COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 14- Class 1: Linked Lists I

Administrivia

- SPA5 due tomorrow, LPA3 due Friday.
- Semester Feedback Survey will post after class,
    - Will count as Quiz 3 - due Wednesday.
- Quiz 4 will be due Friday.
- All grades will be current by Friday.
- Final Exam is on Monday December 4th, 10:00am-12:50pm - (more on this on Friday)
- *Short Class today and no office hours.*
    - I will hold makeup virtual office hours tomorrow from noon-1:00pm.

Today's Agenda

1. Introduction to Linked Lists

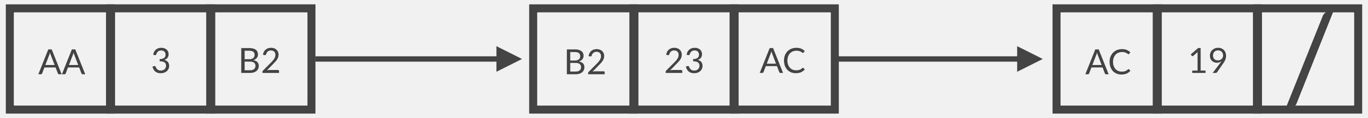
Linked Lists

Data Structures (Review)

- Data structures are a composite of related data items stored under the same name.
- Data structures allows programmers to store data in a more organized fashion.
- You have learned one already… Arrays!
- You will learn more in Computer Science 1
  - Linked List
  - Stack
  - Queues
  - Binary Trees
  - Binary Search Trees
  - Heaps
  - AVL Trees
  - Tries
- If we are covering this then CONGRATLUATIONS! You get a head start to understanding Linked Lists Data Structures!

What is a Linked List?

- A linked list is a sequence of nodes in which each node but the last contains the address of the next node.
- *When to use a Linked List?*
  - You need constant-time insertions/deletions
  - You don't know how many items will be in a list
  - You don't need random access to elements
  - You want to be able to insert items into the middle of the list (Priority Queues)



Setting Up a Linked List in C

- We will use a typedef struct to set up the node that contains the data and pointer to the respective nodes in the linked list.

- Now you may notice **node_s** after struct. Why is that necessary? This allows C to know that the node will point to another type **node_t**. If you don’t, your code won’t compile.

```
typedef struct node_s{
    struct node_s * nextptr;
    int data;
}node_t;
```

Some Things to Know About Linked Lists

- The first node of the linked list is the “head” of the list.
- There are doubly and singly linked list
- In this course we will only observe the singly linked list
- You can only traverse in one direction of a singly linked list.
- Singly linked lists has only one pointer that points in one direction.
- The last node points to NULL.

Connecting Two Nodes

```
node_t *n1;
node_t *n2;
node_t *head;

n1 = (node_t *) malloc(sizeof(node_t));
n2 = (node_t *) malloc(sizeof(node_t));

head = n1;

n1->data = 3;
n2->data = -9;

n1->nextptr = n2;
n2->nextptr = NULL;
```

Linked List Operations

- We can perform the basic operations with a linked lists
  - Insert
  - Remove
  - Display (traverse)
  - Search
  - Empty

Linked List Node Insertion

- *Insert a new node into the list*
- *Insert at the end of the list*
    - Traverse until nextptr is NULL
    - Make last nextptr new node
- *Insert in between two nodes*
    - Traverse to the position of the list
    - You will need reset the nodes pointers to properly maintain the linked list. It’s good practice to draw the list to visually see how pointers work!

Linked List Node Removal

- See if the node even exists in the list
- Similar with inserting between two nodes, you will need to reset the previous adjacent node's next pointer to the old removed node's next pointer.
- Draw the picture to visualize!

Linked List Node Display

- Traverse each node to display the information
- Simple loop traversal

Linked List Node Search

- Traverse the list until the data you are seeking is found.
- Simple loop traversal

Linked List Empty?

- A simple function. Just check if the head is NULL.
- Simple right? 🙂

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course