LAB-7 Software Engineering
Program Inspection, Debugging and Static Analysis

Name: Bansi Patel
Student ID: 202201190

CODE:

Github Link:
https://github.com/girishkumarkh/OldCPPProjects/blob/master/C%2B%2B%20Project%20on%20Banking.cpp

```
//*****************************************************************
//PROJECT BANKING SYSTEM
//*****************************************************************
//INCLUDES HEADER FILES
//*****************************************************************

#include <iostream .h>
#include <fstream .h>
#include <process .h>
#include <string .h>
#include <stdio .h>
#include <ctype .h>
#include <conio .h>
#include <dos .h>
#include <stdlib .h>
#include <iomanip .h>
#include <graphics .h>


typedef char option[15];
```

```
const int ROW = 10,COL = 10;

int scan;    // To hold the special characters for moving the prompt in menu
int ascii;
//*****************************************************************
// To display the main menu options
//*****************************************************************
option a[]= {
                        "NewAccount",
                        "ListofAccounts",
                        "IndAccount",
                        "DailyTrans",
                        "MonthlyReport",
                        "EditAccount",
                        "Exit"};

// Displays the modify menu options
option b[] = {
                                "Modify Account",
                                "Closeaccount",
                                "Quit"
                        };

// Function used to do screening
class main_menu
{
        int i,done;

        public:
                void normalvideo(int x,int y,char *str);
                void reversevideo(int x,int y,char *str);
                void box(int x1,int y1,int x2,int y2);
                char menu();
                void control_menu();
                char e_menu();
                void edit_menu();
                void help(void);
};




//*****************************************************************
/* Class member functions for drawing boxes */
//*****************************************************************

class shape
{
        public:
                void line_hor(int, int, int, char);
```

```
                    void line_ver(int, int, int, char);
                    void box(int, int, int, int, char);
};


// Class contains the initial deposit of customers
class initial
{
 public:
    void add_to_file(int, char t_name[30], char t_address[30], float);
            // For initial deposits in customers account
    void display_list(void); // Displaying customers account list
    void delete_account(int); // Deleting customers account
    void update_balance(int, char t_name[30], char t_address[30], float);
            // For updating the customer account
    void modify(void);      // To modify the customer account information
    int last_accno(void); // To know the last account number
    int found_account(int);
            // To found the account is in "INITIAL.dat" or not
    char *return_name(int);
            // Function for validation entry of customer name
    char *return_address(int);
            // Function for validation entry of customer address
    float give_balance(int);
            // To print the balance amount of a particular customer
    int recordno(int);
    void display(int); // To display the customer account

 private:
    void modify_account(int, char t_name[30], char t_address[30]);
            // Function to modify the customer account
    int accno;
    char name[30], address[30];
    float balance;
};

// Class contains the customers daily transaction entry
class account
{
 public:
    void new_account(void);    // Function to create a new account
    void close_account(void);  // Function to close an account
    void display_account(void);// Function to display the accounts
    void transaction(void);    // To display the transaction process
    void clear(int, int);      // Function to perform a clear screen function
    void month_report(void);   // Function to list monthly transaction report

 private:
    void add_to_file(int,int,int,int,char,char t_type[10],float,float,float);
            // Function to add transaction records
```

```
    void delete_account(int);  // Function to delete a transaction record
    int no_of_days(int, int, int, int, int, int);
            // Function to find the total days
    float calculate_interest(int, float);
            // Function for calculating interest of an account
    void display(int);         // Function to display a transaction account
    void box_for_display(int); // Function for displaying box
    int accno;
    char type[10];             // Account type as Cheque or Cash
    int dd, mm, yy;            // To store the system date/ Enter date
    char tran;                 // As the account type is Deposit or Withdraw
    float interest, amount, balance;
};

// Function to displays all the menu prompt messages from the pointer array of
// option a[]
void main_menu::normalvideo(int x,int y,char *str)
{
    gotoxy(x,y);
    cprintf("%s",str);
}

// Function to move the cursor on the menu prompt with a reverse video color
void main_menu::reversevideo(int x,int y,char *str)
{
    textcolor(5+143);
    textbackground(WHITE);
    gotoxy(x,y);
    cprintf("%s",str);
    textcolor(GREEN);
    textbackground(BLACK);
}

void main_menu::box(int x1,int y1,int x2,int y2)
{
    for(int col=x1;col<x2 ;col++)
    {
     gotoxy(col,y1);
     cprintf("%c",196);
     gotoxy(col,y2);
     cprintf("%c",196);
     }

    for(int row=y1;row<y2;row++)
    {
     gotoxy(x1,row);
     cprintf("%c",179);
     gotoxy(x2,row);
     cprintf("%c",179);
     }
```

```
    gotoxy(x1,y1);
    cprintf("%c",218);
    gotoxy(x1,y2);
    cprintf("%c",192);
    gotoxy(x2,y1);
    cprintf("%c",191);
    gotoxy(x2,y2);
    cprintf("%c",217);
}

char main_menu::menu()
{
    clrscr();
    textcolor(22);
    box(20, 6, 65, 20);
    box(18, 4, 67, 22);
    textcolor(5+143);
    gotoxy(36, 5);
    textbackground(BLUE);
    cprintf("B A N K I N G");
    textbackground(BLACK);
    textcolor(22);
    for(i = 1; i < 7; i++)
    normalvideo(32, i+10, a[i]);
    reversevideo(32, 10, a[0]);
    i = done = 0;
    _setcursortype(_NOCURSOR);
    do
    {
     int key = getch();
     switch (key)
     {
      case 00:
                key = getch();
                switch (key)
                {
                    case 72:
                      normalvideo(32, i+10, a[i]);
                      i--;
                      if (i == -1)
                      i = 6;
                      reversevideo(32,i+10,a[i]);
                      break;
                    case 80:
                      normalvideo(32, i+10, a[i]);
                      i++;
                      if (i == 7)
                      i = 0;
                      reversevideo(32, i+10, a[i]);
                      break;
```

```
                    }
                break;
      case 13:
                done = 1;
      }
     }
    while (!done);
    _setcursortype(_NOCURSOR);
    return(i+49);
}

// The function main_menu() is used to display the main menu of banking system
void main_menu::control_menu()
{
    char choice;
    account a;
    do
    {
     choice = menu();
     clrscr();
     switch (choice)
     {
      case '1':
                    _setcursortype(_NORMALCURSOR);
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    a.new_account(); // New account member function
                    break;
      case '2':
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    initial ini;
                    ini.display_list(); // Global list of account function
                    break;
      case '3':
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    _setcursortype(_NORMALCURSOR);
                    a.display_account();
                    // Displaying individual accounts all transactions
                    break;
      case '4':
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    account a;
                    _setcursortype(_NORMALCURSOR);
                    a.transaction(); // Daily transaction for individual account
                    break;
      case '5':
                    box(3, 1, 75, 24);
```

```
                box(5, 2, 73, 23);
                _setcursortype(_NORMALCURSOR);
                a.month_report(); // Monthly report for any account
                break;
      case '6':
                box(3, 1, 75, 24);
                box(5, 2, 73, 23);
                gotoxy(10,10);
                edit_menu();// Sub menu for modifying or deleting any account
                break;
      case '7' :exit(0);
      }
     } while (choice != 6);
}

// This function is used to return the cursor position to the edit menu
// function where the menu prompt will valid
char main_menu::e_menu()
{
     clrscr();
     textcolor(22);
     box(25,6,60,15);
     box(23,4,62,17);
     textcolor(5+143);
     gotoxy(34,5);
     textbackground(GREEN);
     cprintf("E D I T  -  M E N U");
     textcolor(22);
     textbackground(BLACK);
     for (i = 1;i < 3; i++)
     normalvideo(32, i+10, b[i]);
     reversevideo(32, 10, b[0]);
     i = done = 0;
     _setcursortype(_NOCURSOR);
     do
     {
      int key = getch();
      switch (key)
      {
          case 00:
                key = getch();
                switch (key)
                {
                 case 72:
                   normalvideo(32, i+10, b[i]);
                   i--;
                   if (i == -1)
                   i = 2;
                   reversevideo(32, i+10, b[i]);
                   break;
```

```
            case 80:
                    normalvideo(32, i+10, b[i]);
                    i++;
                    if (i == 3)
                    i=0;
                    reversevideo(32, i+10, b[i]);
                    break;
            }
            break;
            case 13:
                    done = 1;
            }
      }while (!done);
      _setcursortype(_NOCURSOR);
      return(i+49);
}

/* Function for edit menu with account modification and close */
void main_menu::edit_menu()
{
      char choice;
      account a;
      do
      {
       choice = e_menu();
       clrscr();
       switch (choice)
       {
            case '1':
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    initial ini;
                    _setcursortype(_NORMALCURSOR);
                    ini.modify();
                    break;
            case '2':
                    box(3, 1, 75, 24);
                    box(5, 2, 73, 23);
                    account a;
                    _setcursortype(_NORMALCURSOR);
                    a.close_account();
                    break;
            case '3':
                    return;
            }
      } while (choice != 6);
}

/* Function to draw horizontal line */
void shape::line_hor(int column1, int column2, int row, char c)
```

```
{
     for (column1; column1 <= column2; column1++)
     {
      gotoxy(column1, row);
      cout << c;
      }
}

/* Function to draw vertical line */
void shape::line_ver(int row1, int row2, int column, char c)
{
     for (row1; row1 <= row2; row1++)
     {
      gotoxy(column, row1);
      cout << c;
      }
}

/* function for drawing box */
void shape::box(int column1, int row1, int column2, int row2, char c)
{
     char ch = 218;
     char c1, c2, c3, c4;
     char l1 = 196, l2 = 179;
     if (c == ch)
     {
      c1 = 218;
      c2 = 191;
      c3 = 217;
      c4 = 217;
      l1 = 196;
      l2 = 179;
      }
     else
     {
      c1 = c;
      c2 = c;
      c3 = c;
      c4 = c;
      l1 = c;
      c2 = c;
      }
     gotoxy(column1, row1);
     cout << c1;
     gotoxy(column2, row1);
     cout << c2;
     gotoxy(column1, row2);
     cout << c3;
     gotoxy(column2, row2);
     cout << c4;
```

```
    column1++;
    column2--;
    line_hor(column1, column2, row1, l1); //Horizontal line
    line_hor(column1, column2, row2, l1);
    column1--;
    column2++;
    row1++;
    row2--;
    line_ver(row1, row2, column1, l2); // Vertical line
    line_ver(row1, row2, column2, l2);
}


/* Function to display help about this project */
void main_menu::help(void)
{
    clrscr();
    setbkcolor(7);
    settextstyle(7,HORIZ_DIR,4);
    outtextxy(70,20,"Welcome to Banking System");
    settextstyle(2,HORIZ_DIR,5);
    outtextxy(60,100,"You can keep record of daily banking transaction");
    delay(2);
    outtextxy(60,130,"This program is capable of holding any no. of A/c");
    delay(2);
    outtextxy(60,160,"-In first option you can open new A/c");
    delay(2);
    outtextxy(60,190,"-In second option you can see the list of all A/c's");
    delay(2);
    outtextxy(60,220,"-In third option you can see all trans. of ind. A/c");
    delay(2);
    outtextxy(60,250,"-In fourth optiion you can do banking transactions");
    delay(2);
    outtextxy(60,280,"(Deposit/Withdraw)");
    delay(2);
    outtextxy(60,310,"-In fifth opt. you can take monthly ind. A/c report");
    delay(2);
    outtextxy(60,340,"-In sixth opt. you can modify or delete any account");
    delay(2);
    outtextxy(60,370,"Note-:Opening amount should not less that Rs. 500/-");
    delay(2);
    outtextxy(60,400,"-And last option is Quit (Exit to Window)");
    delay(2);
    settextstyle(7,HORIZ_DIR,4);
    outtextxy(80,420,"Press any key to continue...");
    getch();
}

/* Function for modifying the existing accounts */
void initial::modify(void)
```

```
{
     clrscr();
     int j;
     char t_acc[10];
     int t, t_accno;
     gotoxy(17, 1);
     cout << "&lt;0>=Exit";
     gotoxy(5,5);
     cout < < "Enter the account no. ";
     gets(t_acc);
     t = atoi(t_acc);
     t_accno = t;
     if (t_accno == 0)
     return;
     clrscr();
     if (!found_account(t_accno))
     {
      gotoxy(5, 5);
      cout << "\7Account not found";
      getch();
      return;
      }
     gotoxy(71, 1);
     cout << "&lt;0>=Exit";
     textbackground(WHITE);
     gotoxy(3, 3);
     for (j = 1; j< = 76; j++)
     cprintf(" ");
     textbackground(BLACK);
     textcolor(BLACK+BLINK);
     textbackground(WHITE);
     gotoxy(30, 3);
     cprintf("Modify Account Screen");
     textcolor(LIGHTGRAY);
     textbackground(BLACK);
     int d1, m1, y1;
     struct date d;          // For extracting system date
     getdate(&d);
     d1 = d.da_day;
     m1 = d.da_mon;
     y1 = d.da_year;
     gotoxy(4, 2);
     cout << "Date: " << d1 << "/" << m1 << "/" << y1;
     char ch;
     display(t_accno);
     account a;
     do
     {
      a.clear(5, 13);
      gotoxy(5, 13);
```

```
 cout << "Modify this account <y/n>: ";
 ch = getche();
 if (ch == '0')
 return;
 ch = toupper(ch);
 }while (ch != 'N' && ch != 'Y');
if (ch == 'N')
return;
int modified = 0, valid;
char t_name[30], t_address[30];
gotoxy(5, 15);
cout < < "Name : ";
gotoxy(5, 16);
cout << "Address : ";
do
{
 a.clear(15, 15);
 a.clear(5, 23);
 gotoxy(5, 23);
 cout << "Enter Name or Press Enter for No Change";
 valid = 1;
 gotoxy(15, 15);
 gets(t_name);
 strupr(t_name);
 if (t_name[0] == '0')
 return;
 if (strlen(t_name) > 25)
 {
     valid = 0;
     gotoxy(5, 23);
     cprintf("\7Name should not greater than 25");
     getch();
     }
 } while (!valid);
do
{
 a.clear(15, 16);
 a.clear(5, 23);
 gotoxy(5, 23);
 cout < < "Enter Address or press enter for no Change";
 valid = 1;
 gotoxy(15, 16);
 gets(t_address);
 strupr(t_address);
 if (t_address[0] == '0')
 return;
 if (strlen(t_address) > 25)
 {
     valid = 0;
     gotoxy(5, 23);
```

```
        cprintf("\7Address should not greater than 25");
        getch();
        }
     }while (!valid);
    if (strlen(t_address) > 0)
    modified = 1;
    if (!modified)
    return;
    // clears the screen at 23rd row and from 5th column
    a.clear(5,23);
    do
    {
     a.clear(5, 23);
     gotoxy(5, 18);
     cout < < "Do you want to save Changes <Y/N>: ";
     ch = getche();
     if (ch == '0')
     return;
     ch = toupper(ch);
     }while (ch != 'N' && ch != 'Y');
    if (ch == 'N')
    return;
    // Passes the parameter to add in data file
    modify_account(t_accno, t_name, t_address);
    gotoxy(5, 21);
    cout < < "\7Record modified";
    gotoxy(5, 23);
    cout << "Press any key to continue...";
    getch();
}

/* Function for displaying an account when modified */
void initial::display(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    // Displays the record contents matching with t_accno from
    // INITIAL.dat data file
    while (file.read((char *)this, sizeof(initial)))
    {
     if (t_accno == accno)
     {
        gotoxy(8, 5);
        cout << "Account no. " << accno;
        gotoxy(10, 8);
        cout << "Name : ";
        puts(name);
        gotoxy(10, 9);
        cout << "Address : ";
```

```
            puts(address);
            gotoxy(10, 10);
            cout << "Balance : " << setw(15)    // setwidth
            << setprecision(2)    // set position of decimal point
            << setiosflags(ios::left)   // set left justified output
            << setiosflags(ios::showpoint)  // always show decimal point
            << setiosflags(ios::fixed)<< balance;// set fixed notation for display
            break;
            }
       }
      file.close();
}

/* Function for updating the modified account into INITIAL.dat file */
void initial::modify_account(int t_accno,char t_name[30],char t_address[30])
{
    int recno;
    recno = recordno(t_accno);
    fstream file;
    file.open("INITIAL.dat", ios::out|ios::ate);
    strcpy(name, t_name);
    strcpy(address, t_address);
    int location;
    // finds the position in data file
    location = (recno-1) * sizeof(initial);
    file.seekp(location);
    // Overwrites the modified record into INITIAL.dat data file
    file.write((char *)this, sizeof(initial));
    file.close();
    return;
}

/* Function to find the last account number */
int initial::last_accno(void)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    int count = 0;
    // Founds the last account no.
    while (file.read((char *)this, sizeof(initial)))
    count = accno;
    file.close();
    return count;
}

//This function add_to_file() is used to create new/fresh record in data file
void initial::add_to_file(int t_accno,char t_name[30],char t_address[30],float t_balance)
{
    accno = t_accno;
```

```
    strcpy(name, t_name);
    strcpy(address, t_address);
    balance = t_balance;
    fstream file;
    // Appends new account record with the balance into INITIAL.dat data file
    file.open("INITIAL.dat", ios::out|ios::app);
    file.write((char *)this, sizeof(initial));
    file.close();
}

// Function for deleting a account from INITIAL.dat file
void initial::delete_account(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    fstream temp;
    temp.open("TEMP.dat", ios::out);
    file.seekg(0,ios::beg);
    // Uses a copy method to delete the account from INTITAL.dat data file
    while (!file.eof())
    {
     file.read((char *)this, sizeof(initial));
     if (file.eof())
     break;
     if (accno != t_accno)
     temp.write((char *)this, sizeof(initial));
     }
    file.close();
    temp.close();
    file.open("INITIAL.dat", ios::out);
    temp.open("TEMP.dat", ios::in);
    temp.seekg(0, ios::beg);
    // Copy the TEMP.dat contents into INTITAL.dat data file
    while (!temp.eof())
    {
     temp.read((char *)this, sizeof(initial));
     if (temp.eof())
     break;
     if (accno != t_accno)
     file.write((char *)this, sizeof(initial));
     }
    file.close();
    temp.close();
}

// Function for adding account details of daily tranaction into BANKING.dat file

void account::add_to_file(int t_accno,int d1,int m1,int y1,char t_tran,char t_type[10],float
t_interest,float t_amount,float t_balance)
{
```

```
    fstream file;
    file.open("BANKING.dat", ios::app);
    accno = t_accno;
    getch();
    dd = d1;
    mm = m1;
    yy = y1;
    tran = t_tran;
    strcpy(type, t_type);
    interest = t_interest;
    amount = t_amount;
    balance = t_balance;
    // Appends the transaction record into BANKING.dat data file
    file.write((char *)this, sizeof(account));
    file.close();
}

/* Function for deleting an account from BANKING.dat file. */
void account::delete_account(int t_accno)
{
    fstream file;
    file.open("BANKING.dat", ios::in); // Open to read records
    fstream temp;
    temp.open("TEMP.dat", ios::out); // Open to write records
    file.seekg(0, ios::beg);        // Positioned from begining of the file
    // Uses the copy method for deleting the transaction record from
    // BANKING.dat data file
    while (!file.eof())
    {
     file.read((char *)this, sizeof(account));
     if (file.eof())
     break;
     if (accno != t_accno)
     temp.write((char *)this, sizeof(account));
     }
    file.close();
    temp.close();
    file.open("BANKING.dat", ios::out);
    temp.open("TEMP.dat", ios::in);
    temp.seekg(0, ios::beg);
    // Uses copy method to transfer the record from TEMP.dat file to
    // BANKING.dat data file
    while (!temp.eof())
    {
     temp.read((char *)this, sizeof(account));
     if (temp.eof())
     break;
     if (accno != t_accno)
     file.write((char *)this, sizeof(account));
     }
```

```
    file.close();
    temp.close();
}

/* Function for displaying an account from "INITIAL.dat". */
void initial::display_list(void)
{
    clrscr();
    int flag;
    float t_bal = 0.0;
    fstream file;
    gotoxy(25,2);
    cout << "Accounts List in Bank";
    gotoxy(25, 3);
    cout << "======================";
    int d1, m1, y1;
    struct date d;          // For extracting system date
    getdate(&d);
    d1 = d.da_day;
    m1 = d.da_mon;
    y1 = d.da_year;
    gotoxy(62, 3);
    cout << "Date: " << d1 << "/" << m1 << "/" << y1;
    gotoxy(1, 4);
    for (int j = 1; j <= 79; j++)
    cout << "=";
    gotoxy(1, 5);
    cout << "Accno#";
    gotoxy(10,5);
    cout << "Name";
    gotoxy(30,5);
    cout << "Address";
    gotoxy(65,5);
    cout << "Balance";
    gotoxy(1, 6);
    for (j = 1; j <= 79; j++)
    cout << "=";
    file.open("INITIAL.dat", ios::in);
    file.seekg(0,ios::beg);
    int row = 7;
    // Reads all the records to display on the screen
    while (file.read((char *)this, sizeof(initial)))
    {
     flag = 0;
     delay(2);
     gotoxy(3, row);
     cout << accno;
     gotoxy(10, row);
     puts(name);
     gotoxy(30, row);
```

```
      puts(address);
      gotoxy(65, row);
      cout <<setw(15)<<setprecision(2)<<setiosflags(ios::left)
              <<setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<<balance;
      t_bal = t_bal + balance;
      row++;
      if (row > 23)
      {
       flag = 1;
       row = 6;
       gotoxy(4, 24);
       cout < < "Press any key to continue.... ";
       getch();
       clrscr();
      }
     }
    gotoxy(1, row);
    for (j = 1; j <= 79; j++)
    cout << "=";
    row++;
    gotoxy(3, row);
    cout << "Total Balance in Bank is : ";
    gotoxy(65, row);
    cout <<setw(15)<<setprecision(2)<<setiosflags(ios::left)
            <<setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<<t_bal;
    file.close();
    if (!flag)
    {
     gotoxy(4, 24);
     cout << "Press any key to continue...";
     getch();
     }
}

/* Function for clearing specified row and column. */
void account::clear(int col, int row)
{
    for (int j = col; j <= 79; j++)
    {
     gotoxy(j, row);
     cout << " ";
     }
}

/* Function to found an account for display account function. */
int initial::found_account(int t_accno)
{
     fstream file;
     file.open("INITIAL.dat", ios::in);
     file.seekg(0, ios::beg);
```

```
    int found = 0;
    // Searches the specified record in INITIAL.dat data file
    while (file.read((char *)this, sizeof(initial)))
    {
     if (accno == t_accno)
     {
      found = 1;
      break;
      }
     }
    file.close();
    return found;
}

/* Function for return name of the account holder from INITIAL.dat. */
char *initial::return_name(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    char t_name[30];
    // Return the name to display at report screen if found
    while (file.read((char *)this, sizeof(initial)))
    {
     if (accno == t_accno)
     {
      strcpy(t_name, name);
      break;
      }
     }
    file.close();
    return t_name;
}

/* Function for return address of the account holder from INITIAL.dat. */
char *initial::return_address(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    char t_address[30];
    // Return the address to display at report screen if found
    while (file.read((char *)this, sizeof(initial)))
    {
     if (accno == t_accno)
     {
      strcpy(t_address, address);
      break;
      }
     }
```

```
    file.close();
    return t_address;
}

/* Function for display account details */
void account::box_for_display(int t_accno)
{
    int d1, m1, y1;
    struct date d;
    getdate(&d);
    d1 = d.da_day;
    m1 = d.da_mon;
    y1 = d.da_year;
    gotoxy(63, 2);
    cout << "Date: " << d1 << "/" << m1 << "/" << y1;
    gotoxy(4, 2);
    cout << "Account No. " << t_accno;
    initial ini;
    char t_name[30];
    strcpy(t_name, ini.return_name(t_accno));
    char t_address[30];
    strcpy(t_address, ini.return_address(t_accno));
    gotoxy(25, 2);
    cout << t_name;
    gotoxy(25, 3);
    cout << t_address;
    gotoxy(4, 5);
    cout << "Global Report of Account";
    textbackground(WHITE);
    textcolor(BLACK);
    textbackground(WHITE);
    gotoxy(1, 6);
    for (int i = 1; i <=79; i++)
    cout << "=";
    gotoxy(4, 7);
    cprintf("Date      Particular   Deposit      Withdraw              Balance");
    gotoxy(1, 8);
    for (i = 1; i <=79; i++)
    cout << "=";
    textcolor(LIGHTGRAY);
    textbackground(BLACK);
}

/* Function for display an account from BANKING.dat file. */
void account::display_account(void)
{
    clrscr();
    char t_acc[10];
    int j;
    int tamt = 0, damt = 0, wamt = 0;
```

```
int t, t_accno;
gotoxy(71, 1);
cout << "&lt;0>=Exit";
gotoxy(5, 5);
cout < < "Enter account no. ";
gets(t_acc);
t = atoi(t_acc);
t_accno = t;
if (t_accno == 0)
return;
clrscr();
initial ini;
if (!ini.found_account(t_accno))
{
 gotoxy(5, 5);
 cout << "\7Account not found";
 getch();
 return;
 }
// Display the heading from this function
box_for_display(t_accno);
int row = 9, flag;
fstream file;
file.open("BANKING.dat", ios::in);
while (file.read((char *)this, sizeof(account)))
{
 if (accno == t_accno)
 {
  flag = 0;
  delay(2);
  gotoxy(4, row);
  cout << dd << "-" << mm << "-" << yy;
  gotoxy(16, row);
  puts(type);
  if (tran == 'D')
  {
      damt = damt + amount;
      tamt = tamt + amount;
      gotoxy(30, row);
      }
  else
  {
      wamt = wamt + amount;
      tamt = tamt - amount;
      gotoxy(42, row);
      }
  cout <<  setw(15)<< setprecision(2)<< setiosflags(ios::left)
          << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<< amount;
  gotoxy(66, row);
  cout << setw(15)<< setprecision(2)<< setiosflags(ios::left)
```

```
            << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<<balance;
      row++;
      if (row > 23)
      {
          flag = 1;
          row = 7;
          gotoxy(4, 24);
          cout < < "Press any key to continue";
          getch();
          clrscr();
          box_for_display(t_accno);
          }
      }
     }
     file.close();
     gotoxy(1, row);
     for (j = 1; j <= 79; j++)
     cout << "=";
     row++;
     gotoxy(4, row);
     cout << "Total-->:";
     gotoxy(30, row);
     cout < < setw(15)<< setprecision(2)<< setiosflags(ios::left)
            << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<< damt;
     gotoxy(42, row);
     cout << setw(15)<< setprecision(2)<< setiosflags(ios::left)
            << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<< wamt;
     gotoxy(66, row);
     cout << setw(15)<< setprecision(2)<< setiosflags(ios::left)
            << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<< tamt;
     if (!flag)
     {
      gotoxy(4, 24);
      cout << "Press any key to continue...";
      getch();
      }
}

/* Function to list monthWise transaction report. */
void account::month_report(void)
{
    int dd1, mm1, yy1;
    clrscr();
    gotoxy(10, 5);
    cout << "Enter any date of a month ";
    gotoxy(38, 5);
    cin >> dd1;
    gotoxy(40, 5);
    cout < < "-";
    gotoxy(41, 5);
```

```
cin >> mm1;
gotoxy(43, 5);
cout < < "-";
gotoxy(44, 5);
cin >> yy1;
clrscr();
char t_acc[10];
int j;
int tamt = 0, damt = 0, wamt = 0;
int t, t_accno;
gotoxy(71, 1);
cout < < "&lt;0>=Exit";
gotoxy(5, 5);
cout < < "Enter account no. ";
gets(t_acc);
t = atoi(t_acc);
t_accno = t;
if (t_accno == 0)
return;
clrscr();
initial ini;
if (!ini.found_account(t_accno))
{
 gotoxy(5, 5);
 cout << "\7Account not found";
 getch();
 return;
 }
box_for_display(t_accno);
gotoxy(4, 5);
cout << "Statement Month: " << dd1 << "/" << mm1 << "/" << yy1;
getch();
int row = 9, flag;
fstream file;
file.open("BANKING.dat", ios::in);
float pre_balance = 0.0; // Previous balance amount
// The loop finds the last months balance
while (file.read((char *)this, sizeof(account)))
{
//Checks the account no. and till the previous month and till current year
 if((accno == t_accno) && ((mm < mm1 && yy <= yy1) || (mm1 < mm && yy < yy1)))
 {
  pre_balance = balance;
  }
 }
 file.close();
 file.open("BANKING.dat", ios::in);
 gotoxy(54, row);
 cout<<"B/F ....    " <<setw(15)<<setprecision(2)<<setiosflags(ios::left)
        <<setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<<pre_balance;
```

```
row++;
// The loop displays the current months transaction after previous month
while (file.read((char *)this, sizeof(account)))
{
 if ((accno == t_accno) && (mm1 == mm && yy1 <= yy))
 {
  flag = 0;
  delay(2);
  gotoxy(4, row);
  cout << dd << "-" << mm << "-" << yy;
  gotoxy(16, row);
  puts(type);
  if (tran == 'D')
  {
      damt = damt + amount;
      tamt = tamt + amount;
      gotoxy(30, row);
      }
  else
  {
      wamt = wamt + amount;
      tamt = tamt - amount;
      gotoxy(42, row);
      }
  cout <<setw(15)<<setprecision(2)<<setiosflags(ios::left)
          <<setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<<amount;
  gotoxy(66, row);
  cout <<setw(15)<<setprecision(2)<<setiosflags(ios::left)
          <<setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<<balance;
  row++;
  // If row increases 23 then the next screen continues
  if (row > 23)
  {
      flag = 1;
      row = 7;
      gotoxy(4, 24);
      cout < < "Press any key to continue";
      getch();
      clrscr();
      box_for_display(t_accno);
      }
  }
 }
file.close();
gotoxy(1, row);
for (j = 1; j <= 79; j++)
cout << "=";
row++;
gotoxy(4, row);
cout << "Total-->:";
```

```
    gotoxy(30, row);
    // Deposited amount
    cout < < setw(15)                  // setwidth
            << setprecision(2)           // set position of decimal point
            << setiosflags(ios::left)     // set left justified output
            << setiosflags(ios::showpoint)// always show decimal point
            << setiosflags(ios::fixed)    // set fixed notation for display
            << damt;
    gotoxy(42, row);
    // Withdraw amount
    cout << setw(15)<< setprecision(2)<< setiosflags(ios::left)
            << setiosflags(ios::showpoint)<<setiosflags(ios::fixed)<< wamt;
    gotoxy(66, row);
    tamt = tamt + pre_balance;
    // Balance amount
    cout << setw(15)<< setprecision(2)<< setiosflags(ios::left)
            << setiosflags(ios::showpoint)<< setiosflags(ios::fixed)<< tamt;
    if (!flag)
    {
     gotoxy(4, 24);
     cout << "Press any key to continue...";
     getch();
     }
 }

/* Function for creating new account for new customer. */
void account::new_account(void)
{
   char ch;
   int i, valid;
   clrscr();
   initial ini;
   shape s;
   s.box(2, 1, 79, 25, 218);
   s.box(25, 2, 54, 4, 219);
   gotoxy(65, 2);
   cout << "&lt;0>=Exit";
   gotoxy(3,3);
   for (i = 1; i< = 76; i++)
   cprintf(" ");
   textbackground(BLACK);
   textcolor(BLACK+BLINK);
   textbackground(WHITE);
   gotoxy(30, 3);
   cprintf("Open New Account");
   textcolor(LIGHTGRAY);
   textbackground(BLACK);
   int d1, m1, y1;
   struct date d;          // For extracting system date
   getdate(&d);
```

```
d1 = d.da_day;
m1 = d.da_mon;
y1 = d.da_year;
int t_accno;
t_accno = ini.last_accno();
t_accno++;
//Appends and deletes false record to create primary position in data files
if (t_accno == 1)
{
 ini.add_to_file(t_accno, "abc", "xyz", 1.1);
 ini.delete_account(t_accno);
 cout << "Press xxxxxxx";
 getch();
 add_to_file(t_accno, 1, 1, 1997, 'D', "INITIAL", 1.1, 1.1, 1.1);
 delete_account(t_accno);
 }
char t_name[30], t[10], t_address[30];
float t_bal = 0.0, t_balance = 0.0;
gotoxy(5, 6);
cout << "Date: " << d1 << '/' << m1 << '/' << y1;
gotoxy(5, 8);
cout << "Account No # " << t_accno;
gotoxy(5, 10);
cout << "Name : ";
gotoxy(5, 11);
cout << "Address : ";
gotoxy(5, 12);
cout << "Name of verifying Person : ";
gotoxy(5, 14);
cout << "Initial Deposit : ";
do
{
 clear(15, 10);
 clear(5, 23);
 gotoxy(5, 23);
 cout << "Enter Name of the Person";
 valid = 1;
 gotoxy(15, 10);
 gets(t_name);
 strupr(t_name);
 if (t_name[0] == '0')
 return;
 if (strlen(t_name) == 0 || strlen(t_name) > 25)
 {
  valid = 0;
  gotoxy(5, 23);
  cprintf("\7Name should not greater than 25");
  getch();
  }
 }while (!valid);
```

```
do
{
 clear(25, 15);
 clear(5, 23);
 gotoxy(5, 23);
 cout < < "Enter Address of the Person ";
 valid = 1;
 gotoxy(15, 11);
 gets(t_address);
 strupr(t_address);
 if (t_address[0] == '0')
 return;
 if (strlen(t_address) == 0 || strlen(t_address) > 25)
 {
  valid = 0;
  gotoxy(5, 23);
  cprintf("\7Address should not greater than 25");
  getch();
  }
 }while (!valid);
do
{
 char vari[30];
 clear(13, 12);
 clear(5, 23);
 gotoxy(5, 23);
 cout < < "Enter name of the verifying Person ";
 valid = 1;
 gotoxy(31, 12);
 gets(vari);
 strupr(vari);
 if (vari[0] == '0')
 return;
 if (strlen(vari) == 0 || strlen(vari) > 25)
 {
  valid = 0;
  gotoxy(5, 23);
  cprintf("Should not blank or greater than 25");
  getch();
  }
 }while (!valid);
do
{
 clear(13, 12);
 clear(5, 23);
 gotoxy(5, 23);
 cout < < "Enter initial amount to be deposit ";
 valid = 1;
 gotoxy(23, 14);
 gets(t);
```

```
     t_bal = atof(t);
     t_balance = t_bal;
     if (t[0] == '0')
     {
      valid = 0;
      gotoxy(5, 23);
      cprintf("\7Should not less than 500");
      getch();
      }
     }while (!valid);
    clear(5, 23);
    do
    {
     clear(5, 17);
     valid = 1;
     gotoxy(5, 17);
     cout << "Do you want to save the record <Y/N>: ";
     ch = getche();
     if (ch == '0')
     return;
     ch = toupper(ch);
     }while (ch != 'N' && ch != 'Y');
    if (ch == 'N')
    return;
    float t_amount, t_interest;
    t_amount = t_balance;
    t_interest = 0.0;
    char t_tran, t_type[10];
    t_tran = 'D';
    strcpy(t_type, "INITIAL");
    //Appends records contents into both INITIAL.dat and BANKING.dat data files
    ini.add_to_file(t_accno, t_name, t_address, t_balance);
    add_to_file(t_accno, d1, m1, y1, t_tran, t_type, t_interest, t_amount, t_balance);
}

/* Function for returning balance amount of an account. */
float initial::give_balance(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    float t_balance;
    // Gives the last balance of an individual account
    while (file.read((char *)this, sizeof(initial)))
    {
     if (accno == t_accno)
     {
      t_balance = balance;
      break;
      }
```

```
    }
    file.close();
    return t_balance;
}

/* Function for returning the record no. for updating balance */
int initial::recordno(int t_accno)
{
    fstream file;
    file.open("INITIAL.dat", ios::in);
    file.seekg(0, ios::beg);
    int count = 0;

    // Finds the record position in INITIAL.dat data file
    while (file.read((char *)this, sizeof(initial)))
    {
     count++;
     if (t_accno == accno)
     break;
     }
    file.close();
    return count;
}

/* Function for updating the balance for the given account no. */
void initial::update_balance(int t_accno,char t_name[30],char t_address[30],float t_balance)
{
    int recno;
    recno = recordno(t_accno);
    fstream file;
    file.open("INITIAL.dat", ios::out|ios::ate);
    strcpy(name, t_name);
    strcpy(address, t_address);
    balance = t_balance;
    int location;
    location = (recno-1) * sizeof(initial); // Find the location in file
    file.seekp(location);   // Searches the insertion position in data file
    // Updates the balance amount in INITIAL.dat data file
    file.write((char *)this, sizeof(initial));
    file.close();
}

/* Function to return no. days between two dates. */
int account::no_of_days(int d1, int m1, int y1, int d2, int m2, int y2)
{
    static int month[] = {31, 28, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30};
    int days = 0;
   while (d1 != d2 || m1 != m2 || y1 != y2)
    {
     days++;
```

```
    d1++;
    if (d1 > month[m1-1])
    {
     d1 = 1;
     m1++;
     }
    if (m1 > m2)
    {
     m1 = 1;
     y1++;
     }
    }
   return days;
}

/* Function for calculates interest */
float account::calculate_interest(int t_accno, float t_balance)
{
   fstream file;
   file.open("BANKING.dat", ios::in);
   file.seekg(0, ios::beg);
   int d1, m1, y1, days;
   while (file.read((char *)this, sizeof(account)))
   {
    if (accno == t_accno)
    {
     d1 = dd;
     m1 = mm;
     y1 = yy;
     break;
     }
    }
   int d2, m2, y2;
   struct date d;
   getdate(&d);
   d2 = d.da_day;
   m2 = d.da_mon;
   y2 = d.da_year;
   float t_interest = 0.0;
   if((y2 < y1) || (y2==y1 && m2 < m1) || (y2==y1 && m2 == m1) && (d2 < d1))
   return t_interest;
   days = no_of_days(d1, m1, y1, d2, m2, y2);
   int months = 0;
   if (days > 30)
   {
    months = days / 30;
    t_interest = ((t_balance*2)/100 * months);
    }
   file.close();
   return t_interest;
```

```
}

/* Function for making daily transaction (Deposit 'D'/Withdraw 'W'. */
void account::transaction(void)
{
    clrscr();
    char t_acc[10];
    int t, t_accno, valid;
    gotoxy(71,1);
    cout < < "&lt;0>=Exit";
    gotoxy(5, 5);
    cout < < "Enter the account no. ";
    gets(t_acc);
    t = atoi(t_acc);
    t_accno = t;
    if (t_accno == 0)
    return;
    clrscr();
    initial ini;
    if (!ini.found_account(t_accno))
    {
     gotoxy(5, 5);
     cout << "\7Account not found";
     getch();
     return;
     }
    gotoxy(71, 1);
    cout << "&lt;0>=Exit";
    gotoxy(3, 3);
    for (int i = 1; i < = 76; i++)
    cprintf(" ");
    textbackground(BLACK);
    textcolor(BLACK+BLINK);
    textbackground(WHITE);
    gotoxy(29, 3);
    cprintf ("Transaction in Account");
    textcolor(LIGHTGRAY);
    textbackground(BLACK);
    int d1, m1, y1;
    struct date d;
    getdate(&d);
    d1 = d.da_day;
    m1 = d.da_mon;
    y1 = d.da_year;
    gotoxy(5, 6);
    cout << "Date: " << d1 << "/" << m1 << "/" << y1;
    gotoxy(5, 8);
    cout << "Accnount no. " << t_accno;
    char t_name[30];
    char t_address[30];
```

```
float t_balance;
strcpy(t_name, ini.return_name(t_accno));
strcpy(t_address, ini.return_address(t_accno));
t_balance = ini.give_balance(t_accno);
gotoxy(27, 11);
cout << "Name : " << t_name;
gotoxy(27, 12);
cout << "Address : " << t_address;
gotoxy(5, 15);
cout<<"Last balance Rs. "<< setw(15)<< setprecision(2)
      <<setiosflags(ios::left)<<setiosflags(ios::showpoint)
      <<setiosflags(ios::fixed)<<t_balance;
char t_tran, t_type[10], tm[10];
float t_amount, t_amt;
do
{
 clear(5, 10);
 valid = 1;
 gotoxy(5, 10);
 cout << "Deposit or Withdraw (D/W) : ";
 t_tran = getch();
 if (t_tran == '0')
 return;
 t_tran = toupper(t_tran);
 }while (t_tran != 'D' && t_tran != 'W');
do
{
 clear(5, 19);
 clear(5, 23);
 gotoxy(5, 23);
 cout << "Enter Transaction by Cash or Cheque ";
 valid = 1;
 gotoxy(5, 19);
 cout << "Cash/Cheque : ";
 gets(t_type);
 strupr(t_type);
 if (t_type[0] == '0')
 return;
 if (strcmp(t_type, "CASH") && strcmp(t_type, "CHEQUE"))
 {
  valid = 0;
  gotoxy(5, 23);
  cprintf("\7Enter correctly");
  getch();
  }
 }while (!valid);
do
{
 clear(5, 21);
 clear(5, 23);
```

```
     gotoxy(5, 23);
     cout << "Enter Amount for Transaction ";
     valid = 1;
     gotoxy(5, 21);
     cout << "Amount Rs. ";
     gets(tm);
     t_amt = atof(tm);
     t_amount = t_amt;
     if (tm[0] == '0')
     return;
     if ((t_tran == 'W' && t_amount > t_balance) || (t_amount < 1))
     {
      valid = 0;
      gotoxy(5, 23);
      cprintf("\7Invalid Data entered");
      getch();
      }
     }while (!valid);
    char ch;
    clear(5, 23);
    do
    {
     clear(20, 23);
     valid = 1;
     gotoxy(40, 20);
     cout << "Save Transaction <Y/N> : ";
     ch = getche();
     if (ch == '0')
     return;
     ch = toupper(ch);
     }while (ch != 'N' && ch != 'Y');
    if (ch == 'N')
    return;
    float t_interest;
    t_interest = calculate_interest(t_accno, t_balance);
    if (t_tran == 'D')
    t_balance = t_balance + t_amount + t_interest;
    else
    t_balance = (t_balance - t_amount) + t_interest;
    // Modified records are updated in data bases.
    ini.update_balance(t_accno, t_name, t_address, t_balance);
    add_to_file(t_accno,d1,m1,y1,t_tran,t_type,t_interest,t_amount,t_balance);
}

/* Function for closing any account after inputing account number. */
void account::close_account(void)
{
    clrscr();
    char t_acc[10];
    int t, t_accno;
```

```
gotoxy(71, 1);
cout < < "&lt;0>=Exit";
gotoxy(5, 5);
cout < < "Enter the account no. ";
gets(t_acc);
t = atoi(t_acc);
t_accno = t;
if (t_accno == 0)
return;
clrscr();
initial ini;
if (!ini.found_account(t_accno))
{
 gotoxy(5, 5);
 cout << "\7Account not found ";
 getch();
 return;
 }
gotoxy(71, 1);
cout << "&lt;0>=Exit";
gotoxy(3, 3);
textbackground(WHITE);
for (int i = 1; i < = 76; i++)
cprintf(" ");
textbackground(BLACK);
textcolor(BLACK+BLINK);
textbackground(WHITE);
gotoxy(30, 3);
cprintf("Close account screen");
textcolor(LIGHTGRAY);
textbackground(BLACK);
int d1, m1, y1;
struct date d;
getdate(&d);
d1 = d.da_day;
m1 = d.da_mon;
y1 = d.da_year;
gotoxy(5, 6);
cout << "Date: " << d1 << "/" << m1 << "/" << y1;
char ch;
ini. display(t_accno);
do
{
 clear(5, 15);
 gotoxy(5, 15);
 cout << "Close this account <y/n> ";
 ch = getche();
 if (ch == '0')
 return;
 ch = toupper(ch);
```

```
    }while (ch != 'N' && ch != 'Y');
   if (ch == 'N')
   return;
   // Function calls to delete the existing account no.
   ini.delete_account(t_accno);
   delete_account(t_accno);
   gotoxy(5, 20);
   cout < < "\7Account Deleted";
   gotoxy(5, 23);
   cout << "Press any key to continue...";
   getch();
}

// Main program logic which control the class members and member functions.
void main(void)
{
   main_menu  m_menu;
   int gdriver = DETECT, gmode, errorcode;
   initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
   m_menu.help();
   closegraph();
   m_menu.control_menu();
}
```

I. PROGRAM INSPECTION:

Category A: Data Reference Errors

1.Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to "prove" informally that the item has a value at that point.

```
public class Account {

private String accountNumber;

private double balance; // Uninitialized

public Account(String accountNumber) {

this.accountNumber = accountNumber; // Initializes accountNumbe

// balance is uninitialized here

}
```

```
public void deposit(double amount) {

balance += amount; // potential issue if balance is uninitialized

}

  }
```

- The variable balance is declared but not explicitly initialized in the constructor. Although Java initializes it to 0.0 by default, it is good practice to initialize it explicitly. Relying on defaults can lead to misunderstandings in code maintenance.

```
public class Customer {

private String name;

private Account account;

public Customer(String name) {

this.name = name; // Initialized  account is uninitialized

}

public void createAccount(String accountNumber) {

account = new Account(accountNumber); // Initializes account

}

  }
```

- The variable account is declared but not initialized in the constructor. It is initialized later in the createAccount method. If any method that uses account is called before createAccount, it will lead to a NullPointerException.

2.For all array references, is each subscript value within the defined bounds of the corresponding dimension?

```
public void depositMultiple(double[] amounts) {

        for (int i = 0; i <= amounts.length; i++) { // Potential off-by-one error

        balance += amounts[i]; // Accessing out of bounds could lead to
    ArrayIndexOutOfBoundsException
```

```
}

}
```

- The loop should iterate with i < amounts.length. The current implementation could access amounts[i] when i equals the length of the array, leading to an ArrayIndexOutOfBoundsException.

3.For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.

```
public Transaction getTransaction(int index) {

return transactions.get(index); // potential out-of-bounds access

}
```

- There is a potential for index to be out of bounds. If the index is derived from user input or calculations, proper validation should be applied to ensure that it is a valid integer and within the bounds of the transactions list. When converting strings to integers for indexing, proper error handling and bounds checking are necessary to avoid runtime exceptions.

4. For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the "dangling reference" problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.

- In updateBalance(), if account is a pointer and the account object is deleted or goes out of scope elsewhere, it may lead to a dangling reference. Ensure account points to valid memory.

5.When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names? Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an example, a FORTRAN program contains a real variable A and an integer variable B; both are made aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.

- The code does not demonstrate direct aliasing (like EQUIVALENCE in FORTRAN). However, if Account or Transaction classes were to have overlapping field names that might reference the same memory, it could create confusion about which attribute is being accessed.

6. Does a variable's value have a type or attribute other than what the compiler expects? This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical representation of the record differs from the structure definition.

- The amount in the Transaction class is declared as int, which may not cover larger transaction values. If large amounts are expected, it should be changed to a float or double.

7.Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced. This situation also could occur when passing a bit-string argument to a subroutine.

- Potential Issue: The use of character arrays like char t_name[30] and char t_address[30] might result in addressability issues if boundary alignment is not properly maintained during memory operations like reading and writing data.
- Error: If the code writes a char[] to a file using file.write((char *)this, sizeof(initial));, but later reads it back without considering alignment issues (e.g., non-byte-aligned data), it may lead to reading incorrect memory locations.

8. If pointer or reference variables are used, does the referenced memory location have the attributes the compiler expects? An example of such an error is where a C++pointer upon which a data structure is based is assigned the address of a different data structure.

```
strcpy(t_name, ini.return_name(t_accno));
```

- The function return_name(int t_accno) returns a pointer to a local t_name, which can lead to undefined behavior if that memory is modified after the function returns. A deep copy should be made instead.

9. If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?

- The structures initial and account are defined in different contexts but use similar operations for reading and writing data files. If there are subtle differences in how these structures are defined (e.g., field size mismatches), it could lead to reading or writing incorrect values.

10. When indexing into a string, are the limits of the string off by-one errors in indexing operations or in subscript references to arrays?

```
for (i = 1; i <= 76; i++) // Incorrect loop boundary

    cprintf(" ");
```

- The loop runs for i <= 76, which could be off by one if the intended number of iterations is 76 (i.e., the condition should be i < 76 to avoid overrunning the bounds).

11. For object-oriented languages, are all inheritance requirements met in the implementing Class?

- The class main_menu and shape contain no virtual destructors. If these classes are intended to be inherited and used polymorphically, this can lead to memory leaks when deleting derived class objects through base class pointers. Adding virtual destructors to the base classes would solve this.

<u>Category B: Data-Declaration Errors</u>

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A (I)) is interpreted as a function call, leading to the machine's attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?

- All variables seem to be declared, but there is a concern with the initial and account classes where data members like accno, name, balance, etc., are used directly without checks for explicit scope in functions like add_to_file() or delete_account(). Also, it's

worth checking if array parameters (like char[]) passed between functions are correctly declared to avoid being interpreted as functions.

char *return_name(int t_accno); // correctly declares the return type, but further use needs validation.

2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well understood? For instance, the default attributes received in Java are often a source of surprise.

- The attributes for local variables like int i, int t_accno are well-stated in the code. However, consider explicitly declaring array sizes where necessary, or specifying types and initialization values for variables in class member declarations to avoid defaults that could lead to unpredictable behavior

char t_name[30]; // Array size is fixed, but initialization is not always consistent.

3. Where a variable is initialized in a declarative statement, is it properly initialized? In many languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.

- Many variables are declared but not initialized properly at the time of declaration, particularly in arrays like char name[30]. This could result in accessing junk values.

char name[30]; // Should be initialized with empty strings or set to null.

4. Is each variable assigned the correct length and data type?

- The variables seem to have appropriate data types (e.g., char[] for strings, float for balances). However, the string arrays (e.g., char t_name[30]) have fixed sizes, which might not accommodate larger inputs, potentially causing buffer overflows.

char t_name[30]; // Fixed length, ensure it aligns with expected string sizes.

5. Is the initialization of a variable consistent with its memory type?

- The use of char arrays like name[30] and address[30] without initialization could cause inconsistencies, especially when they are passed around in the code. These should ideally be initialized or checked before use.

strcpy(name, ""); // Strings like 'name' should be initialized with empty strings to avoid junk values.

6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

- There are no instances of confusingly similar variable names like VOLTS and VOLT in the provided code. However, ensuring consistency in variable naming conventions across the program is still good practice.

Category C: Computation Errors.

1. **Are there any computations using variables having inconsistent (such as non-arithmetic) data types?**
    - No, the computations in the code seem to involve appropriate data types for their purposes. For example:
        - Variables like `int`, `float`, and `char` are used consistently for arithmetic and string operations.
        - However, functions such as `puts()` and `gets()` are used for character arrays (`char[]`), which might cause issues in modern C++ standards due to unsafe handling of strings (like buffer overflow risks).
2. **Are there any mixed-mode computations?**
    - Yes, mixed-mode computations occur in some instances:
        - In functions like `account::transaction()`, the transaction balance is updated by adding or subtracting floating-point numbers (`float t_balance`) with other variables (e.g., `t_amount`). This involves a mix of floating-point and integer data types (depending on the input for the transaction), but C++ handles this by implicit type conversion.

Example:

```
t_balance = t_balance + t_amount + t_interest;
```

        - Here, `t_balance` and `t_amount` are floats, but depending on how the values are passed, mixed-mode computation could occur.
    - Mixed-mode computations are not errors but can cause precision issues, especially with floating-point numbers.

3. **Are there any computations using variables having the same data type but different lengths?**
    - No obvious issues with variables having the same data type but different lengths are present. Most variables are either `int` or `float`, and there aren't any specific cases where the length of one data type causes problems.
    - However, handling `char[]` with fixed sizes like `char t_name[30]` could lead to truncation if the input exceeds the specified length, but this is more related to buffer overflow than computation length.
4. **Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?**
    - There are no direct cases where the target variable's data type is smaller than the data type of the result of the right-hand expression. However, there are potential issues in terms of precision:
        - In the `transaction()` function, the balance is updated with floating-point arithmetic, which could lose precision if handled as integers, but since all the related variables (`t_balance`, `t_amount`) are `float`, no size mismatch issues arise.

5**. Is an overflow or underflow expression possible during the computation of an expression?**
There is a low chance of overflow/underflow, but potential issues could arise in the handling of **floating-point calculations**, such as interest computation or balance adjustments. For example, in the `transaction()` function, where the balance is updated:
`t_balance = t_balance + t_amount + t_interest;`

- If `t_amount` or `t_interest` is too large (e.g., excessively large deposits or interest accumulation), it might cause an overflow, especially with large floating-point values. However, since these are `float` variables, modern systems should handle the range well unless there is extreme input. For underflow, similar issues might occur for very small `float` values approaching zero.

6**. Is it possible for the divisor in a division operation to be zero?**
Yes, it is possible. While the code doesn't explicitly show many division operations, division by zero could occur in cases like interest calculations:
`float calculate_interest(int t_accno, float t_balance)`

- If `t_balance` is zero or very small and there's any division related to interest calculation, there could be a risk. However, in the current code, this doesn't seem explicitly handled, so caution should be taken to ensure the divisor isn’t zero before division.

7**. If the underlying machine represents variables in base-2 form, are there any sequences of resulting inaccuracy?**
Yes, floating-point arithmetic on a binary machine can lead to inaccuracies. For example, in the code, when performing operations with floating-point numbers (`float`), such as `t_balance`, `t_amount`, and `t_interest`, there could be inaccuracies due to how numbers are represented in base-2 form:
`t_balance = t_balance + t_amount + t_interest;`

- Multiplying or adding floating-point numbers like 0.1 or similar values can lead to inaccuracies because of how binary systems handle fractional values. Therefore, computations involving money or interest might suffer from rounding issues.

8. **Can the value of a variable go outside the meaningful range?**
Yes, it is possible, particularly for the balance (`t_balance`). For example, if a user tries to withdraw more than the available balance, it might lead to a negative balance:
`if (t_tran == 'W' && t_amount > t_balance)`

- Although there is a check to prevent this, no safeguard ensures that other variables remain within reasonable ranges. Additionally, extreme deposit values or interest calculations might result in balance values that exceed the meaningful range (e.g., unrealistic values in terms of monetary limits).

9**. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?**
Yes, the order of evaluation seems correct in the code. For example:

```
t_balance = t_balance + t_amount + t_interest;
```

- The evaluation of this expression follows correct precedence rules, as addition operations are left-associative. There are no complex expressions with mixed operators that could cause confusion in precedence or evaluation order.

10. **Are there any invalid uses of integer arithmetic, particularly divisions?**

- There doesn't appear to be any specific case of invalid integer arithmetic in the code. Most calculations involve floating-point numbers (e.g., balances and amounts are `float`), which reduce the risk of integer-related precision issues. However, care should be taken to handle the potential rounding issues with floating-point division in other parts of the program. For example, if integer arithmetic was used with division, similar to:

```
int result = 2 * i / 2;
```
the result could depend on whether `i` is even or odd, but this doesn't appear in the code.

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?

The code generally avoids comparing incompatible types. However, in functions like transaction():

```
t_amt = atof(tm);
```

tm is converted from a string to a float, and comparisons occur later between t_amt and other numerical values. Since conversion is done explicitly, this comparison should be safe, but always ensure that inputs are properly validated before conversion.

2. Are there any mixed-mode comparisons or comparisons between variables of different lengths? If so, ensure that the conversion rules are well understood.

In the transaction() function, t_amt is a float derived from a string, and it's compared with other floats (t_balance). This is done correctly, but care must be taken when comparing the results of type conversions. For example, strcmp() is used to compare strings like:

```
if (strcmp(t_type, "CASH") && strcmp(t_type, "CHEQUE"))
```

3. Are the comparison operators correct? Programmers frequently confuse such relations as at most, at least, greater than, not less than, less than or equal.

```
if (t_tran != 'D' && t_tran != 'W')
```

The comparison seems correct. However, pay attention to ensuring that != and == are used correctly throughout the code.

4.Does each Boolean expression state what it is supposed to state? Programmers often make mistakes when writing logical expressions involving and, or, and not.

There are no complex logical conditions where common mistakes appear (such as (a==b==c) or 2<i<10 errors). However, it's important to review conditions like:

```
if ((mm < mm1 && yy <= yy1) || (mm1 < mm && yy < yy1))
```

5.Are the operands of a Boolean operator Boolean? Have comparison and Boolean operators been erroneously mixed together? This represents another frequent class of mistakes. Examples of a few typical mistakes are illustrated here. If you want to determine whether i is between 2 and 10, the expression 2<i<10 is incorrect; instead, it should be (2<i) && (i<10). If you want to determine whether i is greater than x or y, i>x||y is incorrect; instead, it should be (i>x)||(i>y). If you want to compare three numbers for equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x>y>z, the correct expression is(x>y)&&(y>z).

The operands in the Boolean operations seem to be of the correct type. For example, in:

```
if ((t_tran == 'D') || (t_tran == 'W'))
```

This comparison uses characters and is valid. Ensure that any arithmetic comparisons, such as x > y, do not mix in Boolean operators.

6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.

```
float t_interest = ((t_balance*2)/100 * months);
```

Since floating-point arithmetic is involved, ensure that results are compared with an acceptable precision margin (e.g., using a small epsilon value) rather than directly comparing floats.

7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an expression such as (if((a==2) && (b==2) || (c==3)), is it well understood whether the and or the or is performed first?

```
if ((mm < mm1 && yy <= yy1) || (mm1 < mm && yy < yy1))
```

Ensure the precedence of && and || is what is intended. Here, && takes precedence over ||, which is correct.

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 && (x/y)>z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers

```
 if ((t_tran == 'D') && (t_balance >= 500))
```

should be safe because C++ generally short-circuits evaluations. However, care must be taken if mathematical operations (e.g., divisions) are involved in conditions, ensuring no unintended operations are performed before a logical condition fails.

Category E: Control-Flow Errors

1. If the program contains a multiway branch such as a computed GO TO, can the index variable ever exceed the number of branch possibilities? For example, in the statement GO TO (200, 300, 400), i will always have the value of 1, 2, or 3?

Answer:

Multiway Branch and Index Variable Limits:

The program does not contain a computed `GO TO` statement, but the closest analogy would be how the `menu()` function handles user input and navigation:

```
char main_menu::menu()

{

    ...

    for(i = 1; i < 7; i++)

        normalvideo(32, i+10, a[i]); // a[] is the options array

    reversevideo(32, 10, a[0]);

    i = done = 0;

    do {

        int key = getch();

        switch (key) {

            case 00:

                key = getch();

                switch (key) {

                    case 72: // Up arrow
```

```
                    normalvideo(32, i+10, a[i]);
                    i--;
                    if (i == -1) i = 6;
                    reversevideo(32,i+10,a[i]);
                    break;
                case 80: // Down arrow
                    normalvideo(32, i+10, a[i]);
                    i++;
                    if (i == 7) i = 0;
                    reversevideo(32, i+10, a[i]);
                    break;
            }
        break;
        case 13: // Enter key
            done = 1;
    }
  } while (!done);
  return(i+49);
}
```

- Here, `i` controls menu navigation and wraps around correctly (between 0 and 6), ensuring it cannot exceed valid options, similar to how computed `GO TO` works. So, `i` will always have a value between 0 and 6, preventing out-of-bounds issues.

2. Will every loop eventually terminate? Devise an informal proof or argument showing that each loop will terminate.

Answer:

In general, yes. Most loops in this program, including the `do-while` loops used for menu navigation and input validation, have clear termination conditions:

- The main menu loop terminates when `done = 1`, which occurs when the user presses the Enter key.
- Input loops such as those in the `transaction()` or `new_account()` functions terminate when valid input is provided or when a specific condition like pressing '0' to exit is met.

Proof of Termination:

- Every loop is based on user input or file processing. For example:
  - The `menu()` loop exits when the user selects a valid menu option (Enter key).
  - The `display_list()` loop reads through account records until EOF, so it terminates when all records are processed.
- Therefore, each loop is designed to terminate either when user input is processed or data processing completes

3. Will the program, module, or subroutine eventually terminate?

Answer:

Yes. The program contains clear exit conditions:

- The user can exit the program through the "Exit" option in the menu, which calls `exit(0)` to terminate the program.
- Each module or function (like `new_account()`, `transaction()`, etc.) completes its task and returns to the menu, which eventually allows the user to exit.

Since there is no infinite loop and every loop or menu has a termination condition, the program as a whole will eventually terminate.

4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over- sight? For instance, if you had the following loops headed by the following statements:

for (i==x ; i<=z; i++) {

...

}

while (NOTFOUND) {

...

}

what happens if NOTFOUND is initially false or if x is greater than z?

Answer:

Yes, it is possible depending on the initial conditions.

For example, in the `for` loop structure:

```
for (i = 0; i <= z; i++) {

    // Loop body

}
```

- If `z < 0`, the loop would never execute because the condition `i <= z` would be false from the start.

In the `while (NOTFOUND)` case:

```
while (NOTFOUND) {

    // Loop body

}
```

- If `NOTFOUND` is initially false, the loop would never execute.

In the provided code: For instance, in the account search function:

```
if (!ini.found_account(t_accno)) {

    cout << "\7Account not found";

    getch();

    return;
```

```
}
```

- If `found_account()` returns `false`, certain operations will not be executed, effectively skipping those parts of the code.
- This behavior is intentional and not an oversight, as it's designed to prevent unnecessary operations when certain conditions aren't met.

5. For a loop controlled by both iteration and a Boolean condition (a searching loop, for example) what are the consequences of loop fall-through? For example, for the psuedo-code loop headed by DO I=1 to TABLESIZE WHILE (NOTFOUND) what happens if NOTFOUND never becomes false?

Answer:

```
DO I = 1 to TABLESIZE WHILE (NOTFOUND)
```

If `NOTFOUND` never becomes `false`, the loop will:

- **Fall through all iterations**: It will continue running for the full range (from 1 to `TABLESIZE`), even though the target condition (finding the element) is never met. This leads to wasted computation if the element is not found early.
- **Consequences**:
    - If the loop is supposed to stop when an item is found, failing to update `NOTFOUND` correctly will result in unnecessary iterations.
    - The program might still terminate when `I` reaches `TABLESIZE`, but it would be inefficient.

In your program:

- Similar behavior can occur in search functions like `found_account()`. If the account is not found, the loop reads through all records. While not technically an infinite loop, it wastes resources reading the entire file even if it knows the account doesn't exist.

6. Are there any off-by-one errors, such as one too many or too few iterations? This is a common error in zero-based loops. You will often forget to count "0" as a number. For

example, if you want to create Java code for a loop that counted to 10, the following would be wrong, as it counts to 11:

for (int i=0; i<=10;i++) {

System.out.println(i);

}

Correct, the loop is iterated 10 times:

for (int i=0; i <=9;i++) {

System.out.println(i);

Answer:

Off-by-one errors occur when the loop includes one iteration too many or too few, often due to incorrect range boundaries (e.g., using `<=` instead of `<`).

Example:

```
for (int i = 0; i <= 10; i++) {

    System.out.println(i);

}
```

- This loop runs 11 times (from `0` to `10`), whereas it should ideally run 10 times (from `0` to `9`).

Correct loop:
cpp
Copy code

```
for (int i = 0; i < 10; i++) {

    System.out.println(i);

}
```

In program:

- The code does not seem to have any off-by-one errors. For example, the menu navigation properly bounds `i` between `0` and `6`, and file reading stops at EOF, ensuring no out-of-bounds access or extra iterations.

7. If the language contains a concept of statement groups or code blocks (e.g., do-while or {...}), is there an explicit while for each group and do the do's correspond to their appropriate groups?

Or is there a closing bracket for each open bracket? Most modern compilers will complain of such mismatches.

Answer:

In modern C++ (and other structured programming languages), every `do` must correspond to a matching `while` condition. Similarly, in block-structured languages (like C++, Java), every opening bracket `{` must have a matching closing bracket `}`.

In your code:

All `do-while` and loop structures are correctly matched. For example:

```
do {

    // menu logic

} while (!done);
```

The `do` corresponds correctly to the `while`.

8. Are there any non-exhaustive decisions? For instance, if an input parameter's expected values are 1, 2, or 3, does the logic assume that it must be 3 if it is not 1 or 2? If so, is the assumption valid?

Answer:

A non-exhaustive decision occurs when you assume that all possible cases are covered, but a potential input is not explicitly handled.

Example:

```
if (option == 1) {

    // Do something
```

```
} else if (option == 2) {

    // Do something else

} else {

    // Assume option == 3

}
```

If `option` is not `1`, `2`, or `3`, this logic will fail to handle unexpected inputs, leading to potential errors.

In your program:

The menu handles all input cases correctly:
cpp
Copy code

```
case '1':

    // Do task 1

    break;

case '2':

    // Do task 2

    break;

case '3':

    // Do task 3

    break;

default:

    // Handle unexpected input
```

- **There is no assumption that the input will always be within a valid range; the defvalid inault case ensures inputs are managed safely.**

Category F: Interface Errors

1. **Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?**

   Yes, the number of parameters in most cases matches the arguments passed. Functions like `add_to_file`, `update_balance`, and `modify_account` consistently receive arguments that correspond to their defined parameters. For example, in `add_to_file(int t_accno, char t_name[30], char t_address[30], float t_balance)`, the number and types of parameters match what is passed during calls the attributes (e.g., data type and size) of each parameter match the attributes of each corresponding argument?**

   Yes, the attributes match in most cases. For example, `t_accno` is consistently used as an `int` in both the function definition and during function calls. Similarly, `name` and `address` are handled as `char[30]`, ensuring consistency between the parameters and arguments **.**

2 .Does system of each parameter match the units system of each corresponding argument?

- No specific unit system (like degrees vs radians) is discussed in the provided code, as the context is focused on data like account numbers, names, and balances. Thus, no mismatch in the unit system can be identified in this case .

3. Does the nuuments transmitted by this module to another module equal the number of parameters expected by that module?

- Yes, the arguments transmitted to other modules align with the number of parameters expected. For example, `add_to_file` consistently passes the correct number of arguments when called within the `new_account` and `transaction` functions

4. Do the nt transmitted to another module match the attributes of the corresponding parameter in that module?

- Yes, the attributes like `int` for account numbers and `float` for balances match when passed between functions like `update_balance` and `add_to_file`. No mismatch was observed

5. Does the units synsmitted to another module match the units system of the corresponding parameter in that module?

- Since the code deals with banking data (such as account numbers and balances), and no specific units (like metric or imperial) are mentioned, this question isn't directly applicable here. The transmitted arguments are of compatible data types without specific unit systems being referenced .

6. If built-in functions are invoked, are thttributes, and order of the arguments correct?

- Yes, built-in functions like `gets()`, `atoi()`, and `strcpy()` are invoked with the correct number of arguments and appropriate data types, ensuring the expected behavior. For example, `gets()` is called with `char[]`, and `atoi()` with a string for conversion

7.Does a subroutine alter a parameter th to be only an input value?

- No evidence suggests that parameters meant to be input-only are unintentionally modified. Functions like `update_balance` modify account data as intended, but these parameters are not altered unless required for that purpose

8. If global variables are me definition and attributes in all modules that reference them?

- Global variables like `scan`, `ascii`, and others are referenced consistently throughout the code without any apparent issues regarding mismatched definitions .

Category G: Input / Output Errors

1. **If files are explicitly declared, are their attributes correct?**
    - Files like "INITIAL.dat", "BANKING.dat", and "TEMP.dat" are used in the code with attributes like ios::in, ios::out, ios::app, and ios::ate. These attributes are correctly assigned for reading, writing, and appending records.
2. **Are the attributes on the file's OPEN statement correct?**
    - Yes, the attributes are correct based on the operations. For instance:
        - ios::in is used for reading data.
        - ios::out | ios::app is used for appending data.
        - ios::out | ios::ate is used for updating a record at the correct position. The attributes match the intended file operations.
3. **Is there sufficient memory available to hold the file your program will read?**
    - The program doesn't explicitly check for available memory before reading files. However, given typical usage, there should be enough memory, but adding a check for file size or available memory would improve robustness.
4. **Have all files been opened before use?**
    - Yes, files are opened before any operations are performed on them.

5. Have all files been closed after use?
    - Yes, the code ensures that all files are properly closed after reading, writing, or modifying records.
6. **Are end-of-file conditions detected and handled correctly?**
    - EOF conditions are handled correctly using file.eof() in loops that process records. This prevents reading past the end of the file.
7. **Are I/O error conditions handled correctly?**
    - The code does not explicitly handle I/O errors like failure to open a file. Adding error handling with checks like if (!file) after opening a file would improve the program.
8. **Are there spelling or grammatical errors in any text that is printed or displayed by the program?**
    - There are a few grammatical issues in the text prompts, such as:
        - "Accnount no." should be "Account no."
        - "Founds the last account no." could be rephrased to "Finds the last account number."
        - "Should not less than 500" should be "Should not be less than 500." Minor adjustments can make the text more professional and readable.

Category H: Other Checks:

1. If the compiler produces a cross-reference listing of identifiers, examine it for variables that are never referenced or are referenced only once.

- In the uploaded code, several variables seem to be defined but used minimally or not at all:
    - ascii and scan: These variables are declared but never used within the code provided.
    - done in the main_menu class is set to 0 and used in loops, but the same logic can be achieved without this variable in some cases. You might want to review its necessity.
- There may be other variables that appear unused depending on the full compilation and cross-referencing.

2. If the compiler produces an attribute listing, check the attributes of each variable to ensure that no unexpected default attributes have been assigned.

- Without a specific attribute listing provided by the compiler, it's challenging to comment on this fully. However, all variables in the code appear to be properly initialized and assigned reasonable default values.

- Example: Most of the variables such as t_accno, t_balance, and accno are declared as integers or floats, which align with their intended purposes.

3. If the program compiled successfully, but the computer produced one or more "warning" or "informational" messages, check each one carefully.

- **Potential warnings**:
  - **Use of deprecated functions**: Functions like gets() and strupr() are considered unsafe in modern compilers and should be replaced with safer alternatives (fgets() or std::getline() for string input and std::transform() for string conversion).
  - **Type casting warnings**: Some places use implicit type conversion, such as atoi() and atof(). You should ensure that all conversions are handled safely, and check for invalid input cases that could trigger warnings.
- **Informational messages**:
  - **Undeclared variables**: Ensure that variables like d1, m1, y1 (for dates) are correctly declared and initialized before use. The structure for date may also trigger warnings if not properly defined in all compilers.

4. Is the program or module sufficiently robust? That is, does it check its input for validity?

- Input validation is present, but it is minimal:
  - For example, in the new_account() function, there is validation for names, addresses, and the initial deposit amount, but improvements can be made:
    - Input like account numbers, transaction amounts, and dates should have better validation to prevent invalid data entry (e.g., non-numeric input for numbers).
    - The program should handle cases where file operations fail (like if the file doesn't open correctly).
- Input for operations like deleting or modifying accounts could benefit from additional checks to ensure valid account numbers are input.

5. Is there a function missing from the program?

- The program seems mostly complete, but a few functionalities might be missing:
  - **Input sanitization**: Functions to sanitize and validate inputs before processing could help improve security and robustness.
  - **Error handling for file operations**: Functions to handle file opening and closing errors would make the system more resilient to I/O issues.
  - **Backup/restore mechanism**: A function to backup and restore data files like INITIAL.dat and BANKING.dat could be beneficial for real-world usage.
  - **Logging**: Adding a logging mechanism to track changes to accounts and transactions might be useful for debugging and audit purposes.

II. CODE DEBUGGING: Debugging is the process of localizing, analyzing, and removing suspected errors in the code (Java code given in the .zip file)

1. Armstrong

A. Program Inspection

1. The program contains one error, specifically related to the computation of the remainder. This error has been identified and corrected.
2. The appropriate program inspection category for this issue is Category C: Computation **Errors**, as the error involves an incorrect computation of the remainder.
3. Program inspection does not address debugging aspects, such as breakpoints or runtime logic errors.
4. Program inspection is a valuable technique for identifying and resolving issues related to code structure and computational errors.

B. Debugging

1. The program contains one error, related to the computation of the remainder, as previously identified.
2. To fix this error, set a breakpoint at the point where the remainder is calculated. This allows you to step through the code and observe the values of variables during execution to ensure correctness.
3. The corrected executable code is as follows:

```
// Armstrong Number
class Armstrong {
    public static void main(String args[]) {
        int num = Integer.parseInt(args[0]);
        int n = num; // Store original number for final comparison
        int check = 0, remainder;

        while (num > 0) {
            remainder = num % 10;
            check = check + (int) Math.pow(remainder, 3);
            num = num / 10;
        }

        if (check == n)
```

```
            System.out.println(n + " is an Armstrong Number");
        else
            System.out.println(n + " is not an Armstrong Number");
    }
}
```

2. GCD and LCM

A. Program Inspection

1. There are two errors in the program:
2. **Error 1**: In the `gcd` function, the condition of the while loop should be `while(b != 0)` instead of `while(a % b == 0)` to calculate the GCD correctly.
3. **Error 2**: In the `lcm` function, the logic is incorrect, which could lead to an infinite loop. The calculation needs to be corrected.
4. The most appropriate category for inspecting this program is **Category C: Computation Errors**, as the issues arise in the computational logic of both the `gcd` and `lcm` functions.
5. Program inspection cannot detect runtime issues or logical errors like infinite loops. It focuses on finding issues related to computation and code structure.
6. Program inspection is valuable for identifying and fixing computation errors, as seen in the GCD and LCM calculations here.

B. Debugging

1. As mentioned earlier, there are two errors in the program.
2. To fix these errors:
3. For **Error 1** in the `gcd` function, set a breakpoint at the start of the while loop to verify that the loop operates correctly.
4. For **Error 2** in the `lcm` function, the logic needs to be reviewed and corrected to avoid an infinite loop and calculate the LCM properly.
5. The corrected and fully functional code is provided below:

```
import java.util.Scanner;

public class GCD_LCM {
```

```
    // Function to calculate GCD
    static int gcd(int x, int y) {
        int a, b;
        a = (x > y) ? x : y;  // 'a' is the larger number
        b = (x < y) ? x : y;  // 'b' is the smaller number

        // Corrected while loop condition for GCD
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Function to calculate LCM using GCD
    static int lcm(int x, int y) {
        return (x * y) / gcd(x, y);  // LCM formula using GCD
    }

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);

        // Input the two numbers
        System.out.println("Enter the two numbers: ");
        int x = input.nextInt();
        int y = input.nextInt();

        // Output GCD and LCM
        System.out.println("The GCD of two numbers is: " +
gcd(x, y));
        System.out.println("The LCM of two numbers is: " +
lcm(x, y));

        input.close();
    }
}
```

3. Knapsack

A. Program Inspection

1. There is one error in the program, which occurs in the line: `int option1 = opt[n++][w];`. The variable `n` is incorrectly incremented. It should be: `int option1 = opt[n][w];`, where n is used without being incremented.
2. The most suitable category of program inspection for this code is Category C: Computation Errors, as the issue is related to an error in the computation within loops.
3. Program inspection cannot identify runtime errors or logical errors that might arise during execution of the program.
4. Program inspection is a valuable technique for detecting and correcting computation-related issues, like the one identified here.

B. Debugging

1. As mentioned above, there is one error in the program.
2. To fix this error, place a breakpoint at the line: `int option1 = opt[n][w];` to ensure that n and w are used correctly without unintended increments.
3. The corrected executable code is as follows:

```
public class Knapsack {
public static void main(String[] args) {
int N = Integer.parseInt(args[0]);  // number of items
int W = Integer.parseInt(args[1]);  // maximum weight of
knapsack
int[] profit = new int[N + 1]; int[] weight = new int[N + 1];
// Generate random instance, items 1..N for (int n = 1; n <= N;
n++) {
profit[n] = (int) (Math.random() * 1000); weight[n] = (int)
(Math.random() * W);
}
```

```
int[][] opt = new int[N + 1][W + 1]; boolean[][] sol = new
boolean[N + 1][W + 1];
for (int n = 1; n <= N; n++) {
for (int w = 1; w <= W; w++) {
int option1 = opt[n - 1][w]; // Fixed the increment here int
option2 = Integer.MIN_VALUE;
if (weight[n] <= w)
option2 = profit[n] + opt[n - 1][w - weight[n]];
opt[n][w] = Math.max(option1, option2); sol[n][w] = (option2 >
option1);
}
}

// Rest of the code is fine
// Print results
System.out.println("Item" + "\t" + "Profit" + "\t" + "Weight" +
"\t" + "Take"); for (int n = 1; n <= N; n++) {
System.out.println(n + "\t" + profit[n] + "\t" + weight[n] +
"\t" + take[n]);
}
}
}
```

4. Magic Number

A. Program Inspection

1. There are two errors in the program:
2. **Error 1**: In the inner while loop, the condition should be `while (sum > 0)` instead of `while (sum == 0)` to ensure the loop executes correctly.
3. **Error 2**: Inside the inner while loop, semicolons are missing from the following lines:
    - `s = s * (sum / 10);`
    - `sum = sum % 10;` The missing semicolons need to be added to avoid syntax errors.
4. The most appropriate category of program inspection for this code is **Category C: Computation Errors**, as the identified issues relate to computation within the while loop.

5. Program inspection does not detect runtime issues or logical errors that might appear during execution.
6. The program inspection technique is useful in identifying and fixing computation-related issues, such as those found in this code.

B. Debugging

1. As mentioned above, there are two errors in the program.
2. To fix these errors, you would place one breakpoint at the beginning of the inner while loop to verify that the loop runs correctly. Additionally, you can set breakpoints to inspect the values of `num` and `s` during execution to ensure correctness.
3. The corrected executable code is as follows:

```java
import java.util.*;

public class MagicNumberCheck {
    public static void main(String args[]) {
        Scanner ob = new Scanner(System.in);

        // Input the number to be checked
        System.out.println("Enter the number to be checked.");
        int n = ob.nextInt();

        int sum = 0, num = n;

        // Outer loop to reduce the number to a single digit
        while (num > 9) {
            sum = num;
            int s = 0;

            // Inner loop to calculate the sum of digits
            while (sum > 0) { // Fixed the condition here
                s = s * (sum / 10);  // Fixed missing semicolon
                sum = sum % 10;      // Fixed missing semicolon
            }
            num = s;
        }
```

```
        // Check if the final number is 1, indicating a Magic
Number
        if (num == 1) {
            System.out.println(n + " is a Magic Number.");
        } else {
            System.out.println(n + " is not a Magic Number.");
        }
    }
}
```

5. Merge Sort

A. Program Inspection

1. There are several errors in the program:
2. **Error 1**: In the `mergeSort` method, the lines `int[] left = leftHalf(array+1);` and `int[] right = rightHalf(array-1);` are incorrect. The array is not split properly. The correct approach is to pass the array itself and split it in helper functions.
3. **Error 2**: The `leftHalf` and `rightHalf` methods are incorrectly implemented. They should return the correct halves of the array rather than a single index shift.
4. **Error 3**: The `merge` method should accept the full `left` and `right` arrays rather than incorrect pointers like `left++` and `right--`.
5. The most appropriate category of program inspection for this code is **Category C: Computation Errors**, since the issues stem from improper computations during the array splits and merges.
6. Program inspection does not detect runtime issues or logical errors that may occur during program execution.
7. Program inspection is useful here for identifying and correcting the computation-related issues in array manipulation.

B. Debugging

1. There are multiple errors in the program, as noted in the inspection.
2. To fix these errors, you would need to set breakpoints in the `mergeSort`, `leftHalf`, `rightHalf`, and `merge` methods to check the values of `left`, `right`, and `array`

during execution. Also, setting breakpoints inside the `merge` method to monitor `i1` and `i2` will help verify correct array merging.

3. The corrected executable code is as follows:

```
import java.util.*;

public class MergeSort {
    public static void main(String[] args) {
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};

        System.out.println("Before: " + Arrays.toString(list));
        mergeSort(list);
        System.out.println("After:  " + Arrays.toString(list));
    }

    // Merge Sort function
    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int[] left = leftHalf(array);   // Correct array
splitting
            int[] right = rightHalf(array); // Correct array
splitting

            mergeSort(left);   // Recursively sort the left half
            mergeSort(right);  // Recursively sort the right half

            merge(array, left, right);  // Merge the sorted halves
        }
    }

    // Function to get the left half of the array
    public static int[] leftHalf(int[] array) {
        int size1 = array.length / 2;
        int[] left = new int[size1];
        for (int i = 0; i < size1; i++) {
            left[i] = array[i];
        }
        return left;
    }
```

```
    // Function to get the right half of the array
    public static int[] rightHalf(int[] array) {
        int size1 = array.length / 2;
        int size2 = array.length - size1;
        int[] right = new int[size2];
        for (int i = 0; i < size2; i++) {
            right[i] = array[i + size1];
        }
        return right;
    }

    // Function to merge the two halves into a sorted array
    public static void merge(int[] result, int[] left, int[] right) {
        int i1 = 0; // Index into the left array
        int i2 = 0; // Index into the right array

        for (int i = 0; i < result.length; i++) {
            if (i2 >= right.length || (i1 < left.length && left[i1]
<= right[i2])) {
                result[i] = left[i1];
                i1++;
            } else {
                result[i] = right[i2];
                i2++;
            }
        }
    }
}
```

6.Multiply Matrices

A. Program Inspection

1. There are several errors in the program:
2. **Error 1**: The loop indices in the nested loops for matrix multiplication should start from 0, not -1. Starting from -1 would cause array index errors.

3. **Error 2**: The error message displayed when matrix dimensions are incompatible contains an incorrect format. It should be corrected to: “Matrices with entered orders can't be multiplied with each other.”
4. The most appropriate category of program inspection for this code is **Category C: Computation Errors**, as the identified errors relate to computation during matrix multiplication.
5. Program inspection cannot detect runtime or logical errors such as incorrect dimension checks or output formatting issues.
6. The program inspection technique is helpful for identifying and correcting computation-related issues.

B. Debugging

1. Multiple errors are present in the program, as identified above.
2. To resolve these errors, breakpoints should be set to monitor the values of variables `c`, `d`, `k`, and `sum` during the matrix multiplication process. Special attention should be paid to the nested loops to verify that matrix indices are handled correctly and that matrix dimensions are compatible.
3. The corrected executable code is as follows

```
import java.util.Scanner;

class MatrixMultiplication {
    public static void main(String args[]) {
        int m, n, p, q, sum = 0, c, d, k;

        // Taking input for the first matrix dimensions
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of
the first matrix");
        m = in.nextInt();
        n = in.nextInt();

        // Declaring the first matrix
        int first[][] = new int[m][n];
        System.out.println("Enter the elements of the first matrix");

        // Input for first matrix elements
        for (c = 0; c < m; c++)
```

```
            for (d = 0; d < n; d++)
                first[c][d] = in.nextInt();

        // Taking input for the second matrix dimensions
        System.out.println("Enter the number of rows and columns of
the second matrix");
        p = in.nextInt();
        q = in.nextInt();

        // Checking if multiplication is possible based on matrix
dimensions
        if (n != p) {
            System.out.println("Matrices with entered orders can't be
multiplied with each other.");
        } else {
            int second[][] = new int[p][q]; // Declaring the second
matrix
            int multiply[][] = new int[m][q]; // Resultant matrix

            System.out.println("Enter the elements of the second
matrix");

            // Input for second matrix elements
            for (c = 0; c < p; c++)
                for (d = 0; d < q; d++)
                    second[c][d] = in.nextInt();

            // Performing matrix multiplication
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++) {
                    sum = 0;
                    for (k = 0; k < n; k++) {
                        sum += first[c][k] * second[k][d];
                    }
                    multiply[c][d] = sum;
                }
            }
```

```
            // Output the product of the matrices
            System.out.println("Product of entered matrices:");
            for (c = 0; c < m; c++) {
                for (d = 0; d < q; d++)
                    System.out.print(multiply[c][d] + "\t");
                System.out.println();
            }
        }
    }
}
```

7. Quadratic Probing

A. Program Inspection

1. There are multiple errors in the program:
2. **Error 1**: The `insert` method contains a typo in the line `i + = (i + h / h–)`. It should be corrected to avoid syntax errors.
3. **Error 2**: In the `remove` method, there is a logic error in the loop that rehashes keys. It should be `i = (i + h * h++) % maxSize;` to correctly update the index.
4. **Error 3**: In the `get` method, the loop to find the key has a logic error. It should also use `i = (i + h * h++) % maxSize;` for proper indexing.
5. The most relevant categories for program inspection for this code are **Category A: Syntax Errors** and **Category B: Semantic Errors**, as both syntax errors and logical issues are present.
6. The program inspection technique is valuable for identifying and fixing these errors but may not catch logical errors that affect the program's behavior during execution.

B. Debugging

1. There are three errors in the program, as detailed above.
2. To address these errors, breakpoints should be set to examine variables like `i`, `h`, `tmp1`, and `tmp2` while stepping through the code. Pay particular attention to the logic within the `insert`, `remove`, and `get` methods.
3. The corrected executable code is as follows:

```
import java.util.Scanner;

class QuadraticProbingHashTable {
```

```
    private int currentSize, maxSize;
    private String[] keys;
    private String[] vals;

    public QuadraticProbingHashTable(int capacity) {
        currentSize = 0;
        maxSize = capacity;
        keys = new String[maxSize];
        vals = new String[maxSize];
    }

    public void makeEmpty() {
        currentSize = 0;
        keys = new String[maxSize];
        vals = new String[maxSize]; // Fixed: corrected the
initialization
    }

    public int getSize() {
        return currentSize;
    }

    public boolean isFull() {
        return currentSize == maxSize;
    }

    public boolean isEmpty() {
        return getSize() == 0;
    }

    public boolean contains(String key) {
        return get(key) != null;
    }

    private int hash(String key) {
        return key.hashCode() % maxSize;
    }
```

```
    public void insert(String key, String val) {
        int tmp = hash(key);
        int i = tmp, h = 1;

        do {
            if (keys[i] == null) {
                keys[i] = key;
                vals[i] = val;
                currentSize++;
                return;
            }
            if (keys[i].equals(key)) {
                vals[i] = val;
                return;
            }
            i = (i + h * h++) % maxSize; // Fixed: corrected
increment logic
        } while (i != tmp);
    }

    public String get(String key) {
        int i = hash(key), h = 1;

        while (keys[i] != null) {
            if (keys[i].equals(key))
                return vals[i];
            i = (i + h * h++) % maxSize; // Fixed: corrected
increment logic
        }
        return null;
    }

    public void remove(String key) {
        if (!contains(key))
            return;

        int i = hash(key), h = 1;
        while (!key.equals(keys[i])) {
```

```
            i = (i + h * h++) % maxSize; // Fixed: corrected
increment logic
        }
        keys[i] = vals[i] = null;

        // Rehashing keys
        for (i = (i + h * h++) % maxSize; keys[i] != null; i = (i + h
* h++) % maxSize) {
            String tmp1 = keys[i], tmp2 = vals[i];
            keys[i] = vals[i] = null;
            currentSize--;
            insert(tmp1, tmp2);
        }
        currentSize--;
    }

    public void printHashTable() {
        System.out.println("\nHash Table: ");
        for (int i = 0; i < maxSize; i++) {
            if (keys[i] != null) {
                System.out.println(keys[i] + " " + vals[i]);
            }
        }
        System.out.println();
    }
}

public class QuadraticProbingHashTableTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Hash Table Test\n\n");
        System.out.print("Enter size: ");
        QuadraticProbingHashTable qpht = new
QuadraticProbingHashTable(scan.nextInt());
        char ch;

        do {
            System.out.println("\nHash Table Operations\n");
```

```
            System.out.println("1. insert");
            System.out.println("2. remove");
            System.out.println("3. get");
            System.out.println("4. clear");
            System.out.println("5. size");
            System.out.print("Choose an operation: ");
            int choice = scan.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Enter key and value: ");
                    qpht.insert(scan.next(), scan.next());
                    break;
                case 2:
                    System.out.println("Enter key: ");
                    qpht.remove(scan.next());
                    break;
                case 3:
                    System.out.println("Enter key: ");
                    System.out.println("Value = " +
qpht.get(scan.next()));
                    break;
                case 4:
                    qpht.makeEmpty();
                    System.out.println("Hash Table Cleared\n");
                    break;
                case 5:
                    System.out.println("Size = " + qpht.getSize());
                    break;
                default:
                    System.out.println("Wrong Entry\n");
                    break;
            }

            qpht.printHashTable();
            System.out.println("\nDo you want to continue (Type y or
n) ");
            ch = scan.next().charAt(0);
```

```
        } while (ch == 'Y' || ch == 'y');
    }
}
```

8. Sorting Array

A. Program Inspection

1. **Errors identified:**
2. **Error 1**: The class name "Ascending Order" contains an extra space and an underscore. It should be corrected to "AscendingOrder."
3. **Error 2**: The first nested for loop has an incorrect condition `for (int i = 0; i ¿= n; i++);`, which should be modified to `for (int i = 0; i < n; i++)`.
4. **Error 3**: There is an extra semicolon (;) after the first nested for loop that should be removed.
5. The most effective categories of program inspection would be **Category A: Syntax Errors** and **Category B: Semantic Errors**, as both syntax and semantic issues are present in the code.
6. Program inspection can identify and fix syntax errors and some semantic issues. However, it may not detect logical errors affecting the program's behavior.
7. The program inspection technique is worth applying to fix syntax and semantic errors, but debugging is necessary to address any logic errors.

B. Debugging

1. There are three errors in the program as identified above.
2. To fix these errors, set breakpoints and step through the code, focusing on the class name, loop conditions, and unnecessary semicolon.
3. The corrected executable code is as follows:

```
import java.util.Scanner;

public class AscendingOrder {
    public static void main(String[] args) {
        int n, temp;
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of elements you want
in the array: ");
        n = s.nextInt();
        int a[] = new int[n];
```

```
        System.out.println("Enter all the elements:");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }

        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (a[i] > a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }

        System.out.print("Ascending Order: ");
        for (int i = 0; i < n - 1; i++) {
            System.out.print(a[i] + ", ");
        }
        System.out.print(a[n - 1]);
    }
}
```

9. Stack Implementation

A. Program Inspection

1. **Errors identified:**
2. **Error 1**: The `push` method has a decrement operation on the `top` variable (`top–`) instead of an increment operation. It should be corrected to `top++` to push values correctly.
3. **Error 2**: The `display` method has an incorrect loop condition `for(int i=0; i ¿ top; i++)`. The condition should be `for (int i = 0; i <= top; i++)` to display the elements correctly.
4. **Error 3**: The `pop` method is missing in the `StackMethods` class. It should be added to provide a complete stack implementation.

5. The most effective category of program inspection would be **Category A: Syntax Errors**, as there are syntax errors in the code. Additionally, **Category B: Semantic Errors** can help identify logic and functionality issues.
6. The program inspection technique is valuable for identifying and fixing syntax errors, but additional inspection is needed to ensure logic and functionality are correct.

B. Debugging

1. There are three errors in the program, as identified above.
2. To fix these errors, set breakpoints and step through the code, focusing on the `push`, `pop`, and `display` methods. Correct the `push` and `display` methods and add the missing `pop` method for complete stack implementation.
3. The corrected executable code is as follows:

```
public class StackMethods {
    private int top;
    int size;
    int[] stack;

    public StackMethods(int arraySize) {
        size = arraySize;
        stack = new int[size];
        top = -1;
    }

    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack is full, can't push a value");
        } else {
            top++;
            stack[top] = value;
        }
    }

    public void pop() {
        if (!isEmpty()) {
            top--;
        } else {
            System.out.println("Can't pop...stack is empty");
        }
```

```
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public void display() {
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
}
```

10. Tower of Hanoi

A. Program Inspection

1. **Errors identified:**
2. **Error 1**: In the line `doTowers(topN ++, inter–, from+1, to+1)`, there are errors in the increment and decrement operators. It should be corrected to `doTowers(topN - 1, inter, from, to)`.
3. The most effective category of program inspection would be **Category B: Semantic Errors** because the errors in the code are related to logic and functionality.
4. The program inspection technique is worth applying to identify and fix semantic errors in the code.

B. Debugging

1. There is one error in the program, as identified above.

To fix this error, you need to replace the line:
`doTowers(topN ++, inter–, from+1, to+1);`
with the correct version:
`doTowers(topN - 1, inter, from, to);`

2. The corrected executable code is as follows:

```
public class MainClass {
```

```
    public static void main(String[] args) {
        int nDisks = 3;
        doTowers(nDisks, 'A', 'B', 'C');
    }

    public static void doTowers(int topN, char from, char inter, char
to) {
        if (topN == 1) {
            System.out.println("Disk 1 from " + from + " to " + to);
        } else {
            doTowers(topN - 1, from, to, inter);
            System.out.println("Disk " + topN + " from " + from + "
to " + to);
            doTowers(topN - 1, inter, from, to);
        }
    }
}
```