PROGRAMMING EXAMPLE: Checking Account Balance

A local bank in your town is looking for someone to write a program that calculates a customer's checking account balance at the end of each month. The data is stored in a file in the following form:

```
467343 23750.40
W 250.00
D 1200.00
W 75.00
I 120.74
.
.
.
```

The first line of data shows the account number followed by the account balance at the beginning of the month. Thereafter, each line has two entries: the transaction code and the transaction amount. The transaction code `W` or `w` means withdrawal; `D` or `d` means deposit; and `I` or `i` means interest paid by the bank. The program updates the balance after each transaction. During the month, if at any time the balance goes below $1000.00, a $25.00 service fee is charged. The program prints the following information: account number, balance at the beginning of the month, balance at the end of the month, interest paid by the bank, total amount of deposits, number of deposits, total amount of withdrawals, number of withdrawals, and service charge, if any.

Input A file consisting of data in the above format.

Output The output is of the following form:

```
Account Number: 467343
Beginning Balance: $23750.40
Ending Balance: $24611.49

Interest Paid: $366.24

Amount Deposited: $2230.50
Number of Deposits:  3

Amount Withdrawn: $1735.65
Number of Withdrawals: 6
```

The output is to be stored in a file.

PROBLEM ANALYSIS AND ALGORITHM DESIGN

The first entry in the input file is the account number and the beginning balance, so the program first reads the account number and beginning balance. Thereafter, each entry in the file is of the following form:

```
transactionCode transactionAmount
```

To determine the account balance at the end of the month, you need to process each entry that contains the transaction code and transaction amount. Begin with the starting balance and then update the account balance after processing each entry. If the transaction code is `D`, `d`, `I`, or `i`, the transaction amount is added to the account balance. If the transaction code is `W` or `w`, the transaction amount is subtracted from the balance. Because the program also outputs the number of withdrawals and deposits, you need to keep separate counts of withdrawals and deposits. This discussion translates into the following algorithm:

1. Declare the variables.
2. Initialize the variables.
3. Get the account number and beginning balance.
4. Get the transaction code and transaction amount.
5. Analyze the transaction code and update the appropriate variables.
6. Repeat Steps 4 and 5 until there is no more data.
7. Print the result.

Variables

The program outputs the account number, beginning balance, balance at the end of the month, interest paid, amount deposited, number of deposits, amount withdrawn, number of withdrawals, and service charge, if any. You need variables to store all this information. So far, you need the following variables:

```
acctNumber          //variable to store the account number
beginningBalance    //variable to store the beginning balance
accountBalance      //variable to store the account balance at the
                    //end of the month
amountDeposited     //variable to store the total amount deposited
numberOfDeposits    //variable to store the number of deposits
amountWithdrawn     //variable to store the total amount withdrawn
numberOfWithdrawals //variable to store the number of withdrawals
interestPaid        //variable to store the interest paid
```

Because the program reads the data from a file and the output is stored in a file, the program needs both input and output stream variables. After the first line, the data in each line is the transaction code and the transaction amount; the program needs a variable to store this information.

Whenever the account balance goes below the minimum balance, a service charge for that month is applied. After each withdrawal, you need to check the account balance. If the balance goes below the minimum after a withdrawal, a service charge is applied. You can potentially have several withdrawals in a month; once the account balance goes below the minimum, a subsequent deposit might bring the balance above the minimum, and another withdrawal might again reduce it below the minimum. However, the service charge is applied only once.

To implement this idea, the program uses a `bool` variable, `isServiceCharged`, which is initialized to `false` and set to `true` whenever the account balance goes below the minimum. Before applying a service charge, the program checks the value of the variable `isServiceCharged`. If the account balance is less than the minimum and `isServiceCharged` is `false`, a service charge is applied. The program needs the following variables:

```
int acctNumber;
double beginningBalance;
double accountBalance;

double amountDeposited;
int numberOfDeposits;
double amountWithdrawn;
int numberOfWithdrawals;

double interestPaid;

char transactionCode;
double transactionAmount;

bool isServiceCharged;

ifstream infile;  //input file stream variable
ofstream outfile; //output file stream variable
```

Named Constants

Because the minimum account balance and the service charge amount are fixed, the program uses two named constants to store them:

```
const double MINIMUM_BALANCE = 1000.00;
const double SERVICE_CHARGE = 25.00;
```

PROBLEM ANALYSIS AND ALGORITHM DESIGN (CONTINUED)

Because this program is more complex than previous ones, before writing the main algorithm, the seven steps on page 2 are described more fully here.

1. **Declare the variables.** Declare variables as discussed previously.
2. **Initialize the variables.** After each deposit, the total amount deposited is updated and the number of deposits is incremented by 1. Before the first deposit, the total amount deposited is 0, and the number of deposits is 0. Therefore, the variables `amountDeposited` and `numberOfDeposits` must be initialized to 0. Similarly, the variables `amountWithdrawn`, `numberOfWithdrawals`, and `interestPaid` must be initialized to `0`. Also, as discussed previously, the variable `isServiceCharged` is initialized to `false`. Of course, you can initialize these variables when you declare them.

   Before the first deposit, withdrawal, or interest paid, the account balance is the same as the beginning balance. Therefore, after reading

the beginning balance in the variable `beginningBalance` from the file, you need to initialize the variable `accountBalance` to the value of the variable `beginningBalance`.

Because the data will be read from a file, you need to open the input file. If the input file does not exist, output an appropriate message and terminate the program. Because the output will be stored in a file, you need to open the output file. Suppose the input data is in the file `Ch5_money.txt`. Also suppose that the output will be stored in the file `Ch5_money.out`. The following code opens the files:

```
infile.open("Ch5_money.txt"); //open the input file

if (!infile)
{
   cout << "Cannot open the input file." << endl;
   cout << "Program terminates!!!" << endl;
   return 1;
}

outfile.open("Ch5_money.out"); //open the output file
```

3. **Get the account number and starting balance.** This is accomplished by the following input statement:

   ```
   infile >> acctNumber >> beginningBalance;
   ```

4. **Get the transaction code and transaction amount.** This is accomplished by the following input statement:

   ```
   infile >> transactionCode >> transactionAmount;
   ```

5. **Analyze the transaction code and update the appropriate variables:** If the `transactionCode` is `'D'` or `'d'`, update `accountBalance` by adding `transactionAmount`, update `amountDeposited` by adding `transactionAmount`, and increment `numberOfDeposits`. If the `transactionCode` is `'I'` or `'i'`, update `accountBalance` by adding `transactionAmount`, and update `interestPaid` by adding `transactionAmount`. If the `transactionCode` is `'W'` or `'w'`, update `accountBalance` by subtracting `transactionAmount`, update `amountWithdrawn` by adding `transactionAmount`, increment `numberOfWithdrawals`, and—if the account balance is below the minimum and service charges have not been applied—subtract the service charge from the account balance and mark the service charges as having been applied. The following `switch` statement accomplishes this task.

```
switch (transactionCode)
{
case 'D':
case 'd':
    accountBalance = accountBalance  + transactionAmount;
    amountDeposited = amountDeposited + transactionAmount;
    numberOfDeposits++;
    break;
case 'I':
case 'i':
    accountBalance = accountBalance + transactionAmount;
    interestPaid = interestPaid + transactionAmount;
    break;
case 'W':
case 'w':
    accountBalance = accountBalance - transactionAmount;
    amountWithdrawn = amountWithdrawn + transactionAmount;
    numberOfWithdrawals++;

    if ((accountBalance < MINIMUM_BALANCE)
                && (!isServiceCharged))
    {
        accountBalance = accountBalance - SERVICE_CHARGE;
        isServiceCharged = true;
    }
    break;

default:
    cout << "Invalid transaction code." << endl;
} //end switch
```

6. **Repeat Steps 4 and 5 until there is no more data.** Because the number of entries in the input file is not known, the program needs an EOF-controlled `while` loop.
7. **Print the result.** This is accomplished by using output statements.

Main Algorithm

Based on the above discussion, the main algorithm is as follows:

1. Declare and initialize the variables.
2. Open the input file.
3. If the input file does not exist, exit the program.
4. Open the output file.
5. To output floating-point numbers in a fixed decimal format with the decimal point and trailing zero, set the manipulators `fixed` and `showpoint`. To output floating-point numbers to two decimal places, set the precision to two decimal places.
6. Read `accountNumber` and `beginningBalance`.
7. Set `accountBalance` to `beginningBalance`.

8. Read `transactionCode` and `transactionAmount`.
9. `while` (not end of input file)
   a. If `transactionCode` is `'D'`
      i. Add `transactionAmount` to `accountBalance`
      ii. Add `transactionAmount` to `amountDeposited`
      iii. Increment `numberOfDeposits`
   b. If `transactionCode` is `'I'`
      i. Add `transactionAmount` to `accountBalance`
      ii. Add `transactionAmount` to `interestPaid`
   c. If `transactionCode` is `'W'`
      i. Subtract `transactionAmount` from `accountBalance`
      ii. Add `transactionAmount` to `amountWithdrawn`
      iii. Increment `numberOfWithDrawals`
      iv. If (`accountBalance < MINIMUM_BALANCE && !isServiceCharged`)
         1. Subtract `SERVICE_CHARGE` from `accountBalance`
         2. Set `isServiceCharged` to `true`
   d. If `transactionCode` is a letter other than `'D'`, `'d'`, `'I'`, `'i'`, `'W'`, or `'w'`, output an error message.
10. Output the results.

Because the data will be read from an input file, you must include the header file `fstream`. Because you will use the manipulator `setprecision`, you must also include the header file `iomanip`. If the input file does not exist, an appropriate message on the screen will be displayed, so the header file `iostream` is also included.

COMPLETE PROGRAM LISTING

```
//*********************************************************
// Author: D.S. Malik
//
// Program -- Checking Account Balance.
// This program calculates a customer's checking account
// balance at the end of the month.
//*********************************************************
```

```
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

const double MINIMUM_BALANCE = 1000.00;
const double SERVICE_CHARGE = 25.00;

int main()
{
        //Declare and initialize variables          //Step 1
    int acctNumber;
    double beginningBalance;
    double accountBalance;

    double amountDeposited = 0.0;
    int numberOfDeposits = 0;

    double amountWithdrawn = 0.0;
    int numberOfWithdrawals = 0;
    double interestPaid = 0.0;
    char transactionCode;
    double transactionAmount;

    bool isServiceCharged = false;

    ifstream infile;
    ofstream outfile;

    infile.open("Ch5_money.txt");                   //Step 2

    if (!infile)                                    //Step 3
    {
        cout << "Cannot open the input file." << endl;
        cout << "Program terminates!!!" << endl;
        return 1;
    }

    outfile.open("Ch5_money.out");                  //Step 4

    outfile << fixed << showpoint;                  //Step 5
    outfile << setprecision(2);                     //Step 5

    cout << "Processing data" << endl;

    infile >> acctNumber >> beginningBalance;       //Step 6

    accountBalance = beginningBalance;              //Step 7

    infile >> transactionCode >> transactionAmount; //Step 8
```

```
while (infile)                                          //Step 9
{
    switch (transactionCode)
    {
    case 'D':                                           //Step 9.a
    case 'd':
        accountBalance = accountBalance
                         + transactionAmount;
        amountDeposited = amountDeposited
                         + transactionAmount;
        numberOfDeposits++;
        break;
    case 'I':                                           //Step 9.b
    case 'i':
        accountBalance = accountBalance
                         + transactionAmount;
        interestPaid = interestPaid
                       + transactionAmount;
        break;
    case 'W':                                           //Step 9.c
    case 'w':
        accountBalance = accountBalance
                         - transactionAmount;
        amountWithdrawn = amountWithdrawn
                         + transactionAmount;
        numberOfWithdrawals++;

        if ((accountBalance < MINIMUM_BALANCE)
                       && (!isServiceCharged))
        {
            accountBalance = accountBalance
                             - SERVICE_CHARGE;
            isServiceCharged = true;
        }
        break;

    default:
        cout << "Invalid transaction code" << endl;
    } //end switch

    infile >>  transactionCode >> transactionAmount;
}//end while

    //Output Results                                    //Step 10
outfile << "Account Number: " << acctNumber << endl;
outfile << "Beginning Balance: $" << beginningBalance
        << endl;
outfile << "Ending Balance: $" << accountBalance
        << endl << endl;
outfile << "Interest Paid: $" << interestPaid << endl
        << endl;
```

```
    outfile << "Amount Deposited: $" << amountDeposited
            << endl;
    outfile << "Number of Deposits: " << numberOfDeposits
            << endl << endl;
    outfile << "Amount Withdrawn: $" << amountWithdrawn
            << endl;
    outfile << "Number of Withdrawals: "
            << numberOfWithdrawals << endl << endl;

    if (isServiceCharged)
        outfile << "Service Charge: $" << SERVICE_CHARGE
                << endl;
    return 0;
}
```


Sample Run: (Contents of the output file `Ch5_money.out`)

```
Account Number: 467343
Beginning Balance: $23750.40
Ending Balance: $24490.75

Interest Paid: $245.50

Amount Deposited: $2230.50
Number of Deposits: 3

Amount Withdrawn: $1735.65
Number of Withdrawals: 6
```

Input File: (`Ch5_money.txt`)

```
467343 23750.40
W 250.00
D 1200.00
W 75.00
W 375.00
D 580.00
I 245.50
W 400.00
W 600.00
D 450.50
W 35.65
```