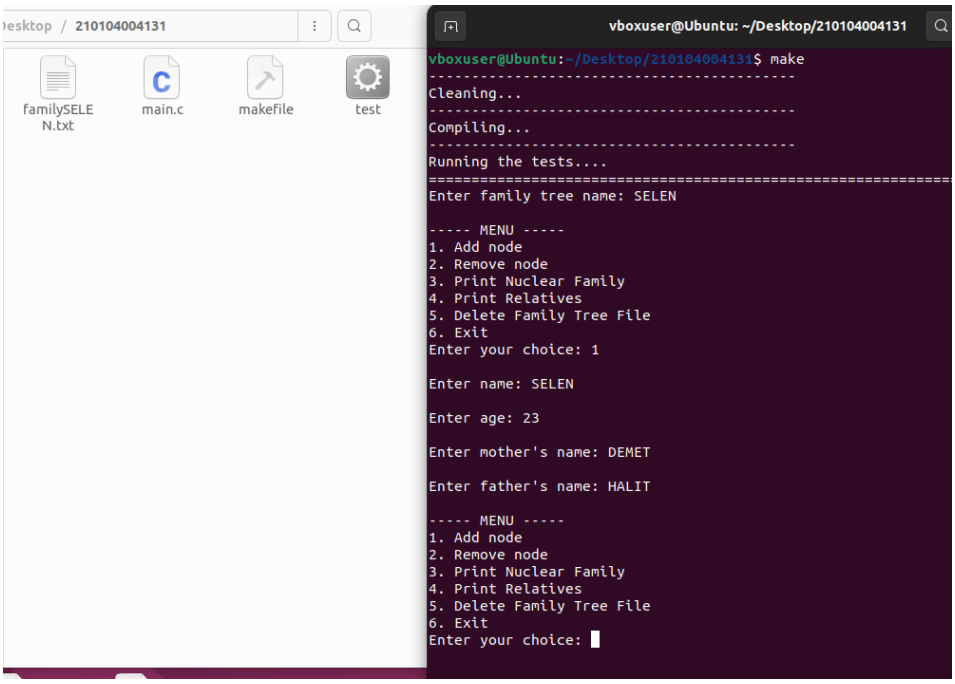
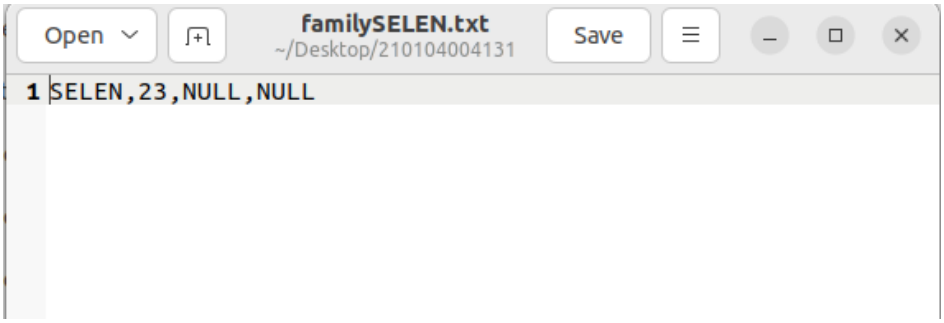
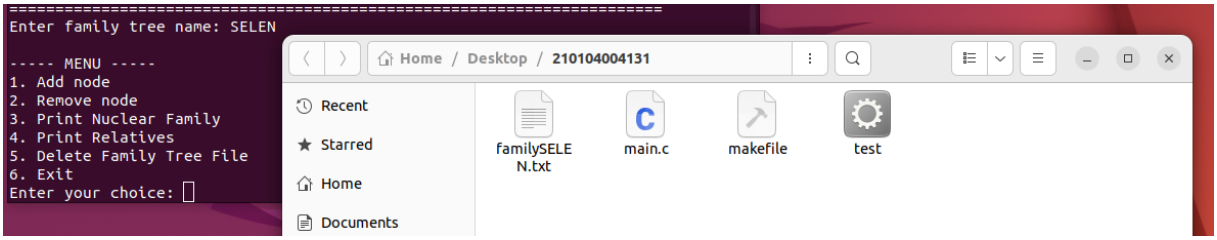
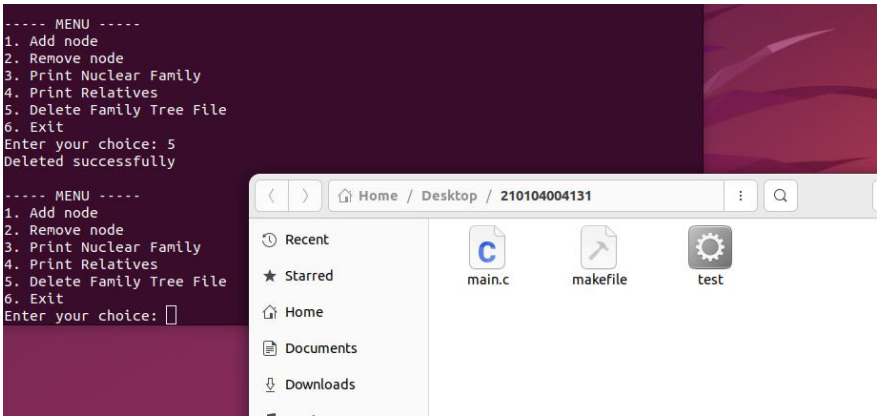
210104004131 SELEN ERDOĞAN

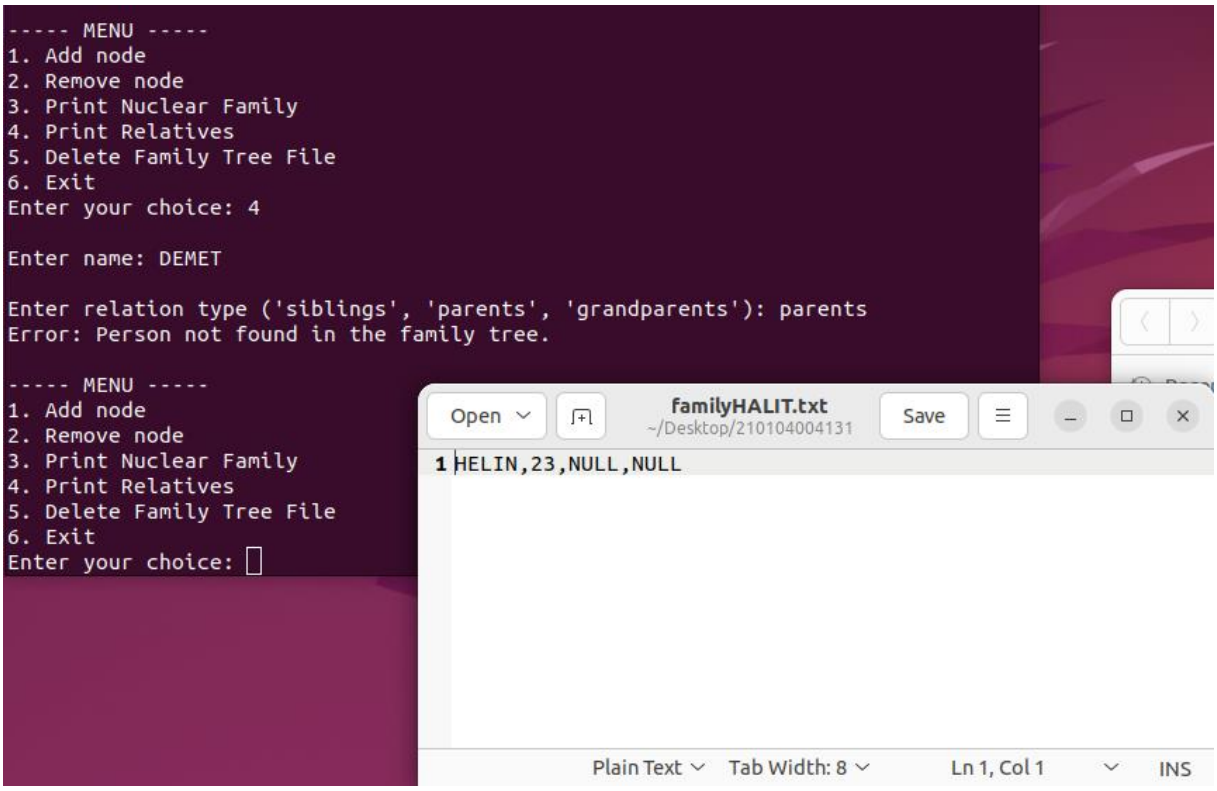




Remove:







Sir, I am not able to complete the homework completely, but I am adding the function structures I wrote below. Some parts of the assignment are missing, but you can see the algorithmic structures I set up below. Since I couldn't write functions exactly in my main function, it was different from the output you gave us, but I edited the main function in order to use the functions I wrote.

```
0
1 // I Created a new node with the given parameters.
2 Node* createNode(char* name, int age, Node* mother, Node* father) {
3     // Allocatedmemory for a new node
4     Node* newNode = (Node*) malloc(sizeof(Node));
5
6     //I Copied  the name to the new node
7     strcpy(newNode->name, name);
8
9     // I Setted the age of the new node
0     newNode->age = age;
1
2     //I Setted the mother pointer of the new node
3     newNode->mother = mother;
4
5     // I Setted the father pointer of the new node
6     newNode->father = father;
7
8     //I Initialized the children pointer of the new node as NULL
9     newNode->children = NULL;
0
1     //I Initialized the next pointer of the new node as NULL
2     newNode->next = NULL;
3
4     //I Returned the newly created node
5     return newNode;
6 }
```

```c
void addNode(Node** root, char name[], int age, char motherName[], char fatherName[], char familyTreeName[]) {
    // Searching for the mother node in the tree
    Node* mother = search(*root, motherName);

    // Searching for the father node in the tree
    Node* father = search(*root, fatherName);

    // If both the mother and father nodes are not found in the tree and the tree is not empty,
    // print an error message and exit the function
    if (mother == NULL && father == NULL && *root != NULL) {
        printf("Parents not found in the tree, unfortuanely. \n");
        return;
    }

    //I Created a new node and fill in the necessary fields
    Node* newNode = createNode(name, age, mother, father);

    // If the tree is empty, set the new node as the root node
    if (*root == NULL) {
        *root = newNode;
    } else {
        // I Added the new node to the mother's children list
        newNode->next = mother->children;
        mother->children = newNode;

        // Add the new node to the father's children list
        newNode->next = father->children;
        father->children = newNode;
    }

    // I Saved the tree with the added node to the file
    saveFamilyTreeToFile(*root, familyTreeName);
}
```

```c
// Searched for a node by its name in the given tree.
Node* search(Node* root, char* name) {
    if (root == NULL) return NULL;

    // Checked if the current node's name matches the target name
    if (strcmp(root->name, name) == 0) return root;

    Node* result = NULL;

    for (Node* child = root->children; child != NULL; child = child->next) {
        result = search(child, name);
        if (result != NULL) break;
    }

    return result;
}

// Recursively delete the given node and its descendants.
void deleteSubtree(Node** node) {
    if (*node == NULL) return;

    Node* currentChild = (*node)->children;

    // Iterate through the children of the current node and recursively delete each child
    while (currentChild != NULL) {
        Node* nextChild = currentChild->next;
        deleteSubtree(&currentChild);
        currentChild = nextChild;
    }

    // Free the memory of the current node and set it to NULL
    free(*node);
    *node = NULL;
}
```

```c
// Write the node and its descendants to the file
void writeNodeToFile(FILE* file, Node* node) {
    if (node == NULL) return;

    // Write the details of the current node to the file
    fprintf(file, "%s,%d,%s,%s\n",
        node->name,
        node->age,
        node->mother ? node->mother->name : "NULL",
        node->father ? node->father->name : "NULL");

    // Recursively call the function for the children of the current node
    Node* child;
    for (child = node->children; child != NULL; child = child->next) {
        writeNodeToFile(file, child);
    }
}


void saveFamilyTreeToFile(Node* root, char* familyTreeName) {
    char filename[25];
    sprintf(filename, "family%s.txt", familyTreeName);

    FILE* file = fopen(filename, "a+");
    if (file == NULL) {
        printf("Could not open file %s\n", filename);
        return;
    }

    // For the root node, we do not print parents, because they are presumably NULL
    fprintf(file, "%s,%d,%s,%s\n", root->name, root->age, root->father ? root->father->name : "NULL", root->mother ? root->mother->name : "NULL");

    writeNodeToFile(file, root);

    fclose(file);
}

void deleteFamilyTreeFile(char* familyTreeName) {
    char filename[25];
    sprintf(filename, "family%s.txt", familyTreeName);

    if (remove(filename) == 0) {
        printf("Deleted successfully\n");
    } else {
        printf("Unable to delete the file\n");
    }
}
```

```c
void loadFamilyTreeFromFile(Node** root, char* familyTreeName) {
    // Create the file name based on the family tree name
    char filename[25];
    sprintf(filename, "family%s.txt", familyTreeName);

    // Opening the file in read mode
    FILE* file = fopen(filename, "r");

    // Check if the file was successfully opened
    if (file == NULL) {
        printf("Could not open file %s\n", filename);
        return;
    }

    // Read each line from the file
    char line[200];
    while (fgets(line, sizeof(line), file)) {
        // Extract the name, age, mother's name, and father's name from the line
        char name[20], motherName[20], fatherName[20];
        int age;

        char *token = strtok(line, ",");
        strcpy(name, token);

        token = strtok(NULL, ",");
        age = atoi(token);

        token = strtok(NULL, ",");
        strcpy(motherName, token);

        token = strtok(NULL, ",");
        strcpy(fatherName, token);

        // Search for the mother and father nodes in the current family tree
        Node* motherNode = search(*root, motherName);
        Node* fatherNode = search(*root, fatherName);

        // If both the mother and father nodes are found, add the new node to the family tree
        if (motherNode != NULL && fatherNode != NULL) {
            addNode(root, name, age, motherNode->name, fatherNode->name, familyTreeName);
        } else {
            printf("CANNOT find parents in the family tree.\n");
        }
    }
```

```c
void printNuclearFamily(Node* root, char* name) {
    Node* person = search(root, name);
    if (person == NULL) {
        printf("Person cannot found in the family tree.\n");
        return;
    }

    printf("Family of %s:\n", name);

    // Print parents
    if (person->mother != NULL) {
        printf("Mother: %s\n", person->mother->name);
    } else {
        printf("Mother: NULL\n");
    }
    if (person->father != NULL) {
        printf("Father: %s\n", person->father->name);
    } else {
        printf("Father: NULL\n");
    }

    // Printinh siblings
    printf("Siblings: ");
    Node* sibling;
    for (sibling = person->mother->children; sibling != NULL; sibling = sibling->next) {
        if (sibling != person) {
            printf("%s ", sibling->name);
        }
    }
    printf("\n");

    // Printing children
    printf("Children: ");
    Node* child;
    for (child = person->children; child != NULL; child = child->next) {
        printf("%s ", child->name);
    }
    printf("\n");
}
```

```c
void printRelatives(Node* root, char* name, char* relationType) {
    Node* person = search(root, name);
    if (person == NULL) {
        printf("Person can not found in the family tree.\n");
        return;
    }

    if (strcmp(relationType, "siblings") == 0) {
        printf("Siblings are %s: ", name);
        Node* sibling;
        for (sibling = person->mother->children; sibling != NULL; sibling = sibling->next) {
            if (sibling != person) {
                printf("%s ", sibling->name);
            }
        }
    } else if (strcmp(relationType, "parents") == 0) {
        printf("Parents of %s: ", name);
        if (person->mother != NULL) {
            printf("%s ", person->mother->name);
        }
        if (person->father != NULL) {
            printf("%s ", person->father->name);
        }
    } else if (strcmp(relationType, "grandparents") == 0) {
        printf("Grandparents of %s: ", name);
        if (person->mother != NULL && person->mother->mother != NULL) {
            printf("%s ", person->mother->mother->name);
        }
        if (person->father != NULL && person->father->father != NULL) {
            printf("%s ", person->father->father->name);
        }
    } else {
        printf("Error of writing. Please enter the correct word'.\n");
    }
    printf("\n");
}
```

```c
void removeNode(Node** root, char name[], char familyTreeName[]) {
    Node* node = search(*root, name);
    if (node == NULL) {
        printf("Node not found in the tree.\n");
        return;
    }

    // Remove node from parent's children list
    if (node->mother != NULL) {
        Node* sibling;
        Node* prev = NULL;
        for (sibling = node->mother->children; sibling != NULL; prev = sibling, sibling = sibling->next) {
            if (sibling == node) {
                if (prev == NULL) {
                    node->mother->children = node->next;
                } else {
                    prev->next = node->next;
                }
                break;
            }
        }
    }

    deleteSubtree(&node);

    // Save the tree to file after removing the node
    saveFamilyTreeToFile(*root, familyTreeName);
```