C: Functions and Pointers

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

Array Example

(Show Completed Array Example)

Separate Compilation

Header files (`*.h`)

- Contain declarations and constant definitions
- "Copied" into files with the `#include` directive
  `#include` < ... > for system headers and
  `#include` " ... " for user headers
- Cannot be included twice; use guards
  (see example)

Separate Compilation

Implementation files (`*.c`)

- Contain the definitions of the the items declared in the corresponding header files, i.e., `my_functions.c` would contain the definitions of the items in `my_functions.h`.

A Tour of the Standard Library

Take a look at cppreference.com

Abstract Data Types (ADT)

Opaque Data State

- struct storing the mutable state of the type
- Implementation unknown to user of ADT
- Manipulated through abstract operations

Abstract Operations

- Manipulate the data type
- Implementation unknown to user
- Behavior defined relative to each other

Example Code

Bounded Stack ADT

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!