CS 141 : Program Design 2
Spring 2021

Project 01 : UIC Bank v1.0

Assignment: C++ program to manage bank accounts
Evaluation: Gradescope followed by manual review
Policy: Individual work only
Complete By:
Part 01 (40pts): Saturday Jan 16th @ 11:59pm CDT, late day Sun 1/17 @ 11:59pm
Part 02 (60pts): Saturday Jan 23rd @ 11:59pm CDT, late day Sun 1/24 @ 11:59pm

Pre-requisites: Panopto videos 01 – 05, HW 01 – 02, Lab 01

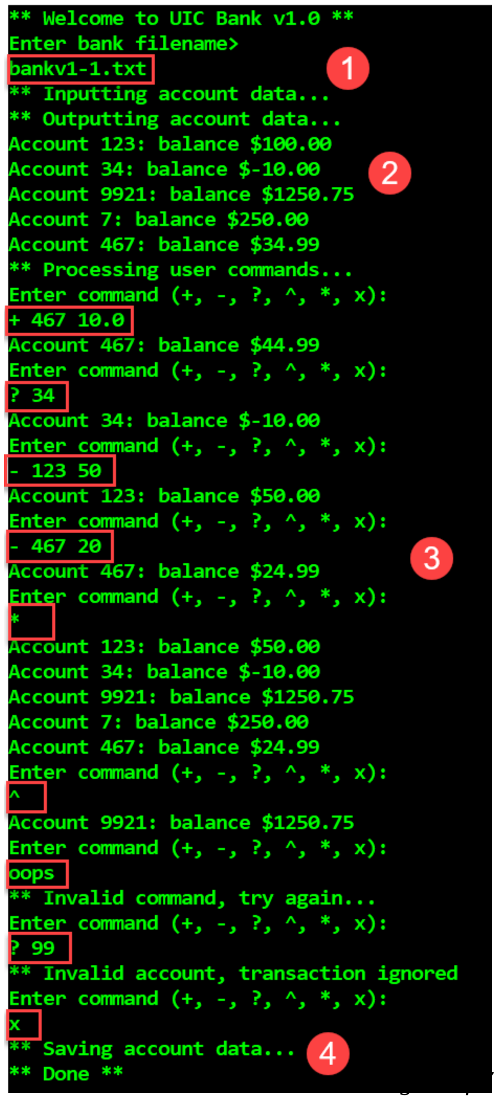
Overview

The goal in project 01 is to build a simple C++ program to manage the bank accounts associated with **UIC Bank v1.0**. The bank has exactly 5 customers, and the program allows the user to perform the following banking operations:

1. Deposit (+)
2. Withdrawal (-)
3. Check balance (?)
4. Find the account with the largest balance (^)
5. List all accounts and balances (*)

Here's a screenshot of how the program behaves --------------------------> The program starts by inputting the name of the banking file, and then inputs the data about the 5 bank accounts from this file (step 1). Then the data is output to the console (step 2). The next step (3) is the interactive step, where the user enters any number of banking commands until "x" is entered --- the exit command. The last step (4) is to write the account data back out to the same banking file so the new balances are saved.

The program will be written and submitted in two parts. In part 01 you'll build the basic skeleton of the program. In part 02, you'll add the interactive loop. To receive full credit you must finish each part by the specified deadline; you cannot earn full credit by skipping part 01 and submitting the complete project at the final deadline.

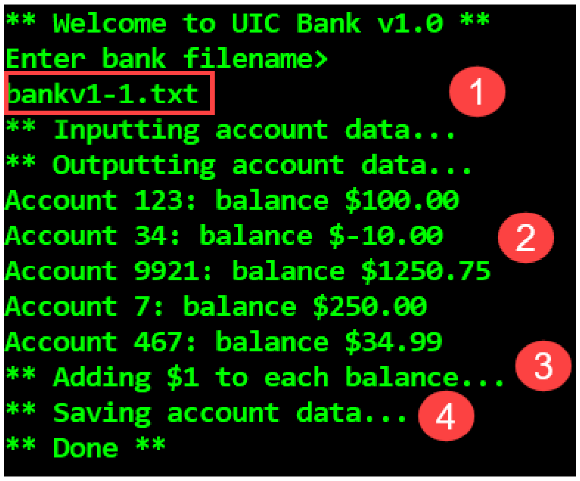


```
** Welcome to UIC Bank v1.0 **
Enter bank filename>
bankv1-1.txt
** Inputting account data...
** Outputting account data...
Account 123: balance $100.00
Account 34: balance $-10.00
Account 9921: balance $1250.75
Account 7: balance $250.00
Account 467: balance $34.99
** Processing user commands...
Enter command (+, -, ?, ^, *, x):
+ 467 10.0
Account 467: balance $44.99
Enter command (+, -, ?, ^, *, x):
? 34
Account 34: balance $-10.00
Enter command (+, -, ?, ^, *, x):
- 123 50
Account 123: balance $50.00
Enter command (+, -, ?, ^, *, x):
- 467 20
Account 467: balance $24.99
Enter command (+, -, ?, ^, *, x):
*
Account 123: balance $50.00
Account 34: balance $-10.00
Account 9921: balance $1250.75
Account 7: balance $250.00
Account 467: balance $24.99
Enter command (+, -, ?, ^, *, x):
^
Account 9921: balance $1250.75
Enter command (+, -, ?, ^, *, x):
oops
** Invalid command, try again...
Enter command (+, -, ?, ^, *, x):
? 99
** Invalid account, transaction ignored
Enter command (+, -, ?, ^, *, x):
x
** Saving account data...
** Done **
```

Part 01: Bank data input, modify, output

In part 01 you're going to write the basic skeleton of the application: input, update, output. **UIC Bank 1.0** has exactly 5 customers, each with an account number (positive integer) and a balance (a real number that can be negative, zero, or positive). Here are the contents of the "bankv1-1.txt" banking file used in the screenshot shown to the right:

```
123 100
34 -10
9921 1250.75
7 250
467 34.99
```



```
** Welcome to UIC Bank v1.0 **
Enter bank filename>
bankv1-1.txt
** Inputting account data...
** Outputting account data...
Account 123: balance $100.00
Account 34: balance $-10.00
Account 9921: balance $1250.75
Account 7: balance $250.00
Account 467: balance $34.99
** Adding $1 to each balance...
** Saving account data...
** Done **
```

The above is just one example; in general a banking file contains 5 lines of data, where each line contains a positive integer followed by a real number; there is one space between the values. The data is in no particular order. You'll want to test your program against different banking files; create multiple text files in Codio containing different values. Both your console and file output must match exactly what we are showing here to pass the tests on Gradescope.

In part 01, the program consists of the following 4 steps (see screenshot above). In step 1, the program inputs the banking filename from the keyboard, opens the file, and then inputs the 5 lines of data from the file. In step 2 it outputs this data to the console, in the order it appears in the file. Notice the balances are output with two digits to the right of the decimal point --- e.g. "$100.00". To force this behavior in C++, you must do the following in main():

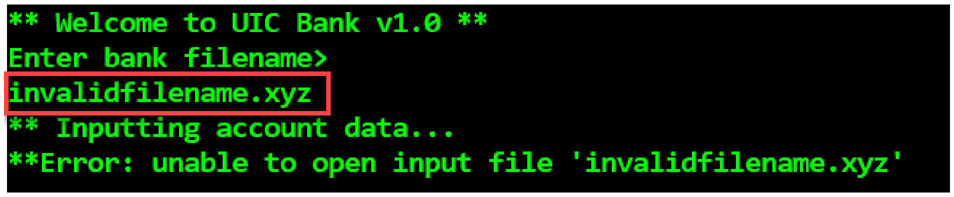
```
int main()
{
   cout << std::fixed;
   cout << std::setprecision(2);
```

You'll also need to add #include <iomanip> to the top of "main.cpp". In step 3, $1 is added to the balance of each account. In step 4, the accounts and updated balances are output to the same banking file, overwriting the previous data. For example, when the program ends, the "bankingv1-1.txt" file shown earlier will now contain

```
123 101
34 -9
9921 1251.75
7 251
467 35.99
```

The data is output in the same order it originally appeared; notice however that each balance has been increased by 1. Note the balances in the file output do not always have 2 digits to the right of the decimal point; this is by design.

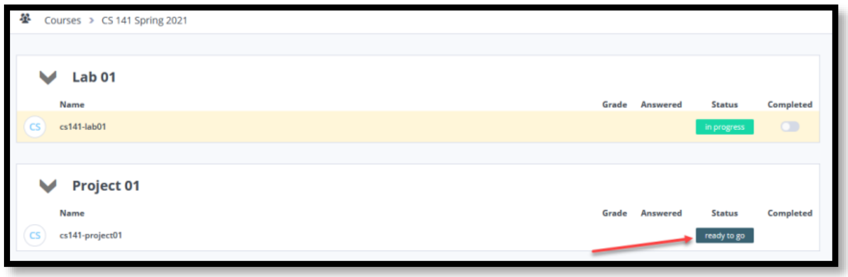
Your program is required to check if the filename entered can be successfully opened for input. If not, your program should output an error message and immediately return 0; likewise if it cannot be opened for output. Example:



```
** Welcome to UIC Bank v1.0 **
Enter bank filename>
invalidfilename.xyz
** Inputting account data...
**Error: unable to open input file 'invalidfilename.xyz'
```

Getting Started

No C++ starter code is being provided. For a programming environment, we are providing a Codio project "cs141-project01" with a sample banking file "bankv1-1.txt". If you have not yet created a Codio account, you can join via the following URL: https://codio.com/p/join-course?token=silicon-polka. Enter the token "silicon-polka" without the ", and register using your UIC email address. After you login, pick "CS 141 Spring 2021" if you have a choice of classes, and you'll see the following list of projects. Click on "ready to go" under Project 01 to get started.



Courses > CS 141 Spring 2021

Lab 01

	Name	Grade	Answered	Status	Completed
CS	cs141-lab01			in progress	

Project 01

	Name	Grade	Answered	Status	Completed
CS	cs141-project01			ready to go	

Panopto video #2 discusses how to compile and run your C++ program on Codio, as does Lab 01.

Project restrictions

How you solve the problem is just as important as developing a correct solution. In this assignment we are going to impose the following restrictions, which you must adhere to. Breaking a restriction typically leads to a score of 0.

1. No global variables. A global variable is a variable declared in "main.cpp" but outside of the main() function. If you end up writing additional functions (which is great), you must pass data using parameters, not global variables.
2. No data structures --- no arrays, no vectors, no linked-lists, no use of the built-in C++ containers. We intentionally want you to solve this program without data structures. This implies the different bank accounts will be represented using different variables.
3. We understand that #2 may lead to come cumbersome code, e.g. lots of if-statements. That's fine. An interesting challenge is crafting a solution that reduces this cumbersome-ness, without introducing a data structure. [*Hint: user-defined functions if you want to read ahead in zybooks.*]
4. Your program must represent a general solution, i.e. the ability to input and process any banking file. Coding to the test cases on Gradescope will lead to a score of 0.

Have a question? Use Piazza, not email

As discussed in the syllabus, questions should be posted to our course Piazza site — questions via email are typically ignored. Remember the guidelines for using Piazza:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question.
2. Post publicly — only post privately when asked by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".
4. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, for "how-to" see http://www.take-a-screenshot.org/ .

Don't post your entire answer / code — if you do, you just gave away the answer to the ENTIRE CLASS. When posting code, do so privately; there's an option to create a private post ("visible to staff only").

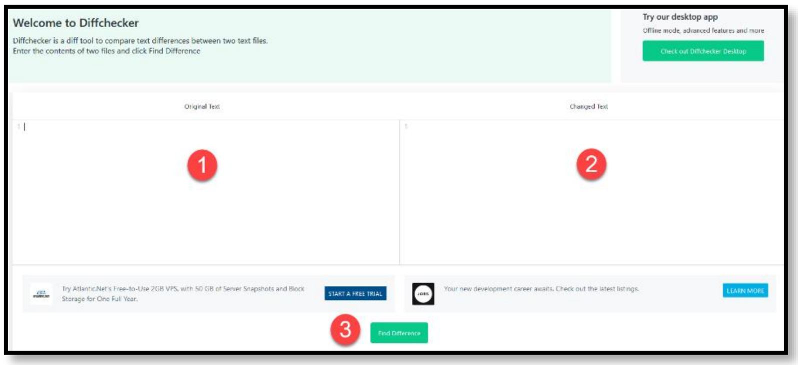
Part 01: submission, late policy, and grading

Submission: submit "main.cpp" to Gradescope under "Project 01: Part 01". You have an unlimited number of submissions, though it is expected you will test locally before submitting. To encourage local testing, the Gradescope submission site will not be made available until a couple days before the due date; do not ask when it will be released, that will just further delay its release. If necessary, you may submit up to 24 hours late for a penalty of 4 points.

Your score on part 01 is based solely on correctness, as determined by Gradescope; correctness will be worth 40 points out of the total 100 project points. Note that the TAs may manually review to ensure adherence to project restrictions, and your score could change as a result.

Suggestion: when you fail a test on Gradescope, we show your output, the correct output, and the difference between the two (as computed by Linux's **diff** utility). Please study the output carefully to see where your output differs. If there are lots of differences, or you can't see the difference, here's a good tip:

1. Browse to https://www.diffchecker.com/
2. Copy your output as given by Gradescope and paste into the left window
3. Copy the correct output as given by Gradescope and paste into the right window
4. Click the "Find Difference" button


Welcome to Diffchecker

Diffchecker is a diff tool to compare text differences between two text files.
Enter the contents of two files and click Find Difference

Try our desktop app

Offline mode, advanced features and more

Try AtlanticNet's Free-to-Use 2GB VPS, with 50 GB of Server Snapshots and Block Storage for One Full Year.

Your new development career awaits. Check out the latest listings.

You'll get a visual representation of the differences. Modify your program to produce the required output, and resubmit.

In terms of grading, note that we expect all submissions to compile, run, and pass at least some of the test cases; do not expect partial credit with regards to correctness unless your program compiles and runs. By default, we grade your **last** submission. Gradescope keeps a complete submission history, so you can **activate** an earlier submission if you want us to grade a different one; this must be done before the due date. We assume *every* submission on your Gradescope account is your own work; do not submit someone else's work for any reason, otherwise it is considered academic misconduct (how else did you end up with someone else's program?).

Part 02: Interactive commands

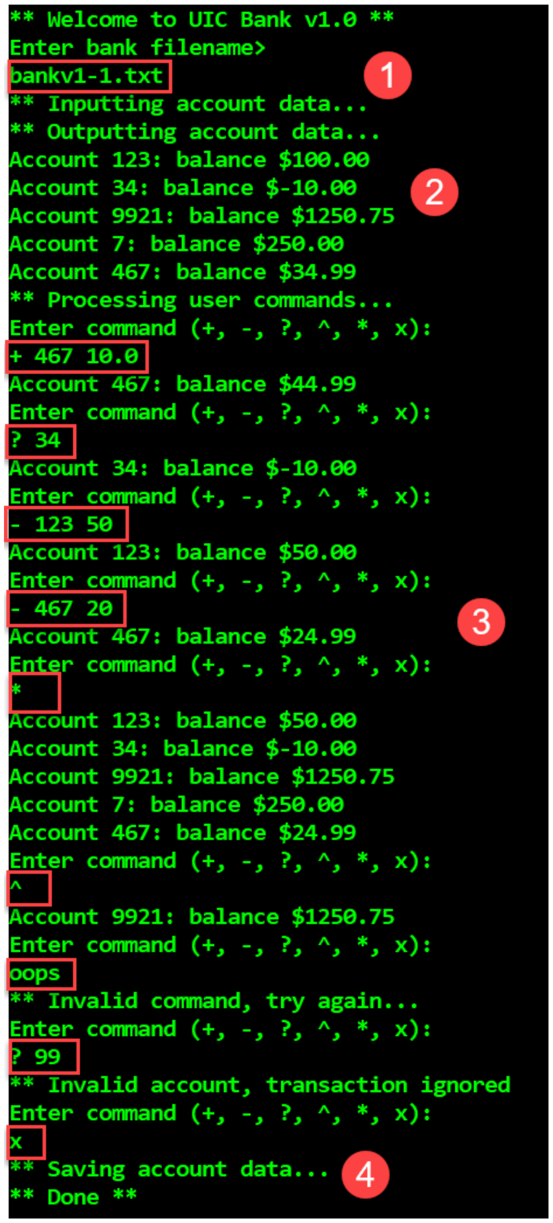
In part 02, steps 1, 2 and 4 remain the same: input the filename and bank data, output to the console, and then output the accounts and new balances to the same banking file. What changes is step 3: the program now accepts commands from the keyboard, and responds accordingly. This is repeated until "x" is entered, which denotes the exit command; this causes the program to exit the interactive loop.

Your program should accept any of following 5 commands from the user:

1. Deposit: **`+ account amount`**
2. Withdrawal: **`- account amount`**
3. Check balance: **`? account`**
4. Find the account with the largest balance: **^**
5. List all accounts and balances: *****
6. Exit: **x**

If the user enters any other command, output an error message as shown in the screenshot. For commands with arguments (+, -, ?), you may assume the user will input numeric values for the account and balance (you don't need to validate). However, the user may enter an account number that doesn't exist --- you need to check for that possibility. Example: ? 99 as shown in screenshot.

The bank is very forgiving, you are allowed to deposit or withdrawal any amount. Example: you can deposit a negative amount such as + 467 -100. When listing accounts to the console, always output in the same order as the original banking file. As before, when the program ends it should output the accounts and updated balances to the same banking file. For example, here are the contents of "bankv1-1.txt" before and after the execution shown above:


```
** Welcome to UIC Bank v1.0 **
Enter bank filename>
bankv1-1.txt
** Inputting account data...
** Outputting account data...
Account 123: balance $100.00
Account 34: balance $-10.00
Account 9921: balance $1250.75
Account 7: balance $250.00
Account 467: balance $34.99
** Processing user commands...
Enter command (+, -, ?, ^, *, x):
+ 467 10.0
Account 467: balance $44.99
Enter command (+, -, ?, ^, *, x):
? 34
Account 34: balance $-10.00
Enter command (+, -, ?, ^, *, x):
- 123 50
Account 123: balance $50.00
Enter command (+, -, ?, ^, *, x):
- 467 20
Account 467: balance $24.99
Enter command (+, -, ?, ^, *, x):
*
Account 123: balance $50.00
Account 34: balance $-10.00
Account 9921: balance $1250.75
Account 7: balance $250.00
Account 467: balance $24.99
Enter command (+, -, ?, ^, *, x):
^
Account 9921: balance $1250.75
Enter command (+, -, ?, ^, *, x):
oops
** Invalid command, try again...
Enter command (+, -, ?, ^, *, x):
? 99
** Invalid account, transaction ignored
Enter command (+, -, ?, ^, *, x):
x
** Saving account data...
** Done **
```

```
123 100                                   123 50
34 -10                                    34 -10
9921 1250.75                              9921 1250.75
7 250                                     7 250
467 34.99                                 467 24.99
```

Part 02: submission, late policy, and grading

Submission: submit "main.cpp" to Gradescope under "Project 01: Part 02". You have an unlimited number of submissions, though it is expected you will test locally before submitting. To encourage local testing, the Gradescope submission site will not be made available until a couple days before the due date; do not ask when it will be released, that will just further delay its release. If necessary, you may submit up to 24 hours late for a penalty of 6 points.

Your score on part 02 is based on two factors: (1) correctness as determined by Gradescope (50 points), and (2) manual review of "main.cpp" for commenting, readability (consistent indentation and whitespace), appropriate variable names, and adherence to project restrictions. In terms of commenting, you must have a header comment at the top of "main.cpp" with your name, UIC, the date, and an overview of the project. Example: a good header comment would explain that the program interacts with the user to perform banking transactions, and supports the following commands:

```
//
// Deposit:          + account amount
// Withdrawal:       - account amount
// Check balance:    ? account
// Find the account with the largest balance:  ^
// List all accounts and balances:  *
// Exit:  x
//
```

The rest of the file is expected to have meaningful comments, e.g. each major step of the program should have a comment above it. In this program, it's clear from the screenshots we have shown that the program has 4 distinct steps, so main() should have 4 header-like comments above each section. Example:

```
//
// (1) Input banking filename, confirm file can be
// opened, and then input the 5 bank accounts:
//
```

Line-by-line comments are only needed to explain code that isn't obvious to another programmer. Comments like the following are meaningless and a waste of time:

```
int count;  // declare a count variable
count = count + 1;  // add one to count
```

Here's an example of a useful line comment:

```
    int result = -1;  // assume search will fail and in that case return -1
```

In terms of grading, note that we expect all submissions to compile, run, and pass at least some of the test cases; do not expect partial credit with regards to correctness unless your program compiles and runs. By default, we grade your **last** submission. Gradescope keeps a complete submission history, so you can **activate** an earlier submission if you want us to grade a different one; this must be done before the due date. We assume *every* submission on your Gradescope account is your own work; do not submit someone else's work for any reason, otherwise it is considered academic misconduct (how else did you end up with someone else's program?).

Academic Honesty

In this assignment, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml

In particular, note that you are guilty of academic dishonesty if you **extend or receive any kind of unauthorized assistance**. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, screen sharing, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml .