Shihang Zhang

===SYNOPSIS===

```
./index <inverted-index file name> <directory or file name> {<-n>}
```

===DESCRIPTION===

This indexing program will index all files in a directory or a single file. Both directory and file may be symbolic links. Also, the directory may contain symbolic link to some a directory or file. This indexer will resolve the symbolic link to trace the file or directory linked by it. (Notice: all the files should be ascii files, otherwise, it will stderr this error).

It will output the inverted index to the file following JSON format.

===IMPLEMENTATION===

Data Structure

```
struct Record {
 char *filename;
 int count;
};
```

this is the structure of record, including file name and the frequency count for particular token. all records of one token will be stored in the sorted list of the token structure in descending sorted order based on frequency counts.

```
struct Token {
 char *token;
 SortedListPtr records;
};
```

this is the structure of token, including the token and the sorted-list for its records. all tokens will be stored in a sorted list in ascending sorted order based on ASCII coding of characters.

Functions

```
void createIndex(char*);
void writeIndex(char*);
void writeFile(char *);
void readDir(char*, char*, int flag);
void readFile(char*, char*, int flag);
void readSymbolicLink(char*, char*);
void readToken(char*, char*);
int compareRecords(void*, void*);
int compareTokens(void*, void*);
int compareFilenames(void*, void*);
int compareStrings(void *p1, void *p2);
```

```
void destroyRecord(void*);
void destroyToken(void*);
void destroyString(void*);
```

see details in index.h

Global Variables

```
int FLAG;
```

this variable will denote if you like to follow the symbolic link according to the third argument -n.

```
SortedListPtr Index;
```

this sorted-list is used to hold the inverted-index.

```
SortedListPtr RelPaths;
```

this sorted-list is used to store all the filenames in inverted-index. since different tokens may appear in the same file, the same filename may be stored in different records. the sorted-list will save heap space and facilitate free malloced space preventing memory leak.

```
SortedListPtr AbsPaths;
```

this sorted list is used to avoid reading the same file. specially, it is able to solve a tough situation described below.

Internal variables and functions

```
char text[256];
```

this char table is defined in index.c and it marks all the characters may appear in ascii file.

```
int __is_ascii_file_(char*, size_t);
char* __construct_path_name_(char*, char*, int);
char* __copy_file_name_(char*);
```

see details in index.h

===EXPLANATION===

this indexer realizes all the tiers EC.
tier 2&3 EC: as to denoting symbolic link, output follows this rule “path1 -> path2”. path1 is still the relative path of that

symbolic link to the the input directory name; path2 is the actual absolute path of that file.
tier 4 EC: as you said, it is a bit tricky. i set a flag when there is -n in the arguments list and then pre-append this flag in the test if this file is a symbolic link, like this,

```
                FLAG && S_ISLNK(s.st_mode), or this,
                FLAG && ent->d_type == DT_LNK
```

tier 5 EC: the most graceful way to handle circular links and broken links or other nasty stuffs i can figure out is to stderr corresponding error information, then ignore that file and keep going.

here will have a check for the path length. if it exceeds the maximum value PATH_MAX defined in <limits.h>, the program will stop and stderr this error.

===CHALLENGES===

1. output format. the last token and record both have no comma at the end. i add another function in sorted-list called SLIsLastItem to test if the iterator reaches the last item.

2. ascii file. test if the file is a ascii file or not. thanks sgg for the help. it is a really magical method.

3. output format of symbolic link. since both file and directory may be pointed by a symbolic file, i need to redesign my readFile and readDir functions to handle the differences of the file names.

4. a tough situation. when there is a directory named ‘dir’ includes two files and one directory called ‘dir1’; dir1 includes a symbolic link which linked with ‘dir’. this will form a circle but not like a circular link in tier 5 EC. the way i figured out is just stop this program. as to how to detect this situation, i use AbsPaths (one of the global variables) to store all the absolute paths of files indexer has read and then indexer can detect if it meets the same file.

5 memory management. this project is complicated since there are many mallocs.

===COMPLEXITY===

functions in this program except system calls have O(n) time complexity.