PRODCONS Source Code

main.cpp:

```
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>
#include <cstdlib>
#include <ctime>
#include <chrono>
#include <fstream>


using namespace std;

/// Functions Prototype
void producer(int storeID);
void consumer(int consumerID);
void inputs();

///Shared Variables

int year = 16;
int aggregateSales = 0;
int itemsGenerated= 0;
int consumerID = 0;

/// synchronization to lock and unlock thread
mutex m;



/// arrays
int storeWideTotalSales[10]={};
int monthWiseTotalSales[12]={};

/// time variables to calculate elapsed time
chrono::time_point<chrono::system_clock> startTime; // start time
chrono::time_point<chrono::system_clock> endTime; // end time

/// Struct for Buffer Item
    struct BufferItems{
    int randomDays, randomMonth, year, storeID, registerNum;
    float salesAmt;
    };

/// vectors
vector<BufferItems> buffer; //vector to store struct Buffer Items


//main function
int main(){

    ///initialize and open file(s)
    ofstream outfileGlobal;
ofstream outfileLocal;
outfileGlobal.open("global.txt", std::ios_base::app);
outfileLocal.open("local.txt", std::ios_base::app);


//semaphore threads
vector<thread> producers; // vector to store producers
vector<thread> consumers; // vector to store consumers

/// Shared variables
int storeWideTotal = 0;
int monthWiseTotal[12] = {0};

/// size of buffers
int b = 10;
int p,c= 0;
/// Inputs
cout<< "How many Producers? "<<endl;
cin>>p;
cout<< "How many Consumers? "<<endl;
cin>>c;


outfileGlobal<<"Producers = "<<p<<endl;
outfileLocal <<"Producers = "<<p<<endl;

outfileGlobal<<"Consumers = "<<c<<endl;
outfileLocal<<"Consumers = "<<c<<endl;


/// Creating Semaphore for Producers
for (int i = 1; i <= p; i++){
    producers.push_back(thread(producer, i + 1));
}

/// Creating Semaphore for Consumers
for (int i = 1; i <= c; i++){
    consumers.push_back(thread(consumer, i + 1));
}

/// Join Producers
for (int i = 0; i < p; i++){
    producers[i].join();
}

/// Join Consumers
for (int i = 0; i < c; i++){
    consumers[i].join();
}

/// Calculation for totalTime
endTime = chrono::system_clock::now();
chrono::duration<double> elapsedTime = endTime - startTime;

BufferItems record;
record.randomDays = (rand() % 30) + 1;
record.randomMonth = (rand() % 12) + 1;
```

```
    record.registerNum = (rand() % 6) + 1;
    record.salesAmt = (rand() % 999) + 50;


    /// Output
    cout << "\n***Global Statistics***\n";

    cout<<"\nStore-wide Total Sales:\n"<<endl;
    for (int i = 0; i < 10; i++)
        cout << "Store " << i+1 << " Total Sales: $" <<
storeWideTotalSales[i] << endl;

    cout<<"\nMonth-wise Total Sales:\n"<<endl;

    for (int i = 0; i < 12; i++)
        cout << (rand() % 12) + 1 <<"/" <<(rand() % 30) + 1 <<"/"<<year<< "
Total Sales: $" << monthWiseTotalSales[i]<<endl;

    cout << "Aggregate Sales: $" << aggregateSales << endl;
    cout << "Total Time for simulation: " << elapsedTime.count() << "
seconds\n";

    /// Outfile data
    outfileGlobal << "\n***Global Statistics***\n";

    outfileGlobal<<"\nStore-wide Total Sales:\n"<<endl;

    for (int i = 0; i < 10; i++)
        outfileGlobal << "Store " << i+1 << " Total Sales: $" <<
storeWideTotalSales[i] << endl;

    outfileGlobal<<"\nStore-wide Total Sales:\n"<<endl;

    for (int i = 0; i < 12; i++)
        outfileGlobal << (rand() % 12) + 1 <<"/" <<(rand() % 30) + 1
<<"/"<<year<< " Total Sales: $" << monthWiseTotalSales[i]<<endl;
        record.randomDays +=1;
    /// Output to global file
    outfileGlobal << "Aggregate Sales: $" << aggregateSales << endl;
    outfileGlobal << "Total Time for simulation: " << elapsedTime.count() <<
" seconds\n";
    outfileGlobal<<endl<<endl;

    outfileGlobal.close();
    outfileLocal.close();

    return 0;
}

//Functions

/*While the total number of items generated by all producers is less than
1000 do
{
Randomly generate DD, MM, register#, sale amount. Create a sales record and
place it in the shared buffer.
Increment the number of records count (in the shared memory)
```

```
Randomly sleep for 5-40 milliseconds }*/

/// Producer Function
void producer(int storeID) {
    BufferItems record;
    while (itemsGenerated < 1000.00) {

        /// Create random data
        record.randomDays = (rand() % 30) + 1;
        record.randomMonth = (rand() % 12) + 1;
        record.storeID = storeID;
        record.registerNum = (rand() % 6) + 1;
        record.salesAmt = (rand() % 999) + 50;

        /// Lock thread w/ mutex
        m.lock();

        /// created a sales record to place back into the shared buffer
        buffer.push_back(record);

        /// increment the number of records count in shared memory
        itemsGenerated++;

        /// Release mutex
        m.unlock();

        /// randomly sleep for 5-40 milliseconds
        int sleepTime = (rand() % 40) + 5;
        this_thread::sleep_for(chrono::milliseconds(sleepTime));
    }
}


/// Consumer Function
void consumer(int consumerID) {
    itemsGenerated=0;
    int consumerThread=0;
    ofstream outfileLocal;

    /// shared variables
    BufferItems record;
    record.randomDays = (rand() % 30) + 1;
    record.randomMonth = (rand() % 12) + 1;
    int storeWideTotal = 0;
    int monthWiseTotal[12] = {0};

    //Program ran until 1000 items produced
    while (itemsGenerated < 1000) {
        /// mutex locks
        m.lock();
        /// remove 'record' from buffer
        if (buffer.size() > 0) {
            BufferItems record = buffer[buffer.size() - 1];
            buffer.pop_back();

            /// Calculating local totals
            storeWideTotal += record.salesAmt;
            monthWiseTotal[record.randomMonth - 1] += record.salesAmt;

            /// Release mutex
            m.unlock();
        }
        else {
            /// Release mutex
            m.unlock();
            /// randomly sleep for 5-40 milliseconds
            int sleepTime = (rand() % 40) + 5;
            this_thread::sleep_for(chrono::milliseconds(sleepTime));
        }
    }

    /// Locking mutex to ensure mutual exclusion of one thread, prevents
other functions from entering the thread
    m.lock();

    /// Add local totals to global totals
    storeWideTotalSales[consumerID] = storeWideTotal;
    /// loop through 12 months and increase monthWiseTotalSales and
consumerID by 1
    for (int i = 0; i < 12; i++)
        monthWiseTotalSales[i] += monthWiseTotal[i];
        consumerID += 1;

    /// Iterate aggregateSales
    aggregateSales += storeWideTotal;

    /// Releasing mutex m allows next command in queue to enter the thread
    m.unlock();

    /// Output local statistics
    cout << "\nConsumer " << consumerID << " Statistics:\n";
    cout << "Store " << consumerID << " Total Sales: $" << storeWideTotal <<
endl;
    for (int i = 0; i < 12; i++)
        //record.randomDays = (rand() % 30) + 1;
        //record.randomMonth = (rand() % 12) + 1;
        cout << (rand() % 12) + 1 <<"/" <<(rand() % 30) + 1 <<"/"<<year<< "
Total Sales: $" << monthWiseTotal[i]<<endl;
    cout<<endl;

    /// Outfile to Outfile Local
    outfileLocal.open("local.txt", std::ios_base::app);
    outfileLocal << "\nConsumer " << consumerID << " Statistics:\n";
    outfileLocal << "Store " << consumerID << " Total Spent: $" <<
storeWideTotal << endl;
    for (int i = 0; i < 12; i++)
        outfileLocal << record.randomMonth <<"/" <<record.randomDays
<<"/"<<year<< " Total Sales: $" << monthWiseTotal[i]<<endl;
        record.randomDays +=1;
    outfileLocal<<endl;

    outfileLocal.close();
    }
```