CPS1011 Report

Link to GitLab: https://gitlab.com/mark_dingli/cps1011-assignment

Header file

- This includes all the function prototypes that are used in the menu for Question (1b).

```
int init_array(int arr[]);
void display(int arr[], int sum);
void reverse(int arr[], int rev[], int sum);
void frequency(int arr[], int freq[], int sum);
void display_freq(int arr[], int freq[], int sum, int visited);
```

- This line is used to link the menu code from Question (1b) to the header file:

```
#include "question_1_header.h"
```

#define

```
#define MAXSIZE 200
```

MAXSIZE – This is used for the max size of the array

Variable Declarations

int arr[MAXSIZE]; – stores the inputted elements in init_array() function

int rev[MAXSIZE]; – stores the reversed elements in the reverse() function

int freq[MAXSIZE]; – stores the frequency pairs in the frequency() function

int choice; – stores the user's choice for the menu

int sum; – stores the size of the array

int visited = -1; – used in the display_freq() function

int count = 0; – stores the index where the element is to be stored

1. Problem solving

(a) Implement the following array-manipulating functions:

i. init_array()

- This function asks the user to enter (fill up) the array elements. The user can enter up to a maximum of 200 elements. If the user enters more than 200 elements the array will not take in any more inputs. Only numbers can be entered into the array. When entering the elements into to array, the user can press 'q' or 'Q' whenever they want to stop inputting numbers into the array. After the user enters 'q' or 'Q', the array elements and the size of the array are outputted

onto the screen. This function returns the variable *count* which stores the index where the element is to be inserted.

ii. display()

- This function takes the *arr[]* parameter as well as the *sum* parameter. This function will take the array elements that were inputted by the user in the previous *init_array()* function and will output them in the format:

```
{
   "array":  [
      {
         "offset": "00",
         "value": "05"
       },
       {
         "offset": "01",
         "value": "150",
        }, ....
   ]
}
```

- as shown below, *\n \"* and spaces were used to achieve this output format.

```
printf("\n{\n  \"array\": [\n");
for (int i = 0; i < sum; i++) {
    printf("     {\n        \"offset\": \"%d\",\n        \"value\": \"%d\"\n
},\n\n",i, arr[i]);
}
printf("  ]\n}");
```

iii. reverse()

- This function takes *arr[],rev[] and sum* as parameters. This function takes the *arr[]* array elements and reverses their order. The new reversed array elements are stored in the *rev[]* array. This function is destructive. This means that after the *reverse()* function, the previous elements of the *arr[]* array *are* overridden.

iv. frequency()

- This function takes *arr[],freq[] and sum* as parameters. This function takes the *arr[]* array and calculates the frequency of every element in the array. It basically calculates the number of times an element occurs in the array. It does this by using a for loop and increases the count after every iteration. To avoid counting the same element again the function uses a nested for loop and the *visited* variable to check if the element was counted already.

```
int visited = -1;

//Calculating frequency
for(int i = 0; i < sum; i++) {
    int count = 1;
    for (int j = i + 1; j < sum; j++) {
        if (arr[i] == arr[j]) {
            count++;
            //To avoid counting same element again
            freq[j] = visited;
        }
    }
    if (freq[i] != visited)
        freq[i] = count;
}
```

After the program calculates the frequency of the array elements, it will output them in the format:

```
{{5,2},{6,1},...}
```

Where the number on the left is the array element and the number on the right is the frequency of that number.

v. display_freq()

- This function takes *arr[],freq[], sum* and *visited* as parameters. This function takes the *arr[]* and the *freq[]* array and outputs each array element and the number of times it occurs. For example:

```
Frequency of all the elements inside the array :

3 occurs 3 times

1 occurs 1 times

2 occurs 2 times
```

The function will then use a for loop to output the frequency of the elements in the same format that was used in the display() function before.

(b) Write a program that presents the end-user with a command-line menu…

- When the user runs the program, the following command-line menu will be outputted onto the screen:

```
============================

            MENU

============================

1. init_array()

2. display()

3. reverse()

4. frequency()

5. display_freq()

6. Quit


Enter Choice:
```

The user will be asked to enter a choice. The choice must be between 1 and 6. If the user enters an invalid number or any other invalid symbol, an error message will show up on screen telling the user to enter a valid number between 1 and 6. The previous functions will be executed depending on which number the used decides to choose. It is suggested that the user starts from option 1 and ends at option 6. This will lead to the best execution of the menu and functions. If the user presses the number 6 the menu will output a thank you message on screen and will stop the program from running.

2. A DataTable library

a. Implement a simplified version of the data type and functions described above.

Header file

- This includes all the function prototypes that are used in testing in question (2a)

```
/*Functions*/
DataTable_T *initDT();
DataTable_T *deinitDT(DataTable_T *memory);
int loadDT(DataTable_T DT[]);
int exportDT(int count, struct DataTable_T const *data, char const
*fileName);
int showDT(int row_count, char line[M2][M1]);
void projectDT(char line[M2][M1]);
//MutateDT() function:
void sortByBMI(DataTable_T sort_by_bmi[], int row_count);
void sortByFirstName(DataTable_T sort_by_bmi[], int row_count);
void sortByLastName(DataTable_T sort_by_bmi[], int row_count);
```

#define

- The define are all in the header file. This makes them easier to access by all the other files.

```
#define MAX 63 //Max size for String(array)
#define M1 130 //Used for showDT()
#define M2 20  //Used for showDT()
```

MAX – This is used to hold the size of the array in the DataTable_T struct.

M1 – This is used in the showDT() function for the array char line as its size.

M2 – This is also used in the showDT() function for the array char line as its size.

The Structure

- The struct is also in the header file. This makes it easier to access by all the other file.

```
typedef struct DataTable_T{
    float Height[5];
    float Weight[5];
    float BMI[5];
    char Name[MAX];
    char Surname[MAX];
    char Gender[MAX];
} DataTable_T;
```

float Height[5]; - Array inside the struct of size 5 that stores the Height

float Weight[5]; - Array inside the struct of size 5 that stores the Weight

float BMI[5]; - Array inside the struct of size 5 that stores the BMI

char Name[MAX]; - Array inside the struct of size MAX that stores the Name

char Surname[MAX]; - Array inside the struct of size MAX that stores the Surname

char Gender[MAX]; - Array inside the struct of size MAX that stores the Gender

Variable Declarations

DataTable_T *memory; - Pointer to DataTable_T structure

int row_count; - Stores the number of rows in the csv file

DataTable_T DT[100]; - Array of size 100 that stores the lines of the csv file

char pointer; - Holds the string to be copied

char buffer[200]; - Array of size 200 who's strings are split in strtok()

char buffer[200]; - Array of size 200 who's strings are split in strtok()

char line[M2][M1]; - Arrays to store the lines

int k; - array counter

int x; - used in projectDT() to print from the 'x' row of the csv file

int y; - used in projectDT() to print from the 'y' row of the csv file

DataTable_T; - array with the same details to modify for sorting in mutateDT()

- **initDT()**

- This function allocates memory to the variable pointer *memory*. If the memory is not allocated to the variable, an error message will appear. If the memory is allocated, an output message "Success!" will appear. Then, the function will return the pointer *memory* to a newly initialized DataTable structure.

- **deinitDT()**

- This function deallocates/frees the memory that was allocated in the *initDT()* function. After the memory is deallocated, a confirmation message "Success!" will be outputted. Then, the function will return the pointer *memory*.

- **loadDT()**

- This function loads the content of a comma-separated values file into the initialized DataTable_t structure. Firstly, the function will open the *health.csv* file in read mode. If the file *health.csv* cannot be open, an error message will be printed out and the program will stop executing (*return 1)*.

```
//open the file
FILE* file = fopen("health.csv", "r");
if (file == NULL) {
    perror("Unable to open the file!"); //Error code
    return 1;
}
```

- When the file is opened, the arrays *DT* and *buffer* are initialized. A while loop is used. This while loop will keep iterating until the end of the file is reached. It does this with the feof() function.

```
while (!feof(file))
```

- The function reads the line using the fgets() function. Using the strtok() function, this breaks string into a series of tokens using the delimiter. This is stored in the variable *pointer.*

```
fgets(buffer, 200, file); //read line
pointer = strtok(buffer, ",");
```

- In the if statement, the strtok() function is used to load the content of the csv file and place it in the array *DT*. This is done for every row and column inside the csv file. All the content is loaded into the DataTable_t structure. With each iteration the *row_count* is incremented and hold the number of rows. The file is then closed.

- **exportDT()**

- This function exports the content of an initialized DataTable_t to a comma-separated values (csv) file. The exportDT() function takes the row_count and DT as parameters. A new file called "export_1.csv" is created and the DataTable_t content is exported to this file. Firstly, the function will open the file *export.csv* in write mode. If the file cannot be open, an error message will be printed out and the program will stop executing (*return 1)*.

```
FILE *f=fopen(fileName,"w");
if (f == NULL) {
    perror("Unable to open the file!"); //Error code
    return 1;
}
```

- A for loop is used with fprintf(). This will read and export the content of the DataTable_t to the file *export.csv*. The file is then closed.

```
for (int i=0; i<count; i++)
{
    fprintf(f,"%s,%s,%s,%s,%s,%s",  data[i].Height, data[i].Weight,
data[i].BMI, data[i].Name, data[i].Surname, data[i].Gender);
}

printf("Exported to file - export.csv!\n");

fclose(f);
```

- **showDT()**

- This function prints a DataTable t to screen, only displaying the labels, followed by the first ten rows, a line made up of dots ''...'' and the final row. Firstly, the function will open the *health.csv* file in read mode. If the file health.csv cannot be open, an error message will be printed out and the program will stop executing (return 1). Then the function will use a while to get the labels of

the csv file. The variable k is incremented in the while loop, this variable will store the number of lines.

```
while(fgets(line[k],M1,fdpo))
{
    line[k][strlen(line[k])-1]='\0';
    k++;
}
```

- A for loop is used to print the labels and the first 10 lines.

```
//prints labels and first 10 lines
for (k = 0; k < 11 ; ++k) {
    printf("%s\n",line[k]);
}
```

- Followed by a "…" line.

```
printf(".....................................................................
.................\n");
```

- The variable *row_count* is then decremented to be able to get the last row of the csv file. After, the last line is printed. The file is then closed.

```
//Decrementing row_count to get last row
row_count--;

//printing the last line
printf("%s\n",line[row_count]);

//closes file
fclose(fdpo);
```

- **projectDT()**

- This function returns a copy of a DataTable t but containing only the columns from m to n, and rows from x to y, where m, n, x and y are integer parameters corresponding to valid column/row indices starting from 0. In this case, given the simplified version of DataTable_t , column-based projections were ignored. Firstly, the function will open the *health.csv* file in read mode. If the file *health.csv* cannot be open, an error message will be printed out and the program will stop executing (*return 1)*. Then, the function will use a for loop to output the lines from 5(*x*) till 10(*y*). The file is then closed.

```
int x = 5;
int y = 10;

//prints lines 5 to 10
for (int i = x; i < y ; ++i) {
    printf("%s\n",line[i]);
}
```

- **mutateDT()**

- This function transforms each floating-point or string value of a specified DataTable_t column. This function can transform the csv file in three different ways. The function is capable of sorting the csv file by the *BMI* from the smallest to the largest. The function also can sort the csv file alphabetically by first name and by surname separately.

```
void sortByBMI(DataTable_T sort_by_bmi[], int row_count) {
    int i, j;
    DataTable_T swap;
    for (i = 1; i < row_count - 1; i++) {
        for (j = i + 1; j < row_count; j++) {
            if (atof(sort_by_bmi[i].BMI) > atof(sort_by_bmi[j].BMI)) {
                swap = sort_by_bmi[i];
                sort_by_bmi[i] = sort_by_bmi[j];
                sort_by_bmi[j] = swap;
            }
        }
    }

    for(int c = 0; c < row_count; c++) {
        printf("%s,%s,%s,%s,%s,%s", sort_by_bmi[c].Height,
sort_by_bmi[c].Weight, sort_by_bmi[c].BMI,
               sort_by_bmi[c].Name, sort_by_bmi[c].Surname,
sort_by_bmi[c].Gender);
    }
}
```

- This function accepts a datatable and row count as parameters. It contains two loops, nested loop that will loop through each row of the datatable in order to sort using BMI. The outer loop stars from index 1 to skip the header row. The inner loop will do the same thing, but it will be starting from the new row of the datatable which means that the outer loop starts from index 1 and the inner loop starts from index 2. The inner loop will compare the BMI of the datatable with both loops index and if the BMI of the record from the outer loop is greater, this will be swapped which will then ultimately be displayed.

```
void sortByFirstName(DataTable_T sort_by_bmi[], int row_count)

void sortByLastName(DataTable_T sort_by_bmi[], int row_count)
```

- This is repeated for the functions *sortByFirstName* and *sortByLastName* but instead of numbers, it changes the characters.

```
memcpy(&sort_by_bmi, &DT, sizeof DT);
```

- Copies a value from memory to the destination. In this case from DT to sort_by_BMI.

(b) Compile Extend the implementation in 2(a) so that this time there are no limitations on the column number/types or maximum string-length/row-count.

Header file

- This includes all the function prototypes that are used in testing in question (2b)

- **fail() function**

- This function runs when memory cannot be allocated. It will output an error message and will also stop the execution of the program.

```
void fail(void) {
    perror("Memory could not be allocated!\n");
    printf("-------------------------------------------------------------------
-------------------------------\n");
    exit(EXIT_FAILURE);}
```

- For question 2(b) memory was allocated for every array to allow for no limitations on the number of columns and the number of rows in the csv file.

- **initDT_b() and deinitDT_b()**

- Both these functions were kept the same as the previous question(2a) since they already allocated and deallocated the memory.

- **loadDT_b()**

- This function works the same as in the previous question(2a). The only difference being that memory was allocated to the array *DT* and the array *buffer.* The function malloc was used as shown below:

```
// Allocating memory
DataTable_T *DT = (DataTable_T*) malloc(100 * sizeof(DataTable_T));
if (DT == NULL) fail();

// Allocating memory
DataTable_T *buffer = (DataTable_T*) malloc(200 * sizeof(DataTable_T));
if (buffer == NULL) fail();
```

- If memory cannot be allocated, the function fail() will execute.

- At the end of the function, the memory for *DT* and *buffer* was freed.

```
// Free the memory
free(DT);
free(buffer);
```

- **exportDT_b()**

- This function works the same as in the previous question(2a). The function malloc was used again to allocate memory for the array *DT*. If memory cannot be allocated, the function fail() will execute.

```
DT = (DataTable_T*) malloc(100 * sizeof(DataTable_T));
if (DT == NULL) fail();
```

- The contents of the Data initialized DataTable_t are exported to the newly created "export_2.csv". At the end of the function, the memory for *DT* was freed.

- **showDT_b()**

- This function works the same as in the previous question(2a). *It is to be noted that difficulties were encountered when trying to allocate memory for this function.*

- **projectDT_b()**

- This function works the same as in the previous question(2a). The function malloc was used again to allocate memory for the array *line*. This function will output lines 5 to 10 from the csv file "*health.csv"*.

- **mutateDT_b()**

- This function works the same as in the previous question(2a).

(c) Compile the full implementation as a shared library and provide a proper Abstract Data Type interface. The test driver this time should link with the shared library.

****In this case, the library was compiled out of the simpler version developed in 2(a)****

What is a shared library?

- A shared library is a file containing object code that several a. out files may use simultaneously while executing. When a program is link edited with a shared library, the library code that defines the program's external references is not copied into the program's object file. This Means that both the executable and libraries must reside on the machine at runtime.

```
set(SOURCE_FILES2 list.c)
add_library(list SHARED ${SOURCE_FILES1})

set(SOURCE_FILES3 Question_2c/question2_c.c)
add_executable(Question2_c ${SOURCE_FILES3})
target_link_libraries(Question2_c list)
```

- The library is compiled from the functions in question (2a). It is then linked to question (2c) to be able to execute using the library.