Pointers & Linked Lists

github.com/JKramerDD
Jeremy Kramer

One of the most IMPORTANT concepts in Computer Science. Pointers enable us to create complex data types such as Linked Lists, Trees, and Graphs; all of which are imperative to various focii throughout Computer Science.

Pointer Syntax:

Declaration:

```
type *x;
void *v;
struct type *y;
type z[];
```

Accessing:

x	- A memory address
*x	- Value stored in address (dereference)
y->a	- Value stored in struct ptr y
&varName	- Memory address of normal var
(type)v	- Dereference void pointer into type

Linked List Node Implementation:

```
typedef struct _node
{
        char *data;
        struct _node *next;
} node;
```

Thanks to the typedef, we no longer have to refer to each node as 'struct _node', but instead simply as 'node'.
Each node provides us a place to store some kind of data, and a link to the next node in the list.
Some linked lists are bi-directional and will also contain a 'struct _node *prev;'

Node Initialization:

```
node x;
x.data = "I am a string";
x.next = NULL;
```

We create an instance of our node by saying 'node x;'.
We then assign the node's data and initialize it's next pointer to a null value.

Head Pointer:

```
node * head = NULL;
```

It is imperative to maintain a pointer to the start of the list. Generally declared before any nodes are ever initialized. Without this reference, we will be unable to access our list.

Adding a Node to the List:

```
void push (node *head, char* val){
        node *current = head;
        while (current->next != NULL)
                current = current->next;
        node* x;
        x.data = val;
        x.next = NULL
        current->next = x;
}
```

Pass head and the value, traverse the list until you find a null node, create a node, and set it to next.

Traversing the List:

```
node* find(node *head, char* val){
        node *current = head;
        while (current->data != val && current->next != NULL)
                current = current->next;
        return &current
}
```

Create a new node pointer to traverse the with list. This way you do not change your head pointer.

Linked List Example: