SAMPLE PROJECTS

Sercan Külcü | 21.06.2023

Contents

1 Sample projects

1.1 Number Guessing Game:

Develop a program that generates a random number and prompts the user to guess it. Provide hints such as "too high" or "too low" to guide the user towards the correct answer.

Here's an example of a Number Guessing Game written in C:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


int main() {

   int number, guess, attempts = 0;


   // Generate a random number between 1 and 100

   srand(time(NULL));

   number = rand() % 100 + 1;


   printf("Welcome to the Number Guessing Game!\n");

   printf("I have chosen a number between 1 and 100.\n");


   do {

      printf("Enter your guess: ");

      scanf("%d", &guess);

      attempts++;
```

```
        if (guess > number) {
            printf("Too high! Try again.\n");
        } else if (guess < number) {
            printf("Too low! Try again.\n");
        } else {
            printf("Congratulations! You guessed the correct number in %d attempts.\n",
attempts);
        }
    } while (guess != number);

    return 0;
}
```

In this program, we use the srand function from time.h to seed the random number generator with the current time. This ensures that each time we run the program, a different random number will be generated.

The program prompts the user to enter their guess and compares it with the randomly generated number. If the guess is higher or lower than the actual number, the program provides the appropriate hint. If the guess matches the number, the program congratulates the user and displays the number of attempts it took to guess correctly.

Note that the program will keep prompting the user for guesses until they guess the correct number.

1.2 Morse Code Translator:

Build a program that translates English text into Morse code and vice versa. Include a lookup table for the Morse code alphabet and implement the conversion logic.

Here's an example of a Morse Code Translator program written in C:

```
#include <stdio.h>
#include <string.h>

char* morseCodeLookup(char character) {
    // Lookup table for Morse code alphabet
    // Change this table as needed to include all required characters
    char* morseCode[] = {
        ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
        "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-",
        "-.--", "--..", " "
    };

    // Convert uppercase characters to lowercase
    if (character >= 'A' && character <= 'Z') {
        character += 32;
    }

    // Calculate index based on ASCII value
    int index = character - 'a';

    // Return corresponding Morse code from lookup table
    if (index >= 0 && index < 26) {
        return morseCode[index];
    } else if (character == ' ') {
        return morseCode[26];
```

```
    }

    return "";
}

void englishToMorseCode(const char* englishText) {
    int length = strlen(englishText);

    printf("Morse Code: ");

    for (int i = 0; i < length; i++) {
        char character = englishText[i];
        char* morseCode = morseCodeLookup(character);

        printf("%s ", morseCode);
    }

    printf("\n");
}

void morseCodeToEnglish(const char* morseCodeText) {
    char* token;
    char* textCopy = strdup(morseCodeText);

    printf("English Text: ");
```

```
token = strtok(textCopy, " ");
while (token != NULL) {
    if (strcmp(token, "") != 0) {
        int found = 0;

        for (char ch = 'a'; ch <= 'z'; ch++) {
            char* morseCode = morseCodeLookup(ch);
            if (strcmp(token, morseCode) == 0) {
                printf("%c", ch);
                found = 1;
                break;
            }
        }

        if (!found && strcmp(token, morseCodeLookup(' ')) == 0) {
            printf(" ");
        }
    }

    token = strtok(NULL, " ");
}

printf("\n");
```

```
    free(textCopy);
}

int main() {
    char input[100];
    int choice;

    printf("Morse Code Translator\n");
    printf("1. English to Morse Code\n");
    printf("2. Morse Code to English\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);
    getchar(); // Consume the newline character

    if (choice == 1) {
        printf("Enter the English text: ");
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0'; // Remove trailing newline

        englishToMorseCode(input);
    } else if (choice == 2) {
        printf("Enter the Morse code: ");
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0'; // Remove trailing newline
```

```
        morseCodeToEnglish(input);
    } else {
        printf("Invalid choice. Please enter either 1 or 2.\n");
    }

    return 0;
}
```

In this program, we have a morseCodeLookup function that takes a character as input and returns its corresponding Morse code from a lookup table. The lookup table contains Morse code representations for the lowercase alphabets 'a' to 'z' and a space character.

The englishToMorseCode function takes an English text string as input and converts it to Morse code. It iterates through each character of the input string, looks up its corresponding Morse code using morseCodeLookup, and prints the Morse code representation.

The morseCodeToEnglish function takes a Morse code string as input and converts it to English text. It tokenizes the input string using spaces as delimiters and compares each token with the Morse code representations in the lookup table to find the corresponding English character.

In the main function, the user is prompted to choose between English to Morse code conversion or Morse code to English conversion. Depending on the choice, the program takes the input text from the user and calls the appropriate conversion function.

Note that the lookup table in the morseCodeLookup function can be expanded to include additional characters if needed.

1.3 Personal Diary:

Create a program that allows users to write and save their personal diary entries. Implement features like adding entries, viewing previous entries, and encrypting the diary for privacy.

Here's an example of a Personal Diary program written in C that allows users to add entries, view previous entries, and encrypt the diary for privacy:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX_ENTRIES 100

#define MAX_ENTRY_LENGTH 1000


typedef struct {

   char date[20];

   char content[MAX_ENTRY_LENGTH];

} DiaryEntry;


void addEntry(DiaryEntry diary[], int* entryCount) {

   if (*entryCount >= MAX_ENTRIES) {

      printf("Diary is full. Cannot add more entries.\n");

      return;

   }


   DiaryEntry newEntry;

   printf("Enter the date (DD/MM/YYYY): ");

   scanf("%s", newEntry.date);

   getchar(); // Consume the newline character
```

```
    printf("Enter the content (up to %d characters):\n", MAX_ENTRY_LENGTH);
    fgets(newEntry.content, sizeof(newEntry.content), stdin);
    newEntry.content[strcspn(newEntry.content, "\n")] = '\0'; // Remove trailing newline

    diary[*entryCount] = newEntry;
    (*entryCount)++;

    printf("Entry added successfully.\n");
}

void viewEntries(DiaryEntry diary[], int entryCount) {
    if (entryCount == 0) {
        printf("No entries found.\n");
        return;
    }

    for (int i = 0; i < entryCount; i++) {
        printf("\nDate: %s\n", diary[i].date);
        printf("Content:\n%s\n", diary[i].content);
    }
}

void encryptDiary(DiaryEntry diary[], int entryCount) {
    // XOR encryption key
```

```
    char key = 'K';

    for (int i = 0; i < entryCount; i++) {
        // Encrypt date
        for (int j = 0; diary[i].date[j] != '\0'; j++) {
            diary[i].date[j] = diary[i].date[j] ^ key;
        }

        // Encrypt content
        for (int j = 0; diary[i].content[j] != '\0'; j++) {
            diary[i].content[j] = diary[i].content[j] ^ key;
        }
    }

    printf("Diary encrypted successfully.\n");
}

void decryptDiary(DiaryEntry diary[], int entryCount) {
    // XOR encryption key
    char key = 'K';

    for (int i = 0; i < entryCount; i++) {
        // Decrypt date
        for (int j = 0; diary[i].date[j] != '\0'; j++) {
            diary[i].date[j] = diary[i].date[j] ^ key;
```

```
        }

        // Decrypt content
        for (int j = 0; diary[i].content[j] != '\0'; j++) {
            diary[i].content[j] = diary[i].content[j] ^ key;
        }
    }

    printf("Diary decrypted successfully.\n");
}

int main() {
    DiaryEntry diary[MAX_ENTRIES];
    int entryCount = 0;
    int choice;

    printf("Personal Diary\n");

    do {
        printf("\n1. Add entry\n");
        printf("2. View entries\n");
        printf("3. Encrypt diary\n");
        printf("4. Decrypt diary\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
```

```
    scanf("%d", &choice);
    getchar(); // Consume the newline character

    switch (choice) {
        case 1:
            addEntry(diary, &entryCount);
            break;
        case 2:
            viewEntries(diary, entryCount);
            break;
        case 3:
            encryptDiary(diary, entryCount);
            break;
        case 4:
            decryptDiary(diary, entryCount);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please enter a number from 1 to 5.\n");
    }
} while (choice != 5);

return 0;
```

}

In this program, we have a DiaryEntry struct that represents a single diary entry, containing the date and content. The maximum number of entries is defined as MAX_ENTRIES, and the maximum length of each entry is defined as MAX_ENTRY_LENGTH.

The addEntry function allows the user to input the date and content for a new diary entry and adds it to the diary array. The viewEntries function displays all the entries in the diary. The encryptDiary and decryptDiary functions use XOR encryption to encrypt or decrypt the dates and content of all the diary entries.

The main function presents a menu to the user, allowing them to choose options like adding entries, viewing entries, encrypting the diary, decrypting the diary, or exiting the program. Based on the user's choice, the corresponding function is called.

Note that this is a basic implementation, and it is recommended to further enhance the program with features like data persistence (e.g., saving entries to a file), authentication, and additional error handling as per your requirements.

1.4 Weather Forecast Application:

Develop a program that retrieves weather information from an API and displays it to the user. Allow users to enter their location and provide details such as temperature, humidity, and weather conditions.

1.5 Cryptography Tool:

Create a program that encrypts and decrypts text using various encryption algorithms like Caesar cipher, Vigenère cipher, or substitution cipher. Allow users to input the text and choose the encryption method.

1.6 Task Manager:

Build a simple task manager program that allows users to create, update, and delete tasks. Implement features like task priority, due dates, and task categorization.

1.7 File Encryption Tool:

Develop a program that encrypts files using encryption algorithms like AES (Advanced Encryption Standard) or DES (Data Encryption Standard). Allow users to select files and specify the encryption key.

1.8 Maze Solver:

Create a program that solves a maze using algorithms like depth-first search (DFS) or breadth-first search (BFS). Allow users to input the maze and display the solved path.

1.9 BMI Calculator:

Build a program that calculates Body Mass Index (BMI) based on user input for weight and height. Provide feedback on the BMI category and display health recommendations.

1.10 Quiz Generator:

Develop a program that generates quizzes based on a set of questions and multiple-choice options. Allow users to take the quiz, display scores, and provide correct answers at the end.

1.11 Network Chat Application:

Create a client-server chat application using sockets, allowing multiple clients to communicate with each other in real-time.

1.12 Sudoku Solver:

Create a program that can solve Sudoku puzzles by implementing backtracking or constraint satisfaction algorithms.

1.13 Data Compression:

Build a program that can compress and decompress files using compression algorithms like Huffman coding or Lempel-Ziv-Welch (LZW).

1.14 Operating System Scheduler:

Implement a basic process scheduler for an operating system, incorporating different scheduling algorithms like round-robin or priority scheduling.