intersect

Column	
Grade	
Language	

Project Requirements

Overview

The purpose of the program is to take a number of command line arguments that are paths to text files. The program will then read through each file and determine which words are in common between all the files. Once the program is done, it will then printout all the words in `alphabetical` order. When reading from the file, the words are delimited by `isspace(3)` (`\f, \n, \r, \t, \v`). The printed word should match the case from when it was first found.

Requirements Checklist

```
-Wall -Wextra -Wpedantic -Waggregate -return -Wwrite-strings -Wvla -Wfloat-equal --std=c18
```

Project Requirements

- [x] ~~Doc files located in `project_root/doc`~~
- [x] ~~Create a design plan `design.pdf`~~
- [x] ~~Create a write up as `jgalindez.pdf`~~
- [ ] Create a test plan `test_plan.pdf`
- [x] ~~Run finished project against Valgrind~~
- [x] ~~Repo named `intersect`~~
- [x] ~~Project must run on the class VM~~

code requirements

- [x] ~~Support `make`~~
- [x] ~~Support `make check`~~
- [x] ~~Support `make profile`~~
- [x] ~~Support `make debug`~~
- [x] ~~Support `make clean`~~
- [x] ~~Program invoked with `intersect <file> <file>`~~

Bonus Requirements

- [x] ~~[10 pts] Support `UTF-8` encoded files~~
- [x] ~~[5 pts] Support `-a` to printout all versions of the matched word~~

- [x] ~~Program must work with long lines and long words~~
- [x] ~~Words must be sorted `alphabetically` not `lexicographical` order~~
- [x] ~~Word case must match the original case of the word~~

Brainstorm and Research

Step By Step

1. parse command line
    1. Run `intersect` and check for at least 2 file paths, else fail.
    2. `validate` file paths to make sure that they resolve to a file
2. create `tree map` structure to start adding the `key_val`
    1. Each node will have a `key` of the `towupper()` of whatever word it matches on. This decreases the amount of `towupper()` calls
    2. Each node will also have a `file_count` this will be used for cleaning up the tree. The idea is that every file that gets parsed will increment this value indicating that it was found. So, after the second file has been parsed, all nodes with a `file_count` of less than 2 will be removed
    3. Each node will contain an array or single linked list of all the string versions of the match
3. Parse each file
    1. The key_val pair should be

```
typedef struct
{
  uint32_t file_count;       // value that indicates what file it was added from this helps delete nodes later on
  wchar_t * key_upper_case;  // this is the key that is used to compare in the main tree
  wchar_t * original;        // the original word
  dict_t * word_versions;    // all versions of the word are stored in a dictionary
}
```

    2. This structure will then be compared to each key_val_t in the main dictionary
        1. `IF WORD NOT FOUND` and it's `FIRST FILE`
            1. add node, set `file_count = 1`
        2. `IF WORD NOT FOUND` and it's after first file
            1. `IGNORE`
        3. `IF WORD FOUND`
            1. Increment `file_count`

        2. Call `dict_put(word)` on the `word_versions` using the `original` that will take care of checking if it's unique

    3. When a file is complete being processed, trim the tree. Delete all nodes that do not have a `file_count` equal to the amount of files processed

4. Print all words in `IN_ORDER` sort

What data structure to use?

- Using a AVL tree it's going to be `O(log (n))` for all operations. There can be up to 10million distinct words
    - `O(log (10m)) = 23`
- Using a Trie it's going to be `O(L)` where L is the number of letters in a word. If words have more than 23 letters we end up with a worse time complexity than AVL tree.
- But, with AVL I will be using `strcmp` which runs in `O(n)` so that will make the comparison `O(L) vs O(n)`. I think I will take those odds to ensure that I finish on time with the baby and all *as she is screaming in the other room...*

Collation Algorithm UTF-8

- Collation is a way to categorize base letters. The base letter c has the same value as all other forms of it with different symbols on it. The way it gets differentiated is by storing the marker in another byte. For this reason there is a Collation algorithm to determine the order of words regardless of the language.
- Looks like I can use the Collation Algorithm built into the `locale` by specifying the `LC_COLLATION` of `en_US.UTF-8` it matches what is displayed on the Collation Algorithm site

```
#include <stdio.h>
#include <string.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>


int main(void)
{
  setlocale(LC_ALL, "en_US.UTF-8");

    wchar_t * s1 = L"côté";
    wchar_t * s2 = L"côte";

    wprintf(L"In the en_US.UTF-8 locale: ");
    if(wcscoll(s1, s2) < 0)
         wprintf(L"%ls before %ls\n", s1, s2);
    else
         wprintf(L"%ls before %ls\n", s2, s1);


  for (int i = 0; i < wcslen(s1); i++)
  {
    putwchar(towupper(s1[i]));
  }

  wprintf(L"\n");
}
```

Testing

Because I'm using UTF-8 it does not work with check.h

printf and wprintf in single C code

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other answers. Making statements based on opinion; back them up with references or personal

https://stackoverflow.com/questions/8681623/printf-and-wprintf-in-single-c-code

Design and Implementation

Dictionary struct

```
typedef struct
{
  uint32_t file_count;       // value that indicates what file it was added from this helps delete nodes later on
  wchar_t * key_upper_case;  // this is the key that is used to compare in the main tree
  wchar_t * original;        // the original word
  dict_t * word_versions;    // all versions of the word are stored in a dictionary
}
```

KanBan

kanban

Name	Tags
Complete Tree Map Interface	COMPLETE
Create Intersect API	COMPLETE
Create Command Line Parse	COMPLETE
Create intersect file parser	IN-PROGRESS
Create Test Cases	IN-PROGRESS
Write Up	NOT STARTED

Resources

Collation Algorithm

Collation Functions (The GNU C Library)

In some locales, the conventions for lexicographic ordering differ from the strict numeric ordering of character codes. For example, in Spanish most glyphs with diacritical marks such as accents are not considered distinct letters for the purposes of collation.

https://www.gnu.org/software/libc/manual/html_node/Collation-Functions.html

Locale Categories (The GNU C Library)

The purposes that locales serve are grouped into categories, so that a user or a program can choose the locale for each category independently. Here is a table of categories; each name is both an environment variable that a user can set, and a macro name that you can use as the first argument to setlocale.

https://www.gnu.org/software/libc/manual/html_node/Locale-Categories.html

Internationalization (i18n) - Collate (Sort) Order, Character Set, Accents, GLOB patterns

This file should help you understand Unix/Linux scripts in a world of increasing internationalization (i18n). I used to say that a shell script only needed to set two things to behave properly no matter what nonsense was set in the parent process that invokes the script: PATH and umask #!/bin/sh -u PATH=/bin:/usr/bin ; export PATH umask 022 Internationalization imposes a third and fourth consideration: character collation order and the input

http://teaching.idallen.com/cst8177/13w/notes/000_character_sets.html

Gnu sort UTF-8 incorrect collation order

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other answers. Making statements based on opinion; back them up with references or personal

https://stackoverflow.com/questions/41667652/gnu-sort-utf-8-incorrect-collation-order

Unicode programming, with examples

Most programming languages evolved awkwardly during the transition from ASCII to 16-bit UCS-2 to full Unicode. They contain internationalization features that often aren't portable or don't suffice. Unicode is more than a numbering scheme for the characters of every language -

https://begriffs.com/posts/2019-05-23-unicode-icu.html

ICU - International Components for Unicode

ICU Project Site

http://site.icu-project.org/

ICU Documentation

The ICU User Guide provides information on i18n topics for which ICU has services, and includes details that go beyond the C, C++, and Java API docs (and avoids some duplication between them). This is the new home of the User Guide (since 2020 August).

https://unicode-org.github.io/icu/

http://teaching.idallen.org/net2003/06w/notes/character_sets.txt printf and wprintf in single C code

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other answers. Making statements based on opinion; back them up with references or personal

https://stackoverflow.com/questions/8681623/printf-and-wprintf-in-single-c-code

C in a Nutshell

Transforms a wide string for easier locale-specific comparison The wcsxfrm() function transforms the wide string addressed by , and copies the result to the wchar_t array addressed by . The third argument, , specifies a maximum number of wide characters (including the terminating null wide

https://learning.oreilly.com/library/view/c-in-a/0596006977/re285.html