Project Title: Splay Tree Password Manager

Developed by

Roll No.:22881A05K7

Name A. Raj Kumar

Class: II B.Tech. CSE-D

Regulation: R22

Under the Guidance of

Dr. Vasantha SV, Associate Professor, CSE Dept.

Github Link: https://github.com/achiever2004/ads-project

Problem Statement:

Design and implement a password manager that utilizes Splay Trees to efficiently store and retrieve passwords for various accounts. The password manager should allow users to add, update, delete, and retrieve passwords securely.

Algorithm/ Explanation of Logic:

- **Splay Tree Implementation:**

  Implement a Splay Tree data structure to store account information.

  Each node in the Splay Tree represents an account with fields for the account name, username, and password.

- **Password Operations:**

  Implement functions for adding a new account, updating an existing account, deleting an account, and retrieving a password.

  During each operation, splay the corresponding node to the root of the tree to optimize future access to recently accessed accounts.

- **Security Measures:**

  Incorporate encryption and secure hashing techniques to store passwords securely.

  Implement a master password system to ensure access control.

- **User Interface:**

  Develop a simple command-line or graphical user interface for users to interact with the password manager.

  Include options for adding, updating, deleting, and retrieving passwords.

Code:

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

typedef struct Account {
    char account_name[50];
    char username[50];
    char password[50];
    struct Account* left;
    struct Account* right;
} Account;

// Splay Tree Operations
Account* rightRotate(Account* x);
Account* leftRotate(Account* x);
Account* splay(Account* root, char key[]);
Account* insert(Account* root, char account_name[], char username[], char password[]);
Account* search(Account* root, char account_name[]);
Account* delete(Account* root, char account_name[]);
void printInOrder(Account* root);

// Password Manager Operations
void addPassword(Account** root, char account_name[], char username[], char
password[]) {
    *root = insert(*root, account_name, username, password);
}
```

```
void updatePassword(Account** root, char account_name[], char username[], char
password[]) {
    *root = delete(*root, account_name);
    *root = insert(*root, account_name, username, password);
}

void deletePassword(Account** root, char account_name[]) {
    *root = delete(*root, account_name);
}

char* retrievePassword(Account* root, char account_name[]) {
    Account* result = search(root, account_name);
    if (result != NULL) {
        return result->password;
    } else {
        return NULL;
    }
}

int main() {
    Account* root = NULL;

    // Test cases
    addPassword(&root, "Email", "user@example.com", "securePassword1");
    addPassword(&root, "Bank", "john_doe", "strongPassword123");
    addPassword(&root, "Social Media", "user123", "password567");

    // Print the Splay Tree in order
    printf("Splay Tree in order:\n");
```

```
    printInOrder(root);

    // Retrieve a password
    char* retrievedPassword = retrievePassword(root, "Bank");
    if (retrievedPassword != NULL) {
        printf("\nRetrieved Password for Bank: %s\n", retrievedPassword);
    } else {
        printf("\nAccount not found\n");
    }

    // Update a password
    updatePassword(&root, "Bank", "john_doe", "newStrongPassword456");

    // Print the updated Splay Tree
    printf("\nSplay Tree after updating Bank account:\n");
    printInOrder(root);

    // Delete an account
    deletePassword(&root, "Email");

    // Print the Splay Tree after deletion
    printf("\nSplay Tree after deleting Email account:\n");
    printInOrder(root);

    return 0;
}

// Splay Tree Implementation
Account* rightRotate(Account* x) {
```

```
    Account* y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}

Account* leftRotate(Account* x) {
    Account* y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}

Account* splay(Account* root, char key[]) {
    if (root == NULL || strcmp(root->account_name, key) == 0) {
        return root;
    }

    if (strcmp(root->account_name, key) > 0) {
        if (root->left == NULL) {
            return root;
        }

        if (strcmp(root->left->account_name, key) > 0) {
            root->left->left = splay(root->left->left, key);
            root = rightRotate(root);
        } else if (strcmp(root->left->account_name, key) < 0) {
            root->left->right = splay(root->left->right, key);
```

```
            if (root->left->right != NULL) {
                root->left = leftRotate(root->left);
            }
        }

        return (root->left == NULL) ? root : rightRotate(root);
    } else {
        if (root->right == NULL) {
            return root;
        }

        if (strcmp(root->right->account_name, key) > 0) {
            root->right->left = splay(root->right->left, key);
            if (root->right->left != NULL) {
                root->right = rightRotate(root->right);
            }
        } else if (strcmp(root->right->account_name, key) < 0) {
            root->right->right = splay(root->right->right, key);
            root = leftRotate(root);
        }

        return (root->right == NULL) ? root : leftRotate(root);
    }
}

Account* insert(Account* root, char account_name[], char username[], char password[]) {
    if (root == NULL) {
        Account* newNode = (Account*)malloc(sizeof(Account));
```

```
    strcpy(newNode->account_name, account_name);
    strcpy(newNode->username, username);
    strcpy(newNode->password, password);
    newNode->left = newNode->right = NULL;
    return newNode;
}

root = splay(root, account_name);

int compareResult = strcmp(account_name, root->account_name);
if (compareResult < 0) {
    Account* newNode = (Account*)malloc(sizeof(Account));
    strcpy(newNode->account_name, account_name);
    strcpy(newNode->username, username);
    strcpy(newNode->password, password);
    newNode->left = newNode->right = NULL;
    newNode->left = root->left;
    root->left = NULL;
    newNode->right = root;
    root = newNode;
} else if (compareResult > 0) {
    Account* newNode = (Account*)malloc(sizeof(Account));
    strcpy(newNode->account_name, account_name);
    strcpy(newNode->username, username);
    strcpy(newNode->password, password);
    newNode->left = newNode->right = NULL;
    newNode->right = root->right;
    root->right = NULL;
```

```
        newNode->left = root;
        root = newNode;
    }

    return root;
}

Account* search(Account* root, char account_name[]) {
    return splay(root, account_name);
}

Account* delete(Account* root, char account_name[]) {
    if (root == NULL) {
        return root;
    }

    root = splay(root, account_name);

    if (strcmp(root->account_name, account_name) != 0) {
        return root;
    }

    Account* temp;
    if (root->left == NULL) {
        temp = root;
        root = root->right;
    } else {
        temp = root;
        root = splay(root->left, account_name);
```

```
        root->right = temp->right;
    }

    free(temp);
    return root;
}

void printInOrder(Account* root) {
    if (root != NULL) {
        printInOrder(root->left);
        printf("Account: %s, Username: %s, Password: %s\n", root->account_name, root->username, root->password);
        printInOrder(root->right);
    }
}
```

Output test cases:

```
// Test case 1: Add a new account
addPassword(&root, "Shopping", "shopper123", "secureShoppingPwd");

// Test case 2: Retrieve a password
char* retrievedPassword = retrievePassword(root, "Shopping");
printf("Retrieved Password for Shopping: %s\n", retrievedPassword);

// Test case 3: Update an existing account
updatePassword(&root, "Shopping", "shopper123", "newSecurePwd123");
```

// Test case 4: Delete an account

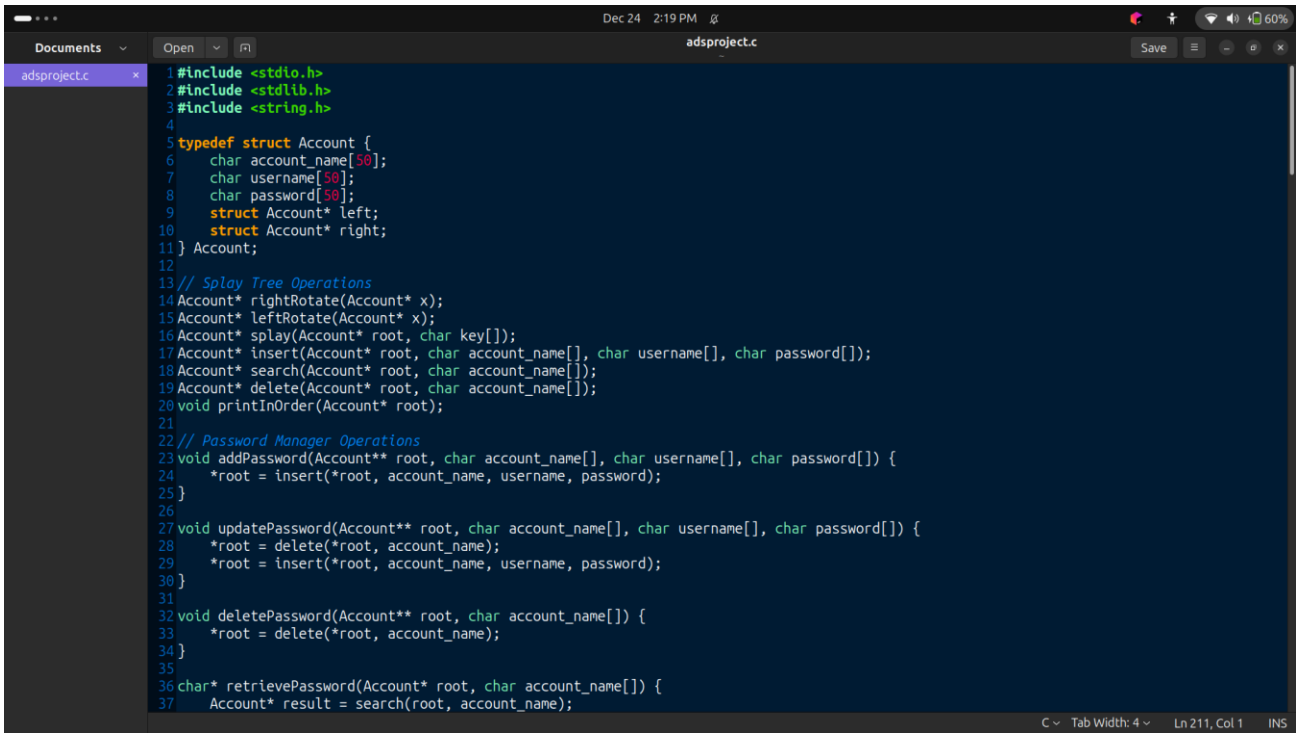
deletePassword(&root, "Shopping");

// Test case 5: Print the Splay Tree after operations

printf("Splay Tree in order:\n");

printInOrder(root);

Screenshot of code and executed output:




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

typedef struct Account {
    char account_name[50];
    char username[50];
    char password[50];
    struct Account* left;
    struct Account* right;
} Account;

// Splay Tree Operations
Account* rightRotate(Account* x);
Account* leftRotate(Account* x);
Account* splay(Account* root, char key[]);
Account* insert(Account* root, char account_name[], char username[], char password[]);
Account* search(Account* root, char account_name[]);
Account* delete(Account* root, char account_name[]);
void printInOrder(Account* root);

// Password Manager Operations
void addPassword(Account** root, char account_name[], char username[], char password[]) {
    *root = insert(*root, account_name, username, password);
}

void updatePassword(Account** root, char account_name[], char username[], char password[]) {
    *root = delete(*root, account_name);
    *root = insert(*root, account_name, username, password);
}

void deletePassword(Account** root, char account_name[]) {
    *root = delete(*root, account_name);
}

char* retrievePassword(Account* root, char account_name[]) {
    Account* result = search(root, account_name);
```

Dec 24 2:19 PM 60%

Documents Open adsproject.c Save

adsproject.c

```c
    Account* result = search(root, account_name);
    if (result != NULL) {
        return result->password;
    } else {
        return NULL;
    }
}

int main() {
    Account* root = NULL;

    // Test cases
    addPassword(&root, "Email", "user@example.com", "securePassword1");
    addPassword(&root, "Bank", "john_doe", "strongPassword123");
    addPassword(&root, "Social Media", "user123", "password567");

    // Print the Splay Tree in order
    printf("Splay Tree in order:\n");
    printInOrder(root);

    // Retrieve a password
    char* retrievedPassword = retrievePassword(root, "Bank");
    if (retrievedPassword != NULL) {
        printf("\nRetrieved Password for Bank: %s\n", retrievedPassword);
    } else {
        printf("\nAccount not found\n");
    }

    // Update a password
    updatePassword(&root, "Bank", "john_doe", "newStrongPassword456");

    // Print the updated Splay Tree
    printf("\nSplay Tree after updating Bank account:\n");
    printInOrder(root);

    // Delete an account
    deletePassword(&root, "Email");
```

C Tab Width: 4 Ln 211, Col 1 INS

Dec 24 2:20 PM 60%

Documents Open adsproject.c Save

adsproject.c

```c
    printInOrder(root);

    // Delete an account
    deletePassword(&root, "Email");

    // Print the Splay Tree after deletion
    printf("\nSplay Tree after deleting Email account:\n");
    printInOrder(root);

    return 0;
}

// Splay Tree Implementation
Account* rightRotate(Account* x) {
    Account* y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}

Account* leftRotate(Account* x) {
    Account* y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}

Account* splay(Account* root, char key[]) {
    if (root == NULL || strcmp(root->account_name, key) == 0) {
        return root;
    }

    if (strcmp(root->account_name, key) > 0) {
        if (root->left == NULL) {
            return root;
        }

```

C Tab Width: 4 Ln 44, Col 1 INS

Dec 24 2:20 PM 60%

Documents Open adsproject.c Save

adsproject.c

```c

        if (strcmp(root->left->account_name, key) > 0) {
            root->left->left = splay(root->left->left, key);
            root = rightRotate(root);
        } else if (strcmp(root->left->account_name, key) < 0) {
            root->left->right = splay(root->left->right, key);
            if (root->left->right != NULL) {
                root->left = leftRotate(root->left);
            }
        }

        return (root->left == NULL) ? root : rightRotate(root);
    } else {
        if (root->right == NULL) {
            return root;
        }

        if (strcmp(root->right->account_name, key) > 0) {
            root->right->left = splay(root->right->left, key);
            if (root->right->left != NULL) {
                root->right = rightRotate(root->right);
            }
        } else if (strcmp(root->right->account_name, key) < 0) {
            root->right->right = splay(root->right->right, key);
            root = leftRotate(root);
        }

        return (root->right == NULL) ? root : leftRotate(root);
    }
}

Account* insert(Account* root, char account_name[], char username[], char password[]) {
    if (root == NULL) {
        Account* newNode = (Account*)malloc(sizeof(Account));
        strcpy(newNode->account_name, account_name);
        strcpy(newNode->username, username);
        strcpy(newNode->password, password);
```

C Tab Width: 4 Ln 44, Col 1 INS

Dec 24 2:20 PM 61%

Documents Open adsproject.c Save

adsproject.c

```c
        strcpy(newNode->account_name, account_name);
        strcpy(newNode->username, username);
        strcpy(newNode->password, password);
        newNode->left = newNode->right = NULL;
        return newNode;
    }

    root = splay(root, account_name);

    int compareResult = strcmp(account_name, root->account_name);
    if (compareResult < 0) {
        Account* newNode = (Account*)malloc(sizeof(Account));
        strcpy(newNode->account_name, account_name);
        strcpy(newNode->username, username);
        strcpy(newNode->password, password);
        newNode->left = newNode->right = NULL;
        newNode->left = root->left;
        root->left = NULL;
        newNode->right = root;
        root = newNode;
    } else if (compareResult > 0) {
        Account* newNode = (Account*)malloc(sizeof(Account));
        strcpy(newNode->account_name, account_name);
        strcpy(newNode->username, username);
        strcpy(newNode->password, password);
        newNode->left = newNode->right = NULL;
        newNode->right = root->right;
        root->right = NULL;
        newNode->left = root;
        root = newNode;
    }

    return root;
}

Account* search(Account* root, char account_name[]) {
    return splay(root, account_name);
```

C Tab Width: 4 Ln 44, Col 1 INS

```c
Account* search(Account* root, char account_name[]) {
    return splay(root, account_name);
}

Account* delete(Account* root, char account_name[]) {
    if (root == NULL) {
        return root;
    }

    root = splay(root, account_name);

    if (strcmp(root->account_name, account_name) != 0) {
        return root;
    }

    Account* temp;
    if (root->left == NULL) {
        temp = root;
        root = root->right;
    } else {
        temp = root;
        root = splay(root->left, account_name);
        root->right = temp->right;
    }

    free(temp);
    return root;
}

void printInOrder(Account* root) {
    if (root != NULL) {
        printInOrder(root->left);
        printf("Account: %s, Username: %s, Password: %s\n", root->account_name, root->username, root->password);
        printInOrder(root->right);
    }
}
```

Output:

```
aimer@aimer-Inspiron-15-7000-Gaming:~$ gedit adsproject.c
aimer@aimer-Inspiron-15-7000-Gaming:~$ gcc adsproject.c
aimer@aimer-Inspiron-15-7000-Gaming:~$ ./a.out
Splay Tree in order:
Account: Bank, Username: john_doe, Password: strongPassword123
Account: Email, Username: user@example.com, Password: securePassword1
Account: Social Media, Username: user123, Password: password567

Retrieved Password for Bank: strongPassword123

Splay Tree after updating Bank account:
Account: Bank, Username: john_doe, Password: newStrongPassword456
Account: Social Media, Username: user123, Password: password567

Splay Tree after deleting Email account:
Account: Bank, Username: john_doe, Password: newStrongPassword456
Account: Social Media, Username: user123, Password: password567
aimer@aimer-Inspiron-15-7000-Gaming:~$
```