Read Me:
Overall Algorithm:
	Create a hashtable to store frequencies and tokens throughout all files. This is needed to build the codebook. Check what flags are called and how many there are. Return errors if the flags are invalid input ex. -R -b -d <file> Codebook
	If input is proper, perform what the flags say. If there is a -R, then call the recursive function that walks through the directory. When a file is reached, call the method for the appropriate flag. (Call compressor if given -c decompressor for -d, build for -b).

Recursion:
	Takes in flags as an int representation (1=b, 2=c, 3=d). Opens the directory path if it was a valid path to a directory. The program traverses through each file/folder in the directory and that level and concatnates the directory name to a string called path. When a directory is found, a recursive call is made by sending in that directory. When a file is reached, the path that is created for that file is passed in as a parameter to the given flag's function.

Build Code Book:
	Structures Used:
		Hashtables - Used to store tokens and update their frequency while files are being parsed

		Linked List - Struct that contains a struct HeapNode* tree and a *next. Used to sort the minheap trees.

		Min Heap - Struct HeapNode* has fields: frequency, name, height, *left child, *right child. Used to create a min heap of the tokens and their frequencies.

		Huffman Tree - Struct HeapNode*, a tree that is organized by

	Algorithm:
		1. Store tokens and frequencies
			Parse the file using getNextToken and insert each token into a hashtable

		2. Build Minheap Array

		3. Build Huffman Tree

		4. Build Codebook
			Traverse through the huffman tree,

Compress:
	Takes a file to compress and codebook. Check if the file is a regular file, if it is then copy the filepath and concatnate ".hcz." Now get each token one by one from the file and send it into a method called retcode that will parse the codebook and check if the target string is in the codebook. If it is not it will return "3" for error and compress will remove the .hcz file created. If it is in the codebook then the huffman code is returned and compress will write this code in to the comrpessed file.

Decompress:
	Takes in the file to decompress the codebook and a struct pointer to the huffman

tree. It builds the huffman tree if it does not already exist. It reads the codebook token by token to create the huffman tree. Once the tree is created, it parses through the file to decompress character by character. If the char is a 0 go left if 1 go right else return error

Global Var:
	HashTable[10000] - Stores the tokens and their frequencies throughout all files. Useful as a global rather than passing it into functions as a pointer because almost every function in the build method needed it. Passing via pointers led to errors when trying to access the table as it was being passed into several functions that were inside of each other.
		ie: create table in main, pass into build() as a pointer, build passes it to hashToArr() to create a minheap array, which passes it into its helper functions etc.

	numToks - Stores the number of tokens inserted into the hash table throughout all files
		This is an int pointe that was being passed to many different functions all within each other as well. It caused a lot of errors this way, so it is simpler to make it into a global variable.

Time Complexity:

Space Complexity:
	Hashtable
	HeapNode
	LLNode