C Lab 2

Intermediate Pointers & Basic Structures

Goals

- Review
  - Pointers
  - Referencing/Dereferencing
  - free
- realloc
- structs
- ArrayList

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 Copy

Pass by copy:

```
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 * and realloc

- “void *” may point to arbitrary types (i.e. int*, char*, etc.)
- Can be casted to appropriate types
- realloc increases the size of memory allotted to pointer
- Preserves data pointed to by original pointer
- Original pointer is NULL, if space is found elsewhere

Realloc and Equivalent

```
ptr = malloc(2);
ptr = realloc(ptr, 1000);
```

Why not:

```
ptr = malloc(2);
realloc(ptr, 1000);
```

```
ptr = malloc(2);
ptr2 = malloc(1000);
memcpy(ptr2, ptr, 2);
free(ptr);
ptr = ptr2;
ptr2 = NULL;
```

Structs

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

Structs

```
typedef struct arraylist {
    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;
```

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

Connect Thoughts

- Begin the lab exercise
- Where/When might realloc be useful?
- Where/When might free be useful?