CS 222: Pointers and malloc/free

Chris Kauffman

Week 4-2

Logistics

Reading

- Ch 8 (pointers)
- Review 6-7 as well
- Ch 9 (file i/o) next week

Feedback

50% Finished with CS 222

- Time to evaluate and adjust as needed
- Anonymous feedback forms distributed

HW 4 now up

- Due next week Tuesday
- More advanced struct, arrays
- Memory allocation

Goals

- Dynamic allocation
  - `malloc`
  - Casting
  - `sizeof`
- Practice

Pointers

- What are they?
- How do we
  - Define, Declare, Access, Assign
- What are the two interesting ops on pointers?
- How are arrays and pointers related?

Exercise: Pointer Max

```
void pointer_max(double *a, double *b, double *max);
```

If the number `a` points at is bigger than what `b` points at, set `max` to what `a` points at. Otherwise set it to what `b` points at.

Examples

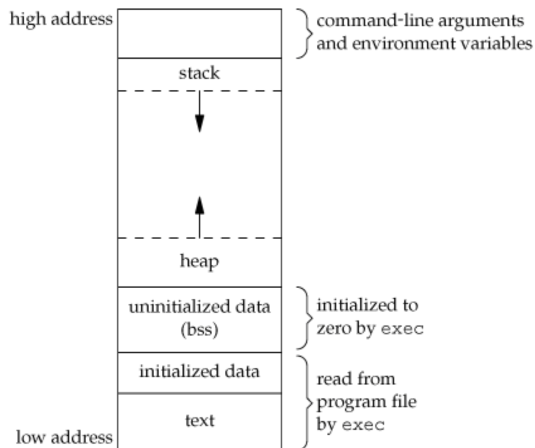
```
double x=7, y=4, max=99;

pointer_max(&x, &y, &max);
// max is now 7

x=-2;

pointer_max(&x, &y, &max);
// max is now 4
```

Remember Memory Layout



Stuff we've had so far is on the stack
What about that other part?

Stack and Heap

Stack

- Grows when functions get called, shrinks when functions finish
- Compiler knows how much to shrink and grow stack
  - For this function I need 2 `ints` and an array of 10 `doubles`
  - 2*4 + 10*8 = 88 bytes
- Stack space is there for you automatically

Heap

- For memory with size not known at compile time
- Used for *run-time allocation*
  - Read `n` from the user
  - Allocate space for `n` integers
- Programmer (you) must manually manage heap space
  - With help from libraries

`malloc` and `free`

`malloc(n)` Allocate `n` bytes somewhere on the heap

- used as `void *p = malloc(n);`
- `p` now points at memory on heap which can be used
- Allocation may *fail* - not enough memory

`free(p)` Deallocate memory pointed to by `p`

- Memory available for further calls from `malloc`
- Gives errors if `p` doesn't point to `malloc'd` memory

Allocating Useful Stuff

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

- `size_t` is an integer-like value (probably `long` on most systems)
- Usually want `int *`, `double *`, `planet_t *`, not `void *`
- Need to figure out how many bytes required
- Use two C features for this: sizeof and casting

Casting

Force conversion of one type to another

Numerical `int i = (int) 45.3 * 0.4432;`

Pointer `char *str = (char *) malloc(100);`

Pointer `planet_t *p = (planet_t *) malloc(100);`

Gross `double d = (double) 'H';`

Old School `int ip = (int) &i;`

Bad Bad `double q = (double) str;`

Compiler, I'm removing the safety net because it's in the way.

sizeof()

Like a function that returns number of bytes for a type

- `sizeof(int)` is # bytes an integer uses
- `sizeof(planet_t)` is # bytes an `planet_t` uses
- `sizeof.c`

malloc useful stuff

Full Malloc: use caste and sizeof()

See malloc.c

```
char *str = (char *) malloc(sizeof(char)*128);
double *arr = (double *) malloc(sizeof(double)*100);
planet_t *p = (planet_t *) malloc(sizeof(planet_t)*9);
```

Casting usually optional

Most compilers don't care if you fail to caste

```
char *str = malloc(sizeof(char)*128);
double *arr = malloc(sizeof(double)*100);
planet_t *p = malloc(sizeof(planet_t)*9);
```

Demonstration: Return an Array of Integers

See `get_ints.c`

Fun things to try

See how much memory you can get: `malloc_madness.c`

`free`

Keep using `malloc` and eventually it wil fail: no memory left

- Use `free` to deallocate
- Important for long-running programs
- Memory leak: `malloc`, lose pointer, can't free, program gets bloated

Exercise: Array Slice

```
int * slice(int *iarr, int start, int len)
```

Creates a slice of an array that is independent from the original

- `iarr` is a pointer an array of integers
- `start` is where to start the slice
- `len` is how long the slice should be
- Return an array that is `len` long with elements copied from `iarr`

Examples

```
int a[] = {0,1,2,3,4,5,6,7,8,9};
int *s;

s = slice(a, 0, 10);
// a = 0 1 2 3 4 5 6 7 8 9
// s = 0 1 2 3 4 5 6 7 8 9
```

```
s = slice(a, 5, 3);
// a = 0 1 2 3 4 5 6 7 8 9
// s = 5 6 7

s = slice(a, 1, 6);
a[3] = 10
// a = 0 1 2 10 4 5 6 7 8 9
// s = 1 2 3 4 5 6
```

Exercise

```
double *sin_sample(double start, double stop,
                   double step, int *len);
```

Creates sample of `sin(x)` function on interval `start` to `stop`

Args/Behavior

- `start` beginning of interval
- `stop` end of interval
- `step` distance between sample points
- `*len` pointer to length, set it
- Allocates space for array of `doubles`
- Fills in the array with samples
- Sets the `len` pointer

Examples

```
int nsamp; double *v;
v = sin_sample(0,PI, PI/2,
               &nsamp);
// v = {0.0, 1.0, 0.0};
// nsamp = 3;
free(v);
v = sin_sample(PI/2, 1.5*PI, PI/4,
               &nsamp);
// v = {1.0, 0.7, 0, -0.7, -1};
// nsamp = 5;
free(v);
v = sin_sample(PI, PI, 0.1,
               &nsamp);
// v = {0.0};
// nsamp = 1;
```