DSA PROJECT ON HANGMAN USING C

Sajja Aryal
Anjita Poudel

CONTENTS

INTRODUCTION

HANGMAN IS A POPULAR WORD GUESSING GAME WHERE ONE PLAYER THINKS OF A WORD.

THE OTHER PLAYER(S) ATTEMPT TO GUESS THE WORD BY SUGGESTING LETTERS.

FOR EACH INCORRECT GUESS, A PART OF A "HANGMAN" IS DRAWN UNTIL THE "HANGMAN" IS FULLY DRAWN OR THE WORD IS CORRECTLY GUESSED.

THE GAME TYPICALLY ENDS WHEN THE "HANGMAN" IS DRAWN OR THE WORD IS CORRECTLY GUESSED.

THIS PROGRAM IS A COMMAND-LINE IMPLEMENTATION OF THE HANGMAN GAME IN C PROGRAMMING LANGUAGE, WHERE A PLAYER PLAYS AGAINST A COMPUTER.

CODE OVERVIEW

The program is a simple console-based Hangman game written in C programming language.

The game randomly selects a word from a linked list and asks the player to guess the word by entering one letter at a time. The player can only make a limited number of incorrect guesses before they lose the game.

The game prints the current state of the word, the letters that have already been guessed, and a stick-figure man that grows with each incorrect guess. When the player wins or loses, the game ends and prints a message to the console.

FUNCTIONS USED IN PROGRAM

clearScreen(): clears the console screen.

printMan(int):prints the stick-figure man with a certain number of body parts, based on the number of incorrect guesses.

printWord(const char[], const char[]): prints the current state of the word and the letters that have already been guessed.

getGuess(char[]): prompts the user to enter a letter and returns the letter if it is valid.

welcome(): prints a welcome message to the console.

select_word(const struct word_node* head, char* word): This function selects a random word from the linked list of words and stores it in the given character.

The **main() function** calls these functions to implement the Hangman game logic. It first creates a linked list of words, selects a random word from the list, and then enters a loop where it repeatedly prompts the user for a guess, updates the display, and checks if the game is over (either because the player has guessed all the letters, or because they've used up all their guesses). Finally, it prints a message indicating whether the player won or lost.

```c
// Function to insert a word into the linked list
void insert_word(struct word_node** head, const char* word) {
    struct word_node* new_node = (struct word_node*)
    malloc(sizeof(struct word_node));
    strcpy(new_node->word, word);
    new_node->next = *head;
    *head = new_node;
}
```

INSERTING WORDS INTO LIST

- Create a new node with the provided word and initialize its next pointer to **NULL**.
- If the linked list is empty, set the head pointer to the new node and return.
- If the new word should be inserted at the beginning of the list (i.e., its alphabetical order is before the first word in the list), set the new node's next pointer to the current head, update the head pointer to point to the new node, and return.
- Traverse the linked list until you find the correct position to insert the new node.
- Once you find the correct position to insert the new node, set the new node's next pointer to the current node's next pointer and set the current node's next pointer to the new node.
- Return from the function.

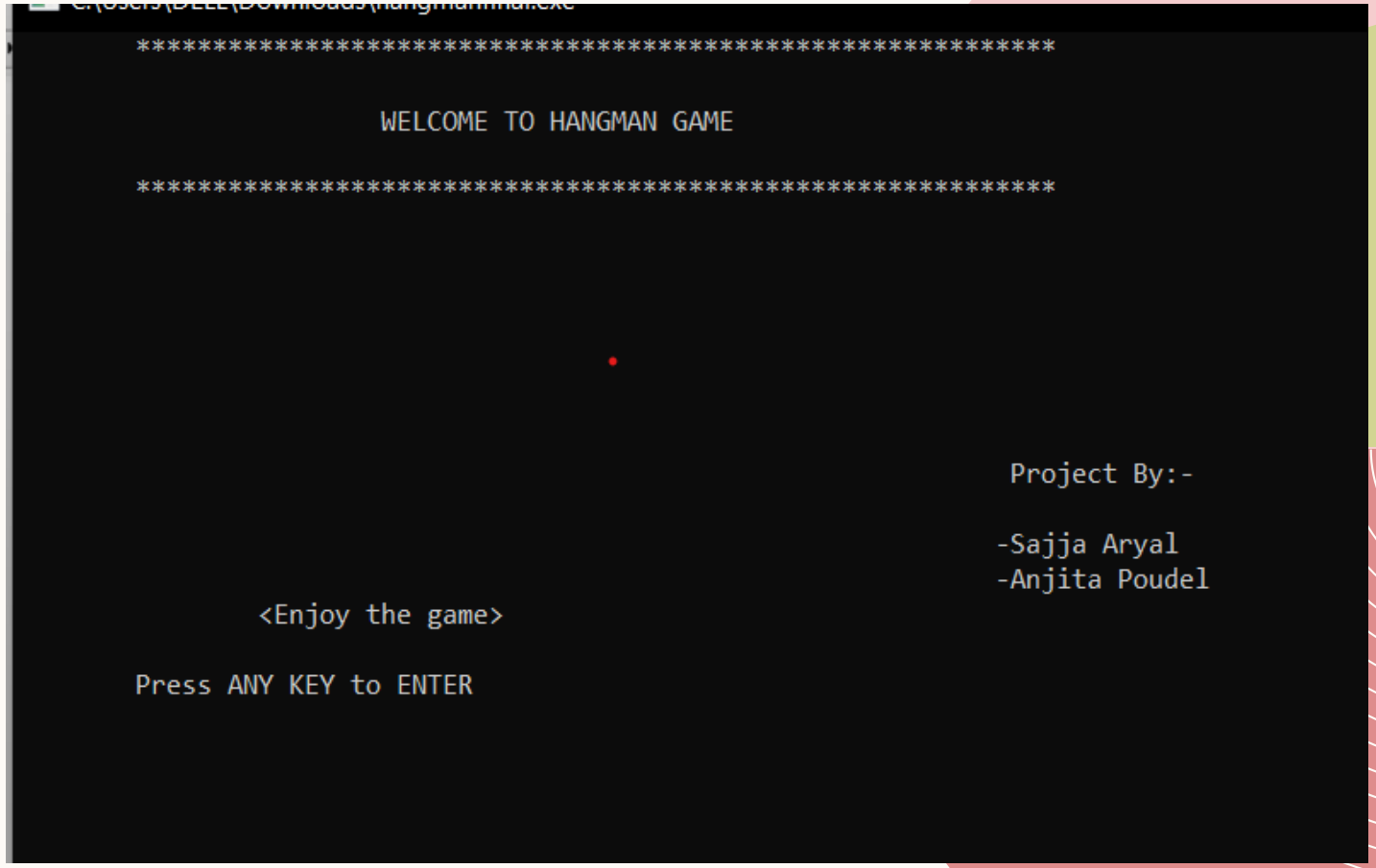
```c
void select_word(const struct word_node* head, char* word)
{
    int count = 0;
    const struct word_node* node = head;

    while (node != NULL) {
        count++;
        if (rand() % count == 0) {
            strcpy(word, node->word);
        }
        node = node->next;
    }
```

SELECTING A RANDOM WORD

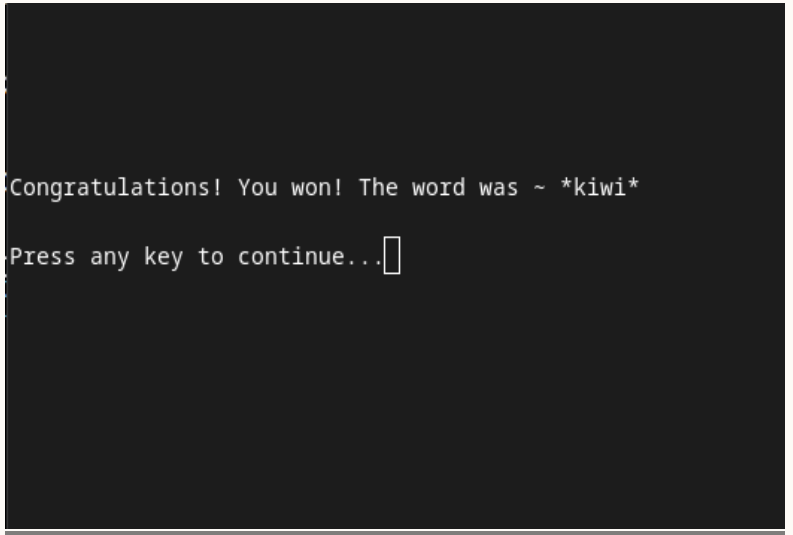
The '**select_word**' function of selects a random word from a linked list of words

- Takes two arguments - a pointer to the head of the linked list and a char array to store the selected word.
- Initializes **count** to zero and a pointer **node** to the head of the linked list.
- Enters a while loop that continues until **node** reaches the end of the linked list.
- In loop, **count** is incremented by 1 and a random number between 0 and **count** is generated using the **rand** function.
- If the random number is 0, the **word** array is set to the word stored in the current node.
- **node** is then moved to the next node in the linked list.
- When the loop ends, the **word** array will contain a randomly selected word.

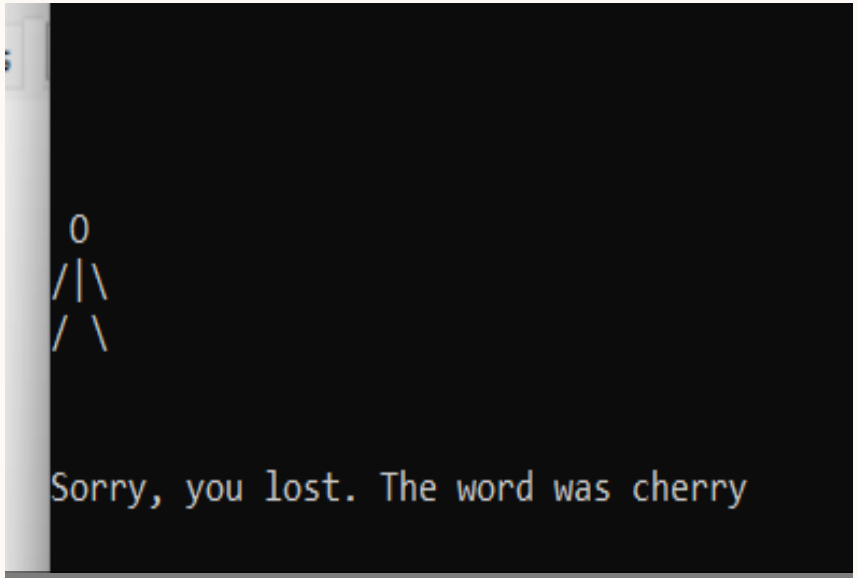
Welcome Page



Main game screen



Win screen



Lose screen

CONCLUSION

The Hangman game project provides a fun and interactive way to practice the C programming language. The project includes several important programming concepts such as linked lists, random number generation, and user interface design. The project can be extended by adding new features such as a scoring system or a graphical user interface. Overall, the project is an excellent exercise for beginners and advanced programmers alike.

THANK YOU