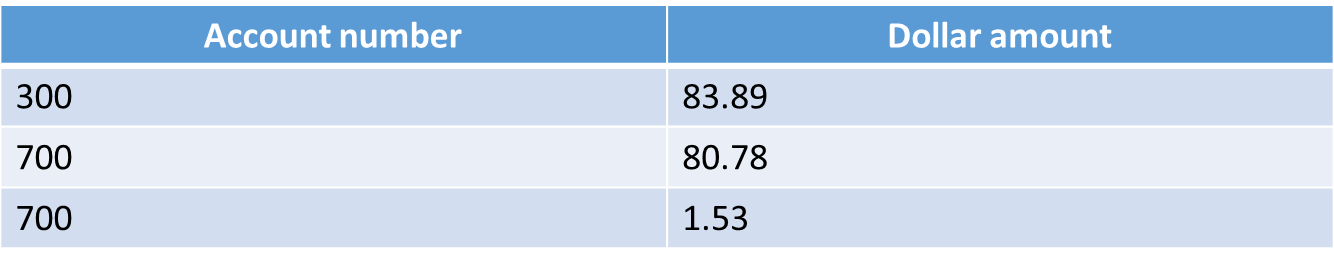
1. Write a program that merges the contents of two text files in such a way that the contents of file2 must be followed by the contents of file 1.
2. **(File Matching)** As transactions occur (i.e., sales are made and cash payments arrive in the mail), they’re entered into a file. At the end of each business period (i.e., a month for some companies, a week for others and a day in some cases) the file of transactions (called “trans.txt”) is applied to the master file (called “oldmast.txt”), thus updating each account’s record of purchases and payments. After each of these updates runs, the master file is rewritten as a new file(“newmast.txt”), which is them used at the end of the next business period to begin the updating process again.

File-matching programs must deal with certain problems that do not exist in single-file programs. For example, a match does not always occur. A customer on the master file might not have made any purchases or cash payments in the current business period, and therefore no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payment might have just moved to this community, and the company may not have had a chance to create a master record for this customer.

When a match occurs (i.e., records with the same account number appear on both the master file and the transaction file), add the dollar amount on the transaction file to the current balance on the master file and write the “newmast.txt” record. (Assume that purchases are indicated by positive amounts on the transaction file, and that payments are indicated by negative amounts.) When there’s a master record for a particular account but no corresponding transaction record, merely write the master record, print the message “Unmatched transaction record for account number…..,” (fill in the account number from the transaction record).

1. **(File Matching with Multiple Transactions)** It’s possible (actually common) to have several transaction records with the same record key. This occurs because a particular customer might make several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program of question 1 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of question 1 to include the following additional transaction records:



1. (Simulation: The Tortoise and the Hare) In this problem, you will implement the simulation for the classic race of the tortoise and the hare. You will use random number generation to develop the simulation. The contenders begin the race at “square 1”of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground. There is a clock that ticks once per second. With each tick of the clock, your program should adjust the position of the animals according to the rules as follows:



Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (i.e., the“starting gate”). If an animal slips left before square 1, move the animal back to square 1. Generate the percentages in the preceding table by producing a random integer, i, in the range 1 ≤ i ≤ 10. For the tortoise, perform a“fast plod”when 1 ≤ i ≤ 5, a“slip”when 6 ≤ i ≤ 7, or a“slow plod”when 8 ≤ i ≤ 10. Use a similar technique to move the hare.

Begin the race by printing

BANG !!!!!

AND THEY'RE OFF !!!!!

Then, for each tick of the clock (i.e., each repetition of a loop), print a 70-position line showing the letter T in the position of the tortoise and the letter H in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare and your program should print OUCH!!! beginning at that position. All print positions other than the T, the H, or the OUCH!!! (in case of a tie) should be blank.

After each line is printed, test whether either animal has reached or passed square 70. If so, then print the winner and terminate the simulation. If the tortoise wins, print TORTOISE WINS!!! YAY!!! If the hare wins, print Hare wins. Yuch. If both animals win on the same tick of the clock, print It's a tie. If neither animal wins, perform the loop again to simulate the next tick of the clock.