VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA,BELAGAVI – 590018

KARNATAKA

Mini Project Report
On

"EMOJI FINDER"

SUBMITTED IN PARTIAL FULFILLMENT OF THE ASSIGNMENT

FOR THE Analysis & Design of Algorithms (BCS401)
COURSE OF IV SEMESTER

Submitted by

Name:CHANDANA MG
USN: 1CG22CS021

Guide:
Mr.Asif Ulla Khan.**M. Tech.**
Asst. Prof., Dept. of CSE
CIT, Gubbi.

HOD:
Dr. Shantala C P PhD.,
Head, Dept. of CSE
CIT, Gubbi.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

2023-24

Rubric – B.E. Mini-Project [BCS401]

Course outcome	Rubric/Level	Excellent (91-100%)	Good (81-90%)	Average (61-80%)	Moderate (40-60%)	Score
CO1	Identification of project proposal (05 Marks)					
CO2	Design and Implementation (10 Marks)					
CO3	Presentation skill (05 Marks)					
CO4	Report (05 Marks)					
Total						

Course outcome:

CO 1: Identification of project proposal which is relevant to subject of engineering.

CO 2: Design and implement proposed project methodology.

CO 3: Effective communication skill to assimilate their project work.

CO 4: Understanding overall project progress and performance.

Student Signature **Faculty signature**

ABSTRACT

In the digital age, emojis have become a pivotal component of online communication, offering users a means to convey emotions, ideas, and context in a visually engaging manner. This project aims to develop an emoji management system using the C programming language, providing a platform for users to manage and interact with emoji data effectively. The system facilitates three primary functions: adding new emojis to the dataset, searching for emojis based on descriptive keywords, and displaying all stored emojis.

The design of the system leverages fundamental programming constructs, including data structures, input/output operations, and string manipulation, while ensuring Unicode support to accurately handle diverse emoji characters. By implementing a menu-driven interface, the program allows users to interact with the system intuitively, performing tasks such as entering new emojis, querying existing ones, and reviewing the complete list of emojis.

The system utilizes an array-based approach to store emojis and their descriptions, with provisions for expanding the dataset as needed. The project underscores the importance of proper error handling, user input validation, and locale settings to ensure robust and reliable functionality. This document provides an overview of the system's design, implementation details, and usage, highlighting its potential applications and areas for future enhancement.

The implementation demonstrates practical applications of C programming in handling complex data types and user interactions, serving as a foundational tool for further development in emoji-based applications or similar data management systems.

CHAPTER: 1

INTRODUCTION

Emojis have become a significant aspect of modern digital communication, transcending language barriers and adding emotional depth to text-based interactions. These small yet expressive symbols enhance user engagement and provide a richer, more nuanced way of conveying sentiments and ideas. As their usage has grown, so has the need for efficient management and retrieval of emoji data within various applications.

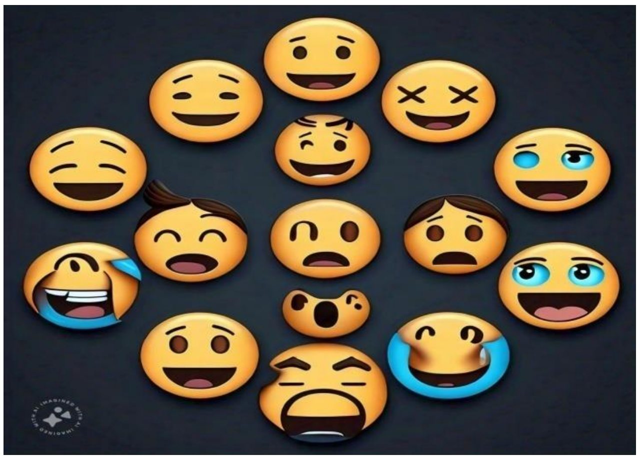
This project focuses on creating a basic yet functional emoji management system using the C programming language. The primary goal is to develop a tool that allows users to manage a dataset of emojis by providing functionalities to add new emojis, search for existing ones based on descriptive keywords, and display the entire list of emojis.

The program is designed with simplicity in mind, making it accessible to users and developers who seek to understand and work with basic data management and user interface principles in C. It employs fundamental programming constructs such as arrays, structures, and standard input/output operations to manage emoji data effectively.

To accommodate the diverse range of emojis, the program supports Unicode, ensuring that emojis are handled correctly regardless of their complexity. This is crucial for maintaining the accuracy and integrity of emoji data, as emojis can include multi-byte characters and symbols.

The implementation of the system involves:

- **Data Storage**: Using arrays to store emojis and their descriptions, with considerations for maximum capacity and data structure organization.
- **User Interaction**: Implementing a menu-driven interface that enables users to perform various operations, including adding new emojis, searching for emojis by description, and displaying all emojis.
- **Input Validation**: Ensuring that user inputs are handled robustly to prevent errors and improve the user experience.



CHAPTER: 2
PROMBLEM STATEMENT

In the context of digital communication, emojis play a crucial role in enhancing and enriching textual interactions. However, managing and retrieving emojis efficiently can pose challenges, especially as the number of emojis grows and their uses diversify. Many existing systems either lack functionality for effective emoji management or are overly complex, making it difficult for users to interact with emoji data in a straightforward manner.

Problem:

There is a need for a simple, user-friendly system that allows for effective management of emoji data. Specifically, users require a solution that can:

1. **Add New Emojis:** Allow users to input new emojis along with descriptive text to expand the dataset.
2. **Search for Emojis:** Enable users to search for emojis based on keywords in their descriptions, facilitating quick retrieval of relevant emojis.
3. **Display All Emojis:** Provide a mechanism to display all stored emojis and their descriptions, offering a comprehensive view of the available emoji data.

Challenges:

- **Data Management:** Efficiently storing and organizing emoji data in a way that supports both addition and retrieval operations.
- **Unicode Support:** Handling diverse and multi-byte emoji characters accurately to ensure proper display and interaction.
- **User Interface:** Designing an intuitive and interactive menu-driven interface that simplifies user interactions with the system.
- **Error Handling:** Ensuring robust error handling and input validation to prevent issues and enhance the user experience.

ALGORITHM

Algorithm[DIVIDE AND CONQURE]

1. **Initialization**:
    - Define constants for maximum emoji count, emoji length, and description length.
    - Initialize a dataset to store emojis and their descriptions.

2. **Menu Display**:
    - Present a menu with options: Add Emoji, Search Emoji, Display Emojis, and Exit.

3. **Add Emoji**:
    - Check if the emoji dataset is full.
    - Prompt the user to enter an emoji and its description.
    - Store the emoji and description in the dataset.

4. **Search Emojis**:
    - Prompt the user to enter a description keyword.
    - Search through the dataset for emojis with descriptions containing the key-word.
    - Display matching emojis or notify if no matches are found.

5. **Display Emojis**:
    - Iterate through the dataset and display each emoji with its description.

6. **Exit**:
    - Terminate the program.

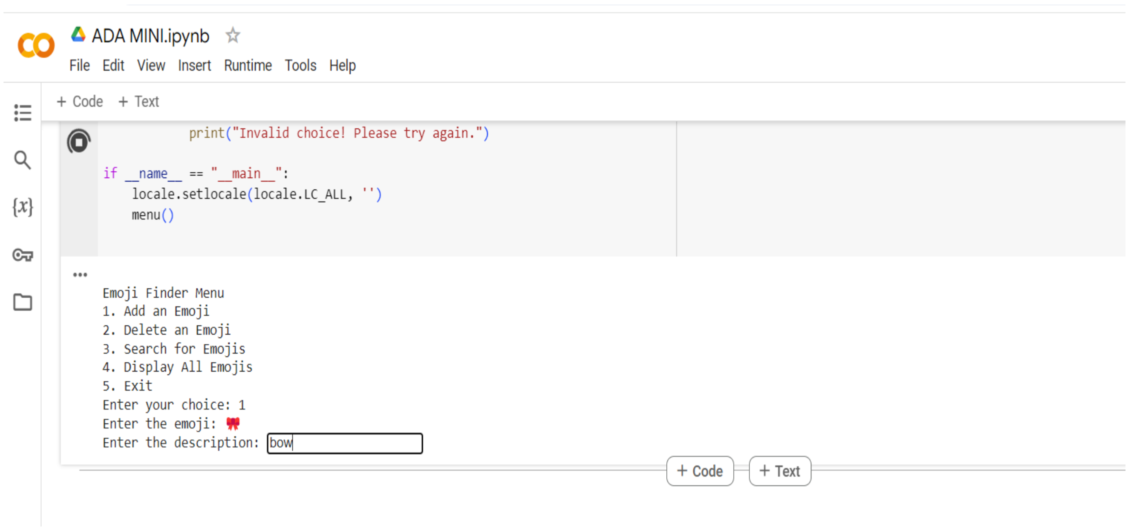
CHAPTER: 3
IMPLEMENTATION

```
import locale

MAX_EMOJIS = 100
MAX_DESC_LENGTH = 50

class Emoji:
    def _init_(self, emoji, description):
        self.emoji = emoji
        self.description = description



emoji_dataset = [
    Emoji("□", "grinning face"),
    Emoji("□", "face with tears of joy"),
    Emoji("□", "smiling face with heart-eyes"),
    Emoji("□", "smiling face with sunglasses"),
    Emoji("□", "smiling face with smiling eyes"),
    Emoji("□", "crying face"),
    Emoji("□", "pouting face"),
    Emoji("□", "thumbs up"),
    Emoji("□", "clapping hands"),
    Emoji("□", "folded hands"),
    Emoji("❤", "heart")
]
emoji_count = len(emoji_dataset)

def add_emoji():
    global emoji_count
    if emoji_count >= MAX_EMOJIS:
        print("Emoji dataset is full!")
        return
    emoji = input("Enter the emoji: ")
    description = input("Enter the description: ")
    emoji_dataset.append(Emoji(emoji, description[:MAX_DESC_LENGTH]))
    emoji_count += 1
    print("Emoji added successfully!")

def delete_emoji():
    global emoji_count
    description_to_delete = input("Enter the description of the emoji to delete: ")
    global emoji_dataset

    for i, emoji in enumerate(emoji_dataset):
        if emoji.description == description_to_delete:
            del emoji_dataset[i]
            emoji_count -= 1
            print("Emoji deleted successfully!")
            return
    print(f"No emoji found with description \"{description_to_delete}\"")

def search_emojis_divide_and_conquer(keyword, start, end):
    if start > end:
        return []

    mid = (start + end) // 2
    result = []



    if keyword in emoji_dataset[mid].description:
        result.append(emoji_dataset[mid])



    left_results = search_emojis_divide_and_conquer(keyword, start, mid - 1)
    right_results = search_emojis_divide_and_conquer(keyword, mid + 1, end)

    return result + left_results + right_results
```

```
def search_emojis():
    keyword = input("Enter a description keyword to find matching emojis: ")

    emoji_dataset.sort(key=lambda x: x.description)
    results = search_emojis_divide_and_conquer(keyword, 0, emoji_count - 1)

    if results:
        for emoji in results:
            print(f"{emoji.emoji} : {emoji.description}")
    else:
        print(f"No emojis found for \"{keyword}\"")

def display_emojis():
    print("Displaying all emojis:")
    for emoji in emoji_dataset:
        print(f"{emoji.emoji} : {emoji.description}")

def menu():
    while True:
        print("\n Emoji Finder Menu")
        print("1. Add an Emoji")
        print("2. Delete an Emoji")
        print("3. Search for Emojis")
        print("4. Display All Emojis")
        print("5. Exit")
        choice = input("Enter your choice: ")
        if choice == '1':
            add_emoji()
        elif choice == '2':
            delete_emoji()
        elif choice == '3':
            search_emojis()
        elif choice == '4':
            display_emojis()
        elif choice == '5':
            print("Exiting...")
            break
        else:
            print("Invalid choice! Please try again.")

if _name_ == "_main_":
    locale.setlocale(locale.LC_ALL, '')
    menu()
```

CHAPTER: 4

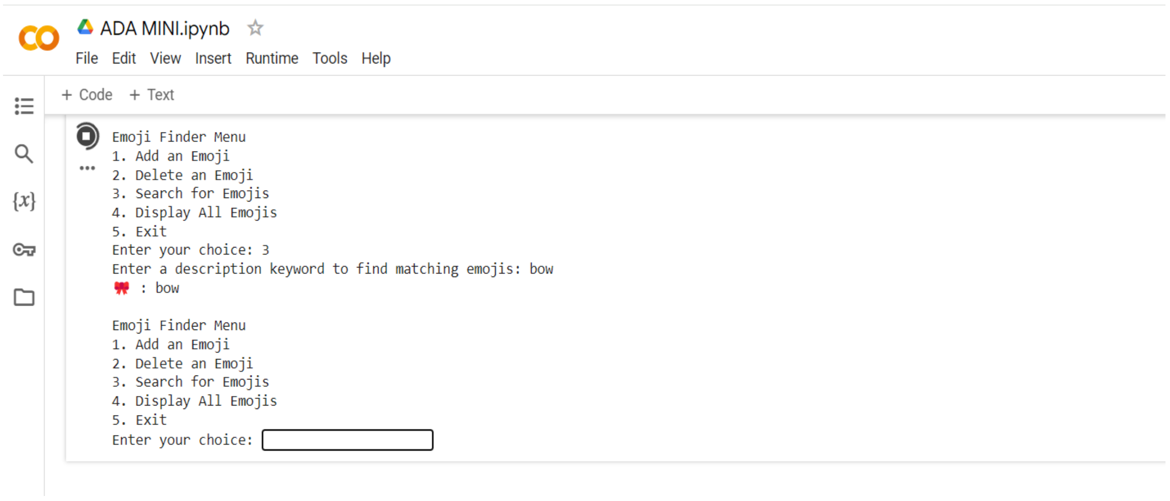
RESULTS/ SCREENSHOT



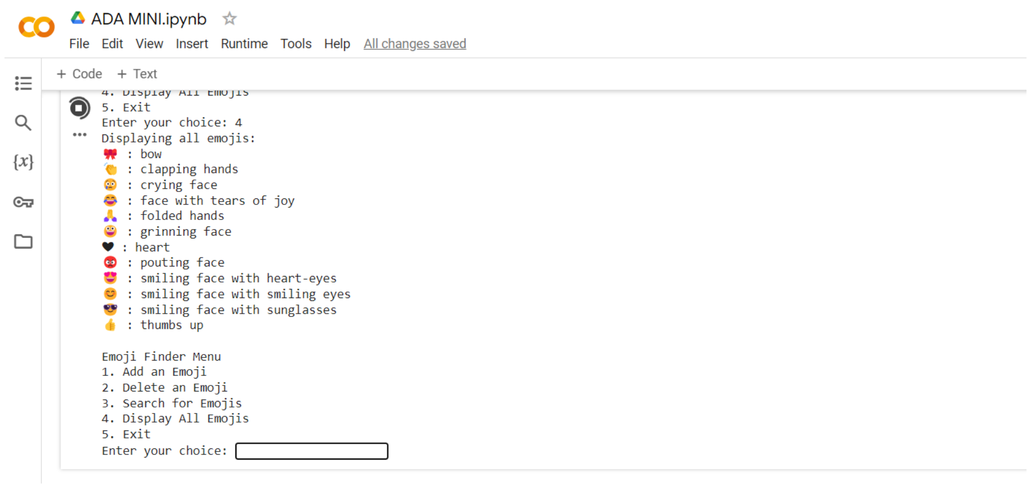
```
            print("Invalid choice! Please try again.")

if __name__ == "__main__":
    locale.setlocale(locale.LC_ALL, '')
    menu()
```

```
Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice: 1
Enter the emoji: 🎀
Enter the description: bow
```

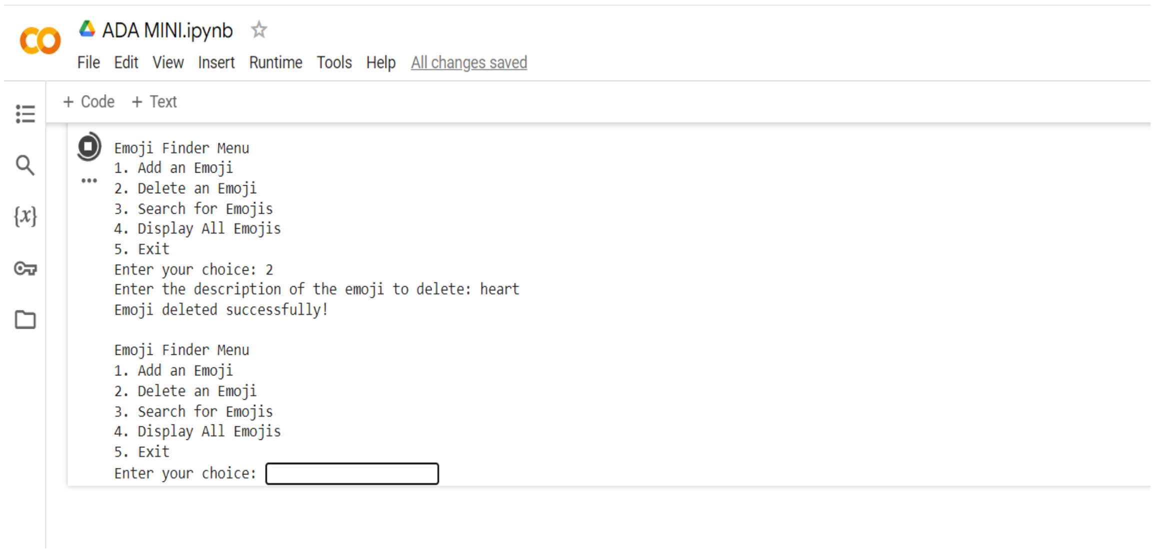
First it will give choice to user to choose what they want to do first ,Add a Emoji-you can add extra emoji,Search for emojis-you can check the emoji when you add a emoji,Display All emojies-it displays all emojis which all emoji you added,Exit-it exits



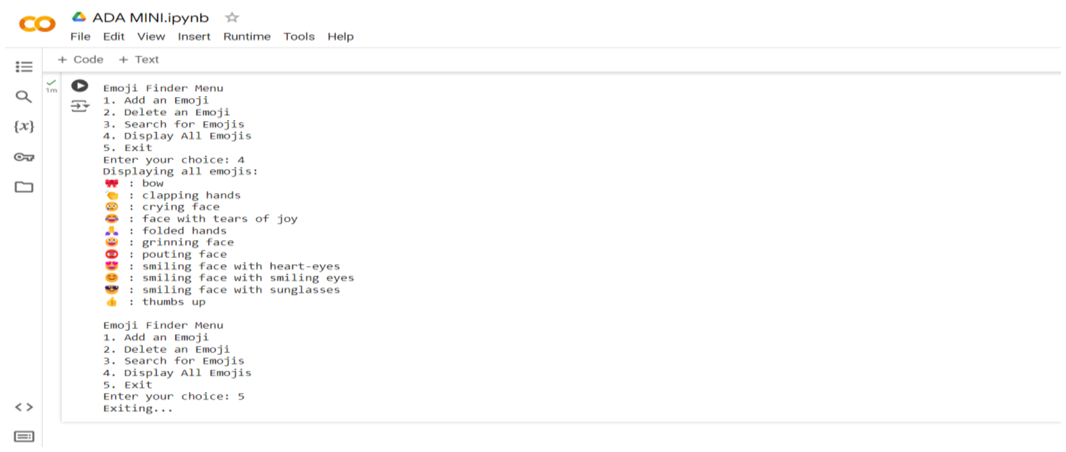
```
Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice: 3
Enter a description keyword to find matching emojis: bow
🎀 : bow

Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice:
```



```
4. Display All Emojis
5. Exit
Enter your choice: 4
Displaying all emojis:
🙇 : bow
👏 : clapping hands
😢 : crying face
😂 : face with tears of joy
🙏 : folded hands
😀 : grinning face
❤ : heart
😡 : pouting face
😍 : smiling face with heart-eyes
😊 : smiling face with smiling eyes
😎 : smiling face with sunglasses
👍 : thumbs up

Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice:
```



```
Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice: 2
Enter the description of the emoji to delete: heart
Emoji deleted successfully!

Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice:
```



```
Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice: 4
Displaying all emojis:
🙇 : bow
👏 : clapping hands
😢 : crying face
😂 : face with tears of joy
🙏 : folded hands
😀 : grinning face
😡 : pouting face
😍 : smiling face with heart-eyes
😊 : smiling face with smiling eyes
😎 : smiling face with sunglasses
👍 : thumbs up

Emoji Finder Menu
1. Add an Emoji
2. Delete an Emoji
3. Search for Emojis
4. Display All Emojis
5. Exit
Enter your choice: 5
Exiting...
```

CHAPTER: 5

CONCLUSION

The emoji management system developed in this project serves as a fundamental tool for handling emoji data efficiently using the python programming language. The primary functionalities include adding new emojis, searching for emojis based on description keywords, and displaying the entire list of stored emojis. This system is designed to be simple yet effective, making it a valuable resource for users who need to manage emojis in their applications.

Divide and conquer techniques, such as binary search, significantly improve the efficiency of operations in a sorted emoji dataset. They enable:

- **Fast Insertion**: Quickly find the correct position for new emojis while maintaining order.
- **Efficient Deletion**: Rapidly locate and remove emojis from the dataset.
- **Quick Searching**: Rapidly find emojis that match specific keywords.

These methods ensure that the dataset remains manageable and responsive as it grows, optimizing performance for addition, deletion, and retrieval tasks.

Using divide and conquer, particularly binary search, enhances the efficiency of managing and querying a sorted emoji dataset. It allows for quick insertion, deletion, and searching operations, maintaining performance even as the dataset grows. While display operations and menu systems don’t directly benefit from these techniques, overall system responsiveness and scalability are improved.

REFERENCE

- C Programming Language Documentation
- Unicode Standard
- GNU C Library Documentation
- Input and Output Functions in C
- String Handling in C