CSCI 301 Computer Science II Summer 2023

Assignment 8 **Database Maintenance**

Due Date: 11:50 pm on August 1, Tuesday

# 

# Introduction

This project consists of building a program that contains a database to save a person’s name and birthday. The user should be able to manage this database by performing the following operations: adding new contents, removing, modifying, searching people name in the database and saving. The program will save this information into a file that will be entered by the user. Furthermore, the program will display the list of the names and their corresponding birthday in a sorted list.

# Data Structure

To successfully implement this project, some data structure was used. Among them we have a binary Search tree, a couple of classes, some string functions, boolean functions and strings.

# Functions

* AddPerson
* RemovePerson
* ModifyPerson
* SearchPerson
* displayOriginalOrder
* displaySortedOrder
* SaveToFile
* clear

# Structure chart of the main program



# Code list

/\*\*

\* problem description: Write a program that maintains a database

\* containing person’s name and birthday

\* Name: Algassimou Diallo

\* startID: el8524jv

\* Instructor: Jie Meichsner

\* Due date: 08/01/2023

\*/

#ifndef BINARY\_SEARCH\_TREE\_H

#define BINARY\_SEARCH\_TREE\_H

#include <functional>

#include <string>

using namespace std;

/\*\*

\* @class BinarySearchTree

\* @brief A templated binary search tree data structure.

\* @tparam T The type of elements stored in the tree.

\*/

template <class T>

class BinarySearchTree {

private:

/\*\*

\* @struct Node

\* @brief Represents a node in the binary search tree.

\*/

struct Node {

T data; /\*\*< The data stored in the node. \*/

Node\* left; /\*\*< Pointer to the left child node. \*/

Node\* right; /\*\*< Pointer to the right child node. \*/

/\*\*

\* @brief Constructor to create a new node with the given value.

\* @param val The value to be stored in the node.

\*/

Node(const T& val) : data(val), left(nullptr), right(nullptr) {}

};

Node\* root; /\*\*< Pointer to the root node of the binary search tree. \*/

void insertNode(Node\*& node, const T& value) {

if (node == nullptr) {

node = new Node(value);

} else if (value < node->data) {

insertNode(node->left, value);

} else if (value > node->data) {

insertNode(node->right, value);

}

}

bool containsNode(const Node\* node, const T& value) const {

if (node == nullptr) {

return false;

} else if (value == node->data) {

return true;

} else if (value < node->data) {

return containsNode(node->left, value);

} else {

return containsNode(node->right, value);

}

}

void removeNode(Node\*& node, const T& value) {

if (node == nullptr) {

return;

}

if (value < node->data) {

removeNode(node->left, value);

} else if (value > node->data) {

removeNode(node->right, value);

} else {

if (node->left == nullptr) {

Node\* temp = node;

node = node->right;

delete temp;

} else if (node->right == nullptr) {

Node\* temp = node;

node = node->left;

delete temp;

} else {

Node\* temp = findMinNode(node->right);

node->data = temp->data;

removeNode(node->right, temp->data);

}

}

}

Node\* findMinNode(Node\* node) const {

while (node->left != nullptr) {

node = node->left;

}

return node;

}

void traverseInorder(const Node\* node, const function<void(const T&)>& visit) const {

if (node != nullptr) {

traverseInorder(node->left, visit);

visit(node->data);

traverseInorder(node->right, visit);

}

}

void traversePreorder(const Node\* node, const function<void(const T&)>& visit) const {

if (node != nullptr) {

visit(node->data);

traversePreorder(node->left, visit);

traversePreorder(node->right, visit);

}

}

public:

/\*\*

\* @brief Default constructor to create an empty binary search tree.

\*/

BinarySearchTree() : root(nullptr) {}

/\*\*

\* @brief Inserts a new value into the binary search tree.

\* @param value The value to be inserted.

\*/

void insert(const T& value) {

insertNode(root, value);

}

/\*\*

\* @brief Checks if the binary search tree contains a specific value.

\* @param value The value to search for.

\* @return True if the value is found; otherwise, false.

\*/

bool contains(const T& value) const {

return containsNode(root, value);

}

/\*\*

\* @brief Removes a value from the binary search tree.

\* @param value The value to be removed.

\*/

void remove(const T& value) {

removeNode(root, value);

}

/\*\*

\* @brief Traverses the binary search tree in inorder and performs an action on each node's value.

\* @param visit The function to be called on each node's value during the inorder traversal.

\*/

void traverseInorder(const function<void(const T&)>& visit) const {

traverseInorder(root, visit);

}

/\*\*

\* @brief Traverses the binary search tree in preorder and performs an action on each node's value.

\* @param visit The function to be called on each node's value during the preorder traversal.

\*/

void traversePreorder(const function<void(const T&)>& visit) const {

traversePreorder(root, visit);

}

/\*\*

\* @brief Destructor to deallocate memory used by the binary search tree.

\*/

~BinarySearchTree() {

clear(root);

}

private:

/\*\*

\* @brief Recursively deallocates memory used by the binary search tree.

\* @param node The current node being considered during the deallocation process.

\*/

void clear(Node\* node) {

if (node != nullptr) {

clear(node->left);

clear(node->right);

delete node;

}

}

};

#endif

/\*\*

\* problem description: Write a program that maintains a database

\* containing person’s name and birthday

\* Name: Algassimou Diallo

\* startID: el8524jv

\* Instructor: Jie Meichsner

\* Due date: 08/01/2023

\*/

#ifndef DATE\_H

#define DATE\_H

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

/\*\*

\* @class Date

\* @brief Represents a date with month, day, and year components.

\*/

class Date {

private:

int month; /\*\*< The month component of the date. \*/

int day; /\*\*< The day component of the date. \*/

int year; /\*\*< The year component of the date. \*/

public:

/\*\*

\* @brief Default constructor. Initializes date to all zeros.

\*/

Date() {}

/\*\*

\* @brief Parameterized constructor to initialize a Date object.

\* @param m The month component of the date.

\* @param d The day component of the date.

\* @param y The year component of the date.

\*/

Date(int m, int d, int y) : month(m), day(d), year(y) {}

/\*\*

\* @brief Sets the month component of the date.

\* @param m The month to set.

\*/

void setMonth(int m) { month = m; }

/\*\*

\* @brief Sets the day component of the date.

\* @param d The day to set.

\*/

void setDay(int d) { day = d; }

/\*\*

\* @brief Sets the year component of the date.

\* @param y The year to set.

\*/

void setYear(int y) { year = y; }

/\*\*

\* @brief Gets the month component of the date.

\* @return The month component.

\*/

int getMonth() const { return month; }

/\*\*

\* @brief Gets the day component of the date.

\* @return The day component.

\*/

int getDay() const { return day; }

/\*\*

\* @brief Gets the year component of the date.

\* @return The year component.

\*/

int getYear() const { return year; }

/\*\*

\* @brief Overloaded output stream operator to print the date in "MM DD YYYY" format.

\* @param os The output stream object.

\* @param date The Date object to be printed.

\* @return The output stream after printing the date.

\*/

friend ostream& operator<<(ostream& os, const Date& date) {

os << date.month << " " << date.day << " " << date.year;

return os;

}

/\*\*

\* @brief Overloaded input stream operator to read the date in "MM DD YYYY" format.

\* @param is The input stream object.

\* @param date The Date object to store the read date.

\* @return The input stream after reading the date.

\*/

friend istream& operator>>(istream& is, Date& date) {

is >> date.month >> date.day >> date.year;

return is;

}

};

#endif

/\*\*

\* problem description: Write a program that maintains a database

\* containing person’s name and birthday

\* Name: Algassimou Diallo

\* startID: el8524jv

\* Instructor: Jie Meichsner

\* Due date: 08/01/2023

\*/

#ifndef PERSON\_H

#define PERSON\_H

#include "Date.h"

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

/\*\*

\* @class Person

\* @brief Represents a person with a name and birthday.

\*/

class Person {

private:

string name; /\*\*< The name of the person. \*/

Date birthday; /\*\*< The birthday of the person. \*/

public:

/\*\*

\* @brief Default constructor. Initializes name and birthday to empty.

\*/

Person() {}

/\*\*

\* @brief Parameterized constructor to initialize a Person object.

\* @param n The name of the person.

\* @param b The birthday of the person (as a Date object).

\*/

Person(string n, Date b) : name(n), birthday(b) {}

/\*\*

\* @brief Sets the name of the person.

\* @param n The name to set.

\*/

void setName(string n) { name = n; }

/\*\*

\* @brief Sets the birthday of the person.

\* @param b The birthday to set (as a Date object).

\*/

void setBirthday(Date b) { birthday = b; }

/\*\*

\* @brief Gets the name of the person.

\* @return The name of the person.

\*/

string getName() const { return name; }

/\*\*

\* @brief Gets the birthday of the person.

\* @return The birthday of the person (as a Date object).

\*/

Date getBirthday() const { return birthday; }

/\*\*

\* @brief Overloaded equality operator to compare two Person objects.

\* @param other The other Person object to compare with.

\* @return True if the names are equal; otherwise, false.

\*/

bool operator==(const Person& other) const { return name == other.name; }

/\*\*

\* @brief Overloaded less than operator to compare two Person objects.

\* @param other The other Person object to compare with.

\* @return True if the name of this person is less than the name of the other person; otherwise, false.

\*/

bool operator<(const Person& other) const { return name < other.name; }

/\*\*

\* @brief Overloaded greater than operator to compare two Person objects.

\* @param other The other Person object to compare with.

\* @return True if the name of this person is greater than the name of the other person; otherwise, false.

\*/

bool operator>(const Person& other) const { return name > other.name; }

/\*\*

\* @brief Overloaded output stream operator to print the person's name and birthday.

\* @param os The output stream object.

\* @param person The Person object to be printed.

\* @return The output stream after printing the person's name and birthday.

\*/

friend ostream& operator<<(ostream& os, const Person& person) {

os << person.name << " " << person.birthday;

return os;

}

/\*\*

\* @brief Overloaded input stream operator to read the person's name and birthday.

\* @param is The input stream object.

\* @param person The Person object to store the read name and birthday.

\* @return The input stream after reading the person's name and birthday.

\*/

friend istream& operator>>(istream& is, Person& person) {

is >> person.name >> person.birthday;

return is;

}

};

#endif

/\*\*

\* problem description: Write a program that maintains a database

\* containing person’s name and birthday

\* Name: Algassimou Diallo

\* startID: el8524jv

\* Instructor: Jie Meichsner

\* Due date: 08/01/2023

\*/

#ifndef PEOPLEDB\_H

#define PEOPLEDB\_H

#include <iostream>

#include <fstream>

#include <string>

#include "BinarySearchTree.h"

using namespace std;

/\*\*

\* @class PeopleDB

\* @brief Represents a database of people and their information.

\*/

class PeopleDB {

private:

BinarySearchTree<Person> db; /\*\*< Binary search tree to store people data. \*/

public:

/\*\*

\* @brief Constructs a PeopleDB object and loads data from a file.

\* @param filename The name of the file containing initial people list.

\*/

PeopleDB(string filename) {

ifstream file(filename);

if (file.is\_open()) {

Person person;

while (file >> person) {

db.insert(person);

}

file.close();

} else {

cerr << "Error: Unable to open file " << filename << endl;

}

}

/\*\*

\* @brief Adds a new person to the database.

\* @param person The person to be added.

\*/

void addPerson(Person person) {

db.insert(person);

}

/\*\*

\* @brief Removes a person from the database.

\* @param person The person to be removed.

\*/

void removePerson(Person person) {

db.remove(person);

}

/\*\*

\* @brief Modifies a person's information in the database.

\* @param oldPerson The person to be modified.

\* @param newPerson The updated information for the person.

\*/

void modifyPerson(Person oldPerson, Person newPerson) {

db.remove(oldPerson);

db.insert(newPerson);

}

/\*\*

\* @brief Searches for a person in the database.

\* @param person The person to search for.

\* @return True if the person is found, false otherwise.

\*/

bool searchPerson(Person person) {

return db.contains(person);

}

/\*\*

\* @brief Displays the database in its original order.

\*/

void displayOriginalOrder() const {

db.traverseInorder([](const Person& person) {

cout << person << endl;

});

}

/\*\*

\* @brief Displays the database sorted by names.

\*/

void displaySortedOrder() const {

db.traverseInorder([](const Person& person) {

cout << person << endl;

});

}

/\*\*

\* @brief Saves the database to a file.

\* @param filename The name of the file to save the data.

\*/

void saveToFile(string filename) const {

ofstream file(filename);

if (file.is\_open()) {

db.traverseInorder([&](const Person& person) {

file << person << endl;

});

file.close();

} else {

cerr << "Error: Unable to open file " << filename << endl;

}

}

};

#endif

/\*\*

\* problem description: Write a program that maintains a database

\* containing person’s name and birthday

\* Name: Algassimou Diallo

\* startID: el8524jv

\* Instructor: Jie Meichsner

\* Due date: 08/01/2023

\*/

#include <iostream>

#include <fstream>

#include <string>

#include "Date.h"

#include "Person.h"

#include "PeopleDB.h"

#include "BinarySearchTree.h"

using namespace std;

/\*\* Main program \*/

int main() {

cout << "Welcome to xxx database system. Please enter the file that contains the initial people list: ";

string inputFileName;

cin >> inputFileName;

PeopleDB peopleDB(inputFileName);

cout << "The initial database built from the input file is displayed by its original order:\n";

peopleDB.displayOriginalOrder();

int choice = 0;

while (choice != 7) {

cout << "Please enter your option\n"

<< "1. Add a new person\n"

<< "2. Remove a person\n"

<< "3. Modify the database\n"

<< "4. Search for a person in the database\n"

<< "5. Display the database\n"

<< "6. Display the database sorted by names\n"

<< "7. Quit and save the database to a file\n"

<< "-->";

cin >> choice;

switch (choice) {

case 1: {

Person person;

cout << "To add, enter name and birthday (month day year):\n";

cin >> person;

peopleDB.addPerson(person);

char addAnother;

cout << "Add another one? ";

cin >> addAnother;

while (tolower(addAnother) == 'y') {

cout << "Enter name and birthday (month day year):\n";

cin >> person;

peopleDB.addPerson(person);

cout << "Add another one? ";

cin >> addAnother;

}

break;

}

case 2: {

Person person;

string name;

cout << "Enter name to remove: ";

cin >> name;

person.setName(name);

if (peopleDB.searchPerson(person)) {

peopleDB.removePerson(person);

cout << "Person removed.\n";

} else {

cout << "Person not found.\n";

}

break;

}

case 3: {

Person oldPerson, newPerson;

string name;

cout << "Enter name to modify: ";

cin >> name;

oldPerson.setName(name);

if (peopleDB.searchPerson(oldPerson)) {

cout << "Enter new name and birthday (month day year): ";

cin >> newPerson;

peopleDB.modifyPerson(oldPerson, newPerson);

cout << "Person modified.\n";

} else {

cout << "Person not found.\n";

}

break;

}

case 4: {

Person person;

string name;

cout << "Enter name to search: ";

cin >> name;

person.setName(name);

if (peopleDB.searchPerson(person)) {

cout << "Person found.\n";

} else {

cout << "Person not found.\n";

}

break;

}

case 5: {

cout << "Displaying the database:\n";

peopleDB.displayOriginalOrder();

break;

}

case 6: {

cout << "Displaying the database sorted by names:\n";

peopleDB.displaySortedOrder();

break;

}

case 7: {

cout << "Exiting and saving the database...\n";

peopleDB.saveToFile(inputFileName);

break;

}

default: {

cout << "Invalid choice.\n";

break;

}

}

}

return 0;

}

# User Document

To successfully run this program, the user will be required to enter a file name in the txt format (e.g people.txt). This file will be used by the program to display existing contents and also to save newly added people. The program will display the current list of people in the database and also the following menu

Please enter your option

1. Add a new person

2. Remove a person

3. Modify the database

4. Search for a person in the database

5. Display the database

6. Display the database sorted by names

7. Quit and save the database to a file

From these options, the user will choose the operation that he intends to perform.

The files of this project are located in the directory el8524jv/csci301/project8. To successfully compile this program in centOS, the user must run the following script:

g++ -std=c++11 main.cpp. A file named a.out will be automatically generated.

Test cases

| Tests | Inputs | Outputs |
| --- | --- | --- |
| 1. Add a person | To add, enter name and birthday (month day year):  John 3 3 3333  Add another one? y  Enter name and birthday (month day year):  Cody 7 7 7777  Add another one? n | Displaying the database:  Alga 3 4 2000  Cody 7 7 7777  Jack 5 5 5555  Jane 2 2 2222  John 3 3 3333  Mari 4 5 2000  Mark 5 7 1990 |
| 1. Remove a person | -->2  Enter name to remove: Jane 2 2 2222 | Person removed.  Please enter your option  1. Add a new person  2. Remove a person  3. Modify the database  4. Search for a person in the database  5. Display the database  6. Display the database sorted by names  7. Quit and save the database to a file  -->5  Displaying the database:  Alga 3 4 2000  Jack 5 5 5555  Mari 4 5 2000  Mark 5 7 1990 |
| 1. Modify the database | -->3^M  Enter name to modify: Gass^H ^H 3 3 3333^M  Enter new name and birthday (month day year): Kakou 1 1 1111 | Person modified |
| 1. Search for a person | -->4  Enter name to search: Alga 3 4 2000  -->4  Enter name to search: Gas 1 1 1111 | Person found  Person not found. |
| 1. Display the database | -->1  To add, enter name and birthday (month day year):  Brinda 1 1 1111  -->5 | Displaying the database:  Alga 3 4 2000  Brinda 1 1 1111  Jack 5 5 5555  Jane 2 2 2222  Mari 4 5 2000  Mark 5 7 1990 |
| 1. Display the database sorted by names | -->1  To add, enter name and birthday (month day year):  Dane 4 4 4444  -->6 | Displaying the database sorted by names:  Alga 3 4 2000  Dane 4 4 4444  Jack 5 5 5555  Jane 2 2 2222  Mari 4 5 2000  Mark 5 7 1990 |
| 1. Quit and save the database to a file | -->1  To add, enter name and birthday (month day year):  Cody 2 30 1990  Add another one? y  Enter name and birthday (month day year):  Amy 12 12 1534  Add another one? n | -->5  Displaying the database:  Alga 3 4 2000  Amy 12 12 1534  Cody 2 30 1990  Dane 8 8 8888  Gas 3 3 3333  Jack 5 5 5555  Jane 2 2 2222  Mari 4 5 2000  Mark 5 7 1990  bob 9 9 9999 |

# Summary

This project was successfully implemented in our local computer using VSCode and then transferred in the centOS. We have registered no failures while compiling this program. This program takes a file containing people's names and their birthday and displays it to the user. The user uses this file to store the database operations that he will perform in this program such as adding, removing, searching existing people in the database etc. Any modification can be saved in the database and displayed later. To get more details on the test cases shown in the table above, please refer to the scripts in the project8 directory in he centOS.