Week 3 Question 1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

int main(void) {

/\* Wait 1 second to make sure that the time isn't the same when we run this

\* program again. \*/

sleep(1);

/\* Setup random number generation. \*/

srandom(time(NULL));

/\* Singly linked list. \*/

struct llstruct {

int integer;

struct llstruct \*next;

};

typedef struct llstruct ll;

typedef ll \*ll\_ptr;

/\* Pointers to locations in the list. \*/

ll\_ptr index = NULL;

ll\_ptr curr = NULL;

/\* Fill out contents of list with random values. \*/

for(int i = 0; i < 10; i++) {

curr = (ll\*) malloc(sizeof(ll));

curr->integer = random() % 51;

curr->next = index;

index = curr;

}

/\* Print the list. \*/

while(curr != NULL) {

printf("%d ", curr->integer);

curr = curr->next;

}

/\* Put a newline between and reset the value of curr. \*/

printf("\n");

curr = index;

/\* Calculate the average. \*/

float average = 0.00;

while(curr != NULL) {

average += (float) curr->integer;

curr = curr->next;

}

average /= 10.00;

printf("Average: %.2f\n", average);

/\* Free memory. \*/

while(index != NULL) {

curr = index->next;

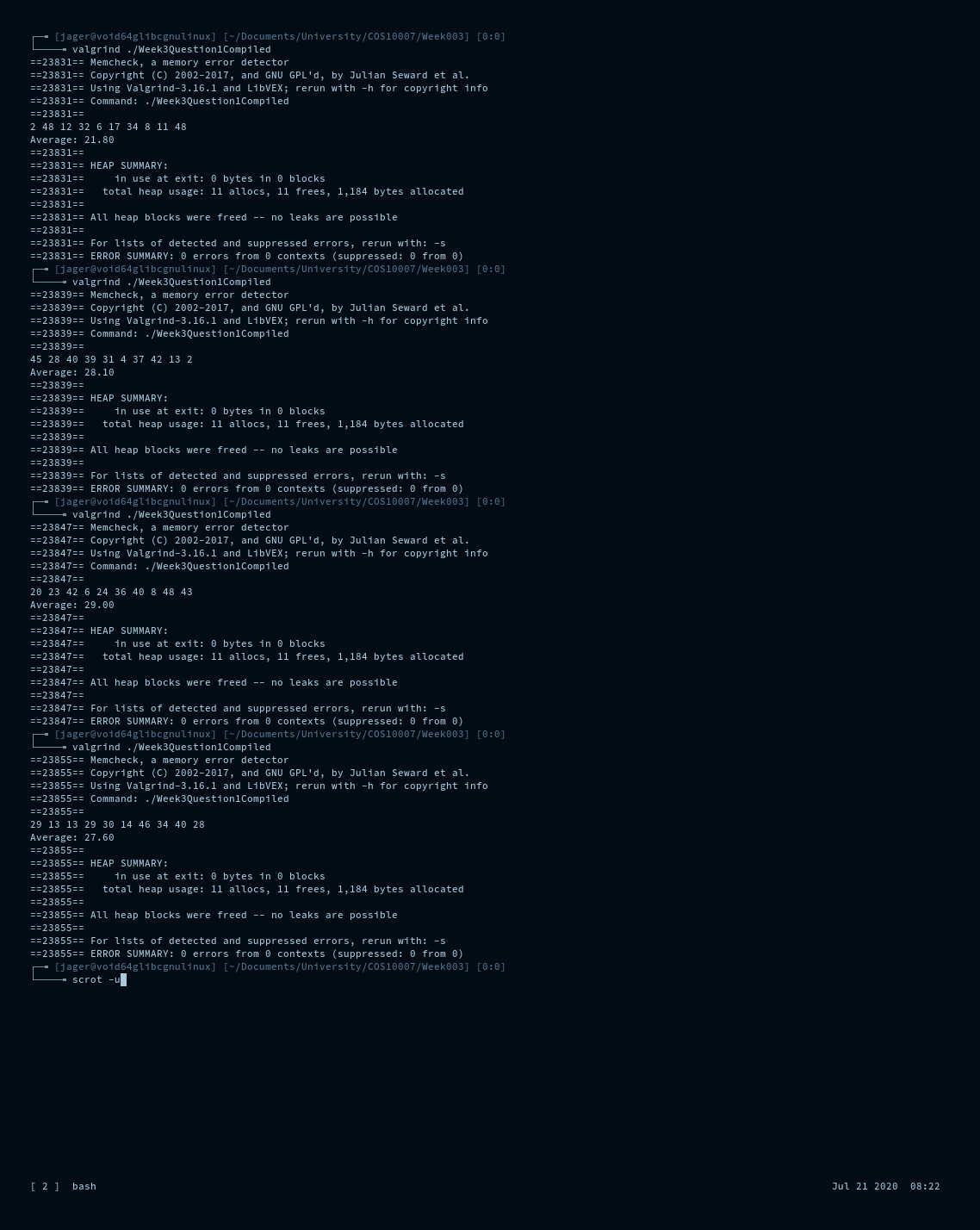
free(index);

index = curr;

}

return 0;

}



Week 3 Question 2

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <stdbool.h>

int main(void) {

// Wait 1 second to make sure that the time isn't the same when we run this

// program again.

sleep(1);

// Setup random number generation.

srandom(time(NULL));

// Singly linked list.

struct student {

char name[20];

int id;

struct student \*next;

};

typedef struct student Student;

typedef Student \*Student\_ptr;

// Pointers to locations in the list.

Student\_ptr index = NULL;

Student\_ptr curr = NULL;

Student\_ptr new = NULL;

// Check if a number corresponds to a student.

// Returns true if student number exists.

bool checkid(unsigned int checkme) {

bool found = false;

Student\_ptr student\_to\_check = index;

while(student\_to\_check != NULL) {

if(checkme == student\_to\_check->id) {found = true;}

student\_to\_check = student\_to\_check->next;

}

return found;

}

bool checkname(char\* checkme) {

bool found = false;

Student\_ptr student\_to\_check = index;

while(student\_to\_check != NULL) {

if(strcmp(checkme, student\_to\_check->name)) {found = true;}

student\_to\_check = student\_to\_check->next;

}

return found;

}

// Place "new" into the correct location in the list.

void insert\_into\_list(void) {

// Insert new student in linked list.

curr = index;

if(index != NULL) {

while(curr != NULL) {

if(curr->id > new->id) {

// Make new the index and link to the old index,

// this should only happen when the index is greater than

// the new value we create.

new->next = curr;

index = new;

curr = NULL;

} else if( curr->next == NULL ||

curr->id < new->id &&

curr->next->id > new->id) {

// Place after curr.

new->next = curr->next;

curr->next = new;

curr = NULL;

} else {

// Move to the next position in the list.

curr = curr->next;

}

}

} else {

// If there isn't an index yet then new becomes the index.

new->next = NULL;

index = new;

}

}

// Create a new student and place them in order of student id.

void create\_student(void) {

int tmpid;

// !!! BUG !!!

// This code will get stuck in a loop if every possible value is used.

// Generate a new student id that is unique.

do {

tmpid = (random() % 100) + 1;

} while(checkid((unsigned int) tmpid));

new = (Student\*) malloc(sizeof(Student));

new->id = tmpid;

// Take user input for name of student.

printf("What is the student's name. (ID: %d)\n", new->id);

char tmpname[20];

fgets(tmpname, 19, stdin);

// Get rid of newline.

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

if(tmpname[i] == '\n') {tmpname[i] = '\0';}

}

strcpy(new->name, tmpname);

insert\_into\_list();

}

// Create students.

for(int i = 0; i < 10; i++) {

create\_student();

}

printf("\n");

// Print the list to screen.

void print\_student\_list(void) {

printf("Student list\n");

curr = index;

while(curr != NULL) {

printf("ID: %d\tName: %s\n", curr->id, curr->name);

curr = curr->next;

}

printf("\n");

}

print\_student\_list();

// Print the list to the screen in reverse.

printf("Student list (reverse order)\n");

curr = index;

// Count how many entries there are.

int i = 0;

while(curr != NULL) {

i++;

curr = curr->next;

}

// Walk backwards through the linked list.

for(i; i > 0; i--) {

curr = index;

for(int a = 1; a < i; a++) {

curr = curr->next;

}

printf("ID: %d\tName: %s\n", curr->id, curr->name);

}

printf("\n");

// Write the list to file.

FILE \*fptr = fopen("./iofile.txt", "w+");

curr = index;

while(curr != NULL) {

fprintf(fptr, "ID: %d\tName: %s\n", curr->id, curr->name);

curr = curr->next;

}

fclose(fptr);

// Setup for user removing and editing entries.

char userinputchar[20];

int userinputint;

void take\_user\_input(void) {

fgets(userinputchar, 19, stdin);

// Filter input.

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

if(userinputchar[i] == '\n') {userinputchar[i] = '\0';}

}

userinputint = atoi(userinputchar);

}

// Remove an entry.

printf("Input a student name or number to delete: ");

take\_user\_input();

// Check if it exists.

if(checkid(userinputint) || checkname(userinputchar)) {

// Find it.

Student\_ptr tmp;

tmp = (Student\*) malloc(sizeof(Student));

tmp->next = index;

curr = tmp;

while( strcmp(curr->next->name, userinputchar) != 0 &&

curr->next->id != userinputint) {

curr = curr->next;

}

// Handle removing and rearranging the list.

if(curr->next == index) {

// The index is the removed value.

index = curr->next->next;

free(curr->next);

} else {

// The removed value is not the index.

free(tmp);

tmp = curr->next;

curr->next = curr->next->next;

}

free(tmp);

}

printf("\n");

print\_student\_list();

// Edit a student's details.

printf("Input a student name or number to edit: ");

take\_user\_input();

// Check if it exists.

if(checkid(userinputint) || checkname(userinputchar)) {

Student\_ptr tmp;

tmp = (Student\*) malloc(sizeof(Student));

tmp->next = index;

curr = tmp;

while( strcmp(curr->next->name, userinputchar) != 0 &&

curr->next->id != userinputint) {

curr = curr->next;

}

// Edit the student ID.

printf("Enter a new ID for %s: ", curr->next->name);

take\_user\_input();

// Remove the student from the list and mark it as "new" for

// re-insertion.

if(curr->next == index) {

// The edited value is the index.

new = curr->next;

index = curr->next->next;

} else {

// The edited value is not the index.

new = curr->next;

curr->next = curr->next->next;

}

// We don't need tmp any more.

free(tmp);

// Check if the new number is already in the list.

if(checkid(userinputint)) {

printf("Error: That student ID is not unique.\n");

} else {

new->id = userinputint;

}

// Re-insert new into the list.

insert\_into\_list();

}

printf("\n");

print\_student\_list();

// Free memory.

while(index != NULL) {

curr = index->next;

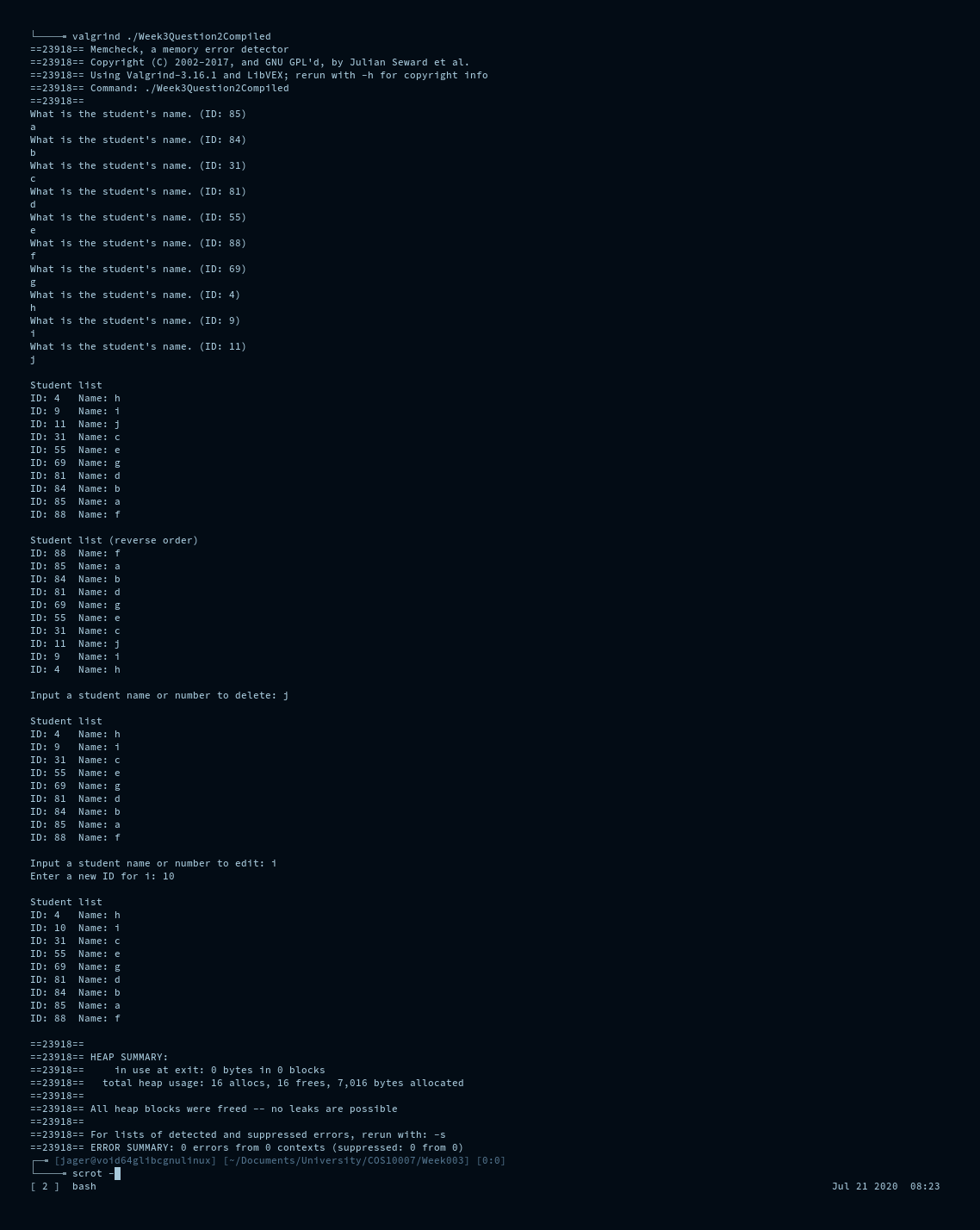
free(index);

index = curr;

}

return 0;

}



(File is written to before user modifications to list)

iofile.txt:

ID: 4 Name: h

ID: 9 Name: i

ID: 11 Name: j

ID: 31 Name: c

ID: 55 Name: e

ID: 69 Name: g

ID: 81 Name: d

ID: 84 Name: b

ID: 85 Name: a

ID: 88 Name: f