Appendix A

Projects

A.1 Minor Project 1: Menu Based Calculation System

Learning Objectives

The designing of the Menu Based Calculation System project will help the students to:

- Create C++ classes with static functions
- Generate and call static functions
- Use the functions of **Math.h** header file
- Develop and display the main menu and its submenus

Understanding the Menu Based Calculation System

The Menu Based Calculation System project is aimed at performing different types of calculations including normal and scientific calculations. In this project, two calculators, Standard and Scientific, are used for performing the calculations. The Standard calculator helps in performing simple calculations such as addition, multiplication, etc. while the Scientific calculator helps in performing mathematical operations such as finding the square or cube of a number.

The first screen contains a menu from which you can select the type of calculator: Standard, or Scientific. The first screen also provides the Quit option to terminate the execution of the application. Figure A.1 shows the first screen of the menu based calculation system.

To select a calculator, enter the integer corresponding to the calculator name. For instance, if you select **1**, the Standard calculator will open up, while selecting **2** will open the Scientific calculator.

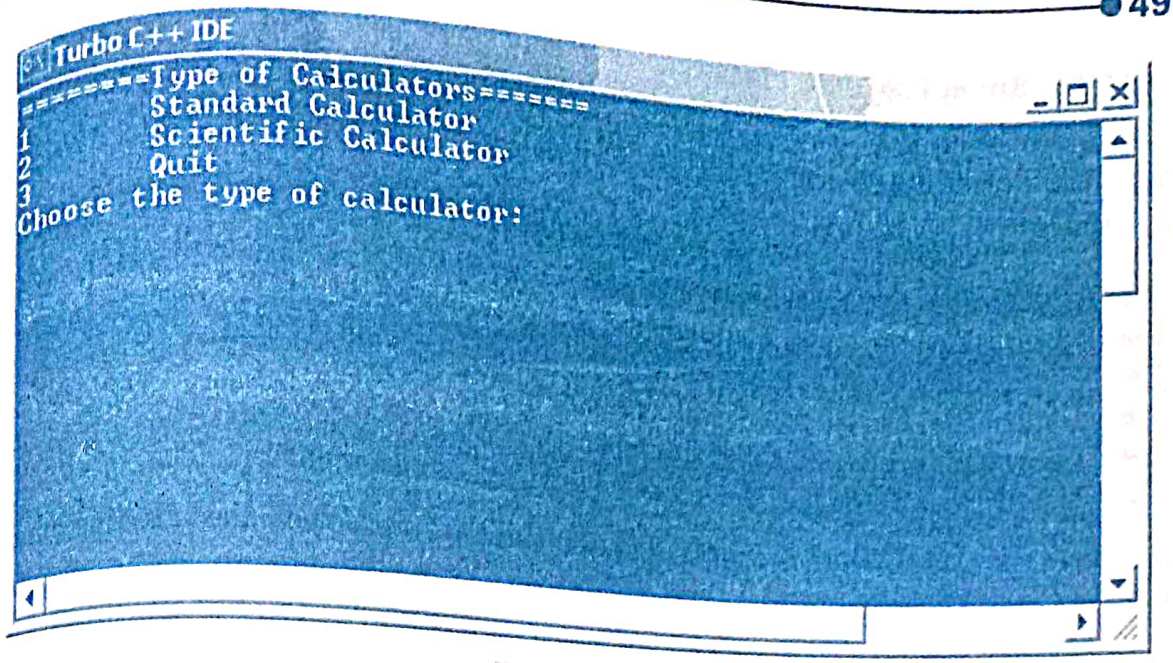

Fig. A.1

Developing the Menu Based Calculation System

The code of the calculator application mainly comprises of two classes **stand_calc** and **scien_calc**. The stand_calc class helps to perform standard calculations. The scien_calc class, on the other hand, helps to perform scientific calculations. Both classes contain static functions so as to ensure that these functions can be called in the main function through class name.

Creating the stand_calc class

The stand_calc class aims at performing specific tasks related to standard calculations. These tasks are:

- Adding two numbers
- Subtracting the second number from the first number
- Multiplying two numbers
- Dividing the first number by the second number
- Modulus of the first number by the second number

To perform the above-mentioned tasks, the stand_calc class implements the following member functions:

Functions	*Description*
Addition	Returns the addition of two input numbers.
Subtraction	Returns the subtraction of two numbers accepted as input from the user.
Multiplication	Returns the multiplication of two numbers accepted as input from the user.
Division	Returns the output obtained after performing the division operation on the input numbers.
Modulus	Returns the output obtained after performing the modulus operation on the input numbers.

Creating the scien_calc class

You need to create the scien_calc class to perform tasks related to scientific calculations, which include finding the square or cube of a number, etc. The scien_calc class performs the following tasks:

- Determines the square of a number
- Determines the cube of a number
- Determines the first number to the power of the second number
- Determines the square root of a number
- Determines the factorial of a number
- Determines the value of sin, cos and tan by passing a number

To perform the above-mentioned tasks, the scien_calc class implements the following member functions:

Functions	*Description*
Square	Accepts a number and returns the square of that number
Cube	Accepts a number and returns the cube of that number
Power	Accepts two numbers and returns the first number to the power of the second number
sq_root	Accepts a number and returns its square root
Fact	Returns the factorial of an input number
sin_func	Returns the sin value of an input number
cos_func	Returns the cos value of an input number
tan_func	Returns the tan value of an input number

Calc

```
/* calc.cpp is a calculator. Initially, it displays a main menu to choose the calculator
type. If a user chooses Standard calculator, then a menu appears for standard calculator
options. If a user chooses Scientific calculator, then a menu appears for scientific
calculator options and the last option is to Quit.
In standard calculator, options are to add, subtract, multiply etc. and in scientific
calculator, options are power, factorial, square root, etc.
In this program, preprocessor are defined for new calculation and old calculation. New
calculation will accept an operand whereas in old calculation, one operand is already
assumed from the result of previous calculation.
Exception handling is not implemented in this project, so do not enter a string when
system asks you for a number.
    */
    //File including and preprocessor declaration
    #include <iostream.h>
    #include <conio.h>
    #include <math.h>
```

```
#include <stdlib.h>
#define new_cal 1
#define old_cal 0
//stand_calc class to define standard calculator functions
class stand_calc
{
  /*protyping of standard calculator functions. These functions are static, therefore
calling of these functions is possible with the name of the class. There is no need
to create an object of the class. */
  public:
  static double addition(double,double);
  static double subtract(double,double);
  static double multiplication(double,double);
  static double division(double ,double *);
  static double modulus(double *,double *);
};
//scien_calc class to define scientific calculator functions
class scien_calc
{
  public:
  static double square(double);
  static double cube(double);
  static double power(double,double);
  static double sq_root(double);
  static long int fact(double);
  static double sin_func(double);
  static double cos_func(double);
  static double tan_func(double);
};
//addition function will add two numbers
double stand_calc::addition(double a, double b)
{
  return(a+b);
}
//subtract function will subtract the second number from the first number
double stand_calc::subtract(double a, double b)
{
  return(a-b);
}
//multiplication function will multiply two numbers
double stand_calc::multiplication(double a, double b)
{
  return(a*b);
}
```

```
/*division function will divide the first number by the second number. This function
accepts two arguments, one is copy of a variable and another is pointer type because
if accepting divisor is zero, then this function will show a message to enter the
divisor again. Using pointer means that the entered value of the divisor for this
function should be updated at the main function also.*/
double stand_calc::division(double a, double *b)
{
   while(*b==0)
   {
          cout<<"\nCannot divide by zero.";
          cout<<"\nEnter second number again:";
          cin>>*b;
   }
   return(a/(*b));
}
/*Modulus function will divide the first number by the second number and return the
remainder part of the division. Similar to division function, it will not accept
zero in the divisor. Modulus cannot be performed on a double number, so we need to
convert it into an integer.*/
double stand_calc::modulus(double *a, double *b)
{
   while(*b==0)
   {
          cout<<"\nCannot divide by zero.";
          cout<<"\nEnter second number again:";
          cin>>*b;
   }
   //Converting double into an integer
   int x=(int)*a;
   int y=(int)*b;
   if(*a-x>0||*b-y>0)
          cout<<"\nConverting decimal number into an integer to perform modulus";
   *a=x;
   *b=y;
   return(x%y);
}
//Declaration of scien_calc class functions starts from here.
//square function of scien_calc class to return accepting number to the power 2
double scien_calc::square(double x)
{
   return(pow(x,2));
}
//cube function of scien_calc class to return accepting number to the power 3
double scien_calc::cube(double x)
{
   return(pow(x,3));
}
```

```
//power function of scien_calc class to return the first number to the power of the
second number
double scien_calc::power(double x,double y)
{
  return(pow(x,y));
}
//sq_rrot function of scien_calc class to return the square root of the entered number
double scien_calc::sq_root(double x)
{
  return(sqrt(x));
}
/*fact function of the scien_calc class to return a long integer as factorial of an
accepting number. This will convert accepting number into an integer before calculating
the factorial*/
long int scien_calc::fact(double x)
{
  int n=(int)x;
  long int f=1;
  while(n>1)
  {
        f*=n;
        n-;
  }
  return f;
}
//sin_func of the scien_calc class to return the sin value of x
double scien_calc::sin_func(double x)
{
  return(sin(x));
}
//cos_func of the scien_calc class to return the cos value of x
double scien_calc::cos_func(double x)
{
  return(cos(x));
}
//tan_func of the scien_calc class to return the tan value of x
double scien_calc::tan_func(double x)
{
  return(tan(x));
}

//Displaying the menus to enter the options and values
void main()
{
  double num1,num2,num3,temp;
  int choice1=0,choice2,flag;
  //Loop of main menu from where the program starts. It will show the menu to choose
the type of calculator.
```

```
do
{
        clrscr();
        cout<<"========Type of Calculators=======";
        cout<<"\n1\tStandard Calculator\n2\tScientific Calculator\n3\tQuit";
        cout<<"\nChoose the type of calculator:";
        cin>>choice1;
        flag=new_cal;
        //To perform an operation according to the entered option in the main menu
        switch(choice1)
        {
                case 1:
                        //Loop to display the standard calculator menu
                        do
                        {
                                clrscr();
                                cout<<"==========Standard Calculator==========";
cout<<"\n1\tAddition\n2\tSubtraction\n3\tMultiplication\n4\tDivision\n5\tModulus\n6\tReturn
to Previous Menu\n7\tQuit";
                                //Option 8 will be displayed only when working on
old calculations. Here, already a number is saved in the calculator memory.
                                if(flag==old_cal)
                                        cout<<"\n8\tClear Memory";
                                cout<<"\nChoose the type of calculation:";
                                cin>>choice2;
                                //To perform operation and call functions of the
stand_calc class
                                switch(choice2)
                                {
                                        case 1:
                                                //If a new calculation is there, then
accept the first number else previous calculation result will be the first number.
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter first number:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nFirst number is
"<<num1<<endl;
                                                }
                                                cout<<"Enter second number:";
                                                cin>>num2;
```

```
num3=stand_calc::addition(num1,num2);
"<<num2<<" is "<<num3;
continue.....";
                                        cout<<"\nAddition of "<<num1<<" and
                                        cout<<"\nPress any key to
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 2:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter first number:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nFirst number is
"<<num1<<endl;
                                        }
                                        cout<<"Enter second number:";
                                        cin>>num2;
num3=stand_calc::subtract(num1,num2);
                                        cout<<"\nSubtraction of "<<num2<<"
from "<<num1<<" is "<<num3;
                                        cout<<"\nPress any key to
continue.....";
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 3:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter first number:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nFirst number is
 "<<num1<<endl;
                                        }
                                        cout<<"Enter second number:";
                                        cin>>num2;
```

```
num3=stand_calc::multiplication(num1,num2);
and "<<num2<<" is "<<num3;
continue.....";
                                        cout<<"\nMultiplication of "<<num1<<"
                                        cout<<"\nPress any key to
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 4:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter first number:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nFirst number is
"<<num1<<endl;
                                        }
                                        cout<<"Enter second number:";
                                        cin>>num2;
num3=stand_calc::division(num1,&num2);
"<<num2<<" is "<<num3;
continue.....";
                                        cout<<"\nDivision of "<<num1<<" by
                                        cout<<"\nPress any key to
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 5:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter first number:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nFirst number is
"<<num1<<endl;
                                        }
                                        cout<<"Enter second number:";
                                        cin>>num2;
```

```
num3=stand_calc::modulus(&num1,&num2);
"<<num2<<" is "<<num3;
continue.....";
                                        cout<<"\nModulus of "<<num1<<" by
                                        cout<<"\nPress any key to
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 6:
                                        cout<<"\nReturning to previous menu.";
continue.....";                         cout<<"\nPress any key to
                                        getch();
                                        break;
                                case 7:
                                        cout<<"\nQuitting.....";
continue.....";                         cout<<"\nPress any key to
                                        getch();
                                        exit(0);
                                case 8:
                                        //If a new calculation is going on
then 8 is an invalid option, else 8 is an option to start a new calculation
                                        if(flag==new_cal)
                                        {
                                                cout<<"\nInvalid choice.";
continue.....";                                 cout<<"\nPress any key to
                                                getch();
                                        }
                                        else
                                        {
                                                temp=0;
                                                flag=new_cal;
                                        }
                                        break;
                                default:
                                        cout<<"\nInvalid choice.";
continue.....";                         cout<<"\nPress any key to
                                        getch();
                                        break;
                        }
                }while (choice2!=6);
                break;
```

```
                case 2:
                        //Loop to display scientific calculator menu
                        do
                        {
                                clrscr();
                                cout<<"==========Scientific Calculator==========";
cout<<"\n1\tSquare\n2\tCube\n3\tPower\n4\tFactorial\n5\tSin\n6\tCos\n7\tTan\n8\tReturn
to previous menu\n9\tQuit";
                                if(flag==old_cal)
                                        cout<<"\n10\tClear Memory";
                                cout<<"\nChoose the type of calculation:";
                                cin>>choice2;
                                switch(choice2)
                                {
                                        case 1:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter number to find
square:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nNumber     is
"<<num1<<endl;
                                                }
                                                num3=scien_calc::square(num1);
                                                cout<<"\nSquare of "<<num1<<" is
"<<num3;
                                                cout<<"\nPress any key to
continue.....";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 2:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter number to find
cube:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nNumber     is
```

```
"<<num1<<endl;

                                        }
                                        num3=scien_calc::cube(num1);
                                        cout<<"\nCube of "<<num1<<" is

"<<num3;

                                        cout<<"\nPress any key to

continue.....";

                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 3:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter first number

for base to find power:";

                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nFirst number is

"<<num1<<endl;

                                        }
                                        cout<<"Enter second number for power

to find power:";

                                        cin>>num2;
                                        num3=scien_calc::power(num1,num2);
                                        cout<<"\n"<<num1<<" to the power

"<<num2<<" is "<<num3;

                                        cout<<"\nPress any key to

continue.....";

                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 4:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter number to find

factorial:";

                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nNumber to find
```

```
factorial is "<<num1<<endl;
                                        }
                                        long int num4=scien_calc::fact(num1);
                                        cout<<"\nFactorial of "<<num1<<" is

"<<num4;
                                        cout<<"\nPress any key to

continue.....";
                                        getch();
                                        temp=num4;
                                        flag=old_cal;
                                        break;
                                case 5:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter number to find

sin value:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nNumber for sin value

is "<<num1<<endl;
                                        }
                                        num3=scien_calc::sin_func(num1);
                                        cout<<"\nSin value of "<<num1<<" is

"<<num3;
                                        cout<<"\nPress any key to

continue.....";
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 6:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter number to find

cos value:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nNumber for cos value

is "<<num1<<endl;
                                        }
                                        num3=scien_calc::cos_func(num1);
                                        cout<<"\nCos value of "<<num1<<" is
```

```
"<<num3;
continue.....";
                                        cout<<"\nPress any key to
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 7:
                                        if (flag==new_cal)
                                        {
                                                cout<<"Enter number to find
tan value:";
                                                cin>>num1;
                                        }
                                        else
                                        {
                                                num1=temp;
                                                cout<<"\nNumber for tan value
is "<<num1<<endl;
                                        }
                                        num3=scien_calc::tan_func(num1);
                                        cout<<"\nTan value of "<<num1<<" is
"<<num3;
                                        cout<<"\nPress any key to
continue.....";
                                        getch();
                                        temp=num3;
                                        flag=old_cal;
                                        break;
                                case 8:
                                        cout<<"\nReturning to previous menu.";
                                        cout<<"\nPress any key to
continue.....";
                                        getch();
                                        break;
                                case 9:
                                        cout<<"\nQuitting.....";
                                        cout<<"\nPress any key to
continue.....";
                                        getch();
                                        exit(0);
                                case 10:
                                        if(flag==new_cal)
                                        {
                                                cout<<"\nInvalid choice.";
                                                cout<<"\nPress any key to
```

```
                                                    getch();
    continue.....";
                                                }
                                                else
                                                {
                                                    temp=0;
                                                    flag=new_cal;
                                                }
                                                break;
                                            default:
                                                cout<<"\nInvalid choice.";
                                                cout<<"\nPress any key to

    continue.....";                             getch();
                                                break;

                                    }
                                }while (choice2!=8);
                                break;
                        case 3:
                                cout<<"\nQuitting.....";
                                cout<<"\nPress any key to continue.....";
                                getch();
                                break;
                        default:
                                cout<<"\nInvalid Choice.";
                                cout<<"\nPress any key to continue.....";
                                getch();
                                break;

                }
        }while (choice1!=3);
}
```

A.2 Major Project 1: Banking System

Learning Objectives

The designing of the Banking System project helps the students to:

- Create C++ classes and call the functions declared in the classes
- Develop and display main menu and its submenus
- Change the menu options during runtime
- Programmatically create files using File System objects
- Perform file transactions such as Updation, Deletion and Display from files
- Use **iomanip** header file in C++ to display formatted output of data using **setw()** function for setting width of the text to be displayed.

Understanding the Banking System Project

The Banking System application helps maintain the data related to the customers and performs the typical banking transactions. In the Banking System application, **fileobject** is used to create two data files, which have the extension .dat. The data related to a customer is stored in the newrecords.dat data file. The data related to transactions, such as withdrawal and deposit are stored in the transaction.dat data file. The following figure shows the main menu of the Banking System application.

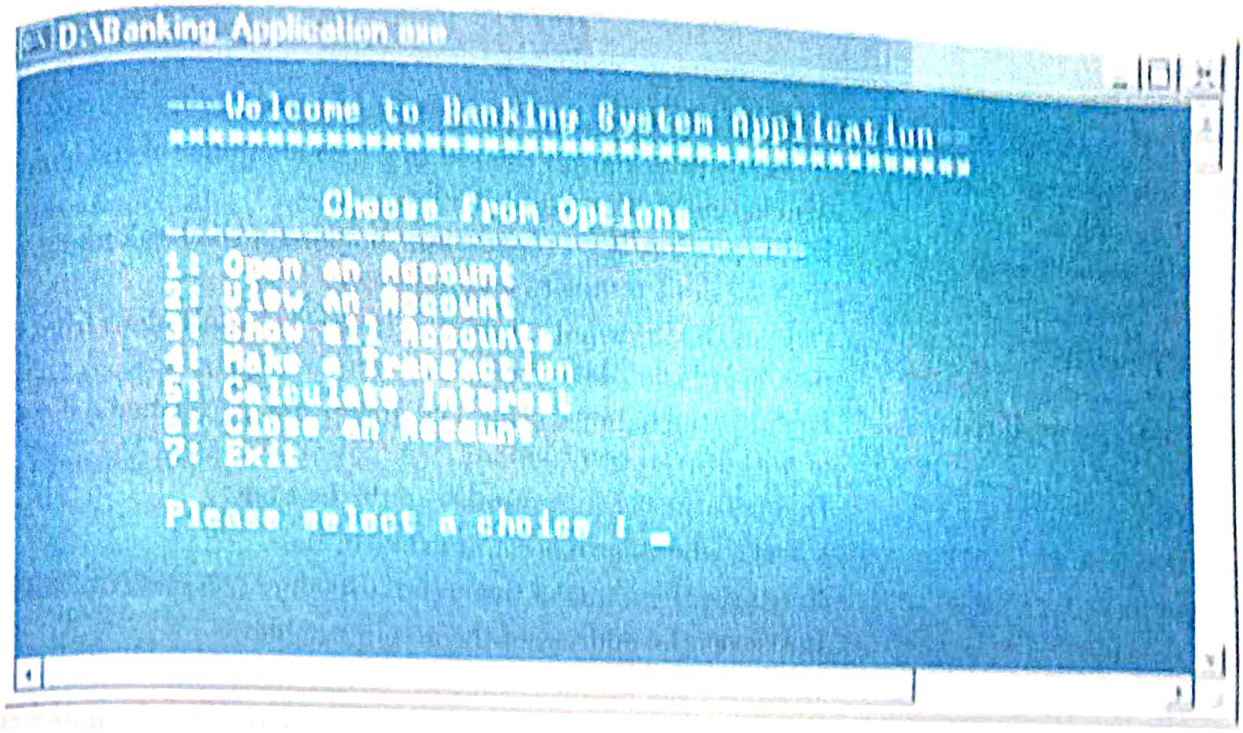

Fig. A.2

Developing the Banking System

The development of Banking System application involves the creation of the following classes:

- menus
- dispRecords
- accountTransactions

Creating the menus Class

You need to create the class **menus** to implement the functionality of displaying a main menu and closing the account before quitting the Banking System application. To create the class menus, you need to define the member functions, **showmenu** and **closemenu** and the variables required for displaying the main menu of the Banking System application. The showmenu member function helps to display the main menu to the users of the Banking System application. The closemenu member function helps to display the Closing Account menu when the user selects the Close an Account option from the main menu.

Creating the dispRecords Class

You need to create the **dispRecords** class to implement the functionality of displaying the information related to the customers of a bank and their accounts. In the dispRecords class, data related to customers is retrieved from the newrecords.dat data file for displaying customer information or adding and closing of customer accounts. You can create the dispRecords class by defining the variables required for displaying customer and account information and the member functions such as **displayCustomer** and **deleteAccount**. The following table lists the member functions that need to be defined in the class dispRecords:

Functions	*Descriptions*
addDetails(int, char name[30], char address[60], float)	Adds the information related to a new customer of the bank who becomes an account holder.
displayCustomers(void)	Displays a list of all the account holders of the bank along with their account numbers and balance.
deleteAccount(int)	Deletes the information related to the account holder from the newrecords.dat data file.
updateBalance(int, float)	Updates the balance after a customer has performed a deposit or withdrawal transaction.
lastAccount()	Displays the account number of the last entry.
accountExists(int)	Checks whether an account exists or not.
getName(int)	Retrieves the name of the account holder.
getAddress(int)	Retrieves the address of the account holder.
getBalance(int)	Retrieves the balance of the account holder.
getRecord(int)	Returns the record number from the newrecords.dat data file when an employee of the bank enters the account number related to an account holder.
display(int)	Displays all the information related to an account holder from the newrecords.dat file on the basis of specified account number.

Creating the accountTransactions Class

You need to create the **accountTransactions** class so that transactions related to an account can be performed. The data related to the transactions are stored in the transaction.dat data file. The accountTransactions class also uses some member functions defined in the dispRecords class. In the class accountTransactions, the Object Oriented Programming (OOP) concepts of Polymorphism are used to manipulate data, which need to be stored in the transaction.dat data file. You can create the accountTransactions class by defining variables and member functions, which include **new_account** and **showAccount**. The following table lists the member functions of the accountTransactions class:

Functions	Descriptions
new_account(void)	Validates the information related to a new customer and adds the information to the transaction.dat data file using the addDetails member function.
closeAccount()	Closes the account of an account holder after verifying the account number.
showAccount(int)	Displays the headings Customer Name, Deposit and Withdrawal, Interest and Balance.
display_account(void)	Displays the data related to a specific account holder.
deleteAccount(int)	Deletes the data related to a transaction from the transaction.dat data file on the basis of the account number of that account holder.
transaction(void)	Helps to perform deposit and withdrawal transactions.
dateDiffer(int, int, int, int, int, int)	Checks the current and account creation dates. If the account in the bank has completed one year, then interest for that account is calculated.
getInterest(int, float)	Generates interest when one year has completed for a particular account.
showInterest(void)	Displays the interest generated using the getInterest member function. The showInterest member function also helps to update the balance of the account holder.

Banking_Application

```
/** A Banking System with normal transactions **/
  #include <iostream.h>
  #include <fstream.h>
  #include <process.h>
  #include <string.h>
  #include <stdlib.h>
  #include <stdio.h>
  #include <ctype.h>
  #include <conio.h>
  #include <dos.h>
  #include <iomanip.h>

  // The Menus Class displays the Menu
  class Menus
  {
    public :
                  void showmenu(void) ;
```

```
    private :
                    void closemenu(void) ;
};

// The Class displays all the Customer Account related functions
class dispRecords
{
  public :
                    void addDetails(int, char name[30], char address[60], float) ;
                    void displayCustomers(void) ;
                    void deleteAccount(int) ;
                    void updateBalance(int, float) ;
                    void updateCustomer(void) ;
                    int  lastAccount(void) ;
                    int  accountExists(int) ;
                    char *getName(int);
                    char *getAddress(int);
                    float getBalance(int) ;
                    int  getRecord(int) ;
                    void display(int) ;
                    void  displayList(void) ;
                    int   AccountNumber ;
                    char  name[50], address[50] ;
                    float intBalance ;
};

// The Class has all the transaction related methods
class accountTransactions
{
  public :
                   void new_account(void);
                   void closeAccount(void);
                   void display_account(void);
                   void transaction(void);
                   void addDetails(int, int, int, int, char, char typeTransaction[15],
float, float, float);
                  void  deleteAccount(int);
                  int   dateDiffer(int, int, int, int, int, int);
                  float getInterest(int, float);
                  void  display(int);
                  void  showAccount(int);
                  int   AccountNumber;  //variable for Account Number
                  char  trantype[10];  // variable of cheque or cash input or output
                  int   dday, mmonth, yyear;  // transaction date
                  char  transactions;         // type of transactions - Deposit or
Withdrawal of Amount
```

```
                float intInterest, intAmount, intBalance;
                static float calcInterest;
                void showInterest(void);//added
};

// showmenu() method to display the Main Menu in the application
void Menus :: showmenu(void)
{
  char choice;
  while (1)
  {
        clrscr();
        cout<<"\n          --Welcome to Banking System Application--    \n";
        cout<<" ****************************************\n\n";
        cout<<"              Choose from Options \n";
        cout<<"              ------------------ \n";
        cout <<"             1: Open an Account\n";
        cout <<"             2: View an Account    \n";
        cout <<"             3: Show all Accounts  \n";
        cout <<"             4: Make a Transaction \n";
        cout <<"             5: Calculate Interest\n";
        cout <<"             6: Close an Account\n";
        cout <<"             7: Exit\n\n";
        choice = getche();

                if (choice == '1')
                {

                        accountTransactions objAT;
                        objAT.new_account();
                }
                else
                if (choice == '2')
                {

                        accountTransactions objAT;
                        objAT.display_account();
                }
                else
                if (choice == '3')
                {

                        dispRecords newRec;
                        newRec.displayCustomers();
                }
                else
                if (choice == '4')
```

```
                {
                        accountTransactions objAT;
                        objAT.transaction();

                }
                else
                if (choice == '5')
                {
                        accountTransactions objAT;
                        objAT.showInterest();

                }
                else
                if (choice == '6') {
                        closemenu();

                }
                else
                if (choice == '7') {
                        cout<<"\n      Thanks for using this application. Please
press any key to exit.\n";
                        getch();
                        break ;

                }
        }
}

// closemenu() method displays the Closing of the Account of the Customer in the
Application
void Menus :: closemenu(void)
{
  char choice;
  while (1)
  {
        clrscr() ;
        cout<<"        -Close Menu- \n" ;
        cout<<"        ********************* \n" ;
        cout <<"       1: Close/Delete an Account\n" ;
        cout <<"       0: Exit from this menu\n\n" ;
        cout <<"       Select a Choice: " ;
        choice = getche() ;

              if (choice == '1')
              {
                      accountTransactions at ;
                      at.closeAccount() ;
                      break ;

              }
              else
              if (choice == '0')
```

```
                {
                        cout<<"\n       You have entered 0 to go back to the previous
Menu. \n";
                        getch();
                        break ;
                }
        }

}

// lastAccount() method returns the Last Account Number from the newrecords.dat file
int dispRecords :: lastAccount(void)
{
  fstream filename ;
  filename.open("newrecords.dat", ios::in) ;
  filename.seekg(0,ios::beg) ;
  int count=0 ;
  while (filename.read((char *) this, sizeof(dispRecords)))
        count = AccountNumber ;
  filename.close() ;
  return count ;
}

// getRecord() method returns the record number from the newrecords.dat file when
a banking staff enters the Account Number
int dispRecords :: getRecord(int retrieve_AccNo)
{
  fstream filename;
  filename.open("newrecords.dat", ios::in) ;
  filename.seekg(0,ios::beg) ;
  int count=0 ;
  while (filename.read((char *) this, sizeof(dispRecords)))
  {
        count++ ;
        if (retrieve_AccNo == AccountNumber)
              break ;
        /*keep on counting the record till the Account Number is found and exit
from the newrecords.dat file. */
  }
  filename.close() ;
  return count ;
}

// display() method displays all the details of the Account Number from the
newrecords.dat file
void dispRecords :: display(int retrieve_AccNo)
```

```
{
   int record ;
   record = getRecord(retrieve_AccNo) ;
   fstream filename ;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::end);
   int location;
   location = (record) * sizeof(dispRecords);
   filename.seekp(location);

   while (filename.read((char *) this, sizeof(dispRecords)))
   {
          if (retrieve_AccNo == AccountNumber)
          {
                 cout <<"\n     ACCOUNT NO. : " <<AccountNumber ;
                 cout <<"\n     Name    : "<<name ;
                 cout <<"\n     Address : " <<address ;
                 cout <<"\n     Balance : " <<intBalance ;
                 break ;
          }
   }
   filename.close() ;
}

// getName() method returns the Account Holder's Name from the newrecords.dat file
char *dispRecords :: getName(int retrieve_AccNo)
{
   fstream filename;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::beg);
   char retrieve_CustName[30];

          while (filename.read((char *) this, sizeof(dispRecords)))
          {
                 if (AccountNumber == retrieve_AccNo)
                 {
                        strcpy(retrieve_CustName,name);
                 }
          }
          filename.close();
          return retrieve_CustName;
}

// getAddress() method returns the Address of the Account Holder from the newrecords.dat
file
char *dispRecords :: getAddress(int retrieve_AccNo)
```

```
{
   fstream filename;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::beg);
   char retrieve_Address[60];

   while (filename.read((char *) this, sizeof(dispRecords)))
   {
          if (AccountNumber == retrieve_AccNo)
          {
                 strcpy(retrieve_Address,address);
          }
   }
   filename.close();
   return retrieve_Address;

}

/* getBalance() method returns the Balance of the Account Holder from the newrecords.dat
file*/
float dispRecords :: getBalance(int retrieve_AccNo)
{
   fstream filename ;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::beg);
   float iBalance ;
   while (filename.read((char *) this, sizeof(dispRecords)))
   {
          if (AccountNumber == retrieve_AccNo)
          {
                 iBalance = intBalance;
          }
   }
   filename.close();
   return iBalance;
}

// accountExists() method checks if the Account exists in the newrecords.dat file or not.
int dispRecords :: accountExists(int retrieve_AccNo)
{
   fstream filename ;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::beg) ;
   int count=0 ;
   while (filename.read((char *) this, sizeof(dispRecords)))
   {
          if (AccountNumber == retrieve_AccNo)
```

```
            {
                    count = 1;
                    break;
            }
      }
      filename.close();
      return count;
   }

   /* displayList() method displays the output of all the Accounts in a proper format
   for the Choice 3*/
   void dispRecords :: displayList()
   {
     cout<<"
                                          \n" ;
     int day1, month1, year1 ;
     struct date dateval;
     getdate(&dateval);
     day1 = dateval.da_day ;
     month1 = dateval.da_mon ;
     year1 = dateval.da_year ;
     cout <<"\n Date: " <<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
     cout<<setw(80)<<"--------------------------------------------------------------------------------\n";
     cout<<setw(23)<<" ACCOUNT NO.";
     cout<<setw(23)<<" NAME OF PERSON";
     cout<<setw(23)<<" BALANCE\n";
     cout<<setw(80)<<"--------------------------------------------------------------------------------\n";
}

// displayCustomers() method displays all the Account Holders/Customers from the
newrecords.dat file
void dispRecords :: displayCustomers(void)
{
   clrscr() ;
   int len1;
   int row=8, check ;
   fstream filename ;

   FILE * pFile;
   pFile = fopen("newrecords.dat","r");
   if (pFile == NULL)
   {
       cout<<"\n  No Account exists. Please go back to the previous menu. \n";
          getch();
          return ;
          //fclose (pFile);

 } else {
```

```
        displayList();
        filename.open("newrecords.dat", ios::in);
        filename.seekg(0,ios::beg);
        while (filename.read((char *) this, sizeof(dispRecords)))
        {
                check = 0 ;

                cout.fill(' ');
                cout <<setw(20);
                cout.setf(ios::right,ios::adjustfield);
                cout<<AccountNumber;
                cout.fill(' ');
                cout <<setw(25);
                cout.setf(ios::internal,ios::adjustfield);
                cout<<name;

                cout <<setw(23);
                cout.setf(ios::right,ios::adjustfield);
                cout<<intBalance<<"\n" ;
                row++ ;
                if (row == 23)
                {
                        check = 1 ;
                        row = 8 ;
                        cout <<" \n\n Continue the application... \n";
                        getch() ;
                        clrscr() ;
                        displayList() ;
                }
        }
  }
  filename.close() ;
  if (!check)
  {
        cout <<"\n\n Continue the application... \n";
        getch() ;
  }
}

// addDetails() method adds new records of Account Holders/Customers in the
newrecords.dat file
void dispRecords :: addDetails(int retrieve_AccNo, char retrieve_CustName[30],
char retrieve_Address[60], float iBalance)
{
  AccountNumber = retrieve_AccNo ;
  strcpy(name,retrieve_CustName) ;
  strcpy(address,retrieve_Address) ;
  intBalance = iBalance ;
```

```
    fstream filename ;
    filename.open("newrecords.dat", ios::out | ios::app) ;
    filename.write((char *) this, sizeof(dispRecords)) ;
    filename.close() ;
}

// deleteAccount() method deletes the particular record from the newrecords.dat
file on the basis of the Account Number.
void dispRecords :: deleteAccount(int retrieve_AccNo)
{
   fstream filename ;
   filename.open("newrecords.dat", ios::in) ;
   fstream temp ;
   temp.open("calculations.txt", ios::out) ;
   filename.seekg(0,ios::beg) ;
   while ( !filename.eof() )
   {
          filename.read((char *) this, sizeof(dispRecords)) ;
          if ( filename.eof() )
                 break ;
          if ( AccountNumber != retrieve_AccNo )
                 temp.write((char *) this, sizeof(dispRecords)) ;
   }
   filename.close() ;
   temp.close() ;
   filename.open("newrecords.dat", ios::out) ;
   temp.open("calculations.txt", ios::in) ;
   temp.seekg(0,ios::beg) ;
   while ( !temp.eof() )
   {
          temp.read((char *) this, sizeof(dispRecords)) ;
          if ( temp.eof() )
                 break ;
          filename.write((char *) this, sizeof(dispRecords)) ;
   }
   filename.close() ;
   temp.close() ;
}

// updateBalance() method updates the balance of the Account Number after a transaction
is done in the newrecords.dat file
void dispRecords :: updateBalance(int retrieve_AccNo, float iBalance)
{
   int record ;
   record = getRecord(retrieve_AccNo) ;
   fstream filename ;
   filename.open("newrecords.dat", ios::out | ios::ate) ;
```

```
    intBalance = iBalance ;
    int location ;
    location = (record-1) * sizeof(dispRecords) ;
    filename.seekp(location) ;
    filename.write((char *) this, sizeof(dispRecords)) ;
    filename.close() ;
}

// addDetails() method adds the details of a transaction in the transactions.dat file
void accountTransactions :: addDetails(int retrieve_AccNo, int day1, int month1, int
year1, char t_tran, char typeTransaction[10], float interest_accrued, float t_amount,
float iBalance)
{
    fstream filename ;
    filename.open("transactions.dat", ios::app) ;
    AccountNumber = retrieve_AccNo ;
    dday = day1 ;
    mmonth = month1 ;
    yyear = year1 ;
    transactions = t_tran ;
    strcpy(trantype,typeTransaction) ;
    intInterest = interest_accrued ;
    intAmount = t_amount ;
    intBalance = iBalance ;
    filename.write((char *) this, sizeof(accountTransactions)) ;
    filename.close();
}

// deleteAccount() method deletes the record of a transaction from the transactions.dat
file
void accountTransactions :: deleteAccount(int retrieve_AccNo)
{
    fstream filename ;
    filename.open("transactions.dat", ios::in) ;
    fstream temp ;
    temp.open("calculations.txt", ios::out) ;
    filename.seekg(0,ios::beg) ;
    while ( !filename.eof() )
    {
            filename.read((char *) this, sizeof(accountTransactions)) ;
            if ( filename.eof() )
                    break ;
            if ( AccountNumber != retrieve_AccNo )
                    temp.write((char *) this, sizeof(accountTransactions)) ;
    }
    filename.close() ;
    temp.close() ;
```

```
    filename.open("transactions.dat", ios::out) ;
    temp.open("calculations.txt", ios::in) ;
    temp.seekg(0,ios::beg) ;
    while ( !temp.eof() )
    {
            temp.read((char *) this, sizeof(accountTransactions)) ;
            if ( temp.eof() )
                    break ;
            filename.write((char *) this, sizeof(accountTransactions)) ;
    }
    filename.close() ;
    temp.close() ;
}

// new_account() method adds a new record in the newrecords file and transaction.dat
files(choice 1)
void accountTransactions :: new_account(void)
{
  char choice ;
  int i, check ;
  clrscr() ;
  dispRecords newRec ;
  cout <<"       Please press 0 to go back to previous menu. \n" ;
                                                          \n";
  cout<<"
  cout<<"        -Open a New Bank Account-      \n";
  cout<<"        ********************  \n";
  int day1, month1, year1 ;
  struct date dateval;
  getdate(&dateval);
  day1 = dateval.da_day ;
  month1 = dateval.da_mon ;
  year1 = dateval.da_year ;
  int retrieve_AccNo ;
  retrieve_AccNo = newRec.lastAccount() ;
  retrieve_AccNo++ ;

  if (retrieve_AccNo == 1)
  {
        newRec.addDetails(retrieve_AccNo,"Ravi","Delhi",1.1) ;
        newRec.deleteAccount(retrieve_AccNo) ;
        addDetails(retrieve_AccNo,1,1,1997,'D',"default value",1.1,1.1,1.1) ;
        deleteAccount(retrieve_AccNo) ;
  }
  char retrieve_CustName[30], tran_acc[10], retrieve_Address[60] ;
  float t_bal, iBalance ;
  cout <<"       Date : "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n" ;
  cout <<"       Account no. # " <<retrieve_AccNo;
```

```
do
{
        cout <<"\n\n   Please enter the Name of the Account Holder    : ";
        check = 1;
        gets(retrieve_CustName);
        if (retrieve_CustName[0] == '0')
        {
                cout<<"\n\t    Invalid Customer Name.";
                getch();
                return;
        }
        strupr(retrieve_CustName);
        if (strlen(retrieve_CustName) == 0 || strlen(retrieve_CustName) > 30)
        {
                check = 0;
                cout<<"\t\n    Customer Name is either blank or its length is
greater than 30 characters.\n";
                getch();
        }
} while (!check);

do
{
        cout <<"\n     Please enter the Account Holder's Address : ";
        check = 1;
        gets(retrieve_Address);
        if (retrieve_Address[0] == '0')
        {
                cout<<"\n\t    Invalid Customer Address.";
                getch();
                return;
        }
        strupr(retrieve_Address);
        if (strlen(retrieve_Address) < 1 || strlen(retrieve_Address) > 60)
        {
                check = 0 ;
                cout<<"\t\n    Customer Address is either blank or its length is
greater than 60 characters.\n" ;
                getch() ;
        }
} while (!check) ;

do
{
        char chr_VerifyingPerson[30] ;
        cout <<"\n     Please enter the Name of the Verifying Person of the
Account Holder : ";
        check = 1 ;
```

```
            gets(chr_VerifyingPerson);
            if (chr_VerifyingPerson[0] == '0')
            {
                    cout<<"\n\t    Invalid Verifying Person Name.";
                    getch();
                    return;
            }
            strupr(chr_VerifyingPerson) ;
            if (strlen(chr_VerifyingPerson) < 1 || strlen(chr_VerifyingPerson) > 30)
            {
                    check = 0 ;
                    cout<<"\t\n    The Verifying Person's Name is either blank or
 greater than 30 characters. Please try again.\n";
                    getch() ;
            }
    } while (!check) ;

    do
    {
            cout <<"\n  Please enter the Deposit Amount while opening a New Account : ";
            check = 1 ;
            gets(tran_acc) ;
            t_bal = atof(tran_acc) ;
            iBalance = t_bal ;
            if (strlen(tran_acc) < 1) {
                    cout<<"\n    Invalid Transaction value. Exiting from the current
Menu.\n ";
                    getch();
                    return ;
            }
            if (iBalance < 1000)
            {
                    check = 0 ;
                    cout<<"\t\n    The Minimum Deposit Amount should be Rs.1000. Please
try again. \n";
                    getch() ;
            }

    } while (!check) ;

    do
    {
            cout <<"\n     Do you want to save the record? (y/n) : " ;
            choice = getche() ;
            choice = toupper(choice) ;
```

```
  } while (choice != 'N' && choice != 'Y') ;
  if (choice == 'N' || choice == 'n')
  {
        cout<<"\n      The Customer Account is not created\n.
Please continue with the application.\n";
        getch();
        return ;

  }
  float t_amount, interest_accrued ;
  t_amount = iBalance ;
  interest_accrued = 0.0 ;
  char t_tran, typeTransaction[10] ;
  t_tran = 'D' ;
  strcpy(typeTransaction," ") ;

  newRec.addDetails(retrieve_AccNo,retrieve_CustName,retrieve_Address,iBalance) ;
  addDetails(retrieve_AccNo,day1,month1,year1,t_tran,typeTransaction,
interest_accrued,t_amount,iBalance);
  cout<<" \n\n          The New Account is successfully created.\n
Please continue with the application.\n";
  getch();
}

// showAccount() method formats the display of the records from the transactions.dat
file for a particular account(choice 2).
void accountTransactions :: showAccount(int retrieve_AccNo)
{
  cout<<"
                                                \n";

  int day1, month1, year1 ;
  struct date dateval;
  getdate(&dateval);
  day1 = dateval.da_day ;
  month1 = dateval.da_mon ;
  year1 = dateval.da_year ;
  cout<<"Date: " <<day1 <<"/" <<month1 <<"/" <<year1<<"\n" ;
  cout <<"Account no. " <<retrieve_AccNo ;
  dispRecords newRec ;

  char retrieve_CustName[30] ;
  strcpy(retrieve_CustName,newRec.getName(retrieve_AccNo)) ;
  char retrieve_Address[60] ;
  strcpy(retrieve_Address,newRec.getAddress(retrieve_AccNo)) ;

  cout<<setw(25)<<"\n Account Holder's Name :  "<<retrieve_CustName;
  cout<<"\nAddress                  :       "<<retrieve_Address<<"\n";
  cout<<setw(80)<<"\n--------------------------------------------------\n";
```

```
    cout<<setw(10)<<"Dated";
    cout<<setw(12)<<"Details";
    cout<<setw(12)<<"Deposited";
    cout<<setw(15)<<"Withdrawn";
    cout<<setw(12)<<"         ";
    cout<<setw(10)<<"Balance";
    cout<<setw(80)<<"\n--------------------------------------------------------------------------------\n";
}

// display_account() method displays records from the transactions.dat file
void accountTransactions :: display_account(void)
{
   clrscr() ;
   char t_acc[10] ;
   int tran_acc, retrieve_AccNo;
   dispRecords obj2;
   cout <<"       Press 0 to go back to previous menu.\n" ;
   cout <<"       Please enter Account No. you want to view :  " ;
   gets(t_acc);
   tran_acc = atoi(t_acc);       /* converting Account Number to integer value */
   retrieve_AccNo = tran_acc;
   if (retrieve_AccNo == 0){
          cout<<"\n      You have pressed 0 to exit. \n";
          getch();
          return ;
   }
   clrscr();
   dispRecords newRec;
   accountTransactions aa;
   int row=8, check ;
   fstream filename ;

   FILE * pFile;
   pFile = fopen("newrecords.dat","r");
   if (pFile == NULL)
   {
       cout<<"\n  No such Account Exists. Please create a New Account. \n";
          getch();
          return ;

   } else if (!newRec.accountExists(retrieve_AccNo)) {
          cout <<"\t\n   Account does not exist.\n";
          getch();
          return;
   } else {

          showAccount(retrieve_AccNo) ;
          filename.open("transactions.dat", ios::in);
```

```
        /* Reading the transaction.dat file and displaying the details of a particular
Account */
        while (filename.read((char *) this, sizeof(accountTransactions)))
        {
                if (AccountNumber == retrieve_AccNo)
                {
                        check = 0 ;
                        cout<<setw(3)<<dday<<"/"<<mmonth<<"/"<<yyear ;
                        cout <<setw(10)<<trantype ;
                        if (transactions == 'D') {
                                cout.setf(ios::right,ios::adjustfield);
                                cout <<setw(15);
                                cout<<intAmount;
                                cout <<setw(20);
                                cout<<" ";

                         } else {
                                cout.setf(ios::right,ios::adjustfield);
                                cout<<setw(25);

                                cout<<intAmount;
                                cout <<setw(10);
                                cout<<"  ";
                         }
                         cout<<setw(15);
                         cout.setf(ios::right,ios::adjustfield);
                         cout <<intBalance <<"\n";
                         row++;

                         if (row == 23)
                         {
                                check = 1 ;
                                row = 8 ;
                                cout <<"\n\n  Please continue with the application.
\n";

                                getch();
                                clrscr();
                                showAccount(retrieve_AccNo);

                         }
                }
        }
  }
  filename.close() ;
  if (!check)
  {
        cout <<"\n\n   Press any key to continue with the application. \n" ;
        getch() ;
  }
}
```

```
// dateDiffer() method displays the difference between 2 dates.
int accountTransactions :: dateDiffer(int day1, int month1, int year1, int day2,
int month2, int year2)
{
  static int monthArr[] = {31,28,31,30,31,30,31,31,30,31,30,31};      //Array of
months for storing the no. of days in each array
  int days = 0 ;
  while (day1 != day2 || month1 != month2 || year1 != year2)
  {
        /* checking if the two dates in days,months and years differ and incrementing
the number of days.*/
        days++ ;
        day1++ ;
        if (day1 > monthArr[month1-1])
        {
              day1 = 1 ;
              month1++ ;
        }
        if (month1 > 12)
        {
              month1 = 1 ;
              year1++ ;
        }
  } return days ;
}

// getInterest() function calculates interest on the balance from the transaction.dat
file
float accountTransactions :: getInterest(int retrieve_AccNo, float iBalance)
{
  fstream filename ;
  filename.open("transactions.dat", ios::in);
  dispRecords newRec;
  filename.seekg(0,ios::beg) ;
  int day1, month1, year1, month_day;
  while (filename.read((char *) this, sizeof(accountTransactions)))
  {
        if (AccountNumber == retrieve_AccNo)
        {
              day1 = dday ;
              month1 = mmonth ;
              year1 = yyear ;
              iBalance = newRec.getBalance(retrieve_AccNo);
              break ;
        }
  }
  int day2, month2, year2;
  struct date dateval;
```

```
  getdate(&dateval);
  day2 = dateval.da_day;
  month2 = dateval.da_mon;
  year2 = dateval.da_year;
  float interest_accrued=0.0;
  int yeardiff = year2 - year1;

  if ((year2<year1) || (year2==year1 && month2<month1) || (year2==year1 &&
month2==month1 && day2<day1)) {

        return interest_accrued;
  }
  month_day = dateDiffer(day1,month1,year1,day2,month2,year2);
  int months;
  if (month_day >= 30)
  {
        months = month_day/30;
  } else {
        months = month_day/30;
  }
        if(interest_accrued == 0 && yeardiff == 1) {
              interest_accrued = ((iBalance*0.5)/100) * (months);
        } else if (yeardiff > 1 && yeardiff < 25 && interest_accrued == 0) {
                    interest_accrued = ((iBalance*0.5)/100) * (months);
        } else {
              interest_accrued = 0;
        }

  filename.close();
  return interest_accrued;
}

/*Method for generating Interest and updation of the Balance and addDetails
methods.(Choice 5)*/
void accountTransactions :: showInterest(void)
{
  clrscr();
  char t_acc[10];
  int tran_acc, retrieve_AccNo, check;

  cout <<strupr("\n      Important Information: Interest should be generated only\n
once in a Year.\n\n\t If you have already generated interest for an Account,\n\t
please ignore that Account.\n\t Thank you.\n");
  cout <<"\n     Press 0 to go back to previous menu.\n" ;
  cout <<"\n     To view the transaction of the Account, please enter it: " ;
  gets(t_acc) ;
```

```
    tran_acc = atoi(t_acc) ;
    retrieve_AccNo = tran_acc ;
    if (retrieve_AccNo == 0)
            return ;
    clrscr() ;
    dispRecords newRec ;
    if (!newRec.accountExists(retrieve_AccNo))
    {
            cout <<"\t\n   Account does not exist.\n";
            getch();
            return;
    }
    cout <<"       Press 0 to go back to previous menu.\n";
                                                          \n";
    cout<<"
    cout<<"\n      -Please enter the Account no. to generate interest-  \n";
    cout<<" *****************************************\n";
    int day1, month1, year1;
    struct date dateval;
    getdate(&dateval);
    day1 = dateval.da_day;
    month1 = dateval.da_mon;
    year1 = dateval.da_year;
    cout <<"       Date : "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
    cout <<"       Account no. " <<retrieve_AccNo<<"\n";
    char retrieve_CustName[30] ;
    char retrieve_Address[60] ;
    float iBalance ;
    strcpy(retrieve_CustName,newRec.getName(retrieve_AccNo)) ;
    strcpy(retrieve_Address,newRec.getAddress(retrieve_AccNo)) ;
    iBalance = newRec.getBalance(retrieve_AccNo);

    cout <<"       Customer Name   :      " <<retrieve_CustName;
    cout <<"\n     Customer Address:      " <<retrieve_Address ;
    cout <<"\n     Bank Balance    :      " <<iBalance ;

    float interest_accrued;
    interest_accrued = getInterest(retrieve_AccNo,iBalance);
                              /* Calculation of interest of the deposit amount*/
    cout<<"\n\tInterest generated:       "<<interest_accrued;
    getch();
    iBalance = iBalance + interest_accrued;
    dispRecords obj2;
    /*Updating the Balance once Interest is generated in a year*/
    obj2.updateBalance(retrieve_AccNo, iBalance);
    /*Adding Interest as a Deposit when it is generated in a year.*/
    addDetails(retrieve_AccNo,day1,month1,year1,'D',"Interest",interest_accrued,
interest_accrued,iBalance);
}
```

```
/* this method does all the Deposit/Withdrawal transactions in the transaction.dat
file(choice 4)*/
void accounttransactions :: transaction(void)
{
  clrscr();
  char t_acc[10];
  int tran_acc, retrieve_AccNo, check;
  cout <<"        Press 0 to go back to previous menu.\n" ;
  cout <<"        to view the transaction of the Account, please enter it: " ;
  gets(t_acc) ;
  tran_acc = atoi(t_acc) ;
  retrieve_AccNo = tran_acc ;
  if (retrieve_AccNo == 0)
        return ;
  clrscr() ;
  dispRecords newRec ;
  if (!newRec.accountExists(retrieve_AccNo))
  {
        cout <<"\t\n   Account does not exist.\n";
        getch();
        return;
  }
  cout <<"        Press 0 to go back to previous menu.\n";
  cout<<"                                                    \n";
  cout<<"\n       -Make correct entry for the Transaction below-  \n";
  cout<<" ********************************************\n";
  int day1, month1, year1;
  struct date dateval;
  getdate(&dateval);
  day1 = dateval.da_day;
  month1 = dateval.da_mon;
  year1 = dateval.da_year;
  cout <<"        Date : "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
  cout <<"        Account no. " <<retrieve_AccNo<<"\n";
  char retrieve_CustName[30] ;
  char retrieve_Address[60] ;
  float iBalance;
  float interest_accrued = 0.0;
  strcpy(retrieve_CustName,newRec.getName(retrieve_AccNo)) ;
  strcpy(retrieve_Address,newRec.getAddress(retrieve_AccNo)) ;
  iBalance = newRec.getBalance(retrieve_AccNo);

  cout <<"        Customer Name    : " <<retrieve_CustName;
  cout <<"\n      Customer Address: " <<retrieve_Address ;
  cout <<"\n      Bank Balance: " <<iBalance ;
  char tranDetails, typeTransaction[10], tm[10] ;
  float t_amount, t_amt ;
```

```
    do
    {
            cout <<"\n  Please enter D for Deposit or W for Withdrawal of Amount : " ;
            tranDetails = getche() ;
            if(tranDetails == '0') {
                    cout<<"\n\n    You have pressed 0 to Exit.";
                    getch();
                    return;
            }
            tranDetails = toupper(tranDetails) ;
    } while (tranDetails != 'W' && tranDetails != 'D') ;

    do
    {
            cout <<"\n     The Transaction type is either Cash or Cheque..\n" ;
            check = 1 ;
            cout <<"       (Cash/Cheque) : " ;
            gets(typeTransaction) ;
            strupr(typeTransaction);
            if(typeTransaction[0] == '0') {
                    cout<<"\n\n    You have pressed 0 to Exit.";
                    getch();
                    return;
            }
            if (strlen(typeTransaction) < 1 || (strcmp(typeTransaction,"CASH") &&
strcmp(typeTransaction,"CHEQUE")) )
            {
                    check = 0 ;
                    cout<<"\n      The Transaction is invalid. Please enter either
Cash or Cheque. \n" ;
                    getch() ;
            }

    } while (!check);

    do
    {
            cout <<"\n     Please enter the Transaction Amount :  \n";
            check = 1 ;
            cout <<"       Amount : Rs. ";
            gets(tm) ;
            t_amt = atof(tm) ;
            t_amount = t_amt ;

            if (t_amount < 1 || (tranDetails == 'W' && t_amount > iBalance) )
```

```
        {
                check = 0;
                cout<<"\n       Either Amount is not a numeric value or\n it is
blank or\n you are trying to withdraw amount more than in the Account.....  \n" ;
                getch() ;
        }
} while (!check) ;
char choice ;

do
{
        cout <<"\n      Save the changes made in the transaction details? (y/n): ";
        choice = getche() ;
        choice = toupper(choice);
} while (choice != 'N' && choice != 'Y') ;

if (choice == 'N' || choice == 'n') {
        cout<<"\n       The Transaction is not saved. \n";
        getch();
        return ;
}

if (tranDetails == 'D') {
        cout<<"\n       The Amount is Deposited in the Bank.\n";
        getch();
        iBalance = iBalance + t_amount;
} else {
        cout<<"\n       The Amount is Withdrawn from the Bank.\n";
        getch();
        iBalance = iBalance - t_amount;
        }
 newRec.updateBalance(retrieve_AccNo,iBalance);
  /* Adding record details for the Transaction done (deposit or withdrawal) and
saving it in the file*/

addDetails(retrieve_AccNo,day1,month1,year1,tranDetails,typeTransaction,interest_accrued,
t_amount,iBalance);
}

/* This method deletes the Account from both the dat files(Choice 6)*/
void accountTransactions :: closeAccount(void)
{
  clrscr() ;
  char t_acc[10] ;
  int tran_acc, retrieve_AccNo ;
  cout <<"       Press 0 to go back to previous menu.\n" ;
  cout <<"       Please enter the Account you want to close : " ;
  gets(t_acc);
```

```
    tran_acc = atoi(t_acc) ; /* changing account no. to integer type. */
    retrieve_AccNo = tran_acc ;
    clrscr() ;
    dispRecords newRec ;
    if (!newRec.accountExists(retrieve_AccNo))
    {
            cout <<"\t\n You have entered an invalid Account or it does not exist.\n";
            cout <<" Please try again.\n";
            getch();
            return ;
    }
    cout <<"\n      Press 0 to go back to previous menu\n" ;
    cout<<"\n       Closing this Account.\n";
    cout<<"**********************************\n\n";
    int day1, month1, year1 ;
    struct date dateval;
    getdate(&dateval);
    day1 = dateval.da_day ;
    month1 = dateval.da_mon ;
    year1 = dateval.da_year ;

    cout <<"Date: "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
    char choice;
    newRec.display(retrieve_AccNo); /*Displaying the Account Details on the basis of
the retrieved Account Number*/

    do
    {
            cout <<"\n      Are you sure you want to close this Account? (y/n): ";
            choice = getche();
            choice = toupper(choice) ;
    } while (choice != 'N' && choice != 'Y');

    if (choice == 'N' || choice == 'n') {
            cout<<"\n       The Account is not closed.\n";
            getch();
            return;
    }
    newRec.deleteAccount(retrieve_AccNo);
    deleteAccount(retrieve_AccNo);
    cout <<"\t\n\n Record Deleted Successfully.\n";
    cout <<"       Please continue with the application....\n";
    getch();
}

/* The Login method checks for the username and the password for accessing the
Banking Application*/
```

```
int login (void)
{
char username[9],ch;
char username1[]="banking";
int i=0;
char a,b[9],pass[]="tatahill";
cout<<"\n\n";
cout<<"\n\t    Login to the Banking Application.\n";
cout<<"\t      **********************************\n";
cout<<"\n\n\tPlease enter Username    :       ";
cin >> username;
cout<<"\n\n\tPlease enter Password to authenticate yourself :       ";
fflush(stdin);
        do
                        {
                        ch=getch();
                        if(isalnum(ch))
                                {
                                b[i]=ch;
                                cout<<"*";
                                i++;
                                }
                        else
                                if(ch=='\r')
                                        b[i]='\0';
                                else if(ch=='\b')
                                        {
                                        i-;
                                        cout<<"\b\b";
                                        }
                        }
                while(ch!='\r');

b[i]='\0';
fflush(stdin);

        if((strcmp(b,pass)==0)&&(strcmp(username1,username)==0))
        {
                        cout<<"\n\n\t  You have entered successfully\n\n";
                        return(1);

        }
        else
        {
                        cout<<"\t\n\n          Incorrect Username or Password.";
                        cout<<"\n";
                        return(0);

        }
```

```
    }

/* This is the Main function which displays the Menu */
void main(void)
{
  clrscr();
  int val,ch;
  a: val=login();
  if (val==0)
  {
        cout<<"\n\t    Want to try again?\n";
        cout<<"\t      1.TRY AGAIN ";
        cout<<"\n\t    2.EXIT";
        cout<<"\n\n\t  Enter your choice and press enter:";
        cin>>ch;
        if (ch==1) {
              clrscr();
              goto a;

        } else {
              exit(0);
        }
  }
  Menus obj1 ;
  obj1.showmenu();
}
```