COMP 2012H Honors Object-Oriented Programming and Data Structures

Assignment 2 A Simple Text Processor

Honor Code

We value academic integrity very highly. Please read the Honor Code section on our course webpage to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment. Please read the Honor Code thoroughly.
- Serious offenders will fail the course immediately, and there will be additional disciplinary actions from the department and university, upto and including expulsion.

End of Honor Code

Objectives & Intended Learning Outcomes

The objective of this assignment is to familiarise you with functions, arrays, and parameter passing. After this assignment you should be able to:

1. Learn how to use functions to divide your program into small components
2. Utilise pass by reference to improve the runtime
3. Use `char` arrays to store and manipulate a string of characters
4. Use const references to ensure that variables are not modified in functions

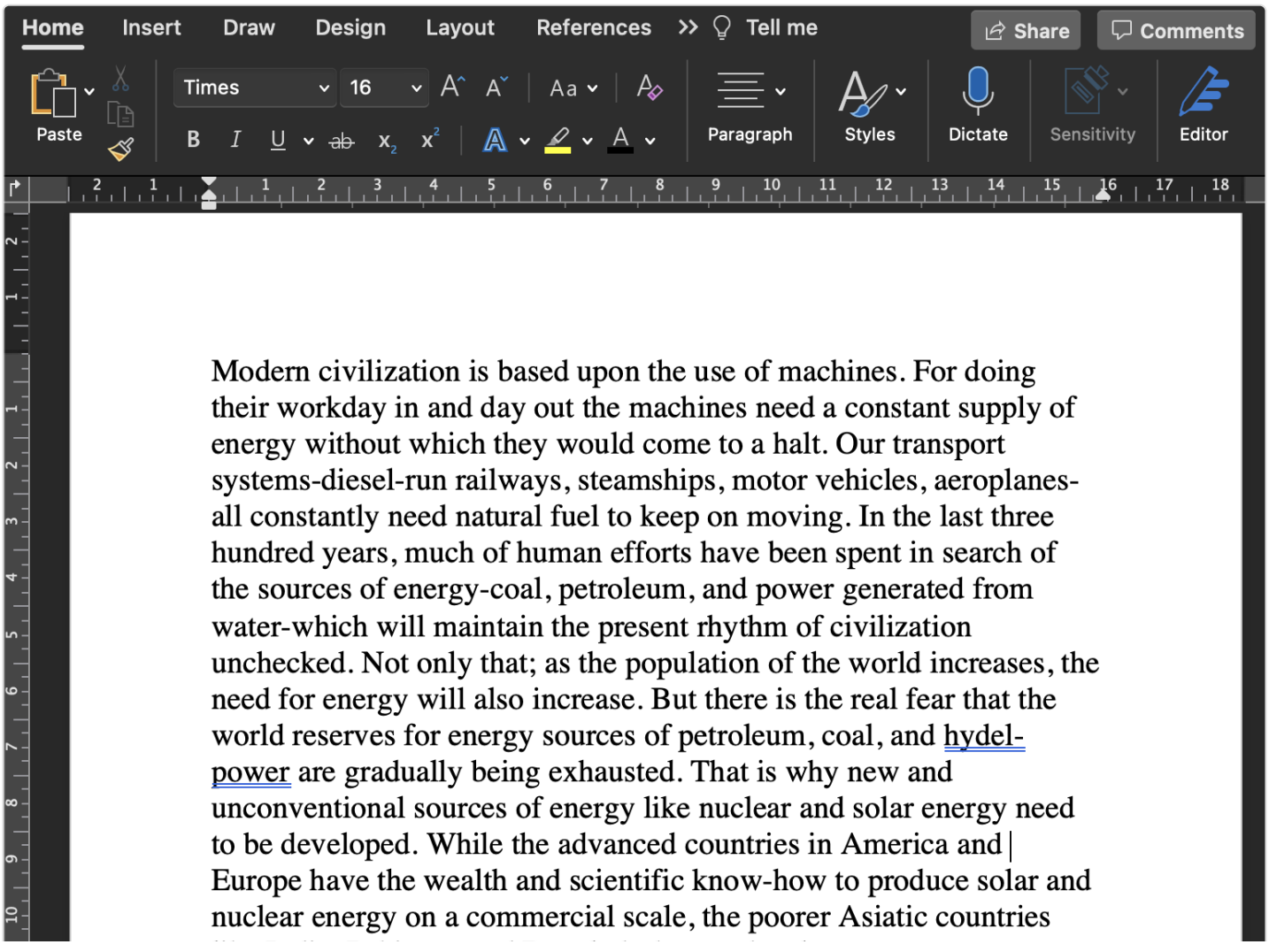
End of Objectives & Intended Learning Outcomes

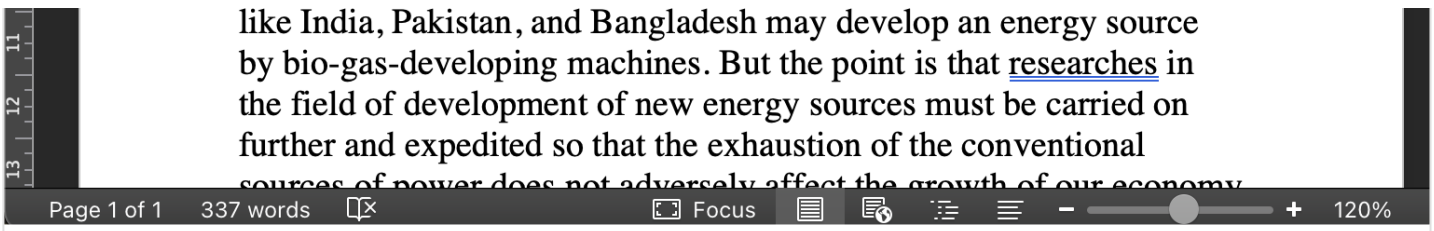
Page maintained by

SONG Sizhe
sizhe.song@connect.ust.hk
Last Modified:
09/13/2022 13:59:17

Homepage

Course Homepage





An example of a text processor: Microsoft Word

Introduction

The objective of this assignment is to implement a few functionalities of a text processor. A text processor allows users to manipulate a string of characters. These manipulations provide innumerable options with which to personalise a document. For instance one can:

- Change the alignment of the text so as to right align, left align, or justify (right and left align) it.
- Perform global replacement of a certain string in the text with another string.
- Keep track of the number of words and characters in the text.
- Change the case (upper or lower) of a certain piece of text.
- Print the text in various formats.
- Count the number of appearances of a certain string in a text.

In this assignment, you will be required to:

- Implement tailored versions of the mechanisms outlined above.
- Implement a simple Caeser Cipher
- Print a character array in an argot called Pig Latin

End of Introduction

Description

Please read the FAQ section regularly, and do check it one day before the deadline to make sure you don't miss any clarification, even if you have already submitted your work by then. You can also raise questions on Piazza and remember to use the "pa2" tag.

Code structure

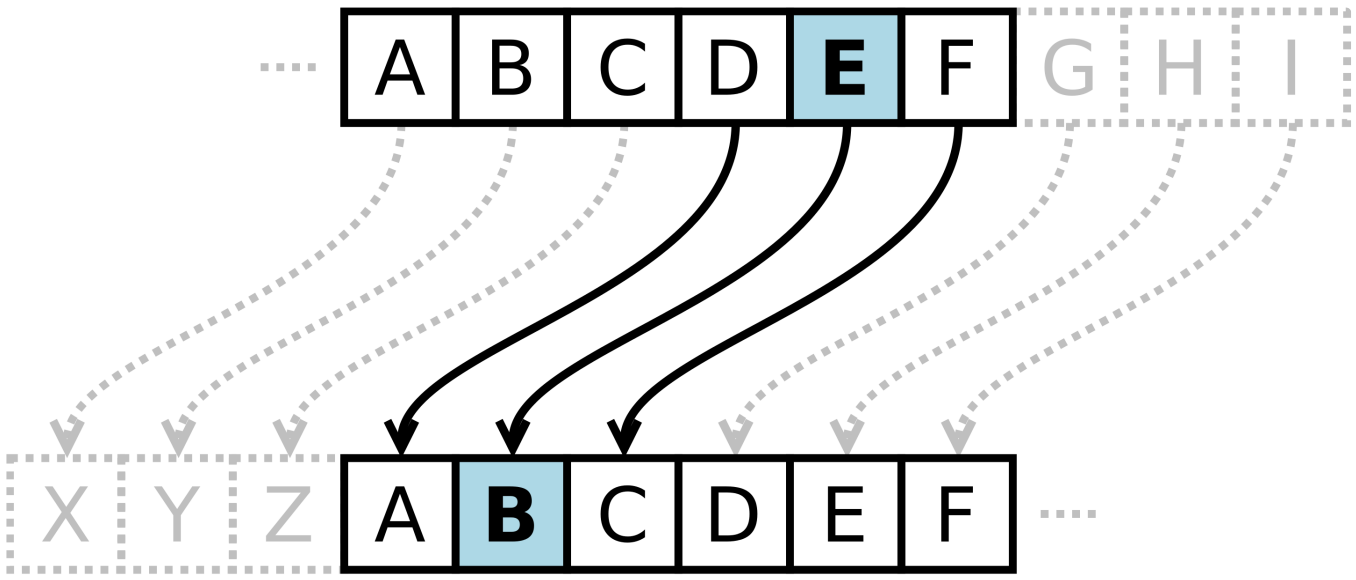
The skeleton code structure is as follows:

```
PA2
└── main.cpp
```

Your task for this PA is to implement 10 functions in `main.cpp`.

Caeser Cipher

The Caeser Cipher is a simple shift cipher that modifies alphabetical words in order to produce a text whose meaning has been obfuscated. Ciphers are used to encrypt text to make it unreadable. A Shift Cipher achieves this by taking a number `shift` and shifting every alphabetical character of the text by the alphabet that is `shift` modulo 26 positions down the alphabet. For instance, with a right shift of 5, 'A' would become 'F', 'G' would become 'L', and 'X' would become 'C'. Note that uppercases and lowercases are maintained.



A simple Shift Cipher that rotates left by 3 (or, alternatively, right by -3).

Source: Link

- If `shift` is 5:
  - "Object-Oriented Programming" would be "Tgojhy-Twnjsyji Uwtlwfrrnsl"
  - "The quick brown fox jumps over the lazy dog" would be "Ymj vznhp gwtbs ktc ozrux tajw ymj qfed itl"

Pig Latin

Pig Latin is a Language Game that modifies English words to produce a "cryptic" rendition that makes it hard to follow and understand. This is often used by children who want to converse with each other without letting adults understand what they are saying. Every word is individually converted from English to Pig Latin using the following conversion rules:

- If the first letter is a vowel (`'a'`, `'e'`, `'i'`, `'o'`, `'u'`) then simply append "yay" to it. For example:
  - "ointment" -> "ointmentyay"
- If the first letter is not a vowel then shift it to the end until the first letter is a vowel and then append "ay" to it. For example:
  - "consume" -> "onsumec" -> "onsumecay"
  - "cheap" -> "heapc" -> "eapch" -> "eapchay"
  - "warm-up" -> "arm-upw" -> "arm-upway"

Input Assumptions

For the string (`char` array) that you are operating on, you can assume that:

- It is correctly terminated with a `'\0'` character.
- The string will not be longer than `MAX_STRLEN`. All the chracters including `'\0'` can be stored inside an array `char str[MAX_STRLEN]`. This one will still hold even after any operations (functions) you need to implement.
- The string will only contain these characters before the `'\0'` character:
  - `'a'` - `'z'` (lowercase letters)
  - `'A'` - `'Z'` (uppercase letters)
  - `'-'` (hyphen)
  - `','` (comma)
  - `'.'` (fullstop)
  - `' '` (whitespace)

Important Requirements

There are a few things you CANNOT do. Failure to observe these rules would potentially result in ZERO mark for your assignment. Please carefully check your code before you submit it.

- This assignment is all about string operation exercises, so you are NOT allowed to include any additional libraries for example `cstring` and `string.h`.
- You are NOT allowed to modify any of the given function prototypes including the return types, function names, and parameter lists.
- You are NOT allowed to define any additional global variable. You can only use those provided in the skeleton codes.

End of Description

Tasks

This section describes the 10 functions that you will need to implement.

```
int countCharacters(const char str[]);
```

Description - Returns the number of characters in `str`.
For example, if:

- `str` is `"Hello world."`,

Then the function would be return 12. (10 letters + 1 whitespace + 1 fullstop)

Parameters

- `str` - string of text.

Return value - Number of characters in `str` until the null-terminator (`\0` itself excluded).

```
int countWords(const char str[]);
```

Description - Returns the number of words in `str`. Read the notes below for the definition of words.

Parameters

- `str` - string of text.

Return value - Number of words in `str` until the null-terminator `\0`.

Notes

- Words will only contain letters and hyphens.
- All consecutive letters or hyphen together forms a word. That means between each two words there are **at least** one non-word character.
- For example:

  The string `"Here is the well-designed program."` contains 5 words.
- There will be some "strange" words under this definition, e.g. "-", "-abc", "abc-", it's fine. We still count them.

```
void swapString(char str[], const char target[], const char to[]);
```

Description - Replaces every instance of the characters in `target` in `str` with the characters in `to`.
For example, if:

- `str` is `"I have a course in the morning"`.
- `target` is `"course"`.
- `to` is `"zoom meeting"`.

Then `str` would be updated to be `"I have a zoom meeting in the morning"`.
If:

- `str` is `"I would like an ice cream"`.
- `target` is `"ice "`.
- `to` is `"ice-"`.

Then `str` would be updated to be `"I would like an ice-cream."`.
Note that each characrer can only be matched at most once, so if:

- `str` is `"ssss"`.
- `target` is `"sss"`.
- `to` is `"a"`.

Then `str` would be updated to be `"as"` but not `"aa"`.

Parameters

- `str` - string of text.
- `target` - the target substring to be replaced.
- `to` - the new substring to replace `target`.

Notes

- The replacement could change the total length of `str`, make sure that the null-terminator is placed correctly after the replacement.

```
void encryptText(char str[], int shift);
```

Description - Encrypts the text in `str` using a Shift Cipher by shift amount `shift` to the right. Note that `shift` can be negative to introduce a left shit. You may want to read the intro on Caeser Cipher first.

For example, if:

- `str` is `"She sells seashells on the seashore."`, and
- `shift` is `15`

Then `str` would be updated to be `"Hwt htaah htphwtaah dc iwt htphwdgt."`.

Parameters

- `str` - string of text.
- `shift` - the amount to **right** shift the alphabet characters by.

Note

- Your codes should be able to handle inputs like `shift == 53` which is simply equivalent to `shift == 1`.
- Your codes should be able to handle negative inputs like `shift == -77` which is simply equivalent to `shift == 1`.
- `'a' <= c && c <= 'z'` will tell you whether character `c` is a lowercase letter.

```
int countNumOccurences(const char str[], const char target[]);
```

Description - Counts the number of occurences of `target` in `str`.

For example, if:

- `str` is `"Why have a ballroom with no balls."`.
- if `target` is `"balls"`, then the function would return 1.
- if `target` is `"ball"`, then the function would return 2.
- if `target` is `" ball"`, then the function would return 2.
- if `target` is `"ball "`, then the function would return 0.

Note that each character can be matched at most once, so if:

- `str` is `"ssss"`.
- `target` is `"ss"`

Then the result should be 2 but not 3.

Parameters

- `str` - string of text.
- `target` - the substring to be counted.

Return value - the number of occurences of `target` in `str`.

```
void convertIntoLines(const char str[], char lines[MAX_LINES][NUM_CHARS_PER_LINE]);
```

Description - Converts `str` into lines which contain at most `NUM_CHARS_PER_LINE` characters (including the null-terminator, that means you actually can only have `NUM_CHARS_PER_LINE - 1` meaningful characters per line) and save the lines with `lines`.

Parameters

- `str` - string of text.
- `lines` - a 2D `char` array, the container of the results.

Rules for Conversion

- Insert as many characters to a line as possible, but remember to leave one position for `'\0'`.
- If there is not enough space for the last word in current line, then terminate the line and start a new line. The word will be put to the new line.
- If a line is full, then terminate the line and start a new line.
- We will not insert any whitespaces to the beginning of a line. They will be skipped.
- Discard all the tailing whitespaces after lines are split. That means, in each line of the final results, the second last character is not `' '` (the last character is `'\0'`).
- If there are unused lines in the end, make sure to put a `'\0'` to the first position of them.
- For example, suppose `NUM_CHARS_PER_LINE = 10`. Then if:
  - `str` is `"Mary had a little lamb, little lamb, little lamb"`.

  Then the corresponding lines would be:
  - `lines[0] = "Mary had\0"`
  - `lines[1] = "a little\0"`
  - `lines[2] = "lamb,\0"`
  - `lines[3] = "little\0"`
  - `lines[4] = "lamb,\0"`
  - `lines[5] = "little\0"`
  - `lines[6] = "lamb\0"`
- In this function, you may assume that the results will not exceed `MAX_LINES` lines. Also there won't be words longer than `NUM_CHARS_PER_LINE - 1` characters, otherwise it's impossible to solve.

```
void printLeftJustified(const char str[]);
```

Description - Prints `str` in a left-aligned format. This can be done simply by calling the `convertIntoLines` function you just implemented and then print the lines one by one.

Parameters

- `str` - string of text.

Notes

- Only print those used lines, do not print those unused lines.
- Use the example in last task again, suppose `NUM_CHARS_PER_LINE = 10`:
  - if `code` is `"Mary had a little lamb, little lamb, little lamb"`.
  - Then the function will output:

```
Mary had
a little
lamb,
little
lamb,
little
lamb
```

```
void printRightJustified(const char str[]);
```

Description - Prints `str` in right-aligned format. It means that you will insert blank spaces to the beginning of each line so that the total character count of each line is exactly `NUM_CHARS_PER_LINE - 1` (`'\0'` excluded as it will not be part of the output).

Parameters

- `str` - string of text.

Notes

- Again, make use of your implemented `convertIntoLines()` function.
- Only print those used lines, do not print those unused lines.
- The example again, suppose `NUM_CHARS_PER_LINE = 10`:
  - if `code` is `"Mary had a little lamb, little lamb, little lamb"`.
  - Then the function will output:

```
Mary had
a little
   lamb,
  little
   lamb,
  little
    lamb
```

```
void printJustified(const char str[]);
```

Description - Prints `str` in right-left-aligned format. It means that you will insert blank spaces to the middle of each line (except the last line) so that the total character count of each line (except the last line) is exactly `NUM_CHARS_PER_LINE - 1` (`'\0'` excluded as it will not be part of the output).

Parameters

- `str` - string of text.

Notes

- Again, make use of your implemented `convertIntoLines()` function.
- You first compute how many white spaces you need to insert into a line.
- Then, find all pieces of consecutive white spaces in the original line.
- Starting from the first piece of consecutive white spaces, add one more white space to it, and then add to the next piece. If after adding to the last piece, you still have extra white spaces to be added, then start from the first piece again.
- Consider such a line, we use black color to represent non-space characters, and white color to represent space characters, the number represents number of characters.

The above line contains 20 characters, let's say `NUM_CHARS_PER_LINE` is 25, then we need to insert 4 extra spaces to this line. There are 3 pieces of consecutive white spaces in this line, so we will add one more space to each of them. Then there is still one more space to add, we start from the first piece again and add that final white space to it. The final printed line will look like this:

- You may assume that there is at least one white space in each line (except the last line).
- The last line will still be printed in left-aligned format.
- Please try the demo program to see how it works.

```
void convertStrToPigLatin(char str[]);
```

Description - Converts the words in `str` to Pig Latin. You may want to read the intro on Pig Latin first.

Parameters

- `str` - string of text.

Notes

- You don't need to worry about letter cases in this task. We will eventually convert all letters in your results to lower cases in the skeleton codes.
- You can assume that all words will have at least one vowel letter for this task.
- An example:
  - if `str` is `"I would like to have an ice-cream."`
  - then str would be updated to `"iyay ouldway ikelay otay avehay anyay ice-creamyay."`
  - Again, the letter case doesn't matter, skeleton codes will take care of it.
- This operation could change the total length of `str`, make sure that the null-terminator is placed correctly after the replacement.

End of Tasks

Resources & Sample I/O

- Skeleton code: PA2_Skeleton.zip
- Demo programs (last update on **9/9 18:38**):
  Windows / Linux / macOS x86 / macOS arm
  (If you find the demo program doesn't behave as you expect, please first download and try the lastest version, if the problem is still there you can let me know. Thanks!)
- Sample program outputs: Sample Outputs

If you encounter problems running the MacOS demo programs, you can try look for a Windows PC on campus or use the Virtual Barn from anywhere to run the Windows demo program.

End of Resources & Sample I/O

Submission & Grading

Deadline: 24 September 2022 Saturday HKT 23:59.
Submit a single `main.cpp` to ZINC.

Grading Scheme

There are 12 pre-deadline test cases in ZINC for you to debug your program, after the deadline, some more test cases will be added and your score for this PA will be the sum of all test cases in ZINC. The table below roughly shows the weight of each task.

Task Function	Weight
Task 1 - `countCharacters()`	5%
Task 2 - `countWords()`	10%
Task 3 - `swapString()`	15%
Task 4 - `encryptText()`	10%
Task 5 - `countNumOccurences()`	10%
Task 6 - `convertIntoLines()` `printLeftJustified()` `printRightJustified()`	15%
Task 7 - `printJustified()`	15%
Task 8 - `convertStrToPigLatin()`	20%

End of Submission & Grading

Frequently Asked Questions

Q: My code doesn't work, here it is, can you help me fix it?
A: As the assignment is a major course assessment, to be fair, you are supposed to work on it by yourself and we should never finish the tasks for you. We are happy to help with explanations and advice, but we are **not allowed** to directly debug for you.

Q: It seems that `convertIntoLines()` can possibly place commas and full stops to the beginning of a line. Is that acceptable?
A: It is guaranteed that this situation will not appear in any test cases. You can implement the function according to the description, and we will make sure that in the correct output of all test cases, none of the lines will begin with a comma or a full stop.

End of FAQ