CIS*2500 W20 - A4 Marks Breakdown

Summary of Mark breakdown per section

Q1a **[50 pts]**
Q1b **[40 pts]**
Q2 **[25 pts]**
Q3 **[25 pts]**

General Marks awarded

[1 pts] Functions should be callable by other programs (*i.e. not in same file as main()*)
[2 pts] Consistent Indentation
[1 pts] Good variable names
[1 pts] Good use of white space
[1 pts] Use of #defines for constants
[1 pts] Proper use of headers
[2 pts] Commenting
[1 pts] Read.me

Total: [150 pts]

1. Your code must compile cleanly with no error or warning messages using the -Wall flags in gcc.
2. The assignment must be written in C and run on the School's Linux server.
3. Your source code should contain brief comments describing the functionality and the major components of each procedure. Any complex structures should also be commented. Your source code should be properly formatted, and meaningful variable names should be used.
4. A readme file should be submitted along with your source code explaining how your program should be run as well as any limitations of the code.
5. **If you hand in an assignment that does not compile you will get a zero grade.**
6. All work in this course is to be done independently.
   Submissions will be electronically examined for similarity.
7. You should hand in your source code file (.c and .h files) and a makefile in a single 'zipped' file.
   If no working makefile is supplied, you will be given a zero grade.

Next pages gives detailed breakdown for each question
Can also be used as a checklist

[*bonus 10%] for Q1 – if doubly linked lists* used

Question 1a: Advanced Linked Lists

[2 pts] Sorted List structure definition

Functions to be implemented

- **[1 pt]** `int size (Sorted_List *)`
- **[3 pts]** `int push ( Sorted_List *, value_t , key_t )`
- **[3 pts]** `int append ( Sorted_List * , value_t , key_t )`
- **[3 pts]** `int remove_first ( Sorted_List * , value_t * , key_t *)`
- **[3 pts]** `int remove_last ( Sorted_List * , value_t * , key_t * )`
- **[3 pts]** `int remove_smallest_key ( Sorted_List * , value_t * , key_t * )`
- **[3 pts]** `int remove_largest_key ( Sorted_List * , value_t * , key_t * )`
- **[3 pts]** `void empty_list ( Sorted_List *)`
- **[1 pt]** `void destroy_list ( Sorted_List *)`

To test the Sorted List ADT

- **[2 pts]** Proper use of `value_t` and `key_t`
- **[2 pts]** two working executable programs called `a4q1a_char` and `a4q1a_int` that use the correct datatypes
- **[1 pt]** make file uses gcc lines with -DINT and DCHAR to compile files containing Sorted_List f^{n}s
- **[1 pt]** Proper use of #ifdef to perform condition compilation in the .c files
  - including files containing Sorted_List f^{n}s
- **[2 pts]** program uses a command line argument to enter the file name of the input text file containing the commands
- **[1 pt]** use of stdin if the file name is not provided as the command line argument
- **[2 pt]** program can read commands from the input file, process them and print the results

  important: *while only worth* 2pt, *if this doesn't function properly, proper testing of your program cannot be done, and you will only get half marks for the functions in the source code that look reasonable, but cannot be properly tested*
- **[2 pt]** freeing the sorted list at the end of the program

Implementing List of Commands

- **[2 pts]** a = append for `a4q1a_int.c`, a = append for `a4q1a_char.c`
- **[2 pts]** p = push for `a4q1a_int.c`, p = push for `a4q1a_char.c`
- **[2 pt]** `rem_first`, `rem_last`
- **[2 pt]** `rem_small`, `rem_large`
- **[2 pt]** `empty`, `size`
- **[2 pt]** `print_all` for `a4q1a_int.c`, `print_all` for `a4q1a_char.c`
- **[2 pt]** `print_sort` for `a4q1a_int.c`, `print_sort` for `a4q1a_char.c`

  important: *the commands must only use the ADT Sorted_List functions.*
  If any ADT implementation details are exposed and used directly in a command implementation, you will not receive any marks for that command

Question 1b: List ADT and Function Pointers

Functions to be implemented

- **[2 pt]** `Sorted_List * map ( Sorted_List *,` *fn ptr*`)`
- **[3 pts]** `value_t reduce ( Sorted_List *,` *reduce fn ptr*`, value_t, int)`
- **[3 pts]** `value_t map_reduce (Sorted_List *,` *map fn ptr*`,` *reduce fn ptr*`, value_t, int)`
- **[3 pts]** `value_t * map_2_array ( Sorted_List *, List_Sort *,` *fn ptr*`, int)`
- **[3 pts]** `value_t map_2_reduce( Sorted_List *, List_Sort *,` *map fptr, reduce fptr*`, int)`
- **[2 pts]** above functions (except map) `produces different results when traversing using the next vs sort links`

To test within, map, reduce, etc.

- **[1 pts]** working executable program called `a4q1b` that use the correct datatypes
- **[1 pt]** make file uses gcc lines with -DINT to compile files containing Sorted_List f^{n}s
- **[1 pt]** Proper use of #ifdef to perform condition compilation in the .c files
  - including files containing Sorted_List f^{n}s
- **[2 pts]** program uses a command line argument to enter the file name of the input text file containing the commands
- **[2 pt]** freeing all arrays and sorted lists at the end of the program

Functions to help implement the new commands

- **[1 pt]** used an array that can hold up to 10 Sorted_List pointers
- **[1 pt]** void print_array(value_t *, int size)
- **[1 pt]** used *map*, *reduce*, *map_reduce*, *map_2_array*, or *map_2_reduce* when implementing below
- **[4 pt]** implemented sum, square, diff, sum_of_sq_diff

Implementing List of Commands

- **[1 pt]** All commands from q1a are available

  important: *while only worth* 1pt, *if this doesn't function properly, proper testing of your program cannot be done, and you will only get half marks for the functions in the source code that look reasonable, but cannot be properly tested*
- **[1 pts]** all commands update the proper list array index *n* in the commands below
- **[2 pt]** `a|`*n*`,` `p|`*n*
- **[1 pt]** `print_all|`*n*, `print_sort|`*n*
- **[1 pt]** `sum|`*n*
- **[1 pt]** `square|`*n*
- **[1 pt]** `diff|`*n*`:`*m order*
  - You need to have *order* working correctly ow only 0.5pt will be awarded
- **[1 pt]** `sum_sq_d|`*n*`:`*m order*
  - You need to have *order* working correctly ow only 0.5pt will be awarded
- **[1 pt]** memory free'd from all new list/arrays produced by map/map_reduce/map_2_array/map_2_reduce

  important: *the commands must only use the ADT Sorted_List functions (including map etc.). If any ADT implementation details are exposed and used directly in a command implementation, you will not receive any marks for that command*

Question 2: Recursion

Recursive functions

- **[2 pts]** Count down from n to 0
- **[2 pts]** Count up from 0 to 2n by 2
- **[4 pts]** nth, nth_sorted
  - this applies to Sorted Lists from Q1a
- **[4 pts]** remove_nth, remove_nth_sorted
- **[1 pts]** long gcd(long, long)
- **[1 pt]** explanation of gcd implementation as a tail recursion in the readme
- **[1 pt**] make sure your make file uses the appropriate gcc flag to run tail recursive code efficiently

note: no marks will be awarded if a function was implemented iteratively, even if the answer produced is correct when run

To test question 2

- **[1 pts]** working executable program called a4q2 that use the correct datatypes
- **[1 pt]** make file uses gcc lines with -DINT to compile files containing Sorted_List f^{n}s
- **[1 pts]** program uses a command line argument to enter the file name of the input text file containing the commands
- **[2 pt]** freeing the sorted list at the end of the program

Implementing List of Commands

- **[1 pt]** All commands from q1a are available

important: *while only worth* 1pt, *if this doesn't function properly, proper testing of your program cannot be done, and you will only get half marks for the functions in the source code that look reasonable, but cannot be properly tested*

- **[1 pt]** `count_up` *n*
- **[1 pt]** `count_down` *n*
- **[1 pt]** `nth` *n order*
  - You need to have *order* working correctly ow only 0.5pt will be awarded
- **[1 pt]** remove_nth *n order*
  - You need to have *order* working correctly ow only 0.5pt will be awarded

Question 3: Interacting Abstract Data Types -

ADT – Fraction Functions

- **[1 pt]** `int set_fraction(Fraction * fract, int num, int denom)`
  - Only 0.5pts if check for 0 value denominators not performed
- **[2 pts]** `print_fract(Fraction * fract, int mode)`
  - in SIMPLE mode and MIXED mode
- **[1 pt]** `void simplify(Fraction * fract)`
- **[1 pt]** `int add_fract(Fraction * result, Fraction * x, Fraction * y)`
- **[1 pt]** Check to make sure that `add_fract` doesn't overflow/underflow

note: no marks will be awarded if a function uses function calls to any C library that implements fractions

Extend Map/Reduce/etc. with Filter

- **[4 pts]** `Sorted_List * filter (Sorted_List * list, filter_fn pointer)`
  - [-1] if the node pointer is copied instead of a new node with copied values
  - [-1] if the new list links are still pointing to the old list
  - [-1 pts] if not implemented using recursion

note: you cannot get a mark less then 0 for this question

To test question 3

- **[1 pts]** working executable program called a4q3 that use the correct datatypes
- **[1 pt]** make file uses gcc lines with -DFRACT to compile files containing Sorted_List f^{n}s
- **[1 pt]** Proper use of #ifdef to perform condition compilation in the .c files
  - including files containing Sorted_List f^{n}s
- **[1 pts]** program uses a command line argument to enter the file name of the input text file containing the commands
- [1 pt] program can read commands from the input file, process them and print the results

important: *while only worth* 1pt, *if this doesn't function properly, proper testing of your program cannot be done, and you will only get half marks for the functions in the source code that look reasonable, but cannot be properly tested*

- **[2 pt]** freeing the sorted list at the end of the program

List of Commands from the Input File

- **[2 pt]** `a` *n/d*, `p` *n/d*
- **[2 pt]** `print_all` *print_mode*, `print_sort` *print_mode*
- **[1 pt]** `sum` *print_mode*
  - [-0.5pts] if behaviour is incorrect If the sum enters an overflow situation
- **[3 pt]** `fract` *print_mode*, `whole_num` *print_mode*, `rem_mixed` *print_mode*

important: *the commands must only use the Sorted_List and Fraction ADT functions (including map etc.) If any ADT implementation details are exposed and used directly in a command implementation, you will not receive any marks for that command*