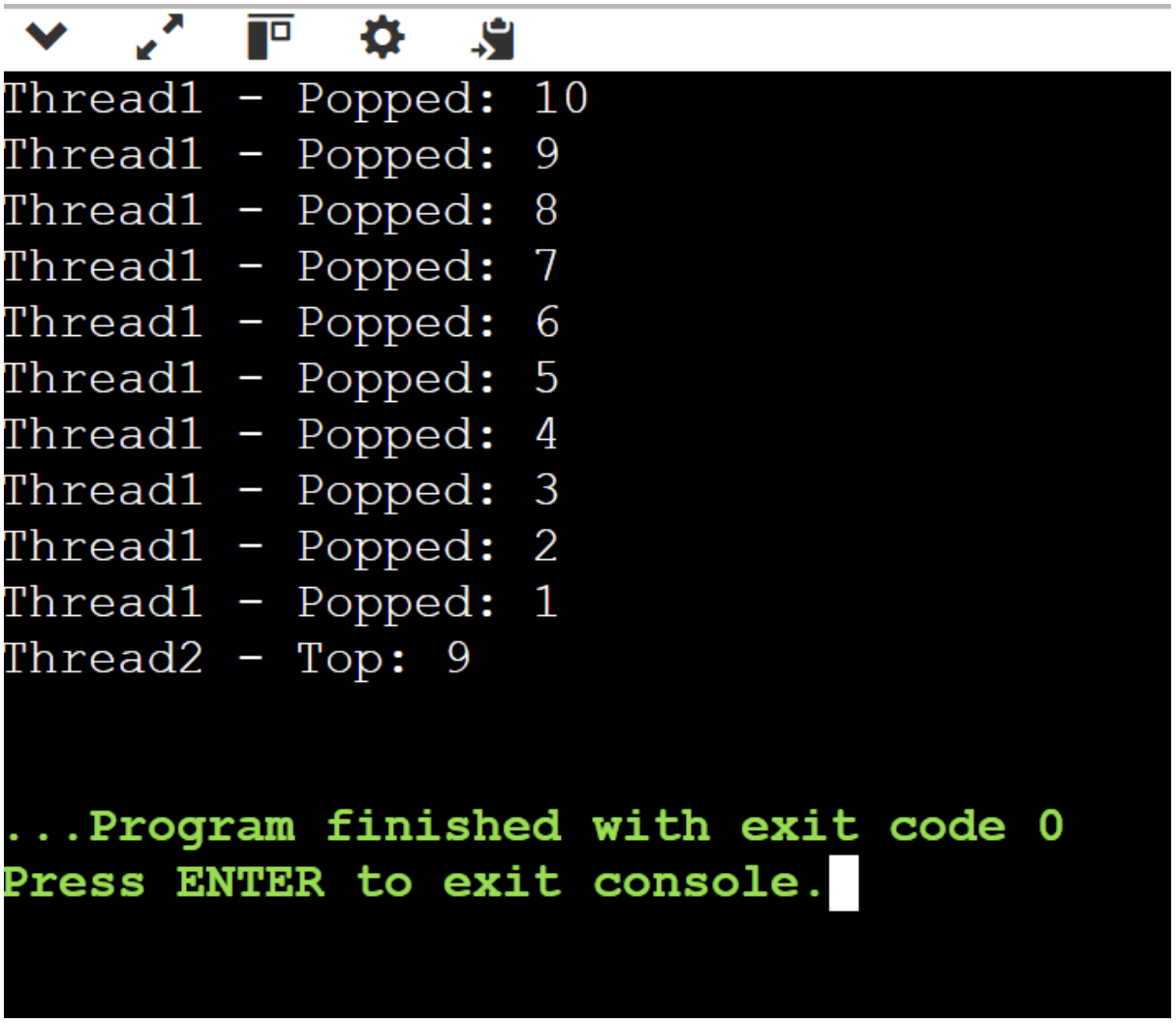
Homework November 7th

Problem 1)




```
#include <iostream>
#include <vector>
#include <mutex>
#include <thread>

using namespace std;

class Stack {
public:
    Stack(size_t limit = 10) : limit(limit) {}

    void push(int item) {
```

```
        lock_guard<mutex> lock(mtx);
        if (stack.size() < limit) {
            stack.push_back(item);
        } else {
            cout << "Stack overflow" << endl;
        }
    }

    int pop() {
        lock_guard<mutex> lock(mtx);
        if (!stack.empty()) {
            int item = stack.back();
            stack.pop_back();
            return item;
        }
        return -1;
    }

    int top() {
        lock_guard<mutex> lock(mtx);
        if (!stack.empty()) {
            return stack.back();
        }
        return -1;
    }

    bool isEmpty() {
        lock_guard<mutex> lock(mtx);
        return stack.empty();
    }

private:
    vector<int> stack;
    size_t limit;
    mutex mtx;
};

Stack stack(10);

void fillStack() {
```

```
    for (int i = 1; i <= 10; i++) {
        stack.push(i);
    }
}

void thread1() {
    while (!stack.isEmpty()) {
        int item = stack.pop();
        if (item != -1) {
            cout << "Thread1 - Popped: " << item << endl;
        }
    }
}

void thread2() {
    while (!stack.isEmpty()) {
        int item = stack.top();
        if (item != -1) {
            cout << "Thread2 - Top: " << item << endl;
        }
    }
}

int main() {
    fillStack();
    thread t1(thread1);
    thread t2(thread2);

    t1.join();
    t2.join();

    return 0;
}
```

Problem 2)


Produced: A
Consumed: A
Produced: B
Consumed: B
Produced: C
Produced: D
Consumed: C
Produced: E
Produced: F
Consumed: D
Produced: G
Produced: H
Consumed: E
Produced: I
Produced: J
Consumed: F
Produced: K
Produced: L
Consumed: G
Produced: M
Produced: N
Consumed: H
Produced: O
Produced: P
Consumed: I
Produced: Q
Produced: R
Consumed: J
Produced: S
Consumed: K
Produced: T
Consumed: L
Produced: U
Consumed: M
Produced: V


input
Consumed: N
Produced: W
Consumed: O
Produced: X
Consumed: P
Produced: Y
Consumed: Q
Produced: Z
Consumed: R
Produced: A
Consumed: S
Produced: B
Consumed: T
Produced: C
Consumed: U
Produced: D
Consumed: V
Produced: E
Consumed: W
Produced: F
Consumed: X
Produced: G
Consumed: Y
Produced: H
Consumed: Z
Produced: I
Consumed: A
Produced: J
Consumed: B
Produced: K
Consumed: C
Produced: L
Consumed: D
Produced: M
Consumed: E


input
Consumed: W
Produced: F
Consumed: X
Produced: G
Consumed: Y
Produced: H
Consumed: Z
Produced: I
Consumed: A
Produced: J
Consumed: B
Produced: K
Consumed: C
Produced: L
Consumed: D
Produced: M
Consumed: E
Produced: N
Consumed: F
Produced: O
Consumed: G
Produced: P
Consumed: H
Produced: Q
Consumed: I
Produced: R
Consumed: J
Produced: S
Consumed: K
Produced: T
Consumed: L
Produced: U
Consumed: M
Produced: V

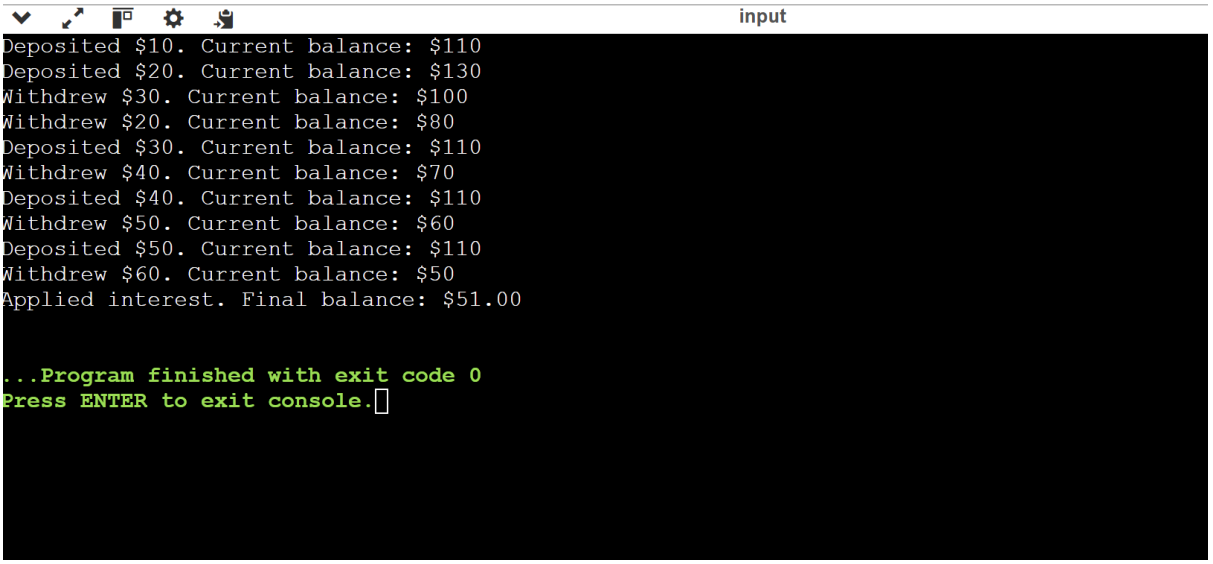

input
Consumed: B
Produced: K
Consumed: C
Produced: L
Consumed: D
Produced: M
Consumed: E
Produced: N
Consumed: F
Produced: O
Consumed: G
Produced: P
Consumed: H
Produced: Q
Consumed: I
Produced: R
Consumed: J
Produced: S
Consumed: K
Produced: T
Consumed: L
Produced: U
Consumed: M
Produced: V
Consumed: N
Consumed: O
Consumed: P
Consumed: Q
Consumed: R
Consumed: S
Consumed: T
Consumed: U
Consumed: V
Program completed successfully!

```
#include <iostream>
#include <thread>
#include <chrono>
#include <vector>
#include <mutex>
#include <condition_variable>

//this one was a tricky one to solve. the output is very long as it loops through a hundred times.

using namespace std;

const int BUFFER_SIZE = 10;
vector<char> buffer(BUFFER_SIZE, '\0');
int produceIndex = 0;
int consumeIndex = 0;
mutex mtx;
condition_variable not_full;
condition_variable not_empty;
bool isFinishedProducing = false;

void functionOne() {
    for (int i = 0; i < 100; i++) {
        unique_lock<mutex> lock(mtx);

        not_full.wait(lock, [&] { return (produceIndex + 1) % BUFFER_SIZE != consumeIndex; });

        char producedCharacter = 'A' + (i % 26);
        buffer[produceIndex] = producedCharacter;
        produceIndex = (produceIndex + 1) % BUFFER_SIZE;

        cout << "Produced: " << producedCharacter << endl;

        lock.unlock();
        not_empty.notify_one();

        this_thread::sleep_for(chrono::milliseconds(50));
    }
```

```
    {
        lock_guard<mutex> lock(mtx);
        isFinishedProducing = true;
    }
    not_empty.notify_one();
}

void functionTwo() {
    while (true) {
        unique_lock<mutex> lock(mtx);

        not_empty.wait(lock, [&] {
            return produceIndex != consumeIndex || isFinishedProducing;
        });

        if (isFinishedProducing && produceIndex == consumeIndex) {
            break;
        }

        char consumedCharacter = buffer[consumeIndex];
        buffer[consumeIndex] = '\0';
        consumeIndex = (consumeIndex + 1) % BUFFER_SIZE;

        cout << "Consumed: " << consumedCharacter << endl;

        lock.unlock();
        not_full.notify_one();

        this_thread::sleep_for(chrono::milliseconds(100));
    }
}

int main() {
    thread producer(functionOne);
    thread consumer(functionTwo);

    producer.join();
    consumer.join();

    cout << "Program completed successfully!" << endl;
```

```
    return 0;
}
```

Problem 3)




```
#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#include <iomanip>
#include <future>

//since the interest is applied daily, it gets added to the end tally of the result
using namespace std;

class Account {
public:
    Account(double initialBalance, double interestRate)
        : balance(initialBalance), interestRate(interestRate) {}

    void deposit(double amount) {
        lock_guard<std::mutex> lock(accountMutex);
        balance += amount;
        cout << "Deposited $" << amount << ". Current balance: $" << balance << endl;
    }

    void withdraw(double amount) {
        lock_guard<std::mutex> lock(accountMutex);
        if (balance >= amount) {
            balance -= amount;
            cout << "Withdrew $" << amount << ". Current balance: $" << balance <<
endl;
```

```
        } else {
            cout << "Insufficient funds for withdrawal of $" << amount << ". Current
balance: $" << balance << endl;
        }
    }

    void applyInterest() {
        balance += balance * interestRate;
        cout << "Applied interest. Final balance: $" << fixed << setprecision(2) <<
balance << endl;
    }

private:
    double balance;
    double interestRate;
    std::mutex accountMutex;
};

void executeTransactions(Account& account, const vector<double>&
depositAmounts, const vector<double>& withdrawalAmounts) {
    vector<future<void>> futures;

    for (size_t i = 0; i < depositAmounts.size(); ++i) {

        futures.push_back(async(launch::async, &Account::deposit, &account,
depositAmounts[i]));

        if (i < withdrawalAmounts.size()) {
            futures.push_back(async(launch::async, &Account::withdraw, &account,
withdrawalAmounts[i]));
        }
    }

    for (auto& future : futures) {
        future.get();
    }
}

int main() {
    Account account(100, 0.02);
```

```
    vector<double> depositAmounts = {10, 20, 30, 40, 50};
    vector<double> withdrawalAmounts = {20, 30, 40, 50, 60};

    executeTransactions(account, depositAmounts, withdrawalAmounts);

    account.applyInterest();

    return 0;
}
```