Programmer's Manual

1.Project Overview:

Introduction to the Project

The objective of this project was to develop an application that manages the accounts of a student bar at a secondary school. The application maintains updated information on all students, including their name, date of birth, year, class, number, and balance. Additionally, it tracks their expenses over a calendar year (amount, description, date) using linked lists. The application is interactive and allows, at a minimum, the following operations:

- Add a new student.
- Remove an existing student.
- List all students in alfabéticas order.
- List students with a balance below a certain threshold.
- Display detailed information about a specific student.
- Record an expense for a specific student.
- Top up a student's account with a specified amount.

Objectives and Goals

The primary goal of this project was to create an efficient and reliable system that ensures accurate tracking of student balances and expenses. The system utilizes appropriate data structures to optimize memory usage, particularly by employing auxiliary data structures with pointers to actual records instead of structures containing the records themselves.

Target Audience

This application is intended for administrators and staff at the student bar, who use it to manage student accounts, track financial transactions, and generate reports.

General Program Structure

The program is designed with a modular structure, separating concerns into different files. Core data structures are defined in an autonomous source file, and the program reads and writes data to text files, ensuring that all relevant information is preserved between sessions. Security features are implemented to protect the integrity of the data.

2.Project Architecture:

File and Directory Organization

The project is organized as follows:

- **main.c:** Contains the main function (main()) that coordinates the execution of the program. It initializes the environment, calls functions from other modules, and manages the overall flow of the system.
- **Interface.c:** Manages the user interface, displaying menus, collecting inputs, and presenting information on the screen. This module separates the interface logic from the business logic.
- **registration.h:** Defines the main data structures (such as Student) and declares the functions for handling student data. This file is crucial for organizing and managing student information.

- **RegisteredStudents.txt:** Stores student data, including name, date of birth, year, class, number, and balance. This file is loaded when the program starts and updated as needed.
- **expenses.txt:** Contains records of student expenses, including amount, description, and date. This file is used to maintain the financial history of the students.

Main Data Structures

The main data structures used in the project are:

- **Student Structure**: A structure that stores information about each student, including name, date of birth, year, class, number, and balance. It is the foundation for the linked lists used to manage students.
- **Linked Lists:** Used to dynamically store students and their expenses. This structure allows efficient insertion, removal, and navigation among records.

3.Description of Core Data Structures:

The project utilizes several core data structures to manage student information and their expenses. The main data structures are:

Date

- **Usage**: Represents a date with day, month, and year.
- **Purpose**: Used in various parts of the system to store important dates, such as student birthdates and the dates of expenses incurred.

- **Definition:**

```
typedef struct Date{ //Date structure for date
    int day;
    int month;
    int year;
}Date;
```

ExpenseNode

- **Usage**: Stores information about a single expense made by a student.
- **Purpose**: Contains the expense amount, a description, and the date when the expense occurred.
- **Definition**:

```
typedef struct ExpenseNode{ //Date structure for student expenses
    Date day;
    double amount;
    char description[50];

}ExpenseNode;
```

Expense

- **Usage**: Forms a linked list of expenses for a specific student.
- **Purpose**: Allows for dynamic management of student expenses, where each node points to the next expense.
- **Definition**:

```
typedef struct Expense{ //Date str
    ExpenseNode StudentExpense;
    struct Expense *next;
}Expense;
```

Registration

- **Usage**: Stores detailed information about a student, including name, birthdate, class, year, registration number, balance, and a pointer to the list of expenses.
- **Purpose**: Centralizes all relevant information about a student in a single structure, facilitating data manipulation and access.
- **Definition**:

```
typedef struct Registration{
    char name[50];
    Date birthday;
    char class;
    char year[20];
    char registration[20];
    char balance[50];
    Expense *ptr;
}Registration;
```

Students

- **Usage**: Forms a linked list of students, where each node contains a student's information and a pointer to the next student.
- **Purpose**: Allows for the management of multiple students, enabling the addition, removal, or navigation through the student list.
- **Definition**:

```
typedef struct Students{ //
    Registration student;
    struct Students *next;
}Students;
```

4.List of Created Functions:

- **void print_expense(ExpenseList list):** Prints the list of expenses.
- **ExpenseList create_expense():** Creates a new list of expenses.
- **void search_expense(ExpenseList list, ExpenseNode key, ExpenseList *previous, ExpenseList *current):** Searches for an expense in the list.
- **void insert_expense(ExpenseList list, ExpenseNode a1):** Inserts a new expense into the list.
- **void load_expense(aList list):** Loads expenses into the list from a file.
- **int validate_balance(Registration *key):** Validates the balance of a student.
- **aList create():** Creates a new list of students.
- **void search_balance(lBalances list, Registration key, lBalances *previous, lBalances *current):** Searches for a balance in the list.
- **lBalances create_balance():** Creates a new balance list.
- **void insert_balance(Registration *key):** Inserts a balance record for a student.
- **void print_balance_list(lBalances list):** Prints the list of balances.
- **void insert_balance_below(lBalances list, Registration a1):** Inserts a balance below a certain threshold.
- **void sort_balance_below(lBalances list, aList list_students):** Sorts the list of balances below a threshold.
- **void insert(aList list, Registration a1):** Inserts a new student into the list.
- **int isEmpty(aList list):** Checks if the list of students is empty.
- **int isEmpty_balances(lBalances list):** Checks if the list of balances is empty.

- **int isEmpty_expenses(ExpenseList list**): Checks if the list of expenses is empty.
- **ExpenseList destroy_expense(ExpenseList list):** Destroys the list of expenses.
- **aList destroy(aList list):** Destroys the list of students.
- **lBalances destroy_balances(lBalances list):** Destroys the list of balances.
- **void search(aList list, Registration key, aList *previous, aList *current):** Searches for a student in the list.
- **void print_student(aList list):** Prints information about a student.
- **int menu():** Displays the main menu and returns the selected option.
- **void find_student(aList list, Registration key, aList *previous, aList *current):** Finds a student in the list.
- **void delete(aList list, Registration key):** Deletes a student from the list.
- **aList present(aList list, Registration key): Presents** a student based on the given key.
- **int validate_year(Registration *student):** Validates the year of a student.
- **void insert_year(Registration *student):** Inserts the year for a student.
- **int validate_name(Registration *key):** Validates the name of a student.
- **int validate_surname(char str[]):** Validates the surname.
- **void insert_name(Registration *key):** Inserts the name of a student.
- **int validate_registration(Registration *student):** Validates the registration number of a student.
- **void insert_registration(Registration *key):** Inserts the registration number of a student.
- **void load(aList list):** Loads the student list from a file.
- **int validate_Date(Date birthday):** Validates the date of birth.

- **int validate_Date_student(char key[]):** Validates the student's birth date.
- **void insert_date(Registration *key):** Inserts the birth date of a student.
- **void insert_date_expense(ExpenseNode *key**): Inserts the date of an expense.
- **int validate_description(ExpenseNode *key):** Validates the description of an expense.
- **void insert_description(ExpenseNode *key):** Inserts a description for an expense.
- **int validate_amount_expense(char aux**[]): Validates the amount of an expense.
- **void insert_amount_expense(ExpenseNode *key):** Inserts the amount of an expense.
- **void write_student(aList list**): Writes the student data to a file.
- **void write_expense(aList list):** Writes the expense data to a file.
- **void iterate_Date(aList list, Registration *student, const char linha[]):** Iterates over dates in the student list.
- **void iterate_name(aList list, Registration *student, const char linha[]):** Iterates over names in the student list.
- **void iterate_class(aList list, Registration *student, const char linha[]):** Iterates over classes in the student list.
- **void iterate_year(aList list, Registration *student, const char linha[]):** Iterates over years in the student list.
- **void iterate_registration(aList list, Registration *student, const char linha[]):** Iterates over registrations in the student list.
- **void iterate_balance(aList list, Registration *student, const char linha[]):** Iterates over balances in the student list.
- **void iterate_registration_expense(Registration *student, const char linha[]):** Iterates over expense registrations for a student.

- **void iterate_Date_expense(ExpenseNode *student, const char linha[]):** Iterates over expense dates for a student.
- **void iterate_description_expense(ExpenseNode *student, const char linha[]):** Iterates over expense descriptions for a student.
- **void iterate_amount_expense(ExpenseNode *student, const char linha[]):** Iterates over expense amounts for a student.
- **void traverse_expenses(aList list):** Traverses through the list of expenses.
- **void traverse_students(aList list): Traverses** through the list of students.
- **int validate_class(char str[]):** Validates the class of a student.
- **void insert_class(Registration *key):** Inserts the class of a student.
- **void sort_class(aList list):** Sorts the students by class.