សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ

**ROYAL UNIVERSITY OF PHNOM PENH**

ប្រធានបទរបាយការណ៍ស្រាវជ្រាវ

**Title in English**

**EXAM MANAGEMENT SYSTEM**

**WITH BINARY SEARCH TREE**

**Group 2:**

SAING SOPHEAK

SOU SODARA

SAN THIS SAK KHAKNA

SOK CHEAPANHA

SROR LYPHENG

OTH KOLTHIDA

**August 2023**

**TABLE OF CONTENTS**

**Page**

Table of Contents (ii)

CHAPTER 1 INTRODUCTION (1)

CHAPTER 2 METHOD AND RESULT (2)

2.1 Planning (2)

2.2 Analysis (2)

2.3 Design (3)

2.4 Implement (6)

2.5 Testing (14)

CHAPTER 3 ALGORITHM (6)

**CHAPTER 1**

**INTRODUCTION**

**Data Structure** is a structure of data that is stored and organized in a computer that can be efficiently accessed and manipulated.

**Data structures** typically provide different methods for accessing and modifying the data, which allows for improved performance and efficiency in various operations.

We divide **Data Structure** into **linear** and **non-linear** data structures. **Linear Data** structures include arrays, linked lists, stacks, and queues. However, **Non-Linear data** structures include Trees, Graphs, and Hash tables.

On the other hand, Algorithm is the sequence of steps or rules to solve a specific problem.

In this project, we applied Data Structure and Algorithms to solve one real problem: Exam Management.

**CHAPTER 2**

**METHOD AND RESULT**

# 2.1 Planning

We can see various in our society but we would scope our project into the education field. Our application is about an exam management system which specifically use for teachers.

Exam management is a critical aspect of a teacher's role in educational institutions. It involves various tasks such as creating, updating, retrieving, and deleting exam-related data. One effective approach to perform these tasks is by utilizing CRUD operations, which stands for Create, Read, Update, and Delete.

# 2.2 Analysis

 We use C++ programming for our programming. We think we would use Binary Tree to do operations on our application. In this application, teachers will be able to:

* **Create Exam**: Teacher can include information of exam including “**subject”**, “**duration**”, “**start date”, and** “**end date”** and the system will calculate the“**duration”** for them.
* **Find Exam**: Teacher can search for any exams including “**subject”, “Data**”, and “**duration**”.
* **Update Exam**: Teachers can update all information as they want that is available in the exam.

**Delete Exam:** Teachers can delete exam.

# 2.3 Design

The code consists of three classes: Exam, Node, and ExamManagementSystem.

The **Exam class** defines the attributes of an exam and has a constructor to initialize them.

class Exam

{

public:

    string subject;

    string date;

    string time;

    int duration;

    string location;

    Exam(const string &sub, const string &dt, const string &t, int dur, const string &loc)

        : subject(sub), date(dt), time(t), duration(dur), location(loc) {}

};

The **Node class** defines a node structure for the binary search tree, holding an Exam and pointers to left and right nodes.

class Node

{

public:

    Exam data;

    Node \*left;

    Node \*right;

    Node(const Exam &exam) : data(exam), left(nullptr), right(nullptr) {}

};

The **ExamManagementSystem class** includes private methods for insertion, search, finding minimum, and removal in the binary search tree. Public methods are used to add, find, update, and delete exams.

class ExamManagementSystem

{

private:

    Node \*root;

    Node \*insert(Node \*root, const Exam &exam)

    {

        ...Code...

    }

    Node \*search(Node \*root, const std::string &subject)

    {

        ...Code...

    }

    Node \*findMin(Node \*node)

    {

        ...Code...

    }

    Node \*remove(Node \*root, const std::string &subject)

    {

        ...Code...

    }

public:

    ExamManagementSystem() : root(nullptr) {}

    void addExam(const Exam &exam)

    {

        ...Code...

    }

    Exam \*findExam(const string &subject)

    {

        ...Code...

    }

    void updateExam(const string &subject, const Exam &newExam)

    {

        ...Code...

    }

    void deleteExam(const string &subject)

    {

        ...Code...

    }

};

The main function drives the user interaction loop for managing exams. It displays a menu, takes user input, and performs actions based on the chosen option.

int main() {

    int choice;

    ExamManagementSystem ems;

    while (true) {

        cout << endl;

        cout << "-----------Exam Management System-----------\n";

        cout << "1. Create Exam\n";

        cout << "2. Find Exam\n";

        cout << "3. Update Exam\n";

        cout << "4. Delete Exam\n";

        cout << "5. Exit\n";

        cout << "--------------------------------------------\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

        case 1:

        {

            ...Code...

        }

        case 2:

        {

            ...Code...

        }

        case 3:

        {

            ...Code...

        }

        case 4:

        {

            ...Code...

        }

        case 5:

        {

            ...Code...

        }

        default:

        {

            ...Code...

        }

        }

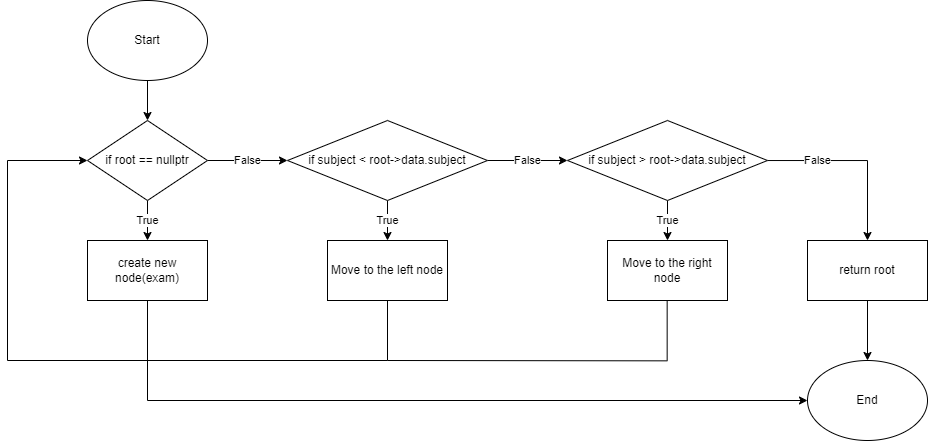
    }

    return 0;

}

# 2.4 Implementation

1. **INSERT NEW NODE**



void addExam(const Exam &exam)

    {

        root = insert(root, exam);

    }

Node \*insert(Node \*root, const Exam &exam)

    {

        if (root == nullptr)

        {

            return new Node(exam);

        }

        if (exam.subject < root->data.subject)

        {

            root->left = insert(root->left, exam);

        }

        else if (exam.subject > root->data.subject)

        {

            root->right = insert(root->right, exam);

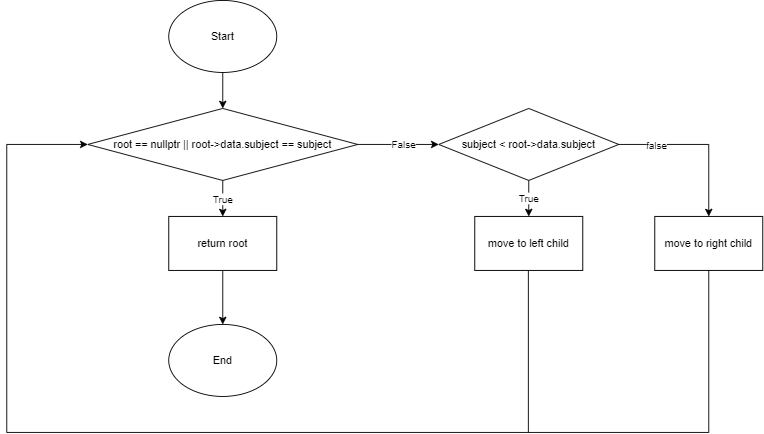
        }

        return root;

    }

This function inserts a new `Exam` object into the binary search tree. It recursively traverses the tree based on the subject's alphabetical order. If the subject is less than the current node's subject, it moves to the left subtree; if greater, it moves to the right subtree. If the subtree is empty, a new node is created with the provided `Exam` object.

1. **SEARCH NODE**

****

Exam \*findExam(const string &subject)

    {

        Node \*result = search(root, subject);

        return (result != nullptr) ? &(result->data) : nullptr;

    }

Node \*search(Node \*root, const std::string &subject)

    {

        if (root == nullptr || root->data.subject == subject)

        {

            return root;

        }

        if (subject < root->data.subject)

        {

            return search(root->left, subject);

        }

        return search(root->right, subject);

    }

This function searches for an `Exam` object with a specific subject in the binary search tree. It recursively traverses the tree, comparing the given subject with the current node's subject. If the subject is found, the function returns a pointer to the node containing the exam. If the subject is less or greater, the function continues the search in the appropriate subtree.

1. **REMOVE NODE**



void deleteExam(const string &subject)

    {

        root = remove(root, subject);

    }

    Node \*remove(Node \*root, const std::string &subject)

    {

        if (root == nullptr)

        {

            return root;

        }

        if (subject < root->data.subject)

        {

            root->left = remove(root->left, subject);

        }

        else if (subject > root->data.subject)

        {

            root->right = remove(root->right, subject);

        }

        else

        {

            if (root->left == nullptr)

            {

                Node \*temp = root->right;

                delete root;

                return temp;

            }

            else if (root->right == nullptr)

            {

                Node \*temp = root->left;

                delete root;

                return temp;

            }

            Node \*temp = findMin(root->right);

            root->data = temp->data;

            root->right = remove(root->right, temp->data.subject);

        }

        return root;

    }

This function removes an `Exam` object with a specific subject from the binary search tree. It uses recursive calls to traverse the tree and locate the node to remove. Once the node is found, the function handles three cases: no children, one child, and two children. For nodes with two children, it replaces the node's data with the minimum value from the right subtree, and then removes the corresponding node.

1. **UPDATE NODE**

    void updateExam(const string &subject, const Exam &newExam)

    {

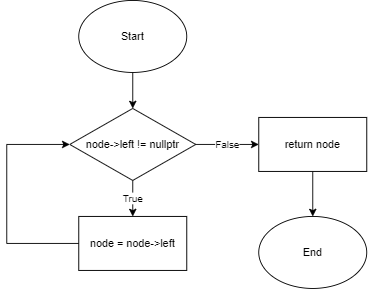
        root = remove(root, subject);

        root = insert(root, newExam);

    }

This public method enables users to update an existing exam's details. It first removes the old exam with the specified subject using the `remove` function. Then, it inserts the new exam details using the `insert` function, effectively updating the exam information.

1. **FIND MIN VALUE FROM THE RIGHT SUBTREE (SUPPORT FUNCTION FOR DELETE)**



    Node \*findMin(Node \*node)

    {

        while (node->left != nullptr)

        {

            node = node->left;

        }

        return node;

    }

This function finds the minimum node (leftmost node) in a subtree. It traverses the left subtree of the given node until it reaches the leftmost leaf node, which contains the smallest subject in that subtree.

1. **MAIN FUNCTION**

int main()

{

    int choice;

    ExamManagementSystem ems;

    while (true)

    {

        cout << endl;

        cout << "-----------Exam Management System-----------\n";

        cout << "1. Create Exam\n";

        cout << "2. Find Exam\n";

        cout << "3. Update Exam\n";

        cout << "4. Delete Exam\n";

        cout << "5. Exit\n";

        cout << "--------------------------------------------\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            string inputSubject, inputDate, inputTime, inputLocation;

            int inputDuration;

            cout << "Enter subject: ";

            cin >> inputSubject;

            cout << "Enter date: ";

            cin >> inputDate;

            cout << "Enter time: ";

            cin >> inputTime;

            cout << "Enter duration: ";

            cin >> inputDuration;

            cout << "Enter location: ";

            cin >> inputLocation;

            ems.addExam(Exam(inputSubject, inputDate, inputTime, inputDuration, inputLocation));

            break;

        }

        case 2:

        {

            string inputSubject;

            cout << "Enter subject: ";

            cin >> inputSubject;

            Exam \*exam = ems.findExam(inputSubject);

            cout << endl;

            if (exam != nullptr)

            {

                cout << "-----------------------------------------------------------\n";

                cout << setw(10) << left << "Subject" << setw(15) << "Date" << setw(15) << "Time" << setw(10) << "Duration" << setw(10) << "Location"

                     << "\n";

                cout << "-----------------------------------------------------------\n";

                cout << setw(10) << left << exam->subject << setw(15) << exam->date << setw(15) << exam->time << setw(10) << exam->duration << setw(10) << exam->location << "\n";

                cout << "-----------------------------------------------------------\n";

            }

            else

            {

                cout << "Exam Not Found\n";

            }

            break;

        }

        case 3:

        {

            string inputSubject;

            cout << "Enter subject: ";

            cin >> inputSubject;

            Exam \*examToUpdate = ems.findExam(inputSubject);

            cout << endl;

            if (examToUpdate != nullptr)

            {

                cout << "----------Update----------\n";

                cout << "1. Update subject\n";

                cout << "2. Update date\n";

                cout << "3. Update time\n";

                cout << "4. Update duration\n";

                cout << "5. Update location\n";

                cout << "Enter choice: ";

                int inputChoice;

                cin >> inputChoice;

                switch (inputChoice)

                {

                case 1:

                    cout << "Enter new subject: ";

                    cin >> examToUpdate->subject;

                    break;

                case 2:

                    cout << "Enter new date: ";

                    cin >> examToUpdate->date;

                    break;

                case 3:

                    cout << "Enter new time: ";

                    cin >> examToUpdate->time;

                    break;

                case 4:

                    cout << "Enter new duration: ";

                    cin >> examToUpdate->duration;

                    break;

                case 5:

                    cout << "Enter new location: ";

                    cin >> examToUpdate->location;

                    break;

                default:

                    cout << "Invalid choice\n";

                    break;

                }

                ems.updateExam(inputSubject, \*examToUpdate);

                cout << "Exam fields updated.\n";

                break;

            }

            else

            {

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

            }

            break;

        }

        case 4:

        {

            string inputSubject;

            cout << "Enter subject: ";

            cin >> inputSubject;

            Exam \*exam = ems.findExam(inputSubject);

            cout << endl;

            if (exam != nullptr)

            {

                ems.deleteExam(inputSubject);

                cout << "Exam deleted.\n";

            }

            else

            {

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

            }

            break;

        }

        case 5:

            return 0;

        default:

        {

            cout << "Invalid choice\n";

            break;

        }

        }

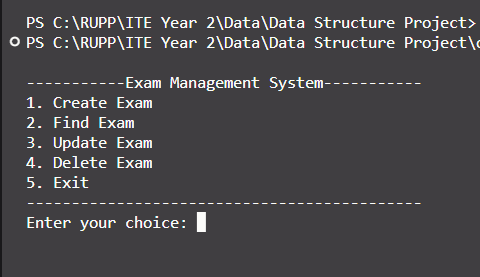
    }

    return 0;

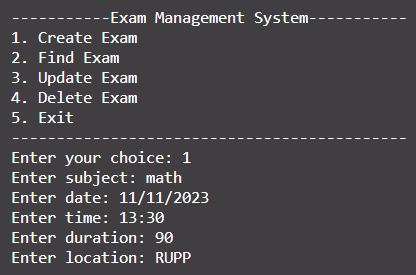
}

# 2.5 Testing

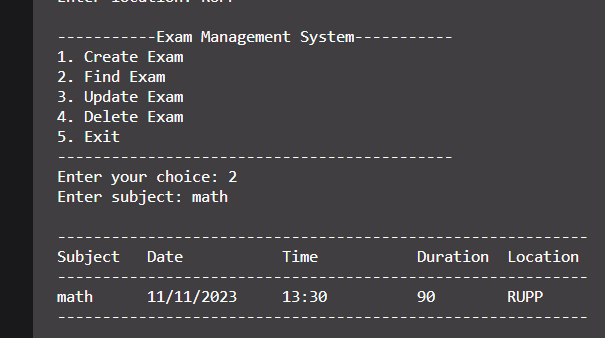
1. **UI Interface**



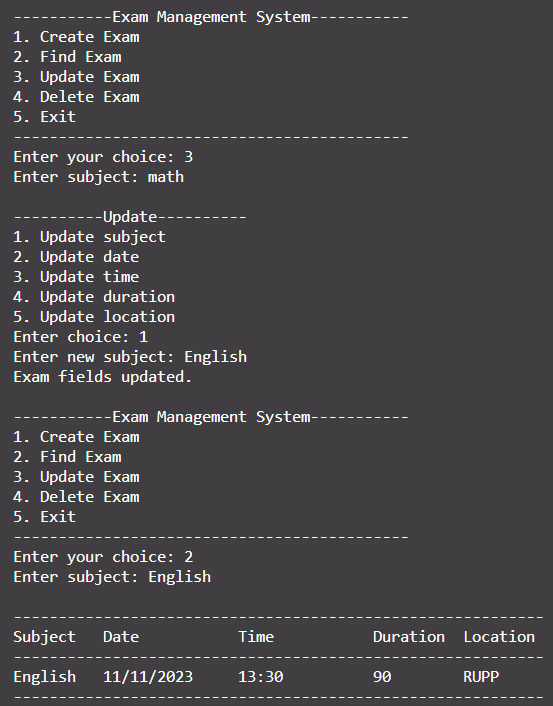
1. **Create Exam**



1. **Find Exam**



1. **Update Exam**



1. **Remove Exam**

