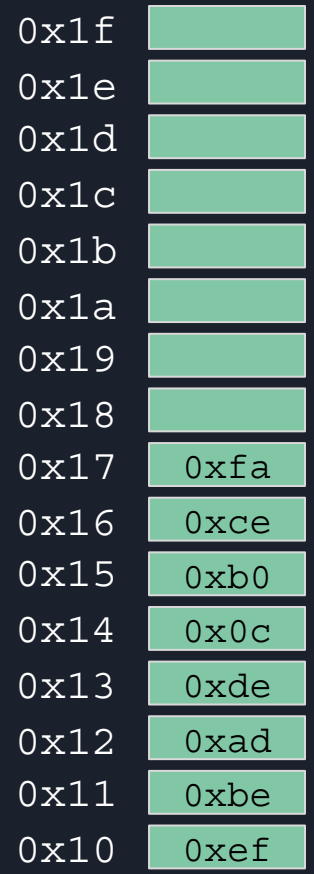
A write function...for `char*`

Client code calling our function

```
char *box; // Assume box is located at 0x10
char *value = NULL;
write_charp(&box, value);
```

Library function we implement

```
void write_charp(char **ptr, char *value) {
  *ptr = value;
}
```



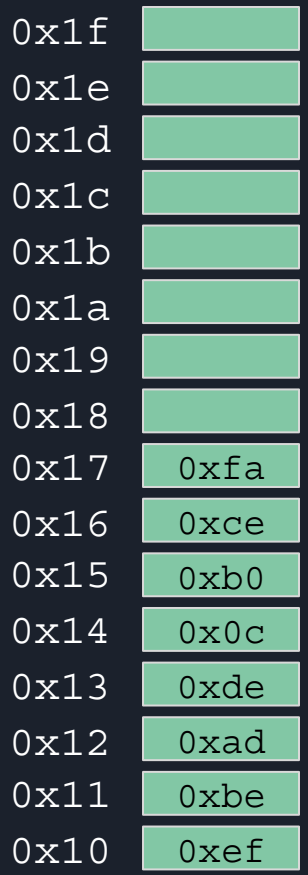
A write function...for `unspecified type`

Client code calling our function

```
Psych! You don't get to know. Gotta handle
any type the client throws at you!
```

Library function we implement

```
void write_smth(void *box, void *value) {
  // what do????
}
```



`void*`

- Represents a generic pointer
  - Points to something, but since it could anything, we use this type when we don't want to assume what we're pointing to
  - It's a pointer not because we like pointers, but because it's the only way to twist C into hiding a type
- You can't dereference it, since that would require knowledge of what we're pointing to
- You can't do pointer arithmetic, since that would require knowledge of what we're pointing to

`void*`

- You can’t do...anything!
  - Unless you’re told more information about the type you’re pointing to
    - 1) Maybe you’re told the size of the type you point to – allows you to use memcpy to copy or move the values around
    - 2) Maybe you’re given a function pointer that you’re told you can use to manipulate values of this mysterious type

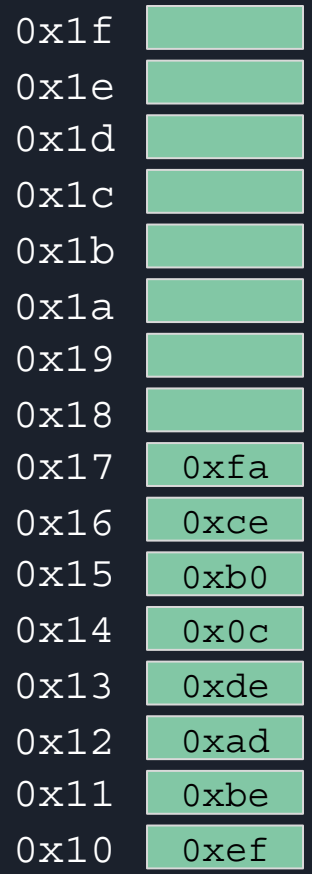
A write function...for `unspecified` type

Client code calling our function

```
Psych! You don't get to know. Gotta handle
any type the client throws at you!
```

Library function we implement

```
/**
 * @param box Pointer to location to write to
 * @param value Pointer to value to write
 * @param size Number of bytes value takes up
 */
void write(void *box, void *value, size_t size)
{
  memcpy(box, value, size);
}
```



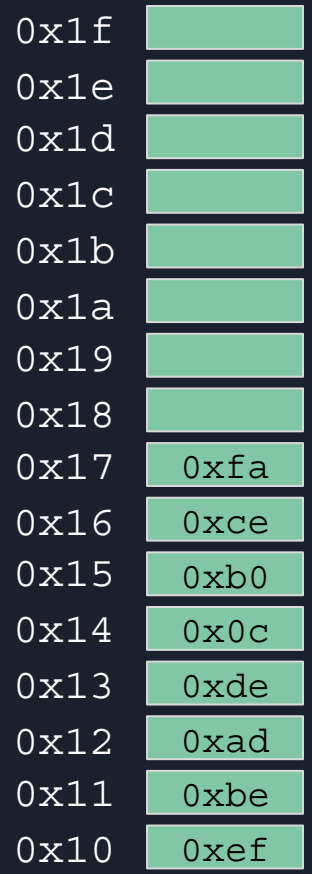
A write function...for `unspecified type`

Example client code, for `int`

```
int box; // Assume box is located at 0x10
int value = 0;
write(&box, &value, sizeof(int));
```

Library function we implement

```
/**
 * @param box Pointer to location to write to
 * @param value Pointer to value to write
 * @param size Number of bytes value takes up
 */
void write(void *box, void *value, size_t size)
{
  memcpy(box, value, size);
}
```



`void*` in practice

Example 1

```
void *malloc(size_t size);
```

```
void free(void *ptr);
```

Example 2

```
void *memcpy(void *dest, const void *src, size_t n);
```

`void*` pitfalls

Pitfall 1

```
int *ptr = malloc(sizeof(int));

int value = 5;

memcpy(ptr, &value, sizeof(int));
```

Pitfall 2

```
void write(void *ptr, void *value) {
  *ptr = *value;
}
```

Pitfall 3

```
void *get_third_elem(void *arr) {
  return arr[3];
}
```

Code Exercises

```
git clone /afs/ir/class/cs107a/WWW/git/section8
```