

*This sheet must be the cover page for every programming assignment.*

Name Joe Student

Assignment Title Bank Account Program

---

(do not write below this line; for grader use only)

**Validity \_\_\_\_\_ (up to 70%)**

Matches assignment specification precisely  
Uses appropriate data structures  
Provides sufficient sample runs  
Produces correct results  
Algorithm is correct and as specified  
Output matches code  
No bugs

**User-friendliness \_\_\_\_\_ (up to 5%)**

Data structure display is visually clear  
Correct spelling and grammar  
Neat, appropriately commented output  
Clear error messages  
Grammatical output  
Robust  
Demonstrates adequate testing

**Documentation \_\_\_\_\_ (up to 5%)**

Introductory comments  
Adequate individual routine comments (including pre/post conditions)  
Clear comments  
Neat, professional presentation  
File names at top of each file  
Header files before implementation files  
User instructions provided

**No-run conditions \_\_\_\_\_ (up to 5%)**

**Programming style \_\_\_\_\_ (up to 5%)**

Good identifier names  
Good indentation  
Easy to follow  
Main program precedes details  
Efficient approach  
Public before private in class  
Good structure

**Modularity \_\_\_\_\_ (up to 10%)**

Modularized to facilitate reuse  
Sufficiently modular  
Separate header and implementation files  
Main program brief and clear  
Header/implementation file pair for each class  
Appropriate use of classes and templates  
Appropriate use of functions

**Extra credit \_\_\_\_\_**

**Total \_\_\_\_\_**

## Program Analysis Statement

*Problems I encountered during the program:* I didn't have any difficulty with this program. If I had, I would describe that difficulty here. For example, I might've had trouble opening an input file and/or reading its contents into an array of integers. If so, I would describe that problem here.

*No Run Conditions:* Program runs under all conditions

- If the program didn't run in certain circumstances, I'd list them here. See the sample function-based program for example no-run conditions.

*Extra Credit Features:* There are no extra credit features in my program, but if there had been I would describe them here.

*STL Code/Code From Text:* I didn't use an STL or any code from the textbook. If I had, I would document the use of that code here.

## The Account ADT

### Data:

the bank account balance, which can never be negative

### Operations:

+createAccount

//Purpose: default constructor

//Precondition: none

//Postcondition: balance will initialized to 0.00 and an account object will exist

+createAccount (in amountToSet:*float*)

//Purpose: constructs an Account object with the value amountToSet

//Precondition: amountToSet must be a positive float

//Postcondition: Account will now be set to the amount specified by amountToSet, if a

//positive value was passed, otherwise set to 0.00 and an account object will exist

+destroyAccount

//Purpose: destructor

//Precondition: none

//Postcondition: none

+ GetAccountBalance():*float*{query}

//Purpose: returns the current Account balance

//Precondition: none

//Postcondition: a float is returned with the value of the balance

+ SetAccountBalance(in amountToSet:*float*): *bool*

//Purpose: sets the current Account balance

//Precondition: amount to be set is passed as a positive float

//Postcondition: if balance was set return true otherwise return false

+ Withdraw (in amountToWithdraw:*float*):*bool*

//Purpose: withdraws money from the account

//Precondition: the amount to withdraw is passed as a positive float,

//Postcondition: true is returned if the amount was withdrawn; false if the amount to

//withdraw was more than account balance

+ Deposit (in amountToDeposit:*float*):*bool*  
//Purpose: deposite money to the account  
//Precondition: the amount to deposit is passed as a float, must be a positive number  
//Postcondition: true is returned if the amount was deposited and false otherwise

+ DisplayBalance(){query}  
//Purpose: displays balance on screen, in currency format, i.e.: \$10.00  
//Precondition: none  
//Postcondition: none

- RoundBalanceToNearestCent()  
//Purpose: rounds the balance to the nearest cent  
//Precondition: none  
//Postcondition: balance is rounded to nearest cent

**Program Output**

Coming soon.

## Program Input

Coming soon.

## **Program Source Code**

```

//main.cpp
//
/////////////////////////////////////////////////////////////////
#include "Account.h"
#include <iostream>
using namespace std;

void main(){

    float amount=12.3456;

    //testing default constructor
    cout <<"\n\naccountDefault";
    Account accountDefault;
    accountDefault.DisplayBalance();

    //testing overloaded constructor with positive and negative input
    cout << "\n\naccountOne: instantiating account with variable
amount=12.3456";
    Account accountOne(amount);
    accountOne.DisplayBalance();

    cout << "\n\naccountTwo: instantiating account with -10.00";
    Account accountTwo(-10.00);
    accountTwo.DisplayBalance();

    //testing Deposit
    cout<<"\n\ndepositing $100.15 to accountOne";
    accountOne.Deposit(100.151);
    accountOne.DisplayBalance();

    //testing Withdraw
    cout<<"\n\nwithdrawing amount=12.3456 from accountOne";
    accountOne.Withdraw(amount);
    accountOne.DisplayBalance();

    //new test
    cout <<"\n\nnew ";
    Account accountTest;
    accountTest.Deposit(12.115);
    accountTest.DisplayBalance();

    cout <<endl<<endl;

    getch();
}

```

## Account.h

```
#if !defined(AFX_ACCOUNT_H_C4B3EF22_288B_11D6_BC8A_0001032102EE_INCLUDED_)
#define AFX_ACCOUNT_H_C4B3EF22_288B_11D6_BC8A_0001032102EE_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Account
{
public:

    Account();
    /*-----
    Purpose:          default constructor
    Precondition:     none
    Postcondition:    balance will initialized to 0.00 and an account object
                    will exist

    *///-----

    Account(const float amountToSet);
    /*-----
    Purpose:          constructs an Account object with the value amountToSet
    Precondition:     amountToSet must be a positive float
    Postcondition:    Account will now be set to the amount specified by
                    amountToSet, if a positive value was passed,
                    otherwise
                    set to 0.00 and an account object will exist

    *///-----

    ~Account();
    /*-----
    Purpose:          destructor
    Precondition:     none
    Postcondition:    none

    *///-----

    float GetAccountBalance() const;
    /*-----
    Purpose:          returns the current Account balance
    Precondition:     none
    Postcondition:    a float is returned with the value of the balance

    *///-----

    bool SetAccountBalance(const float amountToSet);
    /*-----
    Purpose:          sets the current Account balance
    Precondition:     amount to be set is passed as a positive float
    Postcondition:    if balance was set return true otherwise return false
    */
```

## Account.h

```
*/---  
  
bool Withdraw(const float amountToWithdraw);  
/*---  
Purpose:          withdraws money from the account  
Precondition:     the amount to withdraw is passed as a positive float,  
Postcondition:    true is returned if the amount was withdrawn false if  
                  the amount to withdraw was more than account  
                  balance  
*/---
```

```
bool Deposit(const float amountToDeposit);  
/*---  
Purpose:          deposits money to the account  
Precondition:     the amount to deposit is passed as a float, must be a  
                  positive number  
Postcondition:    true is returned if the amount was deposited false  
                  otherwise  
*/---
```

```
void DisplayBalance() const;  
/*---  
Purpose:          displays balance on screen, in currency format, ie:  
$10.00  
Precondition:     none  
Postcondition:    none  
*/---
```

private:

```
float balance;
```

```
void RoundBalanceToNearestCent();  
/*---  
Purpose:          rounds the balance to the nearest cent  
Precondition:     none  
Postcondition:    balance is rounded to nearest cent  
Notes:           this method was adapted from Andrey Butov's round  
                  function  
*/---
```

```
};
```

**Account.h**

```
#endif //  
!defined(AFX_ACCOUNT_H__C4B3EF22_288B_11D6_BC8A_0001032102EE__INCLUDED_)
```

## Account.cpp

```
#include "Account.h"
#include <iostream>
#include <iomanip> //for io manipulation in cout statements
#include <math.h> //for pow function
using namespace std;

//-----

Account::Account()
{
    balance=0.00;
}

//-----

Account::Account(const float amountToSet)
{
    //if we are unable to set the balance with amountToSet
    //set balance to 0.00 (i.e. if amountToSet is negative)
    if(SetAccountBalance(amountToSet)==false){

        balance=0.00;
    }
}

//-----

Account::~~Account()
{
}

//-----

float
Account::GetAccountBalance() const
{
    return balance;
}

//-----

bool
Account::SetAccountBalance(const float amountToSet)
{
    //if the amountToSet is negavie return false
    if(amountToSet<0){

        return false;
    }else{
        //set the balance and round the value
        balance=amountToSet;
        RoundBalanceToNearestCent();
        return true;
    }
}
```

## Account.cpp

```

}

//-----

bool
Account::Withdraw(const float amountToWithdraw)
{
    //if amountToWithdraw is greater than the balance, don't withdraw
    if(balance<amountToWithdraw){
        return false;

    }else{

        //if we are unable to set the balance with amountToSet
        //set balance to 0.00 (i.e. if amountToSet is negative)
        SetAccountBalance(balance-amountToWithdraw);
        return true;
    }
}

//-----

bool
Account::Deposit(const float amountToDeposit)
{
    //if amountToDeposit is negative
    if(amountToDeposit<0){
        return false;

    }else{

        //if we are unable to set the balance with amountToSet
        //return false
        SetAccountBalance(balance+amountToDeposit);
        return true;
    }
}

//-----

void
Account::DisplayBalance() const
{
    cout << setiosflags(ios::fixed );           // Format x.y
    cout << setiosflags(ios::showpoint );       // 0.10
    cout << setprecision(2);                    // 2 dec places
    cout<<"\nAccount Balance: $ " << balance;
}

//-----

void
Account::RoundBalanceToNearestCent()
{

```

## Account.cpp

```
//set the balance to 2 decimal places  
//for example if balance = 10.2451  
//balance*100 = 1024.51, we add .5 to round to get 1025.01  
//now we need to cut off the .01, so we use floor()  
//now put decimal point before the 2, so we divide by 100
```

```
balance=floor(balance * 100 + .5)/100;
```

```
}
```