M.Sc. in High-Performance Computing
5613. C Programming
Assignment 4

R. Morrin <rmorrin@maths.tcd.ie>

November 19, 2019

Instructions

- Gather all your code and pdf/text files into a single tar-ball.
- Submit this tar-ball via `msc.tchpc.tcd.ie` before 23:59 Friday 29th Nov.
- Any questions, please ask me in class or email me at rmorrin@maths.tcd.ie. Don't wait until right before the deadline!
- Late submissions without prior arrangement or a valid explanation will result in reduced marks.
- Read notes at end of this document.

Q1. Linked Lists (100%)

You will write a simple program that creates and manipulates Linked Lists. Your program will accept command line arguments. The program will read data from a file and use that data to create and populate a linked list. It will then filter the list by removing nodes from the list depending on the size of the data in the node, and parameters which you pass in.

The main function that you should use is provided here in the document and also as a file to download from the M.Sc. website. Consider the main function as a final state to aim for - you don't need to wait until all the functions are written for you to start testing individual functions! For example, you could comment out parts of the given main function so that you can test as you proceed.

(A) Makefile

Write a `Makefile` to take care of your assignment. You will write it as you complete the different parts of the assignment. Your final `Makefile` should have four targets

1. assignment4
2. linked_list.o
3. parse.o
4. matrix.o

Set up your Makefile so that the project builds once you run the command `make` in the directory in which it is located. Include all relevant dependencies. (You can assume that library files such as `stdio.h` are fixed and ignore them as a dependency). If you modify any of your input files, then the necessary object files and executable should recompile, but only those necessary.

Note that character to denote a comment in a `Makefile` is `#`.

(B) Reading from command line

Download the file parse.h from the M.Sc website. This file contains a structure `CLOptions` and a prototype for a function. Create a file called parse.c and write the function defintion for this function.

```
void parse_command_line(const int argc, char * const *argv, CLOptions *opts)
```

The purpose of this function is to read and parse command line arguments and store them in the `CLOptions` structure. You can either write it entirely manually or else use a library function such as `getopt`. The function should be able to read in and handle the following command line options:

- `-i <inputfilename>`
- `-x <max_number>`
- `-n <min_number>`
- `-h`

Each argument should be optional in the sense that your program should be able to be run without it being specified explicitly on the command line. Figure out a way to set default values in the case that they are not passed in. The default values can be `data.txt`, `0`, `10000` for <`inputfilename`>, <`min_number`> and <`max_number`> respectively. The final argument, `-h`, should simply print a usage statement:

```
Usage: ./assignment4 [-i input_filename ] [-n min] [-x max] [-h]
```

If an argument is passed which is not listed (e.g. `-z`), or if an incorrect parameter is passed (e.g. `-x a`) the program should exit and print the usage statement. Try to test for possible errors wherever you can.

(C) Matrix functions

Download the file matrix.h which includes two function prototypes for dealing with matrices. Create a file matrix.c and in it, write definitions for these two functions:

Function 1

```
int alloc_cont_mem_sq_mat(double *** A, const int N)
```

This first function will assign contiguous memory for an $N \times N$ square matrix. Two malloc calls should be used - one to allocate a vector of pointers and the second allocates a single segment of memory to hold all the data points. Refer to the slides from class where we discussed the various options for allocating 2D arrays. You should be able to access the data using the `[][]` notation. The address of the vector of pointers is stored in the value pointed to by `A` i.e. stored at `*A`.

Function 2

```
void print_sq_matrix(double *const *const B, const int dim)
```

This function will print an $dim \times dim$ square matrix (B) to screen.

(D) Linked List

Download the file linked_list.h. This file contains two structures typedef'd as `Node` and `List`. It also contains 4 function prototypes. You will write function definitions for these four functions in a file linked_list.c.

Prototypes

structures

```
typedef struct _Node {
    struct _Node *next; /*!< pointer to next node in list        */
    struct _Node *prev; /*!< pointer to previous node in list    */
    int N;              /*!< dimension of the square matrix      */
    double **data;      /*!< matrix data                         */
} Node;

typedef struct _List {
    Node *head;         /*!< pointer to first node in list       */
    Node *tail;         /*!< pointer to last node in list        */
} List;
```

The `List` struture stores pointers to the first and last nodes in a linked list. An empty list is identified by having its `head` pointer set to `NULL`. The `Node` struture represents a node in a linked list which stores information related to a 2D array. `prev` and `next` will point to `NULL` for the first and last nodes respectively.

Function 1

```
int append_new_node(List *const inlist, const int N);
```

This function takes in a pointer to a `List` structure and an integer N. The function should allocate memory for a `Node` structure to hold an $N \times N$ matrix and append this node to the end of the existing linked list, updating `inlist` and appropriate list elements as necessary. You will need to take care of the case when the existing list is empty.

Function 2

```
void read_list_from_file(const char *const filename, List *mylist);
```

This function will read data from a file and store it in a linked list. See the included file `data.txt`. The structure of the file is that it contains lines with a tab-delimited numbers. The first number each line is an integer N. There are then N^2 remaining doubles in the line which are the elements of an $N \times N$

matrix. For example `2 6.2 8.0 9.2 9.3` for a 2×2 matrix. You may parse each line however you wish but strtol and strtof are suggested. For each line in the file, call the function `append_new_node()` you wrote above to add a new node to the exisiting list and then populate the `data` member in that `Node` using the data in the line.

Function 3

```
void remove_node(List *in, Node *node);
```

This function will take a pointer to a `List` structure and also a pointer to a `Node`. The function should remove this node from the list. Make sure to correctly free up any allocated memory. Make sure that your code can handle the cases where the node to be removed is at either end of the list, or is the only exisiting node in the list.

Function 4

```
void print_list_contents(List list);
```

This function will take a list structure as a parameter and will iterate through the nodes of the list, printing the data (matrix) in each using the `print_sq_matrix` function that you wrote earlier.

(E) Main function. Putting it all together.

Use the code below as your main function in assignment4.c. You can also download it from the M.Sc website.

assignment4.c

```
/**
 * \file assignment4.c
 * \brief Main function for HPC Assignment 4
 *       The program will read data from a file, create and populate a linked
 *       list to hold this data. It will print the list to screen. It will
 *       then remove certain nodes from the list depending on specified
 *       criteria and print the final filtered list.
 * \author R. Morrin
 * \version 1.0
 * \date 2019-11-19
 */
#include "linked_list.h"
#include "parse.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    List mylist;
    mylist.head = mylist.tail = NULL;    /* Null indicates empty list */
    CLOptions opts;

    /* Read command line parameters and store in opts            */
    parse_command_line(argc, argv, &opts);

    /* Read data from file. Create list and populate nodes       */
    read_list_from_file(opts.infilename, &mylist);

    /* Print contents of initial linked list                     */
    printf("\n----------\nOriginal Data\n----------\n");
    print_list_contents(mylist);

    /* Filtering. Remove nodes that are not within specified sizes*/
    Node *it=mylist.head;
    while(it!=NULL){
        Node *nextit = it->next;
        if(it->N < opts.min_size || it->N > opts.max_size){
            remove_node(&mylist, it);
        }
        it=nextit;
    }

    /* Print contents of filtered linked list                    */
    printf("\n----------\nAfter Filtering\n----------\n");
    print_list_contents(mylist);

    return 0;
}
```

Put everything together from all the parts that you have written above. Finally make sure that it compiles and runs as expected. You can check the output for evidence of any problems. Run for different combinations of `max` and `min` numbers to check it behaves as expected.

Example Output

Example Output

```
$./assignment4  -i data.txt -n 2 -x 5
---------
Original Data
---------
| 7.3   6.0     5.5     0.0     4.2     7.8     0.6 |
| 7.6   6.9     6.3     5.5     8.4     3.7     0.6 |
| 9.0   8.9     0.3     4.5     8.8     2.1     6.4 |
| 1.4   0.4     3.1     3.1     6.9     3.4     2.6 |
| 0.4   9.7     9.8     1.6     3.9     5.8     6.8 |
| 1.2   1.4     5.2     7.0     1.4     4.9     3.4 |
| 5.7   2.3     5.3     9.3     5.8     2.4     3.9 |

| 6.2   8.0 |
| 9.2   9.3 |

| 6.1   6.0     1.8     5.8     6.4     7.1     6.8     4.8     7.3 |
| 4.7   2.6     2.7     7.4     1.3     5.8     1.8     4.3     6.1 |
| 8.6   2.2     9.3     8.5     3.3     9.5     1.0     0.0     2.0 |
| 6.0   7.6     0.4     6.5     2.5     6.8     0.5     6.1     5.4 |
| 8.9   1.3     5.6     4.9     6.9     3.0     0.5     5.0     5.3 |
| 2.3   5.8     0.2     6.7     3.2     5.3     0.1     5.0     1.1 |
| 6.3   9.7     6.6     6.5     4.8     8.6     8.0     6.0     5.6 |
| 0.2   4.9     8.1     5.8     0.9     2.1     0.8     8.1     4.2 |
| 6.3   7.8     1.0     5.1     4.0     2.4     2.1     5.4     3.3 |

| 8.7 |

| 8.6   3.0     7.4     5.4     5.1 |
| 5.4   7.2     2.5     6.0     4.5 |
| 8.8   5.1     7.9     8.8     4.7 |
| 7.1   9.6     5.4     6.2     5.0 |
| 4.4   5.0     8.3     4.2     6.2 |

| 8.1   7.0 |
| 8.1   3.7 |

| 8.4   0.5     5.2     5.3     9.1     1.5     8.4     7.8     0.7 |
| 1.2   9.6     3.0     2.5     0.1     2.2     1.8     7.4     5.0 |
| 0.2   9.6     6.3     0.0     7.9     3.1     5.9     0.5     7.1 |
| 4.9   6.2     3.8     6.6     8.4     6.1     7.4     9.0     5.8 |
| 1.3   5.1     3.3     4.6     1.7     3.1     4.3     4.0     2.5 |
| 8.2   1.3     4.9     1.8     9.8     2.3     8.2     4.7     1.9 |
| 1.1   9.7     7.6     8.9     5.9     4.3     6.6     9.3     3.2 |
| 7.6   8.1     8.7     1.9     5.9     9.3     9.8     6.5     7.6 |
| 1.3   5.9     1.9     3.7     8.5     8.2     4.1     6.3     6.9 |

| 6.2   4.7     2.3     5.9     3.5 |
| 0.6   3.8     5.0     5.7     7.3 |
| 1.3   2.0     7.2     6.6     0.9 |
| 0.5   2.8     0.3     5.4     5.0 |
| 7.9   2.2     9.5     6.8     9.4 |

| 6.8   8.2     7.5     4.5     3.1     4.3     1.4 |
| 1.8   7.6     5.2     4.9     4.3     8.2     9.1 |
| 3.6   4.4     5.7     9.0     5.8     4.5     9.0 |
| 2.2   3.9     6.1     0.5     1.1     0.4     7.3 |
| 7.9   3.0     7.0     1.2     1.2     4.0     8.8 |
| 8.8   1.9     8.1     8.8     9.4     2.4     8.2 |
| 1.4   1.1     7.3     2.8     6.8     1.4     6.5 |

| 1.6 |


---------
After Filtering
---------
| 6.2   8.0 |
| 9.2   9.3 |

| 8.6   3.0     7.4     5.4     5.1 |
| 5.4   7.2     2.5     6.0     4.5 |
| 8.8   5.1     7.9     8.8     4.7 |
| 7.1   9.6     5.4     6.2     5.0 |
| 4.4   5.0     8.3     4.2     6.2 |

| 8.1   7.0 |
| 8.1   3.7 |

| 6.2   4.7     2.3     5.9     3.5 |
| 0.6   3.8     5.0     5.7     7.3 |
| 1.3   2.0     7.2     6.6     0.9 |
| 0.5   2.8     0.3     5.4     5.0 |
| 7.9   2.2     9.5     6.8     9.4 |
$
```

Notes:

- Always try to avoid using global variables. It is a good idea to try to restrict the scope of any variable to the minimum necessary. Don't use global variables in order to eliminate having to pass variables or arrays in or out of functions.
- Where applicable, 10% of the marks for each part of a question will be allocated for error checking.
- 20% of the respective marks available for each of the coding questions will be given for comments. You do not need to go overboard and please make sure the comments are relevant. i.e. you don't need to say obvious things like `int A; /*This is an integer */`. Note that in the lecture slides, I sometimes include comments such as `/* Function Prototype */` - but this is just for the purposes of the lectures. This would not necessarily be a helpful comment on its own in a proper program. Strive to make your code as self-documenting as possible by using appropriate names for variables and functions.

  ⇒ For this assignment, you don't have to use Doxygen comments. But you can use them if you want. Especially if you think that the headers/commands used give a good template for comments. But I won't assign specific marks to this assignment for it. You can get full marks with or without doxygen comments.
- Continue to use const when you can. It could help you to catch bugs. I will reserve a few marks for its use when appropriate.
- To create your tarball, `cp` everything you want to submit into a directory e.g. `Assignment4`, and then change directory so that you are in the parent directory of `Assignment4`. Then do
- Take care to free up dynamically allocated memory as soon as it is no longer needed. Note that, for the purposes of this assignment, you do not need to explicitly free up the memory associated with the remaining nodes at the end of the program . . . although it is generally good practice to make sure that each malloc has a corresponding free.

  `tar -cvf [username].assignment4.tar Assignment4`

  where you replace `[username]` with your own username.