PIC 10C Section 1 - Homework # 3 (due Friday, April 15, by 11:59 pm)

You should upload each .cpp and/or .h file separately and submit them to Bruin Learn before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine.

Also be sure your code compiles and renders the correct output on g++ with -O3 optimization on the C++20 Standard!!!

SIMPLE PLAGIARISM DETECTION

This homework, in addition to working with the **GNU compiler** is about using the **filesytem** brought forth in C++17. Using the file system features, along with a standard algorithm for detecting the largest common subsequence between two files, it will be possible to do pairwise comparisons of files to determine a measure of overlap. Please refer to the syllabus for how the work will be graded: note that more than half of the marks come from good coding practices and code documentation.

The essence of our approach here can be used in more sophisticated plagiarism detection algorithms: with a little preprocessing for things like comments, handling grammar/keywords/syntax, allowing variable/function names or proper nouns/synonyms to be interchanged, and modifying the form of overlap sought (subsequence vs blocks of sequences), something more appropriate to studying code, English essays, or even entire novels can be built. Hopefully you can appreciate just how awesome this file system stuff can be!

Your Task:

You must write **main.cpp** and **compare.cpp** along with a header file **compare.h** that, when compiled and then run through the command line will compare documents for their longest common subsequence *of characters, including whitespace.*

When running with the command line, **the user will specify the name of a directory as a command line argument in running the program**. Within that directory will be *multiple folders of unspecified names*. Within each of those folders will be *one file of an unspecified name*. The program will do pairwise comparisons of all files and then print the results to the console.

You can assume the code will be compiled with a **makefile**:

```
hw3: main.cpp compare.cpp compare.h
    g++-8 -std=c++2a -O3 main.cpp compare.cpp -o hw3 -lstdc++fs
```

The **makefile** and the files demoed here are included on Bruin Learn.

The format of its printing shall be:

Pairing [*folder_ name_ given*]/[*FolderX*]/[*file_ nameX*]-[*folder_ name_ given*]/[*FolderY*]/[*file_ nameY*]
Common Subsequence Length: [*the_ length*]
Overlap:
[*overlap*]
————————————————————

The **folder_name_given** is what the user enters. The **FolderX**/**FolderY** represent the folders within which their files **file_nameX**/**file_nameY** are being compared.

Do not worry about the pathological cases where there is a tie for the longest common subsequence: just find one. But along the lines of edge cases, you *should be able to handle the case when an empty file* is being compared against another file (and the common subsequence should be empty).

Suppose that four fake students submitted work to print consecutive integers. *Hypothetically* (the case your homeworks are tested against could differ), we suppose that all the submissions are in the folder **submissions** that is local to the current working directory. But within that folder, each student's work is in their own folder. Their foldes/files are:

Alice/main.cpp

```
#include<iostream>
```

```
// prints consecutive values
void print(int low, int up) {
  for(int i = 0; i <= up; ++i){
    std::cout << i;
  }
}

int main(){
  // print from 3 to 30 inclusive
  print(3, 30);

  return 0;
}
```

Bob/bob.cpp

```
#include<iostream>

void print(int low, int up) {



  for(int i = 0; i <= up; ++i){
    std::cout << i;
  }


}

int main(){

  print(3, 30);

  return 0;
}
```

Cecilia/print.cpp

```
#include<iostream>

/**
prints values over a range
@param first the first number to print
@param last the last number to print
```

```
*/
void print(int first, int last) {
  for(; first <= last; ++first){ // keep increasing first and printing it
    std::cout << first;
  }
}

int main(){
  // prints 3 ... 30
  print(3, 30);

  return 0;
}
```

Diego/source.cpp

```
#include<iostream>

void print(int x, int y) {
  while(x <= y){
    std::cout << x++;
  }
}

int main(){
  print(3, 30);
}
```

Assuming the exectable is called **hw3** then here is an illustration for how the program runs (note how the user specifies the directory as an argument):

```
> ./hw3 submissions
Pairing submissions/Bob/bob.cpp-submissions/Alice/main.cpp
Common Subsequence Length: 154
Overlap:
#include<iostream>

void print(int low, int up) {
  for(int i = 0; i <= up; ++i){
    std::cout << i;
  }
}

int main(){
```

```
  print(3, 30);

  return 0;
}
-----------------
Pairing submissions/Cecilia/print.cpp-submissions/Alice/main.cpp
Common Subsequence Length: 173
Overlap:
#include<iostream>

/prints onseutie aue
void print(int , int ) {
  for(; i <= ; ++i){
    std::cout << i;
  }
}

int main(){
  // print 3  30
  print(3, 30);

  return 0;
}
-----------------
Pairing submissions/Cecilia/print.cpp-submissions/Bob/bob.cpp
Common Subsequence Length: 138
Overlap:
#include<iostream>

void print(int , int ) {
  for(; i <= ; ++i){
    std::cout << i;
  }
}

int main(){

  print(3, 30);

  return 0;
}
-----------------
Pairing submissions/Cecilia/print.cpp-submissions/Diego/source.cpp
Common Subsequence Length: 110
```

```
Overlap:
#include<iostream>

void print(int , int ) {
  i <= ){
    std::cout << ;
  }
}

int main(){
  print(3, 30);
}
-----------------
Pairing submissions/Diego/source.cpp-submissions/Alice/main.cpp
Common Subsequence Length: 110
Overlap:
#include<iostream>

void print(int , int ) {
  ( <= ){
    std::cout << ;
  }
}

int main(){
  print(3, 30);
}
-----------------
Pairing submissions/Diego/source.cpp-submissions/Bob/bob.cpp
Common Subsequence Length: 110
Overlap:
#include<iostream>

void print(int , int ) {
  ( <= ){
    std::cout << ;
  }
}

int main(){
  print(3, 30);
}
-----------------
```

Remark: the results would change if instead of comparing **char-by-char** one compared **string-by-string**. That is another variant.

Hints: you can ignore these if you want, but this is a pretty reasonable approach to the problem:

1. Write a class **CompareFiles** that can be constructed from two **path**s (of files to be compared), which has a comparison member function (to run the comparisons and store the results) and a getter to retrieve the longest common subsequence (perhaps stored as **std::vector<char>**).
2. It may make sense for that comparison function to run in three stages:
   (a) call a function to generate the boolean matrix,
   (b) call a function to compute the longest common subsequence length from that matrix and set a matrix of path lengths, and
   (c) then call a function to use the backtracking to compute the longest common subsequence from the boolean and length matrices.
3. Write a function that can scan the appropriate directories and return the paths to all files that need to be compared (perhaps as an **std::vector<std::filesystem::path>**.
4. Write a function that can generate all pairings of paths given the result from the previous function.
5. The overall program then amounts to:
   (a) get all file pairings (from above),
   (b) push a bunch of **CompareFiles** objects constructed with those pairings into a vector,
   (c) run the compare function on all elements of the vector, and
   (d) then extract the statistics from all elements of that vector.