Linked Lists

Last updated 1/30/23

These slides introduce linked lists

Linked Lists

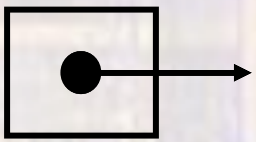
- Motivation
  - Arrays have distinct advantages and disadvantages
    - Easy to use
    - Fixed in size
  - Desire a similar structure but without the disadvantages
  - Linked List
    - Variable size
    - Efficient addition or deletion of elements
    - More difficult to create

Linked Lists

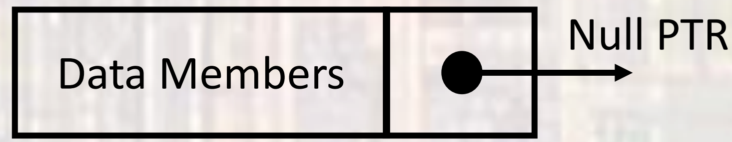
- Basic Structure
  - Each list node is a structure

Can only be traversed in the forward direction

List Node

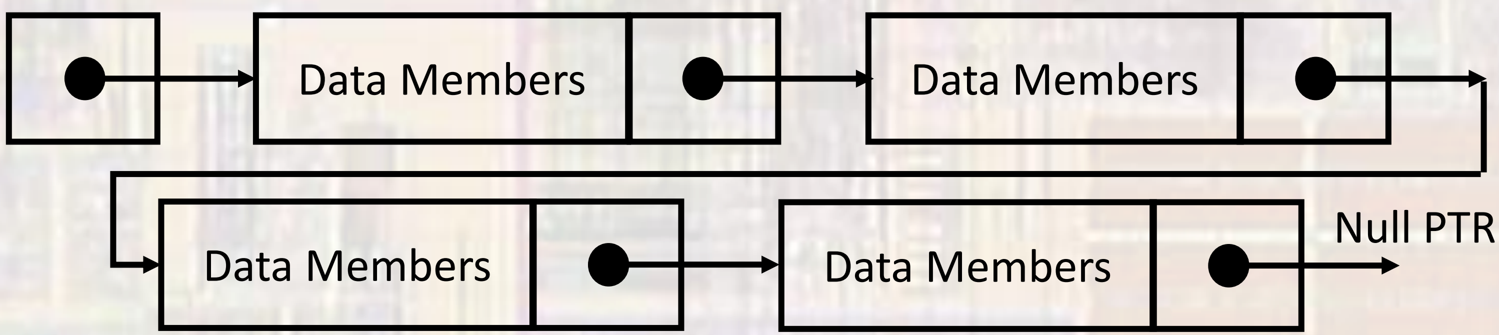


List Head



Last Node

List Head



Linked Lists

- Node structure

```
struct ListNode{
    int id;
    double gpa;
    struct ListNode* next;
};
```

Data (could be anything)

Address of next node

For reasons out of scope for this class – we must use the struct approach to creating the structure (typedef not allowed)

Linked Lists

- Starting a new list

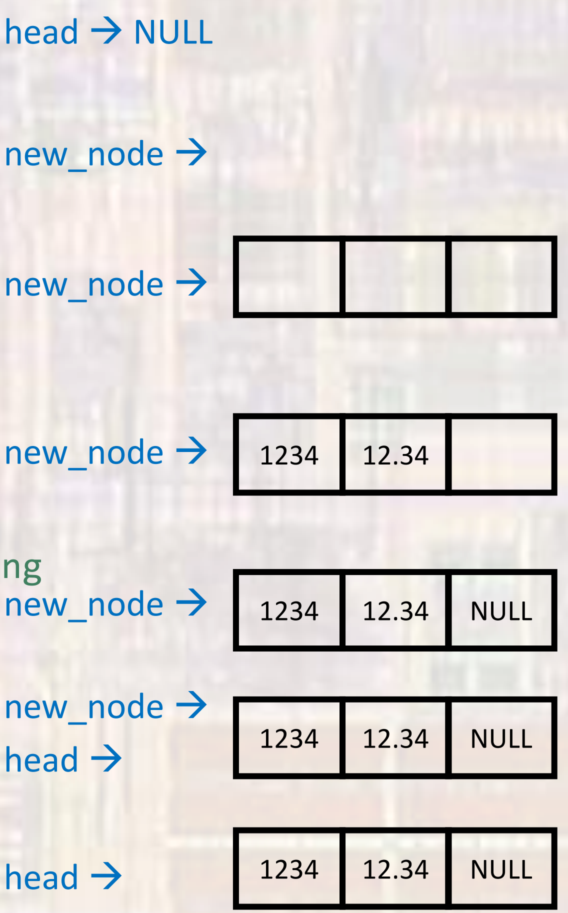
```
// create the list structure
struct ListNode{
   int id;
   float val;
   struct ListNode * next;
};

//
// start a new list
//
// create a pointer to be the head of the linked list
struct ListNode * head = NULL;
```

head → NULL

Linked Lists

- Adding a node to the beginning of the list – 1st time

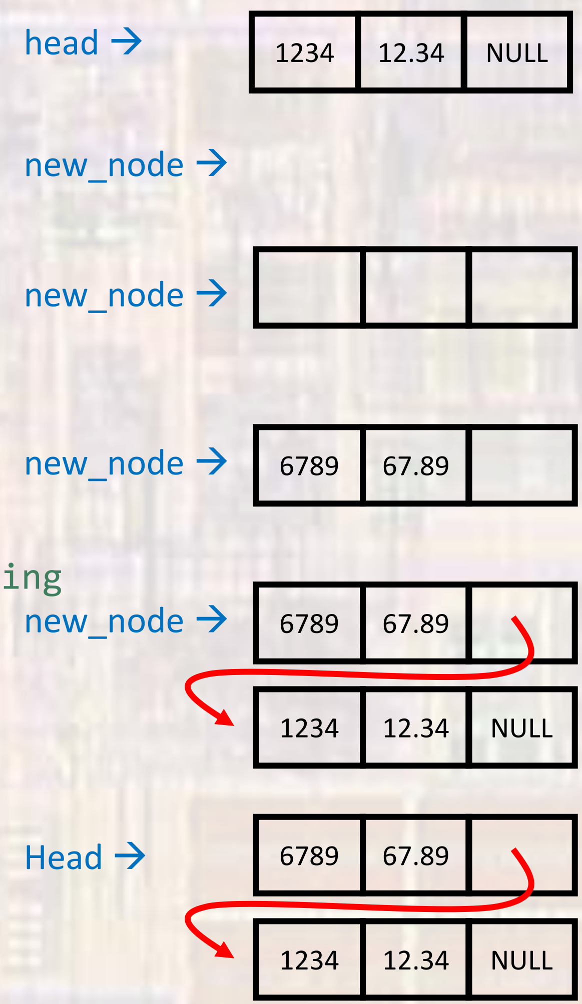
```
//
// add a node to the beginning of the list
//
// create a temporary node for the new node
struct ListNode * new_node;

// allocate memory for the new node
new_node = malloc(sizeof(struct ListNode));

// assign values to the new node
new_node->id = 1234;
(*new_node).val = 12.34;

// point the new node (next) to where head is currently pointing
new_node->next = head;

// point the head to the new node
head = new_node;

// new node no longer needed
free(new_node);
```



Linked Lists

- Adding a node to the beginning of the list

```
//
// add a node to the beginning of the list
//
// create a temporary node for the new node
struct ListNode * new_node;

// allocate memory for the new node
new_node = malloc(sizeof(struct ListNode));

// assign values to the new node
new_node->id = 6789;
(*new_node).val = 67.89;

// point the new node (next) to where head is currently pointing
new_node->next = head;

// point the head to the new node
head = new_node;

// new node no longer needed
free(new_node);
```



Linked Lists

- Example

```c
/*
 * linked_list.c
 *
 *  Created on: Dec 13, 2022
 *      Author: johnsontimoj
 */
////////////////////////////
//
// code to demo linked lists
//
////////////////////////////
#include <stdio.h>
#include <stdlib.h>

// create the list structure
struct ListNode{
    int id;
    float val;
    struct ListNode * next;
};

// function prototype
struct ListNode * add_to_list(struct ListNode * the_list, in                val_val);

int main(void){
    setbuf(stdout, NULL);
    //
    // start a new list
    //
    // create a pointer to be the head of the linked list
    struct ListNode * head = NULL;

    //
    // add nodes to the beginning of the list
    //
    head = add_to_list(head, 5, 5.5);
    head = add_to_list(head, 3, 3.3);
    head = add_to_list(head, 7, 7.7);
    head = add_to_list(head, 1, 1.1);

    // print the list
    struct ListNode * tmp_head = head;
    while(tmp_head != NULL){
        printf("list item is at %p, has values %i %f, and points to %p\n", tmp_head, tmp_head->id, tmp_head->val, tmp_head->next);
        tmp_head = tmp_head->next;
    }

    return 0;
}// end main
```

```c
struct ListNode * add_to_list(struct ListNode * the_list, int id_val, float val_val){
    // adds a new link to the beginning of the provided list
    // returns a pointer to the new head of the list

    // setup new node
    struct ListNode * new_node;
    new_node = malloc(sizeo                ode));
    if(new_node == NUL
        printf("Fai                e memory");
        exit(0
    }

                lues
            ->id = id_val;
        _node->val = val_val;

    // update pointer
    new_node->next = the_list;

    // return a pointer to the new head of the list
    return new_node;
}// end add_to_list
```

Clean up this code – may already be done

```
<terminated> (exit value: 0) Class_Project.exe [C/C++ Application] Z:\msoe_current\ELE1601\eclipse_
list item is at 00C61C88, has values 1 1.100000, and points to 00C61C70
list item is at 00C61C70, has values 7 7.700000, and points to 00C61C58
list item is at 00C61C58, has values 3 3.300000, and points to 00C61C40
list item is at 00C61C40, has values 5 5.500000, and points to 00000000
```

Linked Lists

- Searching a list
  - Traverse the list until:
    - You find the item
    - Hit the end

```
struct ListNode * search_list(struct ListNode * the_list, int id_val){
   // search a list for a given value
   // return a pointer to the node or a null ptr if not found

   struct ListNode * tmp_ptr;
   for(tmp_ptr = the_list; tmp_ptr != NULL; tmp_ptr = tmp_ptr->next){
      if(tmp_ptr->id == id_val)
         return tmp_ptr;
   }

   return NULL;
}// end search_list
```

Linked Lists

- Searching a list

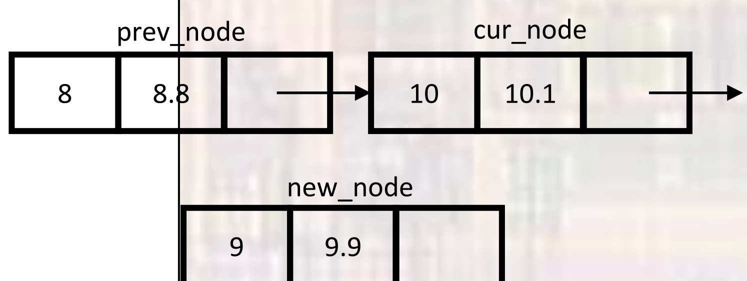
```
...

// search the list
while(1){
   int identity;
   struct ListNode * found_ptr;

   printf("Please enter an ID to search for: ");
   scanf("%i", &identity);

   found_ptr = search_list(head, identity);
   if(found_ptr != NULL)
      printf("ID %i was found, and has a val of %f\n", identity, found_ptr->val);
   else
      printf("ID %i was not found\n", identity);
} // end while

...
```

```
Class_Project.exe [C/C++ Application] [pid: 58]
list item is at 001F1C88, has values 1 1.100000, and points to 001F1C70
list item is at 001F1C70, has values 7 7.700000, and points to 001F1C58
list item is at 001F1C58, has values 3 3.300000, and points to 001F1C40
list item is at 001F1C40, has values 5 5.500000, and points to 00000000
------------------------
Please enter an ID to search for: 7
ID 7 was found, and has a val of 7.700000
Please enter an ID to search for: 4
ID 4 value was not found
Please enter an ID to search for: 3
ID 3 was found, and has a val of 3.300000
Please enter an ID to search for:
```

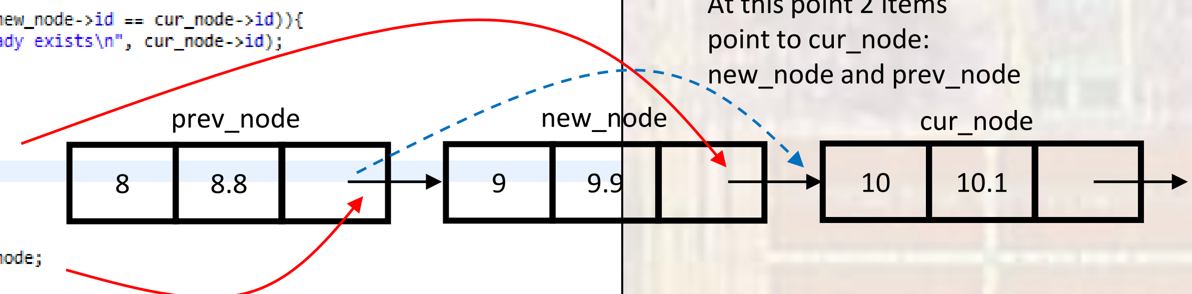
Linked Lists

- Creating an ordered list
  - Traverse list to proper location
  - Insert the new item

```c
struct ListNode * insert_in_list(struct ListNode * the_list, int id_val, float val_val){
    // insert the new node into the_list based on the id value
    // return the modified list

    struct ListNode * new_node;
    struct ListNode * cur_node;
    struct ListNode * prev_node;

    // setup the new list item
    new_node = malloc(sizeof(struct ListNode));
    if(new_node == NULL){
        printf("Failed to allocate memory");
        exit(0);
    }
    new_node->id = id_val;
    new_node->val = val_val;

    // traverse to the spot
    for(cur_node = the_list, prev_node = NULL;
            (cur_node != NULL) && (new_node->id > cur_node->id);
            prev_node = cur_node, cur_node = cur_node->next)
        ;   // do nothing until spot is reached

    // check for duplicate
    if((cur_node != NULL) && (new_node->id == cur_node->id)){
        printf("New ID %i already exists\n", cur_node->id);
        return the_list;
    }

    // insert the new node
    new_node->next = cur_node;
    // check for empty list
    if(prev_node == NULL)
        the_list = new_node;
    else
        prev_node->next = new_node;

    return the_list;
}// end insert_in_list
```



At this point 2 items point to cur_node: new_node and prev_node



Linked Lists

- Creating an ordered list

```
int main(void){
    setbuf(stdout, NULL);
    //
    // start a new list
    //
    // create a pointer to be the head of the linked list
    struct ListNode * head = NULL;

    // insert into list - ordered
    // using ID here
    head = insert_in_list(head, 3, 3.3);
    head = insert_in_list(head, 6, 6.6);
    head = insert_in_list(head, 9, 9.9);
    head = insert_in_list(head, 6, 6.6);
    head = insert_in_list(head, 5, 5.5);

    // print the list
    struct ListNode * tmp_head;

    tmp_head = head;
    while(tmp_head != NULL){
        printf("list item is at %p, has values %i %f, and points to %p\n", tmp_head, tmp_head->id, tmp_head->val, tmp_head->next);
        tmp_head = tmp_head->next;
    }

    // delete items (1 fails because it's not in the list
    head = delete_from_list(head, 8);
    head = delete_from_list(head, 6);

    // print the list
    tmp_head = head;
    while(tmp_head != NULL){
        printf("list item is at %p, has values %i %f, and points to %p\n", tmp_head, tmp_head->id, tmp_head->val, tmp_head->next);
        tmp_head = tmp_head->next;
    }

    return 0;
}// end main
```

```
New ID 6 already exists
list item is at 00781678, has values 3 3.300000, and points to 007816D8
list item is at 007816D8, has values 5 5.500000, and points to 00781690
list item is at 00781690, has values 6 6.600000, and points to 007816A8
list item is at 007816A8, has values 9 9.900000, and points to 00000000
ID 8 is not in the list
list item is at 00781678, has values 3 3.300000, and points to 007816D8
list item is at 007816D8, has values 5 5.500000, and points to 007816A8
list item is at 007816A8, has values 9 9.900000, and points to 00000000
```

Linked Lists

- Doubly Linked List

Can be traversed in either direction