CSE 410 Section 3: Pointers are Meta

Pointers vs. Arrays

Pointers and arrays are very closely-related in C, but there is one subtle difference. Pointers can be reassigned, whereas arrays cannot. Let’s look at a few examples that exhibit the differences:

```c
int a[3];
int *p = a;

a[1] = 2; // OK
p[1] = 3; // OK

p = 0x0514; // OK
a = 0x0514; // Error
```

String Escapes

We talked in class about how strings are just arrays of characters that are null-terminated (i.e., end with a “0x00” byte). Try to guess the output of printing the following strings:

1. `printf("There are 0 things you can't do in C.\n")`;
2. `printf("The "\"\0" character literal is a null terminator.\n")`;
3. `printf("You probably don't want to use \0 in string literals.\n")`;

Then check your answer by actually printing them.

What is the difference between the length of a string (the number of characters it represents) and the number of bytes it occupies in memory?
Pointer Practice

Write the output of the following C program:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a[4];
    int *b;
    int *c;
    int i;

    c = a;
    for (i = 0; i < 4; i++)
        a[i] = 100 + i;
    c[0] = 200;
            a[0], a[1], a[2], a[3]);

    c[1] = 300;
    *(c + 2) = 301;
    3[c] = 302;
    c++;
            a[0], a[1], a[2], a[3]);

    c = c - 1;
    *c = 400;
            a[0], a[1], a[2], a[3]);

    c = (int *) ((char *) c + 4);
    *c = 500;
            a[0], a[1], a[2], a[3]);
    return 0;
}
```

Throwback Thursday

In Java, use `System.out.println` on an array. What does the value that got printed represent?
Challenge Problem

Before approaching the problem, structs need to be introduced.

```c
struct Stuff {
    int value;
    int *p;
};

struct Stuff a; // Creates a Stuff struct with identifier "a"
a.value = 4; // "." is used to access the data of the struct.
printf("a.value=\%d\n", a.value); // Outputs "a.value=4".

a.p = &a.value;
*(a.p) = 2;
printf("a.value=\%d\n", a.value); // Outputs "a.value=2".
```

For the challenge problem, you will implement linked list functionality in C.

```c
struct LinkedList {
    struct Node *first;
};

struct Node {
    int value;
    struct Node *next;
};

// Below is an implementation for 'print'.
void print(struct LinkedList list) {
    struct Node *curr = list.first;
    int i = 0;
    while (curr != NULL) {
        printf("Node %d value: \%d\n", i, (*curr).value);
        i++;
        curr = (*curr).next; // Could also use 'curr->value' here.
    }
}

// Complete the methods below. Note that C does not
// have objects, so we pass the list as an argument.
void add(struct LinkedList list, int value);
void removeFirst(struct LinkedList list);
void removeLast(struct LinkedList list);
```