DATE

09/11/2024

CS 350 DISCUSSION 1

Intro to C concepts and help with HW1

HW 1 CONCEPTS

- Sockets
- Command line arguments
- C Pointers
- Memory Management

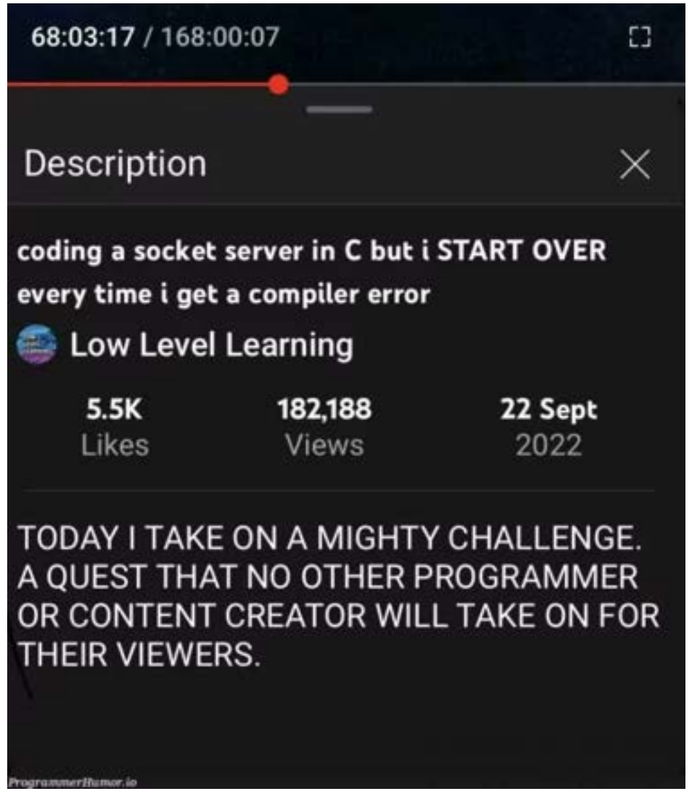
01

SOCKETS

A socket is one endpoint of a two-way communication link between two programs running on the network and it's usually bound to a specific port.

You (students) will be exploring that in your homeworks when running the server and client together!

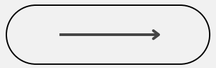
The server creates a socket to listen for incoming connections from clients. Once a client connects, the server creates a separate socket specifically for that client connection, allowing communication.
The client creates its own socket to connect to the server. This socket is used to send data to and receive data from the server.



01

SOCKETS

An example of how we will be using sockets in homework

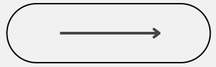


```c
/* Main function to handle connection with the client. This function
 * takes in input conn_socket and returns only when the connection
 * with the client is interrupted. */
static void handle_connection(int conn_socket)
{
        /* IMPLEMENT ME! */
}

/* Template implementation of the main function for the FIFO
 * server. The server must accept in input a command line parameter
 * with the <port number> to bind the server to. */
int main (int argc, char ** argv) {
        int sockfd, retval, accepted, optval;
        in_port_t socket_port;
        struct sockaddr_in addr, client;
        struct in_addr any_address;
        socklen_t client_len;

        /* Get port to bind our socket to */
        if (argc > 1) {
                socket_port = strtol(argv[1], NULL, 10);
                printf("INFO: setting server port as: %d\n", socket_port);
        } else {
                ERROR_INFO();
                fprintf(stderr, USAGE_STRING, argv[0]);
                return EXIT_FAILURE;
        }

        /* Now onward to create the right type of socket */
        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        if (sockfd < 0) {
                ERROR_INFO();
                perror("Unable to create socket");
                return EXIT_FAILURE;
        }

        /* Before moving forward, set socket to reuse address */
        optval = 1;
        setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (void *)&optval, sizeof(optval));
```

01

SOCKETS

Part (2) of the example



```
        /* Convert INADDR_ANY into network byte order */
        any_address.s_addr = htonl(INADDR_ANY);

        /* Time to bind the socket to the right port  */
        addr.sin_family = AF_INET;
        addr.sin_port = htons(socket_port);
        addr.sin_addr = any_address;

        /* Attempt to bind the socket with the given parameters */
        retval = bind(sockfd, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));

        if (retval < 0) {
                ERROR_INFO();
                perror("Unable to bind socket");
                return EXIT_FAILURE;
        }

        /* Let us now proceed to set the server to listen on the selected port */
        retval = listen(sockfd, BACKLOG_COUNT);

        if (retval < 0) {
                ERROR_INFO();
                perror("Unable to listen on socket");
                return EXIT_FAILURE;
        }

        /* Ready to accept connections! */
        printf("INFO: Waiting for incoming connection...\n");
        client_len = sizeof(struct sockaddr_in);
        accepted = accept(sockfd, (struct sockaddr *)&client, &client_len);

        if (accepted == -1) {
                ERROR_INFO();
                perror("Unable to accept connections");
                return EXIT_FAILURE;
        }

        /* Ready to handle the new connection with the client. */
        handle_connection(accepted);

        close(sockfd);
        return EXIT_SUCCESS;

}
```

02

COMMAND LINE ARGUMENTS

Command line arguments are how you'll be running and commanding your server to do certain things for testing

This can be:

- Defining queue size
- Defining the type of queue
- Number of threads/workers involved in the server and MORE!

- Here's how you will be running it soon!

02

HOW DO YOU CODE IT?

Up to you!

recommendation (Hint):

- parse through using getopt function

Its up to you to implement it but we'll be here to help if you run into issues.

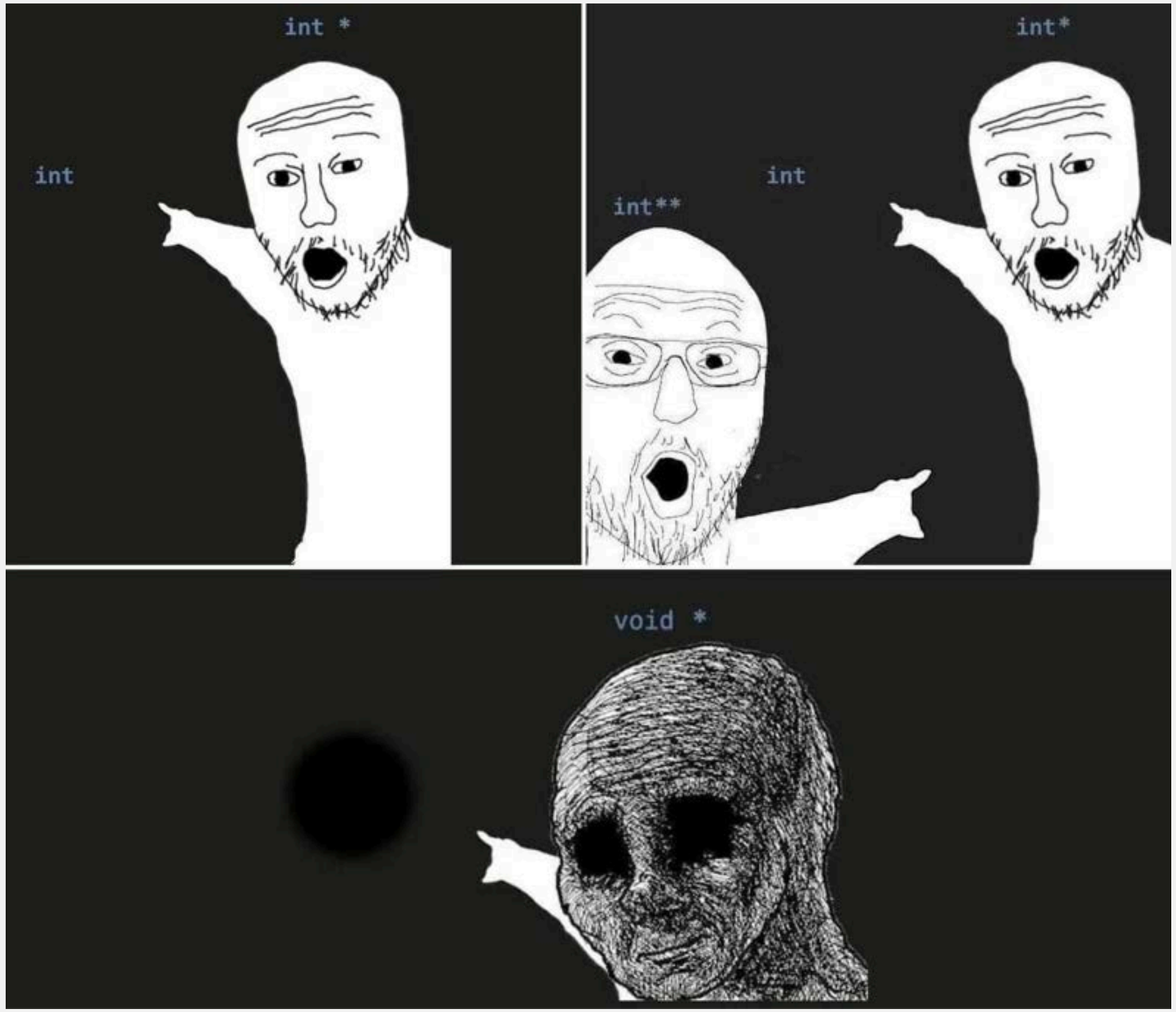
03

C POINTERS AND OTHER CONCEPTS

The most important part of this discussion so do ask question if you feel lost.

We're going to cover the following:

- Defining C pointers
- Pointer Variables vs Dereferencing
- arrows vs dots

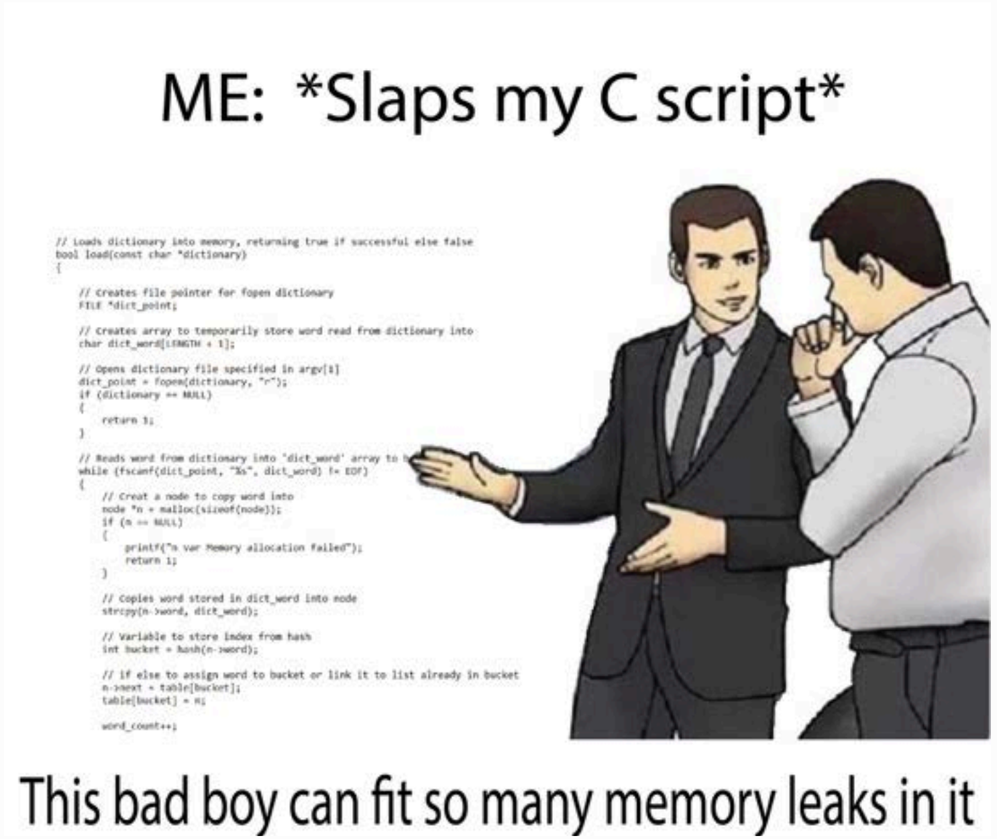


Things that might help:

&: Address of

int* ptr: An integer pointer variable

***ptr:** Value that variable[ptr] is pointing to (Dereferencing)



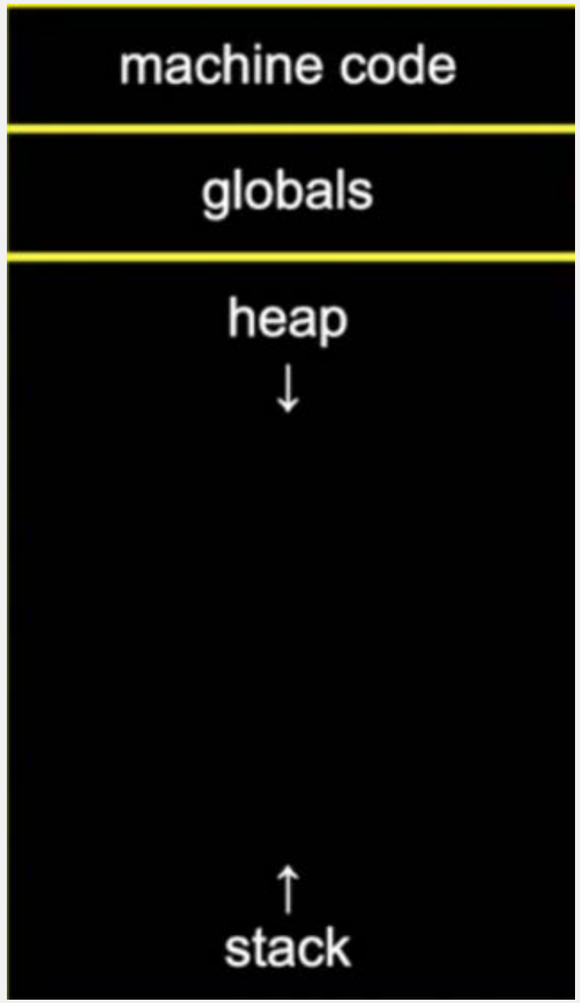
EXAMPLE

C POINTERS VS DEREFERENCING

04

MEMORY & MANAGEMENT

Memory management is extremely important in C because C gives YOU, the programmer, direct control over system memory but it does not provide automatic memory management features like garbage collection, which exist in higher-level languages (e.g., Python or Java).



04

STATIC & DYNAMIC MEMORY ALLOCATION

what is malloc?

It's a built-in function in the stdlib.h header file which dynamically allocates a large block of memory based on the size provided

syntax

```
(void*) malloc(size_t size);
```

important:

When sucessfully allocated memory, malloc returns a pointer pointing to the first byte of memory, else returns NULL

WHY VOID*?

Because Malloc has NO IDEA what it's pointing to. It doesn't care. All it cares about is creating the space in memory!

Define malloc's type!

You can type cast and decide what data type of memory you want to store

Example: int *ptr = (int*) malloc(4);

04

FREEING MEMORY

Freeing memory in C is crucial because C does not have automatic garbage collection like higher-level languages (e.g., Python, Java)
BUT MOST IMPORTANTLY, MEMORY LEAKS!

thank you!

Extra resources if needed:

Link_1
(Static_vs_dynamic_memory)

https://www.geeksforgeeks.org/difference-between-static-and-dynamic-memory-allocation-in-c/