AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

International Credit Hour Programs
(ICHEP)

Computer Programming (CSE131)

Spring 2023

Major Task Milestone 2

NAME	ID
Mahmoud Elsayd Eldwakhly	21P0017
Yousef Ahmed Hesham Hammad	21P0088
Omar Elshawaf Abdel Mohsen	21P0410

Sec: 1, Group: 1

Submitted to:

Dr. Hossam El-Din Hassan.
Eng. Kyrelloss Nashaat.

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <ctime>
#include <fstream>           // for output file
#include <cmath>
using namespace std;
struct Transaction {
    string type;
    int duration;
};

struct Customer {
    int arrivalTime;
    int serviceTime;
    int waitingTime;
    int tellerAssigned;
    bool served;
}
```

- **The first struct**, Transaction, has two member variables: type, which is a string that represents the type of transaction, and duration, which is an integer that represents the duration of the transaction in minutes.
- **The second struct**, Customer, has five member variables: arrivalTime, which is an integer that represents the time the customer arrives.
  serviceTime, which is an integer that represents the time the customer starts to be served.
  waitingTime, which is an integer that represents the time the customer waits before being served.
  tellerAssigned, which is an integer that represents the index of the teller assigned to serve the customer.
  bool served, which is a boolean that represents whether the customer has been served or not.

```
void get_user_input(int& numTellers, int& tellerWorkingHours, int& bankOpenTime, int&
bankCloseTime, int& break_start_time, int& break_end_time, int& numCustomersPerday,
int& numservedCustomersPerYear, int& numTransactions,  Transaction transactions[])
{
    cout << "Enter the number of tellers : ";
    cin >> numTellers;
    cout << "Enter the working hours of the bank (open time and close time in hours
past midnight) : ";
    cin >> bankOpenTime >> bankCloseTime;
     while (bankOpenTime < 0 && bankCloseTime > 24)
    {
        cout << "Enter a valid open time and close time in hours past midnight :";
        cin >> bankOpenTime >> bankCloseTime;
    }

    cout << "Enter break start time and break end time in hours : ";
    cin >> break_start_time >> break_end_time ;
     while (break_start_time< bankOpenTime  && bankCloseTime < break_end_time)
    {
        cout << "Enter a valid break start time and break end time in hours between
bank open time and bank close time:";
        cin >> bankOpenTime >> bankCloseTime;
    }
    cout << "Enter the number of customers per day : ";
    cin >> numCustomersPerday;

    cout << "Enter the number of expected served customers per year (must be greater
than 30,000) : ";
    cin >> numservedCustomersPerYear;
    while (numservedCustomersPerYear <= 30000)
    {
        cout << "Number of served customers per year must be greater than 30,000.
Please enter a valid number :";
        cin >> numservedCustomersPerYear;
    }

    for (int i = 0; i < numTransactions; i++)
    {
        cout << "Enter transaction type " << i + 1 << " : ";
        cin >> transactions[i].type;
        cout << "Enter required duration in minutes: ";
        cin >> transactions[i].duration;
    }
    cout << endl << "Transaction types and required durations :" << endl;
    for (int i = 0; i < numTransactions; i++)
    {
        cout << transactions[i].type << " : " << transactions[i].duration << "
minutes" << endl;
    }
}
```

- **get_user_input** that takes in several input parameters by reference and prompts the user to provide values for them. The function also takes in an array of Transaction objects, which the user will provide details for.

Here is a brief explanation of each parameter:

- **numTellers:** an integer passed by reference that represents the number of tellers in the bank.
- **tellerWorkingHours:** an integer passed by reference that represents the number of hours each teller works in a day.
- **bankOpenTime:** an integer passed by reference that represents the time the bank opens in hours past midnight.
- **bankCloseTime:** an integer passed by reference that represents the time the bank closes in hours past midnight.
- **break_start_time:** an integer passed by reference that represents the time the tellers take a break in hours past midnight.
- **break_end_time:** an integer passed by reference that represents the time the tellers return from their break in hours past midnight.
- **numCustomersPerday:** an integer passed by reference that represents the number of customers the bank serves per day.
- **numservedCustomersPerYear:** an integer passed by reference that represents the expected number of customers the bank serves per year.
- **numTransactions:** an integer that represents the number of different types of transactions the bank handles.
- **transactions:** an array of Transaction objects that holds details about each type of transaction.

- The function prompts the user for input values for each parameter using cout and cin statements. It also includes input validation loops to ensure that the user enters valid values for some of the parameters, such as the bank open and close times and the number of served customers per year.

- After all input values have been obtained, the function displays the transaction types and their required durations using a for loop and cout statements.

```
void generate_customer_data(int numTellers, int bankOpenTime, int
bankCloseTime,int break_start_time, int break_end_time, int
numCustomersPerday,
int numTransactions,  Transaction transactions[] ,  int&
numservedCustomersPerYear)
{
    const int NUM_DAYS = 1;
    const int NUM_CUSTOMERS = numCustomersPerday * NUM_DAYS;
    Customer customers[NUM_CUSTOMERS];

    int randomNumber = 0;
    int waitingTime ;
    int prevTellerAssigned = 0;
    int prevPrevTellerAssigned = 0;
    bool tellersBusy[numTellers] = {false};
    int tellerIdleTime[numTellers] = {0};
    int tellerWorkingTime[numTellers] = {0};
    int numServedCustomers = 0;

    for (int day = 1; day <= NUM_DAYS; day++)     // We can change it if we
want to run the whole program in more than 1 day.
    {
        numServedCustomers = 0;

        for (int i = 0; i < numCustomersPerday; i++)
        {
            int arrivalTime = randomNumber + (rand() % ((bankCloseTime -
bankOpenTime) * 60 / numCustomersPerday));
            int size = rand() % numTransactions;
            int serviceTime = transactions[size].duration;
            int arrivalHours = bankOpenTime + arrivalTime / 60;
            int arrivalMinutes = arrivalTime % 60;


            if (i == 0 || i < numTellers )
            {
                waitingTime = 0;
            }
            else
          {

                waitingTime = fabs(arrivalTime - customers[i - 1].arrivalTime
- customers[i - 1].serviceTime); // customer that have the same teller
             }

            customers[i].arrivalTime = arrivalTime;
            customers[i].serviceTime = serviceTime;
            customers[i].waitingTime = waitingTime;
            int leavingHours = bankOpenTime + (customers[i].arrivalTime +
customers[i].serviceTime+customers[i].waitingTime ) / 60;
            int leavingMinutes = (customers[i].arrivalTime +
customers[i].serviceTime+ customers[i].waitingTime) % 60;
```

```
         if ( leavingHours >= bankCloseTime)
                  customers[i].served = false;
             else
                customers[i].served = true;
         if(leavingHours >=break_start_time && leavingHours < break_end_time
){
                         customers[i].served = false;
         }
         if(customers[i].served == false){
            customers[i].waitingTime= 0;
            customers[i].serviceTime= 1; // one minutes to go to teller then
teller reject transaction type due to not working hour
         }
            int assignedTeller;
            do
               {
                assignedTeller = rand() % numTellers + 1;
               }

            while (assignedTeller == prevTellerAssigned || assignedTeller ==
prevPrevTellerAssigned || tellersBusy[assignedTeller - 1]);

            tellersBusy[assignedTeller - 1] = true;

            customers[i].tellerAssigned = assignedTeller;

            tellerWorkingTime[assignedTeller - 1] +=
customers[i].serviceTime;
            tellersBusy[assignedTeller - 1] = false;

            prevPrevTellerAssigned = prevTellerAssigned;
            prevTellerAssigned = assignedTeller;


                                              cout << "Customer " << (i + 1)
<< " arrives at: " << setw(2) << setfill('0') << arrivalHours << ":" <<
setw(2) << setfill('0') << arrivalMinutes
                 << " and finishes at " << setw(2) << setfill('0') <<
leavingHours << ":" << setw(2) << leavingMinutes<<endl
                 << "Using transaction " << transactions[size].type
                 << ", Teller " << customers[i].tellerAssigned << " assigned"
<< endl;

            if (customers[i].served)
            {
                numServedCustomers++;
            }

            randomNumber += ((bankCloseTime - bankOpenTime) * 60 /
numCustomersPerday);
        }
        cout << " Today "  << ": " << numServedCustomers << " customers 
served." << endl;
        int z =  numServedCustomers*365 ;

         cout << "No. of served customers per year "  << ": " << z << " 
customers served." << endl;
         if ( numservedCustomersPerYear > z )
         cout << "The bank does not meet your expectations " << endl ;
         else
         cout << "The bank meets your expectations " << endl ;

    }

    ofstream outputFile("customer_data.txt");
    if (outputFile.is_open())
    {
        outputFile << "Customer ID    Arrival Time        Leaving Time 
Waiting Time   Served?" << endl;
        for (int i = 0; i < NUM_CUSTOMERS; i++)
        {
            int arrivalHours = bankOpenTime + customers[i].arrivalTime / 60;
            int arrivalMinutes = customers[i].arrivalTime % 60;
            int leavingHours = bankOpenTime + (customers[i].arrivalTime + 
customers[i].serviceTime+customers[i].waitingTime ) / 60;
            int leavingMinutes = (customers[i].arrivalTime + 
customers[i].serviceTime+ customers[i].waitingTime) % 60;

            outputFile << setw(5) << i + 1 << setw(13) << arrivalHours << ":" 
<< setw(2) << arrivalMinutes << "    "
                       << setw(15) << leavingHours << ":" << setw(2) << 
leavingMinutes << "    "
                       << setw(15) << customers[i].waitingTime << "    "
                       << setw(9) << (customers[i].served ? "Yes" : "No") << 
endl;
        }
        outputFile.close();
        cout << "Customer data written to file: customer_data.txt" << endl;

        ofstream outputTellerFile("teller_data.txt");
        if (outputTellerFile.is_open())
        {
            outputTellerFile << "Teller ID    Idle Time        Working Time 
Utility (%)" << endl;
            for (int i = 0; i < numTellers; i++)
            {// edit
                int idleTime = (bankCloseTime - bankOpenTime) * 60 - 
tellerWorkingTime[i] + (break_end_time -break_start_time)*60 ;
                double utility = tellerWorkingTime[i] * 100.0 / 
(bankCloseTime - bankOpenTime)  / 60.0 ;
                outputTellerFile << setw(5) << i + 1 << setw(13) << idleTime 
<< "    "
                           << setw(15) << tellerWorkingTime[i] << "    "
                           << setw(9) << fixed << setprecision(2) << utility
<< endl;
            if (idleTime<0)
            outputTellerFile<< " This teller "<< i+1 << " has overload of "
<< fabs(idleTime) << " minutes " << "to complete his assigned customers "<<
"Due to the completely randomized teller assignment "<<endl;
            }
            outputTellerFile.close();
            cout << "Teller data written to file: teller_data.txt" << endl;
        }
        else
        {
            cout << "Error opening file." << endl;
        }
    }
    else
    {
        cout << "Error opening file." << endl;
    }
}
```

- **generate_customer_data** that takes in several input parameters and generates data for simulating a bank's customer service process. The function also writes the generated data to two output files: customer_data.txt and teller_data.txt.
- **numTellers:** an integer that represents the number of tellers in the bank.
- **bankOpenTime:** an integer that represents the time the bank opens in hours past midnight.
- **bankCloseTime:** an integer that represents the time the bank closes in hours past midnight.
- **break_start_time:** an integer that represents the time the tellers take a break in hours past midnight.
- **break_end_time:** an integer that represents the time the tellers return from their break in hours past midnight.
- **numCustomersPerday:** an integer that represents the number of customers the bank serves per day.
- **numTransactions:** an integer that represents the number of different types of transactions the bank handles.
- **transactions:** an array of Transaction objects that holds details about each type of transaction.
- **numservedCustomersPerYear:** an integer passed by reference that represents the expected number of customers the bank serves per year.
- The function first defines some constant values, including the number of days (NUM_DAYS) and the total number of customers (NUM_CUSTOMERS) to be generated based on the numCustomersPerday parameter.

- Thefunction then initializes several variables, including randomNumber, waitingTime, prevTellerAssigned, prevPrevTellerAssigned, tellersBusy, tellerIdleTime, tellerWorkingTime, and numServedCustomers.

- The function then uses nested for loops to generate data for each customer. It calculates the customer's arrivalTime randomly within a certain time range, chooses a random transaction size, and calculates the serviceTime for that transaction type. It then calculates the waitingTime based on the time the previous customer was served, and sets the arrivalTime, serviceTime, waitingTime, and served values for the current customer.

- The function then uses a do-while loop to assign a teller to the customer randomly, ensuring that the same teller is not assigned twice in a row and that the teller is not currently serving another customer. It updates the tellersBusy array and calculates the teller's tellerWorkingTime.

- The function then outputs the details of the current customer, including the arrival and leaving times, the transaction type, and the assigned teller.

- The function keeps track of the number of served customers and writes this value to the console at the end of each day. It also checks if the total number of served customers for the year meets the expected value provided by numservedCustomersPerYear.

- The function then writes the details of each customer to the customer_data.txt file, including the customer's ID, arrival time, leaving time, waiting time, and whether they were served or not. It also calculates and writes the details of each teller to the teller_data.txt file, including the teller ID, idle time, working time, and utility percentage. If a teller has a negative idle time, indicating an overload, the function outputs a message to the console.

- Finally, the function closes both output files and outputs messages to the console indicating that the customer and teller data have been written to their respective files.

```
int main()
{
    srand(time(0));
    int numTellers, tellerWorkingHours, bankOpenTime, bankCloseTime,
numCustomersPerday, numservedCustomersPerYear, numTransactions,
break_start_time, break_end_time;//break_time , wa

    cout << "How many transactions will you enter? ";
    cin >> numTransactions;
    Transaction transactions[numTransactions];
    get_user_input(numTellers, tellerWorkingHours, bankOpenTime,
bankCloseTime, break_start_time, break_end_time, numCustomersPerday,
numservedCustomersPerYear, numTransactions, transactions);
    generate_customer_data(numTellers, bankOpenTime, bankCloseTime,
break_start_time, break_end_time,  numCustomersPerday,
    numTransactions,  transactions ,  numservedCustomersPerYear);
    return 0;
}
```

- This is a simple main function that uses the **get_user_input** and **generate_customer_data** functions to simulate a bank's customer service process.

- The function first seeds the random number generator using the current time, and then declares several integer variables: numTellers, tellerWorkingHours, bankOpenTime, bankCloseTime, numCustomersPerday, numservedCustomersPerYear, numTransactions, break_start_time, and break_end_time.

- It then prompts the user to enter the number of transactions using cout and cin, and creates an array of numTransactions Transaction objects using the Transaction struct.

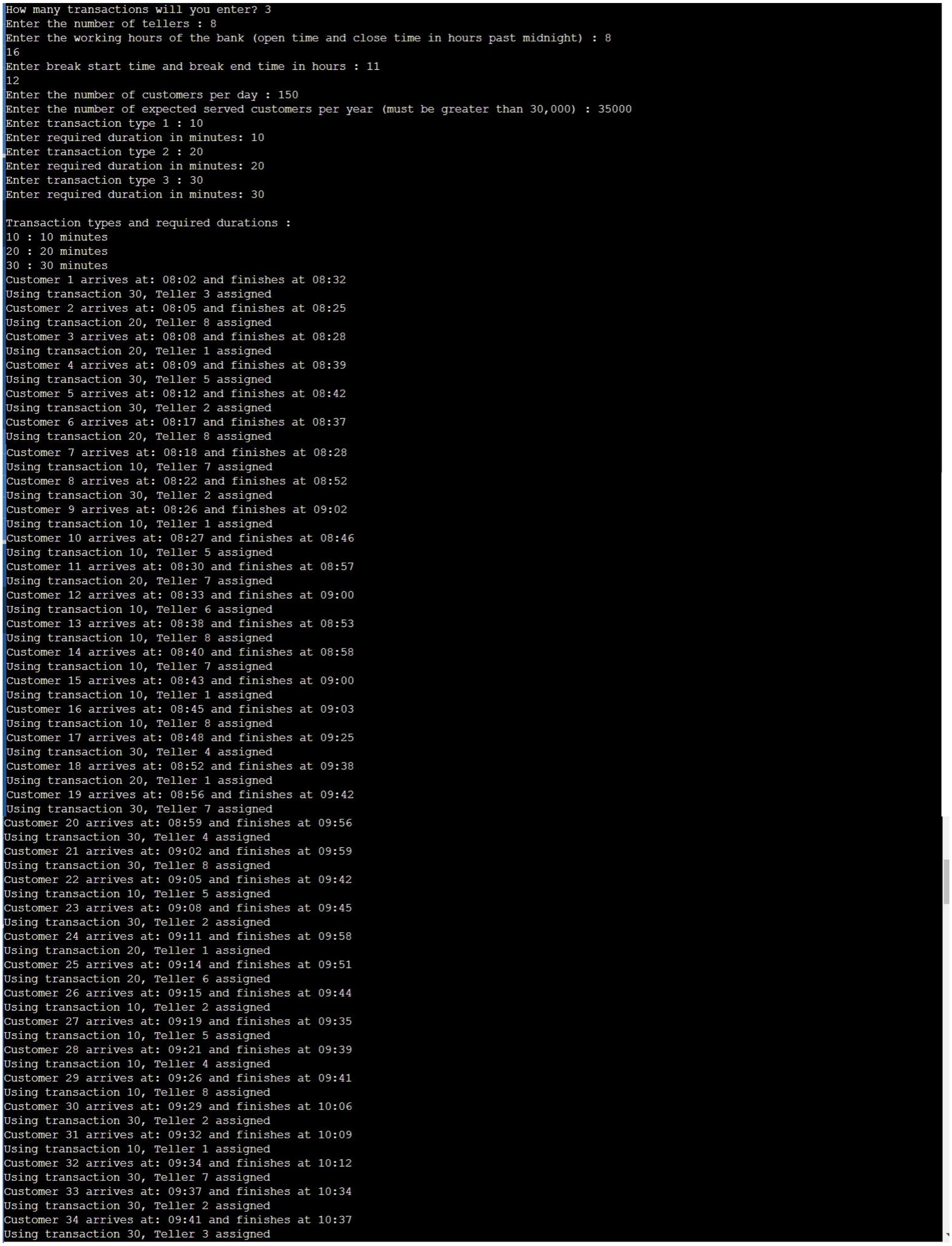
- The function then calls the get_user_input function to get the remaining input parameters from the user, passing in the numTransactions and transactions objects as parameters.

- Finally, it calls the generate_customer_data function, passing in all the input parameters and the transactions array.

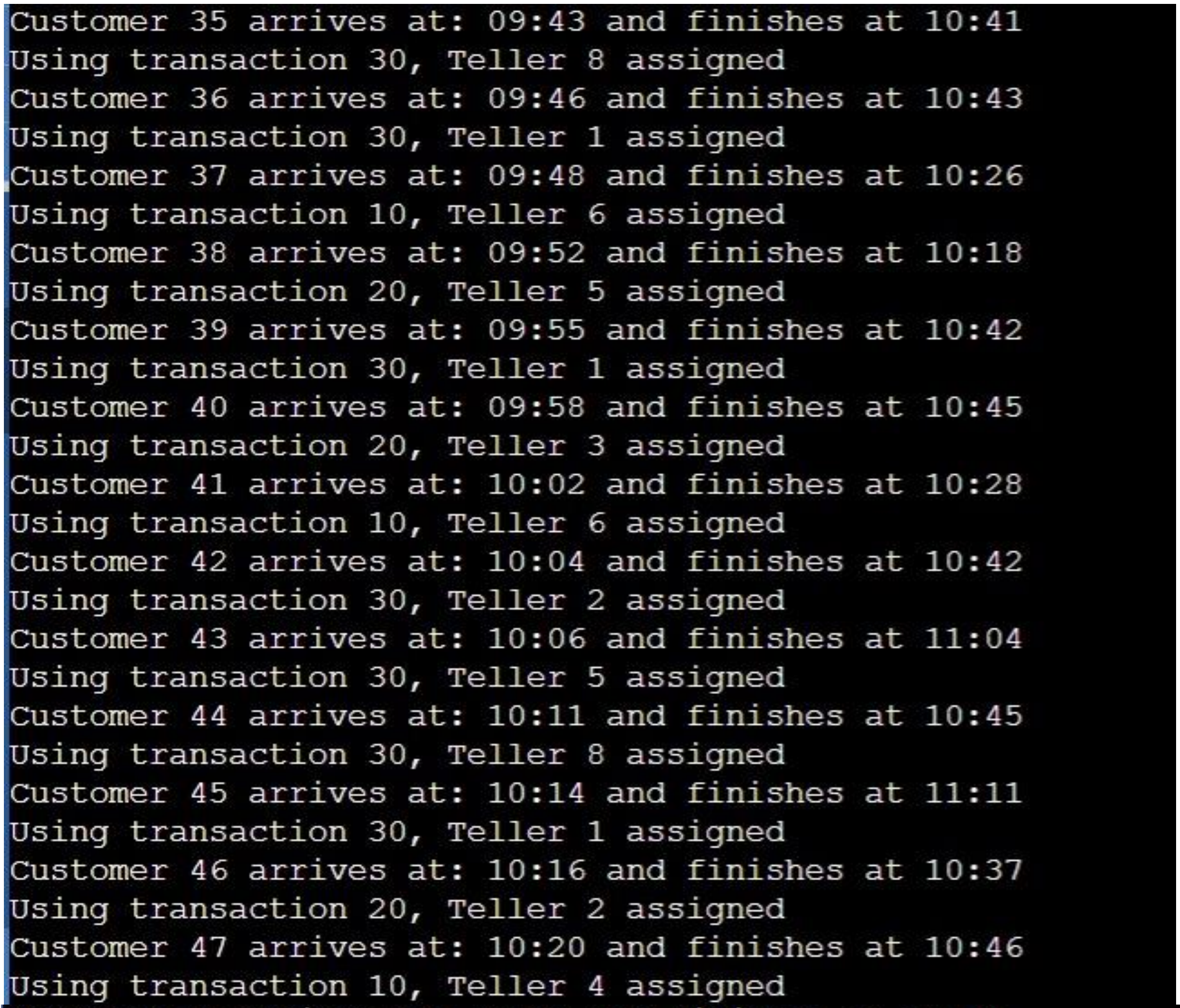
- The main function then returns 0, indicating successful program completion.

- Overall, this is a simple main function that initializes input variables, prompts the user for input, and calls the appropriate functions to simulate a bank's customer service process.

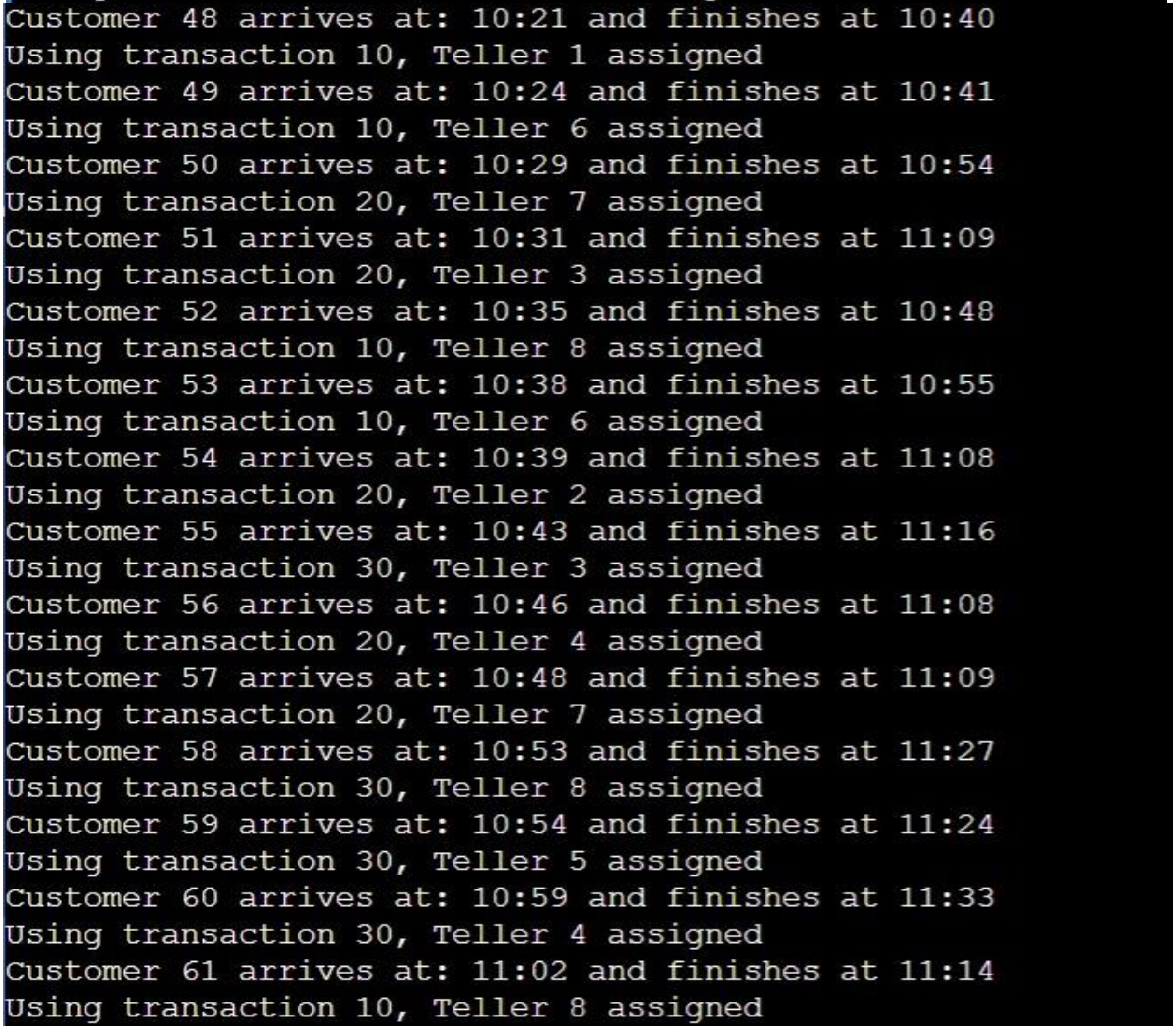
This is the output:



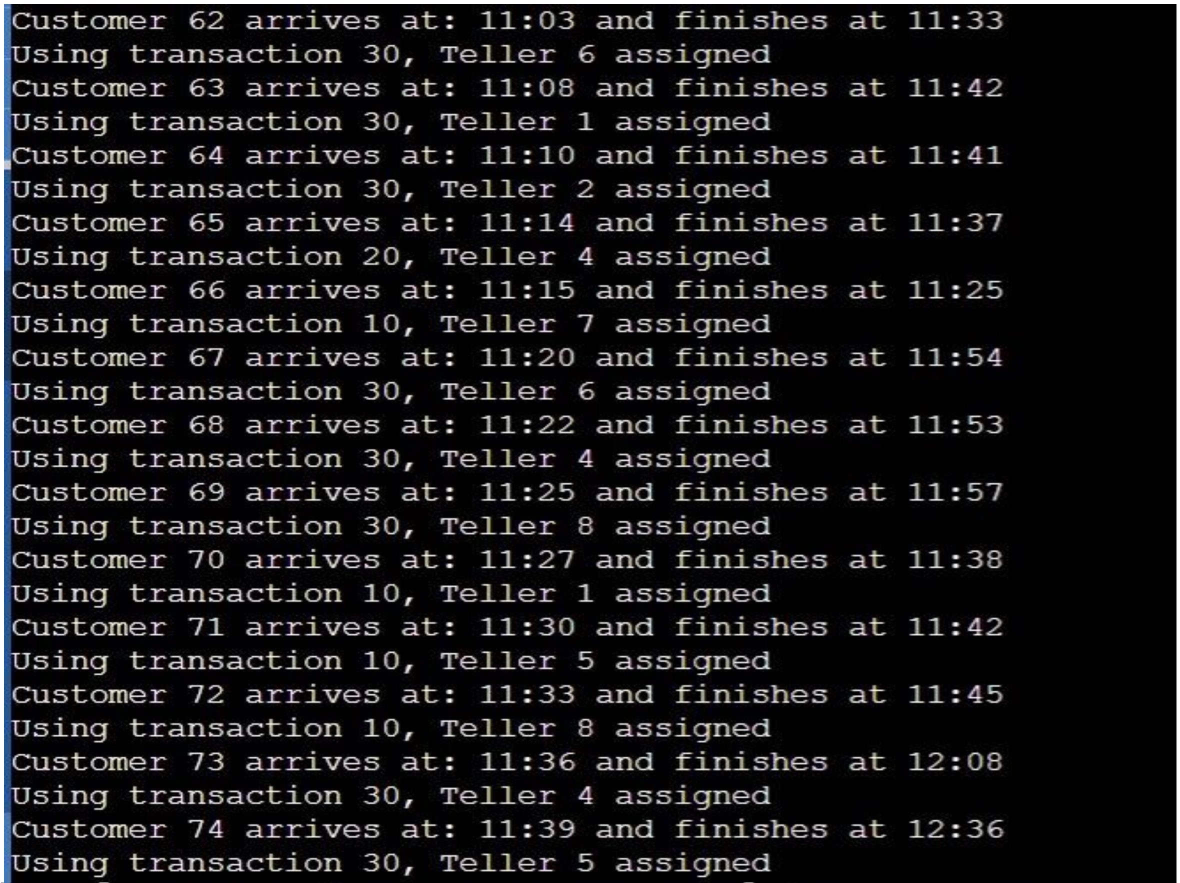
```
How many transactions will you enter? 3
Enter the number of tellers : 8
Enter the working hours of the bank (open time and close time in hours past midnight) : 8
16
Enter break start time and break end time in hours : 11
12
Enter the number of customers per day : 150
Enter the number of expected served customers per year (must be greater than 30,000) : 35000
Enter transaction type 1 : 10
Enter required duration in minutes: 10
Enter transaction type 2 : 20
Enter required duration in minutes: 20
Enter transaction type 3 : 30
Enter required duration in minutes: 30

Transaction types and required durations :
10 : 10 minutes
20 : 20 minutes
30 : 30 minutes
Customer 1 arrives at: 08:02 and finishes at 08:32
Using transaction 30, Teller 3 assigned
Customer 2 arrives at: 08:05 and finishes at 08:25
Using transaction 20, Teller 8 assigned
Customer 3 arrives at: 08:08 and finishes at 08:28
Using transaction 20, Teller 1 assigned
Customer 4 arrives at: 08:09 and finishes at 08:39
Using transaction 30, Teller 5 assigned
Customer 5 arrives at: 08:12 and finishes at 08:42
Using transaction 30, Teller 2 assigned
Customer 6 arrives at: 08:17 and finishes at 08:37
Using transaction 20, Teller 8 assigned
Customer 7 arrives at: 08:18 and finishes at 08:28
Using transaction 10, Teller 7 assigned
Customer 8 arrives at: 08:22 and finishes at 08:52
Using transaction 30, Teller 2 assigned
Customer 9 arrives at: 08:26 and finishes at 09:02
Using transaction 10, Teller 1 assigned
Customer 10 arrives at: 08:27 and finishes at 08:46
Using transaction 10, Teller 5 assigned
Customer 11 arrives at: 08:30 and finishes at 08:57
Using transaction 20, Teller 7 assigned
Customer 12 arrives at: 08:33 and finishes at 09:00
Using transaction 10, Teller 6 assigned
Customer 13 arrives at: 08:38 and finishes at 08:53
Using transaction 10, Teller 8 assigned
Customer 14 arrives at: 08:40 and finishes at 08:58
Using transaction 10, Teller 7 assigned
Customer 15 arrives at: 08:43 and finishes at 09:00
Using transaction 10, Teller 1 assigned
Customer 16 arrives at: 08:45 and finishes at 09:03
Using transaction 10, Teller 8 assigned
Customer 17 arrives at: 08:48 and finishes at 09:25
Using transaction 30, Teller 4 assigned
Customer 18 arrives at: 08:52 and finishes at 09:38
Using transaction 20, Teller 1 assigned
Customer 19 arrives at: 08:56 and finishes at 09:42
Using transaction 30, Teller 7 assigned
Customer 20 arrives at: 08:59 and finishes at 09:56
Using transaction 30, Teller 4 assigned
Customer 21 arrives at: 09:02 and finishes at 09:59
Using transaction 30, Teller 8 assigned
Customer 22 arrives at: 09:05 and finishes at 09:42
Using transaction 10, Teller 5 assigned
Customer 23 arrives at: 09:08 and finishes at 09:45
Using transaction 30, Teller 2 assigned
Customer 24 arrives at: 09:11 and finishes at 09:58
Using transaction 20, Teller 1 assigned
Customer 25 arrives at: 09:14 and finishes at 09:51
Using transaction 20, Teller 6 assigned
Customer 26 arrives at: 09:15 and finishes at 09:44
Using transaction 10, Teller 2 assigned
Customer 27 arrives at: 09:19 and finishes at 09:35
Using transaction 10, Teller 5 assigned
Customer 28 arrives at: 09:21 and finishes at 09:39
Using transaction 10, Teller 4 assigned
Customer 29 arrives at: 09:26 and finishes at 09:41
Using transaction 10, Teller 8 assigned
Customer 30 arrives at: 09:29 and finishes at 10:06
Using transaction 30, Teller 2 assigned
Customer 31 arrives at: 09:32 and finishes at 10:09
Using transaction 10, Teller 1 assigned
Customer 32 arrives at: 09:34 and finishes at 10:12
Using transaction 30, Teller 7 assigned
Customer 33 arrives at: 09:37 and finishes at 10:34
Using transaction 30, Teller 2 assigned
Customer 34 arrives at: 09:41 and finishes at 10:37
Using transaction 30, Teller 3 assigned
```



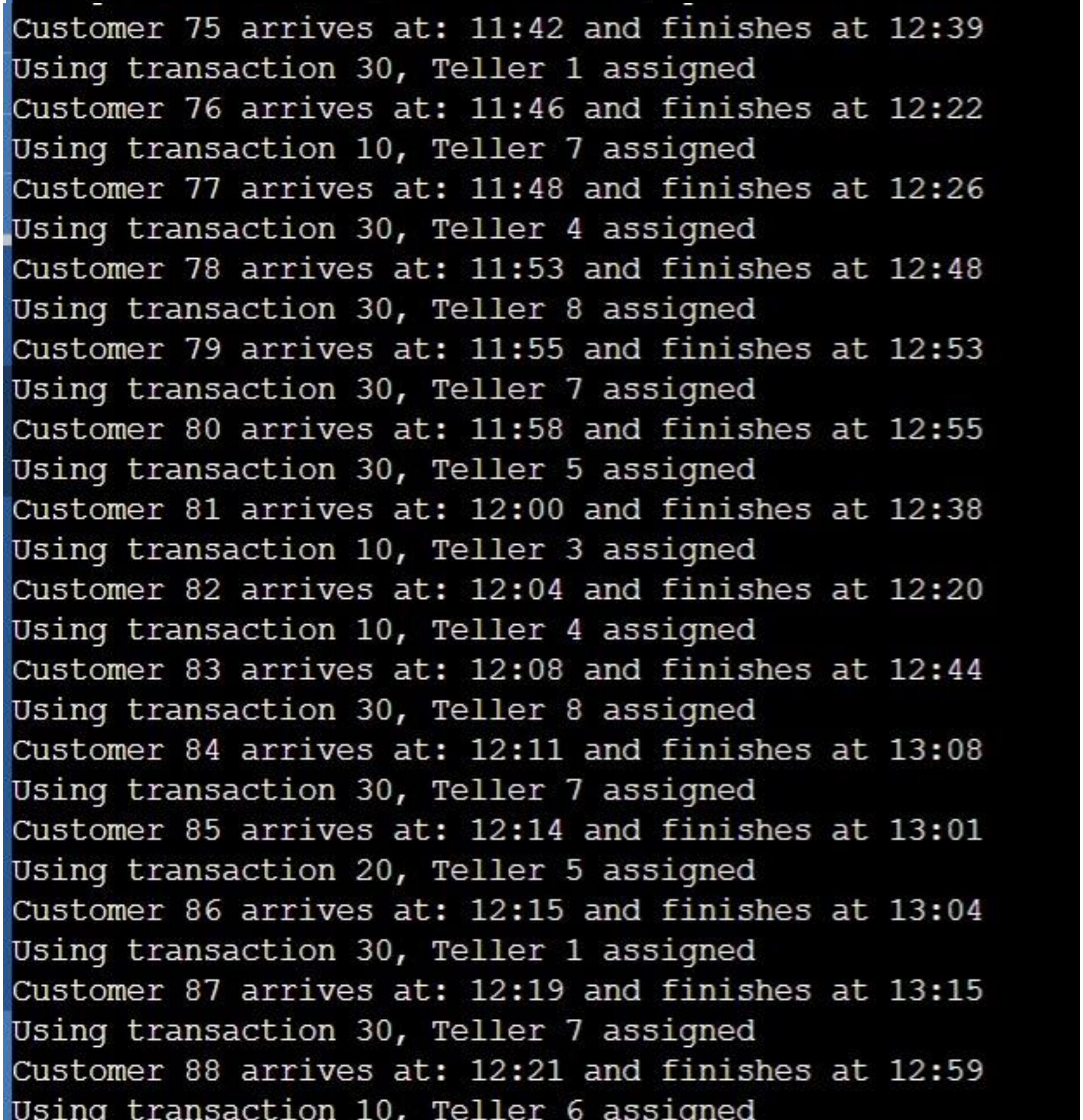
```
Customer 35 arrives at: 09:43 and finishes at 10:41
Using transaction 30, Teller 8 assigned
Customer 36 arrives at: 09:46 and finishes at 10:43
Using transaction 30, Teller 1 assigned
Customer 37 arrives at: 09:48 and finishes at 10:26
Using transaction 10, Teller 6 assigned
Customer 38 arrives at: 09:52 and finishes at 10:18
Using transaction 20, Teller 5 assigned
Customer 39 arrives at: 09:55 and finishes at 10:42
Using transaction 30, Teller 1 assigned
Customer 40 arrives at: 09:58 and finishes at 10:45
Using transaction 20, Teller 3 assigned
Customer 41 arrives at: 10:02 and finishes at 10:28
Using transaction 10, Teller 6 assigned
Customer 42 arrives at: 10:04 and finishes at 10:42
Using transaction 30, Teller 2 assigned
Customer 43 arrives at: 10:06 and finishes at 11:04
Using transaction 30, Teller 5 assigned
Customer 44 arrives at: 10:11 and finishes at 10:45
Using transaction 30, Teller 8 assigned
Customer 45 arrives at: 10:14 and finishes at 11:11
Using transaction 30, Teller 1 assigned
Customer 46 arrives at: 10:16 and finishes at 10:37
Using transaction 20, Teller 2 assigned
Customer 47 arrives at: 10:20 and finishes at 10:46
Using transaction 10, Teller 4 assigned
```



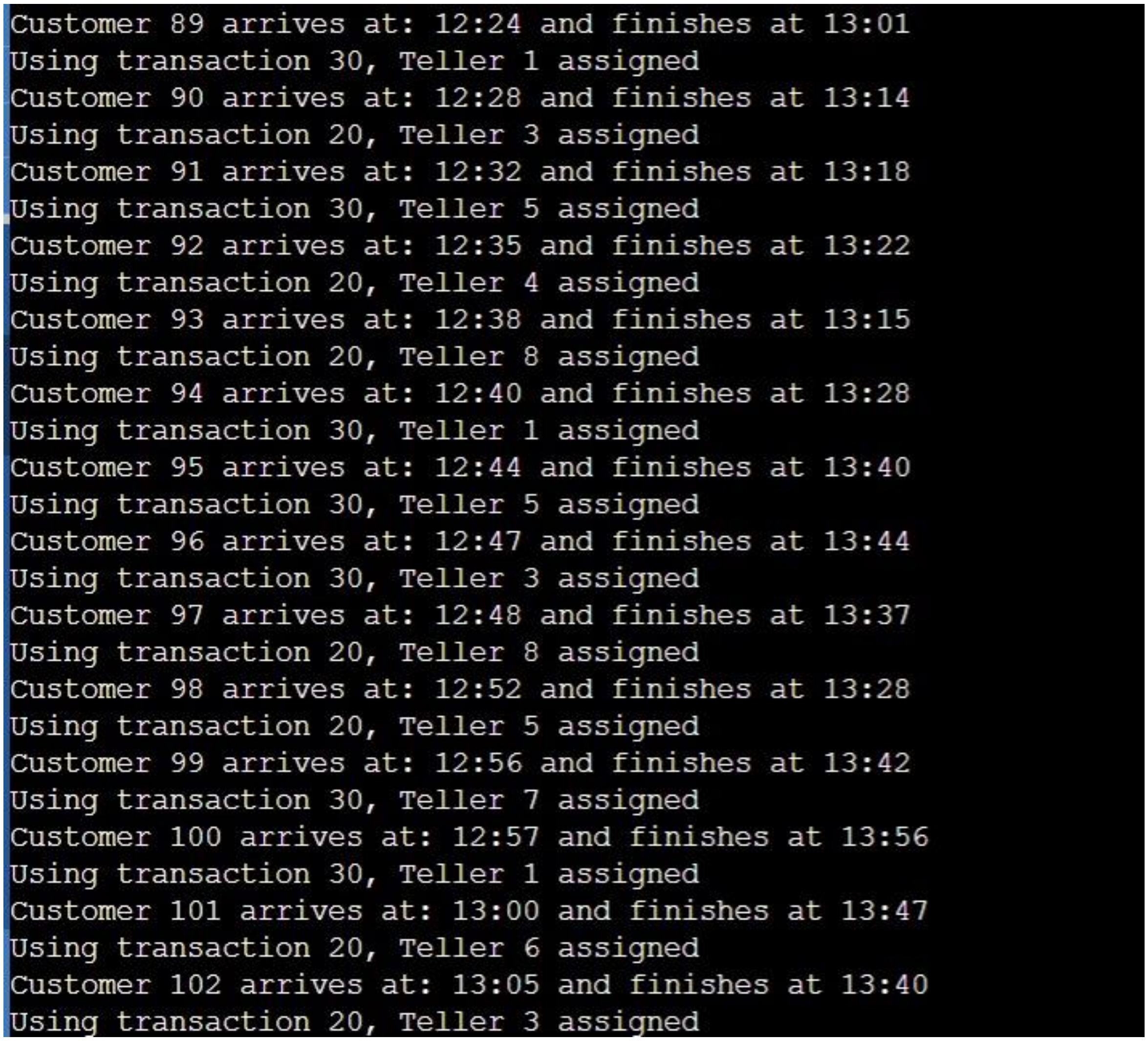
```
Customer 48 arrives at: 10:21 and finishes at 10:40
Using transaction 10, Teller 1 assigned
Customer 49 arrives at: 10:24 and finishes at 10:41
Using transaction 10, Teller 6 assigned
Customer 50 arrives at: 10:29 and finishes at 10:54
Using transaction 20, Teller 7 assigned
Customer 51 arrives at: 10:31 and finishes at 11:09
Using transaction 20, Teller 3 assigned
Customer 52 arrives at: 10:35 and finishes at 10:48
Using transaction 10, Teller 8 assigned
Customer 53 arrives at: 10:38 and finishes at 10:55
Using transaction 10, Teller 6 assigned
Customer 54 arrives at: 10:39 and finishes at 11:08
Using transaction 20, Teller 2 assigned
Customer 55 arrives at: 10:43 and finishes at 11:16
Using transaction 30, Teller 3 assigned
Customer 56 arrives at: 10:46 and finishes at 11:08
Using transaction 20, Teller 4 assigned
Customer 57 arrives at: 10:48 and finishes at 11:09
Using transaction 20, Teller 7 assigned
Customer 58 arrives at: 10:53 and finishes at 11:27
Using transaction 30, Teller 8 assigned
Customer 59 arrives at: 10:54 and finishes at 11:24
Using transaction 30, Teller 5 assigned
Customer 60 arrives at: 10:59 and finishes at 11:33
Using transaction 30, Teller 4 assigned
Customer 61 arrives at: 11:02 and finishes at 11:14
Using transaction 10, Teller 8 assigned
```



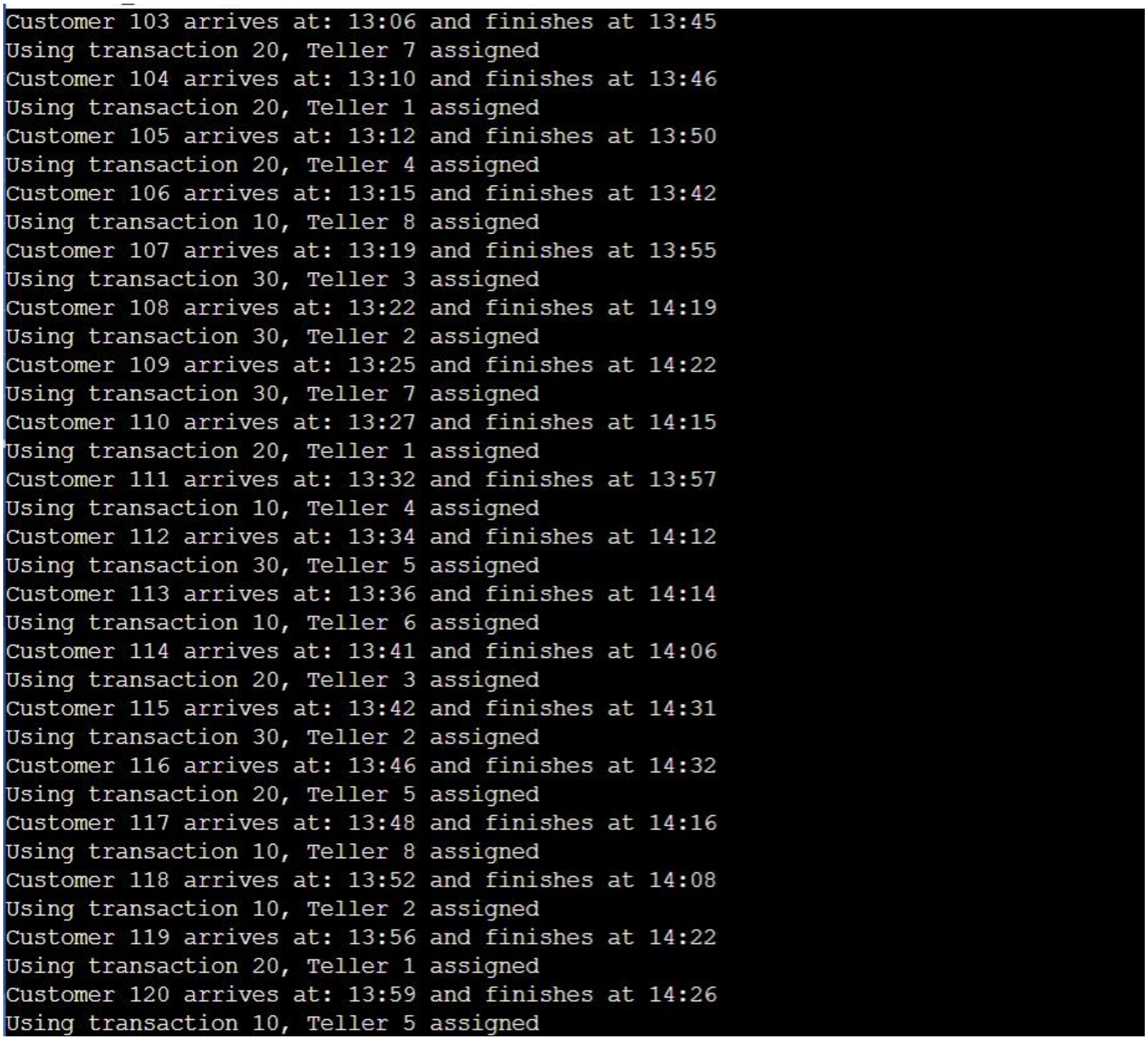
```
Customer 62 arrives at: 11:03 and finishes at 11:33
Using transaction 30, Teller 6 assigned
Customer 63 arrives at: 11:08 and finishes at 11:42
Using transaction 30, Teller 1 assigned
Customer 64 arrives at: 11:10 and finishes at 11:41
Using transaction 30, Teller 2 assigned
Customer 65 arrives at: 11:14 and finishes at 11:37
Using transaction 20, Teller 4 assigned
Customer 66 arrives at: 11:15 and finishes at 11:25
Using transaction 10, Teller 7 assigned
Customer 67 arrives at: 11:20 and finishes at 11:54
Using transaction 30, Teller 6 assigned
Customer 68 arrives at: 11:22 and finishes at 11:53
Using transaction 30, Teller 4 assigned
Customer 69 arrives at: 11:25 and finishes at 11:57
Using transaction 30, Teller 8 assigned
Customer 70 arrives at: 11:27 and finishes at 11:38
Using transaction 10, Teller 1 assigned
Customer 71 arrives at: 11:30 and finishes at 11:42
Using transaction 10, Teller 5 assigned
Customer 72 arrives at: 11:33 and finishes at 11:45
Using transaction 10, Teller 8 assigned
Customer 73 arrives at: 11:36 and finishes at 12:08
Using transaction 30, Teller 4 assigned
Customer 74 arrives at: 11:39 and finishes at 12:36
Using transaction 30, Teller 5 assigned
```



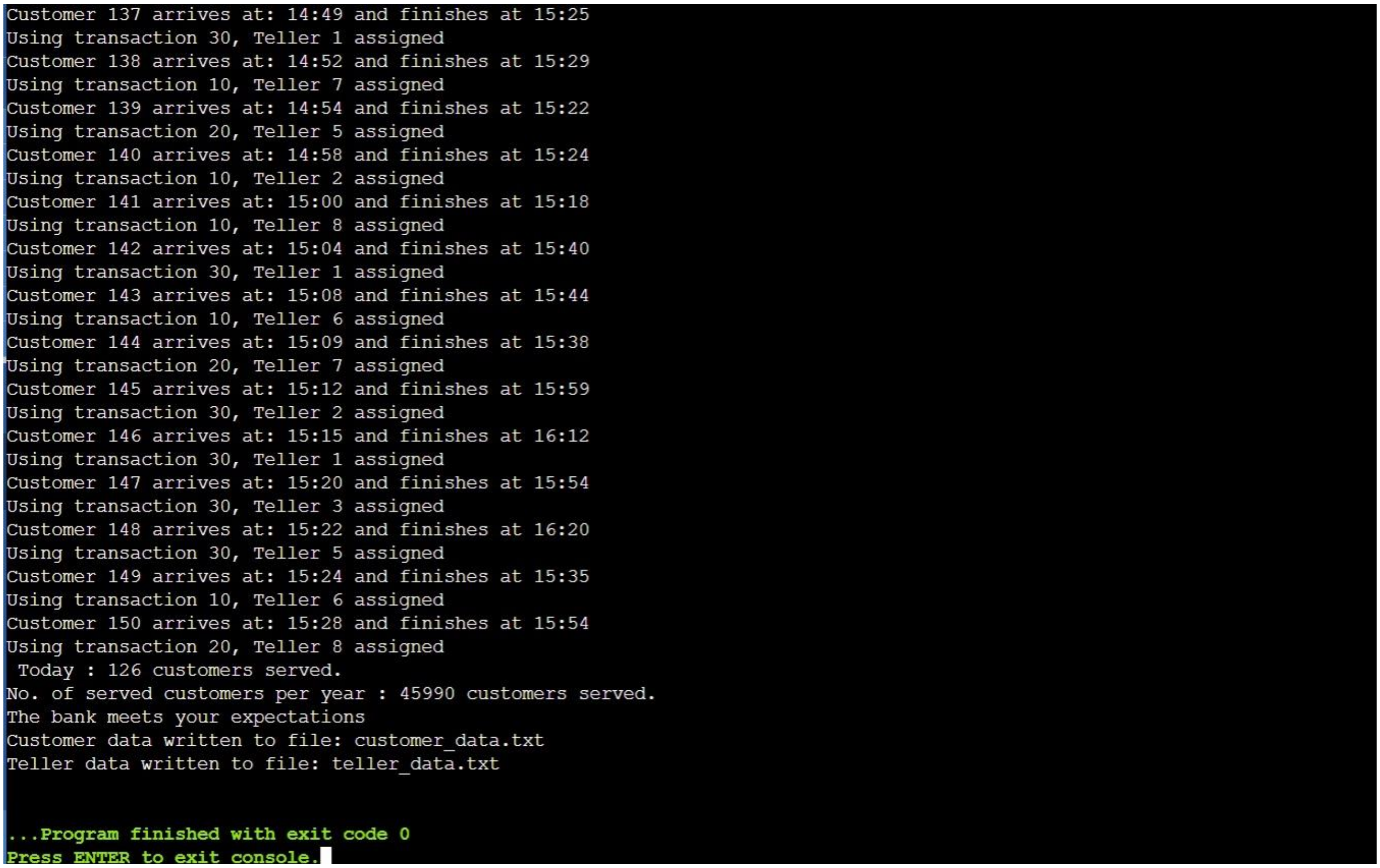
```
Customer 75 arrives at: 11:42 and finishes at 12:39
Using transaction 30, Teller 1 assigned
Customer 76 arrives at: 11:46 and finishes at 12:22
Using transaction 10, Teller 7 assigned
Customer 77 arrives at: 11:48 and finishes at 12:26
Using transaction 30, Teller 4 assigned
Customer 78 arrives at: 11:53 and finishes at 12:48
Using transaction 30, Teller 8 assigned
Customer 79 arrives at: 11:55 and finishes at 12:53
Using transaction 30, Teller 7 assigned
Customer 80 arrives at: 11:58 and finishes at 12:55
Using transaction 30, Teller 5 assigned
Customer 81 arrives at: 12:00 and finishes at 12:38
Using transaction 10, Teller 3 assigned
Customer 82 arrives at: 12:04 and finishes at 12:20
Using transaction 10, Teller 4 assigned
Customer 83 arrives at: 12:08 and finishes at 12:44
Using transaction 30, Teller 8 assigned
Customer 84 arrives at: 12:11 and finishes at 13:08
Using transaction 30, Teller 7 assigned
Customer 85 arrives at: 12:14 and finishes at 13:01
Using transaction 20, Teller 5 assigned
Customer 86 arrives at: 12:15 and finishes at 13:04
Using transaction 30, Teller 1 assigned
Customer 87 arrives at: 12:19 and finishes at 13:15
Using transaction 30, Teller 7 assigned
Customer 88 arrives at: 12:21 and finishes at 12:59
Using transaction 10, Teller 6 assigned
```



```
Customer 89 arrives at: 12:24 and finishes at 13:01
Using transaction 30, Teller 1 assigned
Customer 90 arrives at: 12:28 and finishes at 13:14
Using transaction 20, Teller 3 assigned
Customer 91 arrives at: 12:32 and finishes at 13:18
Using transaction 30, Teller 5 assigned
Customer 92 arrives at: 12:35 and finishes at 13:22
Using transaction 20, Teller 4 assigned
Customer 93 arrives at: 12:38 and finishes at 13:15
Using transaction 20, Teller 8 assigned
Customer 94 arrives at: 12:40 and finishes at 13:28
Using transaction 30, Teller 1 assigned
Customer 95 arrives at: 12:44 and finishes at 13:40
Using transaction 30, Teller 5 assigned
Customer 96 arrives at: 12:47 and finishes at 13:44
Using transaction 30, Teller 3 assigned
Customer 97 arrives at: 12:48 and finishes at 13:37
Using transaction 20, Teller 8 assigned
Customer 98 arrives at: 12:52 and finishes at 13:28
Using transaction 20, Teller 5 assigned
Customer 99 arrives at: 12:56 and finishes at 13:42
Using transaction 30, Teller 7 assigned
Customer 100 arrives at: 12:57 and finishes at 13:56
Using transaction 30, Teller 1 assigned
Customer 101 arrives at: 13:00 and finishes at 13:47
Using transaction 20, Teller 6 assigned
Customer 102 arrives at: 13:05 and finishes at 13:40
Using transaction 20, Teller 3 assigned
```

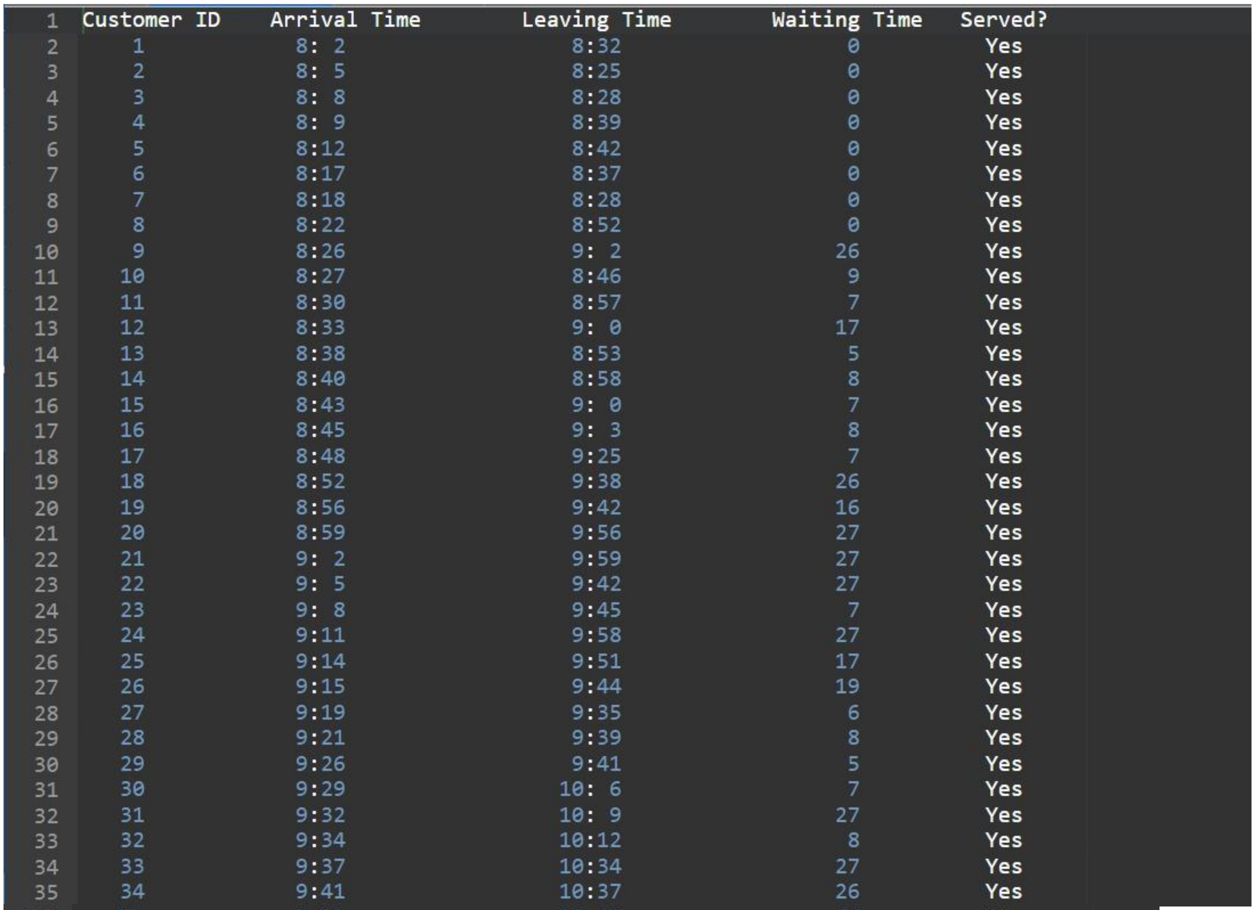


```
Customer 103 arrives at: 13:06 and finishes at 13:45
Using transaction 20, Teller 7 assigned
Customer 104 arrives at: 13:10 and finishes at 13:46
Using transaction 20, Teller 1 assigned
Customer 105 arrives at: 13:12 and finishes at 13:50
Using transaction 20, Teller 4 assigned
Customer 106 arrives at: 13:15 and finishes at 13:42
Using transaction 10, Teller 8 assigned
Customer 107 arrives at: 13:19 and finishes at 13:55
Using transaction 30, Teller 3 assigned
Customer 108 arrives at: 13:22 and finishes at 14:19
Using transaction 30, Teller 2 assigned
Customer 109 arrives at: 13:25 and finishes at 14:22
Using transaction 30, Teller 7 assigned
Customer 110 arrives at: 13:27 and finishes at 14:15
Using transaction 20, Teller 1 assigned
Customer 111 arrives at: 13:32 and finishes at 13:57
Using transaction 10, Teller 4 assigned
Customer 112 arrives at: 13:34 and finishes at 14:12
Using transaction 30, Teller 5 assigned
Customer 113 arrives at: 13:36 and finishes at 14:14
Using transaction 10, Teller 6 assigned
Customer 114 arrives at: 13:41 and finishes at 14:06
Using transaction 20, Teller 3 assigned
Customer 115 arrives at: 13:42 and finishes at 14:31
Using transaction 30, Teller 2 assigned
Customer 116 arrives at: 13:46 and finishes at 14:32
Using transaction 20, Teller 5 assigned
Customer 117 arrives at: 13:48 and finishes at 14:16
Using transaction 10, Teller 8 assigned
Customer 118 arrives at: 13:52 and finishes at 14:08
Using transaction 10, Teller 2 assigned
Customer 119 arrives at: 13:56 and finishes at 14:22
Using transaction 20, Teller 1 assigned
Customer 120 arrives at: 13:59 and finishes at 14:26
Using transaction 10, Teller 5 assigned
```

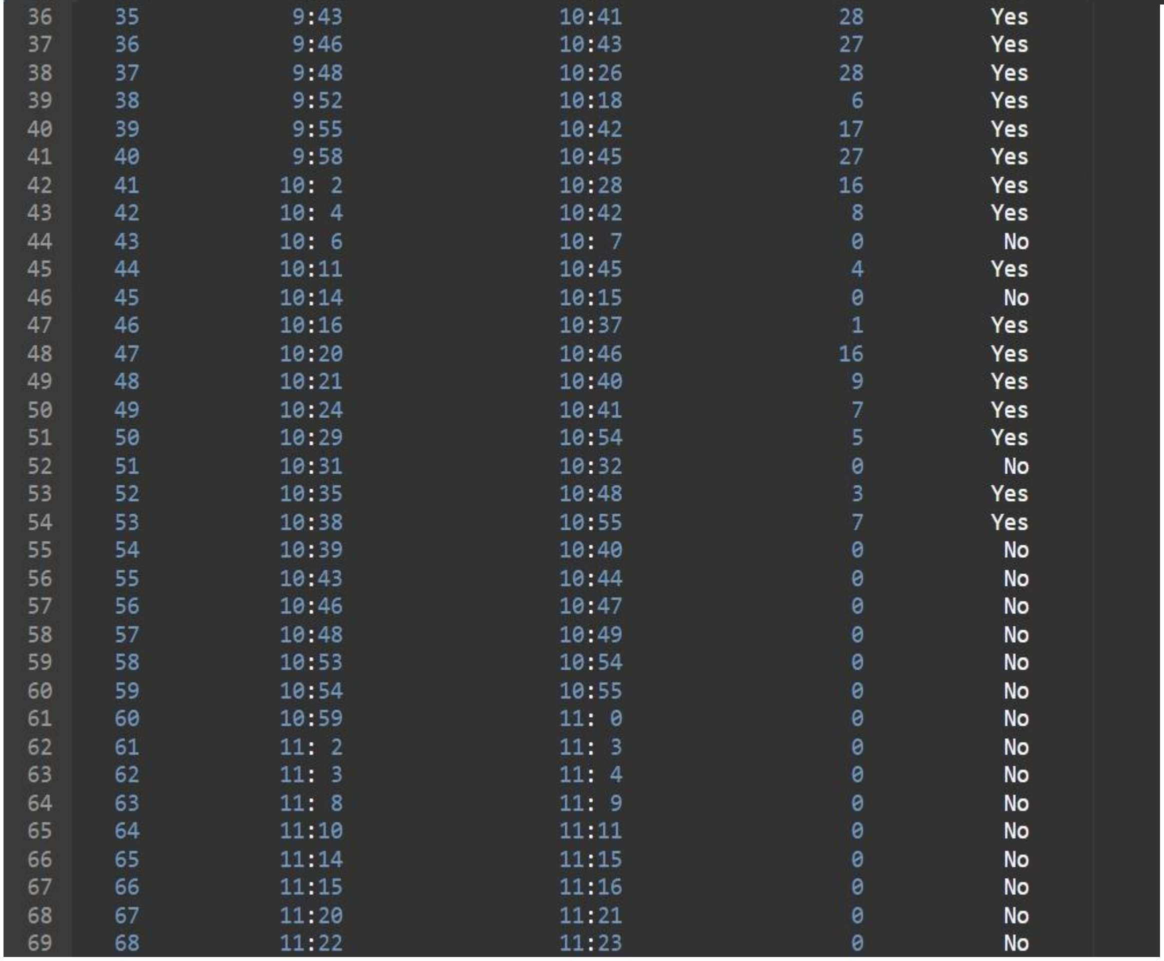


```
Customer 137 arrives at: 14:49 and finishes at 15:25
Using transaction 30, Teller 1 assigned
Customer 138 arrives at: 14:52 and finishes at 15:29
Using transaction 10, Teller 7 assigned
Customer 139 arrives at: 14:54 and finishes at 15:22
Using transaction 20, Teller 5 assigned
Customer 140 arrives at: 14:58 and finishes at 15:24
Using transaction 10, Teller 2 assigned
Customer 141 arrives at: 15:00 and finishes at 15:18
Using transaction 10, Teller 8 assigned
Customer 142 arrives at: 15:04 and finishes at 15:40
Using transaction 30, Teller 1 assigned
Customer 143 arrives at: 15:08 and finishes at 15:44
Using transaction 10, Teller 6 assigned
Customer 144 arrives at: 15:09 and finishes at 15:38
Using transaction 20, Teller 7 assigned
Customer 145 arrives at: 15:12 and finishes at 15:59
Using transaction 30, Teller 2 assigned
Customer 146 arrives at: 15:15 and finishes at 16:12
Using transaction 30, Teller 1 assigned
Customer 147 arrives at: 15:20 and finishes at 15:54
Using transaction 30, Teller 3 assigned
Customer 148 arrives at: 15:22 and finishes at 16:20
Using transaction 30, Teller 5 assigned
Customer 149 arrives at: 15:24 and finishes at 15:35
Using transaction 10, Teller 6 assigned
Customer 150 arrives at: 15:28 and finishes at 15:54
Using transaction 20, Teller 8 assigned
 Today : 126 customers served.
No. of served customers per year : 45990 customers served.
The bank meets your expectations
Customer data written to file: customer_data.txt
Teller data written to file: teller_data.txt


...Program finished with exit code 0
Press ENTER to exit console.
```

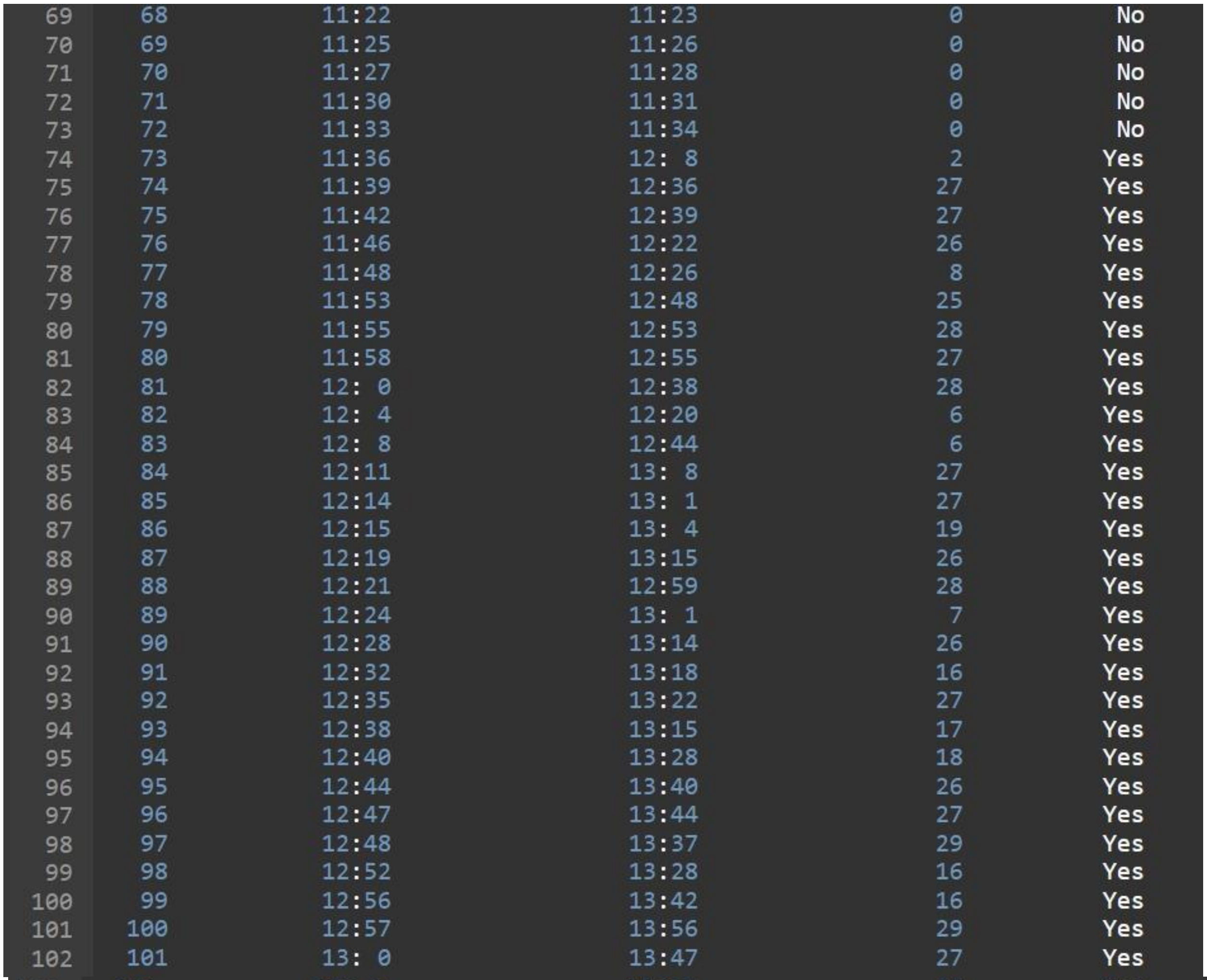
This is the output file of customer



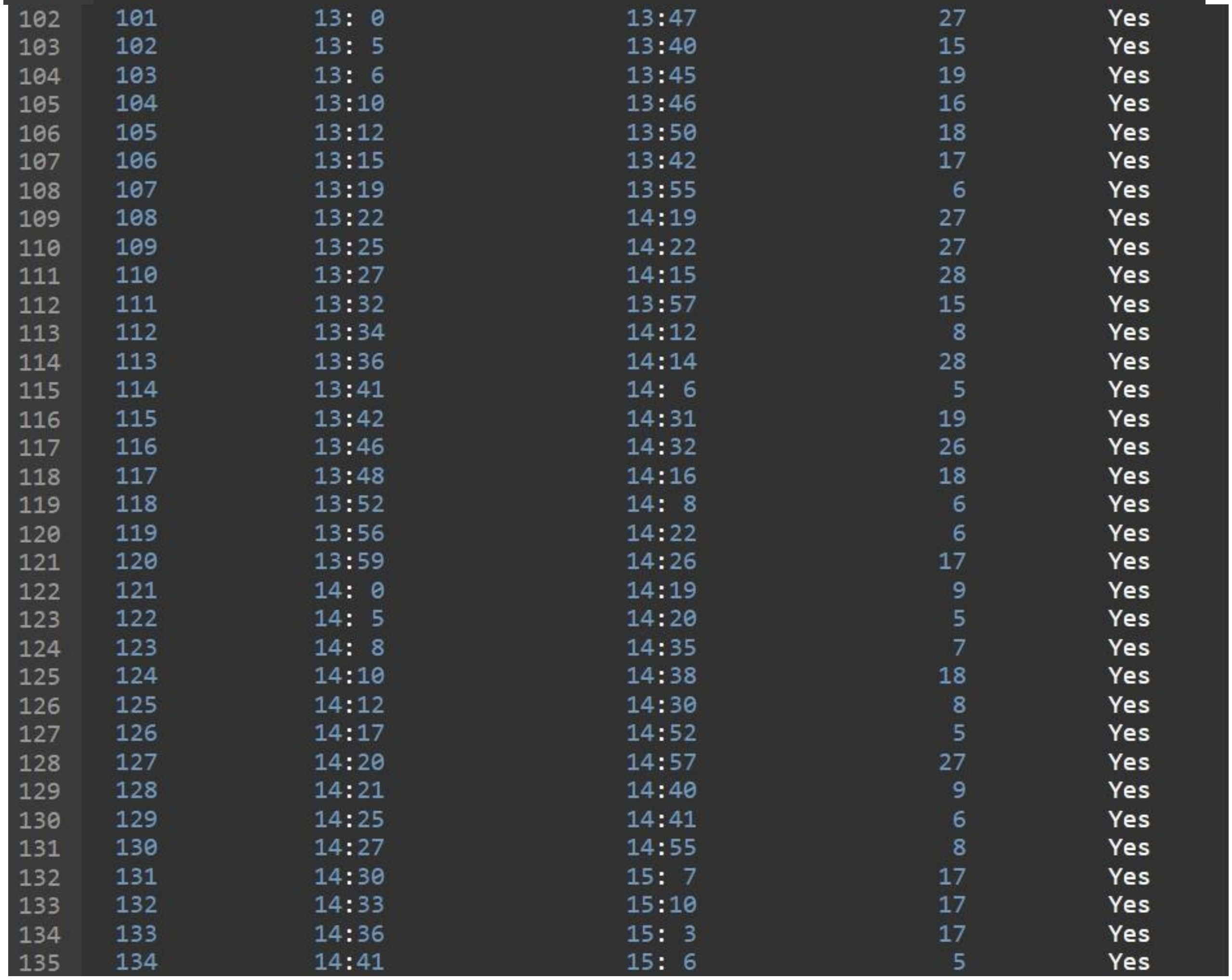
```
Customer ID    Arrival Time        Leaving Time        Waiting Time    Served?
     1            8: 2                8:32                  0            Yes
     2            8: 5                8:25                  0            Yes
     3            8: 8                8:28                  0            Yes
     4            8: 9                8:39                  0            Yes
     5            8:12                8:42                  0            Yes
     6            8:17                8:37                  0            Yes
     7            8:18                8:28                  0            Yes
     8            8:22                8:52                  0            Yes
     9            8:26                9: 2                 26            Yes
    10            8:27                8:46                  9            Yes
    11            8:30                8:57                  7            Yes
    12            8:33                9: 0                 17            Yes
    13            8:38                8:53                  5            Yes
    14            8:40                8:58                  8            Yes
    15            8:43                9: 0                  7            Yes
    16            8:45                9: 3                  8            Yes
    17            8:48                9:25                  7            Yes
    18            8:52                9:38                 26            Yes
    19            8:56                9:42                 16            Yes
    20            8:59                9:56                 27            Yes
    21            9: 2                9:59                 27            Yes
    22            9: 5                9:42                 27            Yes
    23            9: 8                9:45                  7            Yes
    24            9:11                9:58                 27            Yes
    25            9:14                9:51                 17            Yes
    26            9:15                9:44                 19            Yes
    27            9:19                9:35                  6            Yes
    28            9:21                9:39                  8            Yes
    29            9:26                9:41                  5            Yes
    30            9:29               10: 6                  7            Yes
    31            9:32               10: 9                 27            Yes
    32            9:34               10:12                  8            Yes
    33            9:37               10:34                 27            Yes
    34            9:41               10:37                 26            Yes
```



```
    35            9:43               10:41                 28            Yes
    36            9:46               10:43                 27            Yes
    37            9:48               10:26                 28            Yes
    38            9:52               10:18                  6            Yes
    39            9:55               10:42                 17            Yes
    40            9:58               10:45                 27            Yes
    41           10: 2               10:28                 16            Yes
    42           10: 4               10:42                  8            Yes
    43           10: 6               10: 7                  0             No
    44           10:11               10:45                  4            Yes
    45           10:14               10:15                  0             No
    46           10:16               10:37                  1            Yes
    47           10:20               10:46                 16            Yes
    48           10:21               10:40                  9            Yes
    49           10:24               10:41                  7            Yes
    50           10:29               10:54                  5            Yes
    51           10:31               10:32                  0             No
    52           10:35               10:48                  3            Yes
    53           10:38               10:55                  7            Yes
    54           10:39               10:40                  0             No
    55           10:43               10:44                  0             No
    56           10:46               10:47                  0             No
    57           10:48               10:49                  0             No
    58           10:53               10:54                  0             No
    59           10:54               10:55                  0             No
    60           10:59               11: 0                  0             No
    61           11: 2               11: 3                  0             No
    62           11: 3               11: 4                  0             No
    63           11: 8               11: 9                  0             No
    64           11:10               11:11                  0             No
    65           11:14               11:15                  0             No
    66           11:15               11:16                  0             No
    67           11:20               11:21                  0             No
    68           11:22               11:23                  0             No
```

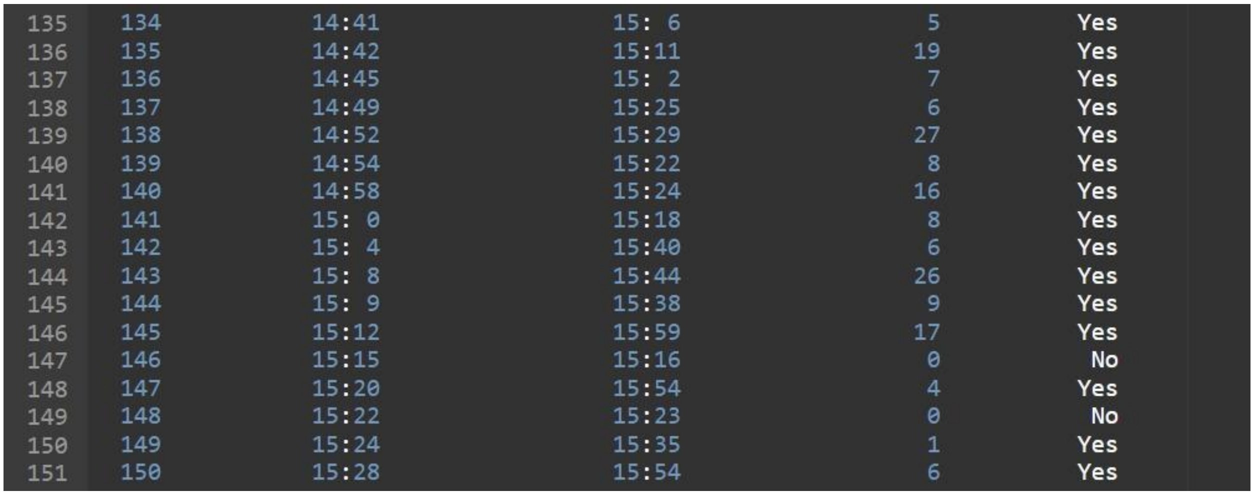


```
 68      11:22      11:23       0     No
 69      11:25      11:26       0     No
 70      11:27      11:28       0     No
 71      11:30      11:31       0     No
 72      11:33      11:34       0     No
 73      11:36      12: 8       2     Yes
 74      11:39      12:36      27     Yes
 75      11:42      12:39      27     Yes
 76      11:46      12:22      26     Yes
 77      11:48      12:26       8     Yes
 78      11:53      12:48      25     Yes
 79      11:55      12:53      28     Yes
 80      11:58      12:55      27     Yes
 81      12: 0      12:38      28     Yes
 82      12: 4      12:20       6     Yes
 83      12: 8      12:44       6     Yes
 84      12:11      13: 8      27     Yes
 85      12:14      13: 1      27     Yes
 86      12:15      13: 4      19     Yes
 87      12:19      13:15      26     Yes
 88      12:21      12:59      28     Yes
 89      12:24      13: 1       7     Yes
 90      12:28      13:14      26     Yes
 91      12:32      13:18      16     Yes
 92      12:35      13:22      27     Yes
 93      12:38      13:15      17     Yes
 94      12:40      13:28      18     Yes
 95      12:44      13:40      26     Yes
 96      12:47      13:44      27     Yes
 97      12:48      13:37      29     Yes
 98      12:52      13:28      16     Yes
 99      12:56      13:42      16     Yes
100      12:57      13:56      29     Yes
101      13: 0      13:47      27     Yes
```



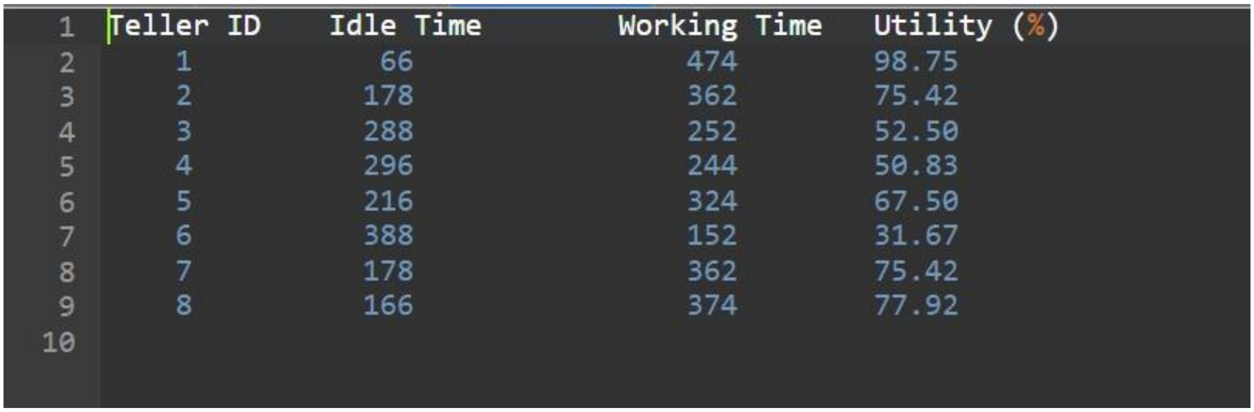
101   13: 0   13:47   27   Yes

```
102      13: 5      13:40      15     Yes
103      13: 6      13:45      19     Yes
104      13:10      13:46      16     Yes
105      13:12      13:50      18     Yes
106      13:15      13:42      17     Yes
107      13:19      13:55       6     Yes
108      13:22      14:19      27     Yes
109      13:25      14:22      27     Yes
110      13:27      14:15      28     Yes
111      13:32      13:57      15     Yes
112      13:34      14:12       8     Yes
113      13:36      14:14      28     Yes
114      13:41      14: 6       5     Yes
115      13:42      14:31      19     Yes
116      13:46      14:32      26     Yes
117      13:48      14:16      18     Yes
118      13:52      14: 8       6     Yes
119      13:56      14:22       6     Yes
120      13:59      14:26      17     Yes
121      14: 0      14:19       9     Yes
122      14: 5      14:20       5     Yes
123      14: 8      14:35       7     Yes
124      14:10      14:38      18     Yes
125      14:12      14:30       8     Yes
126      14:17      14:52       5     Yes
127      14:20      14:57      27     Yes
128      14:21      14:40       9     Yes
129      14:25      14:41       6     Yes
130      14:27      14:55       8     Yes
131      14:30      15: 7      17     Yes
132      14:33      15:10      17     Yes
133      14:36      15: 3      17     Yes
134      14:41      15: 6       5     Yes
```



```
134       14:41          15: 6          5      Yes
135       14:42          15:11         19      Yes
136       14:45          15: 2          7      Yes
137       14:49          15:25          6      Yes
138       14:52          15:29         27      Yes
139       14:54          15:22          8      Yes
140       14:58          15:24         16      Yes
141       15: 0          15:18          8      Yes
142       15: 4          15:40          6      Yes
143       15: 8          15:44         26      Yes
144       15: 9          15:38          9      Yes
145       15:12          15:59         17      Yes
146       15:15          15:16          0       No
147       15:20          15:54          4      Yes
148       15:22          15:23          0       No
149       15:24          15:35          1      Yes
150       15:28          15:54          6      Yes
```

This is the output file of teller



```
Teller ID     Idle Time         Working Time     Utility (%)
    1            66                474           98.75
    2           178                362           75.42
    3           288                252           52.50
    4           296                244           50.83
    5           216                324           67.50
    6           388                152           31.67
    7           178                362           75.42
    8           166                374           77.92
```

Here is the full code:

```
#include <iostream>
#include <iomanip>
#include <cstring>
#include <ctime>
#include <fstream>           // for output file
#include <cmath>
using namespace std;

struct Transaction {
    string type;
    int duration;
};

struct Customer {
    int arrivalTime;
    int serviceTime;
    int waitingTime;
    int tellerAssigned;
    bool served;
};

void get_user_input(int& numTellers, int& tellerWorkingHours, int&
bankOpenTime, int& bankCloseTime, int& break_start_time, int&
break_end_time, int& numCustomersPerday, int& numservedCustomersPerYear,
int& numTransactions,  Transaction transactions[])
{
    cout << "Enter the number of tellers : ";
    cin >> numTellers;
    cout << "Enter the working hours of the bank (open time and close
time in hours past midnight) : ";
    cin >> bankOpenTime >> bankCloseTime;
     while (bankOpenTime < 0 && bankCloseTime > 24)
    {
        cout << "Enter a valid open time and close time in hours past
midnight :";
        cin >> bankOpenTime >> bankCloseTime;
    }

    cout << "Enter break start time and break end time in hours : ";
    cin >> break_start_time >> break_end_time ;
     while (break_start_time< bankOpenTime  && bankCloseTime <
break_end_time)
    {
        cout << "Enter a valid break start time and break end time in
hours between bank open time and bank close time:";
        cin >> bankOpenTime >> bankCloseTime;
    }
    cout << "Enter the number of customers per day : ";
    cin >> numCustomersPerday;

    cout << "Enter the number of expected served customers per year (must
be greater than 30,000) : ";
    cin >> numservedCustomersPerYear;
    while (numservedCustomersPerYear <= 30000)
    {
        cout << "Number of served customers per year must be greater than
30,000. Please enter a valid number :";
        cin >> numservedCustomersPerYear;
    }

    for (int i = 0; i < numTransactions; i++)
    {
        cout << "Enter transaction type " << i + 1 << " : ";
        cin >> transactions[i].type;
        cout << "Enter required duration in minutes: ";
        cin >> transactions[i].duration;
    }

    cout << endl << "Transaction types and required durations :" << endl;
    for (int i = 0; i < numTransactions; i++)
    {
        cout << transactions[i].type << " : " << transactions[i].duration
<< " minutes" << endl;
    }
}

void generate_customer_data(int numTellers, int bankOpenTime, int
bankCloseTime,int break_start_time, int break_end_time, int
numCustomersPerday,
int numTransactions,  Transaction transactions[] ,  int&
numservedCustomersPerYear)
{
    const int NUM_DAYS = 1;
    const int NUM_CUSTOMERS = numCustomersPerday * NUM_DAYS;
    Customer customers[NUM_CUSTOMERS];

    int randomNumber = 0;
    int waitingTime ;
    int prevTellerAssigned = 0;
    int prevPrevTellerAssigned = 0;
    bool tellersBusy[numTellers] = {false};
    int tellerIdleTime[numTellers] = {0};
    int tellerWorkingTime[numTellers] = {0};
    int numServedCustomers = 0;

    for (int day = 1; day <= NUM_DAYS; day++)     // We can change it if
we want to run the whole program in more than 1 day.
    {
        numServedCustomers = 0;

        for (int i = 0; i < numCustomersPerday; i++)
        {
            int arrivalTime = randomNumber + (rand() % ((bankCloseTime -
bankOpenTime) * 60 / numCustomersPerday));
            int size = rand() % numTransactions;
            int serviceTime = transactions[size].duration;
            int arrivalHours = bankOpenTime + arrivalTime / 60;
            int arrivalMinutes = arrivalTime % 60;


            if (i == 0 || i < numTellers )
            {
                waitingTime = 0;
            }
            else
          {
                waitingTime = fabs(arrivalTime - customers[i -
1].arrivalTime - customers[i - 1].serviceTime);
            }

            customers[i].arrivalTime = arrivalTime;
            customers[i].serviceTime = serviceTime;
            customers[i].waitingTime = waitingTime;
            int leavingHours = bankOpenTime + (customers[i].arrivalTime +
customers[i].serviceTime+customers[i].waitingTime ) / 60;
            int leavingMinutes = (customers[i].arrivalTime +
customers[i].serviceTime+ customers[i].waitingTime) % 60;


         if ( leavingHours >= bankCloseTime)
                  customers[i].served = false;
             else
                customers[i].served = true;
         if(leavingHours >=break_start_time && leavingHours <
break_end_time ){
                         customers[i].served = false;
         }
         if(customers[i].served == false){
            customers[i].waitingTime= 0;
            customers[i].serviceTime= 1; // one minutes to go to teller
then teller reject transaction type due to not working hour
         }
            int assignedTeller;
            do
               {
                assignedTeller = rand() % numTellers + 1;
               }

            while (assignedTeller == prevTellerAssigned || assignedTeller
== prevPrevTellerAssigned || tellersBusy[assignedTeller - 1]);

            tellersBusy[assignedTeller - 1] = true;

            customers[i].tellerAssigned = assignedTeller;

            tellerWorkingTime[assignedTeller - 1] +=
customers[i].serviceTime;
            tellersBusy[assignedTeller - 1] = false;

            prevPrevTellerAssigned = prevTellerAssigned;
            prevTellerAssigned = assignedTeller;


                                              cout << "Customer " << (i +
1) << " arrives at: " << setw(2) << setfill('0') << arrivalHours << ":"
<< setw(2) << setfill('0') << arrivalMinutes
                 << " and finishes at " << setw(2) << setfill('0') <<
leavingHours << ":" << setw(2) << leavingMinutes<<endl
                 << "Using transaction " << transactions[size].type
                 << ", Teller " << customers[i].tellerAssigned << "
assigned" << endl;

            if (customers[i].served)
            {
                numServedCustomers++;
            }

            randomNumber += ((bankCloseTime - bankOpenTime) * 60 /
numCustomersPerday);
        }

        cout << " Today "  << ": " << numServedCustomers << " customers
served." << endl;
        int z =  numServedCustomers*365 ;

         cout << "No. of served customers per year "  << ": " << z << "
customers served." << endl;
         if ( numservedCustomersPerYear > z )
         cout << "The bank does not meet your expectations " << endl ;
         else
         cout << "The bank meets your expectations " << endl ;

    }

    ofstream outputFile("customer_data.txt");
    if (outputFile.is_open())
    {
        outputFile << "Customer ID    Arrival Time        Leaving Time
Waiting Time   Served?" << endl;
        for (int i = 0; i < NUM_CUSTOMERS; i++)
        {
            int arrivalHours = bankOpenTime + customers[i].arrivalTime /
60;
            int arrivalMinutes = customers[i].arrivalTime % 60;
            int leavingHours = bankOpenTime + (customers[i].arrivalTime +
customers[i].serviceTime+customers[i].waitingTime ) / 60;
```

```
            int leavingMinutes = (customers[i].arrivalTime +
customers[i].serviceTime+ customers[i].waitingTime) % 60;

            outputFile << setw(5) << i + 1 << setw(13) << arrivalHours <<
":" << setw(2) << arrivalMinutes << "    "
                       << setw(15) << leavingHours << ":" << setw(2) <<
leavingMinutes << "    "
                       << setw(15) << customers[i].waitingTime << "    "
                       << setw(9) << (customers[i].served ? "Yes" : "No")
<< endl;
        }
        outputFile.close();
        cout << "Customer data written to file: customer_data.txt" <<
endl;

        ofstream outputTellerFile("teller_data.txt");
        if (outputTellerFile.is_open())
        {
            outputTellerFile << "Teller ID    Idle Time        Working
Time   Utility (%)" << endl;
            for (int i = 0; i < numTellers; i++)
            {// edit
                int idleTime = (bankCloseTime - bankOpenTime) * 60 -
tellerWorkingTime[i] + (break_end_time -break_start_time)*60 ;
                double utility = tellerWorkingTime[i] * 100.0 /
(bankCloseTime - bankOpenTime)  / 60.0 ;
                outputTellerFile << setw(5) << i + 1 << setw(13) <<
idleTime << "    "
                           << setw(15) << tellerWorkingTime[i] << "    "
                           << setw(9) << fixed << setprecision(2) <<
utility << endl;
            if (idleTime<0)
            outputTellerFile<< " This teller "<< i+1 << " has overload of
" << fabs(idleTime) << " minutes " << "to complete his assigned customers
"<< "Due to the completely randomized teller assignment "<<endl;
            }
            outputTellerFile.close();
            cout << "Teller data written to file: teller_data.txt" <<
endl;
        }
        else
        {
            cout << "Error opening file." << endl;
        }
    }
    else
    {
        cout << "Error opening file." << endl;
    }
}
int main()
{
    srand(time(0));
    int numTellers, tellerWorkingHours, bankOpenTime, bankCloseTime,
numCustomersPerday, numservedCustomersPerYear, numTransactions,
break_start_time, break_end_time;//break_time , wa

    cout << "How many transactions will you enter? ";
    cin >> numTransactions;
    Transaction transactions[numTransactions];
    get_user_input(numTellers, tellerWorkingHours, bankOpenTime,
bankCloseTime, break_start_time, break_end_time, numCustomersPerday,
numservedCustomersPerYear, numTransactions, transactions);
    generate_customer_data(numTellers, bankOpenTime, bankCloseTime,
break_start_time, break_end_time,  numCustomersPerday,
    numTransactions,  transactions ,  numservedCustomersPerYear);
    return 0;
}
```