

---

## PA203 – Tiny Taurahe Translator

---

We would have troubles writing a program to actually translate English sentences in into another natural language. Of course, if we make up the language and fake a translation process, then things get suddenly easier.

This PA encourages you to leverage the available standard library functions instead of re-implementing your own versions.

### Files you will be working with

---

You will be provided with several files to get you started working on this assignment.

- You must not alter the file names, remove or add files to the project
- You must only modify the ones marked below with a **yes** in “Modify it?”
- You must not insert any comments or code in the *tests.c* file which, when read by another student, would give them any insights about the solutions you implemented in *tools.c*.

Important Academic Honesty Note;

*The role of tests.c is to allow you to test your program to verify it adheres to requirements. Your instructor might allow you to exchange this file, and this file only, with other students. Therefore, you must uphold academic honesty standards by not inserting any information, besides the tests, which would divulge your design or implementation of the solutions to another student. Failure to do so will earn you a FF for the offering.*

Here are the files;

File name	Modify it?	Role
<i>tools.c</i>	Yes	Implementation of your solution to the assignment
<i>tests.c</i>	Yes	Implementation of your test functions
<i>tools.h</i>	No	Header file for <i>tools.c</i>
<i>main.c</i>	No	Implementation of the main function starting your tests
<i>testlib.h</i>	No	Definition of the <i>TEST</i> function you must use in your tests
<i>testlib.c</i>	No	Implementation of the above

## Task #1 –Implementing and testing taurahize\_word

---

The first function we need to implement in *tools.c* is the one which, given a string parameter with an English word, will return a newly allocated string containing its translation in “Taurahe”. Its prototype is as follows;

```
char* taurahize_word( const char * const word );
```

The translation process is pretty straightforward; we are measuring the length of the **word** which was passed as parameter, assuming it uses the whole string to make things simpler, and we look up in a table to see what all words of this length translate into.

# letters in word	Taurahe Translation	# letters in word	Taurahe Translation
0	“”	8	“Akiticha”
1	“A”	9	“Echeyakee”
2	“Ba”	10	“Awakahnahe”
3	“Aki”	11	“Aloakihshne”
4	“Aoke”	12	“Awakeekieloh”
5	“Aehok”	13	“Ishnehawahalo”
6	“Aloaki”	14	“Awakeeahmenalo”
7	“Ishnelo”	15	“Ishnehalohporah”

This “translation table” will have to be a static array of strings inside your function. At index *i*, ranging from 0 to 15, we will have a string representing our “translation” in Taurahe of all English words of length *i*.

Translating means measuring the length of **word**, looking up the Taurahe word in the table and returning a new copy of it obtained with **strdup**. Attempting to translate a word longer than 15 characters will result in the function returning a copy of “#@%” which is Taurahe for “*whatever, man*”.

## Task#2 – Implementing and testing `taurahize_phrase`

---

Now that we know how to “Taurahize” one word at a time, let’s translate entire phrases! Here is the prototype of this new function;

```
char* taurahize_phrase( const char * const phrase);
```

This new function will start by measuring the length of the string to translate and allocate dynamically a string of same length. The address of this “translation buffer” will be stored in a **translation** pointer and will be initialized to have a ‘\0’ as first character. It will be used to build, step by step, the translation of the whole string that was given to us.

We will then use **strdup** to make a duplicate of the string to translate. This “working copy” will be passed to the **strtok** function in order to extract all its words one by one. We will assume that only the space character ‘ ’ is used as a delimiter between words.

Each word will then be passed to **taurahize\_word** so that we obtain a new string representing its translation. Each of these will be appended, using **strcat**, to our **translation**. Keep at least a space between each word in **translation**. However, it’s fine if we have just a single space between translated words even if the original string had multiple spaces between words. We don’t guarantee that we keep the spacing as it originally was during our translation process since it doesn’t alter the “meaning” of the translation.

Now, make sure you return the address of your translation **translation** and de-allocate any memory you don’t need. This includes the words returned by **strtok**. However, you need to really understand how **strtok** works so that you are able to de-allocate these properly. Do not de-allocate the string you plan on returning!