**Project Documentation Template**

1. **Project Title:**

**Otaku Outreach Network Management System**

1. **Project Description:**

* **Overview:** This project is a C program that simulates a hierarchical member network, called the "Otaku Outreach Network," using a binary tree data structure. The program allows for managing members, tracking their positions within the network, and calculating commissions based on their level in the hierarchy.
* **Objective:** The primary goal is to create a system that efficiently manages members in a multi-level network, allowing for easy tracking of member relationships and automated commission calculations based on a predefined level structure.
* **Scope:** The current implementation is limited to adding new members under existing members, displaying the immediate subordinates of a given member, and displaying the entire network tree. It does not support member removal, persistent data storage (data is lost on program exit), or more sophisticated commission schemes beyond a simple level-based system. Error handling is basic, focusing on memory allocation and preventing adding more than two subordinates per member.

**Scope - Additional Considerations:**

* **Data Storage:** The program currently stores data in memory only, meaning it is lost when the program ends. Adding a feature to save the network structure to a file (like a text file or a simple database format) would allow for persistence and data retrieval across multiple sessions.
* **User Interface:** The program uses a simple text-based menu. A more user-friendly interface could be implemented using a graphical library (like ncurses for a terminal-based UI or a GUI toolkit like GTK or Qt). This would provide a more visually appealing and interactive experience for the user.
* **Member Modification:** The program currently only allows adding members. Implementing features to modify member names, change their recruiter, or remove members would enhance its functionality.
* **Commission Calculation:** The current commission calculation is basic, based solely on level. A more complex system could be implemented, perhaps taking into account factors like the number of downlines, overall sales generated by a member's branch, or other performance metrics.
* **Security:** The program uses a hardcoded password for authentication. In a real-world application, more robust security measures are essential, including hashing passwords, using a secure authentication system, and protecting sensitive data.
* **Error Handling:** The current error handling is limited. Expanding it to provide more informative error messages, handle invalid input gracefully, and prevent potential crashes or unexpected behavior would improve the program's reliability.

1. **Features:**

* **Feature 1: Add Member:**

This feature enables the addition of new members to the network. Users provide the name of the existing member (recruiter) and the name of the new member. The system verifies the recruiter's existence and whether they have available slots for downlines (a maximum of two). Upon successful validation, the new member is added to the tree, and their recruiter's commission is updated to reflect the new downline.

* **Feature 2: Display Downlines of a Member:**

This allows users to view the immediate subordinates of a specific member. The user inputs a member's name, and the program searches the binary tree to locate that member. It then displays the names, commissions, and levels of all direct downlines of that member.

* **Feature 3: Display Entire Network Tree:**

This feature provides a visual representation of the entire network hierarchy. The program recursively traverses the binary tree and prints each member's name, commission, and level, formatted to show the hierarchical structure clearly

1. **Technologies Used**

* **Programming Languages:**

 C is an imperative procedural language, supporting structured programming, lexical variable scope, and recursion, with a static type system. It was designed to be compiled to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support.

* **Frameworks/Libraries:**

Standard C library (<stdio.h>, <stdlib.h>, <string.h>, <ctype.h>)

* **Tools:**

Code::Blocks

* **Databases:**

None. The data is stored in memory during program execution.

**5. Project Structure**

The project consists of a single file: main.c. There is no complex folder structure. If the project were to be expanded, a more organized structure would be beneficial, e.g.:

**SOURCE CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

typedef struct Member {

char name[50];

int commission;

struct Member\* left;

struct Member\* right;

int level;

} Member;

Member\* createMember(const char\* name, int level) {

Member\* newMember = (Member\*)malloc(sizeof(Member));

if (newMember == NULL) {

fprintf(stderr, "Memory allocation failed!\n");

exit(1);

}

strcpy(newMember->name, name);

newMember->commission = 0;

newMember->left = NULL;

newMember->right = NULL;

newMember->level = level;

return newMember;

}

int memberExists(Member\* root, const char\* name) {

if (root == NULL) return 0;

if (strcmp(root->name, name) == 0) return 1;

return memberExists(root->left, name) || memberExists(root->right, name);

}

void addDownline(Member\* root, const char\* parentName, const char\* newName) {

if (root == NULL) return;

if (strcmp(root->name, parentName) == 0) {

if (root->left == NULL) {

root->left = createMember(newName, root->level + 1);

root->commission += calculateCommission(root->level + 1);

} else if (root->right == NULL) {

root->right = createMember(newName, root->level + 1);

root->commission += calculateCommission(root->level + 1);

} else {

printf("Member %s already has two downlines.\n", parentName);

}

} else {

addDownline(root->left, parentName, newName);

addDownline(root->right, parentName, newName);

}

}

int calculateCommission(int level) {

int baseCommission = 500;

int commissionPercentage;

if (level == 0) commissionPercentage = 50;

else if (level == 1) commissionPercentage = 30;

else if (level == 2) commissionPercentage = 20;

else commissionPercentage = 10;

return (baseCommission \* commissionPercentage) / 100;

}

void displayDownlines(Member\* root, const char\* memberName) {

if (root == NULL) return;

if (strcmp(root->name, memberName) == 0) {

printf("\nDownlines of Member %s:\n", memberName);

if (root->left)

printf(" \nName: %s, Commission: P%d, Level: %d\n", root->left->name, root->left->commission, root->left->level);

if (root->right)

printf(" \nName: %s, Commission: P%d, Level: %d\n", root->right->name, root->right->commission, root->right->level);

} else {

displayDownlines(root->left, memberName);

displayDownlines(root->right, memberName);

}

}

void displayTree(Member\* root, int level) {

if (root == NULL) return;

displayTree(root->right, level + 1);

for (int i = 0; i < level; i++) printf(" ");

printf("\nName: %s, Commission: P%d, Level: %d\n", root->name, root->commission, root->level);

displayTree(root->left, level + 1);

}

void displayMenu() {

//Preserve welcome message

printf("\n O +===========================================================+ O\n");

printf(" oo oo ----> | Welcome to the Otaku Outreach Network | <---- oo oo\n");

printf(" O +===========================================================+ O\n\n");

// Menu remains on screen

printf("+-----------------------------------------------+\n");

printf("| OtaKu Outreach Network Menu |\n");

printf("+-----------------------------------------------+\n");

system("cls");

printf("1. Add Member\n");

printf("2. Display Downlines of a Member\n");

printf("3. Display Entire Network Tree\n");

printf("4. Leave\n");

printf("Enter your choice: ");

}

int authenticateBoss(const char\* password) {

char inputPassword[50];

printf("Enter password: ");

fgets(inputPassword, sizeof(inputPassword), stdin);

inputPassword[strcspn(inputPassword, "\n")] = 0;

return strcmp(inputPassword, password) == 0;

}

void clearInputBuffer() {

int c;

while ((c = getchar()) != '\n' && c != EOF);

}

void freeTree(Member\* root) {

if (root == NULL) return;

freeTree(root->left);

freeTree(root->right);

free(root);

}

int main() {

printf("\n O +===========================================================+ O\n");

printf(" oo oo ----> | Welcome to the Otaku Outreach Network | <---- oo oo\n");

printf(" O +===========================================================+ O\n\n");

char rootName[50];

printf("Enter the name of the Boss: ");

fgets(rootName, sizeof(rootName), stdin);

rootName[strcspn(rootName, "\n")] = 0;

Member\* root = createMember(rootName, 0);

const char\* password = "otaku123";

int authenticated = 0;

int attempts = 0;

while (attempts < 3 && !authenticated) {

system("cls"); // Clear screen

printf("\n+-----------------------------------------------+\n");

printf("| Authenticate as the Boss |\n");

printf("+-----------------------------------------------+\n");

if (authenticateBoss(password)) authenticated = 1;

else {

attempts++;

printf("Invalid password. Please try again.\n");

}

}

if (!authenticated) {

printf("Maximum attempts exceeded. Exiting...\n");

return 0;

}

system("cls"); // Clear screen

printf("\n+-----------------------------------------------+\n");

printf("| Authentication Successful |\n");

printf("+-----------------------------------------------+\n");

int choice;

char parentName[50], newName[50];

while (1) {

displayMenu();

if (scanf("%d", &choice) != 1) {

printf("Invalid input. Please enter a number.\n");

clearInputBuffer();

continue;

}

clearInputBuffer();

switch (choice) {

case 1:

system("cls"); // Clear screen after each operation

printf("\n O +===========================================================+ O\n");

printf(" oo oo ----> | Welcome to the Otaku Outreach Network | <---- oo oo\n");

printf(" O +===========================================================+ O\n\n");

printf("+-----------------------------------------------+\n");

printf("| OtaKu Outreach Network Menu |\n");

printf("+-----------------------------------------------+\n");

printf("Adding Member...\n");

printf("Enter the recruiter's name: ");

fgets(parentName, sizeof(parentName), stdin);

parentName[strcspn(parentName, "\n")] = 0;

if (!memberExists(root, parentName)) {

printf("Member %s does not exist in the network.\n", parentName);

break;

}

printf("Enter the new member's name: ");

fgets(newName, sizeof(newName), stdin);

newName[strcspn(newName, "\n")] = 0;

addDownline(root, parentName, newName);

printf("Member %s added successfully.\n", newName);

break;

case 2:

system("cls"); // Clear screen

printf("\n O +===========================================================+ O\n");

printf(" oo oo ----> | Welcome to the Otaku Outreach Network | <---- oo oo\n");

printf(" O +===========================================================+ O\n\n");

printf("+-----------------------------------------------+\n");

printf("| OtaKu Outreach Network Menu |\n");

printf("+-----------------------------------------------+\n");

printf("Displaying Downlines...\n");

printf("Enter the member's name to display downlines: ");

fgets(parentName, sizeof(parentName), stdin);

parentName[strcspn(parentName, "\n")] = 0;

displayDownlines(root, parentName);

break;

case 3:

system("cls"); // Clear screen

printf("\n O +===========================================================+ O\n");

printf(" oo oo ----> | Welcome to the Otaku Outreach Network | <---- oo oo\n");

printf(" O +===========================================================+ O\n\n");

printf("+-----------------------------------------------+\n");

printf("| OtaKu Outreach Network Menu |\n");

printf("+-----------------------------------------------+\n");

printf("Displaying Entire Tree...\n");

displayTree(root, 0);

break;

case 4:

printf("\n+-----------------------------------------------+\n");

printf("| Leaving the System |\n");

printf("+-----------------------------------------------+\n");

printf("Leaving...\n");

return 0;

freeTree(root);

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

system("pause"); //Pause to view the output before clearing the screen. Remove if not needed.

}

freeTree(root);

return 0;

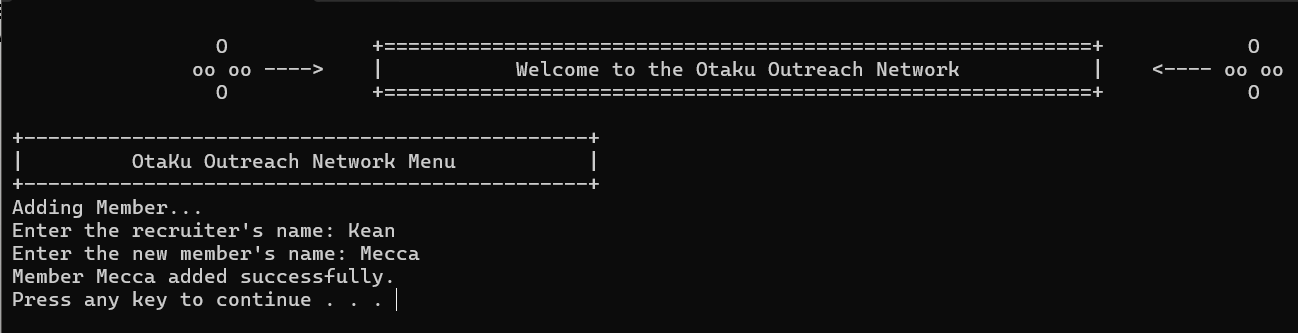
}**6. Usage**

To use the project:

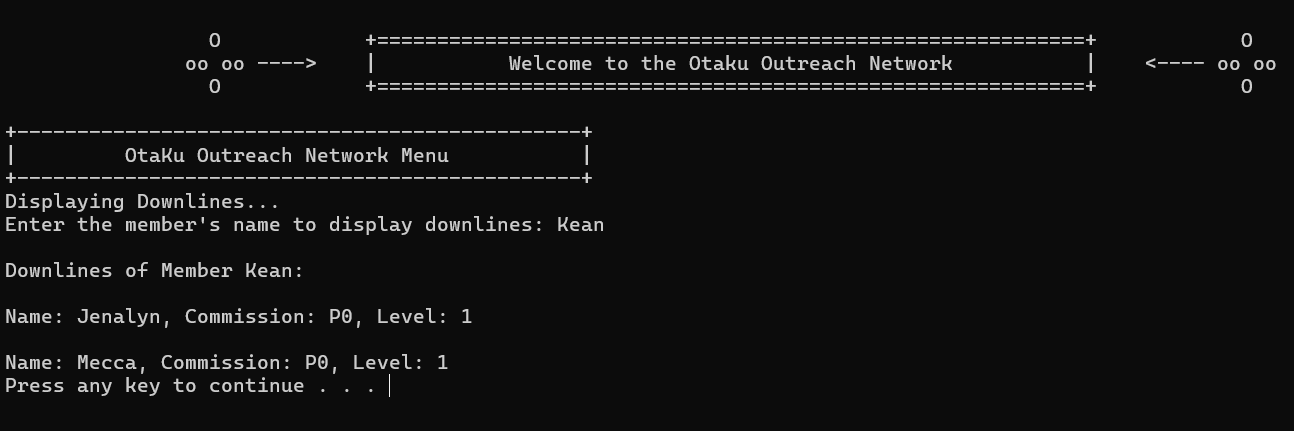
1. **Compile:** Save the code as main.c and compile it using a Code::Blocks compiler.
2. **Run:** Execute the compiled program.
3. **Input:** The program will prompt you to enter the name of the root member. Then, you'll be presented with a menu:
   * **1. Add Member:** Enter the parent's name and the new member's name.
   * **2. Display Downlines:** Enter the name of the member whose downlines you want to see.
   * **3. Display Entire Network Tree:** Shows the entire network hierarchy.
   * **4. Exit:** Terminates the program.

**Example Output:**

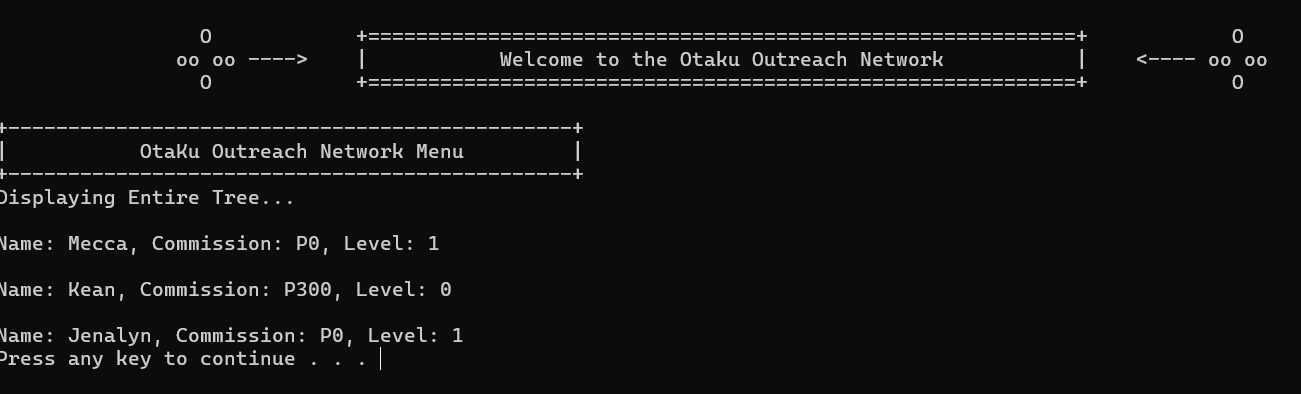
**(After adding several members)**



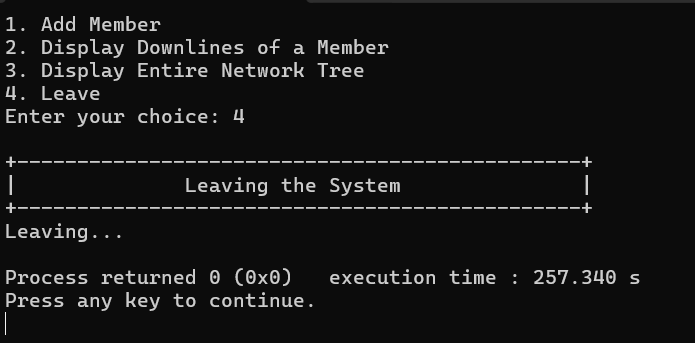
**(Displaying downlines of a member):**

****

**(Displaying Entire Network Tree):**

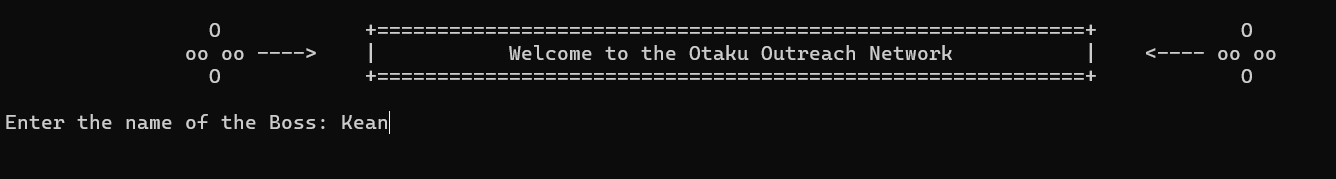
****

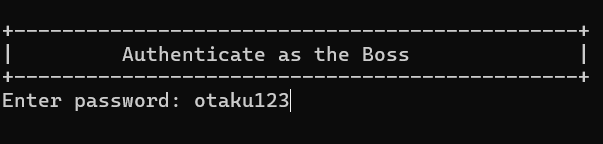
**(Leaving the program):**

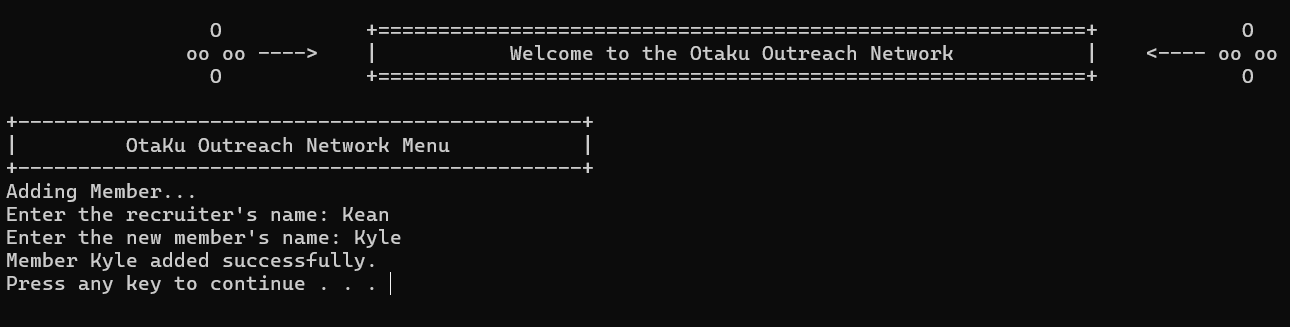
****

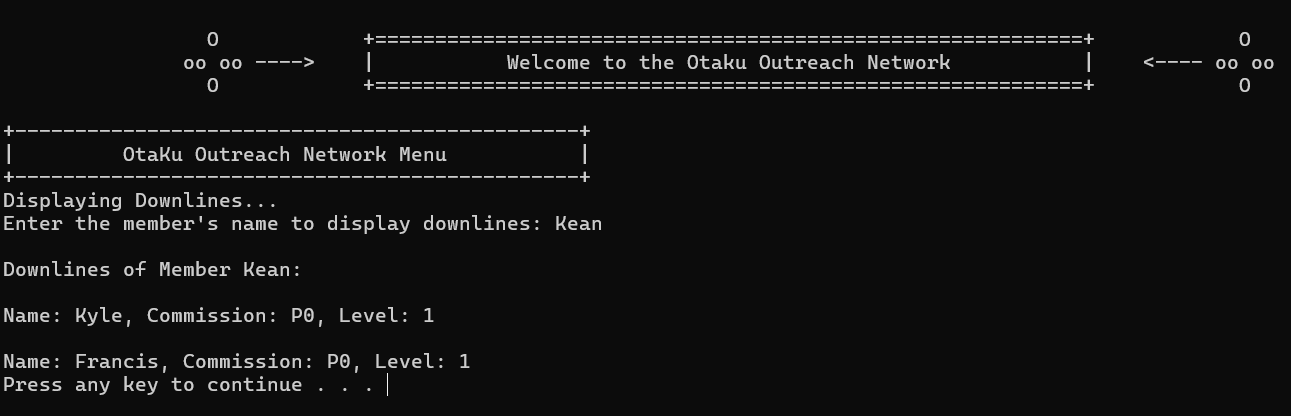
**7. Testing**

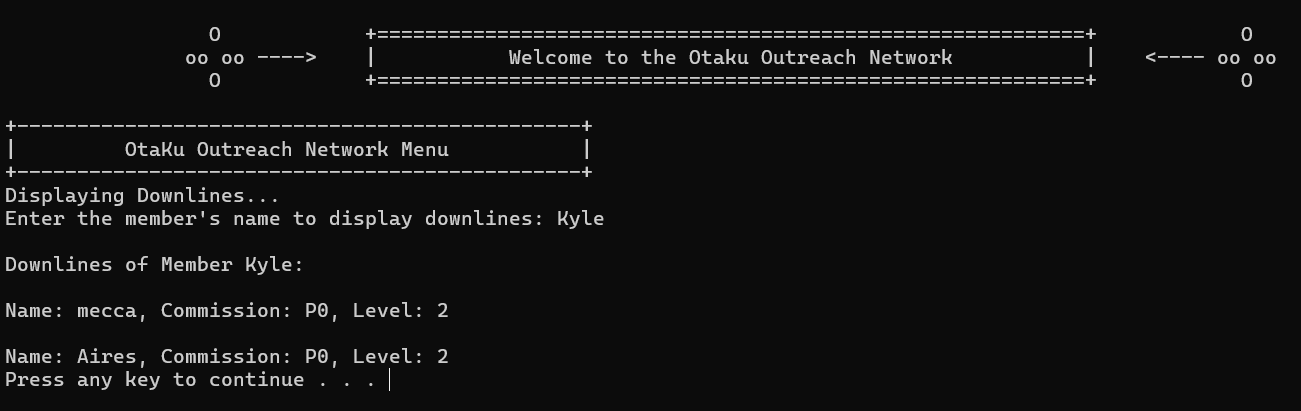
Testing can be done manually by adding members and verifying the output of the displayDownlines and displayTree functions. More rigorous testing would involve using unit testing frameworks (like Unity or CUnit) to automatically test individual functions with various inputs and expected outputs. This is beyond the scope of the current simple implementation.

****

****

****

****

****

****

**9. Contributions**

To contribute to this project :

1. **Fork:** Fork the repository to your own GitHub account.
2. **Clone:** Clone the forked repository to your local machine.
3. **Branch:** Create a new branch for your changes (git checkout -b feature/your-feature).
4. **Code:** Make your changes and commit them (git commit -m "Your descriptive commit message").
5. **Push:** Push your changes to your forked repository (git push origin feature/your-feature).
6. **Pull Request:** Submit a pull request to the original repository.

**10. Acknowledgments**

This project utilizes the standard C library.

**11. Contact Information**

* **Author:** Malinao K., Perez JK.,Quijano M.,Batistin A.,Disabelle J., Banan F.
* **Email:** johnkyleperez26@gmail.com
* **GitHub:** https://github.com/JohnKyle1705