Hangman Game

Data Structures Mini project

Prepared for

Dr. Kiran Sharma

Assistant Professor

School of Engineering and technology

By

Group 26

Radhika Bhati

Khushi

BML Munjal University

Kapriwas, India

Acknowledgement

We'd like to show my appreciation to everyone who has assisted us with this project.

First and foremost, we'd want to thank to Dr. Kiran Sharma for her assistance and support. Her assistance was critical to the effective completion of this project.

We'd also like to express our gratitude to our friends for their encouragement and support during this effort. Their encouraging comments and inspiration kept me going when we faced difficulties.

Finally, We want to thank everyone who helped, whether directly or indirectly. Your assistance and participation were tremendously valued.

Thank you for being a part of this effort and for your ongoing support.

Table of Contents

Contents

01. PROBLEM STATEMENT

This project's problem statement is a simple implementation of the game "Hangman". The goal of the game is to guess a word letter by letter before running out of attempts. The 'ARRAY' data structure is used in the game implementation.

The game begins when a word is chosen at random from a pre-defined list of animal names. The word's length is then utilised to construct a secret word, which is presented as a series of asterisks (*).

The player must guess a letter, which is then compared to the letters in the word chosen by computer. The game continues until either the player successfully guesses the word or the player runs out of attempts.

RULES

- Each turn, the player must guess a letter.
- The player can make no more than five inaccurate predictions before losing the game.
- The word is chosen at random from a list of animal names.
- The length of the word is shown as a series of asterisks.
- If the player correctly guesses a letter, it is revealed in the hidden word in the corresponding positions.
- If the player guesses a letter incorrectly, the number of remaining attempts is reduced.
- When the player either correctly guesses the word or runs out of attempts, the game is over.
- The player wins the game if they properly guess the term.
- If the player runs out of attempts before correctly guessing the word, the game is over and a hangman figure appears on the screen.

This game could be implemented using a variety of data structures:

- ❖ Linked List: Instead of an array, a linked list could be used to store the words. Each linked list node might represent a word and provide a pointer to the next node in the list.
- ❖ Trees: The words could be stored in a hierarchical form using a tree data structure. Each node in the tree may represent a letter in the word, and the branches could represent the possible next letters.

02. ALGORITHM

Inputs:

- A list of words to choose from
- Maximum number of tries allowed
- Player's input (letter guess)

Outputs:

- Secret word with the guessed letters filled in
- Number of remaining guesses
- End of game message with the result

Algorithm:

01. Start the program.
02. Define the constant MAX_TRIES as 5.
03. Declare the necessary variables: name, letter, num_of_wrong_guesses, word, and words.
04. Initialize the array words [ ] with animal names.
05. Generate a random number between 0 to 9 using rand() function and assign it to variable n.
06. Retrieve the word from the words array using the index n.
07. Initialize the unknown string with * character of the same length as the word.
08. Display a welcome message to the user and provide instructions for the game.
09. Start a loop to accept user inputs until the num_of_wrong_guesses becomes equal to MAX_TRIES.
10. Inside the loop, display the current state of the unknown string.
11. Prompt the user to enter a letter and read it into the letter variable.
12. Call the letterFill() function with letter, word, and unknown as arguments.
13. If the function returns 0, the guess was incorrect, so increment num_of_wrong_guesses by 1 and display a message to the user accordingly.
14. If the function returns a positive value, the guess was correct, so display a message to the user accordingly.
15. Display the number of remaining guesses left to the user.

16. Check if the user has guessed the word by comparing word and unknown.
17. If the user has guessed the word, display a message to the user and break out of the loop.
18. If the user has used all the guesses, display a losing message to the user along with the word.
19. End the program.

Here's the algorithm for the letterFill() function:
01. Declare and initialize the necessary variables: i, matches, len.
02. Start a loop to iterate through each character of the secret word.
03. If the guess is already present in the guess word, return 0.
04. If the guess matches a character of the secret word, replace the corresponding character in the guess word with the guess and increment matches.
05. Return 1 and function ends.

03. C++ CODE IMPLEMENTATION

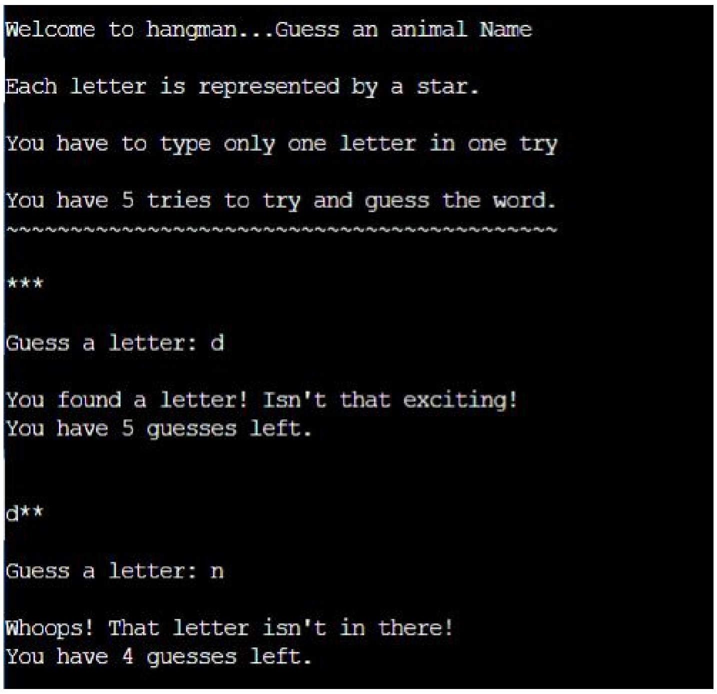
```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int MAX_TRIES=5;
int letterFill (char, string, string&);

int main ()
{
        string name;
        char letter;
        int num_of_wrong_guesses=0;
        string word;
        string words[] =
        {
                "elephant",
                "dog",
                "cat",
                "zebra",
                "monkey",
                "horse",
                "rabbit",
                "hippopotamus",
                "wolf",
            "kangaroo"

        };


        srand(time(NULL));

        int n=rand()% 10;

        word=words[n];


        string unknown(word.length(),'*');


        cout << "\n\nWelcome to hangman...Guess an animal Name";

        cout << "\n\nEach letter is represented by a star.";

        cout << "\n\nYou have to type only one letter in one try";

        cout << "\n\nYou have " << MAX_TRIES << " tries to try and guess the
word.";

        cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~";


        while (num_of_wrong_guesses < MAX_TRIES)

        {       cout << "\n\n" << unknown;

                cout << "\n\nGuess a letter: ";

                cin >> letter;


                if (letterFill(letter, word, unknown)==0)

                { cout << endl << "Whoops! That letter isn't in there!" << endl;

                  num_of_wrong_guesses++;

                }
```

```
            else
            { cout << endl << "You found a letter! Isn't that exciting!" << endl;
            }

            cout << "You have " << MAX_TRIES - num_of_wrong_guesses;
            cout << " guesses left." << endl;

            if (word==unknown)
            {       cout << word << endl;
                    cout << "Yeah! You got it!";
                    break;
            }
    }
    if(num_of_wrong_guesses == MAX_TRIES)
    {       cout << "\nSorry, you lose...you've been hanged." << endl;
            cout << "   \n\
           --------       \n\
           |     |       \n\
           |     O       \n\
           |    \\|/      \n\
           |     |       \n\
           |    / \\      \n\
           -              \n",
            cout << "The word was : " << word << endl; }
    return 0; }
```

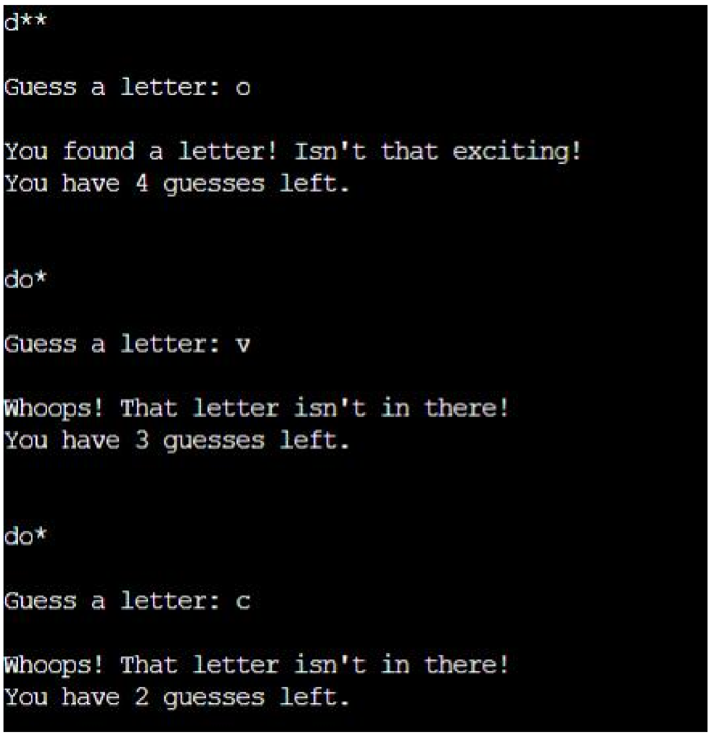
```
int letterFill (char guess, string secretword, string &guessword)
{        int i;
         int matches=0;
         int len=secretword.length();
         for (i = 0; i< len; i++)
         {

                  if (guess == guessword[i])
                           return 0;
                  if (guess == secretword[i])
                  { guessword[i] = guess;
                    matches++;
                  }
         }
         return  matches;
}
```

04. TEST CASES

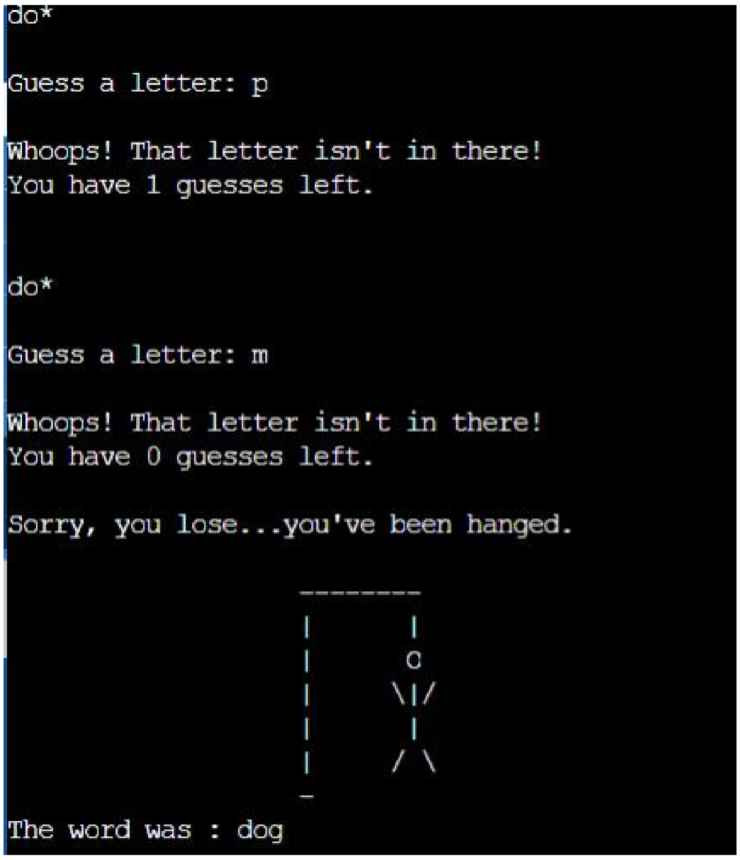
TEST CASE 1:



```
Welcome to hangman...Guess an animal Name

Each letter is represented by a star.

You have to type only one letter in one try

You have 5 tries to try and guess the word.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

***

Guess a letter: d

You found a letter! Isn't that exciting!
You have 5 guesses left.


d**

Guess a letter: n

Whoops! That letter isn't in there!
You have 4 guesses left.
```



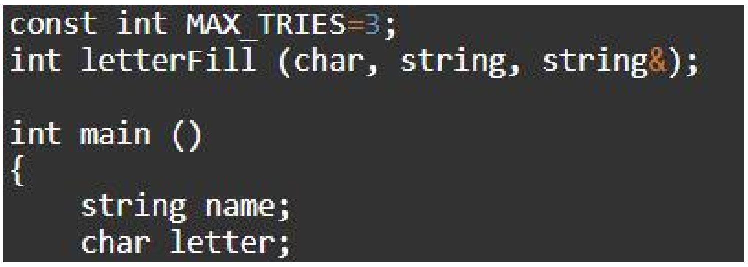
```
d**

Guess a letter: o

You found a letter! Isn't that exciting!
You have 4 guesses left.


do*

Guess a letter: v

Whoops! That letter isn't in there!
You have 3 guesses left.


do*

Guess a letter: c

Whoops! That letter isn't in there!
You have 2 guesses left.
```



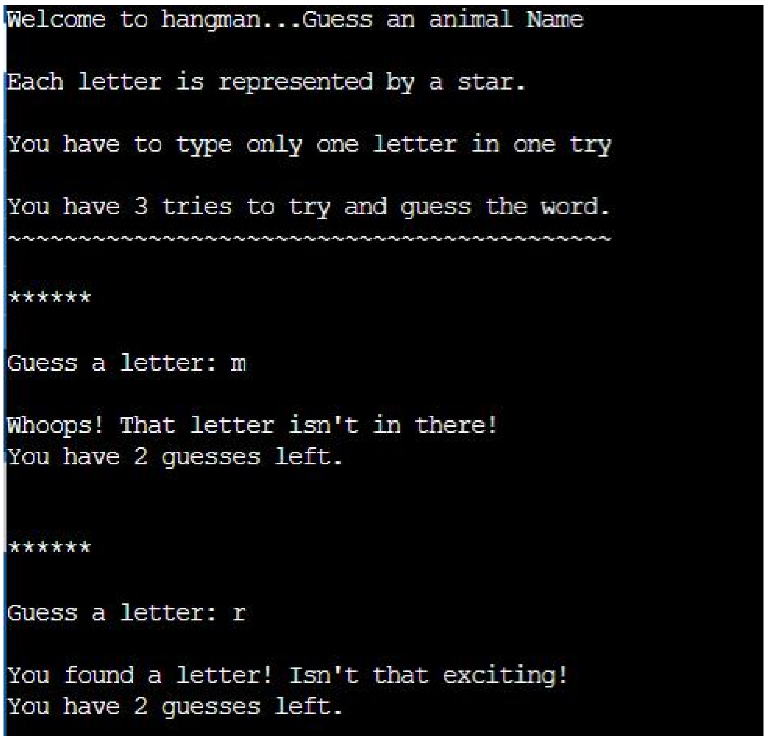
```
do*

Guess a letter: p

Whoops! That letter isn't in there!
You have 1 guesses left.


do*

Guess a letter: m

Whoops! That letter isn't in there!
You have 0 guesses left.

Sorry, you lose...you've been hanged.

                  --------
                  |      |
                  |      O
                  |     \|/
                  |      |
                  |     / \
                  _
The word was : dog
```

TEST CASE 2: We now modify the code and change the number of tries to 3 to increase the difficulty level of the game.

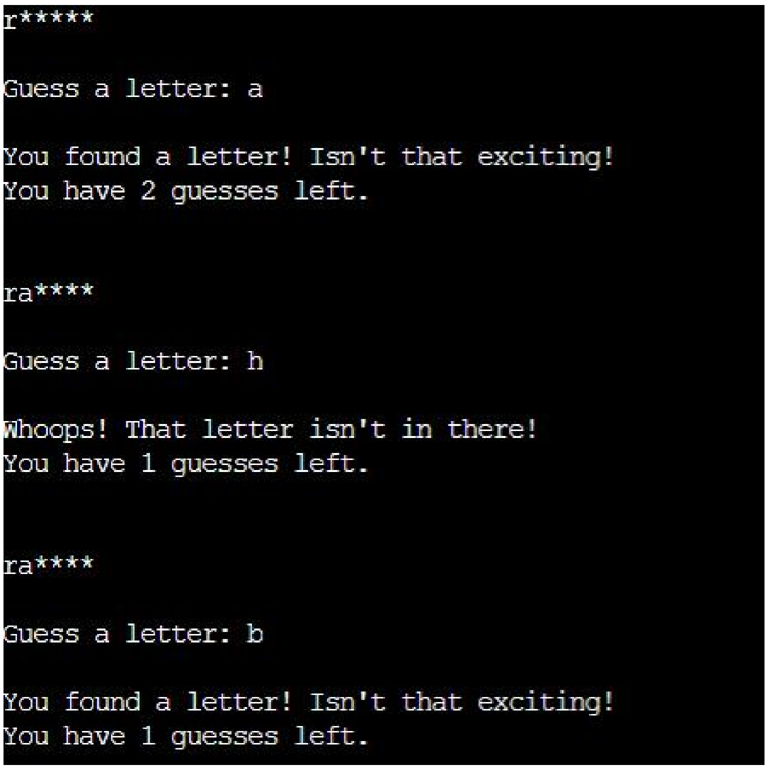
Code snippet of the edited part –



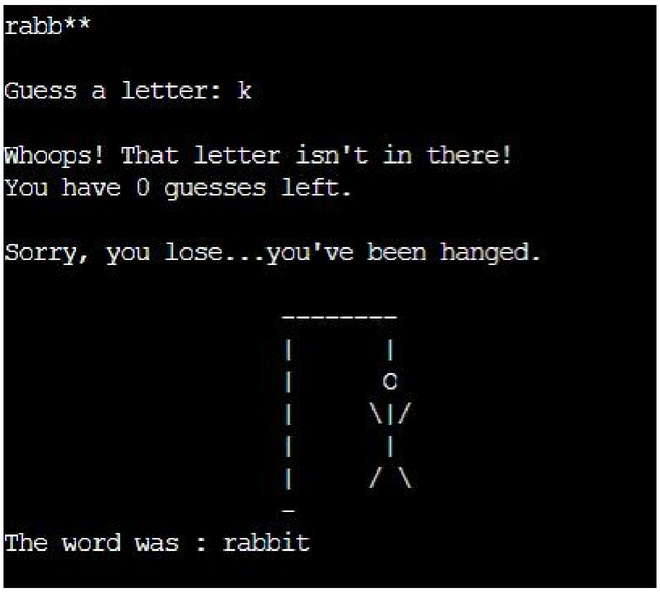
```
const int MAX_TRIES=3;
int letterFill (char, string, string&);

int main ()
{
    string name;
    char letter;
```



```
Welcome to hangman...Guess an animal Name

Each letter is represented by a star.

You have to type only one letter in one try

You have 3 tries to try and guess the word.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

******

Guess a letter: m

Whoops! That letter isn't in there!
You have 2 guesses left.


******

Guess a letter: r

You found a letter! Isn't that exciting!
You have 2 guesses left.
```



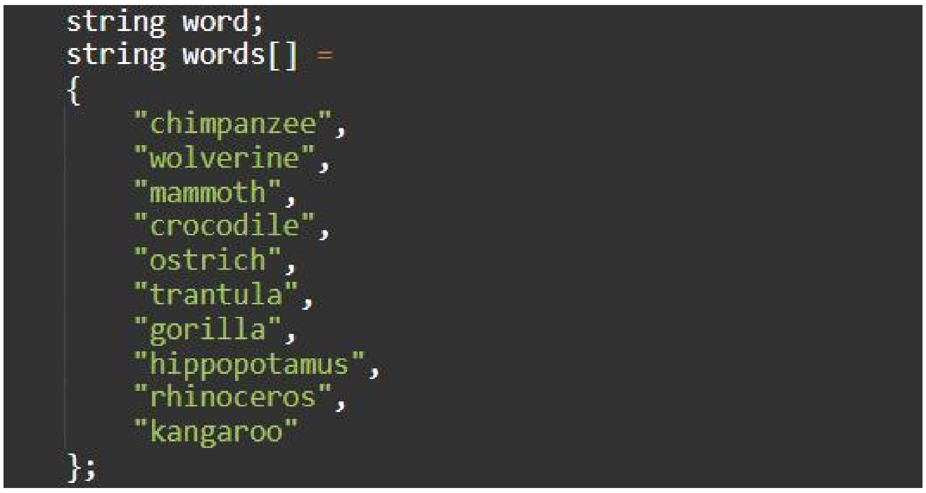
```
r*****

Guess a letter: a

You found a letter! Isn't that exciting!
You have 2 guesses left.


ra****

Guess a letter: h

Whoops! That letter isn't in there!
You have 1 guesses left.


ra****

Guess a letter: b

You found a letter! Isn't that exciting!
You have 1 guesses left.
```



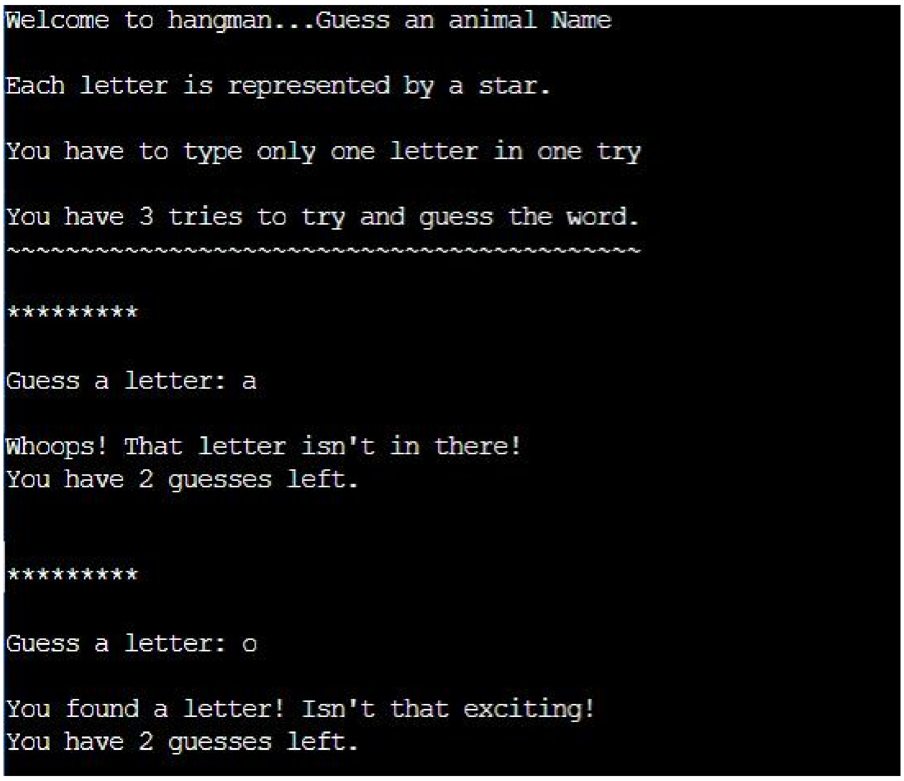
```
rabb**

Guess a letter: k

Whoops! That letter isn't in there!
You have 0 guesses left.

Sorry, you lose...you've been hanged.

                    --------
                    |      |
                    |      O
                    |     \|/
                    |      |
                    |     / \
                    -
The word was : rabbit
```

TEST CASE 3: Modifying the game to make it more challenging by increasing the length of the words to be guessed and by allowing lower limited number of incorrect guesses before the player loses the game.

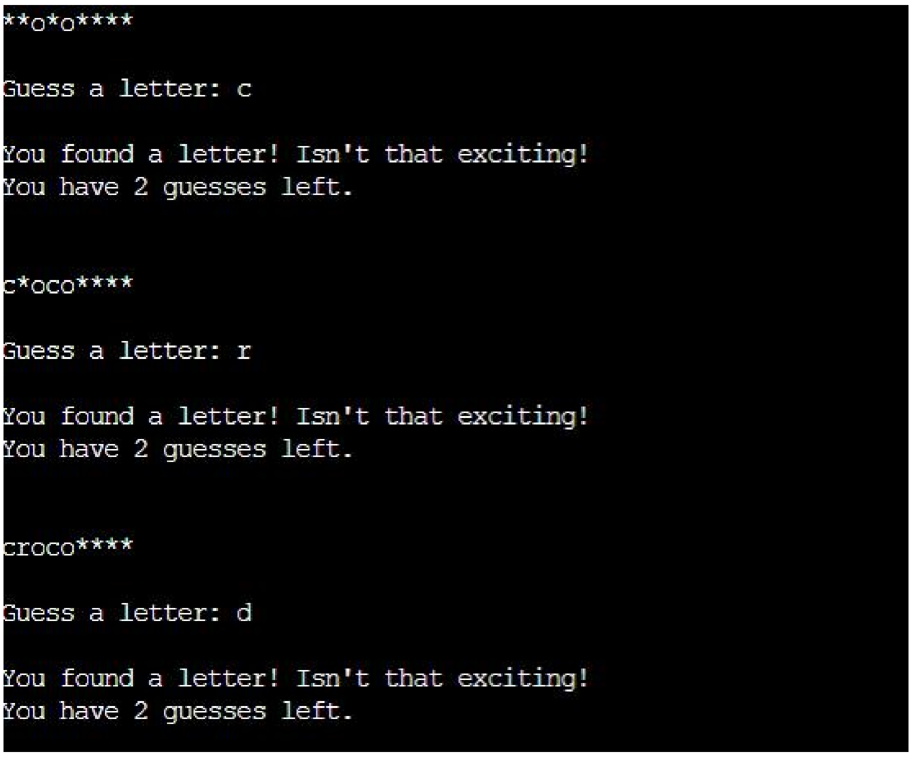
Code snippet of the edited part –



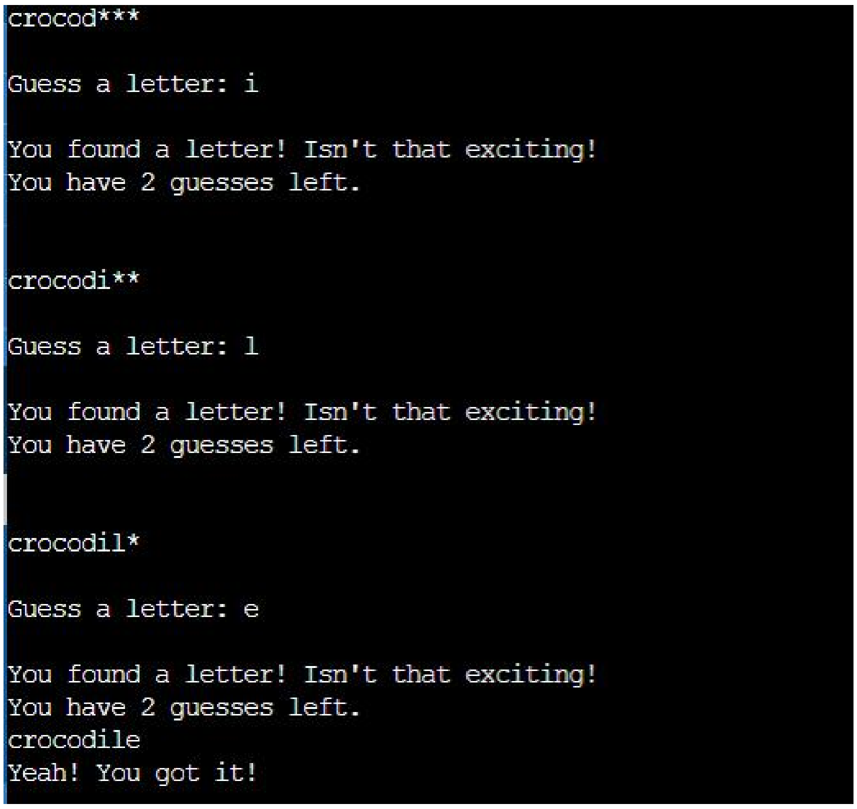
```
    string word;
    string words[] =
    {
        "chimpanzee",
        "wolverine",
        "mammoth",
        "crocodile",
        "ostrich",
        "trantula",
        "gorilla",
        "hippopotamus",
        "rhinoceros",
        "kangaroo"
    };
```



```
Welcome to hangman...Guess an animal Name

Each letter is represented by a star.

You have to type only one letter in one try

You have 3 tries to try and guess the word.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

*********

Guess a letter: a

Whoops! That letter isn't in there!
You have 2 guesses left.


*********

Guess a letter: o

You found a letter! Isn't that exciting!
You have 2 guesses left.
```



```
**o*o****

Guess a letter: c

You found a letter! Isn't that exciting!
You have 2 guesses left.


c*oco****

Guess a letter: r

You found a letter! Isn't that exciting!
You have 2 guesses left.


croco****

Guess a letter: d

You found a letter! Isn't that exciting!
You have 2 guesses left.
```



```
crocod***

Guess a letter: i

You found a letter! Isn't that exciting!
You have 2 guesses left.


crocodi**

Guess a letter: l

You found a letter! Isn't that exciting!
You have 2 guesses left.


crocodil*

Guess a letter: e

You found a letter! Isn't that exciting!
You have 2 guesses left.
crocodile
Yeah! You got it!
```

05. TIME COMPLEXITY ANALYSIS

<u>Analysing each step of the code, we get the complexity as:</u>

01. Variable Initialization: O(1)
02. Using rand() to generate a random number: O(1)
03. Initialising a string of '*' with the selected word's length: O(n), where n is the length of the selected word.
04. Looping until the number of incorrect guesses equals the maximum number of attempts:
    a. String printing: O(1)
    b. Acquiring user input: O(1)
    c. Looping through the chosen word, filling in the guessed letter if it exists: O(n), where n is the length of the chosen word.
    d. If the letter does not exist, increasing the number of incorrect guesses: O(1)
    e. Printing the number of remaining guesses: O(1)
    f. Determining whether the user correctly predicted the word: O(n), where n is the length of the picked word.
    g. Exiting the loop if the user correctly guesses the word: O(1)
05. O(1) for printing a string
06. O(1) Printing the hung figure
07. O(1) Printing the selected word

Overall, the code has a time complexity of O(n) + O(1) * (number of loop iterations), where n is the length of the picked word and the number of loop iterations is limited to MAX_TRIES. As a result, the code's time complexity is O(n) + O(MAX_TRIES) = O(n) + O(5), which can be simplied to O(n).