МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«**Многопоточность»

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Сычук Алексей Александрович*  «18» ноября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Глеб Андреевич*  «18» ноября 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы:** изучение принципов реализации

многопоточности и практические применение возможностей языков

высокого уровня при моделировании бизнес-процессов.

**Задание к работе:** Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Методика выполнения работы:**

1. Разработать алгоритмы решения задачи по индивидуальному заданию.

2. Написать и отладить программы решения задачи (С++, Go или Rust).

3. Протестировать работу программ на различных исходных данных.

4. По запросу преподавателя быть готовым модифицировать/добавить алгоритмы/блоки кода.

5. Ответить на теоретические вопросы к лабораторной работе на выбор преподавателя (не менее трех вопросов).**Задание 1.** Реализовать параллельный запуск заданного числа потоков, содержащих случайные символы из ASCII таблицы в формате гонки. Проанализировать особенности работы примитивов синхронизации (сравнительный анализ):

* Mutexes
* Semaphore
* SemaphoreSlim
* Barrier
* SpinLock
* SpinWait
* Monitor

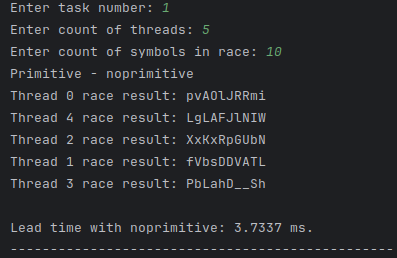
Проанализировать скорости работы примитивов с использованием StopWatch и BenchMarkDotNet.

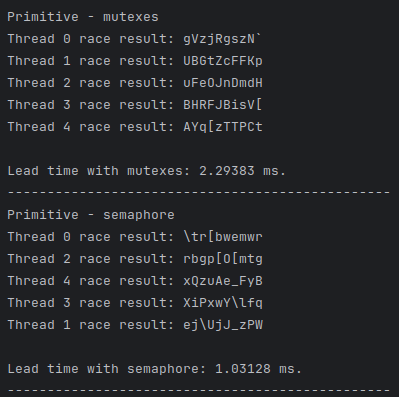
**Задание 2.** Структура содержит данные об автомобиле, находящемся в продаже на сегодняшний день (марка автомобиля, цена, пробег, тип кузова, год выпуска). Необходимо вывести список автомобилей, подходящих под критерии: диапазон цены от Ц1 до Ц2, максимальный пробег П, минимальный год выпуска Г.

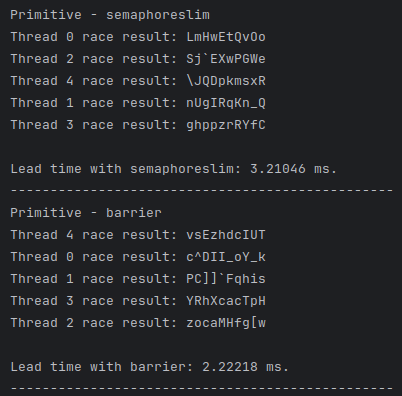
**Задание 3.** Задача обедающих философов.

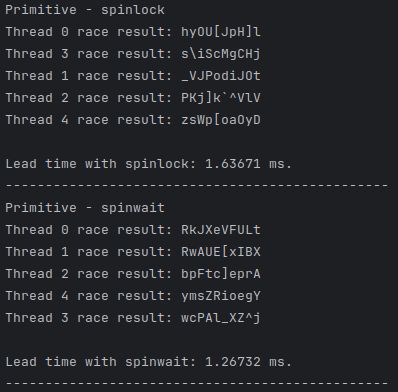
**Github:** <https://github.com/golimghostpy/Laba_4.git>

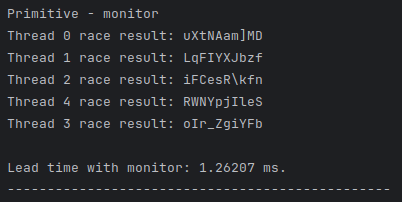
**Задание 1:**







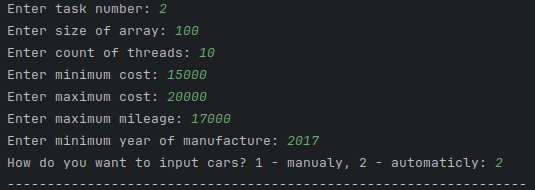


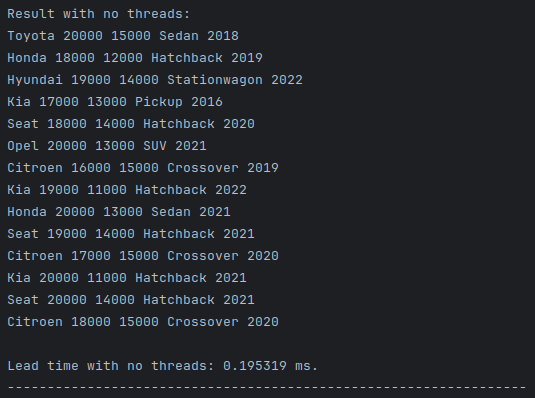


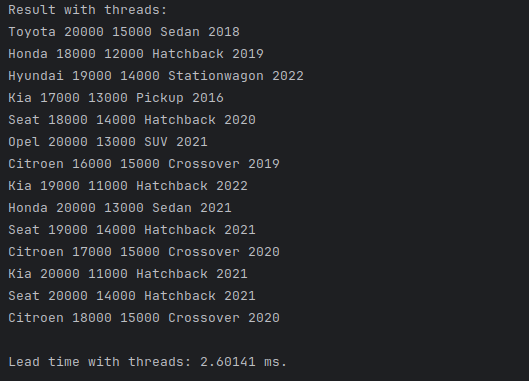
Работа Semaphor получилась быстрее SemaphoreSlim по причине того, что в С++ нет реализации SemaphoreSlim, из-за чего было необходимо реализовать самостоятельно, ввиду чего имеет место потеря производительности.

Результаты тестов показали, что даже использование самописных примитив синхронизации для многпоточной работы более выгодное, чем однопоточная работа.

**Задание 2:**







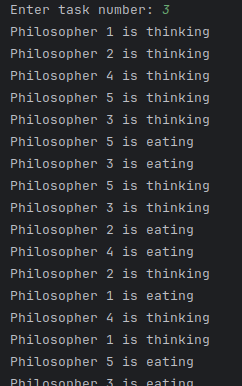
Как мы можем увидеть, полученные результаты совпадают, однако работа одного потока заняла в несколько раз меньше времени, чем работа 10 потоков.

Это происходит из-за того, что работа, предоставленная потокам, была линейной (проверка совокупности условий), то есть не требовала больших вычислительных возможностей. В то же время в С++ выделение нового потока – одна из самых затратных операций, что приводит к большой задержке. Также мы используем блокируем вывод для нескольких потоков одновременно, из-за чего выводят они также по очереди.

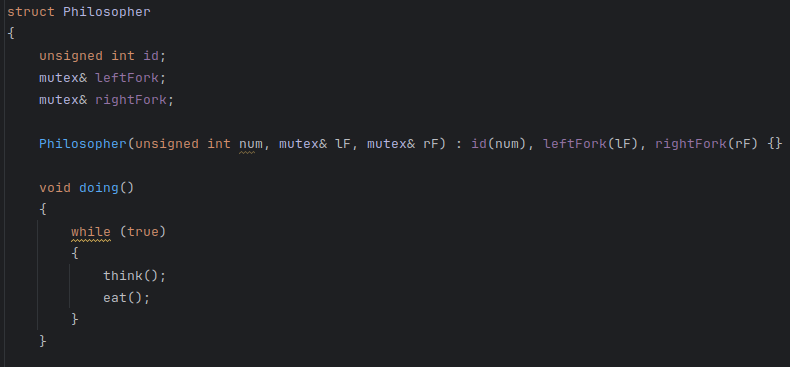
Из всего этого можно сделать вывод о том, что для оптимального использования многопоточной реализации необходимо увеличивать сложность выполняемой работы.

**Задание 3:**

Вывод задания выглядит так:



Для реализации была выбрана такая модель, что все философы одновременно сидят за столом, при этом имея одно из двух состояний: думать или кушать.



У каждого философа есть доступ как к левой, то есть своей, вилке, так и к вилке соседа справа. Тип данных для вилок – mutex, то есть если вилка будет занята кем-то другим философ не сможет получить к ней доступ.

Также есть бесконечный цикл действия, в котором философ поочередно думает и ест.

**Вывод**

В ходе данной лабораторной работы были изучены некоторые из примитивов синхронизации как языка С++, так и других языков, с адаптацией на С++. Было выявлено, что использование нескольких потоков для некоторых задач может быть неэффективно, из-за затратности ресурсов на их выделение.

**Исходный код:**

**Main.cpp**

#include <iostream>

#include "task1.h"

#include "task2.h"

#include "task3.h"

using namespace std;

int main()

{

cout << "Enter task number: ";

int choice;

cin >> choice;

switch(choice)

{

case 1: task\_1(); break;

case 2: task\_2(); break;

case 3: task\_3(); break;

default: cout << "No such a task" << endl;

}

return 0;

}

**Task1.h**

#include <iostream>

#include "task1.h"

#include "task2.h"

#include "task3.h"

using namespace std;

int main()

{

cout << "Enter task number: ";

int choice;

cin >> choice;

switch(choice)

{

case 1: task\_1(); break;

case 2: task\_2(); break;

case 3: task\_3(); break;

default: cout << "No such a task" << endl;

}

return 0;

}

**Task1.cpp**

#include "task1.h"

mutex muter; // закрывает блок кода для остальных потоков, блокируя их, пока место не освободится

counting\_semaphore cntSem(1); // может предоставить доступ нескольким потокам

barrier<>\* bar = nullptr; // собирает все потоки в указанном месте

SpinLock spinlock; // не получив доступ, поток не блокируется, а "крутится" и проверяет состояние блокировки

SpinWait spinwait; // использует более оптимизированные методы ожидания (например, yield())

Monitor monitor; // в отличие от mutex, имеет механизмы ожидания и оповещения

SemaphoreSlim\* slim = nullptr; // более облегченная версия обычного semaphore (не может работать с потоками из разных контекстов)

void race\_simulation(const unsigned int& cntSymbols, unsigned int threadId, const string& primitive)

{

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<int> distribution(65, 122);

string raceRes;

for (auto i = 0; i < cntSymbols; ++i)

{

raceRes += static\_cast<char>(distribution(gen));

}

if (primitive == "noprimitive")

{

cout << "Thread " << threadId << " race result: " << raceRes << endl;

}

else if (primitive == "mutexes")

{

lock\_guard<mutex> lock(muter);

cout << "Thread " << threadId << " race result: " << raceRes << endl;

}

else if (primitive == "semaphore")

{

cntSem.acquire();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

cntSem.release();

}

else if (primitive == "semaphoreslim")

{

slim->wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

slim->release();

}

else if (primitive == "barrier")

{

bar->arrive\_and\_wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

bar->arrive\_and\_wait();

}

else if (primitive == "spinlock")

{

spinlock.lock();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

spinlock.unlock();

}

else if (primitive == "spinwait")

{

spinwait.wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

spinwait.reset();

}

else

{

monitor.wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

monitor.notify();

}

}

void use\_primitives(const unsigned int& cntThreads, const unsigned int& cntSymbols)

{

vector<string> primitives = {"noprimitive", "mutexes", "semaphore", "semaphoreslim", "barrier", "spinlock", "spinwait", "monitor"};

for (auto i: primitives)

{

cout << "Primitive - " << i << endl;

thread allThreads[cntThreads];

auto start = chrono::high\_resolution\_clock::now();

for (unsigned int j = 0; j < cntThreads; ++j)

{

allThreads[j] = thread(race\_simulation, ref(cntSymbols), j, ref(i));

}

for (auto& i: allThreads)

{

if (i.joinable())

{

i.join();

}

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> duration = end - start;

cout << endl << "Lead time with " << i << ": " << duration.count() \* 1000 << " ms." << endl;

cout << "------------------------------------------------" << endl;

}

}

void task\_1()

{

cout << "Enter count of threads: ";

unsigned int cntThreads;

cin >> cntThreads;

cout << "Enter count of symbols in race: ";

unsigned int cntSymbols;

cin >> cntSymbols;

bar = new barrier<>(cntThreads);

slim = new SemaphoreSlim(1);

use\_primitives(cntThreads, cntSymbols);

}

**Task2.h**

#include "task1.h"

mutex muter; // закрывает блок кода для остальных потоков, блокируя их, пока место не освободится

counting\_semaphore cntSem(1); // может предоставить доступ нескольким потокам

barrier<>\* bar = nullptr; // собирает все потоки в указанном месте

SpinLock spinlock; // не получив доступ, поток не блокируется, а "крутится" и проверяет состояние блокировки

SpinWait spinwait; // использует более оптимизированные методы ожидания (например, yield())

Monitor monitor; // в отличие от mutex, имеет механизмы ожидания и оповещения

SemaphoreSlim\* slim = nullptr; // более облегченная версия обычного semaphore (не может работать с потоками из разных контекстов)

void race\_simulation(const unsigned int& cntSymbols, unsigned int threadId, const string& primitive)

{

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<int> distribution(65, 122);

string raceRes;

for (auto i = 0; i < cntSymbols; ++i)

{

raceRes += static\_cast<char>(distribution(gen));

}

if (primitive == "noprimitive")

{

cout << "Thread " << threadId << " race result: " << raceRes << endl;

}

else if (primitive == "mutexes")

{

lock\_guard<mutex> lock(muter);

cout << "Thread " << threadId << " race result: " << raceRes << endl;

}

else if (primitive == "semaphore")

{

cntSem.acquire();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

cntSem.release();

}

else if (primitive == "semaphoreslim")

{

slim->wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

slim->release();

}

else if (primitive == "barrier")

{

bar->arrive\_and\_wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

bar->arrive\_and\_wait();

}

else if (primitive == "spinlock")

{

spinlock.lock();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

spinlock.unlock();

}

else if (primitive == "spinwait")

{

spinwait.wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

spinwait.reset();

}

else

{

monitor.wait();

cout << "Thread " << threadId << " race result: " << raceRes << endl;

monitor.notify();

}

}

void use\_primitives(const unsigned int& cntThreads, const unsigned int& cntSymbols)

{

vector<string> primitives = {"noprimitive", "mutexes", "semaphore", "semaphoreslim", "barrier", "spinlock", "spinwait", "monitor"};

for (auto i: primitives)

{

cout << "Primitive - " << i << endl;

thread allThreads[cntThreads];

auto start = chrono::high\_resolution\_clock::now();

for (unsigned int j = 0; j < cntThreads; ++j)

{

allThreads[j] = thread(race\_simulation, ref(cntSymbols), j, ref(i));

}

for (auto& i: allThreads)

{

if (i.joinable())

{

i.join();

}

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> duration = end - start;

cout << endl << "Lead time with " << i << ": " << duration.count() \* 1000 << " ms." << endl;

cout << "------------------------------------------------" << endl;

}

}

void task\_1()

{

cout << "Enter count of threads: ";

unsigned int cntThreads;

cin >> cntThreads;

cout << "Enter count of symbols in race: ";

unsigned int cntSymbols;

cin >> cntSymbols;

bar = new barrier<>(cntThreads);

slim = new SemaphoreSlim(1);

use\_primitives(cntThreads, cntSymbols);

}

**Task2.cpp**

#include "task2.h"

unsigned int MAX\_THREADS;

StringList split(const string& str, const string& delimiter) { // разбиение строки в односвязный список

StringList result;

string currentPart;

int delimiterLength = delimiter.size();

for (auto i = 0; i < str.size(); ++i) {

int j = 0;

while (j < delimiterLength && i + j < str.size() && str[i + j] == delimiter[j]) {

++j;

}

if (j == delimiterLength) {

if (currentPart != "") {

result.push\_back(currentPart);

currentPart = "";

}

i += delimiterLength - 1;

} else {

currentPart += str[i];

}

}

if (!currentPart.empty()) {

result.push\_back(currentPart);

}

return result;

}

// проверка без многопотока

void no\_threads(CarArray& carShowroom, unsigned int minCost, unsigned int maxCost, unsigned int maxMileage, uint8\_t minYear){

mutex Mute; // заглушка для проверяющей функции

for (unsigned int i = 0; i < carShowroom.size; ++i){

is\_suitable(carShowroom.get\_at(i), minCost, maxCost, maxMileage, minYear, Mute);

}

}

void worker(CarArray& carShowroom, unsigned int minCost, unsigned int maxCost, unsigned int maxMileage, unsigned int minYear, unsigned int& checkedCars, mutex& Mute) {

while (true) {

if (checkedCars >= carShowroom.size) {

break;

}

is\_suitable(carShowroom.get\_at(checkedCars++), minCost, maxCost, maxMileage, minYear, Mute);

}

}

// проверка с многопотоком

void with\_threads(CarArray& carShowroom, unsigned int minCost, unsigned int maxCost, unsigned int maxMileage, uint8\_t minYear){

thread allThreads[MAX\_THREADS];

unsigned int checkedCars = 0; // общий счетчик для потоков

mutex Mute;

for (auto i = 0; i < MAX\_THREADS; ++i) {

allThreads[i] = thread(worker, ref(carShowroom), minCost, maxCost, maxMileage, minYear, ref(checkedCars), ref(Mute));

}

for (auto i = 0; i < MAX\_THREADS; ++i) {

if (allThreads[i].joinable()) {

allThreads[i].join();

}

}

}

void task\_2()

{

cout << "Enter size of array: ";

size\_t arrSize;

cin >> arrSize;

cout << "Enter count of threads: ";

cin >> MAX\_THREADS;

++MAX\_THREADS;

cout << "Enter minimum cost: ";

unsigned int minCost;

cin >> minCost;

cout << "Enter maximum cost: ";

unsigned int maxCost;

cin >> maxCost;

cout << "Enter maximum mileage: ";

unsigned int maxMileage;

cin >> maxMileage;

cout << "Enter minimum year of manufacture: ";

unsigned int minYear;

cin >> minYear;

CarArray carShowroom(arrSize);

cout << "How do you want to input cars? 1 - manualy, 2 - automaticly: ";

int inputChoice;

cin >> inputChoice;

if (inputChoice == 1){ // ручной ввод автомобилей

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Enter " << arrSize << " strings according to the template" << endl;

cout << "<marka> <cost> <mileage> <body type> <year of manufacture>" << endl;

for (auto i = 0; i < arrSize; ++i){

string temp;

getline(cin, temp);

carShowroom.push\_back(split(temp, " "));

}

}

else if (inputChoice == 2){ // выборка данных из файла

ifstream sourceFile("data.txt");

if (!sourceFile.is\_open()){

cout << "No such a file" << endl;

return;

}

for (auto i = 0; i < arrSize; ++i){

string temp;

getline(sourceFile, temp);

carShowroom.push\_back(split(temp, " "));

}

}

else {

cout << "No such a choice!" << endl;

return;

}

cout << "-----------------------------------------------------------------" << endl;

cout << "Result with no threads:" << endl;

auto startNoThreads = chrono::high\_resolution\_clock::now();

no\_threads(carShowroom, minCost, maxCost, maxMileage, minYear);

auto endNoThreads = chrono::high\_resolution\_clock::now();

chrono::duration<double> durationNoThreads = endNoThreads - startNoThreads;

cout << endl << "Lead time with no threads: " << durationNoThreads.count() \* 1000 << " ms." << endl;

cout << "-----------------------------------------------------------------" << endl;

cout << "Result with threads:" << endl;

auto startWithThreads = chrono::high\_resolution\_clock::now();

with\_threads(carShowroom, minCost, maxCost, maxMileage, minYear);

auto endWithThreads = chrono::high\_resolution\_clock::now();

chrono::duration<double> durationWithThreads = endWithThreads - startWithThreads;

// cout << endl << "Lead time with no threads: " << durationNoThreads.count() \* 1000 << " ms." << endl;

cout << endl << "Lead time with threads: " << durationWithThreads.count() \* 1000 << " ms." << endl;

}

**Task3.h**

#ifndef TASK3\_H

#define TASK3\_H

#include <iostream>

#include <mutex>

#include <thread>

#include <chrono>

#include <vector>

using namespace std;

void task\_3();

#endif //TASK3\_H

**Task3.cpp**

#ifndef TASK3\_H

#define TASK3\_H

#include <iostream>

#include <mutex>

#include <thread>

#include <chrono>

#include <vector>

using namespace std;

void task\_3();

#endif //TASK3\_H

**Structures.h**

#ifndef STRUCTURES\_H

#define STRUCTURES\_H

#include <iostream>

#include <cstdint>

#include <mutex>

using namespace std;

struct Node{

string data;

Node\* next;

Node(string val) : data(val), next(nullptr) {}

};

struct StringList{

int listSize;

Node\* first;

Node\* last;

StringList() : listSize(0), first(nullptr), last(nullptr) {}

bool is\_empty();

void push\_back(string);

Node\* find(int);

void print(string);

};

struct Car {

string marka;

unsigned int cost;

unsigned int mileage;

string bodyType;

unsigned int year;

Car(){}

Car(StringList newCar) {

marka = newCar.find(0)->data;

cost = static\_cast<unsigned int>(stoul(newCar.find(1)->data));

mileage = static\_cast<unsigned int>(stoul(newCar.find(2)->data));

bodyType = newCar.find(3)->data;

year = static\_cast<unsigned int>(stoul(newCar.find(4)->data));

}

void print();

};

void is\_suitable (Car, unsigned int, unsigned int, unsigned int, unsigned int, mutex& Mute);

struct CarArray {

Car\* data;

int size;

int capacity;

CarArray(int initialCapacity = 10) : size(0), capacity(initialCapacity){

data = new Car[capacity];

}

~CarArray() {

delete[] data;

}

void push\_back(const Car&);

Car get\_at(unsigned int);

};

#endif //STRUCTURES\_H

**Structures.cpp**

#include "structures.h"

bool StringList::is\_empty(){

return first == nullptr;

}

void StringList::push\_back(string data){

++listSize;

Node\* newElem = new Node(data);

if (is\_empty()){

first = newElem;

last = newElem;

return;

}

last->next = newElem;

last = newElem;

}

Node\* StringList::find(int index){

if (index >= listSize || index < 0) cout << index << endl;

int counter = 0;

Node\* current = first;

while (counter < index){

current = current->next;

++counter;

}

return current;

}

void StringList::print(string delimiter){

for (auto i = first; i != nullptr; i = i->next){

cout << i->data << delimiter;

}

cout << endl;

}

void is\_suitable (Car car, unsigned int from, unsigned int to, unsigned int maxMileage, unsigned int minYear, mutex& Mute){

if (car.cost >= from && car.cost <= to && car.mileage <= maxMileage && car.year >= minYear)

{

lock\_guard<mutex> locker(Mute);

cout << car.marka << " " << car.cost << " " << car.mileage << " " << car.bodyType << " " << car.year << endl;

}

}

void Car::print(){

cout << marka << " " << cost << " " << mileage << " " << bodyType << " " << year;

}

void CarArray::push\_back(const Car& element) {

if (size == capacity) {

capacity \*= 2;

Car\* newData = new Car[capacity];

for (int i = 0; i < size; i++) {

newData[i] = data[i];

}

delete[] data;

data = newData;

}

data[size++] = element;

}

Car CarArray::get\_at(unsigned int index) {

if (index < 0 || index >= size) {

throw out\_of\_range("Index out of range");

}

return data[index];

}