What is a Linked List?

2024 Winter APS 105: Computer Fundamentals
Jon Eyolfson

Lecture 29
1.0.0

Arrays Lack Flexibility

What if we used an array to represent a line of people taking turns

Assume we have four people in line, and we assign them each a number

The order of people is the same as the order of the array

- The number at index 0 is at the front of the line

Let’s code putting the person at the front of the line to the back

Moving the First Number to the Back of an Array

```c
#define ARRAY_LENGTH(array) (sizeof((array))/sizeof((array)[0]))

void moveArray(int array[], int arrayLength) {
    int first = array[0];
    for (int i = 0; i < arrayLength - 1; ++i) {
        array[i] = array[i + 1];
    }
    array[arrayLength - 1] = first;
}

int main(void) {
    int array[] = {1, 2, 3, 4};
    int arrayLength = ARRAY_LENGTH(array);
    moveArray(array, arrayLength);
    return EXIT_SUCCESS;
}
```

That’s A Lot of Assignments

Every time we shift by one we need to do $n + 1$ assignments, where n is the array length

Generally, we don’t want the amount of work to scale

You’ll learn to analyze algorithms in the follow-up course

The Order Between the Other Numbers Doesn’t Change

For each number, we can keep track which one is next

We can also keep track of who is at the front of the line

This is also equivalent to having an ordered list

The Data Structure that Implements That Idea is a Linked List

We create a “node” structure in C that contains our data, and which is next

```
typedef struct node node_t;

typedef struct node {
    int val;
    node_t *next;
} node_t;
```

The Data Structure that Implements That Idea is a Linked List

We create a “node” structure in C that contains our data, and which is next

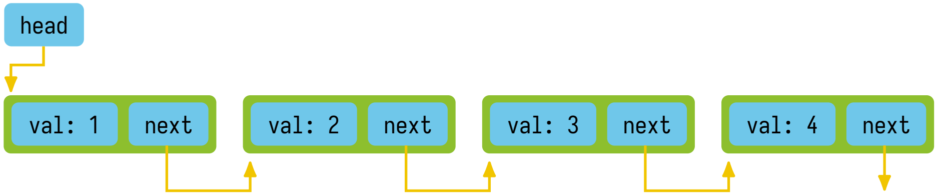
```
typedef struct node node_t;

typedef struct node {
    int val;
    node_t *next;
} node_t;
```

We can’t use `node_t next;` because that would be infinite recursion!

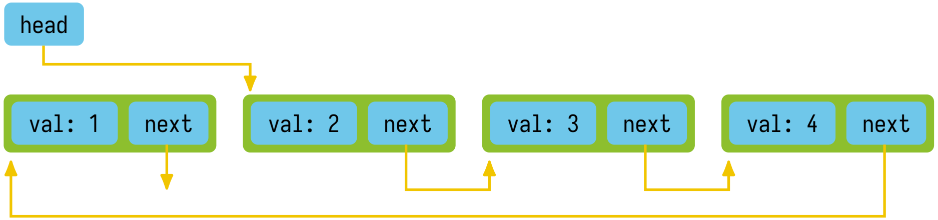
A pointer lets us use `NULL` to represent there is no next node

We track the head (first node) of the list with a `node_t *` variable

We Can Represent Our Numbers as a Linked List



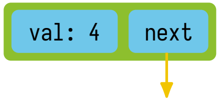
Our Final Goal Changes 3 Pointers



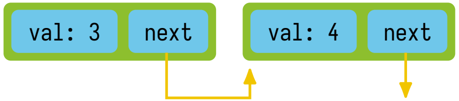
Let’s Create That List in C

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

int main(void) {
    node_t n4 = {4, NULL};
    node_t n3 = {3, &n4};
    node_t n2 = {2, &n3};
    node_t n1 = {1, &n2};
    node_t *head = &n1;
    return EXIT_SUCCESS;
}
```

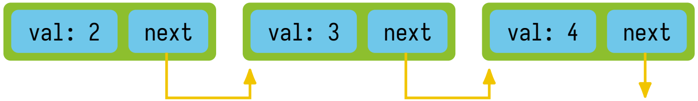
How We Created The List, Step by Step



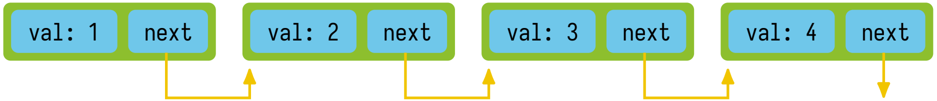
How We Created The List, Step by Step



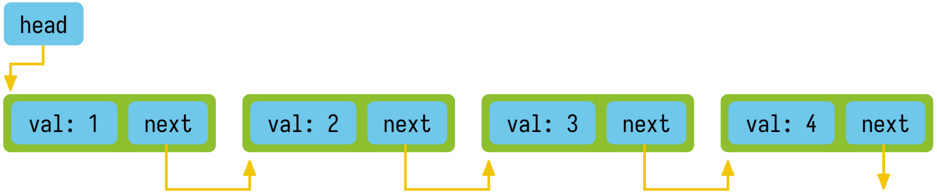
How We Created The List, Step by Step



How We Created The List, Step by Step



How We Created The List, Step by Step



We Can Write a Function to Print the List Values

```
void printList(node_t *head);
```

We could do this either iteratively (with a loop), or recursively

Printing a Linked List Iteratively

```c
void printList(node_t *head) {
    node_t *current = head;
    printf("list:");
    while (current != NULL) {
        printf(" %d", current->val);
        current = current->next;
    }
    printf("\n");
}
```

First, Let’s Create a Node Dynamically

We don’t want to create every node ourselves

More importantly, we don’t want to name them and have memory issues

Code to Create a Node Dynamically

```c
node_t *createNode(int val) {
    node_t *node = malloc(sizeof(node_t));
    if (node == NULL) {
        exit(EXIT_FAILURE);
    }
    node->val = val;
    node->next = NULL;
    return node;
}
```

Now, Create a Function to Insert at the Front of the List

This function should create a new node with the value provided,
make its next pointer point to the current head,
make the new head point to the new node, and
return a pointer to the new node

The function prototype we'll use is:

```
node_t *insertFront(node_t *head, int val);
```

Initial Attempt at Writing `insertFront`

```
node_t *insertFront(node_t *head, int val) {
    node_t *node = createNode(val);
    node->next = head;
    head = node;
    return node;
}
```

Why doesn’t this work?

It Doesn’t Work Because of Call by Value

```
int main(void) {
    node_t *head = NULL;
    insertFront(head, 4);
    printList(head);
    return EXIT_SUCCESS;
}
```

If we want the function to modify `head` then we should use `&head`

It Doesn’t Work Because of Call by Value

```
int main(void) {
    node_t *head = NULL;
    insertFront(head, 4);
    printList(head);
    return EXIT_SUCCESS;
}
```

If we want the function to modify `head` then we should use `&head`

`head` actually represents the linked list, so we should make a `struct` for it

Creating a `struct` for Our Linked List

```c
typedef struct linked_list {
    node_t *head;
} linked_list_t;

linked_list_t *createLinkedList() {
    linked_list_t *linked_list = malloc(sizeof(linked_list_t));
    if (linked_list == NULL) {
        exit(EXIT_FAILURE);
    }
    linked_list->head = NULL;
    return linked_list;
}
```

Now We Can Re-Write `insertFront`

```c
node_t *insertFront(linked_list_t *linked_list, int val) {
    node_t *node = createNode(val);
    node->next = linked_list->head;
    linked_list->head = node;
    return node;
}
```

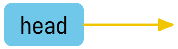
We Need to Re-Write printList Too

```
void printList(linked_list_t *linked_list) {
    node_t *current = linked_list->head;
    printf("list:");
    while (current != NULL) {
        printf(" %d", current->val);
        current = current->next;
    }
    printf("\n");
}
```

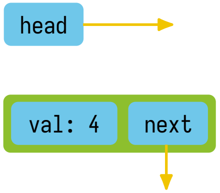
Now, Let’s Use Everything Together

```
int main(void) {
    linked_list_t *linked_list = createLinkedList();
    node_t *n4 = insertFront(linked_list, 4);
    printList(linked_list);
    free(n4);
    free(linked_list);
    return EXIT_SUCCESS;
}
```

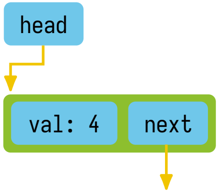
Now, How We Created the List Step by Step



Now, How We Created the List Step by Step



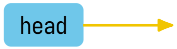
Now, How We Created the List Step by Step



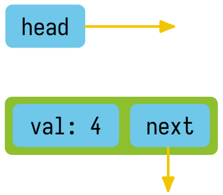
Let’s Check That Adding Another Node Works

```c
int main(void) {
    linked_list_t *linked_list = createLinkedList();
    node_t *n4 = insertFront(linked_list, 4);
    node_t *n3 = insertFront(linked_list, 3);
    printList(linked_list);
    free(n3);
    free(n4);
    free(linked_list);
    return EXIT_SUCCESS;
}
```

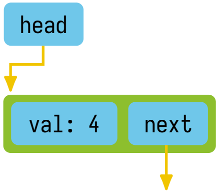
Inserting Two Nodes in the List, Step by Step



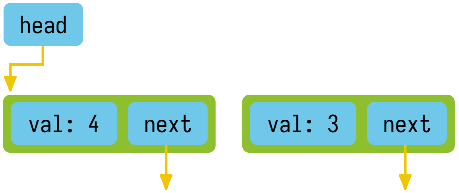
Inserting Two Nodes in the List, Step by Step



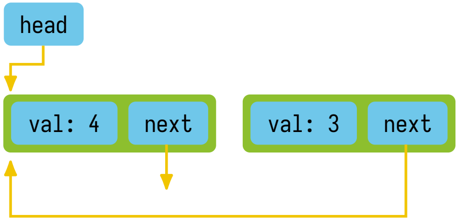
Inserting Two Nodes in the List, Step by Step



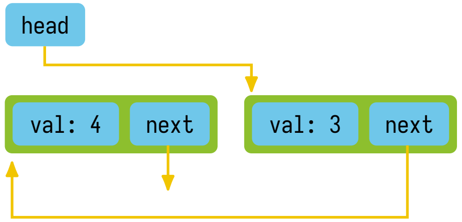
Inserting Two Nodes in the List, Step by Step



Inserting Two Nodes in the List, Step by Step



Inserting Two Nodes in the List, Step by Step



What Happens If We Accidentally Swap a Line of Code

```
node_t *insertFront(linked_list_t *linked_list, int val) {
    node_t *node = createNode(val);
    linked_list->head = node;
    node->next = linked_list->head;
    return node;
}
```

What Would We Expect to See Now with the Lines Swapped?

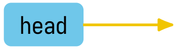
```c
int main(void) {
    linked_list_t *linked_list = createLinkedList();
    node_t *n4 = insertFront(linked_list, 4);
    node_t *n3 = insertFront(linked_list, 3);
    printList(linked_list);
    free(n3);
    free(n4);
    free(linked_list);
    return EXIT_SUCCESS;
}
```

What Would We Expect to See Now with the Lines Swapped?

```
int main(void) {
    linked_list_t *linked_list = createLinkedList();
    node_t *n4 = insertFront(linked_list, 4);
    node_t *n3 = insertFront(linked_list, 3);
    printList(linked_list);
    free(n3);
    free(n4);
    free(linked_list);
    return EXIT_SUCCESS;
}
```

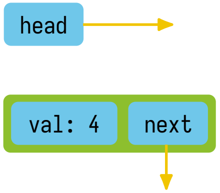
An infinite loop!

Incorrectly Adding Two Nodes into the List



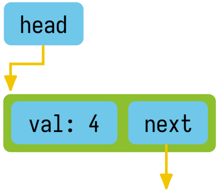
We also can't access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



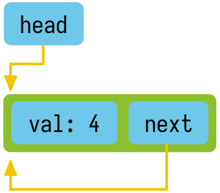
We also can’t access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



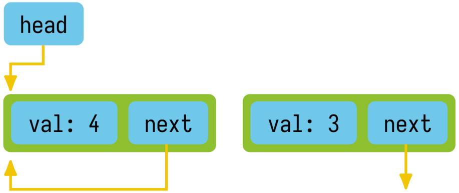
We also can't access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



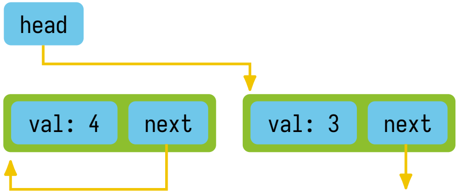
We also can't access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



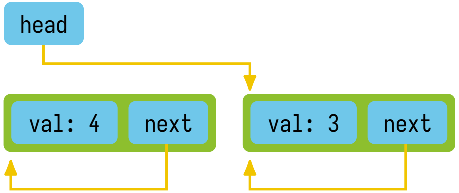
We also can't access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



We also can't access the node with the value 4 anymore!

Incorrectly Adding Two Nodes into the List



We also can't access the node with the value 4 anymore!

A Linked List is Sequence of Nodes

To represent a linked list, we only need to keep track of the first node

- Each node keeps track of the next node in the list

This allows some more flexibility over arrays in certain circumstances

- We can update pointers instead of changing where values are in memory