Week 4 Prac

Question 1

1. A data structure is a way of organizing data so it can be accessed or stored effectively such as an array or binary tree.
2. Linear data structures are linked to each other in a chain in one or two directions, whereas non-linear data structures will have all kinds of relationships between elements in the data structure.
3. Stacks:

push

b -> c -> d -> e -> NULL

a b -> c -> d -> e -> NULL

a -> b -> c -> d -> e -> NULL

pop

a -> b -> c -> d -> e -> NULL

a b -> c -> d -> e -> NULL

b -> c -> d -> e -> NULL

1. Stacks are a type of linked list that is only interacted with from the head of the list.

Linked lists are lists of elements that are organized by their relationships with each other and linked by pointers.

1. A stack can be used to store operations that have higher priority the later they are created.

A stack can be used to store data without having to store much information about the location of the data, as the head of the stack is all that is needed to address it.

Question 2

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Sentence {

char word[20];

struct Sentence \* next;

};

typedef struct Sentence sentence;

typedef sentence \* sent\_ptr;

sent\_ptr head = NULL;

int main(void) {

// Push a new word to the top of the stack.

sent\_ptr pushWord(void) {

sent\_ptr newWord = (sentence \*) malloc(sizeof(sentence));

newWord->next = head;

head = newWord;

return head;

}

// Pop a word from the top of the stack.

sent\_ptr popWord(sent\_ptr popped) {

head = head->next;

return popped;

}

// Take user input.

printf("Please enter a sentence no longer than 100 characters.\n");

char input[100];

fgets(input, 99, stdin);

//printf(input);

printf("\n");

// There is a bug where the program will consider leading whitespace to be a

// word instead of ignoring it and starting at the first valid character but

// I can't be bothered fixing it.

// Cut the words up into individual strings and place them into the stack.

size\_t cache\_pos = 0;

char cache[100] = "";

for(int i = 0; i < strlen(input); i++) {

if(input[i] != ' ' && input[i] != '\0' && input[i] != '\n') {

// Valid character that is part of the word.

cache[cache\_pos] = input[i];

cache\_pos++;

} else {

// End of the word.

printf("%s\n", cache);

// Push the word to the top of the stack.

strcpy((pushWord())->word, cache);

// Reset the cache.

cache\_pos = 0;

for(int a = 0; a < 100; a++) {

cache[a] = '\0';

}

// Skip the rest of the whitespace if it's not a single character.

while(input[i + 1] == ' ') {i++;}

}

}

// Separate the two stacks.

printf("\n");

// Print and free the stack.

sent\_ptr temp;

while(head != NULL) {

temp = popWord(head);

printf("%s\n", temp->word);

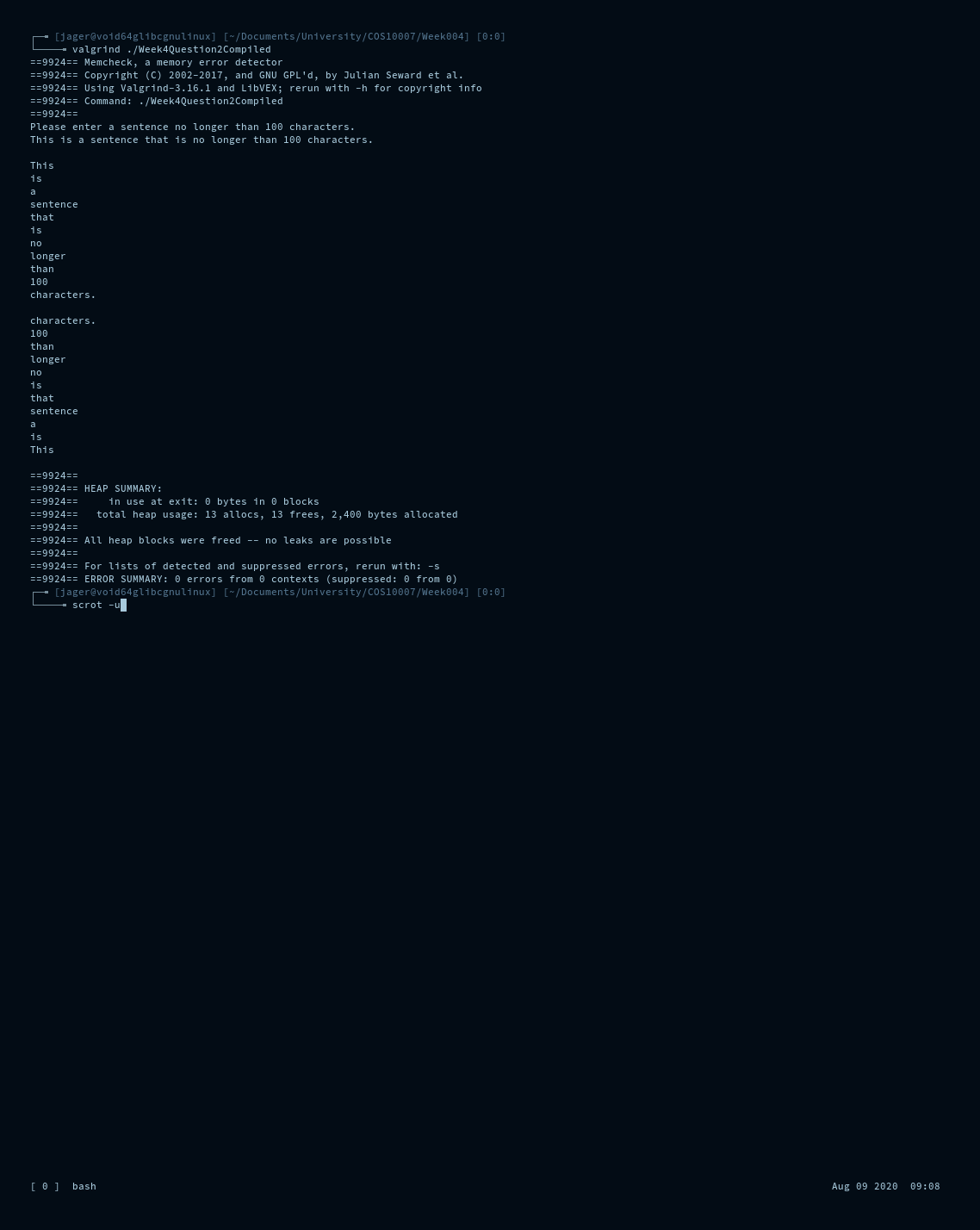
free(temp);

}

printf("\n");

return 0;

}



Question 3

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <stdbool.h>

struct CharChain {

char stored\_character;

struct CharChain \* next;

};

typedef struct CharChain cc;

typedef cc \* cc\_ptr;

cc\_ptr head0 = NULL;

cc\_ptr head1 = NULL;

cc\_ptr push(cc\_ptr \* h, char c);

cc\_ptr pop(cc\_ptr \* h);

void nukelist(cc\_ptr \* h);

int main(void) {

// Take user input.

printf( "Palindrome tester\n"

"Type a string to be tested.\n");

char input[100];

fgets(input, 99, stdin);

printf("\n");

// Push string to stack removing non alnum.

for(int i = 0; i < strlen(input); i++) {

if(isalnum(input[i]) != 0) {

push(&head0, tolower(input[i]));

// Printing the original string here so that I don't have to create

// a loop just to print it from.

printf("%c", tolower(input[i]));

}

}

printf("\n");

// Copy contents of stack to new stack in reverse order.

cc\_ptr temp = head0;

while(temp != NULL) {

push(&head1, temp->stored\_character);

temp = temp->next;

}

// Compare contents of stacks.

bool same = true;

while(head0 != NULL) {

// This is where the reversed loop is printed.

printf("%c", head0->stored\_character);

if(head0->stored\_character != head1->stored\_character) {

same = false;

}

free(pop(&head0));

free(pop(&head1));

}

printf("\n\n");

// Tell the user if the input was a palindrome.

if(same == true) {

printf("This is a palindrome.\n");

} else {

printf("This is not a palindrome.\n");

}

// Cleanup. (Just in case there's something left of these lists somehow.)

nukelist(&head0);

nukelist(&head1);

return 0;

}

cc\_ptr push(cc\_ptr \* h, char c) {

cc\_ptr newcc = (cc\*) malloc(sizeof(cc));

newcc->stored\_character = c;

newcc->next = \*h;

\*h = newcc;

return newcc;

}

cc\_ptr pop(cc\_ptr \* h) {

cc\_ptr p = \*h;

\*h = (\*h)->next;

return p;

}

void nukelist(cc\_ptr \* h) {

while(\*h != NULL) {

cc\_ptr temp = \*h;

\*h = (\*h)->next;

free(temp);

}

}

