|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <unistd.h> |
|  | #include "pnmrdr.h" |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | FILE \*input; |
|  |  |
|  | //If there is an argument, open the file if valid. If invalid, print an error |
|  | if (argc == 2) { |
|  | input = fopen(argv[1], "rb"); |
|  | if (!input) { |
|  | perror(argv[1]); |
|  | exit(1); |
|  | } |
|  |  |
|  | //checks the file extention |
|  | char \* ext = strrchr(argv[1], '.'); |
|  | if (strcmp(ext, ".pgm")){ |
|  | fprintf(stderr, "The file provided does not have a .pgm extention\n"); |
|  | exit(0); |
|  | } |
|  | } |
|  |  |
|  | //If there are no arguments, read from standard input |
|  | else if (argc == 1) { |
|  | input = stdin; |
|  | //Print an error message to stderr if no data was delivered |
|  | if (isatty(0)){ |
|  | fprintf(stderr, "You provided the progrmam with no data\n"); |
|  | exit(1); |
|  | } |
|  | } |
|  |  |
|  | //Print an error message to stderr if more than one argument is given |
|  | else { |
|  | fprintf(stderr, "You provided the progrmam with more than one argument\n"); |
|  | exit(1); |
|  | } |
|  |  |
|  |  |
|  | Pnmrdr\_T r = Pnmrdr\_new(input); |
|  | //Get the image data |
|  | Pnmrdr\_mapdata data = Pnmrdr\_data(r); |
|  |  |
|  | //Calculates the total number of pixels |
|  | int pixelNum = data.width \* data.height; |
|  | float totalBrightness = 0; |
|  |  |
|  | //Adds the pixel values divided by the denominator together |
|  | for (int i = 0; i < pixelNum; i++){ |
|  | unsigned pixel = Pnmrdr\_get(r); |
|  | totalBrightness += pixel; |
|  | } |
|  |  |
|  | //The total brightness must be divided by the total number of pixels |
|  | //to get a number between 0 and 1 |
|  | float avgBrightness = totalBrightness / (pixelNum\*data.denominator); |
|  |  |
|  | //Print the average brightness to three decimal places |
|  | printf("%.3f \n", avgBrightness); |
|  | Pnmrdr\_free(&r); |
|  |  |
|  |  |
|  | return 0; |
|  | } |

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | #include "atom.h" |
|  | #include "table.h" |
|  | #include "stack.h" |
|  |  |
|  |  |
|  | //Struct to hold the key value pairings after parsing the line |
|  | typedef struct keyValuePair { |
|  | int goodPairing; |
|  | char \* fingerprintArray; |
|  | char \* nameArray; |
|  | } keyValuePair; |
|  |  |
|  | void readInput(); |
|  | keyValuePair\* parseInputLn(char\* line); |
|  | void addToTable(char\* fingerprintArray, char\* nameArray, Table\_T table, Table\_T fGroupTable); |
|  | void printGroups(Table\_T fGroupTable); |
|  |  |
|  |  |
|  | int main() { |
|  | //table will hold the fingerprint and name pairings (as atoms) |
|  | //fGroupTable will hold the fingerprint and stack pairings |
|  | Table\_T table = Table\_new(0,NULL,NULL); |
|  | Table\_T fGroupTable = Table\_new(0,NULL,NULL); |
|  |  |
|  | //Reads the input file from standard input |
|  | FILE \*input; |
|  | input = stdin; |
|  |  |
|  | int maxLineLen = 1024; |
|  | char buf[maxLineLen]; |
|  |  |
|  | //Passes each line to be parsed and checked if valid |
|  | while (fgets(buf, maxLineLen, input) != NULL) { |
|  | keyValuePair\* parsedLine = parseInputLn(buf); |
|  |  |
|  | //Adds the pairing to fGroupTable if valid |
|  | if (parsedLine->goodPairing) { |
|  | addToTable(parsedLine->fingerprintArray, parsedLine->nameArray, table, fGroupTable); |
|  | } |
|  |  |
|  | free(parsedLine); |
|  | } |
|  |  |
|  | //Print the fgroups and free the tables |
|  | printGroups(fGroupTable); |
|  | Table\_free(&table); |
|  | Table\_free(&fGroupTable); |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* This function reads each line parses it into a fingerprint and a name char array. If \* |
|  | \* the line is badly formatted or invalid, the line will be discarded and the program \* |
|  | \* will continue. Otherwise, the valid line will become a keyValuePair and returned. \* |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | keyValuePair\* parseInputLn(char\* line){ |
|  | int maxFingerprintLen = 512; |
|  | int maxNameLen = 512; |
|  |  |
|  | //Creates the fingerprint and name arrays |
|  | char \* fingerprintArray = calloc(maxFingerprintLen, sizeof(char)); |
|  | char \* nameArray = calloc(maxNameLen, sizeof(char)); |
|  |  |
|  | //Valid fingerprint and name pairings will be stored as a keyValuePair to be returned |
|  | keyValuePair \* pairing = calloc(1, sizeof(keyValuePair)); |
|  |  |
|  | int lineLength = strlen(line) / sizeof(char); |
|  | int badInput = 0; |
|  | int i = 0; |
|  |  |
|  | //Checks if the line begins with a non-whitespace character -invalid if otherwise |
|  | if (line[0] == ' ') { |
|  | badInput = 1; |
|  | } |
|  |  |
|  | //Loops through to get the fingerprint and stores it in the fingerprint array |
|  | while (i <= maxFingerprintLen && line[i] != ' '){ |
|  | if (line[i] == '\n'){ |
|  | badInput = 1; |
|  | break; |
|  | } |
|  | fingerprintArray[i] = line[i]; |
|  | i++; |
|  | } |
|  |  |
|  | //Skips over the white space sperating the fingerprint and its name |
|  | while (i <= lineLength && line[i] == ' ') { |
|  | i++; |
|  | } |
|  |  |
|  | //Checks for a badly formed line and writes an error message to stderr |
|  | if (i == maxFingerprintLen+1 || badInput || i == lineLength-1) { |
|  | fprintf(stderr, "A line was poorly formated due to one of the following reasons:\nfingerprint exceeds %d characters,\nthere was only one argument,\nor the line started with a space\n", maxFingerprintLen); |
|  | free(fingerprintArray); |
|  | free(nameArray); |
|  | pairing->goodPairing = 0; |
|  |  |
|  | } |
|  |  |
|  | else { |
|  | int j = 0; |
|  | //Loops through the rest of the line to get the name and store it in the name array |
|  |  |
|  | if (lineLength > maxNameLen){ |
|  | free(nameArray); |
|  | nameArray = calloc(maxNameLen\*2, sizeof(char)); |
|  | } |
|  | while (i < lineLength-1 ) { |
|  | nameArray[j] = line[i]; |
|  | j++; |
|  | i++; |
|  | } |
|  |  |
|  | //Creates a keyValuePair pairing because the line was valid |
|  | pairing->goodPairing = 1; |
|  | pairing->fingerprintArray = fingerprintArray; |
|  | pairing->nameArray = nameArray; |
|  | } |
|  |  |
|  | return pairing; |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* This function takes in the valid pairing arrays and turns the fingerprint into an \* |
|  | \* atom. The pairings will then be added to a stack. \* |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | void addToTable(char\* fingerprintArray, char\* nameArray, Table\_T table, Table\_T fGroupTable) { |
|  | Stack\_T groupStack; |
|  |  |
|  | //Converts the fingerprint into an atom |
|  | const char \* fingerprintAtom = Atom\_string(fingerprintArray); |
|  |  |
|  | void \* valueCheck = Table\_put(table, fingerprintAtom ,nameArray); |
|  |  |
|  | //If Table\_put returns NULL then the name is the first one of the fingerprint group |
|  | //A new stack is created to hold the pairings |
|  | if (valueCheck == NULL) { |
|  | groupStack = Stack\_new(); |
|  | } |
|  |  |
|  | //Otherwise, we will retrieve the stack associated with the name and add the new name |
|  | else { |
|  | groupStack = Table\_get(fGroupTable, fingerprintAtom); |
|  | } |
|  |  |
|  | Stack\_push(groupStack, nameArray); |
|  | Table\_put(fGroupTable,fingerprintAtom,groupStack); |
|  |  |
|  |  |
|  | free(fingerprintArray); |
|  | } |
|  |  |
|  |  |
|  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* This function prints the fgroups, seperated by a newline, if there are more than one \* |
|  | \* name associated with the fingerprint. If there are no groups, it will print nothing. \* |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |  |
|  | void printGroups(Table\_T fGroupTable){ |
|  | //The fGroupTable will be converted into an array |
|  | void \*\*array = Table\_toArray(fGroupTable, NULL); |
|  |  |
|  | //Used to check if it is the first group so it will print out with newlines between groups |
|  | int firstGroup = 1; |
|  |  |
|  | for (int i = 0; array[i]; i += 2) { |
|  | //Ignore stacks of size one |
|  | if (\*(int \*)array[i+1] > 1) { |
|  | if (!firstGroup) { |
|  | printf("\n"); |
|  | } |
|  | else { |
|  | firstGroup = 0; |
|  | } |
|  |  |
|  |  |
|  | Stack\_T stack = array[i+1]; |
|  |  |
|  | //Print the names within the stacks |
|  | while (! Stack\_empty(stack)) { |
|  | char\* name = Stack\_pop(stack); |
|  | printf("%s\n", name); |
|  | free(name); |
|  | } |
|  |  |
|  | Stack\_free(&stack); |
|  | } |
|  |  |
|  | else { |
|  | Stack\_T stack = array[i+1]; |
|  | char\* name = Stack\_pop(stack); |
|  | free(name); |
|  | Stack\_free(&stack); |
|  | } |
|  | } |
|  |  |
|  | free(array); |
|  | } |