C: Dynamic Memory

CS 62 - Spring 2016
Michael Bannister

This Week

- Weekly Assignment
  - Bounded heap based priority queue
  - Implemented as an ADT
- Weekly Lab
  - Doubly linked lists in C
  - Dynamic memory management

Dynamic Memory Management

Java

- Everything (well most things) are objects an allocated on the heap
- Variables contain references (similar to pointers) to objects
- The heap is garbage collected

C

- Every thing is primitive and can in principle be stack allocated
- Stack variables are de-allocated when scope exits
- Heap allocation and de-allocation is done by the programmer
- You are the garbage collector!

Memory Allocation

We allocate memory with: `void* malloc(size_t size);`

- Allocates size man bytes on the heap
- Ignorant of type of data you are allocating
- Implicitly casts from `void*` to other pointer types
- Use `sizeof` function to get the size of a type

Examples

- `int* A = malloc(10 * sizeof(int)); // Array of 10 ints`
- `node* N = malloc(sizeof(node)); // A single node`
- `char* str = malloc(100 * sizeof(char)); // String of length 99`

Memory Deallocation

Deallocate memory with: `void free(void* ptr)`

- Deallocates memory allocated with `malloc`
- Does nothing if `ptr` is `NULL`
- Undefined if `ptr` did not come from `malloc`
- Undefined if `ptr` has already been freed

Common error

- Use of a pointer after free
- Double free
- Memory leaks (not freeing)

After you call free set the ptr to NULL!

Forward Declared struct

stack.h:

```
typedef struct stack stack;
```

stack.c:

```
struct stack {
  /* details */
};
```

Observations

- Outside of stack.c only pointers to stack are allowed
- Cannot malloc a stack outside of stack.c
- Must have a "creator" function which allocates a stack
- Hides fields of stack from users

Example Code

(Linked List)

A Tour of the Standard Library

Take a look at cppreference.com