Project 2

Project 2

- Implement a set ADT for strings
    - Unsorted and sorted
    - Use arrays
- Interface and implementation must be separate
- Download project2.tar for header file and test files
    - set.h - header file that contains the interface
    - parity.c & unique.c - test files that will call your set ADT interface

Interface

- `SET *createSet(int maxElts);`
  return a pointer to a new set with a maximum capacity of *maxElts*
- `void destroySet(SET *sp);`
  deallocate memory associated with the set pointed to by *sp*
- `int numElements(SET *sp);`
  return the number of elements in the set pointed to by *sp*
- `void addElement(SET *sp, char *elt);`
  add *elt* to the set pointed to by *sp*
- `void removeElement(SET *sp, char *elt);`
  remove *elt* from the set pointed to by *sp*
- `char *findElement(SET *sp, char *elt);`
  if *elt* is present in the set pointed to by *sp* then return the matching element, otherwise return `NULL`
- `char **getElements(SET *sp);`
  allocate and return an array of elements in the set pointed to by *sp*

Unsorted.c

- Use an unsorted array of length m > 0
  - Keep track of length
- The first n <= m slots are used to hold n strings in arbitrary order
  - Keep track of count
- Use sequential search to locate an element in the array
  - Implement separate search function so you can call from other functions (add, remove, find)
  - Need to search if element exists when adding because sets do not allow duplicates
- Don’t need to shift when deleting because unsorted - can simply move last element to the index of the deleted element

Sorted.c

- Use a sorted array of length m > 0
  - Keep track of length
- The first n <= m slots are used to hold n strings in ascending order
  - Keep track of count
- Use binary search to locate an element in the array
  - Implement separate search function so you can call from other functions (add, remove, find)
  - Need to search if element exists when adding because sets do not allow duplicates
- Need to shift elements when adding or deleting because must maintain order

Things to Remember

- Cannot compare strings with ==, must use **strcmp()**
  - **int strcmp(const char *str1, const char *str2)**
- Allocate memory with **malloc()**
  - **void *malloc(size_t size)**
- Call **assert()** where necessary - when dealing with allocated memory
- Don’t forget to free all of the space you allocate
- Use **strdup()** to duplicate a string
  - **char *strdup(const char *s)**
- Use **memcpy()** to copy memory
  - **void *memcpy(void *dest, const void * src, size_t n)**
- Don’t forget comments!

Compiling and Testing

- Verify that your set ADTs work with both unique.c and parity.c
- **Unique.c** - takes one or two files and inserts the words in the first into a set and then removes the words in the second file from the set
- **Parity.c** - takes a file and uses a set to maintain a collection of words that occur an odd number of times
- Compile:
  - `gcc -o unique unique.c unsorted.c`
- Run:
  - `time ./unique /scratch/coen12/Macbeth.txt`
  - `time ./unique /scratch/coen12/Macbeth.txt /scratch/coen12/Bible.txt`

Submission Details

- Due Sunday, April 18th at 11:59 pm
- Demo by end of your lab section the following week
- Include average real time and algorithmic complexity of each file
- Submit one tar file that contains:
    - unsorted.c
    - sorted.c
    - report.txt
    - project2 directory downloaded from Camino
- How to tar a file:
    - tar -czvf project2.tar project2