Lecture 32: Arrays in C

CS 62
Fall 2016
Kim Bruce & Peter Mawhorter

Lab This Week

- Convert singly-linked list to doubly-linked.
- Review linked lists -- will write in C
- Assignment: Animals game that we talked about in Java. Java code with lecture 15, but do in C.

Review

- Functions (& types) must be declared before used (*called function prototypes*):
  - Declare: int sum(int x, int y);
  - Then use, e.g. z = sum (3,7);
  - Can define anywhere (early or late)
- Call by value is assignment
  - Makes a copy of arguments

Review Call-by-Value

- What happens with:

```
void badswap(int m, int n) {
        int temp = m;
        m = n;
        n = temp;
}
...
swap(s,t)
```

No effect on arguments s, t!

Fix w/Pointers

- Modify:

```
void swap(int *m, int *n) {
        int temp = *m;
        *m = *n;
        *n = temp;
}
```

- New swap works!
- if s = 3 and t = 7
  - swap (&s, &t) exchanges values

Pointers & Addresses

- int *p // p is an address holding an int
- int q // &q is an address holding an int
  - Sometimes called l-value of q
  - Thus p = &q is legal, as is (*p) + q
  - *(& x) = x // as values
- void* is pointer to anything
  - No type safety!!!

Exercises

- int p = 6;
- int * q = &p;
- *q = 47;
- What are values of p and q?

Memory Management

- Java
  - Everything (excepts primitive types) is an object and heap allocated using "new C(...)"
  - Variables contain references (pointers) to objects
  - Heap is garbage collected
- C
  - Everything is primitive and can be allocated on stack
  - Stack variables deallocated when exit scope of declaration
  - Heap allocation & deallocation up to programmer
  - You are garbage collector!!

Allocating Memory

- Allocate memory w/ void* malloc(size_t size)
  - where size_t is unsigned in
  - Allocates size bytes on heap
  - It doesn't know or care what kind of data using
  - implicit cast from void* to actual pointer type
  - Use sizeof function to get size of types

Examples

- int * A = malloc(10 * sizeof(int)) // array of 10 ints
- node * N = malloc(sizeof(node)) // single node
- char * str

Deallocating Memory

- Use: void free (void* ptr)
  - Deallocates memory allocated by malloc
  - Does nothing is ptr is NULL
  - Undefined if ptr not come from malloc or ptr already freed.
- Common errors:
  - Use of stale pointer, i.e., points to freed memory
  - Double freeing pointer
  - Memory leaks

After freeing ptr, set it to NULL!!

Separate Compilation

- Header files (*.h)
  - Contain declarations and constant defs
  - "Copied" into files with "include" directive
    - #include <...> for system headers and
      #include "..." for user headers
  - Cannot be included twice!
    - See linked lists "ifndef", "define" clauses
  - Information hiding: "Abstract data type" (ADT)

Separate Compilation

- Implementation files (*.c)
  - Contain definitions of everything declared in .h file
  - myfile.h paired with myfile.c

Explore Singly-Linked List

- Look at linked_list.h (header file)
  - Provides publicly available info
  - Imported by implementation (.c) and users: main.c
  - #ifndef and #define are used to make sure only 1 copy imported into using program
  - Method parameters that are pointers usually reflect "out" parameters (something to be changed). See implementations!
  - Most ops return list so can chain commands
  - Ignored in main.c (*just style of writing*)

main.c & linked_list.c

- main.c
  - Build and manipulate list
  - Must destroy at end
- linked_list.c
  - Forward declaration of linked_list_node
  - Create functions: malloc, node->data *notation*
    - *Always check result of malloc!!*
  - Destroy: free
  - Notice functions end with return self