C Lab 2

Intermediate Pointers & Basic Structures

Goals

- Review
  - Pointers
  - Referencing/Dereferencing
- free
- malloc
- structs

Review: What are Pointers?

- A pointer is an address on either the stack or heap.
- EX: “double *” **should** address a double in memory.
- For the pointer to contain data, some other function must create the data it will point to.
- This is typically a call to malloc.

Getting Pointer/Reference

- To get pointer to something, use ‘&’
- ‘&’ allows to pass items by reference
- To dereference or get item pointed to use ‘*’
- ‘*’ is the opposite of ‘&’

Pass by Value

Pass by value:

```
void plus(int num){
  num++;
}
```

```
void main(){
  int num = 3;
      plus(num);
  printf(“%d\n”, num);
}
```

What does main print?

Pass by Reference

Pass by reference:

```
void plus(int *num){
  (*num)++;
}
```

```
void main(){
  int num = 3;
      plus(&num);
  printf(“%d\n”, num);
}
```

What does main print now?

Void*

- “void*” may point to arbitrary types (i.e. int*, char*, etc.)
- Can be cast to appropriate types

Malloc

- Malloc returns a void* pointer
  - The memory it allocated doesn’t inherently have any type
- Malloc returns null if it fails. Always check the return value!

Structs

- Personalized types, somewhat like classes
- May contain items of choice
- Often the basis of (data) structures

Structs

```
typedef struct {
  int *buffer;
  int buffersize;
  int length;
} arraylist;
```

Editing Struct Fields

- You may declare structs on the stack
- You may access/edit fields of struct using ‘.’
- Think about why this works (Hint: pointers)

Editing Struct Fields

```
arraylist a;
a.buffer = NULL;
a.buffer_size = 0;
a.length = 0;
```

Editing Struct Fields

- You may declare structs on the heap
- Now you access/edit fields using ‘->’
- This syntax is more helpful visually

Editing Struct Fields

```
arraylist *a = (arraylist *)malloc(sizeof(arraylist));
a->buffer = NULL;
a->buffer_size = 0;
a->length = 0;
```

Default values and null

Null is a pointer value going “nowhere”

If a pointer shouldn’t be pointing at anything set it to null

In C, values aren’t initialized to any particular value

-They take whatever happened to be in memory

Memory Management

- You must free what you malloc (heap)
- Stack manages itself

arraylist ***a*** = (arraylist *)malloc(sizeof(arraylist));

free(**a**); //yaaaaaaaaay

Memory Management

- Do not free what you did not malloc!!!
- Do not free address consecutively!!!

```
int num = 3;
free(&num); // :,O
```

```
int *num = malloc(4)
free(num); //yaaaayyy
free(num); //staaahp
```

Memory Takeaways

- Only free what has been malloc’d
- Only free malloc’d memory once

- For more on stack vs. heap:

  http://gribblelab.org/CBootcamp/7_Memory_Stack_vs_Heap.html#sec-4

References

From the C reference manual:

- Pointers: 4.5.2, 5.3
- Structs: 5.6
- Free: 16.1 (near end), 16.1.1