CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

# C PROGRAMMING

## Assignment 1: Changing Words Game

### BACKGROUND.

A word can be represented as a sequence of letters and, by changing some of these letters, we can turn an original word into a new target one.

In this assignment we are going to create a C application allowing the user to interact with the letters of a word. The goal of the game will be for the user to turn the original word into the target one given a maximum amount of movements/letter changes supported.

### INTERACTIVE SESSION.

To represent the “logic” of our game we are going to use the following variables:

- A char array `current_word`, representing the letters of the original word (e.g., `char current_word[4] = {‘b’, ‘e’, ‘l’, ‘l’};` represents the word *bell*.
- A char array `target_word`, representing the letters of the word we want to come up with (e.g., `char target_word[4] = {‘c’, ‘o’, ‘a’, ‘t’};` represents the word *coat*.
- An char pointer `letter_ptr`, representing the address of `current_word` we are pointing at (e.g., if `current_word` has 4 letters, `letter_ptr` can be pointing at `&current_word[0]`, `&current_word[0]`, `&current_word[0]` or `&current_word[3]`.
- An integer `game_over` representing the status of the game:
    - `game_over = 0` --> The game is still on (game_over = 0).
    - `game_over = 1` --> The game is finished with a win; the user has turned the original word into the target word.
    - `game_over = 1` --> The game is finished with a loose; the user has reached the maximum amount of movements without turning the word.
- An integer `max_movs`, representing the maximum amount of movements/letter changes allowed before finishing the game. This value is equal to the double of letters `current_word` and `target_word` contain.
- An integer `num_movs`, representing the amount of movements/letter changes we have applied so far since the start of the game.
  The set of commands supported is:

- Command ‘>’: It moves letter_ptr to point to the next address/position of current_word (e.g., if letter_ptr was pointing to &current_word[1] now it will be pointing to &current_word[2]. If letter_ptr was pointing to the very last position of current_word then the command ‘>’ does nothing.
- Command ‘<’: It moves letter_ptr to point to the previous address/position of current_word (e.g., if letter_ptr was pointing to &current_word[2] now it will be pointing to &current_word[1].
- If letter_ptr was pointing to the very first position of current_word then the command ‘<’ does nothing.
- Command ‘a-z’: It turns the letter of current_word that letter_ptr is pointing at into the new letter typed. If the new letter typed is the same letter_ptr was pointing at, then the command does nothing.

Given current_word = {‘b’, ‘e’, ‘l’, ‘l’} and target_word = {‘c’, ‘o’, ‘a’, ‘t’} a possible sequence of movements in the interactive session is described next:

```
------ Game Status -------
Target = coat

bell
-
Num Movements = 0
-------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
%
NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
&
NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
8
NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
c
```

```
------ Game Status -------
Target = coat

cell
-
Num Movements = 1
--------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
>

------ Game Status -------
Target = coat

cell
 -
Num Movements = 2
--------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
<

------ Game Status -------
Target = coat

cell
-
Num Movements = 3
--------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
<

------ Game Status -------
Target = coat

cell
-
Num Movements = 3
--------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >      <      'a'--'z'
>
```

```
------ Game Status -------
Target = coat

cell
 -
Num Movements = 4
-------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >       <       'a'--'z'
o

------ Game Status -------
Target = coat

coll
 -
Num Movements = 5
-------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >       <       'a'--'z'
>

------ Game Status -------
Target = coat

coll
  -
Num Movements = 6
-------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >       <       'a'--'z'
a

------ Game Status -------
Target = coat

coal
  -
Num Movements = 7
-------------------------

NEW MOVEMENT: Enter a valid command by keyword:
Valid commands: >       <       'a'--'z'
>
```

```
------ Game Status -------
Target = coat

coal
    -
Num Movements = 8
-------------------------


-------------------------
Game over: Target Word not reached after max movements. Try again
-------------------------
```

As we can see in the example:

- We show to the user the current status of the game. To do so, we print the current value of our logic variables. In the case of the array `target_word` and `current_word`, this means printing their letters one by one. In the case of the pointer `letter_ptr`, this means printing the symbol '_' under the position/address of `current_word` that `letter_ptr` is pointing to.
- Given the current status, we ask the user for a valid command introduced by keyboard. If the user introduces a non-valid command, we just ask again, until a valid command is introduced.
- Left (`<`) and right (`>`) commands only count as a movement if the pointer is actually moved (e.g., if `letter_ptr` is pointing at `&current_word[0]` and a left command `<` is introduced, then nothing happens; the command is accepted as it is a valid one, but neither `letter_ptr` nor `num_movs` are modified). Same with turning a letter, it only counts if the new letter is different from the previous one.
- When `current_word` is turned into `target_word` or `num_movs` reach `max_movs` the game is over (as a win and a loose, resp.) In the previous session the game ended with a loose as we reached the 8 movements, but a different way of playing would have led to a win!

```
------ Game Status -------
Target = coat

coat
  -
Num Movements = 7
-------------------------


-------------------------
Congratulations: Target Word reached after 7 movements
-------------------------
```

## ASSIGNMENT 1 (Weeks 5 and 6)

Stack array with fixed size. Fixed set of commands.

### GOAL.

Create a C program providing an interactive session like the one showed before. On it, the user can apply the valid commands to convert a fixed word of 4 characters into the target one.

### FILES.

### 1. main.c

This is the file containing the main entry point of the project. You **do not** need to modify it. You can play with different words of 4 letters if you want, but that's all.

### 2. word_game.h

This is an external resource interface imported from 'main.c'. It specifies all the functions of 'word_game.c' that can be used in 'main.c'. You **do not** need to modify it.

### 3. word_game.c

This is the file implementing all the functions of word_game.h. You **have to** implement it.

## ASSIGNMENT 1 – ADDITIONAL FUNCTIONALITY

Once your C application is fully working, you can focus on achieving the following additional functionality:

1. Support words of whatever (but predetermined) size.
2. At the end of the game, show the list of words being constructed.
3. After each new movement, notify the user of how many letters match target_word.

**MARK BREAKDOWN.**

Assignment 1: 35 marks.

- ask_for_new_command: 4 marks
- process_movement: 5 marks
- print_status: 5 marks.
- is_game_over: 5 marks.
- user_game_word: 5 marks.
- additional functionality: 11 marks.
  - Additional feature 1: 3 marks
  - Additional feature 2: 4 marks
  - Additional feature 3: 4 marks

To evaluate each function we will check that the code is generic enough to work with different words and that the pointers are correctly.

For each function, there are 5 possible scenarios:

A. The function is generic enough and the pointers are well used: 100% of marks.
B. The function is not generic enough or the pointers are not well used: 70% of marks.
C. The function does not work properly by some mistakes, but it compiles and does not generate any exception (make the program crash): 35% of marks.
D. The function was barely attempted, it does not compile or it generates and exception (makes the program crash): 15% of marks.
E. The function was not attempted: 0% of marks.

## SUBMISSION DETAILS.

### Deadline.

Sunday 10th of March, 11:59pm.

### Submission Details.

Please submit to Canvas (folder 6. Submissions, A01) the following file:

- File “word_game.c”.

### Lab Demo.

A brief individual interview about the assignment will take place on our lab session on Week 7. The demo is mandatory for the assignment to be evaluated.