Ramit Das(Reg:999)

Anmol Mishra(Reg:975)

GRADE CARD GENERATION FROM FILE DATA

INTRODUCTION:

The "Grade Card Generation from File Data" project is aimed at developing a program that automates the process of generating grade cards for students based on data stored in a file. The project leverages the power of data structures and file handling in C programming language to efficiently manage and process student information.

OBJECTIVE:

The primary objective of this project is to provide a convenient and organized way to generate grade cards, enabling easy access to student data, grades, CGPA, and other relevant information. By reading the data from a file and utilizing a binary search tree (BST) data structure, the program allows efficient storage, retrieval, and manipulation of student records.

The key features of the Grade Card Generation program include:

Data Storage and Organization: The program uses a binary search tree to store and organize student information. Each node of the tree represents a student, and the tree is constructed based on the students' CGPA, enabling efficient searching and ranking operations.

File Data Processing: The program reads the student data from a file (grades.csv) and extracts relevant information such as name, registration number, and grades in 5 subjects. The data is then processed and stored in the binary search tree, ensuring accurate representation and easy retrieval of student records.

Grade Card Generation: The program provides functionalities to generate grade cards for students. It can display the grade card of all students in the class, search for a specific student's grade card by name, search for grade cards based on CGPA, rank, or grade achieved, and generate a summary report including the average CGPA of the class.

User-Friendly Interface: The program offers a menu-driven interface that allows users to interact with the system effortlessly. Users can select various

options from the menu to access different functionalities of the program, making it intuitive and easy to navigate.

ADVANTAGES:

Efficient Data Storage and Retrieval: The code utilizes a binary search tree (BST) data structure to store and organize student information. This allows for efficient searching, insertion, and retrieval of student records based on their CGPA, providing quick access to specific student grade cards.

```
void insertNode(Node **root, Student student)
{
    if (*root == NULL)
    {
        Node *newNode = (Node *)malloc(sizeof(Node));
        newNode->student = student;
        newNode->left = NULL;
        newNode->right = NULL;
        *root = newNode;
    }
    else
    {
        if (student.cgpa < (*root)->student.cgpa)
        {
            insertNode(&((*root)->left), student);
        }
        else
        {
            insertNode(&((*root)->right), student);
        }
    }
}
```

Automated Grade Card Generation: The code automates the process of generating grade cards by reading student data from a file and populating the BST with the relevant information. This eliminates the need for manual calculations and paperwork, saving time and effort.

Below is the Excel file from which the data is taken input

A	B	C	D	E	F	G	H
Ramit	999	10	5	8	2	3	
Ankit	974	9	8	8	2	3	
Aman	973	9	5	4	2	3	
Anmol	975	10	10	10	10	10	
Harsh	989	9	9	5	8	3	
Arin	980	7	2	9	1	6	
Toufik	1016	7	2	9	1	6	
Aman jaisv	972	10	10	10	9	8	

Flexible Search Functionality: The code provides various search options, allowing users to search for grade cards based on different criteria such as student name, CGPA, rank, and grade achieved. This flexibility enables users to retrieve specific information quickly and accurately.

```
2)SEARCH STUDENT'S GRADE CARD BY NAME
3)SEARCH STUDENT'S GRADE CARD BY CGPA
4)SEARCH STUDENT'S GRADE CARD BY RANK
5)SEARCH STUDENTS BY GRADE
```

Ranking of Students: The code includes a ranking mechanism that assigns a rank to each student based on their CGPA. This ranking information is stored in the BST and can be easily accessed. It provides a clear understanding of the relative performance of students within the class.

```
Name: Aman, Registration Number: 973, CGPA: 4.60 , Rank: 8
Name: Arin, Registration Number: 980, CGPA: 5.00 , Rank: 7
Name: Toufik, Registration Number: 1016, CGPA: 5.00 , Rank: 6
Name: Ramit, Registration Number: 999, CGPA: 5.60 , Rank: 5
Name: Ankit, Registration Number: 974, CGPA: 6.00 , Rank: 4
Name: Harsh, Registration Number: 989, CGPA: 6.80 , Rank: 3
Name: Aman jaiswal, Registration Number: 972, CGPA: 9.40 , Rank: 2
Name: Anmol, Registration Number: 975, CGPA: 10.00 , Rank: 1
```

Summary Report Generation: The code can generate a summary report that includes the average CGPA of the entire class. This feature provides valuable insights into the overall performance of the class, allowing teachers and administrators to evaluate the academic progress of students collectively.

```
.....
..... WELCOME TO GRADE CARD GENERATION .....
1)DISPLAY ALL STUDENTS GRADE CARD
2)SEARCH STUDENT'S GRADE CARD BY NAME
3)SEARCH STUDENT'S GRADE CARD BY CGPA
4)SEARCH STUDENT'S GRADE CARD BY RANK
5)SEARCH STUDENTS BY GRADE
6)DISPLAY CLASS AVG CGPA SCORE
7)EXIT
Enter choice: 6


Class Average CGPA: 6.55
```

User-Friendly Interface: The code offers a menu-driven interface that makes it user-friendly and easy to navigate. Users can choose from different options in the menu to perform specific operations, enhancing the overall usability of the program.

```
.....
..... WELCOME TO GRADE CARD GENERATION .....
1)DISPLAY ALL STUDENTS GRADE CARD
2)SEARCH STUDENT'S GRADE CARD BY NAME
3)SEARCH STUDENT'S GRADE CARD BY CGPA
4)SEARCH STUDENT'S GRADE CARD BY RANK
5)SEARCH STUDENTS BY GRADE
6)DISPLAY CLASS AVG CGPA SCORE
7)EXIT
```

Scalability: The code can handle a large number of student records since it uses a BST for efficient data management. This scalability ensures that the program can handle data from a substantial student population without compromising performance.

Code Modularity: The code demonstrates modularity by using separate functions for different operations such as inserting a node, displaying the grade card, searching by name, CGPA, rank, and grade, and generating summary reports. This modular approach enhances code readability, maintainability, and reusability.

```
void insertNode(Node **root, Student student);
void Ranking(Node *root, int *c);
void displayTree(Node *root);
void searchStudent(Node *root, char *name);
void searchCgpa(Node *root, float grade);
void searchRank(Node *root, int Rank);
void searchgrade(Node *root, int grade);
void generateReportSummary(Node *root, int c);
```

DISADVANTAGES:

Fixed Number of Subjects: The code assumes a fixed number of subjects (5) for each student. If the project requirements change in the future to accommodate a different number of subjects, the code would need to be modified accordingly. This lack of flexibility may require additional effort and maintenance.

Inefficient Memory Usage: The code uses a binary search tree (BST) to store student records. While BSTs offer efficient searching, insertion, and retrieval, they can consume more memory compared to other data structures. In situations with a large number of student records, this memory usage may become a concern.

Limited Sorting Options: The code only sorts the student records based on CGPA. It does not provide options to sort based on other criteria, such as name or registration number. This limitation may restrict certain sorting requirements or preferences of users.