**Bank Management Project**

****

Session: 2021-2024

**Submitted by:**

Abdullah Nasir 2021-CS-113

**Supervised by:**

Dr. Awais Hassan

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Description:**

A bank is place for safe storage of your money. This system helps different people related to a bank to work efficiently and easily. It is designed for a manager, employees working under the manager and customers using different bank services. This system helps the manager to add, or delete employees, keep a check on employees and customers, and sort data according to his needs. Employee can add customers, update his daily working hours, and help customers to update their accounts. Similarly, Customers can deposit, withdraw and transfer money.

**Users:**

There are three main users of this application.

1. Manager
2. Employee
3. Customer

**Functional Requirements:**

Manager

* can hire new employees
* can fire existing employees
* can check employee list
* can sort employee by age
* can sort employee by working hours of employees
* can delete existing customers
* can check cash outflow of bank
* can check cash inflow of bank
* can check customers issue and complain
* can check customers list
* can sort customers by account balance

Employee

* can add new customers
* can update accounts of existing customers
* can update daily working hours

Customer

* can check account balance
* can deposit cash
* can withdraw cash
* can transfer cash
* can permanently close bank account
* can complain any issue

**Data Structures:**

Following data structures are used in the program.

1. Employee names
2. Employee education
3. Employee age
4. Employee experience
5. Employee working hours
6. Customer names
7. Customer account balance

**Functions Prototypes:**

Following are the prototypes of functions used in the program.

1. void header ()
2. char whichUser ()
3. int menu (int user\_password)
4. void hireEmployee ()
5. void fireEmployee (string employee)
6. void displayEmployee ()
7. void ageSort (int start)
8. void workSort (int start)
9. void deleteCustomer (string customer)
10. void displayCustomers ()
11. void customerSort (int start)
12. void addCustomer ()
13. void updateCustomer (string change)
14. void cashDeposit (string customer)
15. void cashWithdraw (string customer)
16. void cashTransfer (string recipient, int transfer\_cash, string customer)
17. void closeAccount (string customer)
18. void load ()
19. string parseRecord (string record, int field)
20. void store ()

**Functions work flow diagram:**

main ()

load ()

header ()

parseRecord ()

whichUser ()

menu ()

Customer

Manager

Employee

Manager

fireEmployee ()

hireEmployee ()

displayCustomer ()

displayEmployee ()

workSort ()

ageSort ()

customerSort ()

deleteCustomer ()

Store ()

Employee

addCustomer ()

updateCustomer ()

Store ()

Customer

cashWithdraw ()

cashDeposit ()

Store ()

cashTransfer ()

closeAccount ()

**Code with Wireframes:**

**Loading data from file**

Load ();

void load()

{

fstream file;

string line;

file.open( "customer.txt", ios :: in);

while ( file >> line )

{

customer\_name[ customerCount ] = parseRecord( line, 0);

in\_account[ customerCount ] = stoi( parseRecord( line, 1) );

customerCount ++;

}

file.close();

file.open( "employee.txt", ios :: in);

while ( file >> line )

{

employee\_name[ employeeCount ] = parseRecord( line, 0);

education[ employeeCount ] = parseRecord( line, 1);

age[ employeeCount ] = stoi( parseRecord( line, 2) );

experience[ employeeCount ] = stoi( parseRecord( line, 3) );

working\_hour[ employeeCount ] = stoi( parseRecord( line, 4) );

employeeCount ++;

}

file.close();

}

//\_\_\_\_\_\_\_\_\_\_\_Parse Record\_\_\_\_\_\_\_\_\_\_\_

string parseRecord(string record, int field)

{

int commaCount = 0;

string item;

for (int x = 0; x < record.length(); x++)

{

if (record[x] == ',')

{

commaCount = commaCount + 1;

}

else if (commaCount == field)

{

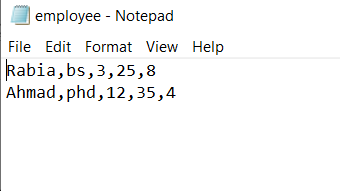
item = item + record[x];

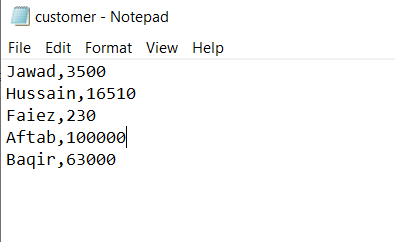
}

}

return item;

}





**Displaying header and asking for user category**

void header ()

{

cout <<"$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" <<endl;

cout <<"$ $" <<endl;

cout <<"$ Bank Management System $" <<endl;

cout <<"$ $" <<endl;

cout <<"$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" <<endl<<endl;

}

char whichUser ()

{

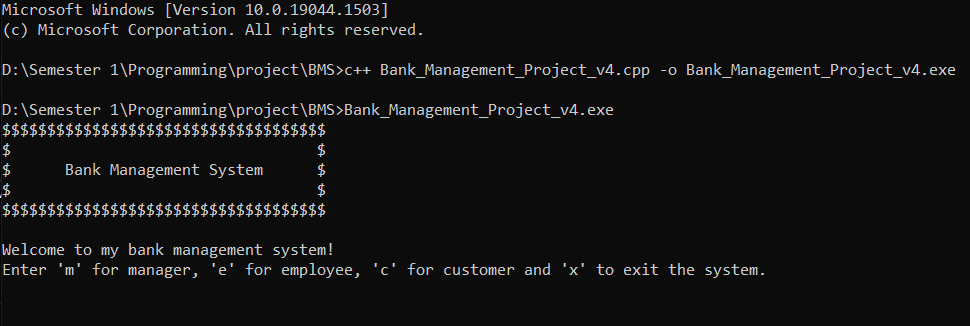
cout <<"Welcome to my bank management system!"<<endl;

cout <<"Enter 'm' for manager, 'e' for employee, 'c' for customer and 'x' to exit the system."<<endl;

cin >>user;

return user;

}



**Asking for username and password**

if (user == 'm')

{

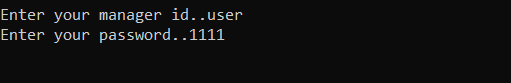
cout<<"Enter your manager id..";

cin>>user\_id;

cout<<"Enter your password..";

cin>>user\_password;

}



else if (user == 'e')

{

cout<<"Enter employee name: ";

cin >>employee;

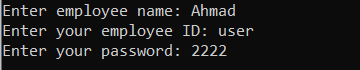
cout<<"Enter your employee ID: ";

cin>>user\_id;

cout<<"Enter your password: ";

cin>>user\_password;

}



else if (user == 'c')

{

cout <<"Enter your name: ";

cin >> customer;

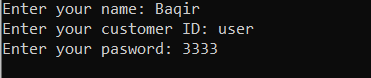
cout <<"Enter your customer ID: ";

cin >>user\_id;

cout <<"Enter your password: ";

cin >>user\_password;

}



else if (user == 'x')

{

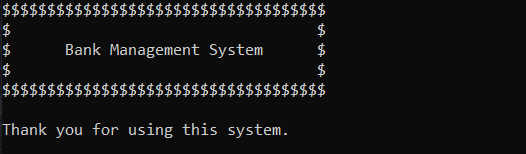
break;

}

system("cls");

header ();

cout<<"Thank you for using this system." <<endl;



else

{

cout <<"Invalid user."<<endl <<"Press a key to continue."<<endl;

getch();

system("cls");

continue;

}



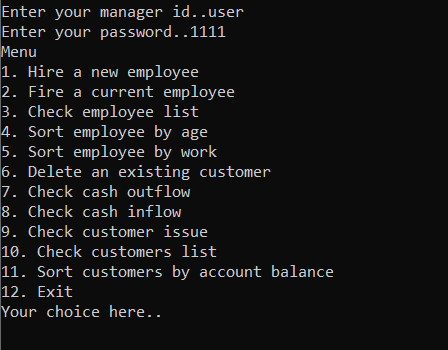
**Verifying password and displaying menu**

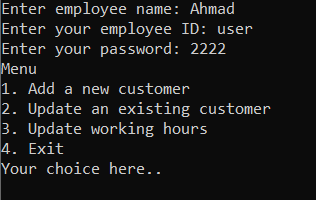
if ( user\_id == "user" && ( user\_password == 1111 || user\_password == 2222 || user\_password == 3333 ) )

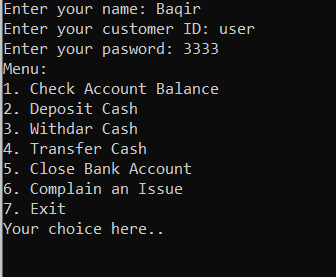
{

choice = menu ( user\_password );

}







else

{

cout <<"Invalid ID or password." <<endl;

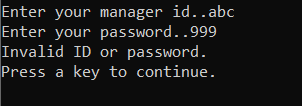
cout <<"Press a key to continue." <<endl;

getch();

system("cls");

break;

}



**Entering manager’s menu**

**Hiring new employee:**

if (choice == 1)//Hiring a new employee

{

hireEmployee();

}//end of hiring

void hireEmployee()

{

for ( ; employeeCount < MAX\_RECORDS; )

{

cout <<"Enter name: ";

cin >>employee\_name[employeeCount];

cout <<"Enter education: ";

cin >>education[employeeCount];

cout <<"Enter experience: ";

cin >>experience[employeeCount];

cout <<"Enter age: ";

cin >>age[employeeCount];

employeeCount ++;

cout <<endl <<"Username for this employee is 'user'."<<endl;

cout <<"Password for this employee is '2222'."<<endl;

break;

}

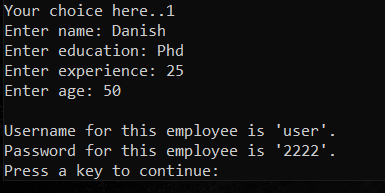
if ( employeeCount == MAX\_RECORDS )

{

cout <<"Cannot add further employee.";

}

}



**Firing an Employee:**

else if (choice == 2)//firing an employee

{

cout <<"Enter name of employee: ";

cin >>employee;

fireEmployee ( employee );

}//end of firing

void fireEmployee( string employee )

{

for (int i = 0; i < employeeCount; i++)

{

if ( employee == employee\_name[i] )

{

cout <<employee\_name[i] <<" has been fired." <<endl;

experience[i] = age[i] = -1;

employee\_name[i] = education[i] = "zzzzzz";

break;

}

if ( i == employeeCount - 1 )

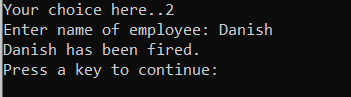
{

cout <<"No Employee with this name." <<endl;

}

}

}



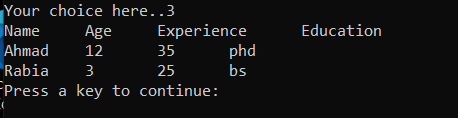
**Checking Employee list:**

else if (choice == 3)//checking employee list

{

displayEmployees();

}//end of checking employee list



**Sorting employee by age**

else if (choice == 4)//sorting by age

{

for (int i = 0; i < employeeCount; i++)

{

ageSort( i );

}

cout <<"Name \t Age \t Experience \t Education \t Working Hour" <<endl;

for (int i = 0; i < employeeCount; i++)

{

if ( age[i] != -1 && age[i] != 0 )

{

cout <<employee\_name[i] <<" \t " <<age[i] <<" \t\t " <<experience[i] <<" \t\t " <<working\_hour[i] <<endl;

}

}

}//end of sorting by age

void ageSort( int start )

{

string l\_employee, l\_education;

int l\_experience, l\_age, l\_working\_hour;

for (int o = start + 1; o < employeeCount; o++)

{

if ( age[ start ] < age[o] )

{

l\_age = age[ start ];

age[ start ] = age[o];

age[o] = l\_age;

l\_employee = employee\_name[ start ];

employee\_name[ start ] = employee\_name[o];

employee\_name[o] = l\_employee;

l\_working\_hour = working\_hour[ start ];

working\_hour[ start ] = working\_hour[o];

working\_hour[o] = l\_working\_hour;

l\_experience = experience[ start ];

experience[ start ] = experience[o];

experience[o] = l\_experience;

l\_education = education[ start ];

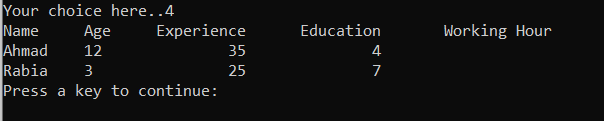
education[ start ] = education[o];

education[o] = l\_education;

}

}

}



**Sorting employee by working hours**

else if (choice == 5)//sorting by work

{

for ( int i = 0; i < employeeCount; i++)

{

workSort( i );

}

cout <<"Name \t Age \t Experience \t Education \t Working Hour" <<endl;

for (int i = 0; i < employeeCount; i++)

{

if ( working\_hour[i] != 0 && working\_hour[i] != -1 )

{

cout <<employee\_name[i] <<" \t " <<age[i] <<" \t\t " <<experience[i] <<" \t\t " <<working\_hour[i] <<endl;

}

}

}//end of sorting by work

void workSort( int start )

{

string l\_employee, l\_education;

int l\_experience, l\_age, l\_working\_hour;

for (int o = start + 1; o < employeeCount; o++)

{

if ( working\_hour[ start ] < working\_hour[o] )

{

l\_working\_hour = working\_hour[ start ];

working\_hour[ start ] = working\_hour[o];

working\_hour[o] = l\_working\_hour;

l\_employee = employee\_name[ start ];

employee\_name[ start ] = employee\_name[o];

employee\_name[o] = l\_employee;

l\_age = age[ start ];

age[ start ] = age[o];

age[o] = l\_age;

l\_experience = experience[ start ];

experience[ start ] = experience[o];

experience[o] = l\_experience;

l\_education = education[ start ];

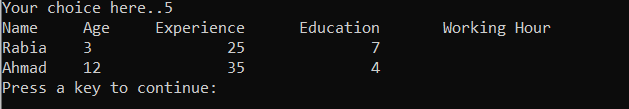
education[ start ] = education[o];

education[o] = l\_education;

}

}

}



**Deleting customers**

else if (choice == 6)//deleting customer

{

cout <<"Enter customer you want to delete: ";

cin >>customer;

deleteCustomer( customer );

}// end of deleting customers

else if (choice == 7)//checking cash outflow

{

cout<<"Cash outflow is: "<<cash\_outflow <<endl;

}//ending cash flow check

else if (choice == 8)//checking cash inflow

{

cout<<"Cash inflow today is: "<<cash\_inflow <<endl;

}//ending cash inflow

void deleteCustomer( string customer )

{

for (int i = 0; i < customerCount; i++)

{

if ( customer\_name[i] == customer )

{

cout <<"Account of " <<customer[i] <<" has been blocked."<<endl;

customer\_name[i] = "zzzzzz";

in\_account[i] = -1;

break;

}

if ( i == customerCount - 1 )

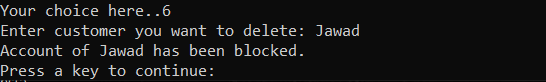
{

cout <<"No Customer with this name." <<endl;

}

}

}



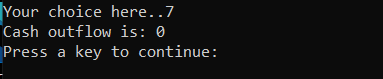
**Cash outflow today**

else if (choice == 7)//checking cash outflow

{

cout<<"Cash outflow is: "<<cash\_outflow <<endl;

}//ending cash flow check



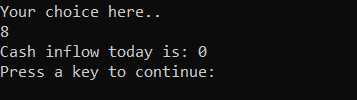
**Cash inflow today**

else if (choice == 8)//checking cash inflow

{

cout<<"Cash inflow today is: "<<cash\_inflow <<endl;

}//ending cash inflow



**Displaying customer issue**

else if (choice == 9)//customer issues

{

if( issue[0] != '\0' )

{

cout<<issue<<endl;

}

else

{

cout <<"No issues reported."<<endl;

}

}//ending customer issues



**Displaying customers list**

else if (choice == 10)//checking customer list

{

displayCustomers();

}//end of checking customer list

void displayCustomers()

{

for (int i = 0; i < customerCount; i++)

{

if ( in\_account[i] != 0 && in\_account[i] != -1 )

{

cout <<"Name \t Account Balance" <<endl;

break;

}

}

for (int i = 0; i < customerCount; i++)

{

if ( in\_account[i] != 0 && in\_account[i] != -1 )

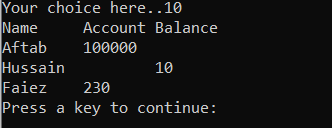
{

cout <<customer\_name[i]<<" \t " <<in\_account[i] <<endl;

}

}

}



**Sorting customers by account balance**

else if (choice == 11)//sorting customers

{

for (int i = 0; i < customerCount; i++)

{

customerSort( i );

}

cout <<"Name \t Account Balance" <<endl;

for (int i = 0; i < customerCount; i++)

{

if ( in\_account[i] != -1 && in\_account != 0 )

{

cout <<customer\_name[i] <<" \t " <<in\_account[i] <<endl;

}

}

}//end of customer sorting

void customerSort( int start )

{

string l\_customer;

int l\_in\_account;

for ( int o = start + 1; o < customerCount; o++)

{

if ( in\_account[ start ] < in\_account[o] )

{

l\_in\_account = in\_account[ start ];

in\_account[ start ] = in\_account[o];

in\_account[o] = l\_in\_account;

l\_customer = customer\_name[ start ];

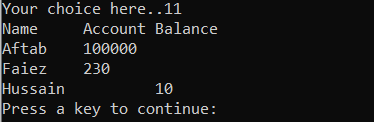
customer\_name[ start ] = customer\_name[o];

customer\_name[o] = l\_customer;

}

}

}



**Exiting from manager menu**

else if (choice == 12)//exiting

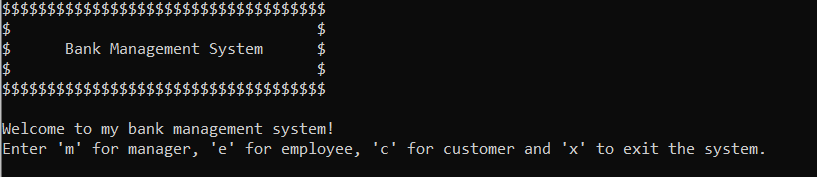
{

system("cls");

break;

}//end of exiting





**Entering employee menu**

**Adding new customer**

if (choice == 1)//adding new customer

{

addCustomer();

}//ending adding new customer

void addCustomer()

{

for ( ; customerCount < MAX\_RECORDS; )

{

cout <<"Enter full name: ";

cin >>customer\_name[customerCount];

cout <<"Enter cash deposited: ";

cin >>in\_account[customerCount];

customerCount ++;

cout <<endl <<"Username for this customer is 'user'."<<endl;

cout <<"Password for this customer is '3333'."<<endl;

break;

}

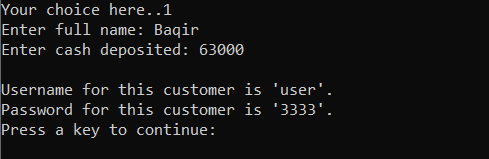
if ( customerCount == MAX\_RECORDS )

{

cout <<"Cannot add further customers." <<endl;

}

}



**Updating an existing customer**

else if (choice == 2)//updating an existing account

{

cout <<"Enter the name of customer: ";

cin >>changes;

updateCustomer( changes );

}//end of updating customer

void updateCustomer( string changes )

{

for (int i = 0; i < customerCount; i++)

{

if ( changes == customer\_name[i] )

{

cout <<"Enter 'd' to deposit cash and 'w' to withdraw cash: ";

cin >>changes\_character;

if (changes\_character == 'd')

{

cout <<"Enter cash you want to deposit: ";

cin >>cash\_change;

in\_account[i] = in\_account[i] + cash\_change;

cout <<"New account balance is "<< in\_account[i] <<endl;

cash\_inflow = cash\_inflow + cash\_change;

break;

}

else if (changes\_character == 'w')

{

cout <<"Enter cash you want to withdraw: ";

cin >>cash\_change;

if ( in\_account[i] > cash\_change )

{

in\_account[i] = in\_account[i] - cash\_change;

cout <<"New account balance is "<< in\_account[i] <<endl;

cash\_outflow = cash\_outflow + cash\_change;

break;

}

else

{

cout <<"Account limit Exceeded." <<endl;

break;

}

}

else

{

cout <<"Invalid choice." <<endl;

break;

}

}

if ( i == customerCount )

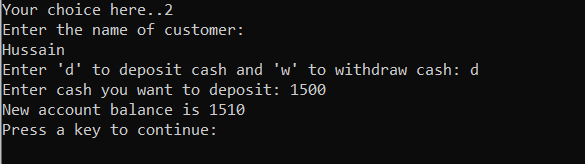
{

cout <<"No Customer with such name." <<endl;

}

}

}



**Updating today’s working hours**

else if ( choice == 3)//update working hours

{

for (int i = 0; i < employeeCount; i++)

{

if( employee == employee\_name[i] )

{

cout <<"Enter your today's working hour: ";

cin >>working\_hour[i];

break;

}

if ( i == employeeCount - 1 )

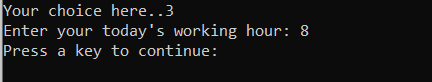
{

cout <<"No employee with this name yet."<<endl;

}

}

}//end of updating working hours



**Exiting employee menu**

else if (choice == 4)//exiting

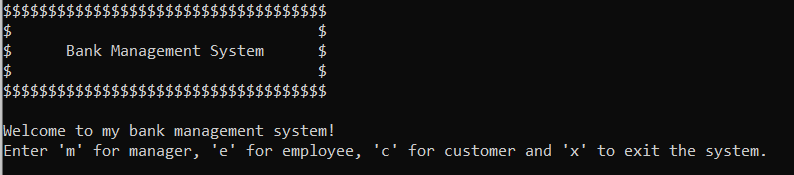
{

system("cls");

break;

}//end of exiting





**Entering customer menu**

**Checking account balance**

if (choice == 1)//checking account balance

{

for ( int i = 0; i < customerCount; i++)

{

if (customer == customer\_name[i])

{

cout <<"Your balance is " <<in\_account[i] <<endl;

break;

}

if ( i == customerCount - 1 )

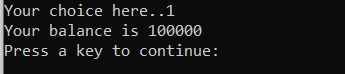
{

cout <<"No customer with this name.";

}

}

}//end of checking account balance



**Depositing cash**

else if (choice == 2)//depositing cash in account

{

cashDeposit( customer );

}//end of cash deposit

void cashDeposit( string customer )

{

for (int i = 0; i < customerCount; i++ )

{

if ( customer == customer\_name[i] )

{

cout <<"Enter cash you want to deposit: ";

cin >>cash\_change;

in\_account[i] = in\_account[i] + cash\_change;

cout <<"New account balance is "<< in\_account[i] <<endl;

cash\_inflow = cash\_inflow + cash\_change;

break;

}

if ( i == customerCount - 1 )

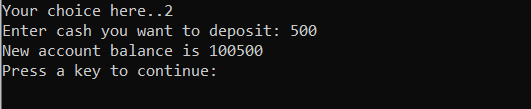
{

cout <<"No customer with this name." <<endl;

}

}

}



**Withdrawing cash**

else if (choice == 3)//withdraw cash from account

{

cashWithdraw( customer );

}//end of cash withdraw

void cashWithdraw( string customer )

{

for (int i = 0; i < customerCount; i++)

{

if ( customer == customer\_name[i] )

{

cout <<"Enter cash you want to withdraw: ";

cin >>cash\_change;

if ( in\_account[i] > cash\_change )

{

in\_account[i] = in\_account[i] - cash\_change;

cout <<"New account balance is "<< in\_account[i] <<endl;

cash\_outflow = cash\_outflow + cash\_change;

break;

}

else

{

cout <<"Account limit Exceeded." <<endl;

break;

}

}

if ( i == customerCount - 1 )

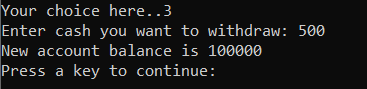
{

cout <<"No customer with this name." <<endl;

}

}

}



**Transferring cash**

else if (choice == 4)//transfer cash from account

{

cout <<"Enter recipient's name: ";

cin >>recipient;

cout <<"Enter the amount to transfer: ";

cin >>transfer\_cash;

cashTransfer( recipient, transfer\_cash, customer );

}//end of transferring

void cashTransfer( string recipient, int transfer\_cash, string customer )

{

for (int i = 0; i < customerCount; i++)

{

for (int o = 0; o < customerCount; o++)

{

if (customer == customer\_name[i] )

{

if ( recipient == customer\_name[o] )

{

if ( transfer\_cash < in\_account[i] )

{

in\_account[i] = in\_account[i] - transfer\_cash;

in\_account[o] = in\_account[o] + transfer\_cash;

cout <<"Your new account balance is "<< in\_account[i] <<endl;

break;

}

else

{

cout <<"Account limit exceeded." <<endl;

}

}

else if ( o == customerCount)

{

cout <<"No recipient with this name." <<endl;

}

}

else if ( customerCount == i - 1 )

{

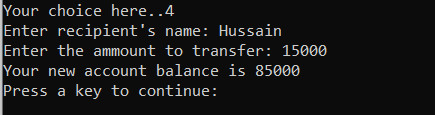
cout <<"No customer with this name." <<endl;

}

}

}

}



**Closing bank account**

else if (choice == 5)//deleting account

{

cout <<"Are you sure you want to permanently close your bank account?" <<endl;

cout <<"Enter 'y' for yes: ";

cin >>changes\_character;

if ( changes\_character == 'y')

{

closeAccount( customer );

}

else

{

cout <<"Account not deleted." <<endl <<"You can still use all the services provided by the bank." <<endl;

}

}// end of deleting account

void closeAccount( string customer )

{

for (int i = 0; i < customerCount; i++)

{

if ( customer\_name[i] == customer )

{

cout <<"Account of " <<customer[i] <<" has been closed."<<endl;

cash\_outflow = cash\_outflow + in\_account[i];

customer\_name[i] = "zzzzzz";

in\_account[i] = -1;

break;

}

if ( i == customerCount - 1 )

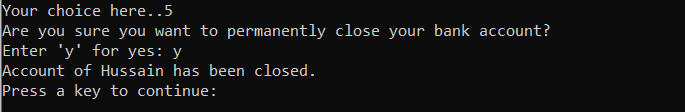
{

cout <<"No Customer with this name." <<endl;

}

}

}



**Store in file:**

store();

void store()

{

fstream file;

file.open( "customer.txt", ios :: out);

for (int i = 0; i < customerCount; i++)

{

if (in\_account[i] != 0 && in\_account[i] != -1)

{

file << customer\_name[i] <<"," <<in\_account[i] <<endl;

}

}

file.close();

file.open( "employee.txt", ios :: out);

for (int i = 0; i < employeeCount; i++)

{

if (age[i] != 0 && age[i] != -1)

{

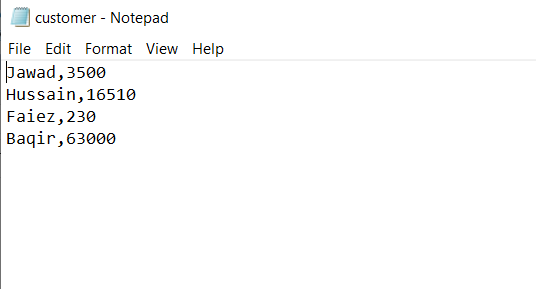
file << employee\_name[i] <<"," <<education[i] <<"," <<age[i] <<"," <<experience[i] <<"," <<working\_hour[i] <<endl;

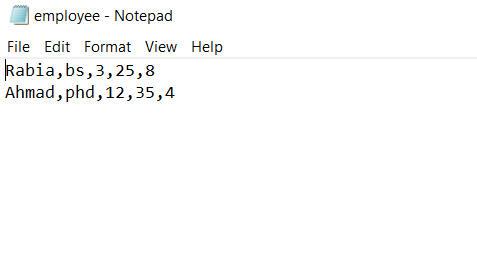
}

}

file.close();

}

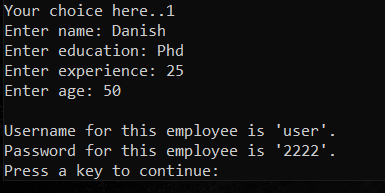




**Test Cases:**

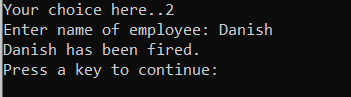
**Input-** Danish, Phd, 25, 50 (hiring employee)

**Output-**



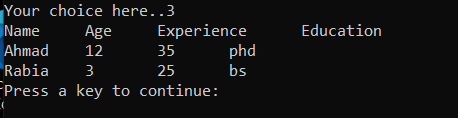
**Input-** Danish (firing employee)

**Output-**



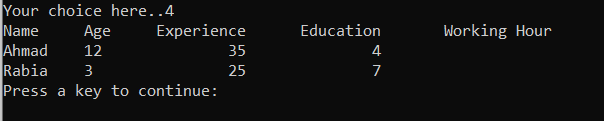
**Input-** (display Employee)

**Output-**



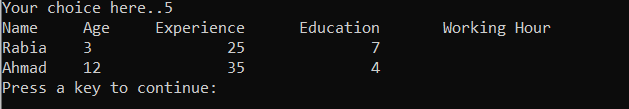
**Input-** (sorting by Age)

**Output-**



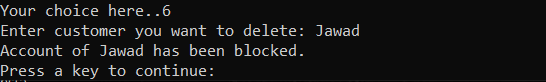
**Input-** (sort by working hours)

**Output-**



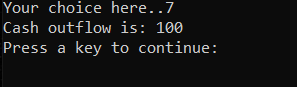
**Input-** Jawad (blocking account)

**Output-**



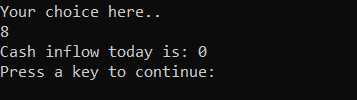
**Input-** (cash outflow from bank today)

**Output-**



**Input-** (cash inflow from bank today)

**Output-**



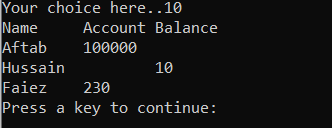
**Input-** (no issues)

**Output-**



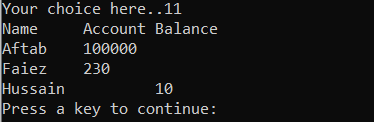
**Input-** (display customer’s list)

**Output-**



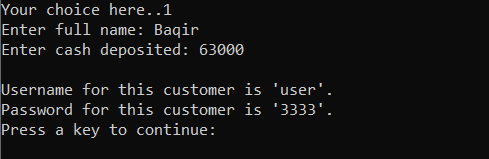
**Input-** (sorting by account balance)

**Output-**



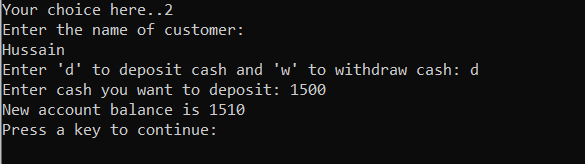
**Input-** Baqir, 63000 (entering customers)

**Output-**



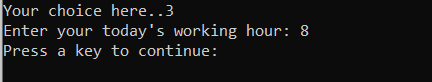
**Input-** Hussain, d, 1500 (depositing and withdrawing cash)

**Output-**



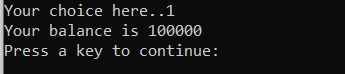
**Input-** 8 (entering daily working hours)

**Output-**



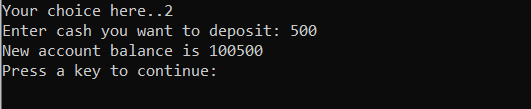
**Input-** Aftab(checking account balance)

**Output-**



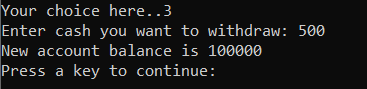
**Input-** Aftab, 500 (depositing cash in bank)

**Output-**



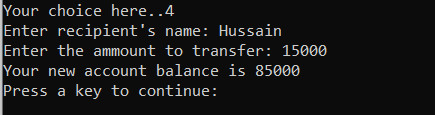
**Input-** Aftab, 500 (withdrawing cash from bank)

**Output-**



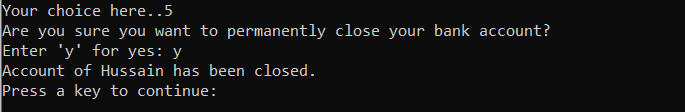
**Input-** Aftab, Hussain, 15000 (transferring cash)

**Output-**



**Input-** Hussain, y (closing bank account)

**Output-**



**Weakness in System:**

This system works well for pre-defined usernames and passwords but does not provide new users with new unique username and password. Employee and Customer are able to see menu even if name is incorrect but is successfully prohibited from using any of the system services.

**Future Recommendation:**

Bank management system can be improved if unique usernames and passwords are automatically generated by the program. This will help the system to work more efficiently and provide better security.

**Conclusion:**

This project is made on C++ and it helped us grab the concepts of this language very well. We came to know about the pros and cons of C++ and were able to practically implement the use of different functions of this language.

**Student Reg. No: Student Name:**

**Checked by:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting  **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title Page**, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project Poster is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined - Project **Code**. - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:** Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure  (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting  **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title Page**, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project Poster is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description - **Algorithms** and Flow Charts of all functions- **Test Cases** are defined - Project **Code**. - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:** Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure  (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Sorting Features  **Grade:** | Sort working 100% and generating useful report | Sorting Feature is working but sorted data is not useful for project. | Sorting feature is partial implemented | Project do not contain sorting |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types)- Demo Data Functionality Added-At least Two Unit Tests are defined. | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| Recommendation Feature | Proper meaning full recommendation is present into system | Partial Recommendation is implemented | Implemented but not meaning full. | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

**Student Reg. No: Student Name:**

**Checked by:**