AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

International Credit Hour Programs
(ICHEP)

Computer Programming (CSE131)

Spring 2023

Major Task Milestone 1

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.

Introduction

- This program is a simulation of a bank's customer service system.
  It takes user input for the number of tellers, working hours for each teller, bank's opening and closing time, the number of customers per day, and the number of transactions the bank offers along with the duration of each transaction type.
- Then, the program generates customer data for one day using random numbers and the customer's arrival time, service time, waiting time, and teller assigned are calculated. The customer data generated includes the time at which the customer arrives, the time when the customer leaves, the transaction type used, and the teller assigned.

```
struct Transaction {
    string type;
    int duration;
};
```

This block of code defines a Transaction struct, which contains a string to hold the transaction type, and an int to hold the transaction duration.

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

This block of code defines a customer struct, which contains an int to hold the customer's arrival time, and int to hold the service time, and integer to hold the waiting time, and integer to hold the teller assigned to the customer, and a bool to indicate whether or not the customer was served.

```
void get_user_input(int& numTellers, int& tellerWorkingHours, int&
bankOpenTime, int& bankCloseTime,
int& numCustomersPerday, int& numCustomersPerYear, int& numTransactions,
Transaction transactions[])
```

The **get_user_input()** function is a void function that takes in several parameters by reference and prompts the user to input various parameters related to the bank's customer service process. Here's a detailed illustration of how the function works.

```
void get_user_input(int& numTellers, int& tellerWorkingHours, int&
bankOpenTime, int& bankCloseTime,
int& numCustomersPerday, int& numCustomersPerYear, int& numTransactions,
Transaction transactions[])
{
    cout << "Enter the number of tellers : ";
    cin >> numTellers;
    cout << "Enter the working hours for each teller (in hours) : ";
    cin >> tellerWorkingHours;
    cout << "Enter the working hours of the bank (open time and close time
in hours past midnight) : ";
    cin >> bankOpenTime >> bankCloseTime;
    int WORKING_MINUTES_PER_DAY = (bankCloseTime - bankOpenTime) * 60;
    cout << "Enter the number of customers per day : ";
    cin >> numCustomersPerday;
    cout << "Enter the number of served customers per year (must be greater
than 30,000) : ";
    cin >> numCustomersPerYear;
    while (numCustomersPerYear <= 30000)
    {
        cout << "Number of served customers per year must be greater than
30,000. Please enter a valid number :";
        cin >> numCustomersPerYear;
    }

    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;
    }
}
```

The function first prompts the user to input the number of tellers, working hours for each teller, and the bank's opening and closing times. It then calculates the total working minutes per day based on the bank's opening and closing times.

Next, the function prompts the user to input the number of customers per day and the number of served customers per year. It validates the input for the number of served customers per year to ensure that it is greater than 30,000.

The function then prompts the user to input the types and durations of transactions. It stores this information in an array of Transaction structures.

Finally, the function outputs the transaction types and durations entered by the user.

```
void generate_customer_data(int numTellers, int bankOpenTime, int
bankCloseTime, int numCustomersPerday,
int numTransactions, Transaction transactions[])
```

The **generate_customer_data()** function takes in several parameters related to the bank's customer service process and generates customer data based on these parameters.
These parameters include the number of tellers, the bank's opening and closing times, the number of customers per day, and an array of transaction types and durations from the past function. The function also uses random number generation to simulate the randomness of customer arrival times and transaction types. Here's a detailed illustration of how the function works:

```
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};
```

The **NUM_DAYS** constant is set to 1, indicating that the function generates customer data for a single day.

The **NUM_CUSTOMERS** constant is calculated based on the number of customers per day and the number of days.

An array of Customer structs is created with size **NUM_CUSTOMERS** to store the generated customer data.

random Number is initialized to 0 and will be used to generate random arrival times for customers to the close time of bank.

waiting Time is declared but not initialized. It will be used to calculate the waiting time for each customer.
prevTellerAssigned and **prevPrevTellerAssigned** are initialized to 0, They will be used to keep track of the tellers assigned to the previous two customers to ensure that the same teller is not assigned to consecutive customers.

Finally, an array of bool values called **tellersBusy** is created with size **numTellers**. It is initialized to false for all elements, indicating that no tellers are initially busy. It will be used to keep track of which tellers are currently serving customers.

```
for (int day = 1; day <= NUM_DAYS; day++)
    {
        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;
            serviceTime = transactions[rand() % numTransactions].duration;

            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;

            int assignedTeller;
            do
               {
                assignedTeller = rand() % numTellers + 1;
               }

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

            tellersBusy[assignedTeller - 1] = true;

            customers[i].tellerAssigned = 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;


            tellersBusy[assignedTeller - 1] = false;

            prevPrevTellerAssigned = prevTellerAssigned;
            prevTellerAssigned = assignedTeller;

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

The **customer data generated** by the function includes the following information:

Arrival time: The time at which the customer arrives at the bank.

Service time: The time it takes for the customer to complete their transaction.

Waiting time: The time the customer waits before being served.

Leaving time: The time at which the customer leaves the bank.

Transaction type: The type of transaction the customer is conducting.

Teller assigned: The teller assigned to serve the customer.

The **generated customer data** can be used to analyze various aspects of the bank's customer service process, such as the average waiting time and the efficiency of teller assignment. This analysis can help identify areas for improvement in the customer service process.

⇒ loops through each day and each customer and generates customer data**.** It first generates a random arrival time for the customer and chooses a random transaction type and duration. It also calculates the customer's service time based on the transaction type.

It then calculates the customer's waiting time based on the arrival time of the previous customer and their service time. If the current customer is the first customer or there are still available tellers, their waiting time is set to 0.

The code then calculates the customer's departure time and determines whether or not the customer was served before the bank closes.

It assigns a teller to the customer by generating a random number between 1 and numTellers and checking that the teller is not currently busy.

It then outputs the customer's arrival time, departure time, transaction type, and assigned teller.

Finally, the code updates the teller assignment variables and the randomNumber variable to generate the next customer's arrival time.

Overall, the code above generates customer data based on the bank's customer service process parameters and outputs the generated data.

```
  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;
    }
    else
    {
        cout << "Error opening file." << endl;
    }
}
```

The code opens the **"customer_data.txt"** file using an **ofstream** object, **ofstream** object is a type of stream object in C++ that is used for writing output to a **file.To** use an **ofstream** object, you first need to include the **<fstream>** header file in the program. It then checks if the file was opened successfully using the **is_open()** method of the **ofstream** object.

If the file was opened successfully, the function writes the customer data to the file in a formatted table. The first line of the table contains column headers, and each subsequent line contains the customer data like that header in our code.
outputFile << "Customer ID Arrival Time Leaving Time Waiting Time Served?" <<endl;
The customer ID is simply the index of the customer in the customers array incremented by 1. The arrival time, leaving time, and waiting time are calculated using the customer data generated by the **generate_customer_data()** function. The served column is set to "Yes" if the customer was served before the bank closed, and "No" otherwise.

The function then closes the file and outputs a message to the console indicating that the customer data was written to the file successfully.

If the file was not opened successfully, the function outputs an error message to the console indicating that the file could not be opened.
To end up, the code provides a way to write the generated customer data to a file in a formatted table, making it easier to analyze and interpret the data.

```
int main()
{
    srand(time(0));
    int numTellers, tellerWorkingHours, bankOpenTime, bankCloseTime,
numCustomersPerday, numCustomersPerYear, numTransactions;
    cout << "How many transactions will you enter? ";
    cin >> numTransactions;
    Transaction transactions[numTransactions];
    get_user_input(numTellers, tellerWorkingHours, bankOpenTime,
bankCloseTime, numCustomersPerday, numCustomersPerYear, numTransactions,
transactions);
    generate_customer_data(numTellers, bankOpenTime, bankCloseTime,
numCustomersPerday, numTransactions, transactions);
    return 0;
}
```

At the end the main function:

- It begins by seeding the random number generator with the current time, ensuring that each run of the program will produce different random numbers. As the current time is always changeable.

- Next, it declares several variables for user input, including the number of tellers, the number of hours each teller will work per day, the opening and closing times of the bank, the number of customers per day, the number of customers per year, and the number of transactions found in the bank.

- Then it declares an array of transactions with the specified size which is equal to the number of transactions. Then it prompts the user to enter the number of transactions using cin.

- Then calls get_user_input function to prompt the user to enter the simulation parameters and transaction data. This function takes in the variables declared earlier and stores the user input in them.

- After the user input has been gathered, the program calls the generate_customer_data function to generate customer data for the simulation. This function takes in the number of tellers, the opening and closing times of the bank, the number of customers per day, the number of transactions, and the array of transactions as arguments. The function generates data for each customer, such as the arrival time and the type of transaction and stores this data in the array of transactions.

- Finally, it ends the main function and returns 0, indicating that the program has run successfully.

Observations:

- We use structures to define the transaction and customer data types. This allows for better organization and management of the data.

- We use a random number generator to simulate the arrival times of customers. The arrival times are generated based on the number of customers per day and the bank's opening and closing times.

- The program assigns tellers to customers randomly, but we ensure that no teller is assigned to two consecutive customers to balance the workload among tellers. This is done by keeping track of the previously assigned tellers.

- We calculate the waiting time for each customer based on their arrival time and the service time of the pr**evious customer on the same teller**. This assumes that each customer arrives after the previous customer has finished being served.

- We write the customer data to a file named "customer_data.txt". This allows for easy access and analysis of the data.

- We only generate customer data for one day. It would be useful to extend the program to simulate customer data for multiple days to obtain a better understanding of the bank's performance over time.

- Most input data is related to each other, although the input data that is not used in calculations like the number of served customers per year (must be greater than 30,000) as if the input is 36500, so the number of customers per day should be 100 customers daily.

Conclusion:

The program satisfies the functions and the objectives set before, but it goes through many iterations and changes to make sure that it reaches its objectives like:

1. The program simulates a bank's customer service operations by generating customer data based on user input.
2. The program generates customer data for a single day, with the number of customers determined by the user input for the number of customers per day.
3. The program randomizes customer's arrival time, service time, and assigned teller, so it calculates waiting time.
4. The program assigns each customer to a teller based on the teller's availability and the customer's arrival time.
5. The program prints the generated customer data to the console and writes it to a file named "customer_data.txt".

From the problems faced:

1- Make the arrival time of customers completely random but in ascending order within a good range with the working hours of the bank, so we ensure that customers are served based on the first in first out criteria.

2- Every customer is assigned to tellers randomly and making sure that no customer is assigned to a busy teller who is working with another customer at this time. The problem solved by using bool tellerBusy.

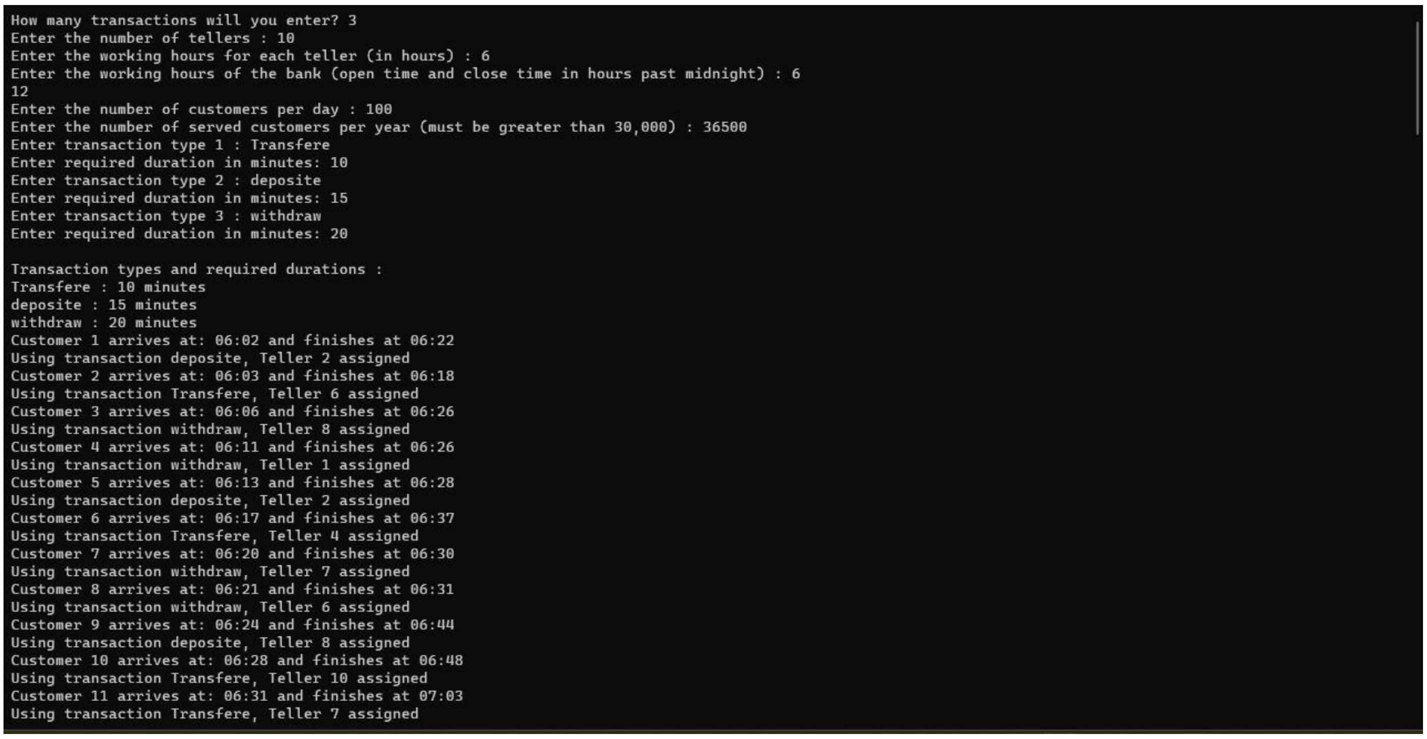
This is the full code:

```cpp
#include <iostream>
#include <iomanip>
#include <cstring>
#include <ctime>
#include <fstream>
#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& numCustomersPerday, int& numCustomersPerYear, int&
numTransactions, Transaction transactions[])
{
    cout << "Enter the number of tellers : ";
    cin >> numTellers;
    cout << "Enter the working hours for each teller (in hours) : ";
    cin >> tellerWorkingHours;
    cout << "Enter the working hours of the bank (open time and close
time in hours past midnight) : ";
    cin >> bankOpenTime >> bankCloseTime;
    int WORKING_MINUTES_PER_DAY = (bankCloseTime - bankOpenTime) * 60;
    cout << "Enter the number of customers per day : ";
    cin >> numCustomersPerday;
    cout << "Enter the number of served customers per year (must be
greater than 30,000) : ";
    cin >> numCustomersPerYear;
    while (numCustomersPerYear <= 30000)
    {
        cout << "Number of served customers per year must be greater
than 30,000. Please enter a valid number :";
        cin >> numCustomersPerYear;
    }

    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 numCustomersPerday,
int numTransactions, Transaction transactions[])
{
    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};

    for (int day = 1; day <= NUM_DAYS; day++)
    {
        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;
            serviceTime = transactions[rand() %
numTransactions].duration;

            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;
```

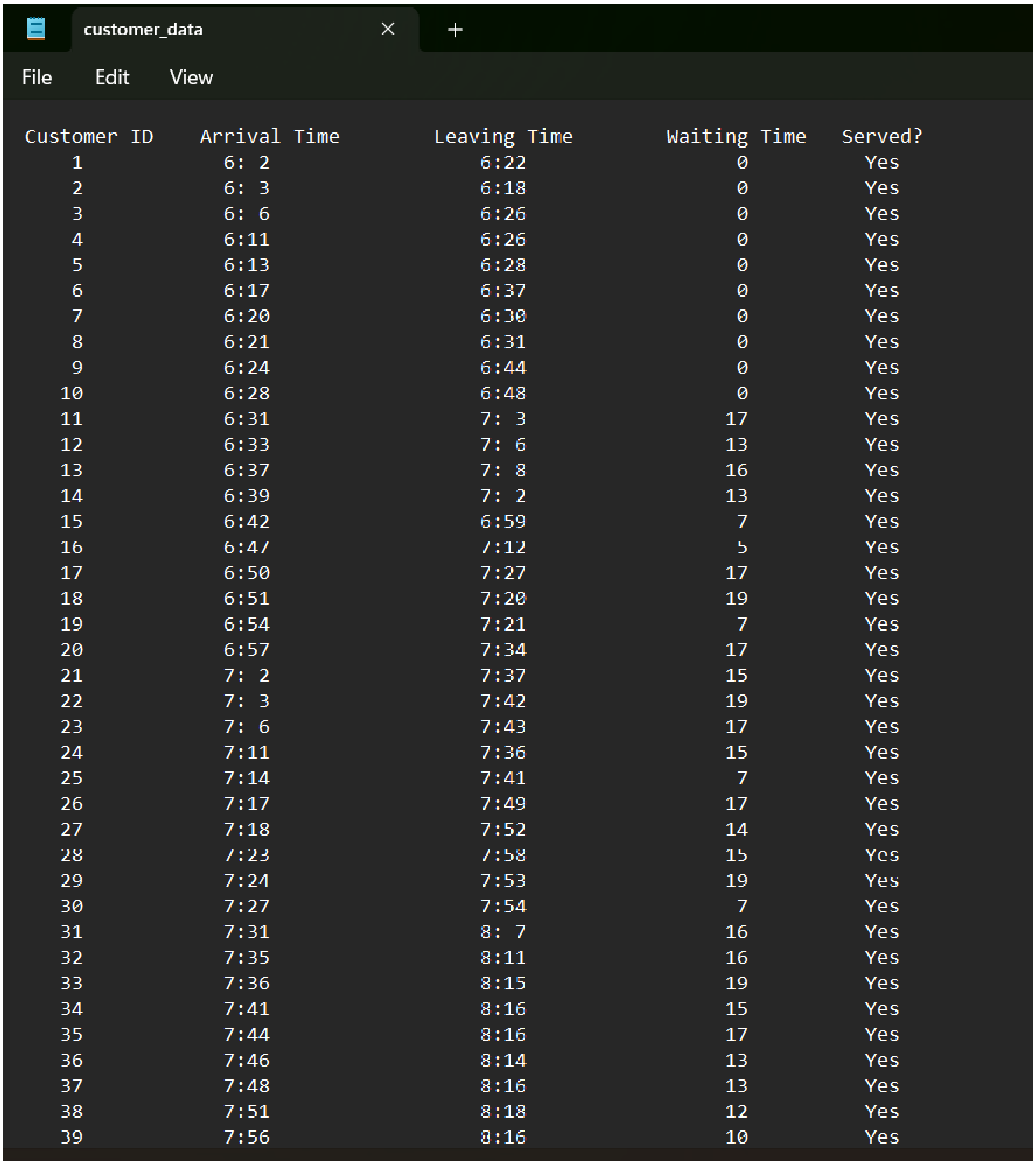
```


           if (leavingHours == bankCloseTime )
                customers[i].served = false;
            else
                customers[i].served = true;

            int assignedTeller;
            do
               {
                assignedTeller = rand() % numTellers + 1;
               }

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

            tellersBusy[assignedTeller - 1] = true;

            customers[i].tellerAssigned = 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;


            tellersBusy[assignedTeller - 1] = false;

            prevPrevTellerAssigned = prevTellerAssigned;
            prevTellerAssigned = assignedTeller;

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

    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;
    }
    else
    {
        cout << "Error opening file." << endl;
    }
}
int main()
{
    srand(time(0));
    int numTellers, tellerWorkingHours, bankOpenTime, bankCloseTime,
numCustomersPerday, numCustomersPerYear, numTransactions;
    cout << "How many transactions will you enter? ";
    cin >> numTransactions;
    Transaction transactions[numTransactions];
    get_user_input(numTellers, tellerWorkingHours, bankOpenTime,
bankCloseTime, numCustomersPerday, numCustomersPerYear,
numTransactions, transactions);
    generate_customer_data(numTellers, bankOpenTime, bankCloseTime,
numCustomersPerday, numTransactions, transactions);
    return 0;
}
```

This is the output:



```
How many transactions will you enter? 3
Enter the number of tellers : 10
Enter the working hours for each teller (in hours) : 6
Enter the working hours of the bank (open time and close time in hours past midnight) : 6
12
Enter the number of customers per day : 100
Enter the number of served customers per year (must be greater than 30,000) : 36500
Enter transaction type 1 : Transfere
Enter required duration in minutes: 10
Enter transaction type 2 : deposite
Enter required duration in minutes: 15
Enter transaction type 3 : withdraw
Enter required duration in minutes: 20

Transaction types and required durations :
Transfere : 10 minutes
deposite : 15 minutes
withdraw : 20 minutes
Customer 1 arrives at: 06:02 and finishes at 06:22
Using transaction deposite, Teller 2 assigned
Customer 2 arrives at: 06:03 and finishes at 06:18
Using transaction Transfere, Teller 6 assigned
Customer 3 arrives at: 06:06 and finishes at 06:26
Using transaction withdraw, Teller 8 assigned
Customer 4 arrives at: 06:11 and finishes at 06:26
Using transaction withdraw, Teller 1 assigned
Customer 5 arrives at: 06:13 and finishes at 06:28
Using transaction deposite, Teller 2 assigned
Customer 6 arrives at: 06:17 and finishes at 06:37
Using transaction Transfere, Teller 4 assigned
Customer 7 arrives at: 06:20 and finishes at 06:30
Using transaction withdraw, Teller 7 assigned
Customer 8 arrives at: 06:21 and finishes at 06:31
Using transaction withdraw, Teller 6 assigned
Customer 9 arrives at: 06:24 and finishes at 06:44
Using transaction deposite, Teller 8 assigned
Customer 10 arrives at: 06:28 and finishes at 06:48
Using transaction Transfere, Teller 10 assigned
Customer 11 arrives at: 06:31 and finishes at 07:03
Using transaction Transfere, Teller 7 assigned
```



```
Customer ID    Arrival Time         Leaving Time         Waiting Time    Served?
     1            6: 2                 6:22                   0            Yes
     2            6: 3                 6:18                   0            Yes
     3            6: 6                 6:26                   0            Yes
     4            6:11                 6:26                   0            Yes
     5            6:13                 6:28                   0            Yes
     6            6:17                 6:37                   0            Yes
     7            6:20                 6:30                   0            Yes
     8            6:21                 6:31                   0            Yes
     9            6:24                 6:44                   0            Yes
    10            6:28                 6:48                   0            Yes
    11            6:31                 7: 3                  17            Yes
    12            6:33                 7: 6                  13            Yes
    13            6:37                 7: 8                  16            Yes
    14            6:39                 7: 2                  13            Yes
    15            6:42                 6:59                   7            Yes
    16            6:47                 7:12                   5            Yes
    17            6:50                 7:27                  17            Yes
    18            6:51                 7:20                  19            Yes
    19            6:54                 7:21                   7            Yes
    20            6:57                 7:34                  17            Yes
    21            7: 2                 7:37                  15            Yes
    22            7: 3                 7:42                  19            Yes
    23            7: 6                 7:43                  17            Yes
    24            7:11                 7:36                  15            Yes
    25            7:14                 7:41                   7            Yes
    26            7:17                 7:49                  17            Yes
    27            7:18                 7:52                  14            Yes
    28            7:23                 7:58                  15            Yes
    29            7:24                 7:53                  19            Yes
    30            7:27                 7:54                   7            Yes
    31            7:31                 8: 7                  16            Yes
    32            7:35                 8:11                  16            Yes
    33            7:36                 8:15                  19            Yes
    34            7:41                 8:16                  15            Yes
    35            7:44                 8:16                  17            Yes
    36            7:46                 8:14                  13            Yes
    37            7:48                 8:16                  13            Yes
    38            7:51                 8:18                  12            Yes
    39            7:56                 8:16                  10            Yes
```