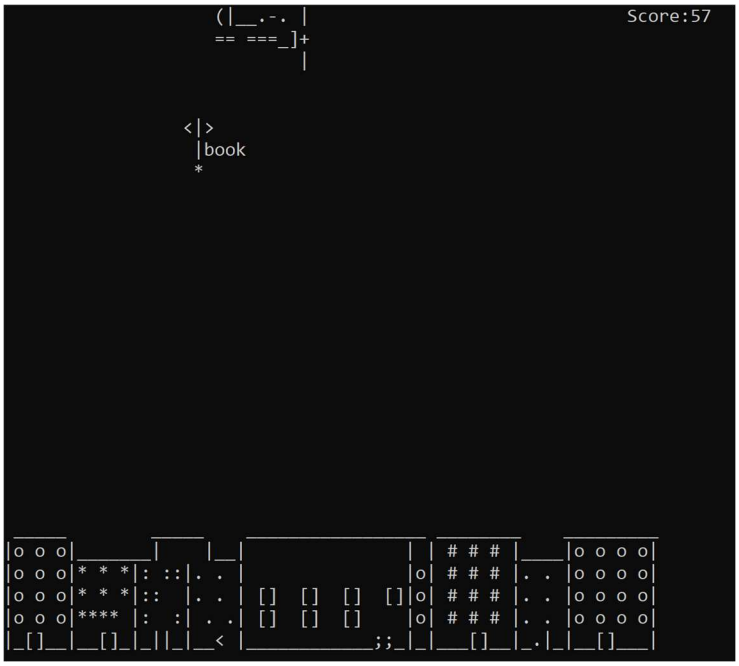
# EHB110E Final Project

## Typing of the Bombs




### Overview

In this project, you will implement a fun typing game in C. The aim of the game is to destroy the bombs dropped by a plane before they hit the city. To destroy a bomb, the user has to successfully type the deactivation code that appears next to each bomb.

### Game Play

The game screen will be drawn using ASCII characters as shown in the picture above. You may find the ASCII characters for the city, the plane and the bomb in txt files provided with the project.

The plane will move from left to right, starting over from the very first column of the screen as it reaches the score on the right edge.

The plane will drop bombs successively. There may be at most one bomb on the screen at once. If the user types the code associated with a bomb, that bomb will disappear and the plane will immediately drop the next bomb. Bombs will move vertically down once they are dropped (i.e., you may ignore the horizontal speed).

There will be a scoreboard displayed on screen at the top right corner as shown above. The scoreboard will display the current score of the user. The score will be updated each time the user types a codeword completely (not each time user types a letter). It will be updated according to the following formula:

**(New Score) = (Old Score) + (Number of letters typed correctly) – (Number of letters mistyped)**

If the user types a letter of the codeword correctly, the codeword appearing next to the bomb will shift one letter to the left displaying the remaining letters of the codeword only. For example:

book -> ook -> ok -> k

The bomb drop speed will increase as the user deactivates a multiple of 5 bombs successively. You can adjust the drop speed as you wish as long as the speed increases with increasing score.

The deactivation codes the user should type will be read from a file. The format of this file is up to you. A sample codeword txt file that you can use is provided with the project. Initially, the user will be presented 3 letter codes randomly from the file. The number of letters in a code will increase by one letter as the user deactivates a multiple of 5 bombs successively. For instance, when the user deactivates 5 bombs, bombs will start to have 4 letter random codewords; when the user deactivates 10, bombs will start to have 5 letter random codewords and so on up to 10 letter codewords. After 10 letters, the words will keep having 10 letters until the user loses the game.

The game will be lost if a bomb reaches the city. After the game is lost, a GAME OVER text will be printed to the middle of the screen and the user will be asked whether to start a new game or not. If the user presses, “y”, the game will start over. The codewords will start having 3 letters and bombs will drop at the slowest speed again. If the user presses, “n” the game will exit to the menu (see below).

## Game Menu

The game will have a menu with the following options:

1. **New Game**
2. **Load a Saved Game**
3. **Save Current Game**
4. **Return to Game**
5. **Exit**

The menu will appear at the entry to the game or anytime the user presses the **ESC** key during the game.

The **“New Game”** option will reset the game state and will start a new game from scratch.

The game will be able to save/read its state to/from a file. The state will be kept in a C struct and will be written to or read from the file.

The **“Load a Saved Game”** option will ask the user for a filename that holds the previously saved game state. After the user types the filename and presses enter, the game state will be restored from the file and the game will start from that state. If the user types an incorrect filename, the game will prompt the user with the message “No such file” and will return to the menu.

The **“Save Current Game”** option will ask the user for a filename to hold the game state to be saved. After the user types the filename and presses enter, the game state will be saved to the file and the game will return to the menu.

If the user chooses the **“Return to Game”** option, the menu will disappear and game will continue. This option will be visible only if ESC key is pressed (i.e., user interrupted a running game to enter the menu). It will not be visible when the game is first launched.

The **“Exit”** option will quit the game without saving it.

## Implementation Notes

Below are the implementation notes for Visual Studio on Windows. For other platforms, you may use equivalent functions.

You will need to clear the screen completely before making an update to the game screen and then draw the entire screen again. In order to clear the screen, you may use the following C statement:

```
system("cls");
```

In order to read a character from the keyboard without blocking the game, you may use the following code piece:

```
while (_kbhit()) {
   char c = _getch();
}
```

**_kbhit()** function checks if a key is pressed on the keyboard and returns true if so. **_getch()** function returns the ASCII code for the pressed key. Note that you cannot use scanf function for this purpose because scanf will wait for the user input indefinitely during which you will not be able to update the game screen.

In order to be able to use _kbhit() and _getch(), you need to add the **<windows.h>** header file.

## Deliverables

1. Your submission should include:

a. Your C source file

b. Your codeword file

c. Your project report in PDF format

Please use the provided project template for your project report.

**You should not include any other file than above.**

2. Your source code should be commented. Uncommented code will get partial credit.

3. Your source code should be modularized using multiple functions. Do not write the entire code in main(). Try to implement separate functions for tasks like drawing the screen, printing the menu, reading user input, updating the score, etc.

3. Zip your files and send them as a single zip file. Name the zip file with your full name such as Ali_Bilir.zip. Do not send the files individually.