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

typedef struct User {
  char* username;
  char passwd_sha[32];
} User;

static User users[1000];

void load_users(char* path) {
  FILE* f = fopen(path, "r");
  char buffer[10000];
  int i = 0;
  while(fgets(buffer, 10000, f) != NULL) {
    char* username_part = strtok(buffer, " ");
    char* password_part = strtok(NULL, " ");
```

read line by line

interior ptrs into buffer
internal

```c
    // buffer is reused, so need to make some
    // space for the username for use in the struct
    char* username = malloc(strlen(buffer));
    strcpy(username, username_part);
    printf("%s@%p\n", username, username);
```

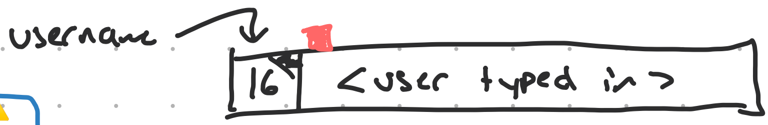
heap-allocated string for username

```c
    // Make a User struct with that username,
    // and then copy the (fixed-length) hash part into it
    User current_user = { username, {} };
    strncpy(current_user.passwd_sha, password_part, 32);
```

copying fixed 32-byte pw_hash

```c
    users[i] = current_user;
    i += 1;

    // done with username, so free it now ⚠
    free(username);
  }
}
```

not done with the data at 🟥 0x...31b10



users[0] = { 🟥, "abcdef..." }
users[1] = { 🟥, "123456..." }
users[2] = { 🟥, "9876...." }

```c
int main() {
  load_users("users.txt");
  char* username = malloc(7);
  printf("Enter your username: ");
  fgets(username, 6, stdin);
  username[strcspn(username, "\n")] = '\0';
  for(int i = 0; i < 1000; i += 1) {
    char* username = users[i].username;
    if(username == NULL) { break; }
    printf("%s@%p: %.32s\n", username, username, users[i].passwd_sha);
  }
}
```

populate global users array from file

Imagine this is a text box on the web

used after we free'd in the loop above

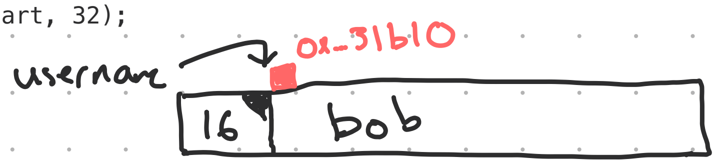
```
> ./login
jpolitz@0x102e31b10
gsoosairaj@0x102e31b10
aschulman@0x102e31b10
Enter your username: bob
bob@0x102e31b10: abcdef1234567890abcdef1234567890
bob@0x102e31b10: 1234567890abcdef1234567890abcdef
bob@0x102e31b10: 9876543210abcdef9876543210abcdef
```

```
jpolitz abcdef1234567890abcdef1234567890
gsoosairaj 1234567890abcdef1234567890abcdef
aschulman 9876543210abcdef9876543210abcdef
```

users.txt

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

typedef struct User {
  char* username;
  char passwd_sha[32];
} User;

static User users[1000];

void load_users(char* path) {
  FILE* f = fopen(path, "r");
  char buffer[10000];
  int i = 0;
  while(fgets(buffer, 10000, f) != NULL) {
    char* username_part = strtok(buffer, " ");
    char* password_part = strtok(NULL, " ");

    // buffer is reused, so need to make some
    // space for the username for use in the struct
    char* username = malloc(strlen(buffer));
    strcpy(username, username_part);
    printf("%s@%p\n", username, username);

    // Make a User struct with that username,
    // and then copy the (fixed-length) hash part into it
    User current_user = { username, {} };
    strncpy(current_user.passwd_sha, password_part, 32);

    users[i] = current_user;
    i += 1;

    // done with username, so free it now ⚠️
    free(username);
  }
}

int main() {
  load_users("users.txt");
  char* username = malloc(7);
  printf("Enter your username: ");
  fgets(username, 6, stdin);
  username[strcspn(username, "\n")] = '\0';
  for(int i = 0; i < 1000; i += 1) {
    char* username = users[i].username;
    if(username == NULL) { break; }
    printf("%s@%p: %.32s\n", username, username, users[i].passwd_sha);
  }
}
```

don't store username_part directly, stack-allocated



users[0] = { 🟥, "abc" }
users[1] = { 🟥, "123…" }
users[2] = { 🟥, "987…" }

populates global users array from file

login

use after free

imagine this is a login textbox on a webpage!

```
> gcc login.c -o login
> ./login
jpolitz@0x102e31b10
gsoosairaj@0x102e31b10
aschulman@0x102e31b10
Enter your username: bob
bob@0x102e31b10: abcdef1234567890abcdef1234567890
bob@0x102e31b10: 1234567890abcdef1234567890abcdef
bob@0x102e31b10: 9876543210abcdef9876543210abcdef
```

users.txt

```
jpolitz abcdef1234567890abcdef1234567890
gsoosairaj 1234567890abcdef1234567890abcdef
aschulman 9876543210abcdef9876543210abcdef
```

/etc/shadow

```
struct Node {
    char* val;
    Node* next;
}
```

```
Node* mk_node(char* val,
              Node* next) {
  Node* n = malloc(sizeof(Node));
  n->val = val;
  n->next = next;
  return n;
}
```

Python

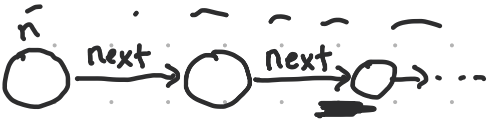
```
class Node
    def _init_(self, val, next):
        ....

n = Node("abc", None)
```

Java

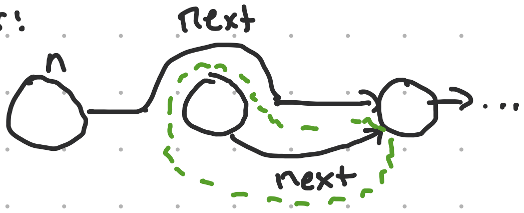
```
class Node {
    public Node (String val, Node next) {
        ...
    }
}

Node n = new Node("abc", null)
```

```
void remove_next (Node* n) {
  Node* to_remove = n->next;
  n->next = n->next -> next;
  free(to_remove);

}
```

pointer update



after:



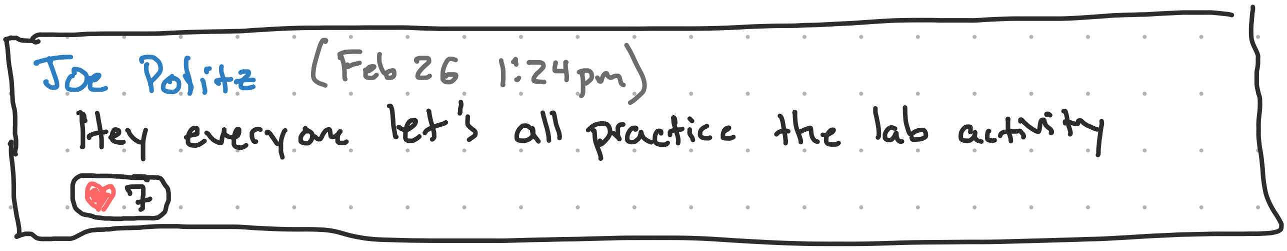
no incoming refrences to the middle node

java - garbage collection

python - reference counting

rust - "ownership"
"lifetimes"

RQ1: Design a struct to represent a message/chat in a chat room:


Joe Politz (Feb 26 1:24pm)

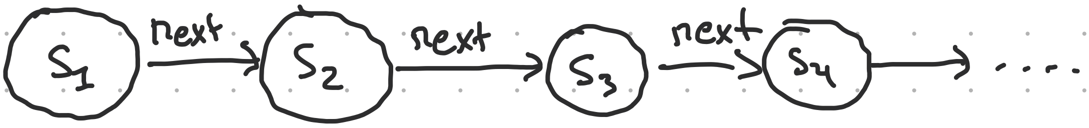
Hey everyone let's all practice the lab activity

❤ 7

```
struct Chat {                    8
    char* name;          16                   16
    char date[16];   char time[16];
    char* message_content; 8
    int id; 4
    Reaction reactions[1024];
}
```

```
struct Reaction {    4
    char symbol[4];
    int count; 4
}
```

RQ2: Design a struct to represent a node in a linked list with string values



```
struct Node {
    char* val;
    Node next; X
}
```

sizeof(char*)
+
sizeof(Node)

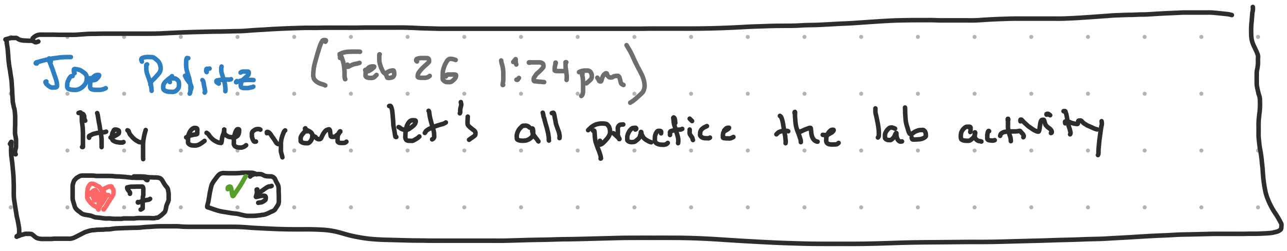
```
struct Node {
    char* val;   ✓
    Node* next;
}
```

sizeof(Node) = 16

RQ3: What is sizeof for each struct you designed?

sizeof(Chat) = 52 + (1024 * 8) ← reactions[1024]

RQ1: Design a struct to represent a message/chat in a chat room:


Joe Politz (Feb 26 1:24pm)

Hey everyone let's all practice the lab activity

♥ 7 ✓ 5

```
struct Chat {
  uint64t timestamp; 8
  char*   name; 8
  char*   message; 8
  ~~int hearts;~~   Reaction* reactions; 8
                    int react_count; 4
}
```

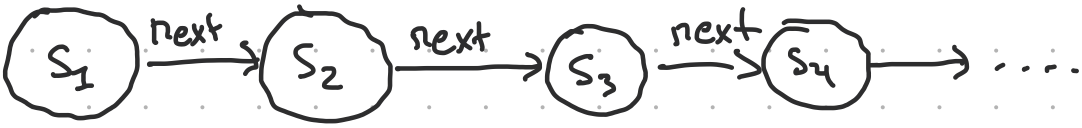
↑ like 2 different reactions (♥, ✓)

```
struct Reaction {
    char emoji[4];
    int times_reacted;
}
```

↑ Like the 7 or 5 above

sizeof(Chat) = 36 (40 w/padding?)

RQ2: Design a struct to represent a node in a linked list with string values



```
struct Node {
    char* val;
    Node next; X
}
```

sizeof(char*) + sizeof(Node)

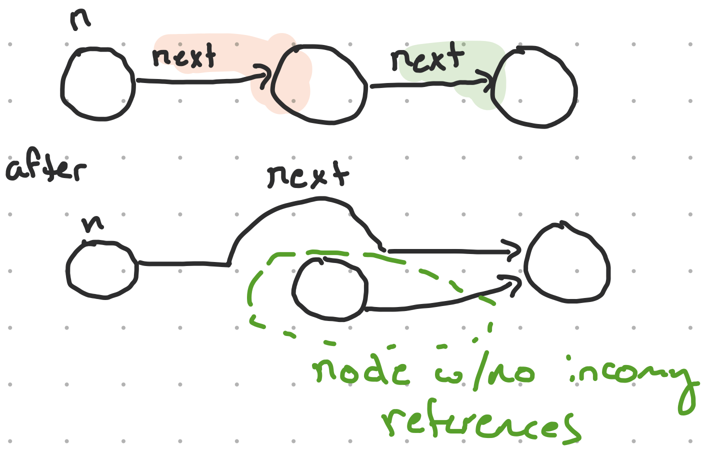
```
struct Node {
    char* val;
    Node* next;
}
```

sizeof(Node) = 16

RQ3: What is sizeof for each struct you designed?

```
struct Node {
    char* val;
    Node* next;
}


Node* mk_node(char* val, Node* next) {
    Node* n = malloc( sizeof(Node) );
    n->val = val;
    n->next = next;
    return n;
}


void remove_next(Node* n) {
    Node* next_n = n->next;

    n->next = next_n->next;   pointer
                              update
    free(next_n);
}
```

```
class Node:
    def _init_(self, val,
               next):
        self.val = val
        self.next = next


class Node {
    String val; Node next;
    public Node(String val,
                Node next) {
        this.val = val;
        this.next = next;
    }
}
```



n
next
next
after
n
next
node w/no incoming
references