RIPHAH INTERNATIONAL UNIVERSITY ISLAMABAD

Project-Title: Daily expenditures tracking application.

Submitted to: Sir Ahsan Abbasi

Sr No.	**Student Name**	**Sap Id**
1	Naqqash haider	1980
2	Mussab bin shahid	2024
4	Ibrahim kan	1981

Class: BSCS

Semester & Section: 3rd semester section(1)

Faculty of computing (FC)

Introduction:

Spending Tracker is essentially an application which keeps record of the day by day Income and Expenses of the client. It causes the client to keeps up his every day financial plan and sets aside cash by watching out for his spending.

Features:

- Totally secure Sign up with putting away client certifications in Binary File Format which is confused by human. No duplication in Usernames.
- Text File Report Generation which client can use for any reason like printing or putting away as reinforcement.
- Summary Graphs and Percentage Analysis to each cost class which makes it simple for client to decide its costs.
- Separate record to store exchanges for every client which can't be gotten to without secret key.
- Add Incomes/Expenses with data, for example, Amount, Date, Payment Method utilized and Category.

- Displays Final Balance in Account after every exchange is made.
- Displays Transactions List as indicated by client's decision as most recent 7 Days, a month ago or Yearly.
- Best and most significant component is that it very well may be utilized on a Public PC where every client has its own USERNAME and PASSWORD and has its own different record which can't be gotten to without secret key.

Project Code

USER HEADERFILE

```
#pragma once
  #include <iostream>
#include<string>
  using namespace std;
class user
  {
public:
        string username, password; //to store login credentials
  public:
          user(void);
            void getData(int); //to get login credentials rom user
            string getUsername(); //get username function
            string getPassword(); //get password function


};
```

USER.CPP

```
#include "user.h"
  #include<string>
  #include"transaction.h"
  #include <iostream>
#include<fstream>
  using namespace std;

  user::user(void)
  {
          }

  void user::getData(int type)
  {
          if (type == 1) //log in
                  {
                  bool checkpoint; //flag to check credentials accuracy
                  string temp_username; //temporary username variable
                  string temp_password; //temporary password variable
                do //loop to get input again and again untill correct username is entered
                          {
                          checkpoint = false; //flag set to false

                                  cout << endl;
                          cout << "Enter Username:" << endl;
                          cin >> temp_username; //username input
```

```
                    cout << endl << "Enter Password:" << endl;

                    cin >> temp_password; //password input


                        ifstream read_file("credentials.txt", ios::binary); //read mode file open w



                                    read_file.read(reinterpret_cast<char*>(this), sizeof(*this));


                            if ((temp_username == this->username) && (temp_password == this
->password)) //it checks either username and password are correct or not

 {

                        checkpoint = true; //if username and password are entered correctly

                        }



                        if (checkpoint == false) //if username or password is incorrectly typed

                        {

                        cout << "Incorrect USERNAME or PASSWORD!!! TRY AGAIN" << endl << endl;

                        }

                    read_file.close();

              }

               while (checkpoint == false); //let it get input from user unless correct username

                    if (checkpoint == true) //condition when username and password are correctly typed


               {

                    this->username = temp_username; //stores username

                    this->password = temp_password; //stores password

                    }


          }

           if (type == 2) //sign up

              {
                    bool checkpoint; //flag to check credentials uniqueness
                    string temp_username; //stores username temporarily
                    string temp_password; //stores password temporarily

                        do //asks user for username and password untill he enters unique username(i.e.

                        {
                        checkpoint = true; //set flag to false

                            cout << endl;
                       cout << "Enter Username:" << endl;
                        cin >> temp_username; //username input

                            cout << endl << "Enter Password:" << endl;
                       cin >> temp_password; //password input

                           ifstream read_file("credentials.txt", ios::binary); //file open read mode t o check
either entered username is already registered or not

                           read_file.read(reinterpret_cast<char*>(this), sizeof(*this)); //reads

                              if (temp_username == this -> username) //if entered username is registered
 {
                               cout << "USERNAME Already Found!!! TRY AGAIN" << endl << endl;
                               checkpoint = false; //set flag to true

                                }

                        read_file.close(); //file close
                        }
                    while (checkpoint == false); //continues loop till checkpoint is false (i.e. use
                       if (checkpoint == true) //if duplicate username not found
                        {
                        this->username = temp_username; //stores username
                        this->password = temp_password; //stores password
                        transaction t; //transaction class object
                        t.setZero(this->username, this ->password);
                        ofstream write_file("credentials.txt", ios::binary | ios::app);
                           write_file.write(reinterpret_cast<char*>(this), sizeof(*this));
                                           write_file.close(); //file close
                       }

               }

           }

string user::getUsername() //get username function
               {
              return(username);
               }

               string user::getPassword() //get password function
               {
               return(password);

              }
```

DATE HEADERFILE

```
#pragma once
```

```
#include<iostream>
 using namespace std;
 class date
 {
 public:
        int day, month, year; //date variables
 public:
        date(void);
        void getInput(); //to get input date
        void displayDate(); //to display date
        };
```

DATE.CPP

```
#include"date.h"
 date::date(void)
 {
        }

 void date::getInput()
 {
        cout << endl << "Enter Date:-" << endl;
        try //exception handling//
        {
               cout << "Month:" << endl;
               cin >> month;
               if (month > 12)
                      throw;


        }
        catch (...)
        {


        }
        cout << "Day:" << endl;
        cin >> day;
        while (cin.fail())
        {
            cout << "error"<<"\n";
                cin.clear();
                cin.ignore();
                cin >> day;
        }

        cout << "Year:" << endl;
        cin >> year;
        while (cin.fail())
        {
            cout << "error please enter integer value" << "\n";
            cin.clear();
            cin.ignore();
            cin >> year;


        }


        }

void date::displayDate()
{
     cout << day << "/" << month << "/" << year;
}
```

TRANSACTIONNODE HEADERFILE

```
#pragma once
#include"date.h"
#include<string>
class transactionNode
 {
 public:
        int amount; //transaction amount
        int balance; //final balance
        string category; //type of transaction
        string paymentType; //paymentType where transaction made
        date d; //to store date of transaction
        transactionNode *next;
 public:
        transactionNode(void);
        };
```

TRANSACTION NODE.CPP

```
#include "transactionNode.h"
 #include<string>
 #include "date.h"

 transactionNode::transactionNode(void)
{
      }
```

TRANSACTION HEADER FILE

```
#pragma once
  #include "date.h"
  #include "transactionNode.h"
  #include <iostream>
#include<string>
#include<fstream>
using namespace std;

  class transaction
  {
    public:
          transactionNode * head;
  public:
          transaction(void); //constructor
          void setZero(string username, string password); //set all values of new user to zero

                void getAmount(string, string, int); //function to get input about transaction data

                  void displayTransaction(string); //to display transactions with respect to user's

                  void finalBalance(string); //to display final balance in account
          transactionNode * fileToList(string); //converts file data to linked list
          void generateReport(string); //to generate textual report of transactions
          void summary(string);

  };
```

TRANSACTION.CPP

```
 #include "transaction.h"
 #include<conio.h>
 #include<iomanip>
 #include <stdio.h>
 #include <stdlib.h>
 #include <windows.h>
 #include <iostream>
 using namespace std;

 transaction::transaction(void) //constructor
 {
         head = new transactionNode;
         }

void transaction::setZero(string username, string password)
{
         head->amount = 0; //set to zero
         head->balance = 0; //set to zero
         head->d.day = head->d.month = head->d.year = 0;
         head->next = NULL;
       ofstream username_write(username.c_str(), ios::binary);
       username_write.write(reinterpret_cast<char*>(head), sizeof(*head));
       ofstream password_write(password.c_str(), ios::binary);
       password_write.write(reinterpret_cast<char*>(head), sizeof(*head));
       password_write.close();
       }

void transaction::getAmount(string username, string password, int type) //function to g

 {
         if (type == 1) //income
                 {
                 transactionNode * ptr = new transactionNode; //node created to add new transaction
```

```
            int temp_amount; //temporary amount storing variable

            cout << "Enter Income Amount:" << endl;
        cin >> temp_amount; //income amount input

            ifstream username_read(username.c_str(), ios::binary);

            username_read.read(reinterpret_cast<char*>(ptr), sizeof(*ptr));

            ptr->balance = ptr -> balance + temp_amount; //adds amount to final balance
(i.e.updating final balance)
        username_read.close();

            ptr->amount = temp_amount; //sets amount

            cout << endl << "Enter Income Category" << endl;
              cin.ignore();
            getline(cin, ptr->category);



        cout << endl << "Enter Income Payment Method:" << endl;
        getline(cin, ptr->paymentType);





           ptr->d.getInput(); //date input

           ofstream username_write(username.c_str(), ios::binary); //file write mode open

            username_write.write(reinterpret_cast<char*>(ptr), sizeof(*ptr));
```

```
                    username_write.close(); //file close


                    ofstream password_write(password.c_str(), ios::binary | ios::app);
                            password_write.write(reinterpret_cast<char*>(ptr), sizeof(*ptr));


                   password_write.close(); //file close
               }
         if (type == 2) //expense
                {
          transactionNode * ptr = new transactionNode; //node created to add new transaction
         int temp_amount; //temporary amount storing variable


                    cout << "Enter Expense Amount:" << endl;
                cin >> temp_amount; //expense amount input


                    temp_amount = (temp_amount * (-1)); //set debit amount to negative





                     ifstream username_read(username.c_str(), ios::binary);
                      username_read.read(reinterpret_cast<char*>(ptr), sizeof(*ptr)); //reads object



                    ptr->balance = ptr -> balance + temp_amount; //adds amount to final balance (i.e.
updates final balance)
                     username_read.close(); //file close


                         ptr->amount = temp_amount; //amount set


                        cout << endl << "Enter Expense Category" << endl;
                    cin.ignore();
                        getline(cin, ptr->category);//gets transaction type



                         cout << endl << "Enter Expense Payment Method:" << endl;
                         getline(cin, ptr->paymentType);//gets transaction


                        ptr->d.getInput(); //gets date input

                         ofstream username_write(username.c_str(), ios::binary);
                        username_write.write(reinterpret_cast<char*>(ptr), sizeof(*ptr)); //writes obj

                        username_write.close(); //file close


                        ofstream password_write(password.c_str(), ios::binary | ios::app);
                     password_write.write(reinterpret_cast<char*>(ptr), sizeof(*ptr));

                         password_write.close(); //file close

         }

}

 void transaction::finalBalance(string filename) //to display final balance in ac

 {
          transactionNode * ptr = new transactionNode;

                ifstream file_read(filename.c_str(), ios::binary);

          file_read.read(reinterpret_cast<char*>(ptr), sizeof(*ptr));
```

```
                cout << endl << "Account Balance: " << ptr->balance << endl;
                        file_read.close(); //file close
        }

transactionNode* transaction::fileToList(string filename) //converts file data t o linked list
        {
          transactionNode * ptr = new transactionNode;
          transactionNode * temp = new transactionNode;
      transactionNode * list = new transactionNode;
      ifstream file_read(filename.c_str(), ios::binary);



        file_read.read(reinterpret_cast<char*>(ptr), sizeof(*ptr));


              temp = ptr;
        list = temp;


                while (!file_read.eof())
                {
                file_read.read(reinterpret_cast<char*>(ptr), sizeof(*ptr));


                temp -> next = ptr; //connecting each file read data (node) to linked list
                ptr = new transactionNode;
                temp = temp->next;
        }
        temp->next = NULL;


                return list;


}
void transaction::displayTransaction(string filename) //to display transactions
```

```
        {
        int duration;

                cout << "Select Time Period" << endl;
        cout << "1. Last 7 Days." << endl;
        cout << "2. Last Month." << endl;
        cout << "3. Last 6 Months." << endl;
        cout << "4. All." << endl;
        cin >> duration;


                system("CLS");
        cout << endl <<
"================================================================================ = ";
                HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(m_hConsole,
                BACKGROUND_RED |
                BACKGROUND_GREEN |
                BACKGROUND_BLUE);
        cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E R ";
        SetConsoleTextAttribute(m_hConsole,
                FOREGROUND_RED |
                FOREGROUND_GREEN |
                FOREGROUND_BLUE);
        cout <<
"================================================================================ =
"<<endl;
              transactionNode * temp = fileToList(filename);

                cout <<
"================================================================================ = ";
                cout << " " << setw(20) << left << "TRANSACTION AMOUNT";
        cout << setw(16) << left << "CATEGORY";
```

```
            cout << setw(14) << left << "paymentType";

            cout << setw(18) << left << "FINAL BALANCE";

            cout << "DATE" << endl;

            cout <<
"=================================================================================== =
"<<endl;


                    if (duration == 1) //1 is specified for last 7 days(week)

                    {

                    int sevenDay = (temp->d.day) - 7;

                    int thisDay = temp->d.day; //last transaction's day

                    int thisMonth = temp->d.month; //last transaction's month

                    int thisYear = temp->d.year; //last transaction's year


                            temp = fileToList(filename); //converts file data to linked list


                            while (temp->next != NULL)

                            {

                            if (temp->d.day >= sevenDay && temp->d.day <= thisDay && temp -> d.month ==
thisMonth && temp -> d.year == thisYear) //condition for last 7 days transactions

{

                                    cout << " " << setw(20) << left << temp->amount << setw(16) << left << temp
-> category << setw(14) << left << temp->paymentType << setw(18) << left << temp->balance;

                                    temp->d.displayDate();

                                    cout << endl;

                                    }

                            else

                                    {

                                    }

                            temp = temp->next;

                            }

                    }

              else if (duration == 2) //2 is specified for last 30 days(month)

                    {
```

```
                    int thirtyDay = (temp->d.day) - 30;

                    int thisDay = temp->d.day; //last transaction's day

                    int thisMonth = temp->d.month;

                    int thisYear = temp->d.year;


                        temp = fileToList(filename); //converts file data to linked list


                        while (temp->next != NULL)

                        {

                        if (temp->d.day >= thirtyDay && temp->d.day <= thisDay && temp -> d.month ==
thisMonth && temp -> d.year == thisYear) //condition for last 30 days transactions

  {

                          cout << " " << setw(20) << left << temp->amount << setw(16) << left << temp ->
category << setw(14) << left << temp->paymentType << setw(18) << left << temp->balance;

                            temp->d.displayDate();

                            cout << endl;

                            }

                        else

                            {

                            }

                        temp = temp->next;

                        }

                    }

            else if (duration == 3) //3 is specified for last 6 months

                    {

                   int month = (temp->d.month) - 6;

                    //int thisDay = temp->d.day; //last transaction's day

                        int thisMonth = temp->d.month;

                    int thisYear = temp->d.year;

                    temp = fileToList(filename);


                         while (temp->next != NULL)

                         {
```

```
                        if (temp->d.month >= month && temp->d.month <= thisMonth && temp -> d.year
== thisYear) //condition for last year transactions
 {
                              cout << " " << setw(20) << left << temp->amount << setw(16) << left <<
temp -> category << setw(14) << left << temp->paymentType << setw(18) << left << temp->balance;
                              temp->d.displayDate();
                              cout << endl;
                              }
                         else
                              {
                              }
                         temp = temp->next;
                         }

                    }
                      else //display all transactions
                         {
                         while (temp->next != NULL)
                              {
                              cout << " " << setw(20) << left << temp->amount << setw(16) << left <<
temp -> category << setw(14) << left << temp->paymentType << setw(18) << left << temp->balance;
                              temp->d.displayDate();
                              cout << endl;
                              temp = temp->next;
                              }
                         }
                      cout <<
"=============================================================================== =
"<<endl;

 }

          void transaction::generateReport(string filename) //to generate textual report

          {
```

```
        transactionNode * temp = fileToList(filename); //converts file data to linked



           ofstream report_write("Transactions_Report.txt");


            report_write <<
"============================================================================================== = ";
            report_write << endl;
        report_write << " " << setw(20) << left << "TRANSACTION AMOUNT";
        report_write << setw(16) << left << "CATEGORY";
        report_write << setw(14) << left << "paymentType";
        report_write << setw(18) << left << "FINAL BALANCE";
        report_write << "DATE";
        report_write << endl;
        report_write <<
"============================================================================================== = ";
           report_write << endl;


            while (temp->next != NULL)
            {
            report_write << " " << setw(20) << left << temp -> amount << setw(16) << left <<
temp->category << setw(14) << left << temp -> paymentType << setw(18) << left << temp->balance;
            report_write << temp->d.day << "/" << temp->d.month << "/" << temp -> d.year;
             report_write << endl;
             temp = temp->next;


            }


                  report_write << endl;
             report_write <<
"============================================================================================== = ";


                 cout << endl;
             cout << "DONE!!! Report Saved" << endl;
```

```
                    report_write.close(); //file close

          }

                 void transaction::summary(string filename)
                 {

cout<<"==============================================================================
=== = "<<endl;

                        cout<<" S U M M A R Y"<<endl;

cout<<"======================================================================
=========== = "<<endl;
                 cout << endl <<
"=============================================================================== = ";
                     HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                 SetConsoleTextAttribute(m_hConsole,
                         BACKGROUND_RED |
                         BACKGROUND_GREEN |
                         BACKGROUND_BLUE);
                 cout << " S U M M A R Y ";
                 SetConsoleTextAttribute(m_hConsole,
                     FOREGROUND_RED |
                       FOREGROUND_GREEN |
                         FOREGROUND_BLUE);
                 cout <<
"============================================================================== =
"<<endl;

                       cout << setw(10) << left << "CATEGORY" << "";
                cout << setw(3) << left << "%AGE";
                cout << left << " PERCENTAGE GRAPH" << endl;
                 cout << "------------------------------------------------------------------------------- - "<<endl;

                         transactionNode * temp1 = fileToList(filename);
```

```
        int incomeSum = 0; //to store sum of incomes
    int expenseSum = 0; //to store sum of expenses

        while (temp1->next != NULL)
        {
        if (temp1->amount >= 0)
            {
            incomeSum = (temp1 -> amount + incomeSum); //to calculate total income of
user
            }
        else
        {
            int amount = (temp1->amount * (-1)); //to make expense positive
             expenseSum = (amount + expenseSum); //to calculate total income of


        }
         temp1 = temp1->next;

        }

             temp1 = fileToList(filename); //renewing head again to start of linked list


             while (temp1->next != NULL) //loop to make sum of transactions of similar
category and then determine

             {
                 int expenseCategory = 0; //to calculate sum of the transations of same
c


                 transactionNode* temp2 = fileToList(filename); //converts file data to l
```

```
                                        while (temp2->next != NULL) //compares each item of list to the
remaining list

                                        {

                                                if (temp1->category == temp2->category && temp2->amount < 0)
//if category is same and expense amount only(amount>0)

                                                {

                                                        expenseCategory = (expenseCategory + (-1 *
temp2->amount));

                                                }

                                                temp2 = temp2->next;

                                        }


                                        int percentage = (((expenseCategory * 1.0) / (expenseSum * 1.0)) *
65.0);

                                        cout << setw(10) << left << temp1->category << ":";

                                        cout << setw(3) << left << percentage << "% ";

                                        while (percentage)

                                        {

                                                percentage--;

                                                cout << "]";


                                        }

                                        cout << endl;

                                        temp1 = temp1->next;


                                }

                        cout << "--------------------------------------------------------------------------- - "<<endl;


                                cout << "Total Income: " << incomeSum << endl;

                        cout << "Total Expense: " << expenseSum << endl;


                }
```

FEEDEPARTEMENT HEADERFILE

```
#include<iostream>
#include<fstream>
#include<string>
#define SIZE 10
using namespace std;
class forms
{
private:
        unsigned front = -1, rear = -1, top = -1;
        int serialno[SIZE], serial, admissionfee;
        int noofperson[SIZE], no;
        int admission[SIZE];
        int universityfee[SIZE], registrationfee;
        int universityidcard[SIZE], idcard, tutionfee[SIZE], tution;
        int exam[SIZE], examfee;
        long semester[SIZE], semesterenrolfee, total[SIZE], totalfee;
        string detail;
        string name[SIZE];
        int temp, lower, j;
        int n;
        int *pers;
public:
        void personenqueue();
        void persondequeue();
        void displayform();
        bool isemptyqueue();
```

```
	void stackformsadd();
	bool isemptystack();
	void stackformsremove();
	void displayformstack();
	void selectionsort();
	void bubblesort();
	void quicksort(int* b, int s, int e);
	int partitioning(int* a, int start, int end);

};
```

FEEDEPARTEMENT.CPP

```
#include"feedepartement.h"
void forms::personenqueue() //Queue implementation//
{
	cout << "Enter persons to add in queue " << "\n";
	cin >> no;
	cout << "Enter serial number of your fomr" << "\n";
	cin >> serial;
	if (cin.fail())
	{
		cout << "please enter a integer" << "\n";
		cin.clear();
	}
	if (rear == SIZE - 1)
	{
		cout << "Only 10 persons allowed in queue" << "\n";
	}
	else
	{
		if (front == -1)
			front = 0;
		rear++;
		noofperson[rear] = no;
		serialno[rear] = serial;

	}



}

bool forms::isemptyqueue()
{
	if (front == -1 && rear == -1)
		return true;
	else
		return false;
}
void forms::persondequeue()
{
	if (isemptyqueue())

		cout << "No one is in the queue " << "\n";

	else
		if (front == rear)
			front = rear = -1;
		else
			front++;

}
void forms::displayform()
{
        if (isemptyqueue())
        {
                cout << "No one is in the queue " << "\n";
        }
        else
        {
                cout << "No of Persons waiting in queue to submit their challan form" << "\n\n";
                for (int i = front; i <= rear; i++)
                {



                        cout << noofperson[i] << "\n\n";
                        cout << "serial no :" << " " << serialno[i] << "\n\n";
                }

        }

}
void forms::selectionsort() //selectionsorting//
{
        cout << "enter size of array" << "\n\n";
        cin >> n;
        pers = new int[n];//dynamic array
        cout << "enter elements to add in array" << "\n\n";
        for (int i = 0; i < n; i++)
        {
                cin >> pers[i];
        }

        for (int i = 0; i < n - 1; i++)
        {
                lower = i;
                for (j = i + 1; j < n; j++)
                {
                        if(pers[j]<pers[i])
                        lower = j;
                }
                temp = pers[i];
                pers[i] = pers[lower];
                pers[lower] = temp;
        }
        cout << "array after selection sort" << "\n";

        for (int i = 0; i < n; i++)
        {
                cout << pers[i]<<" "<<"\n";
        }

}

void forms::bubblesort()
{
        int pass, temp, flag;
        cout << "enter size of array" << "\n\n";
        cin >> n;
        cout << "enter elements to add in array" << "\n\n";
        for (int i = 0; i < n; i++)
        {
                cin >> pers[i];
        }
        for (pass = 1; pass < n; pass++)
        {
                flag = 0;
                for (int i = 0; i < n - pass; i++)
                {
                        if (pers[i] > pers[i + 1])
                        {
                                temp = pers[i];
                                pers[i] = pers[i + 1];
                                pers[i + 1] = temp;
                                flag = 1;
                        }

                }
                if (flag == 0)
                        break;
                cout << "array after bubble sorting" << "\n\n";
                for (int i = 0; i < n; i++)
                {
                        cout << pers[i] << " " << "\n";
                }
        }


}
int forms::partitioning(int* a, int start, int end) //partition to put pivot in middle//
{
        int pivot = a[end];
        int partitionindex = start;
        int temp, i;
        for (i = start; i < end; i++)
        {
                if (a[i] <= pivot)
                {
                        temp = a[i];
                        a[i] = a[partitionindex];
                        a[partitionindex] = temp;
                        partitionindex++;
                }
```

```
	}
	temp = a[end];
	a[end] = a[partitionindex];
	a[partitionindex] = temp;
	return partitionindex;


}

void forms::quicksort(int* a, int start, int end)
{
	if (start < end)
	{
		int P_index = partitioning(a, start, end);
		quicksort(a, start, P_index - 1);
		quicksort(a, P_index + 1, end);
	}
}


void forms::stackformsadd() //Stack implementation//
{





	cout << "Enter name of person" << "\n";
	getline(cin, detail);
	if (getline(cin, detail))
	{

	}
```

```
cout << "enter admission fee to add on bill" << "\n";
cin >> admissionfee;

cout << "enter university registration fee to add on bill" << "\n";
cin >> registrationfee;

cout << "enter university id card fee to add on bill" << "\n";
cin >> idcard;



cout << "enter tution fee to add on bill" << "\n";
cin >> tution;

cout << "enter exam fee to add on bill " << "\n";
cin >> examfee;



cout << "enter semester examination fee to add on bill" << "\n";
cin >> semesterenrolfee;
if (cin.fail())
{
        cout << "please enter a integer" << "\n";
        cin.clear();
}
if (top == SIZE - 1)
{
        cout << "Stack is full" << "\n";
}
else
{
        top++;
        name[top] = detail;
        serialno[top] = serial;
        admission[top] = admissionfee;

        universityfee[top] = registrationfee;

        universityidcard[top] = idcard;

        tutionfee[top] = tution;

        exam[top] = examfee;

        semester[top] = semesterenrolfee;



    }


}


bool forms::isemptystack()

{

    if (top == -1)

        return true;

    else

        return false;


}

void forms::stackformsremove()

{

    if (isemptystack())

    {

        cout << "stack is empty" << "\n";

    }

    else

    {

        detail = name[top];

        serial = serialno[top];

        admissionfee = admission[top];

        registrationfee = universityfee[top];

        idcard = universityidcard[top];
```

```
		tution = tutionfee[top];
		examfee = exam[top];
		semesterenrolfee = semester[top];
		top--;
	}

}
void forms::displayformstack()
{
	if (isemptystack())
	{
		cout << "stack is empty" << "\n";
	}
	else
	{
		cout << "CASHIER MAINTAINING FORM STACK With serial number " << "\n\n";
		for (int i = 0; i <= top; i++)
		{
			cout << "-----------------------------------------------------" << "\n";
			cout << "Ref Riphah-ADM-Fall 2020" << "\n\n";
			cout << "23-01-20" << "\n\n";
			cout << "SERIAL NO :" << "\t" << serialno[i] << "\n\n";
			cout << "Candidates NAME :" << "\t" << name[i] << "\n\n";
			cout << "Details of your dues is given below:" << "\n\n";
			cout << "Particulars" << "\t\t" << "Account Heads" << "\t\t" << "Amount(Pak Rs:)" <<
"\n\n";
			cout << "One Time Charges" << "\t" << "Admission Fee" << "\t\t" << admission[i] <<
"\n\n";
			cout << "\t\t" << "University Registration Fee" << "\t" << universityfee[i] << "\n\n";
			cout << "\t\t\t" << "University ID Card" << "\t" << universityidcard[i] << "\n\n";
			cout << "\t\t\t" << "Tution Fee" << "\t" << "\t" << tutionfee[i] << "\n\n";
			cout << "\t\t\t" << "Examination Fee" << "\t" << "\t" << exam[i] << "\n\n";
			cout << "\t\t\t" << "Semester Enrollment Fee" << "\t" << semester[i] << "\n\n";
			totalfee = admission[i] + universityfee[i] + universityidcard[i] + tutionfee[i] + exam[i] + 
semester[i];
			cout << "Total fee & Dues for First Semester" << "\t\t" << totalfee << "\n\n";




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




		}



	}



}
```

TREE HEADERFILE

```
#include<iostream>
using namespace std;
class tree
{
private:
	struct bstnode
	{
		int data;
		bstnode* left;
		bstnode* right;
	};
	bstnode* root;
	bstnode* temp;
	bstnode* par;
	bstnode* rightnode;
	int value;
	bstnode* searchnode;
public:
	tree()
	{
		root = NULL;
	}
	void insertdata();
	void printbsttree(bstnode *newptr);
	void search();
	void displaybsttree();
};
```

TREE.CPP

```
#include"Tree.h"
void tree::insertdata() //inserting items in Tree//
{
	cout << "enter value to insert in tree" << "\n\n";
	temp = new bstnode;
	cin >> value;
	temp->data = value;
	temp->left = NULL;
	temp->right = NULL;
	par = NULL;
	if (root == NULL)
		root = temp;

	else
	{
		rightnode = root;
		while (rightnode != NULL)
		{
			par = rightnode;
			if (value > rightnode->data)
				rightnode = rightnode->right;
			else
				rightnode = rightnode->left;
		}
		if (value < par->data)
			par->left = temp;
		else
			par->right = temp;









	}
}
void tree::displaybsttree()
{
	printbsttree(root);
```

```
}



void tree::printbsttree(bstnode *newptr)
{
	if(newptr != NULL)
	{
		printbsttree(newptr->left);
		cout << " " << newptr->data<<"\n";
		printbsttree(newptr->right);
	}
}
void tree::search()
{
	int item;
	int depth = 0;
	bstnode* searching = new bstnode;
	searching = root;
	cout << "enter item to search" << "\n";
	cin >> item;
	while (searching != NULL)
	{
		depth++;
		if (searching->data == item)
		{
			cout << "data found in tree" << "\n";
			return;
		}
		else if (searching->data > item)
			searching = searching->left;
		else
			searching = searching->right;
```

```
        }
        cout << "Data not found" << "\n";
        return;
}
```

MAIN.CPP

```
#include<iomanip>
#include<stdio.h>
#include<stdlib.h>
  #include<windows.h>
  #include "transaction.h"
  #include "user.h"
  #include <conio.h>
  #include <iostream>
#include"feedepartement.h"
#include"Tree.h"
  using namespace std;

  int main()
{
          system("CLS");
          cout << endl <<
"================================================================================";
                  HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(m_hConsole,
                  BACKGROUND_RED |
                  BACKGROUND_GREEN |
                  BACKGROUND_BLUE);
        cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E R ";
          SetConsoleTextAttribute(m_hConsole,
          FOREGROUND_RED |
                FOREGROUND_GREEN |
                FOREGROUND_BLUE);
        cout <<
"=============================================================================="<
<endl;
        forms f1;
          int choice1, choice2, choice3; //variables to get user choices
          user * u; //pointer to user class
          u = new user; //user object
          transaction * t = new transaction; //dynamic memory allocation to pointer of transac



                cout << "Enter Mode:" << endl;
        cout << "1. Log In." << endl;
          cout << "2. Sign Up." << endl;
          cin >> choice1; //gets mode input
          u->getData(choice1); //gets login or sign up credentials from user


                do
                {
          system("CLS");
                cout << endl <<
"==============================================================================";
                        HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
              SetConsoleTextAttribute(m_hConsole,
                BACKGROUND_RED |
                        BACKGROUND_GREEN |
                        BACKGROUND_BLUE);
              cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E R ";
                SetConsoleTextAttribute(m_hConsole,
                FOREGROUND_RED |
                    FOREGROUND_GREEN |

                    FOREGROUND_BLUE);

                cout <<
"===============================================================================<
<endl;


                     cout << "Enter Operation:" << endl;

                cout << "1. Add Income." << endl;

                 cout << "2. Add Expense." << endl;

                 cout << "3. Display Account Balance." << endl;

                 cout << "4. Display All Transactions." << endl;

                 cout << "5. Genrate Text File Report." << endl;

                 cout << "6. Show Summary(Graph)." << endl;

                 cout << "7. Stack and Queue of Fee departement" << endl;

                 cout << "8.Various Sortings of persons" << endl;

                 cout << "9.Make a tree of persons" << endl;

                 cout << "10.Exit" << endl;

                 cin >> choice2;

                 if (choice2 == 1)

                      {

                      system("CLS");

                      cout << endl <<
"===============================================================================";

                         HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

                     SetConsoleTextAttribute(m_hConsole,

                        BACKGROUND_RED |

                         BACKGROUND_GREEN |

                          BACKGROUND_BLUE);

                     cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E R
";

                     SetConsoleTextAttribute(m_hConsole,

                         FOREGROUND_RED |

                          FOREGROUND_GREEN |

                          FOREGROUND_BLUE);
```

```
                              cout <<
"==================================================================================="<
<endl;

                                      t->getAmount(u->getUsername(), u -> getPassword(), 1); //files of the user
will be created on its username.txt(final balan


                                    }
                      else if (choice2 == 2)
                            {
                              system("CLS");
                              cout << endl <<
"===================================================================================";
                                      HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                            SetConsoleTextAttribute(m_hConsole,
                                      BACKGROUND_RED |
                                      BACKGROUND_GREEN |
                                  BACKGROUND_BLUE);
                            cout << " RIPAHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E
R ";
                              SetConsoleTextAttribute(m_hConsole,
                                    FOREGROUND_RED |
                                      FOREGROUND_GREEN |
                                      FOREGROUND_BLUE);
                            cout <<
"==================================================================================="<
<endl;


                                      t->getAmount(u->getUsername(), u -> getPassword(), 2); //files of the user
will be created on its username.txt(final balan


                                      }
                    else if (choice2 == 3)
                    {
                              system("CLS");
                            cout << endl <<
"================================================================================ = ";
```

```
                        HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                    SetConsoleTextAttribute(m_hConsole,
                        BACKGROUND_RED |
                        BACKGROUND_GREEN |
                        BACKGROUND_BLUE);
                    cout << " RIPHAH INTERNATIONAL UNIVERSITY - S P E N D I N G - T R A C K E R ";
                    SetConsoleTextAttribute(m_hConsole,
                        FOREGROUND_RED |
                        FOREGROUND_GREEN |
                        FOREGROUND_BLUE);
                   cout <<
"============================================================================ =
"<<endl;


                        t->finalBalance(u -> getUsername()); //to display final balance in account


              }
               else if (choice2 == 4)
                    {
                    system("CLS");
                    cout << endl <<
"============================================================================ = ";
                        HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                    SetConsoleTextAttribute(m_hConsole,
                        BACKGROUND_RED |
                        BACKGROUND_GREEN |
                        BACKGROUND_BLUE);
                    cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E
R ";
                    SetConsoleTextAttribute(m_hConsole,
                        FOREGROUND_RED |
                        FOREGROUND_GREEN |
                        FOREGROUND_BLUE);
```

```
                              cout <<
"============================================================================ =
"<<endl;


                                      t->displayTransaction(u->getPassword());
                              }
                      else if (choice2 == 5)
                              {
                              system("CLS");
                              cout << endl <<
"============================================================================ = ";
                                    HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                              SetConsoleTextAttribute(m_hConsole,
                                    BACKGROUND_RED |
                                      BACKGROUND_GREEN |
                                      BACKGROUND_BLUE);
                            cout << " RIPHAH INTERNATIONAL - S P E N D I N G - T R A C K E R ";
                            SetConsoleTextAttribute(m_hConsole,
                                    FOREGROUND_RED |
                                    FOREGROUND_GREEN |
                                      FOREGROUND_BLUE);
                              cout <<
"============================================================================ =
"<<endl;
                              t->generateReport(u->getPassword());


                    }
                      else if (choice2 == 6)
                              {
                              system("CLS");
                              cout << endl <<
"============================================================================ = ";
                                      HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                              SetConsoleTextAttribute(m_hConsole,
                                      BACKGROUND_RED |
                                        BACKGROUND_GREEN |
                                        BACKGROUND_BLUE);
                              cout << " RIPHAH INTERNATIONAL - U N I V E R S I T Y - S P E N D I N G - T R A C K E
R ";
                              SetConsoleTextAttribute(m_hConsole,
                                        FOREGROUND_RED |
                                        FOREGROUND_GREEN |
                                        FOREGROUND_BLUE);
                              cout <<
"=============================================================================== =
"<<endl;


                                        t->summary(u->getPassword());
                                        break;


                              }


                    else if (choice2 == 7)
                    {
                    forms f1;
                    string choice;
                    int flag = 1;
                    while (flag == 1)
                    {
                              cout << "A.Insert person in queue\nB.Remove persons\nC.Display persons in
queue\nD.Push persons info\nE.POP persons info\nF.Display persons info\nG.EXIT\n" << "\n";
                              cin >> choice;
                              if (choice != "A" & choice != "B" & choice != "C" & choice != "D" & choice != "E" &
choice != "F" & choice != "G")
                              {
                                        cout << "incorrect choice " << "\n";
                              }
                              else if (choice == "A")
                              {
                                        f1.personenqueue();
}
else if (choice == "B")
{
        f1.persondequeue();


}
else if (choice == "C")
{
        f1.displayform();


}


if (choice == "D")
{
        f1.stackformsadd();
}


else if (choice == "E")
{
        f1.stackformsremove();
}
else if (choice == "F")
{
        f1.displayformstack();
}
else if (choice == "G")
{
        goto q;


}
```

```
                    }

        }
                    else if (choice2 == 8)
                    {
                    string dec;
                    cout << "A.Selection Sort\nB.Bubble Sort\nC.Quick Sort\nD.Exit" << "\n";
                    cin >> dec;
                    if (dec != "A" & dec != "B" & dec != "C"&dec!="C"&dec!="D")
                    {
                        cout << "wrong choice" << "\n\n";
                   }
                    else if (dec == "A")
                    {
                        f1.selectionsort();
                    }
                    else if (dec == "B")
                    {
                        f1.bubblesort();
                    }
                    else if (dec == "C")
                    {
                        int size;
                        int* arr;

                        cout << "enter number of elements in array" << "\n";
                        cin >> size;
                        arr = new int[size];
                        cout << "enter elements" << "\n";
                        for (int i = 0; i < size; i++)
                        {
                            cin >> arr[i];
                        }
```

```
                f1.quicksort(arr, 0, size - 1);
                cout << "elements after quick sorting is" << "\n";
                for (int i = 0; i < size; i++)
                {
                        cout << arr[i] << " ";
                }

        }










}
        else if (choice2 == 9)
        {
        tree t1;
        int flag=1;
        string ch;
        while (flag == 1)
        {
                cout << "A.Insert item\nB.display\nC.Search\nD.Exit" << "\n";
                cin >> ch;

                if (ch != "A" & ch != "B" & ch != "C"&ch!="D")
```

```
                    {
                            cout << "wrong choice" << "\n";
                    }
                    else if (ch == "A")
                    {
                            t1.insertdata();
                    }
                    else if (ch == "B")
                    {
                            t1.displaybsttree();
                    }
                    else if (ch == "C")
                    {
                            t1.search();
                    }
                    else if (ch == "D")
                    {
                            flag = 0;
                            goto e;
                    }

            }


 }
            else if (choice2 == 10)
                  {
                    break;
                    }
            else
                    {
                    system("CLS");
                    cout << endl <<
"=========================================================================== = ";
```

```
                        HANDLE m_hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
                    SetConsoleTextAttribute(m_hConsole,
                         BACKGROUND_RED |
                        BACKGROUND_GREEN |
                         BACKGROUND_BLUE);
                     cout << " RIPHAH INTERNATIONAL UNIVERSITY- S P E N D I N G - T R A C K E R ";
                     SetConsoleTextAttribute(m_hConsole,
                       FOREGROUND_RED |
                         FOREGROUND_GREEN |
                         FOREGROUND_BLUE);
                     cout <<
"=============================================================================== =
"<<endl;


                        cout << "SORRY!!! INVALID CHOICE SELECTED." << endl;
                     }



                    cout << endl;
                  e:    q:
                 cout << endl << "Do you want to do operation again?" << endl;
                 cout << "1. YES." << endl;
                 cout << "2. NO." << endl;
                 cin >> choice3;
                 }
                 while (choice3 == 1);


                     _getch();
                 }
```