Devon Caleb Mealey
Final Project Plan
CSPB 1300 – ATM Project Part 2

I have a full grasp on the final project requirements and required program structure.

The strictest requirement of the project is that more than one, if not all/the majority of the functions must use a pass-by-pointer method of argument passing – i.e. no altering or requesting the account balances and amounts to be deposited, withdrawn, or transferred directly.

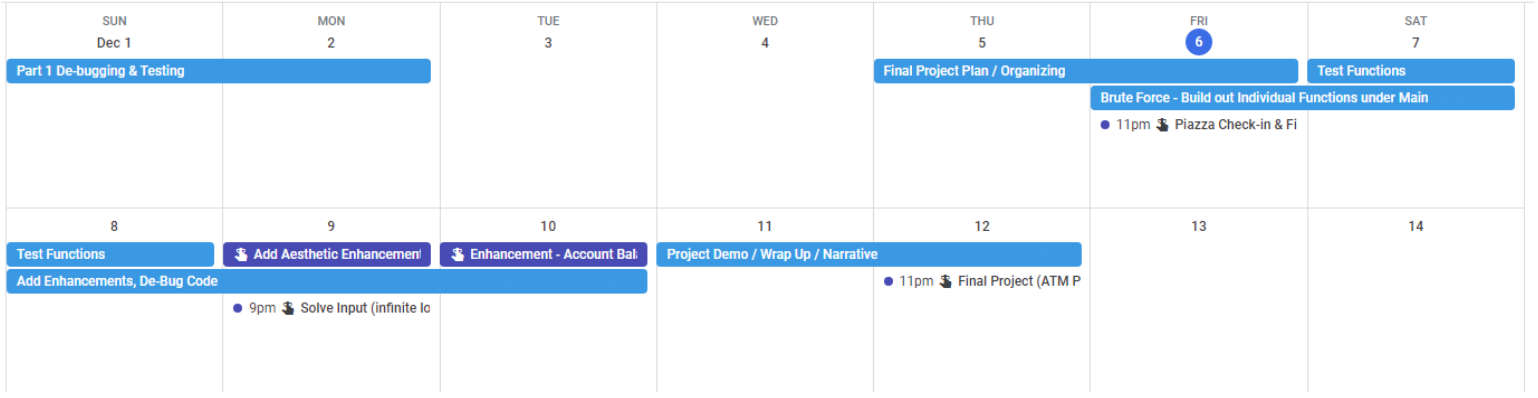
I'm aiming to finish the project early, and submit by midnight on Thursday December 12th, so I can retain the flexibility to not take the final exam.

I will proceed at first with a brute force method of organizing my code to run predominately all under one main function – like was prompted for part 1. Next, I will break up the main function and divvy up the various functions into their own respective functions: Deposit, Withdrawal, Transfer, and Balance Inquiry.

I had an error issue in my part 1 submission, I did not catch until I tested my submission further in the last day as I began to dive deeper into my planning for the final project outline, part 2. That error was an infinite invalid input loop, which occurred whenever the user inputted a string or character into the main menu input. It appears the program would get stuck in the infinite invalid input loop, and not allow the user to break the loop without closing the terminal completely – not exemplifying bug-free code at all! I am going to look at fixing this error several ways, and am curious about testing the input and including an else statement instead of a default statement at the end of my main function switch statement which will be powering the main menu interface response.

I'm going to attempt to also write this using source control, setting up my own public repository on my Github profile, so I can practice using Git and eventually post this final project on fledgling portfolio website.

I put together a basic organization calendar for how I plan be developing my program with a preliminary google calendar chart:

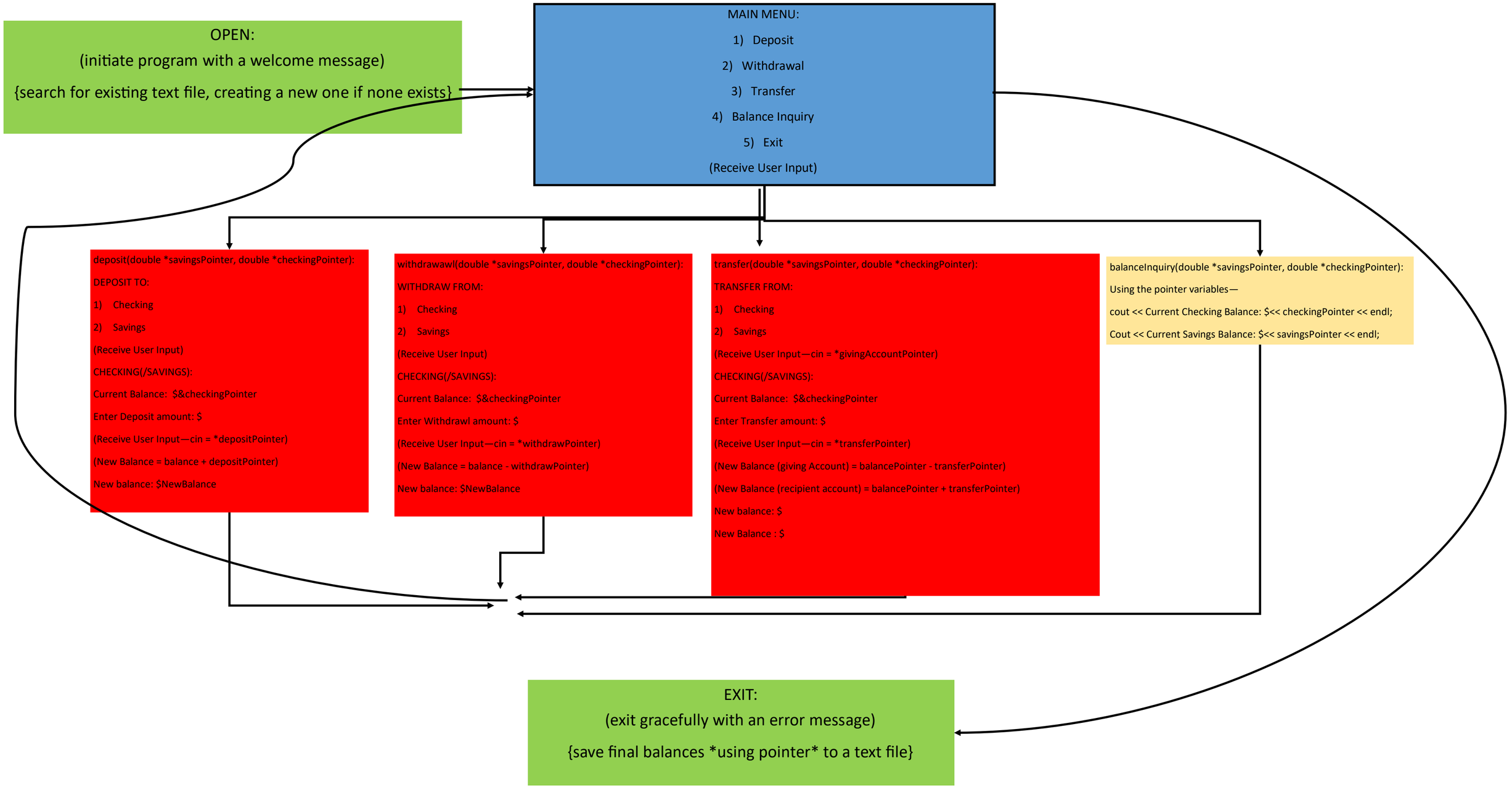


I am giving myself lots of time in the last two days before the submission is due, to write up and document my final program. I want to provide an in-depth user manual as a pdf with screenshots, and reasons on why I made the decisions on how to format and execute the program as I did.

The enhancements I would like to add to my ATM project are the following in order of priority:

1) Error Handling – prevent the infinite invalid input loop I had in my Part 1 Submission
2) Increase the aesthetics of the interface – add different colors for the console output display text, extra spacing in between the various options in the main menu, and sub menu. I am absolutely drawing inspiration from Michael Taylor's screenshots and examples of the enhanced aesthetics of the UI in his Piazza post from Wednesday
3) Store account balance information in a file so that the amounts can remain after exiting the program, a text file which the program will open, create if not existing, and close with current balance information at the end of the program's execution as it "exits gracefully back to the console"
4) Add limits to how much can be deposited and withdraw in a single session
5) Add a pin number, which defaults to "0000" in the initial account setup, and then can be changed as an additional option in the main menu, and then saved onto the text file which contains the account balances.

I created a basic wireframe chart for the workflow of my ATM program, which I will attach in my Piazza Post and submit with this pdf as the final page.


OPEN:
(initiate program with a welcome message)
{search for existing text file, creating a new one if none exists}
MAIN MENU:
1) Deposit
2) Withdrawal
3) Transfer
4) Balance Inquiry
5) Exit
(Receive User Input)
deposit(double *savingsPointer, double *checkingPointer):
DEPOSIT TO:
1) Checking
2) Savings
(Receive User Input)
CHECKING(/SAVINGS):
Current Balance: $&checkingPointer
Enter Deposit amount: $
(Receive User Input—cin = *depositPointer)
(New Balance = balance + depositPointer)
New balance: $NewBalance
withdrawawl(double *savingsPointer, double *checkingPointer):
WITHDRAW FROM:
1) Checking
2) Savings
(Receive User Input)
CHECKING(/SAVINGS):
Current Balance: $&checkingPointer
Enter Withdrawl amount: $
(Receive User Input—cin = *withdrawPointer)
(New Balance = balance - withdrawPointer)
New balance: $NewBalance
transfer(double *savingsPointer, double *checkingPointer):
TRANSFER FROM:
1) Checking
2) Savings
(Receive User Input—cin = *givingAccountPointer)
CHECKING(/SAVINGS):
Current Balance: $&checkingPointer
Enter Transfer amount: $
(Receive User Input—cin = *transferPointer)
(New Balance (giving Account) = balancePointer - transferPointer)
(New Balance (recipient account) = balancePointer + transferPointer)
New balance: $
New Balance : $
balanceInquiry(double *savingsPointer, double *checkingPointer):
Using the pointer variables—
cout << Current Checking Balance: $<< checkingPointer << endl;
Cout << Current Savings Balance: $<< savingsPointer << endl;
EXIT:
(exit gracefully with an error message)
{save final balances *using pointer* to a text file}