CIS*2500 W20 - Assignment 4

Linked Lists, Recursion and ADTs

Question 1a: Advanced Linked Lists

```
typedef struct NODE {
    value_type value;
    key_type key;
    struct NODE * next;
    struct NODE * sort;
} Node;
```

or

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

```
typedef struct NODE {
    value_type value;
    key_type key;
    struct NODE * next;
    struct NODE * sort;
    struct NODE * prev;
    struct NODE * prev_sorted;
} Node;
```

In this linked list:

- The datatype for the value being stored is called value_type
- The datatype for the key being stored is called key_type
- As in lab 4, `next` links to the node in the order it was added to the list (either at the head or the tail)
    - This will be referred to as *insertion order*
- Similar to lab 4, `sort` links to the node where the key is greater or equal to its key
    - i.e. the list is kept in ascending order by key
    - This will be referred to as *key sort order*
    - Note: unlike lab 4, there is only one key

Create a Sorted List abstract data type

- Has two heads (head *for insertion order*, `head_sort` *for key sort order*)
- Has two tails (`tail` *for insertion order*, `tail_sort` *for key sort order*)
- Has an `int` field called size that stored the node count (the number of elements in the list)
- The datatype should be called Sorted_List

  Note: technically you will be implementing only be a subset of the Sorted List ADT as you will not be asked to implement all functions of the full ADT

Functions to be implemented

All functions, except where noted, return SUCCESS if the function can complete or FAIL if not

- `int size (Sorted_List *)`
  - returns the number of nodes in the list (not SUCCESS/FAIL as the function cannot fail)

- `int push ( Sorted_List *, value_type , key_type )`
  - add the node to the head of the list
  - the node must also be inserted in **ascending** sort order by key, using the sort link

- `int append ( Sorted_List * , value_type , key_type )`
  - similar to push, except the node gets added to `tail`

- `int remove_first ( Sorted_List * , value_type * , key_type  *)`
  - removes the node from the head of the list
  - returns the value and key of the removed node through the parameter values (and frees the node)
  - returns SUCCESS (alternatively you can change the signature to return void)
  - remember to update the sort order links
    - if not using doubly linked lists, you will need to find the previous sorted node to change its sort order link

- `int remove_last ( Sorted_List * , value_type * , key_type * )`
  - similar to remove_first, except it removes the node from the tail

- `int remove_smallest_key ( Sorted_List * , value_type * , key_type * )`
  - removes the node with the smallest key
  - returns the value and key of the removed node (and frees the node)
  - remember to update the insertion order links
    - if not using doubly linked lists, you will need to find the previous insert order node to change its insertion order link

- `int remove_largest_key ( Sorted_List * , value_type * , key_type * )`
  - similar to remove_smallest_key, except it removes the node with the largest key

- `void empty_list ( Sorted_List *)`
  - empties the contents of the list
  - remember to free the memory of the contents

- `void destroy_list ( Sorted_List *)`
  - empties the contents of the list, as well as freeing the list itself

To test the Sorted List ADT

Write two programs called `a4q1a_char.c` *and* `a4q1a_int.c`

- `Data types used`
  - `a4q1a_int.c`
    - has its `value_type` datatype set equal to `int`
    - has it `key_type` datatype set equal to `double`
  - `a4q1a_char.c`
    - has its `value_type` datatype set equal to `char[80]`
      - i.e. it can take strings up to 79 characters in length
    - has its `key_type` datatype set equal to `int`
      - its value is set equal to the length of the string
- Both programs read in a text file that contains a series of commands, one per line (i.e each ending with a newline)
  - The name of the text file should be entered as a command line argument
    - If there is no file name, read from `stdin`
      - this can use IO redirect, i.e. `a4q1a_int < filename.txt`
    - If using keyboard input, exit using ^d
- All commands are echoed to `stdout`, followed by a colon :,
  - After that the results of the command follows,
    - usually on the same line following 11 – strlen(cmd name) spaces or on the next line when noted

  Note: Silent commands do not have the colon : after the command, but rather after the command name
- Remember to free the sorted list at the end of the program (use `destroy_list`)

General Note: The two programs should be almost identical, with the following differences
  - The file input will be slightly different depending on the data type and nature of the input data
  - Your will have to write similar, but not identical `void print_list_all ( Sorted_List * )` and `void print_list_sort ( Sorted_List * )` functions
    - These functions print out the lists according to their respective sort orders
    - See the report commands section below for details (the `print_all` and `print_sort` commands)
  - You will have to have your make file recompile all files that mention or use value_type and key_type variables or Sort_List structs when compiling the two programs
    - To do this you will need to use condition compilation (see Week1 lecture notes)
    - In specific, use #ifdef CHAR to compile using the char[80] typedef definition of value_type
      and #ifdef INT to compile using the int typedef definition of value_type
    - E.g. if you stored all your Sort_List ADT functions in a single file called sort_list.c
      Then for a4q1a_char.c you could have in your make file a command like

      ```
      gcc  -Wall -ansi -DCHAR -c sort_list.c
      ```

List of Commands

Silent Commands (modifies the list but does not print anything other than the command itself)

- a = append
  - `a4q1a_int.c`
    - input line: `a    key value`

      note: there can be any number of spaces in the input been the command and args, or between args
    - example
      - *commands, as stored in the input file*
        ```
        a   3.27   1427
        a   0.94   984
        a   7.21   346
        ```
    - *output* (11 – 1 spaces after the colon)
      ```
      a:          3.27  1427
      a:          0.94  984
      a:          7.21  346
      ```
  - `a4q1a_char.c`
    - input line: `a    value`
    - example
      - *commands, as stored in the input file*
        ```
        a   The sun did not shine.
        a   It was too wet to play.
        a   So we sat in the house
        a   All that cold, cold, wet day.
        ```
        Note: skip the white space between the command 'a' and the input string
      - The key values for the above are 22, 23, 22, 29
        - e.g. `strlen("The sun did not shine.") == 22`
      - *output* (11 – 1 spaces after the colon)
        ```
        a:          The sun did not shine.
        a:          It was too wet to play.
        a:          So we sat in the house
        a:          All that cold, cold, wet day.
        ```
- p = push
  - same as a except it pushes instead of appends the key/value pair onto the list

Verbose Commands (modifies the list and then reports to stdout)

- `rem_first` = remove first node of the list by insertion order
  - also prints the element's key-value pair,
    with two spaces between the key and the value
  - Example for `a4q1a_int.c` assuming the first list element key is 2.465 and value is 212
    ```
    rem_first:  2.465  212
    ```
  - Note the two spaces after rem_first:
    - "rem_first" is 9 characters in length,
      so the number of spaces following should be 11 – 9 = 2
- `rem_last` = remove last node of the list by insertion order and print the element's key-value pair
- `rem_small` = remove the node with the smallest key and print the element's key-value pair
- `rem_large` = remove the node with the largest key and print the element's key-value pair
- `empty` = empty the list
  - the output of this command should be
    ```
    empty:      size = 0
    ```

Report Commands (prints information, but does not modify the list)

- `size` = size of sorted linked list
  - if there are 21 nodes in the list it prints
    size: List size = 21
- `print_all` = print list in insertion order
  - The type of order is printed on the same line as the command
  - The list starts printing on the next line, one element per line
  - Each element is prefaced by 5 spaces, then the key, then 2 spaces, then the value
  - Example using the input from the append examples
    - a4q1a_int.c

```
print_all:  Insertion Order
     3.27  1427
     0.94  984
     7.21  346
```

    - a4q1a_char.c

```
print_all:  Insertion Order
     22  The sun did not shine.
     23  It was too wet to play.
     22  So we sat in the house
     29  All that cold, cold, wet day.
```

- `print_sort` = print list in key sort order
  - The output is the same as with print_all except the order of the lines are in key sort order and the command line will read Key Sort Order
  - Example using the input from the append examples
    - a4q1a_int.c

```
print_sort: Key Sort Order
     0.94  984
     3.27  1427
     7.21  346
```

    - a4q1a_char.c

```
print_sort: Key Sort Order
     22  The sun did not shine.
     22  So we sat in the house
     23  It was too wet to play.
     29  All that cold, cold, wet day.
```

The assignment continues with Question 1b Function Pointers
to be released by March 26

*the relevant lecture notes for Q1b, presented the last week
of face-to-face classes, have now been posted*