Structures

Topics

- What is a structure
- How to define a structure
- How to populate and use structures

What is a structure

- An array can only contains elements of the same type
- What if you need to store data about a student including the name, and GPA:
  - Name is a string
  - GPA is float
- Instead of using two separate arrays, a structure is a container for related data

How it works

- A structure allows you to store information related to one entity
  - An entity can be a student, a book, a bank customer, and so on
- Each piece of information (member) in the structure can be of any other data type.
- For example, an element could be: name, GPA, Date of birth, and Email.
- You can then search for a student by their name, or DOB, or both.

A structure

- A structure is a data type
  - You can create an array of int, and
  - Similarly, you can create an array of a structure
  - Each element of the array will be made the members of the structure

Working With ListNode

- Declare A List
```
LISTNODE * list;
```
- Declare A List And Initialize It To Null
```
LISTNODE * list = NULL;
```

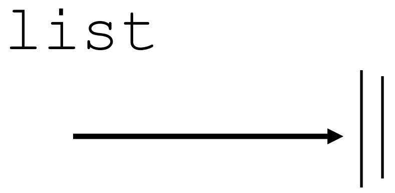
Working With ListNode

- Declare A List

```
LISTNODE * list;
```

- Declare A List And Initialize It To Null

```
LISTNODE * list = NULL;
```



Working With ListNode

- Declare A List

```
LISTNODE * list;
```

- Declare A List And Initialize It To Null

```
LISTNODE * list = NULL;
```

- Declare A List And Allocate It

```
LISTNODE * list = (LISTNODE *)
malloc( sizeof(LISTNODE) );
```

Working With ListNode

- Declare A List

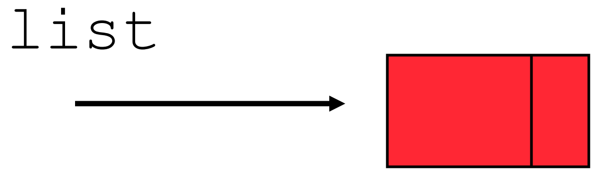
```
LISTNODE * list;
```

- Declare A List And Initialize It To Null

```
LISTNODE * list = NULL;
```

- Declare A List And Allocate It

```
LISTNODE * list = (LISTNODE *)
malloc( sizeof(LISTNODE) );
```



Working With ListNode

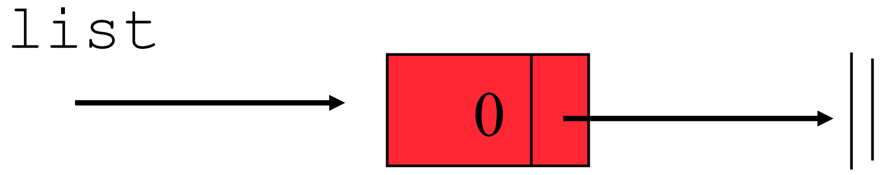
- After Allocating It, Initializing The List

```
initialize( list );
```

Working With ListNode

- After Allocating It, Initializing The List

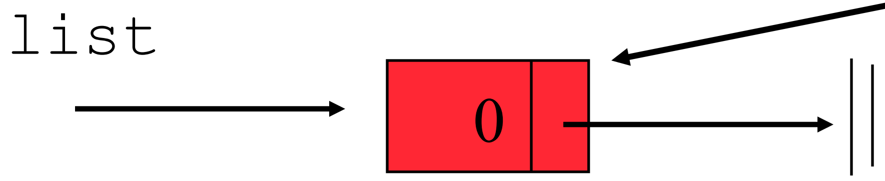
```
initialize( list );
```



Working With ListNode

- After Allocating It, Initialize The List

```
initialize( list );
```



This Opening Element Is Extra And Not Considered To Be "Real" Data On The List….

Insertion Into The List

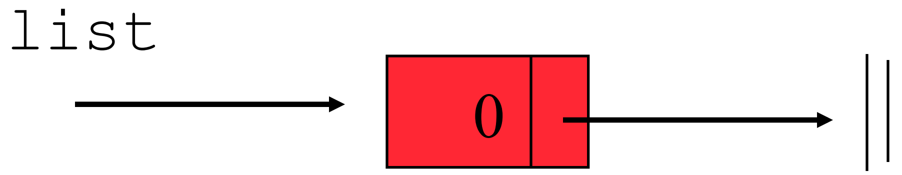
- Data Gets Added At The Front Of The List

```
int data;
LISTNODE * list = (LISTNODE
*)
malloc( sizeof(LISTNODE) );
initialize( list );
insert( data, list );
```

Insertion Into The List

- Data Gets Added At The Front Of The List

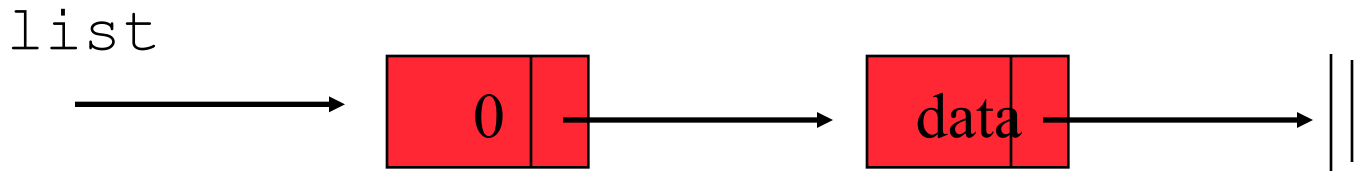
```
int data;
LISTNODE * list = (LISTNODE
*)
malloc( sizeof(LISTNODE) );
initialize( list );
insert( data, list );
```



Insertion Into The List

- Data Gets Added At The Front Of The List

```
int data;
LISTNODE * list = (LISTNODE
*)
malloc( sizeof(LISTNODE) );
initialize( list );
insert( data, list );
```



Insertion Into The List

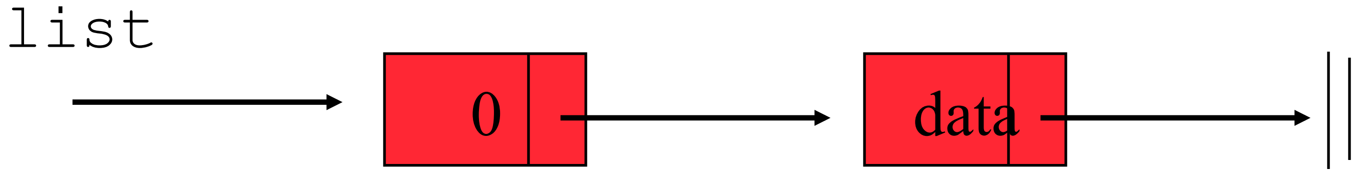
- Insertion Are Pretty Easy To Understand Because
  - New node always goes in the same place – in front of all the other nodes on the list

Lists Know Their Size

- You Can Figure Out How Many Nodes Are On The List…
  - No, The Opening Node That `initialize` Put There Doesn't Count As A Node…

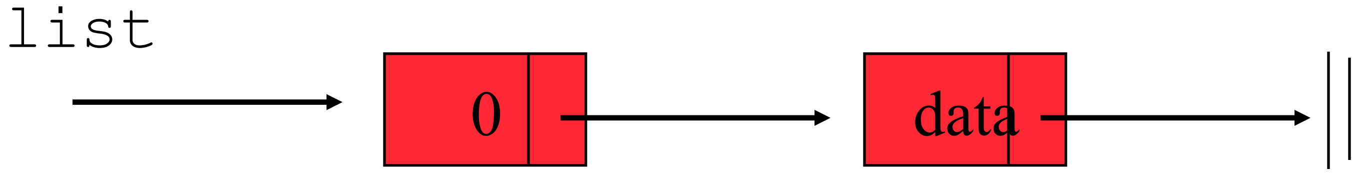
Lists Know Their Size

- You Can Figure Out How Many Nodes Are On The List…
  - No, The Opening Node That `initialize` Put There Doesn't Count As A Node…



Lists Know Their Size

- You Can Figure Out How Many Nodes Are On The List…
  - No, The Opening Node That `initialize` Put There Doesn't Count As A Node…



- `length( list )` Would Return 1 For This List

Erasing Data From The List

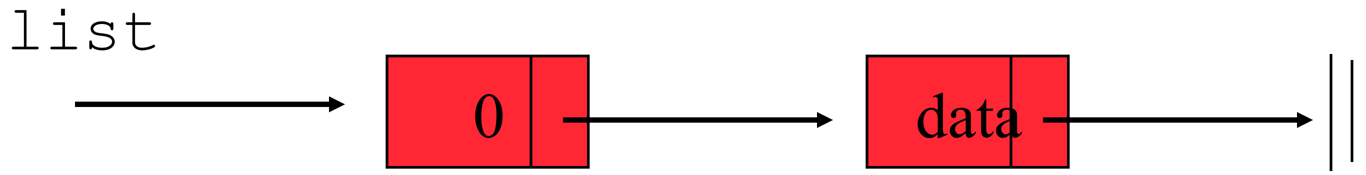
- You Can Try To Remove Whatever You Like
  - If It Is Not On The List, The List Won't Change…

```
erase( data, list );
```

Erasing Data From The List

- You Can Try To Remove Whatever You Like
  - If It Is Not On The List, The List Won't Change…

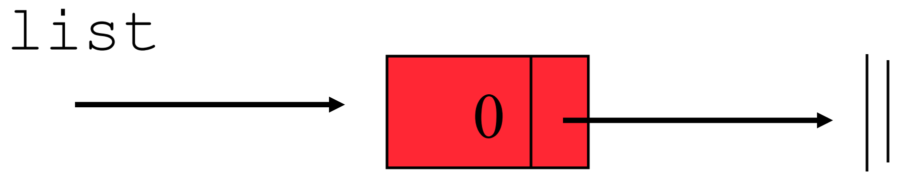
```
erase( data, list );
```



Erasing Data From The List

- You Can Try To Remove Whatever You Like
  - If It Is Not On The List, The List Won't Change…

```
erase( data, list );
```



Printing The List

- You Can See What Is On The List
  - No, The Opening Node Placed There By `initialize` Will Not Be Printed…

```
print( list );
```

Free' ing List

- When You Are All Done With The List, You Must Free The Nodes That Have Been malloc' ed
- Call Release To Perform This Cleanup…

```
 release( &list );
```

Time For Our Next Demo!

- LinkedList.c

Summarizing Our First Demo!

- Pointers Allow For Sophisticated Data Structures
- Linked List Seems Like An Array
  - But You Don't Have To Know How Big It Is Before You Start…
  - It Get Allocated Bit-By-Bit, Rather Than All At Once…
- Linked List Code Is Quite Messy…
- Key Issue: Hide This Complexity From Consumers

Summary

- Linked Lists