Shaheed Zulfikar Ali Bhutto Institute of Science & Technology

COMPUTER SCIENCE DEPARTMENT

Total Marks: 7.5

Obtained Marks: _____

DSA (LAB)

Project Report

Project Title: Cash Withdrawal System Using ATM

Submitted To: Ma'am Tehreem Saboor

Student names: Shaish Khurshid, Hammad Ahmed Qureshi, Shayan Akbar Yousafzai.

Reg. Numbers: 2112136, 2112114, 2112137.

Shaheed Zulfikar Ali Bhutto Institute of Science & Technology

Contribution Of Members:

The idea of project is given by Shayan Akbar Yousafzai. Coding is done by Hammad Ahmed Qureshi, Shaish Khurshid. The report is prepared by all three members of group.

Title: Cash Withdrawal System Using ATM.

Introduction:

The project we choose about the cash withdrawal system through ATM. Where we take different kind of inputs from user like ATM PIN. After that if he enter the right PIN, compiler will display the message that "**Pin is Valid**" but if PIN is wrong "**Invalid**" message will be displayed. After that user will enter the amount he want to withdraw. After transaction remaining balance in account will be updated. We have used arrays (1 dimensional and 2 dimensional), stack, link list, linear search and recursion.

Goals & Objectives:

- Our main objective is to speed up the transactions done by customer.
- Our second objective is to save the time which is very important nowadays.
- To render accurate services to customer.
- No need to maintain the bulk of papers.

Code:

```
#include <iostream>
#define MAX 10
using namespace std;
int adjMatrix[MAX][MAX];
const int stackSize = MAX;


struct Vertex {
string label;
bool visited;
};

class Stack {

  private:
    int top;
  int arr[stackSize];

  public:
```

```
  Stack() {
    top = -1;
  }

void push(int value) {

  if (top + 1 >= stackSize) {
    cout << "Stack Overflow" << endl;
  } else {
    top = top + 1;
    arr[top] = value;
  }

}

int pop() {

  int stackPopVal;

  if (top <= -1) {
    cout << "Stack underflow" << endl;
  } else {
    stackPopVal = arr[top];
    top--;
  }
  return stackPopVal;

}

bool isStackEmpty() {
  if (top == -1) {
      return true;
  } else {
      return false;
  }
}
int peek() {
  return arr[top];
}


int getStackTop(){
 return top;
}

void display() {
  if (top >= 0) {
    cout << "Stack elements are:";
    for (int i = top; i >= 0; i--)
      cout << arr[i] << " ";
    cout << endl;
```

```
    } else
      cout << "Stack is empty";
  }

};

class DepthFirstSearch{

public:
struct Vertex* lstVertices[MAX];
int vertexCount = 0;
string bankServer;
bool isServerFound = false;

DepthFirstSearch(string bankName) {
 int i,j;
 for(i = 0; i< MAX; i++) {
  for(j = 0; j< MAX; j++) {
     adjMatrix[i][j] = 0;
    }
  }

 initializeServers();
 initializeEdges();

 bankServer = bankName;

}

void initializeServers(){
 addVertex("RAZI");
 addVertex("ALFALAH");
 addVertex("ALLIED");
 addVertex("MCB");
 addVertex("HBL");
 addVertex("BANK AL HABIB");
 addVertex("STATE BANK");
}

void initializeEdges(){
 addEdge(0, 1); //S - A
 addEdge(0, 2); // S - B
 addEdge(0, 3); // S - C
 addEdge(1, 4); // A - D
 addEdge(2, 4); // B - D
 addEdge(2, 5); // B - D
 addEdge(3, 5); // C - D

}

void addVertex(string label) {
```

```
 Vertex *vertex = new Vertex;
 vertex->label = label;
 vertex->visited = false;
 lstVertices[vertexCount++] = vertex;
}

void addEdge(int edgeStart,int edgeEnd) {
 adjMatrix[edgeStart][edgeEnd] = 1;
 adjMatrix[edgeEnd][edgeStart] = 1;
}

//display the vertex
void displayVertex(int vertexIndex) {
 if(lstVertices[vertexIndex]->label == bankServer && isServerFound == false){;
   isServerFound = true;
 }

 }

 int getAdjUnvisitedVertex(int vertexIndex) {
    int i;
    for(i = 0; i<vertexCount; i++) {
        if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)
        return i;
    }
    return -1;
}



void DFS(){
 Stack objStack;
 lstVertices[0]->visited = true;
 displayVertex(0);
 objStack.push(0);

 while(!objStack.isStackEmpty()){
 int unvisitedVertexIndex = getAdjUnvisitedVertex(objStack.peek());

 if(unvisitedVertexIndex == -1){
     objStack.pop();
  }else{
    lstVertices[unvisitedVertexIndex]->visited = true;
    displayVertex(unvisitedVertexIndex);
    objStack.push(unvisitedVertexIndex);
  }

 }

 } //Depth first search closes here/
```

```
};


struct Node {
  public: int value;
  Node * next;

};

Node * head;
Node * tail;

class SingleLinkList {

  public:

   SingleLinkList( ){
     head = NULL;
     tail = NULL;
    }

    void addCustomer(int d) {

     Node * newNode = new Node();
     newNode -> value = d;

     /*check head & tail are already set or not */
     if (head == NULL and tail == NULL) {
       newNode -> next = NULL;
       head = newNode;
       tail = newNode;
       return;
     }

     if (head == tail) {
       newNode -> next = NULL;
       head -> next = newNode;
       tail = newNode;
       return;
     }

     if (head != tail) {
       newNode -> next = NULL;
       tail -> next = newNode;
       tail = newNode;
     }

    }

  void showCustomer() {
```

```
    Node* temp = head;

    // Check for empty list.
    if (head == NULL) {
        cout << "List empty" << endl;
        return;
    }

  cout << "List of customers who used ATM Machine is: " << endl;

    // Traverse the list.
    while (temp != NULL) {
        cout << temp->value << " ";
        temp = temp->next;
    }

    cout << endl;
  }

};


struct TreeNode {
  int value;
  TreeNode * LeftNode;
  TreeNode * RightNode;

};

TreeNode *root = NULL;

class BST{
public:

    int transaction[8];
    int transactionCout = -1;

    BST( ){
     transactionCout = -1;
    }
    void insertBST(int data){

    TreeNode *newNode = new TreeNode();
    newNode->value = data;
    newNode->LeftNode = NULL;
    newNode->RightNode = NULL;


    if (root == NULL)
          {
        root = newNode;
```

```
      return;
    }
          else
          {
     TreeNode *tempNode = root;

     while( tempNode != NULL ){
       if(data < tempNode->value)
                  {
         if(tempNode->LeftNode == NULL){
           tempNode->LeftNode = newNode;
           break;
         }
                           else
                           {
           tempNode = tempNode->LeftNode;
         }

       }
                  else
                  {

         if(tempNode->RightNode == NULL)
                           {
           tempNode->RightNode = newNode;
           break;

         }
                           else
                           {
           tempNode = tempNode->RightNode;
         }

        }
       }
      }
    }

 void preOrderTraversal(TreeNode* node){

    if(node == NULL){
       return;
    }
    transactionCout = transactionCout + 1;
    transaction[transactionCout] = node->value;
    preOrderTraversal(node->LeftNode);
    preOrderTraversal(node->RightNode);
  }

};
```

```
class ATMMachine {

 int customerPin[8][2];
 int customerBalance[8][2];
 Stack transctionInStack;
 SingleLinkList customerUsesMachine;
 BST objBST;
 int transacitons[8];
 int transCount = -1;
 public:
 int highestTransactionAmount = 0;
  ATMMachine() {
    /* initialize all details about the customers on instance creation*/
    initCustomerPin();
    initCustomerBalance();
    transCount = -1;
 }
 bool isCardValid(int cardNumber) {
  //Validate customr card via linear search on Array//
  for (int i = 0; i < sizeof(customerBalance); i++) {
    if (cardNumber == customerBalance[i][0]) {
      return true;
    }
  }
  return false;
 }
 bool isPinValid(int cardNumber, int pin) {
  //search customer PIN via linear search on Array//
  for (int i = 0; i < sizeof(customerPin); i++) {
    if (customerPin[i][0] == cardNumber && customerPin[i][1] == pin) {
      return true;
    }
  }
  return false;

 }

 bool withdrawAmount(int cardNumber, int amount) {

  bool is_success = false;

  for (int i = 0; i < sizeof(customerBalance); i++) {
    if (cardNumber == customerBalance[i][0]) { /* first check customer with card no*/
      if (amount <= customerBalance[i][1]) {
        /* withdraw customer amount and update balance*/
        customerBalance[i][1] = customerBalance[i][1] - amount;
        is_success = true;
      } else {
```

```
        cout << "Insufficient Balance" << endl;
       }
     }
   }
  return is_success;

}
void initCustomerBalance() {

  /* initialize card balance....card no zero column card balance firt column*/
  customerBalance[0][0] = 1001;
  customerBalance[0][1] = 40000;

  customerBalance[1][0] = 1002;
  customerBalance[1][1] = 50000;

  customerBalance[2][0] = 1003;
  customerBalance[2][1] = 60000;

  customerBalance[3][0] = 1004;
  customerBalance[3][1] = 15000;

  customerBalance[4][0] = 1005;
  customerBalance[4][1] = 100;

  customerBalance[5][0] = 1006;
  customerBalance[5][1] = 20000;

  customerBalance[6][0] = 1007;
  customerBalance[6][1] = 800;

  customerBalance[7][0] = 1008;
  customerBalance[7][1] = 3000000;

}
void initCustomerPin() {

  /* initialize card one pin....card no zero column card pin firt column*/
  customerPin[0][0] = 1001;
  customerPin[0][1] = 1001;

  /* initialize card one pin*/
  customerPin[1][0] = 1002;
  customerPin[1][1] = 1002;

  /* initialize card three pin*/
  customerPin[2][0] = 1003;
  customerPin[2][1] = 1003;

  /* initialize card 4 pin*/
  customerPin[3][0] = 1004;
```

```
  customerPin[3][1] = 1004;

  /* initialize card 5 pin*/
  customerPin[4][0] = 1005;
  customerPin[4][1] = 1005;

  /* initialize card 6 pin*/
  customerPin[5][0] = 1006;
  customerPin[5][1] = 1006;

  /* initialize card 7 pin*/
  customerPin[6][0] = 1007;
  customerPin[6][1] = 1007;

  /* initialize card 8 pin*/
  customerPin[7][0] = 1008;
  customerPin[7][1] = 1008;


}

string getCustomerBank(int cardNumber) {

switch(cardNumber) {
 case 1001:
  return "HBL";
  break;
 case 1002:
  return "ALFALAH";
  break;
 case 1003:
  return "MCB";
  break;
 case 1004:
  return "ALLIED";
  break;
 case 1005:
  return "RAZI";
  break;
 case 1006:
  return "HBL";
  break;
 case 1007:
  return "HBL";
  break;
 case 1008:
  return "BANK AL HABIB";
  break;
 default:
  return "STATE BANK";
 }
```

```
 }

 int geStackTop(){
   int top = transctionInStack.getStackTop();
   return top;
 }

 void startTransaction(){
   int cardNumber, pin, amount;
   //operations/
   //first check Card is Valid/
   cout << "Please Enter Card Number:" << endl;
   cin >> cardNumber;

   bool is_card_valid = isCardValid(cardNumber);
   if (is_card_valid == false) {
     cout << "Invalid Card" << endl;
     return;
   }
   string customerBank = getCustomerBank(cardNumber);
   cout << "Using Depth First Serach to find " << " " << customerBank << " " << "availble
server(s) for tranacation.....Please wait" << endl;
   //using DFS to found right server for transaction...
   DepthFirstSearch objDFS(customerBank);
   objDFS.DFS();

   if (objDFS.isServerFound == true) {
     cout << customerBank << " " << "Server is setup for transaction" << endl;
   }else{
     cout << customerBank << " " << "Server is not availble. Please try later....." << endl;
     return;
   }

   cout << "Please Enter Card Pin:" << endl;
   cin >> pin;
   bool is_pin_valid = isPinValid(cardNumber, pin);
   if (is_pin_valid == false) {
     cout << "Invalid Pin" << endl;
     return;
   }

   cout << "Please Enter Amount to Withdraw :" << endl;
   cin >> amount;
   if (amount <= 0) {
     cout << "Invalid Amount" << endl;
     return;
   }

   //withdrawAmount Amount & update Customer Balance/
   bool is_withdraw = withdrawAmount(cardNumber, amount);
   if (is_withdraw == true) {
```

```
    transctionInStack.push(amount);     //transactions are pushed into Stack/
    objBST.insertBST(amount);
    transCount = transCount + 1;
    transacitons[transCount] = amount;

  }
   //customers are added into linked list/
  customerUsesMachine.addCustomer(cardNumber);
}

void sortTransactions() {

  if( transCount < 0){
     cout << "Transactional data is missing!" << endl;
     return;
  }

   cout <<"Without sorting Transactional data....." << endl;

  //getting all transacitons
  for (int k = 0; k <=transCount ; k++) {
     cout << transacitons[k] << endl;
  }

  //sorting via bubble sort

  int i,j,temp;
  for(i = 0; i<=transCount; i++) {
  for(j = i+1; j<=transCount; j++)
  {
    if(transacitons[j] < transacitons[i]) {
       temp = transacitons[i];
       transacitons[i] = transacitons[j];
       transacitons[j] = temp;
    }
  }

  }

   cout <<"Sorted Transactional data.....\n" << endl;
  for(i = 0; i<=transCount; i++) {
    cout <<transacitons[i]<<"\t";
  }



}



 void searchTransaction() {
```

```
    if( transCount < 0){
     cout << "Transactional data is missing!" << endl;
     return;
    }

    int transAmount;
    sortTransactions();
    cout << endl;
    cout << "Please Enter Transactional Amount to Search :" << endl;
    cin >> transAmount;

   int n = sizeof(transacitons)/ sizeof(transacitons[0]);
   int index = binarySearch (transacitons, 0, n-1, transAmount);
   if(index == -1){
      cout<< transAmount <<" is present in the Transactional Array";
   }else{
      cout<< transAmount <<" is present at index "<< index <<" in the Transactional Array";
   }

  }



  //Recursive method to find highhest tranacation from Stack/
  void highestTrasacation(int stackTop) {
    if (stackTop == -1) {
      return;
    }

    int amount = transctionInStack.pop();
    if (amount > highestTransactionAmount) {
      highestTransactionAmount = amount;
    }
    stackTop = stackTop - 1;
    highestTrasacation(stackTop);

  }


  void customerUsedATM() {
    customerUsesMachine.showCustomer();
  }


int binarySearch(int arr[], int start, int end, int key)
{
while (start <= end)
{
int mid = (start +end)/2;
// Check if x is present at mid
if (arr[mid] == key)
```

```
return mid;
// If x greater, ignore left half
if (arr[mid] < key)
start = mid + 1;
// If x is smaller, ignore right half
else
end = mid- 1;
}

return -1;
}



};

int main() {
  int ch;
  ATMMachine obj;
  cout << "1) to Start using ATM Machine" << endl;
  cout << "2) Display highest Transaction (if customer use at least once ATM Machine)" <<
endl;
  cout << "3) Customer list who used ATM Machine" << endl;
  cout << "4) Search a Transaction via Binary Search" << endl;
  cout << "5) Exit" << endl;

  do {
    cout << "Enter Operation....: " << endl;
    cin >> ch;
    switch (ch) {
    case 1: {
      obj.startTransaction();
      break;
    }
    case 2: {
      obj.highestTrasacation( obj.geStackTop() );
      cout << "Highest Transactional Amount is: " << obj.highestTransactionAmount << endl;
      break;
    }
    case 3: {
      obj.customerUsedATM();
      break;
    }
    case 4: {
      obj.searchTransaction();
      break;
    }
    case 5: {
      cout << "Exit" << endl;
      break;
    }
```

```
    default: {
      cout << "Invalid Choice" << endl;
    }
    }
  } while (ch != 6);
  return 0;
}
```

Conclusion:

The whole project is designed in c++. The mini project is easy to operate and understand by user. This simple project enhance the beginners to develop their programming skills. This system is a project which is used to access bank accounts in order to make cash withdraws. The ATM will service one customer at a time. User can perform one or more than one transaction.