P0:File Sorter

Daniel Doan, dd891
Graham Duebner, gd
Systems Programming, Spring 2020

Synopsis:

FileSort contains two files:

- fileSort.c
- fileSort.h

fileSort.h contains all the definitions and structures used in fileSort.c.

FileSort takes in a file that consists of either only integer(no degenerate dashes) or only lowercase ASCII character tokens, separated by a comma. Tabs, dashes, and new line characters are allowed in the file, but are not to be considered parts of the token and thus must be removed in the final output. The program then sorts the file and outputs a list of the tokens in ascending order.

Two examples of properly formatted command line arguments for insertion sort and quick sort respectively would be:

1. ./a.out -i <file path>
2. ./a.out -q <file path>

Design/Methodology:

**Indicates a function call, please check Function/Structure section to learn more about them*
Before starting the sorting process, the program checks the commands entered in the terminal and ensures two things through *errorCheck:

1. There are at least two commands
2. Both commands are properly formatted

If either of these checks fail, the program will print an error message and self terminate.
If a file is not found, an error message will also be printed and the program will self terminate.

On file found program will take file into *readFile where it will read in all bytes of the file, parse them by commas, and then add them to a linked list to which it will return the head of. Afterwards, head is then fed into *setFileType, which will determine if the file consists of integers or only characters. Then, head is fed into *makeArray, where an array is made out of the linked list. The array is then sorted and printed out by *sortFile by the method specified in the command line when the program is first initialized.

Functions/Structures:

Structures(s):

```
typedef struct NODE{
  char* str;
  struct NODE* next;
  int delimTerm;
}node;
```

Node structure used for storing each token in file.

void sortFile(char command, void array, int fileType);*

Function in main that takes in a char command, void* array,and int fileType, and makes appropriate function calls to either *insertionSort* or *quickSort* depending on command and then frees array on completion before returning to main.

*int errorCheck(int argc, char** argv);*

Function in main that takes in an int argc and char** argv and checks if argc is equal to 3 or if the command given in argv is a valid flag. Valid flags consist of “-i” or “-q”. If errorCheck finds that flags are NOT valid, then the error message “`Fatal Error: Please enter two properly formatted commands”` will print and program will self terminate. Likewise, if argc is not equal to 3, then the same procedure as before will occur. On success the program will return 0.

void printArray(void array, int ft);*

Function in that takes in a void* *array* and an int *ft* found in *insertionSort* and quickSort that prints each element/token of array separated by a newline character after sorting is finished and returns.

int setFileType(node head);*

Function in main that takes in a node* *head* that checks if the string contained in *head* is either an integer or character string and modifies a global variable, fileType. If string is empty, next node is checked, and the process is repeated until a non empty string is found. If string is an integer string, a 0 is returned, and if it is a character string, a 1 is returned. -1 is returned on errors.

void makeArray(node* head);*

Function in main that takes in a node* *head* and mallocs and creates either a char** or int**(Dependent on fileType global) array of size equal to the global variable arraySize, and then populates the array with linked list elements before returning the array.

int compareChar(void TOKENA, void* TOKENB);*

Function that can be used in either *insertionSort* or *quickSort* that takes in a void* *TOKENA* and void* *TOKENA* and compares them to determine which one is greater. The function casts both tokens into char* types. Returns 1 if *TOKENA* > *TOKENB*, returns 0 if *TOKENA* == *TOKENB*, and returns -1 if *TOKENA* < *TOKENB*. This comparison function works only for void casted char*.

int compareInt(void TOKENA, void* TOKENB);*

Function that can be used in either *insertionSort* or *quickSort* that takes in a void* *TOKENA* and void* *TOKENA* and compares them to determine which one is greater. The function casts both tokens into long types(reasoning for long is that the size of long data type is same as int* on iLab machines, where as casting to int gives a warning message on compile). Returns 1 if *TOKENA* > *TOKENB*, returns 0 if *TOKENA* == *TOKENB*, and returns -1 if *TOKENA* < *TOKENB*. This comparison function works only for void casted ints.

char cleanStr(char* str);*

Function found in parseString that takes in a char* *str* and removes all tabs, spaces, and newline characters from *str* before returning the modified *str*.

void printLL(node head);*

Testing function that takes in a node* *head* that was made to print out the created linked list. Not used in the program explicitly but could have debugging purposes.

node parseString(char* str, char delim, node* head);*

Function found in *readFile* that takes in a char* *str,* char *delim,*and node* *head* and parses the string by the given delimiter character *delim* and creates tokens out given string and adds them to the linked list. If the token is broken, (i.e. string does NOT end with a delimiter and the next string given to function does NOT begin with a delimiter), the next string's characters, up to the delimiter, are added to the previous node. On completion function will return *head*.

node readFile(int arguments, char* fileName);*

Function found in main that takes in an int *arguments* and char* *fileName*, and reads bytes from *fileName* and calls *parseString* within to parse strings read in and to add them to a linked list. On success function will return *head*.

Potential Errors/Warnings:
On file not found, error message `"Fatal Error: File %s does not exist."` before program self terminates.
On failed malloc `"Warning: Unable to malloc. Attempt number:<attempt number>"` will be printed before an attempt is tried again.
If mallocing has been attempted 5 times however, error message `"Fatal Error: Malloc was unsuccessful."` will be printed and program will self terminate.

int insertionSort(void array,int (*comparator)(void*,void*));*

Function found in *sortFile* that takes in a void **array* and a function pointer int (**comparator*)(void*,void*) and sorts the array by using the insertion sort algorithm. Comparisons between elements of the array are done by use of given *comparator* (either compareChar or compareInt). The function then calls *printArray* to print out the sorted *array* before returning 1.

int quickSort(void array,int (*comparator)(void*,void*));*

Function found in *sortFile* that takes in a void **array* and a function pointer int (**comparator*)(void*,void*) and calls split to sort *array*. The function then calls *printArray* to print out the sorted *array* before returning 1.

void split(void array,int lo,int hi,int (*comparator)(void*,void*));*

Function found in *quickFile* and *split* that takes in a void **array*, int *lo* and int *hi*, and a function pointer int (**comparator*)(void*,void*) and sorts the array recursive using the quick sort algorithm with the pivot being declared at the first element(or more simply the element given at lo). Comparisons between elements of the array are done by use of given *comparator* (either compareChar or compareInt). Function returns once sorting is finished.