COS10007 Assignment 1

Section A Part 1

// size\_t

#include <stdlib.h>

// sleep

#include <unistd.h>

// text interaction

#include <stdio.h>

// bool

#include <stdbool.h>

// strcpy

#include <string.h>

// toupper

#include <ctype.h>

// isnan

#include <math.h>

int main(int argc, char \*argv[]) {

// Handle arguments.

// Check ammount of arguments.

if(argc != 2) {

// Incorrect number of arguments,

// display a terse message then kill the program.

printf("One argument accepted, no more, no less.\n");

return 1;

} else {

FILE \*fptr = NULL;

if((fptr = fopen(argv[1], "r+")) == NULL) {

// File didn't open, probably doesn't exist.

printf("File failed to open, does it exist?\n");

return 1;

} else {

// Don't keep the file open unless using it.

fclose(fptr);

}

}

// This will always open the one argument to the program.

FILE \*open\_file(const char \*mode) {

FILE \*fptr = NULL;

if((fptr = fopen(argv[1], mode)) == NULL) {

// The file didn't open.

printf("File failed to open.\n");

// Actual error handling goes here somewhere.

}

return fptr;

}

// Student details.

struct person\_tag {

char name[20];

char id[10];

};

// Course details.

struct course\_tag {

char course\_name[20];

int no\_of\_units;

int marks[4];

float avg;

};

// Linked list for storing student details alongside course details.

struct student\_tag {

struct person\_tag student\_info;

struct course\_tag course\_info;

struct student\_tag \*next;

};

// typedef laziness

typedef struct student\_info Student\_Info;

typedef struct course\_info Course\_Info;

// malloc Person for a list element.

typedef struct student\_tag Student\_Tag;

typedef Student\_Tag \*Person;

// Main data holding variables.

Person Students\_Details = NULL;

FILE \*Students\_Details\_FILE = NULL;

// Function to free memory allocated to Students\_Details.

void clean\_up\_Students\_Details(void) {

while(Students\_Details != NULL) {

Person cleanup\_pointer = Students\_Details;

Students\_Details = Students\_Details->next;

free(cleanup\_pointer);

}

}

// Take user input and return a number.

unsigned int user\_input\_number(void) {

unsigned int answer;

char text\_answer[100];

fflush(stdin);

fgets(text\_answer, 99, stdin);

answer = atoi(text\_answer);

return answer;

}

// Place Person into the linked list in alphabetical order of name.

void place\_into\_list(Person addme) {

// START DEBUG CODE

/\* Prepend to list. \*

addme->next = Students\_Details;

Students\_Details = addme;

/\* END DEBUG CODE \*/

Person pos = Students\_Details;

// I don't want to bloat my if condition so I'm making it

// a function.

bool compare\_next\_string(void) {

if(strcmp( addme->student\_info.name,

pos->next->student\_info.name) >= 0) {

// addme is equal to or greater than the ascii value

// of pos->next (later in alphabet)

return true;

} else {

// addme is less than the ascii value of pos->next

// (earlier in the alphabet)

return false;

}

}

bool compare\_string(void) {

if(strcmp( addme->student\_info.name,

pos->student\_info.name) >= 0) {

// addme is equal or greater than the ascii value of

// pos (later in alphabet)

return true;

} else {

// addme is less than the ascii value of pos

// (earlier in the alphabet)

return false;

}

}

// Place item into list in alphabetical order of name.

// pos is a pointer off of the end of the index pointer for

// this ll.

// pos -> Students\_Details (list index) -> node -> node -> NULL

/\*\*/

if(Students\_Details == NULL) {

// There is nothing in the list yet, start a new one.

Students\_Details = addme;

} else if(!compare\_string()) {

// addme is smaller than the index.

addme->next = Students\_Details;

Students\_Details = addme;

} else {

// There are already things in the list.

while(pos != NULL) {

// Compare the strings.

if( pos->next == NULL ||

compare\_string() &&

!compare\_next\_string()) {

// addme is smaller than pos->next

addme->next = pos->next;

pos->next = addme;

pos = NULL;

} else {

// addme is larger than pos->next

pos = pos->next;

}

}

}

}

/\*\*/

// Method for displaying a student's details.

// Takes a pointer to a student as an argument.

void display\_single\_student(Person single\_student) {

// Name

printf( "Name:\t\t\t%s",

single\_student->student\_info.name);

// ID

printf( "ID:\t\t\t%s",

single\_student->student\_info.id);

// Course name

printf( "Course name:\t\t%s",

single\_student->course\_info.course\_name);

// Marks

for( int i = 0;

i < single\_student->course\_info.no\_of\_units;

i++) {

printf( "Mark #%d:\t\t%d\n",

i + 1,

single\_student->course\_info.marks[i]);

}

// Average mark

if(!isnan(single\_student->course\_info.avg)) {

printf( "Average mark:\t\t%.2f\n",

single\_student->course\_info.avg);

}

printf("\n");

}

// Menu option 1 display\_students

void display\_students(void) {

Person pos = NULL;

pos = Students\_Details;

printf("\n");

while(pos != NULL) {

display\_single\_student(pos);

// Step through list.

pos = pos->next;

}

}

// Read file passed as argument.

// Create a linked list of elements described in file.

// Output pointer to index of linked list created.

void read\_file() {

clean\_up\_Students\_Details();

//

// students.txt file formatting for reference:

//

// char[20] student\_info->name

// char[10] student\_info->id

// char[20] course\_info->course\_name

// int course\_info->no\_of\_units

// int course\_info->marks[0]

// int course\_info->marks[1]

// int course\_info->marks[2]

// int course\_info->marks[3]

//

// One variable is not provided by the file.

// float course\_tag GENERATED IN RUNTIME

//

// Open file before starting, don't open it in the function

// because the function is called repeatedly.

Students\_Details\_FILE = open\_file("r");

// Store the pointer to pass to the inserter function.

Person temp = NULL;

// Create a new Person, return it.

Person create\_person(void) {

// The return value.

Person newPerson = NULL;

// Line content pointer to string malloc'd by getline.

char \*line = NULL;

// Line length.

// (For -1 when line not there use ssize\_t and grab it

// from the return value of getline())

// This value is ignored because line == NULL,

// but we still need it.

size\_t len = 0;

// Only continue if there is a line to be read.

if((getline(&line, &len, Students\_Details\_FILE)) != -1) {

// Create the person to be returned.

newPerson = (Student\_Tag\*) malloc(sizeof(Student\_Tag));

// Make next == NULL so we don't cause errors later.

newPerson->next = NULL;

// Get the name.

// (using getline line from while statement)

strcpy(newPerson->student\_info.name, line);

// Get the id.

getline(&line, &len, Students\_Details\_FILE);

strcpy(newPerson->student\_info.id, line);

// Get the course name.

getline(&line, &len, Students\_Details\_FILE);

strcpy(newPerson->course\_info.course\_name, line);

// Get the number of units.

getline(&line, &len, Students\_Details\_FILE);

newPerson->course\_info.no\_of\_units = atoi(line);

if(newPerson->course\_info.no\_of\_units > 4) {

newPerson->course\_info.no\_of\_units = 4;

printf("ERROR: Max subjects == 4\n");

}

// Get the marks.

float average = 0.00;

for(int i = 0;

i < newPerson->course\_info.no\_of\_units;

i++) {

getline(&line, &len, Students\_Details\_FILE);

newPerson->course\_info.marks[i] = atoi(line);

average += newPerson->course\_info.marks[i];

}

average /= newPerson->course\_info.no\_of\_units;

newPerson->course\_info.avg = average;

// Testing line to print average as it's calculated.

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

}

// Free the line malloc'd by getline.

free(line);

return newPerson;

}

// Run until create\_person no longer returns anything.

while((temp = create\_person()) != NULL) {

place\_into\_list(temp);

}

// Close file only after running create\_person function since

// it relies on it being open.

fclose(Students\_Details\_FILE);

// Display the results afterwards.

display\_students();

}

// Menu option 2 search\_student

void search\_student(void) {

printf("Please type the student's name. (caps insensitive) ");

char needle[20];

fflush(stdin);

fgets(needle, 19, stdin);

// Get rid of newline.

for(int i = 0; needle[i] != '\0'; i++) {

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

needle[i] = toupper(needle[i]);

}

printf("Searching for: %s\n\n", needle);

Person haystack = Students\_Details;

size\_t ammount\_found\_counter = 0;

while(haystack != NULL) {

char temp[20] = "";

for(int i = 0;

haystack->student\_info.name[i] != '\0';

i++) {

temp[i] = toupper(haystack->student\_info.name[i]);

}

if(strstr(temp, needle) != NULL) {

display\_single\_student(haystack);

ammount\_found\_counter++;

}

haystack = haystack->next;

}

printf("%d student(s) found.\n", ammount\_found\_counter);

}

// Menu option 3 find\_maximum

void find\_maximum(void) {

Person pos = Students\_Details;

Person highestpos = Students\_Details;

if(Students\_Details != NULL) {

// Determine highest average mark.

while(pos->next != NULL) {

if( pos->next->course\_info.avg >

highestpos->course\_info.avg) {

highestpos = pos->next;

}

pos = pos->next;

}

printf( "The highest average mark is held by %s\n",

highestpos->student\_info.name);

display\_single\_student(highestpos);

} else {

printf("No student data to search.");

}

}

// Menu option 4 find\_failed

void find\_failed(void) {

//while(pointer != NULL) {

// if(FAIL) {display\_single\_student(pointer)}

//}

printf( "Displaying students who failed at least one subject."

"(mark < 50)\n\n");

Person pos = Students\_Details;

bool failcheck(Person checkme) {

for(int i = 0;

i < checkme->course\_info.no\_of\_units;

i++) {

if(checkme->course\_info.marks[i] < 50) {return true;}

}

return false;

}

while(pos != NULL) {

if(failcheck(pos)) {display\_single\_student(pos);}

pos = pos->next;

}

}

// Menu option 5 update\_file

// Open the file and add new student details.

void update\_file(void) {

// Append new details to file.

Students\_Details\_FILE = open\_file("a");

char uinstring[20] = "";

size\_t uinint = 0;

void remove\_newline\_char(char \*purify) {

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

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

}

}

printf("Add new student to file\n");

void textin(const char \*askfor, const size\_t textin\_len) {

strcpy(uinstring, "");

while(strlen(uinstring) == 0) {

printf("%s", askfor);

fflush(stdin);

fgets(uinstring, textin\_len, stdin);

remove\_newline\_char(uinstring);

if(strlen(uinstring) == 0) {

printf("ERROR: String empty.\n");

}

}

fprintf(Students\_Details\_FILE, "%s\n", uinstring);

}

textin("Name : ", 19);

textin("ID : ", 9);

textin("Course name : ", 19);

uinint = 5;

while(uinint > 4) {

printf("Number of units (Max 4) : ");

uinint = user\_input\_number();

if(uinint > 4) {

printf("ERROR: Maximum number of units == 4\n");

}

}

fprintf(Students\_Details\_FILE, "%d\n", uinint);

for(int i = uinint; i > 0; i--) {

uinint = 101;

while(uinint < 0 || uinint > 100) {

printf("Mark (0-100) : ");

uinint = user\_input\_number();

if(uinint < 0 && uinint > 100) {

printf("ERROR: Mark not within range (0-100)\n");

}

}

fprintf(Students\_Details\_FILE, "%d\n", uinint);

}

fclose(Students\_Details\_FILE);

// Re-parse the file.

// Average is calculated in here.

read\_file();

}

// Menu option 0 debug\_menu

void debug\_menu(void) {

printf("\n\n\nDebug menu\n\n\n");

printf("incomplete\n");

}

// Main menu

bool menu(void) {

void clear(void) {

for(int i = 0; i < 100; i++) {printf("\n");}

}

clear();

void anykey(void) {

printf("Press enter to continue...\n");

// Take input but ignore it in order to halt the terminal.

user\_input\_number();

}

printf( "\n"

" ,---.| | | \n"

" `---.|--- . .,---|,---.,---.|--- \n"

" || | || ||---'| || \n"

" `---'`---'`---'`---'`---'` '`---' \n"

" \n"

" | | \n"

" | ,---.,---.|\_\_/ . .,---. \n"

" | | || || \\ | || | \n"

" `---'`---'`---'` ``---'|---' \n"

" | \n"

"\n"

"(1) Display students' details\n"

"(2) Search for a student's marks\n"

"(3) Find the student with the highest average mark\n"

"(4) Display students who have failed subjects\n"

"(5) Add a new student to the record\n"

"(6) Quit program\n"

"\n");

switch(user\_input\_number()) {

case 1:

clear();

display\_students();

anykey();

break;

case 2:

clear();

search\_student();

anykey();

break;

case 3:

clear();

find\_maximum();

anykey();

break;

case 4:

clear();

find\_failed();

anykey();

break;

case 5:

clear();

update\_file();

anykey();

break;

case 0:

debug\_menu();

anykey();

break;

default:

printf("Exiting");

fflush(stdout);

for(int i = 0; i < 5; i++) {

usleep(500000);

printf(".");

fflush(stdout);

}

printf("\n");

return false;

}

return true;

}

// Make sure the data is read from the file before starting.

read\_file(Students\_Details\_FILE);

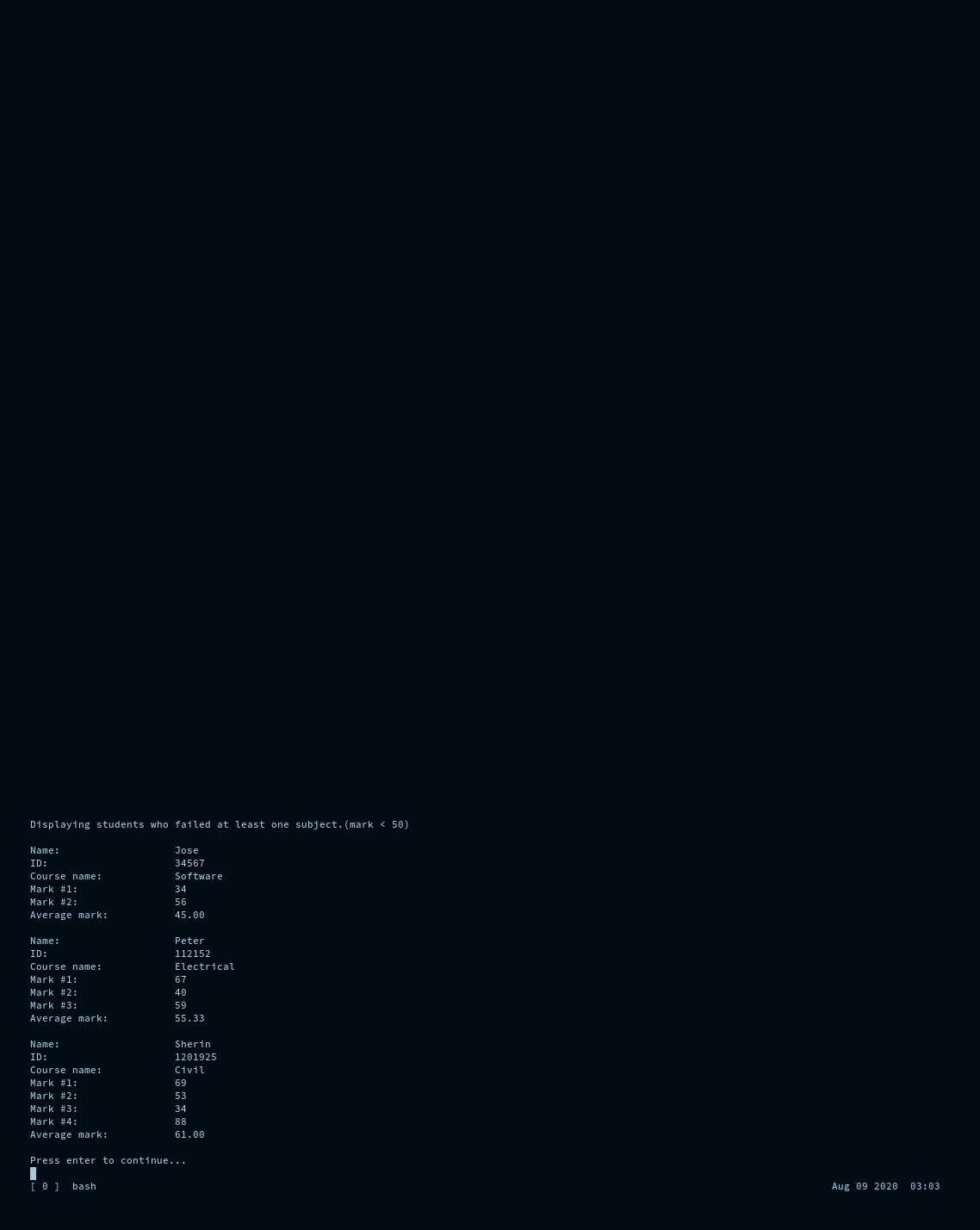
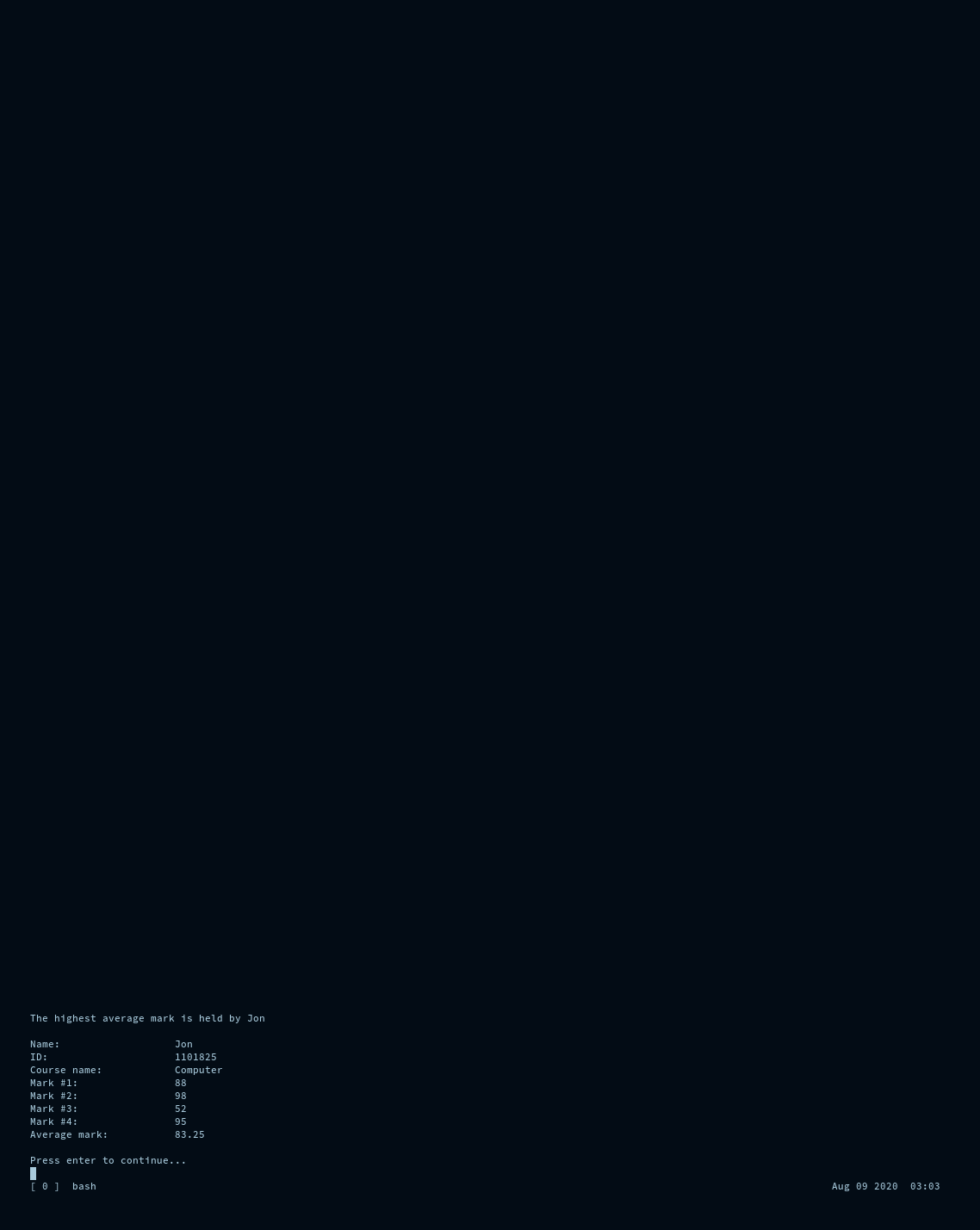
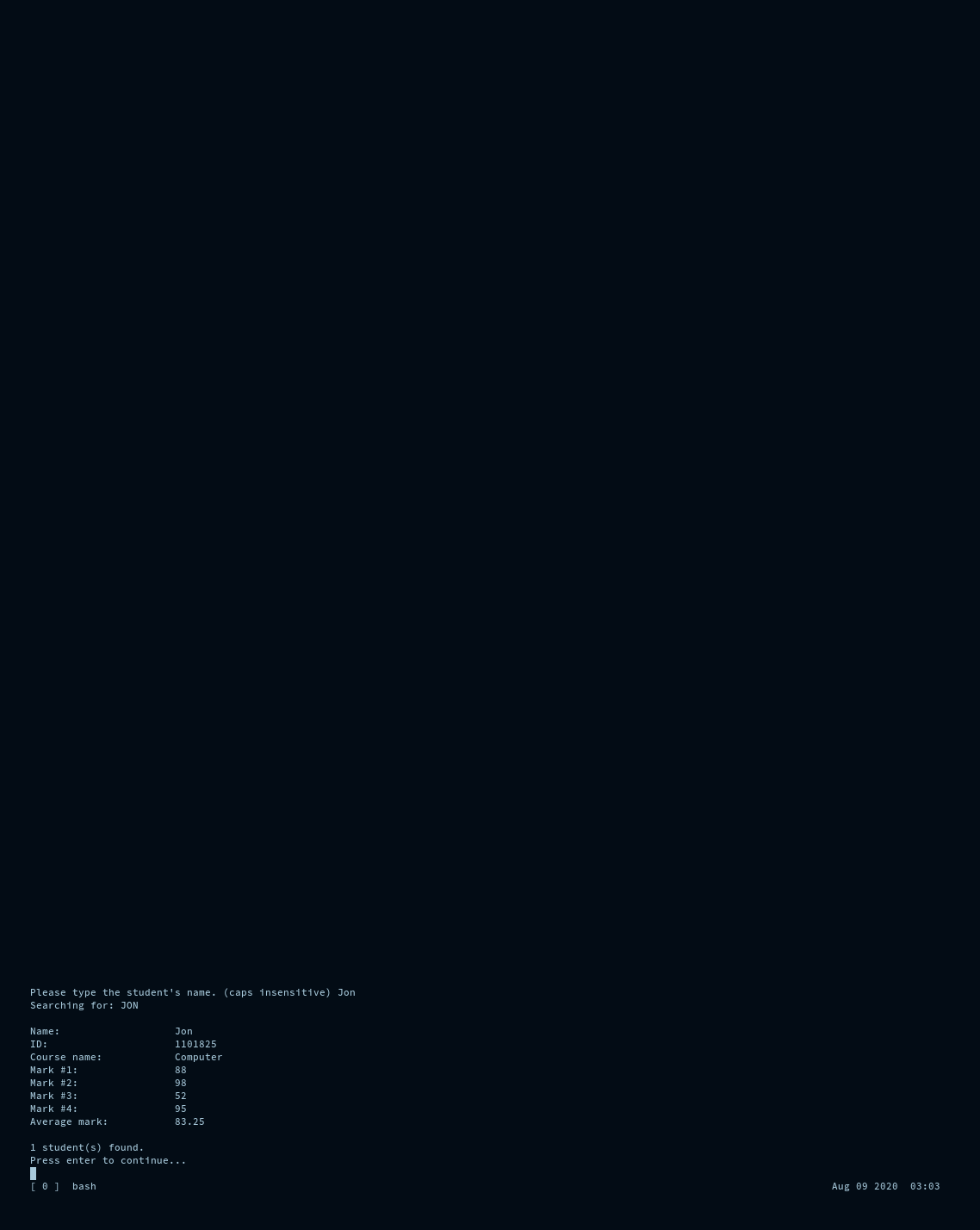
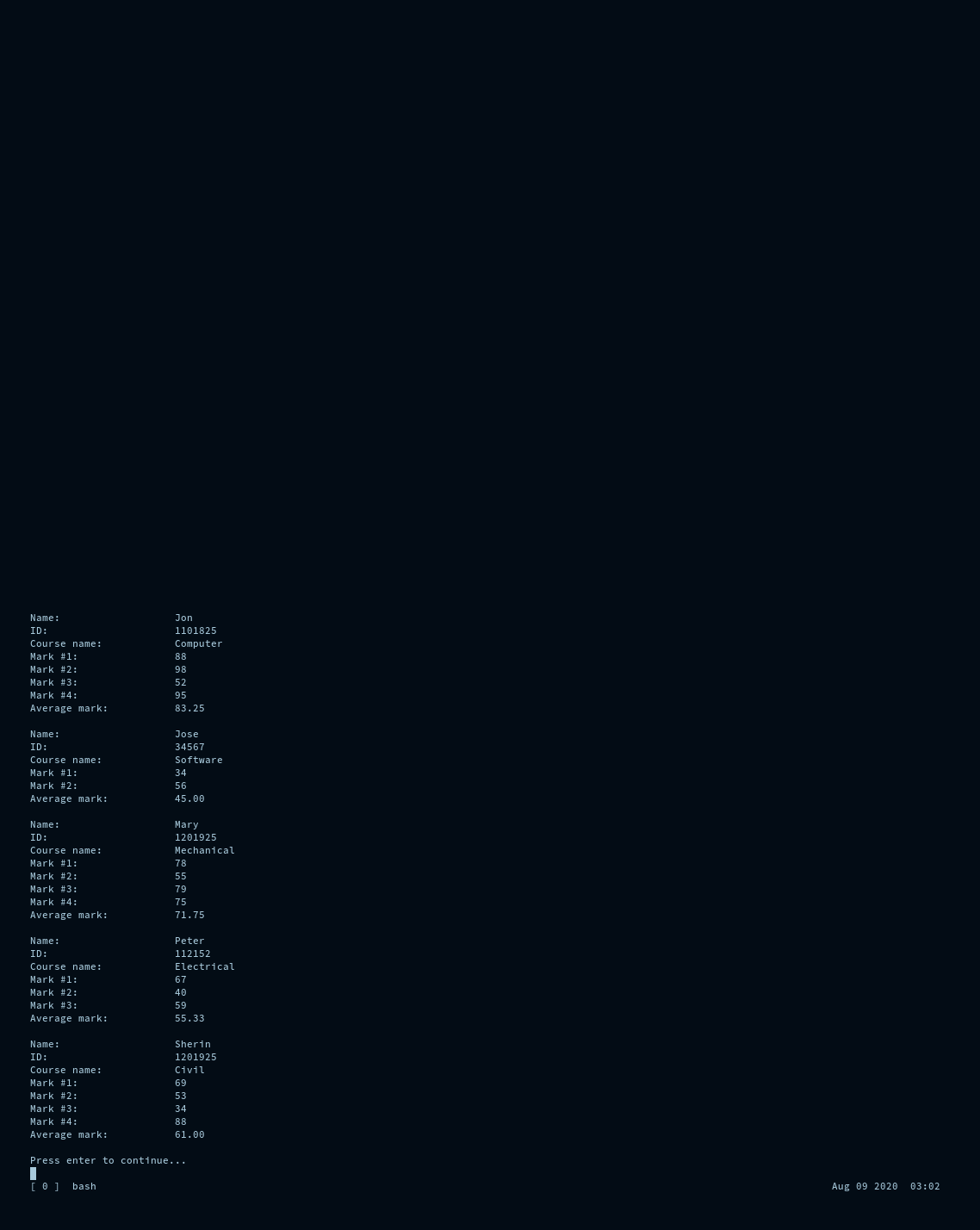
// Loop the menu till the menu returns false.

while(menu()) {}

clean\_up\_Students\_Details();

return 0;

}



## Original File

Jon

1101825

Computer

4

88

98

52

95

Peter

112152

Electrical

3

67

40

59

Mary

1201925

Mechanical

4

78

55

79

75

Sherin

1201925

Civil

4

69

53

34

88

Jose

34567

Software

2

34

56

## Updated File

Jon

1101825

Computer

4

88

98

52

95

Peter

112152

Electrical

3

67

40

59

Mary

1201925

Mechanical

4

78

55

79

75

Sherin

1201925

Civil

4

69

53

34

88

Jose

34567

Software

2

34

56

Boris Johnson

240719

PM

2

64

56

Section A Part 2

// size\_t

#include <stdlib.h>

// sleep

//#include <unistd.h>

// text interaction

#include <stdio.h>

// bool

//#include <stdbool.h>

// strcpy

#include <string.h>

// toupper

//#include <ctype.h>

// isnan

//#include <math.h>

int main(int argc, char \*argv[]) {

// Handle arguments.

// Check ammount of arguments.

if(argc != 1) {

// Incorrect number of arguments,

// display a terse message then kill the program.

printf("No arguments accepted.\n");

return 1;

}

// !!! CAN'T CHANGE THIS STRUCT THIS IS A FIXED PART OF THE ASSIGNMENT !!!

struct studentID {

int value;

struct studentID \*next;

};

typedef struct studentID STUDENTID;

typedef STUDENTID \* STUDENTIDPtr;

// !!! END FIXED PART OF THE ASSIGNMENT !!!

void clear(void) {

const size\_t lines\_to\_clear = 100;

for(int i = 0; i < lines\_to\_clear; i++) {printf("\n");}

}

clear();

STUDENTIDPtr index\_ptr = NULL;

STUDENTIDPtr temp\_ptr = NULL;

STUDENTIDPtr new\_ptr = NULL;

STUDENTIDPtr curr\_ptr = NULL;

new\_ptr = (STUDENTID\*) malloc(sizeof(STUDENTID));

new\_ptr->value = 4;

new\_ptr->next = NULL;

index\_ptr = new\_ptr;

temp\_ptr = new\_ptr;

new\_ptr = (STUDENTID\*) malloc(sizeof(STUDENTID));

new\_ptr->value = 5;

new\_ptr->next = NULL;

temp\_ptr->next = new\_ptr;

temp\_ptr = new\_ptr;

new\_ptr = (STUDENTID\*) malloc(sizeof(STUDENTID));

new\_ptr->value = 3;

new\_ptr->next = NULL;

temp\_ptr->next = new\_ptr;

temp\_ptr = new\_ptr;

new\_ptr = (STUDENTID\*) malloc(sizeof(STUDENTID));

new\_ptr->value = 4;

new\_ptr->next = NULL;

temp\_ptr->next = new\_ptr;

temp\_ptr = new\_ptr;

new\_ptr = (STUDENTID\*) malloc(sizeof(STUDENTID));

new\_ptr->value = 5;

new\_ptr->next = NULL;

temp\_ptr->next = new\_ptr;

temp\_ptr = new\_ptr;

// Please just give me a number with duplicates next time,

// my student number doesn't have any.

// I am using the student number from the example in the PDF.

void printit(void) {

curr\_ptr = index\_ptr;

int i = 1;

while(curr\_ptr != NULL) {

printf("Value %d: %d\n", i, curr\_ptr->value);

curr\_ptr = curr\_ptr->next;

i++;

}

printf("\n");

}

// Print the ll before the changes.

printf("This is the linked list before removing duplicates.\n");

printit();

void remove\_node(STUDENTIDPtr remove\_me, STUDENTIDPtr remove\_me\_prev) {

remove\_me\_prev->next = remove\_me->next;

free(remove\_me);

}

// This is where we remove the duplicate instances.

curr\_ptr = index\_ptr;

int inspect;

while(curr\_ptr != NULL) {

inspect = curr\_ptr->value;

temp\_ptr = curr\_ptr;

while(curr\_ptr->next != NULL) {

if(curr\_ptr->next->value == inspect) {

remove\_node(curr\_ptr->next, curr\_ptr);

} else {

curr\_ptr = curr\_ptr->next;

}

}

curr\_ptr = temp\_ptr->next;

}

printf("This is the linked list after removing duplicates.\n");

printit();

printf("\n\n");

// Clean up memory afterwards.

while(index\_ptr != NULL) {

curr\_ptr = index\_ptr->next;

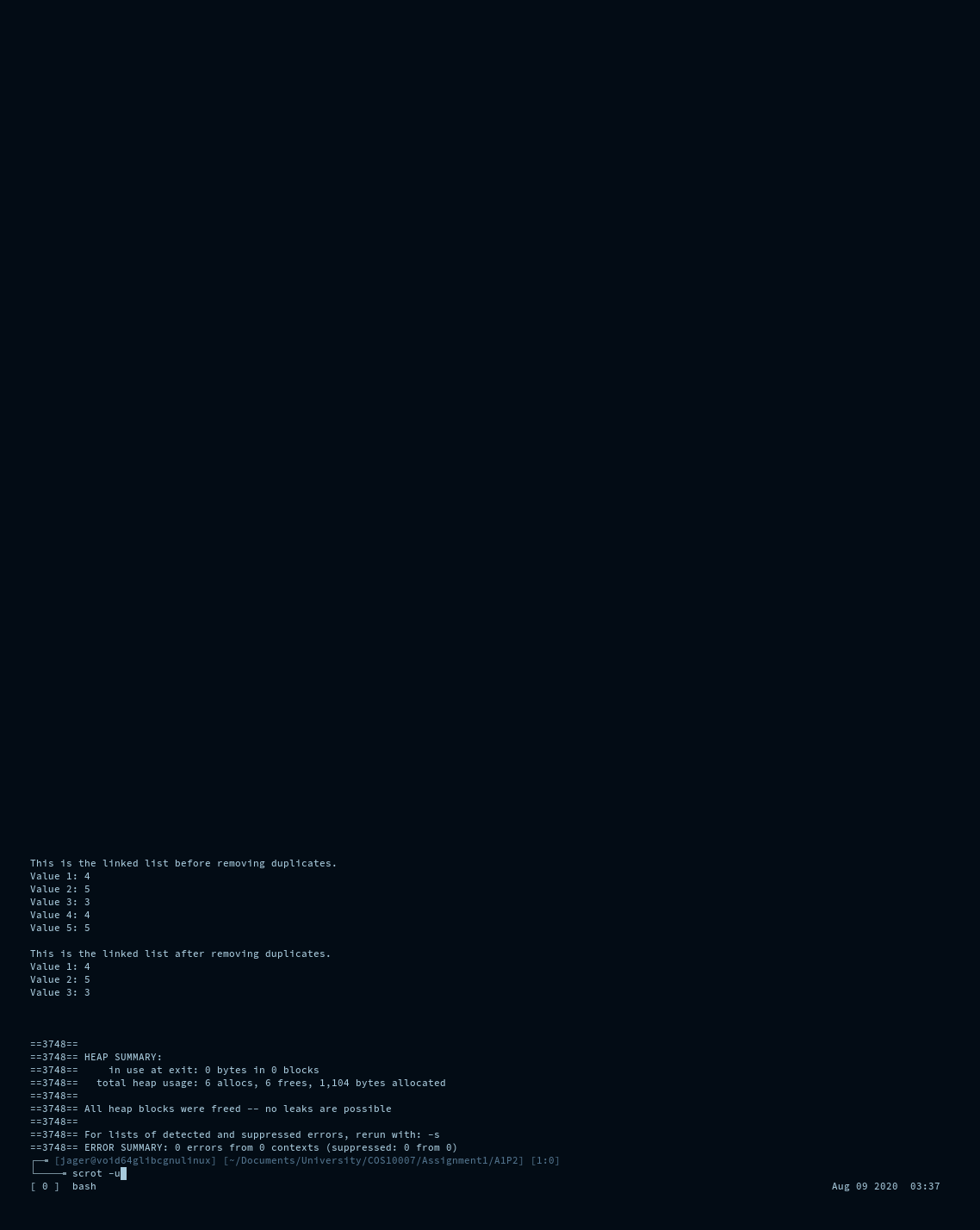
free(index\_ptr);

index\_ptr = curr\_ptr;

}

return 0;

}



Section A Part 3

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

/\*

\*

\* Assignment 1 Section A Question 3

\*

\*/

// I'm using line comments instead of block comments deal with it.

// !!! FIXED PART OF ASSIGNMENT DO NOT CHANGE !!!

struct Bus {

// Unique value. Example: 1

int BusID;

// Unique value. Example: 1001

int RouteID;

// Scheduled departure time

time\_t schedule;

};

// !!! END FIXED PART OF ASSIGNMENT !!!

// Typedef laziness.

typedef struct Bus bus;

typedef bus \* bus\_ptr;

// !!! FIXED PART OF ASSIGNMENT DO NOT CHANGE !!!

bus Depot[10];

// !!! END FIXED PART OF ASSIGNMENT !!!

// FUNCTION PROTOTYPES

// This is to keep the menu happy so I don't have it resetting it

// to a null pointer every time the menu loops.

bus\_ptr Depot\_pointer = NULL;

bool menu(void);

void clear(void);

int userinputint(void);

void printBuses(void);

bus\_ptr createBuses(void);

void scheduleBuses(void);

void alignupBuses(void);

struct Bus\_ll {

int BusID;

int RouteID;

time\_t schedule;

struct Bus\_ll \* next;

};

// Typedef laziness.

typedef struct Bus\_ll bus\_ll;

typedef bus\_ll \* bus\_ll\_ptr;

// Lazy pile of global variables so I don't have to pass the variable

// into functions as often.

bus\_ll\_ptr Depot\_ll\_head = NULL;

bus\_ll\_ptr Depot\_ll\_pos = NULL;

bus\_ll\_ptr Depot\_ll\_tail = NULL;

bus\_ll\_ptr releaseBus(bus\_ll\_ptr prev\_head);

bus\_ll\_ptr emergency(bus\_ll\_ptr prev\_tail);

void cleanup(void);

void debug\_menu(void);

int main(void) {

srandom(time(NULL));

// Loop the main menu till it returns false.

while(menu()){}

// Free the memory that was allocated to the linked list.

cleanup();

return 0;

}

// Menu : Looped from main function until it returns false.

bool menu(void) {

clear();

printf( "Bus depot scheduling program\n\n"

"1 : Initialize depot\n"

"2 : Print depot contents\n"

"3 : Schedule buses\n"

"4 : Align buses\n"

"5 : Release bus\n"

"6 : Emergency release bus\n"

"7 : Exit program\n\n");

// Generic error message for the menu to display when the depot

// hasn't been initialized yet.

void depoterror(void) {

printf("The Depot hasn't been initialized yet.\n");

}

// Generic error message for when the alignupBuses function

// hasn't been run yet.

void alignerror(void) {

printf( "The depot hasn't been aligned yet or there are no "

"buses in the list.\n"

"You may realign the buses to repopulate the list if "

"you have initialized the depot already.\n");

}

switch(userinputint()) {

case 1:

// init depot

// Depot\_pointer is for determining if the bus creation

// function has run. It is NULL by default.

if(Depot\_pointer == NULL) {

Depot\_pointer = createBuses();

printf("Depot initialized.\n");

} else {

printf("Depot already initialized.\n");

}

break;

case 2:

// print depot

if(Depot\_pointer != NULL) {

printBuses();

} else {

depoterror();

}

break;

case 3:

// schedule depot

if(Depot\_pointer != NULL) {

scheduleBuses();

printf("Buses scheduled.\n");

} else {

depoterror();

}

break;

case 4:

// alignup buses

alignupBuses();

printf("Buses aligned.\n");

break;

case 5:

// release buses

if(Depot\_ll\_head != NULL) {

free(releaseBus(Depot\_ll\_head));

} else {

alignerror();

}

break;

case 6:

// emergency release bus

if(Depot\_ll\_head != NULL) {

free(emergency(Depot\_ll\_tail));

} else {

alignerror();

}

break;

case 0:

// debug menu

printf("Secret debug menu.\n");

debug\_menu();

break;

default:

return false;

}

// Make sure to stop after running the function as we may want to

// read what is on the screen.

printf("Press any key to continue...");

userinputint();

return true;

}

// Clear the screen of any text the lazy way.

void clear(void) {

// Ammount of newlines to print.

const size\_t newlinecount = 100;

for(int i = 0; i < newlinecount; i++) {printf("\n");}

}

// Take user input and return int.

int userinputint(void) {

char temp[32];

fflush(stdin);

fgets(temp, 31, stdin);

return atoi(temp);

}

// Print the details of all the buses in the Depot array.

void printBuses(void) {

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

printf( "Depot[%d]\n"

"BusID : %d\n"

"RouteID : %d\n"

"schedule : %s\n\n",

i,

Depot[i].BusID,

Depot[i].RouteID,

ctime(&Depot[i].schedule));

}

}

// Create 10 buses and store them in the Depot array.

// BusID and RouteID to be filled with random integer values.

// Schedule to be left blank.

bus\_ptr createBuses(void) {

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

Depot[i].BusID = random() % 10000;

Depot[i].RouteID = random() % 100;

}

return Depot;

}

// The departure time for each bus to be filled, refer to sample code.

void scheduleBuses(void) {

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

Depot[i].schedule = time(NULL)+random()%10000;

}

}

// Buses to be rearranged in the Depot array based on each buses

// scedule.

// The earliest scheduled bus to be placed at the bottom of the Depot

// array.

void alignupBuses(void) {

// Place for storing a bus that we will move.

bus temp;

// Rearrange array such that earliest is at the "bottom"

// of the array, which I'm assuming is the end of the array.

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

for(int a = i + 1; a < 10; a++) {

if(Depot\_pointer[i].schedule < Depot\_pointer[a].schedule) {

// Switch positiions.

temp = Depot\_pointer[i];

Depot\_pointer[i] = Depot\_pointer[a];

Depot\_pointer[a] = temp;

}

}

}

// Export the data to the linked list which is where the data

// should have been in the first place.

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

bus\_ll\_ptr newBus = (bus\_ll\*) malloc(sizeof(bus\_ll));

newBus->BusID = Depot\_pointer[i].BusID;

newBus->RouteID = Depot\_pointer[i].RouteID;

newBus->schedule = Depot\_pointer[i].schedule;

if(Depot\_ll\_head == NULL) {

Depot\_ll\_tail = newBus;

}

newBus->next = Depot\_ll\_head;

Depot\_ll\_head = newBus;

}

}

// Release one bus at a time from the Depot based on the schedule.

// Bus with the earliest schedule should leave the Depot first.

// Queue concept. (FIFO)

// Remove the head of the linked list, which will also be the bus

// with the earliest departure time.

bus\_ll\_ptr releaseBus(bus\_ll\_ptr prev\_head) {

Depot\_ll\_head = Depot\_ll\_head->next;

// Return the pointer to the old head of the list in case it

// needs to be used elsewhere. (It is freed in the menu function.)

return prev\_head;

}

// Release one bus at a time from the Depot,

// with the last scheduled bus to leave the Depot array first.

// Stack concept. (FILO)

// Remove the tail of the linked list, which will also be the bus

// with the latest departure time.

bus\_ll\_ptr emergency(bus\_ll\_ptr prev\_tail) {

// Find new tail. (2nd from end)

// I don't actually need the pointer to the tail to do this,

// I'm just using it to demonstrate that it's the tail of the list.

// This while loop could just be:

// while(Depot\_ll\_pos->next->next != NULL) {

// Depot\_ll\_pos = Depot\_ll\_pos->next;

// }

Depot\_ll\_pos = Depot\_ll\_head;

if(Depot\_ll\_head != Depot\_ll\_tail) {

while(Depot\_ll\_pos->next != Depot\_ll\_tail) {

Depot\_ll\_pos = Depot\_ll\_pos->next;

}

// Set new tail.

Depot\_ll\_tail = Depot\_ll\_pos;

Depot\_ll\_tail->next = NULL;

} else {

Depot\_ll\_head = NULL;

Depot\_ll\_tail = NULL;

}

// Return the pointer to the old tail of the list in case it

// needs to be use elsewhere. (It is freed in the menu function.)

return prev\_tail;

}

// Print the linked list that isn't printed by the print function

// because the assignment has a stipulation that the printBuses

// function uses an array instead of the linked list.

// The fact that I need this is absurd, whoever designed this

// assignment definitely didn't think it through propperly.

void debug\_menu(void) {

Depot\_ll\_pos = Depot\_ll\_head;

while(Depot\_ll\_pos != NULL) {

// Print whether this is considered to be the head of the list.

if(Depot\_ll\_pos == Depot\_ll\_head) {printf("Head\n");}

// Print whether this is considered to be the tail of the list.

if(Depot\_ll\_pos == Depot\_ll\_tail) {printf("Tail\n");}

// Print data stored in struct.

printf("BusID : %d\n", Depot\_ll\_pos->BusID);

printf("RouteID : %d\n", Depot\_ll\_pos->RouteID);

printf("schedule : %s\n", ctime(&Depot\_ll\_pos->schedule));

// Move to next position.

Depot\_ll\_pos = Depot\_ll\_pos->next;

}

}

// Cleanup function to free the data in the linked list.

// This doesn't free any items stripped from the linked list using:

// releaseBus()

// emergency()

// Those are freed elsewhere.

void cleanup(void) {

while(Depot\_ll\_head != NULL) {

Depot\_ll\_pos = Depot\_ll\_head;

Depot\_ll\_head = Depot\_ll\_pos->next;

free(Depot\_ll\_pos);

}

}

