**Retaj Tarek 20010606**

**Rowan Ashraf Ali 20011312**

**Hassan Nader Hassan 20010492**

**Haneen Ahmed Shabaan 20010505**

**Rowan Tarek Elsayed 20010586**

**Mohamed Khamis Awad 20011560**

**Youssef Galal Mohamed 20012259**

|  |  |
| --- | --- |
| Hangman project  Under the supervision of Doctor Mohamed Alhabrouk | Abstract  Hangman is a classic word game in which you must guess as many secret words as you can before your trials run out.  Most people would have played the game Hangman when they were kids as a fun way of learning words and improve spelling. |

C-project

27-Hangman Game

Introduction:

Hangman game is a game in which we store ten words in an array in the code (hardcoded array) and the user (player) will choose the order of a word from the ten words and then there will be dots for each letter will appear on the screen ,also gallows will be drawn and we will ask the user to guess the letters of the word with order he has chosen ,the user will have 10 trials to make wrong attempts for each wrong attempt part of the hanged man will be drawn part by part from hair to legs and there will be list of the user’s wrong attempts will appear on the screen to make sure the user don’t choose any of these letters again , if the user succeeded to guess the word before finishing the trials so he won the game but , if the user failed to guess the word , so the hanged man will be drawn then the game is over, then the word will appear on the screen and we will ask the user if he wants to complete playing the game , the he should press “1” , and if he wants to close the game , he should press “0” and that is the game in a simple way.

Why did we choose this project?

Because some of us played this game in our childhood and because it’s a purposeful game as it helps us to improve our understanding of words, pronouncing the word, and using the word based on the context given, it’s also useful for young people where it puts them under the pressure of helping the man to not be hanged so, it improves the ability to work under pressure.

While making the code we faced plenty of problems, of course, we will talk about them in the problem analysis and we will discuss each of them in the algorithm, flowchart and the bugs faced us while performing the code, we will talk in the next topic about the problem analysis.

Problem Analysis:

We here have some problems that need to be solved in writing the code we will talk about them in the following lines:

We should firstly make the outline of the game as we should welcome the user and notify the user about the rules of the game and start drawing the gallows, Show the output after each trial by showing the place of the letter he inputted or showing the number of remaining trials and the list of wrong attempts he made and so on. All of that will be made in the code by using (printf) function, we should also store the 10 words which the user will guess it’s words, store the wrong letters to screen them, and compare the letter which the user entered and the letters of the chosen word and these problems will be solved using (arrays) and (functions), we should limit the order that user enters to be less than 10 and greater than 1, make sure that user didn’t choose the same order he has chosen before in his last trials if he played more than once, limit the inputs of the user in the letters to make him input only letters and not numbers or any other character, compare each letter the user inputs with the letters of the chosen word, draw new part of the hanged man on the gallows after each wrong attempt made by the user, all of that will be made by (conditional if and switches) by using them many times,

We also should increase the counter each time the user inputs a letter, add the wrong letters together, compare the inputted letter with the letters in the chosen word, decrease the number of trials after each wrong attempt the user made, and get the right order of the word and replace each letter with dot and screen it to the user, if the user inputs a right letter, place these letters in their right places and not to decrease the number of trials and all of that can be made by (looping either by while or for-loop).

All these cases, it’s a solution and the code will be discussed briefly in the next parts of the report in the flowchart of the code, the algorithm, source of code, and edition made to it.

Algorithm of Hangman game

**#CHOOSE A RANDOM WORD**

\_ The user should be asked to enter the order of the word he wants to guess.

\_The order must be between 1 and 10.

\_ The chosen word includes only letters (no numbers, symbols, or whitespace).

\_ The chosen word should be selected with respect to the order which user enters from a predefined hardcoded array in the database of the program {"equip","wizard","jumbo","zipper"scratch","pixel","luxury", "stretch”,"ivory","matrix"};

chosen\_word = hardcoded.sample

puts chosen\_word

**#SHOW WORD PROGRESS**

\_ Each round the word is displayed (e.g. Word: ..mm.t)

\_ Guessed letters are unmasked.

\_ Hidden letters are shown as dots.

**#SHOW REMAINING GUESSES**

\_ Each round display the number of chances remaining.

\_ Display Chances remaining: CHANCES where CHANCES is an integer.

**#PROMPT FOR GUESS**

\_ Each round, the player is prompted for input.

\_ The player's input should appear on the same line (no newline after the prompt).

**#GUESS LETTER**

\_ When a player enters a single character, it guesses a letter.

\_CHECK GUESSED LETTERS

\_ If the guessed letter is within the word, those letters are unveiled.

\_ Display the number of times the letter is found when it is present.

\_ If the guessed letter is not within the word, subtract one from the chances remaining.

\_The hangman details are displayed one by one every wrong guess

\_The wrong alphabet will be stored in an array of characters.

\_Each wrong attempt the wrong letters will be displayed on the screen.

\_ The user is alerted if the guess is not found.

\_ If the guessed letter was already submitted, do not decrement the chances remaining.

**#User’s win**

\_If the user guessed all the letters of the word correctly and doesn’t take all the opportunities then the user becomes the winner.

**#GAME OVER WHEN NO MORE CHANCES**

\_The game ends when the number of chances remaining reaches zero.

\_ The user is alerted that they have run out of guesses

\_The game displays the word to the user.

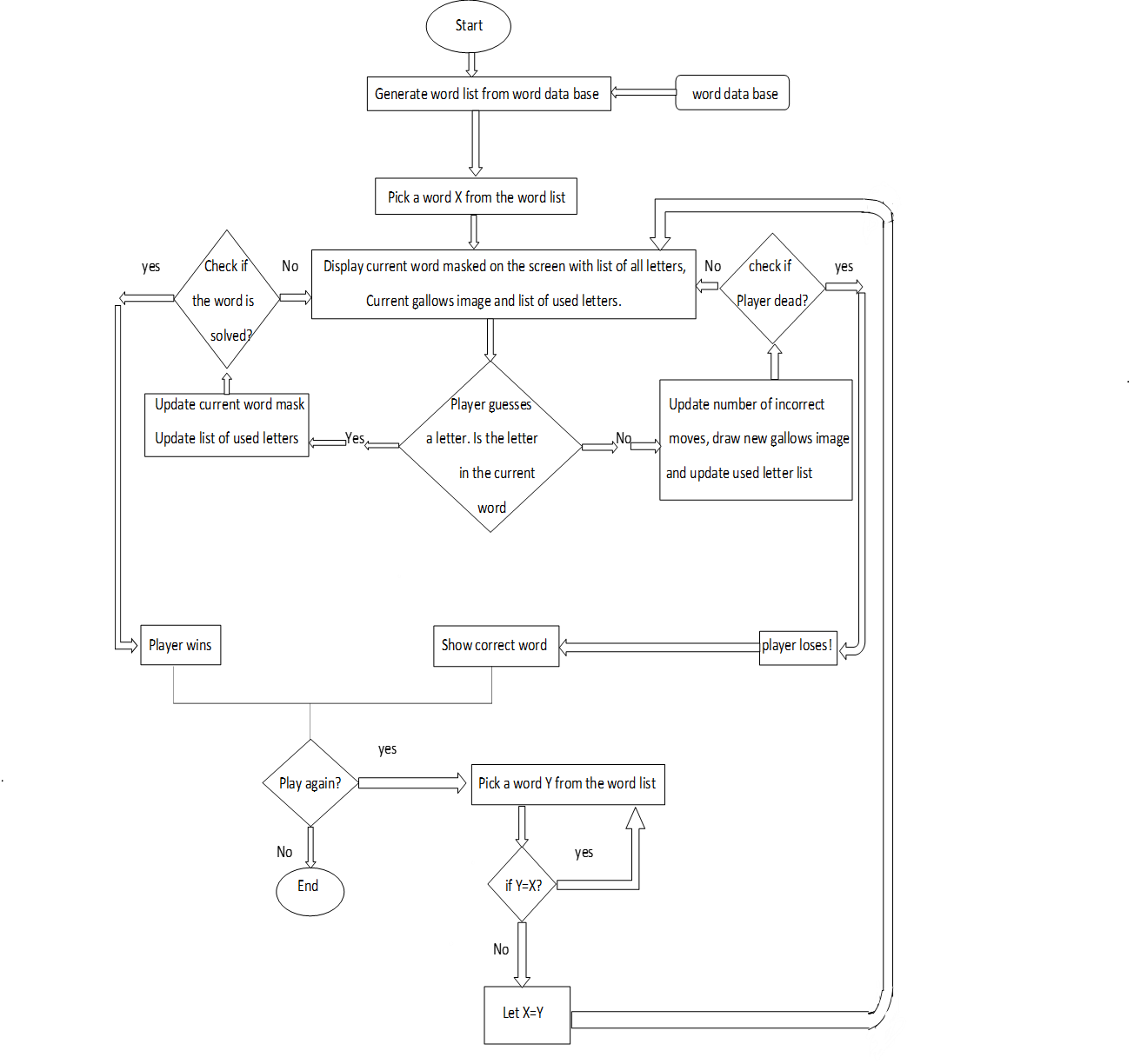
**#In case of playing more than once**

\_In case of winning or losing, the game asks the user if he wants to play again.

\_As the user choose to play again, he won’t be able to choose the same word from the previous game.

\_If he chooses no, the game will end.

Program Flowchart:



Bugs encountered during implementation and solutions used to overcome them:-

The C programming language sometimes gets a bad reputation because it is not memory safe like more recent programming languages. But with a little extra code, you can avoid the most common and most serious C programming bugs. So in the upcoming section, we will be familiar with some problems we have encountered during the implementation of our code:

## 1. Overflowing a string

Strings are just arrays of a different kind. In the C programming language, a string is an array of char values, with a zero character to indicate the end of the string.

And so, like arrays, you need to avoid going outside the range of the string. This is sometimes called overflowing a string.

Some easy way to overflow a string is to read data with the scanf() function. we can also use gets function. The gets function is very dangerous because it doesn't know how much data it can store in a string, and it naively reads data from the user. This is fine if your user enters short strings like foo but can be disastrous when the user enters a value that is too long for your string value.

It’s important for us to determine the size of our input string to avoid string overflow. So we can declare a length of string bigger than the expected length that will be entered by the user. We can do that using direct declaration or using macro function #define.

## 2. Logical Error

The logical error is considered the most common error faced by programmers and the hardest one to be detected. Everything looks like it is working; you have just programmed the computer to do the wrong thing. Technically the program is correct, but the results won’t be what you expected.

If you didn’t check the requirements beforehand and wrote code to return the oldest user in your system when you needed the newest, you would have a logic error.

For example, in our “hangman” game, it was displayed to the user that he/she has only 10 tries to guess the unknown word. However, in executing the program it was noticed the user in fact has 11 attempts not 10. So that caused a logical error.

In order to overcome Logical error, we can also use many /\*comments\*/ in program code to help us detecting the error easily, we can also Break our code into small chunks. Our methods should say what they’ll do. Write tests to make sure your chunks work. If you have a logic program, you should write a test that breaks. So Look at your code. Does it make sense? Fix the code. Run your tests. Make sure you didn’t break anything else. Make sure your broken test fails.

Therefore, it’s very important to for us to understand what our code does and concentrate while writing it, in order to we can avoid logical errors.

3. Uninitialized variables

When the program starts up, the system will assign it a block of memory that the program uses to store data. That means your variables will get whatever random value was in memory when the program started.

So if our used variable isn’t assigned to a specific value then we can face undetectable results or it can take a garbage value.

Some environments will intentionally "zero out" the memory as the program starts up, so every variable starts with a zero value. And it can be tempting to assume in your programs that all variables will begin at zero. However, the C programming specification says that the system does not initialize variables.

In order to overcome this problem, we can assign that variable to a specific value as zero then we can change its value during our code if we want by for asking the user to assign a new value for it, or we can also use macro function #define if we aren’t going to change its value during the program.

## 4. Compilation Errors

Some programming languages require a compilation step. The compilation is where your high-level language converts into a lower-level language that the computer can understand better. A compilation of compile-time error happens when the compiler doesn’t know how to turn your code into the lower-level code.

In our syntax error example, if we were compiling print(“welcome”, the compiler would stop and tell us it doesn’t know how to convert this into a lower-level language because it expected a ‘ ) ’ after the ‘ ” ’.

If there is a compile-time error in your software, you won’t be able to get it tested or launched.

Like syntax errors, you will get better at avoiding these with time, but in general, the best thing you can do is get early feedback when it happens.

Compilation happens across all files of your project at the same time. If you’ve made lots of changes and see lots of compiler warnings or errors, it can be very daunting. By running the compiler often, you will get the feedback you need sooner, and you will more easily know where to address the issues.

## 5. Runtime Errors

Runtime errors happen as a user is executing your program. The code might work correctly on your machine. But, there might be a different configuration.

If your system took the input from a form and tried to capitalize the first letter of a name by doing something like params[:first\_name].capitalize, this would break if the form was sent without a first name.

Runtime errors are particularly annoying as they directly impact your end user. A lot of these other errors will happen when you’re working on the code. These errors occur when the system is running and can stop someone from doing what they need to do.

Make sure you have good error reporting in place to capture any runtime errors and automatically open up new bugs in your [ticketing system](https://textexpander.com/blog/write-better-issue-tracking-tickets-consistency-is-key/). Try and learn from each bug report so that in the future you can.

## 6. Syntax Error

Just like human languages, computer languages have grammar rules. But while humans are able to communicate with less-than-perfect grammar, computers can’t ignore mistakes, i.e. syntax errors.

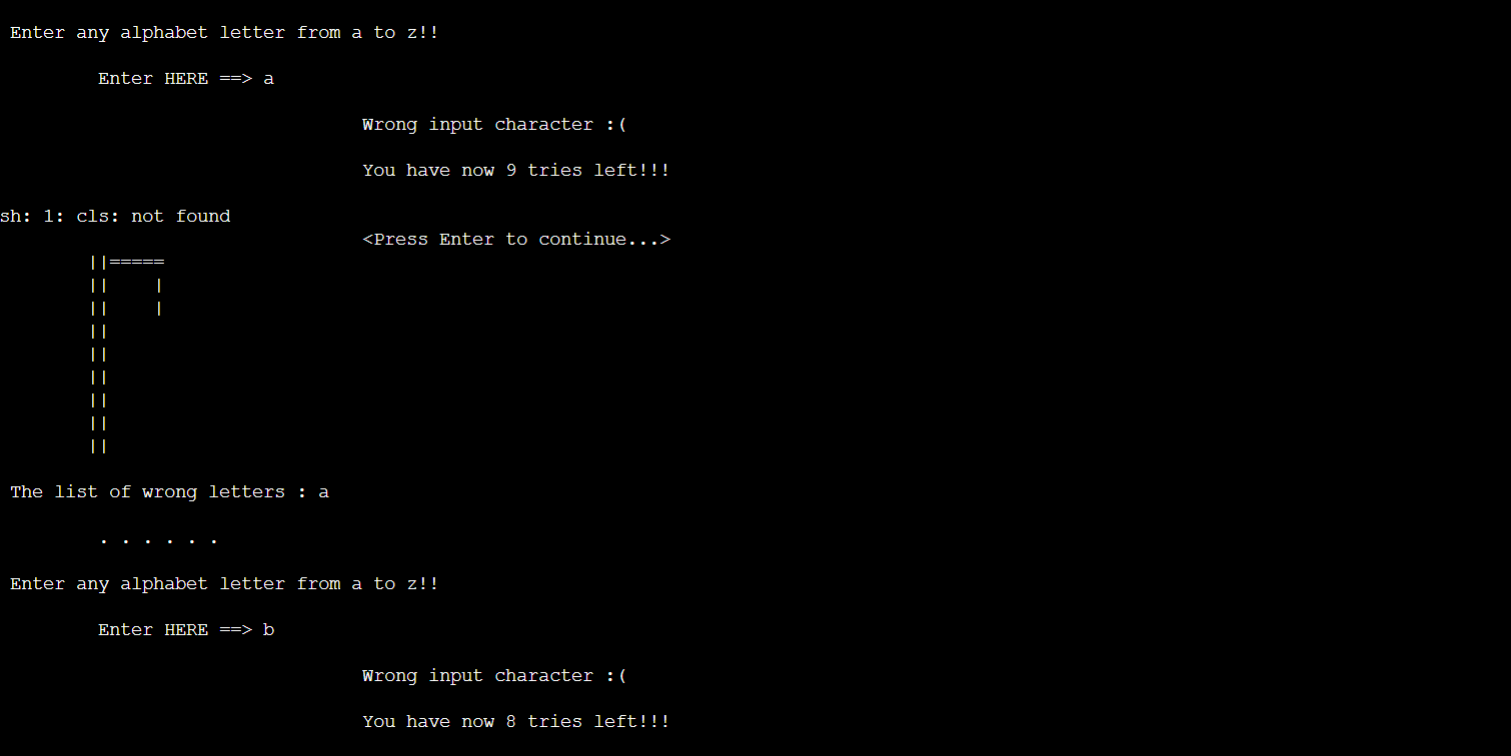
For example, let’s say the correct syntax for printing something is printf(‘welcome’), and we accidentally use single coots instead of double coots while coding, or also if we forgot a semicolon at the end and line. A syntax error will happen, and this will stop the program from running.

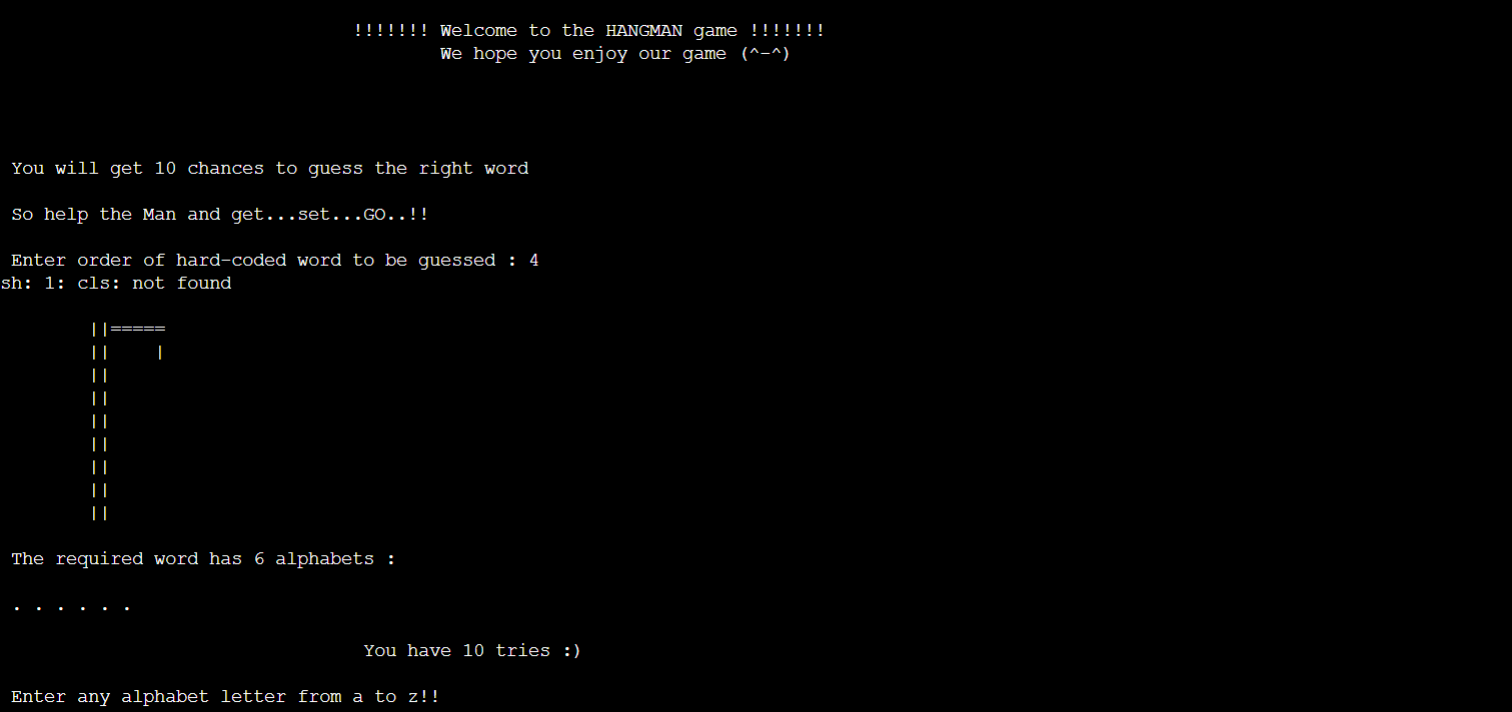
As your proficiency with programming language increases, you will make syntax errors less frequently. The easiest way to prevent them from causing you problems is to be aware of them early. Many text editors or IDEs will come with the ability to warn you about syntax errors at the time of writing it. It’s also recommended to run the code constantly after writing each part of the program.

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

Sample input test cases and corresponding outputs:

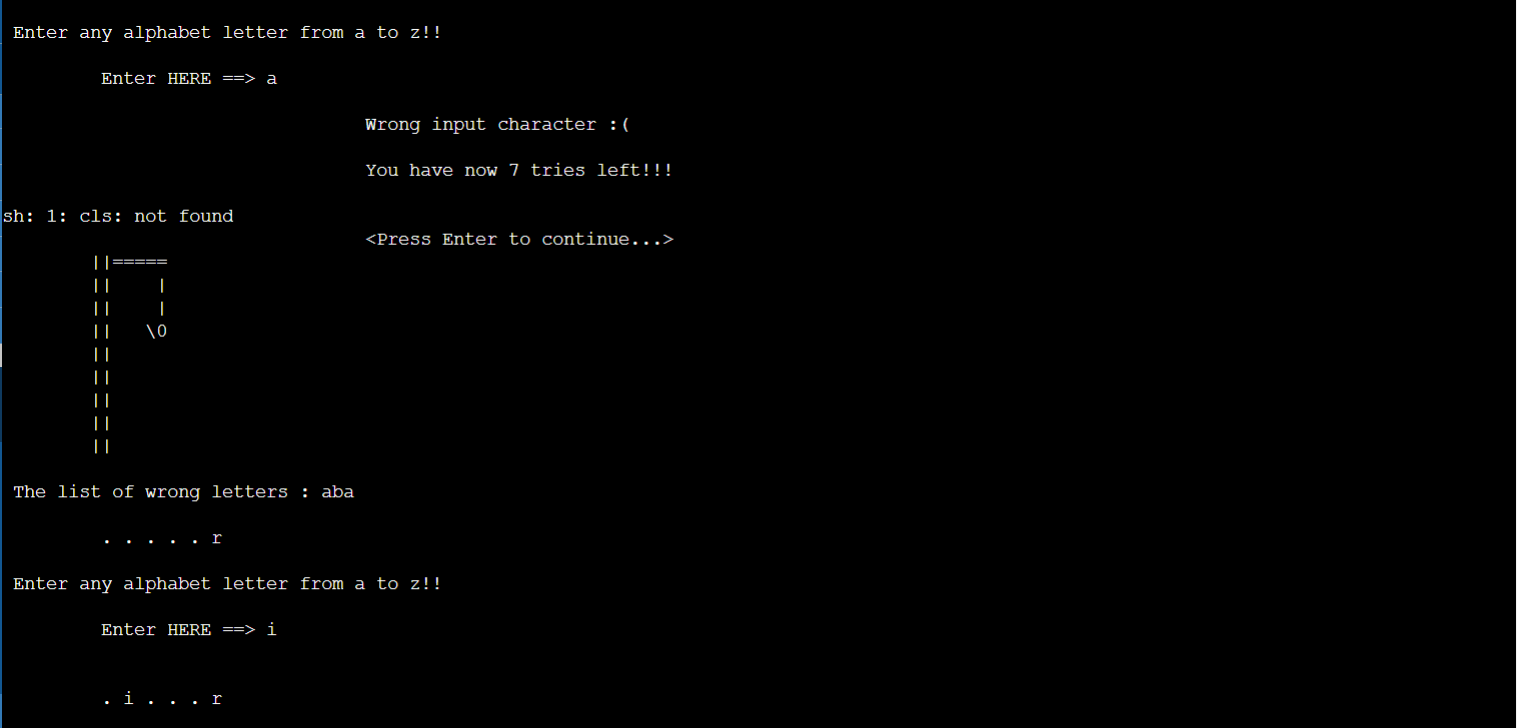
Firstly: -

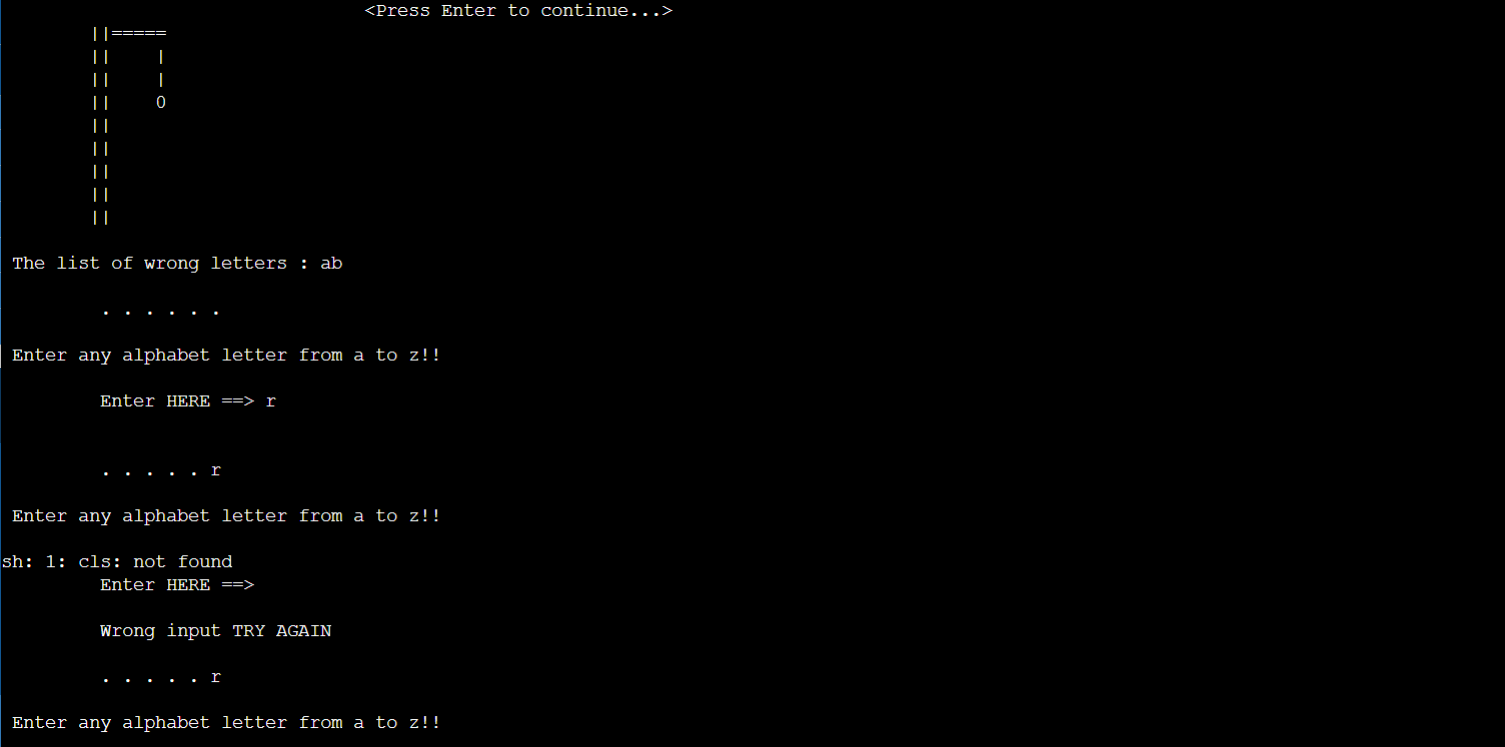
-The user is asked to enter the number of the word He would like to guess, and He has 10 trials afterwards.

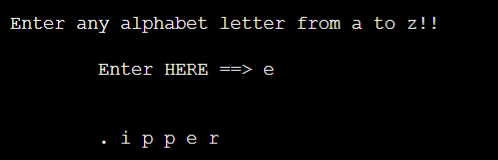
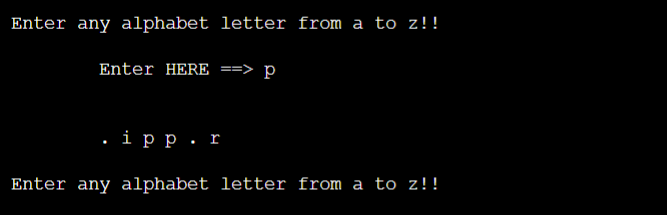


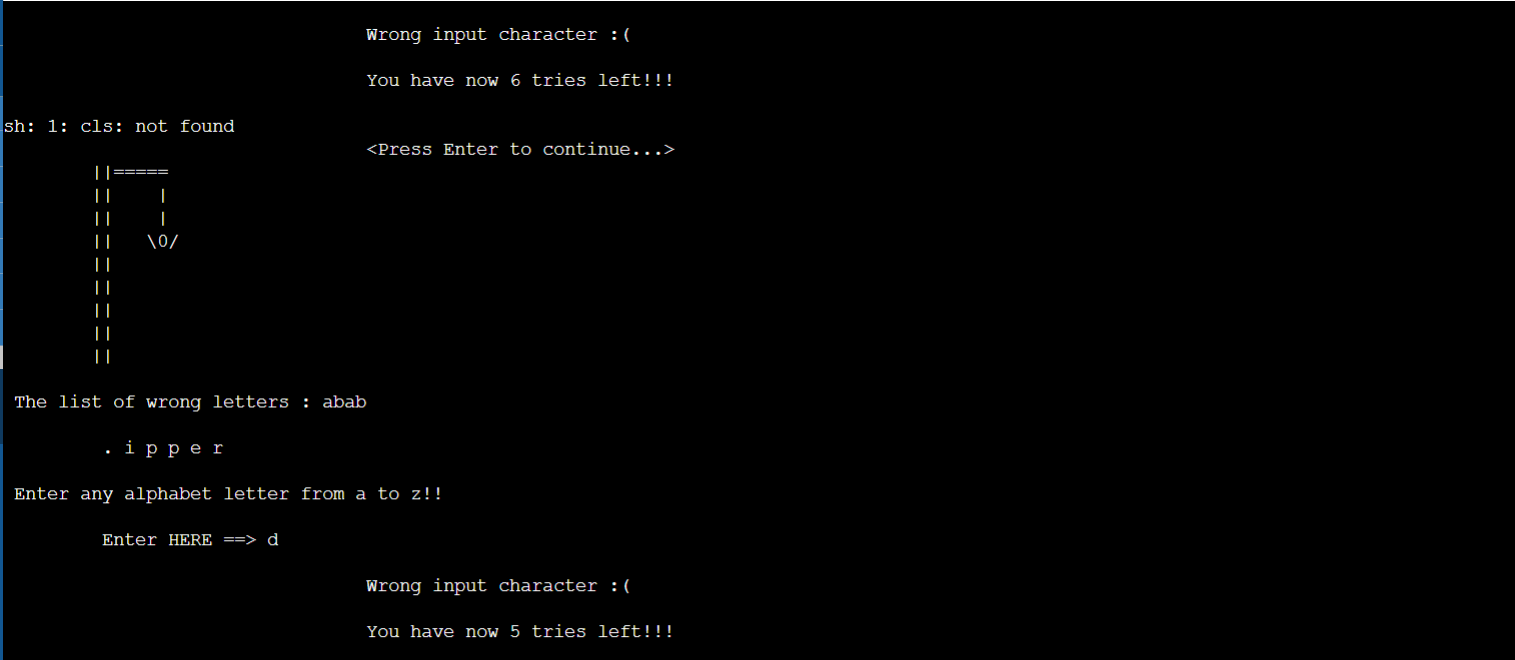
The user Entered a wrong character so the program so has less number of trials and the program outputted “Wrong input Character”

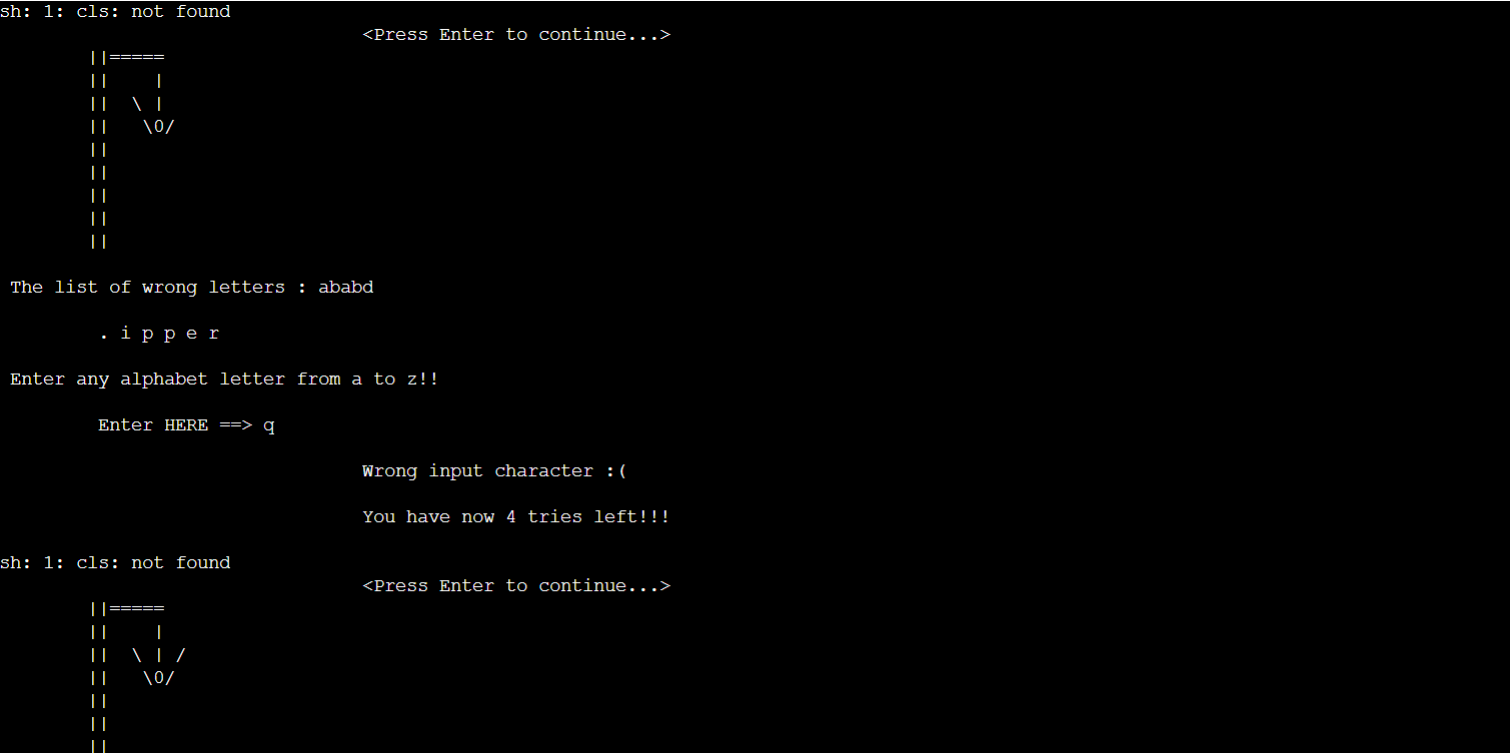
A list of wrong entered characters is displayed

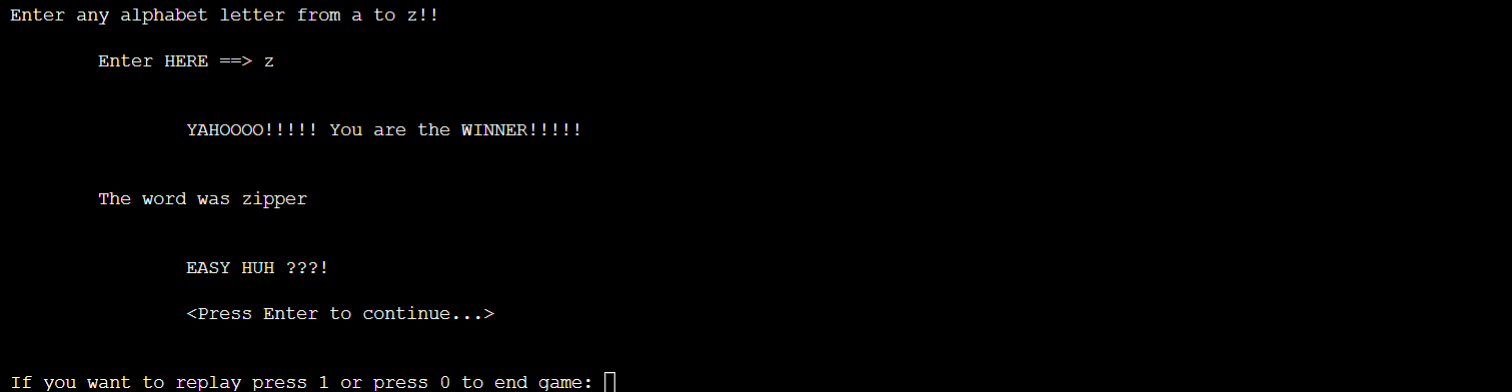
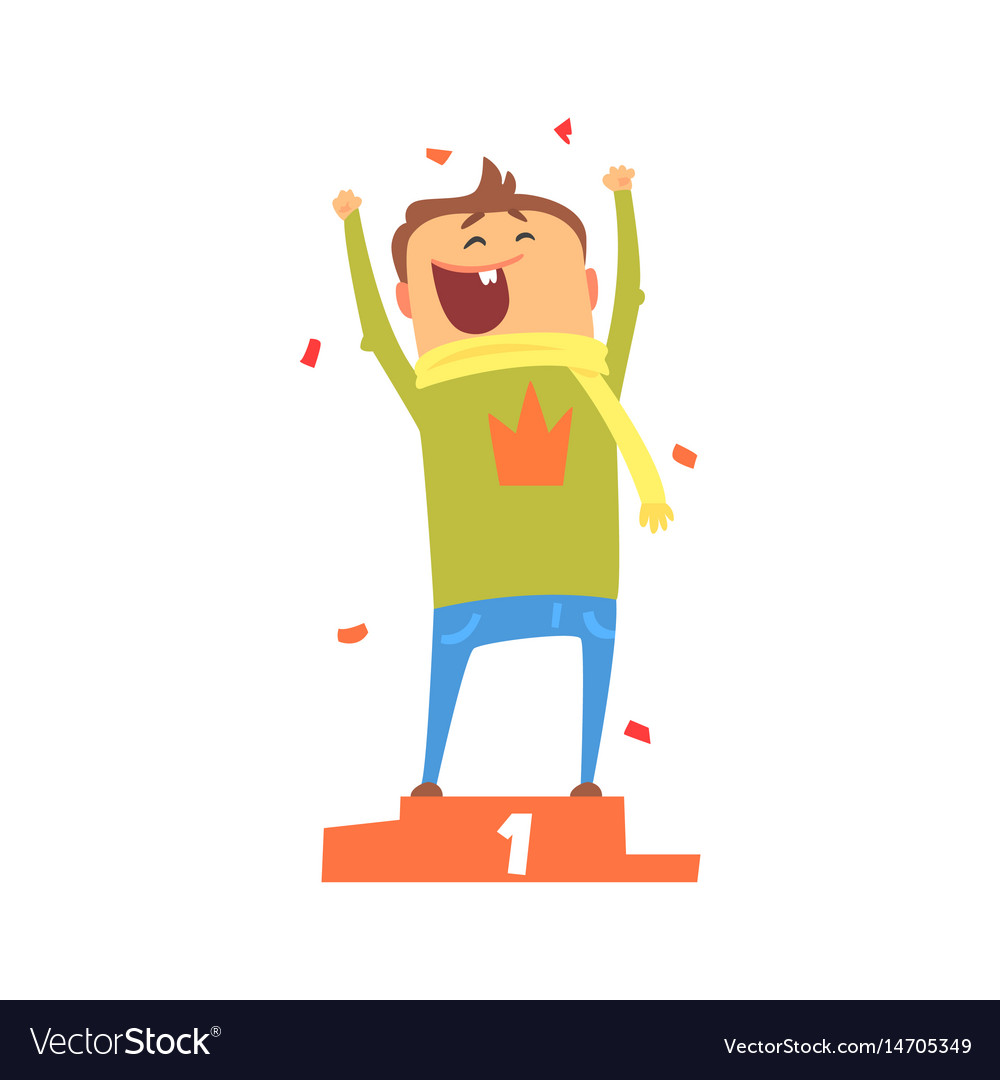
The user entered a correct character in these successive attempts, so The letters are stored in their correct place.











Future improvements:

-Refactoring our code to make it more orderly and easier to update.

-Accepting just one-letter input.

-Expanding our dictionary to include much more words.

-Language diversity in future editions.

-Incorporating graphics into our game to encourage users to play it more frequently.

-Including difficulty levels and online playing to create a more competitive atmosphere

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

References:

* <https://github.com/tintinmj/Hangman-game/blob/master/game.c>
* <https://gist.github.com/c13iscuit/e67c6f992cbfb8b316ab>
* <https://textexpander.com/blog/the-7-most-common-types-of-errors-in-programming-and-how-to-avoid-them>
* <https://opensource.com/article/21/10/programming-bugs>
* <https://www.b4x.com/android/forum/threads/hangman-tutorial-part-1.32893/#post-191861>