# TRAFFIC SIMULATION

Programozás alapjai 2.

Dokumentáció

Király Bálint
EQF1M0

2023. május 28.

---

**TARTALOM**

# 1. Feladat

## Autópálya forgalma

Készítsen objektummodellt az autópálya forgalmának modellezésére! Egy L cellára osztott autópályán N autó van. Egy cellában csak egy autó tartózkodhat egyszerre, így L-N cella üres. Minden autónak van egy egész értékű sebessége. A szimulációt ciklusonként végezzük. Minden ciklusban minden autóra elvégezzük a következő műveleteket:

1. Ha egy autó sebessége még nem érte el a maximumot (5), akkor a sebességét eggyel megnöveljük.
2. Ha egy autó előtt levő üres cellák száma (az előtte levő autóig) kisebb, mint a sebessége, akkor az autó sebességet lecsökkentjük az előtte levő üres cellák számának megfelelő értékre.
3. Egy adott p(=0.15) valószínűséggel csökkentjük a mozgó autók sebességet eggyel. (Vezetők figyelmetlensége).
4. Minden autót előremozgatunk annyi cellával, amennyi a sebessége.

Egyszerű karakteres kimenetet feltételezve "rajzolja ki" az autópálya állapotát egy-egy szimulációs ciklus után.
Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz **ne** használjon STL tárolót!

# 2. Feladatspecifikáció

Classes: Simulation
Highway
Vehicle
Car

A Szimuláció elején generálódik egy Simulation object, ami még nem tartalmaz autópályát, a ciklus számlálója 0.

A konfiguráló függvényét meghívva egy config fileból beolvassa az autópálya hosszát, az járnűvek típusát helyzetét, illetve sebességét, és dinamikusan eltárolja létrehozott Highway, és Vehicle objectekben.
Az újonnan létrehozott Highway object pointerét hozzáfűzi az autópálya pointereket tároló dinamikus tömbjéhez.

Igény szerint további config fileokból is olvashatunk be adatokat. Minden file egy autópálya kiindulási állapotát tartalmazza.

A szimuláció simulate() tagfüggvényét hívva a ciklus léptethető. Minden autónak kiszámolja a sebességét, és az új helyzetét. A ciklus számláló eggyel nő.

A megjelenítésért a printState() tagfüggvény a felelős, mely a szimuláció jelen állását írja ki kimenetre karakteres megjelenítéssel.

A felhasználói felület egészen egyszerű: lehetőség van új autópálya hozzáadására, ciklus léptetésre, illetve a szimuláció befejezésére.

A projectben később megvalósításra került heterogén kollekció, így tetszőlegesen kiegészíthető bármilyen Járművel, ami a Vehicle osztályból származtatható. Ennek fényében a leírt algoritmusoknál Car osztály helyett általános Vehicle osztály alkalmazandó.

# 3. Terv

### 3.1.1. Konfiguráció

A szimulációhoz hozzáadhatunk uj autópályát külső file-ból. Ehhez a Simulation addHighWay tagfüggvényét kell a config file nevével meghívni.

void addHighWay(const char* filename)

*Config file formátuma:*

[L] [N]
[CAR1.TYPE] [CAR1.POSITION] [CAR1.SPEED]
[CAR2.TYPE] [CAR2.POSITION] [CAR2.SPEED]
...
...
[CARN.TYPE] [CARN.POSITION] [CARN.SPEED]

*A konfiguráció algoritmusa:*

OPEN file

Highway FOGLALÁS, ahol length = L, carCount = N

highwayCount ++

cars tömb FOGLALÁS

MINDEN Car-ra:

position = car[i].position, speed = car[i].speed

CIKLUS VÉGE

CLOSE file

A vehicles tömb egy length méretű dinamikus tömb, benne Vehicle pointerek jelzik az adott helyen lévő járművet, a többi cella null pointerrel van feltöltve.

### 3.1.2. Ciklus léptetés

A cycle számlálót a Simulation object tartalmazza, így ennek a tagfüggvényét hívjuk meg minden ciklusléptetés alkalmával. Ez meghívja a többi osztály saját léptető függvényét is, és visszaadja a ciklusszámot.

int Simulation::simulate()

MINDEN Highway-re:

MINDEN Car-ra:

Accelerate();

Decelerate(emptyCells)

randomDeceleration(p)

moveVehicle()

CIKLUS VÉGE

print

CIKLUS VÉGE

RETURN cycleCount

### 3.1.3. Emberi tényező

P valószínűséggel lassít.

void randomDeceleration(double p)

r = RANDOM szám 0-99 közt

HA r < p*100: speed--

### 3.1.4. Megjelenítés

Minden ciklusban a kimeneti streamre kirajzolja a szimuláció aktuális állapotát.

void Highway::printState(std::ostream& os)

Megnézi hogy az adott helyen milyen pointer van, meghívja annak a kirajzoló függvényét. Ha null pointerrel találkozik, oda ’-’-t ír.

### 3.1.5. Main

Az egyes funkciókat bizonyos karakterekkel lehet vezérelni. Az autópálya hozzáadása [2], a ciklus léptetés [1], a program bezárása [3] lehetőségek érhetőek el. A program magját a következő algoritmus alkotja:

Simulation

PRINT vezérlés[2,3]

AMÍG nincs EOF, és olvasunk az input stream-ről:

HA [1]: Simulation::simulate()

HA [2]: config()

HA [3]: EXIT

PRINT vezérlés

CIKLUS VÉGE

---

# 4. Dokumentáció

# Traffic Simulation

Generated by Doxygen 1.9.7

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

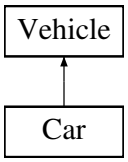
Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 Car Class Reference

Inheritance diagram for Car:




#### Public Member Functions

- **Car** (int position, int speed)
- void display (std::ostream &os)

### Public Member Functions inherited from Vehicle

- **Vehicle** (int position, int speed, int maxSpeed)
- int getspeed () const
- int getpos () const
- void setpos (int pos)
- void accelerate ()
- void decelerate (int emptyCells)
- void randomDeceleration (double p)
- virtual void display (std::ostream &os)=0

#### Additional Inherited Members

### Protected Attributes inherited from Vehicle

- int **position**
- int **speed**
- int **maxSpeed**

### 4.1.1 Member Function Documentation

#### 4.1.1.1 display()

```
void Car::display (
            std::ostream & os )  [virtual]
```

displays the car as symbol

**Parameters**

| | |
|---|---|
| *os* | |

Implements Vehicle.

The documentation for this class was generated from the following files:

- Car.h
- Car.cpp

## 4.2 Highway Class Reference

**Public Member Functions**

- Highway (size_t length)
- void addVehicle (char type, int position, int speed)
- void moveVehicle (Vehicle ∗vehicle, int numOfCells)
- void simulate ()
- void printState (std::ostream &os)
- ∼Highway ()

### 4.2.1 Constructor & Destructor Documentation

#### 4.2.1.1 Highway()

```
Highway::Highway (
            size_t length )  [inline], [explicit]
```

Highway constructor

**Parameters**

| | |
|---|---|
| *length* | |

#### 4.2.1.2 ∼Highway()

```
Highway::∼Highway ( )  [inline]
```

Highway destructor

## 4.2.2 Member Function Documentation

### 4.2.2.1 addVehicle()

```
void Highway::addVehicle (
            char type,
            int position,
            int speed )
```

Adds new vehicle to the array using dynamic memory allocation

**Parameters**

| | |
|---|---|
| *type* | |
| *position* | |
| *speed* | |

### 4.2.2.2 moveVehicle()

```
void Highway::moveVehicle (
            Vehicle * vehicle,
            int numOfCells )
```

Moves vehicle numOfCells cells forward

**Parameters**

| | |
|---|---|
| *vehicle* | |
| *numOfCells* | |

### 4.2.2.3 printState()

```
void Highway::printState (
            std::ostream & os )
```

Prints the current state

**Parameters**

| | |
|---|---|
| *os* | |

### 4.2.2.4 simulate()

```
void Highway::simulate ( )
```

Transforms the state of the Highway into the next cycle

The documentation for this class was generated from the following files:

- Highway.h
- Highway.cpp

## 4.3 Simulation Class Reference

### Public Member Functions

- Simulation ()
- void addHighWay (const char *filename)
- int simulate ()
- void printState (std::ostream &os)
- ~Simulation ()

### 4.3.1 Constructor & Destructor Documentation

#### 4.3.1.1 Simulation()

```
Simulation::Simulation ( )  [inline]
```

Simulation constructor

#### 4.3.1.2 ~Simulation()

```
Simulation::~Simulation ( )  [inline]
```

Simulation destructor

### 4.3.2 Member Function Documentation

#### 4.3.2.1 addHighWay()

```
void Simulation::addHighWay (
            const char * filename )
```

Adds new Highway to Simulation from config file

**Parameters**

| | |
|---|---|
| *filename* | |

#### 4.3.2.2 printState()

```
void Simulation::printState (
            std::ostream & os )
```

Calls every Highway's print function

#### 4.3.2.3 simulate()

```
int Simulation::simulate ( )
```

Calls the simulate function of all the Highways it contains

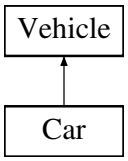
**Returns**

The documentation for this class was generated from the following files:

- Simulation.h
- Simulation.cpp

## 4.4 Vehicle Class Reference

Inheritance diagram for Vehicle:




### Public Member Functions

- **Vehicle** (int position, int speed, int maxSpeed)
- int getspeed () const
- int getpos () const
- void setpos (int pos)
- void accelerate ()
- void decelerate (int emptyCells)
- void randomDeceleration (double p)
- virtual void display (std::ostream &os)=0

### Protected Attributes

- int **position**
- int **speed**
- int **maxSpeed**

### 4.4.1 Member Function Documentation

#### 4.4.1.1 accelerate()

```
void Vehicle::accelerate ( )
```

increases speed by one if it is below the maximum

#### 4.4.1.2 decelerate()

```
void Vehicle::decelerate (
            int emptyCells )
```

decreases speed if it is greater than the emptyCells

**Parameters**

| | |
|---|---|
| *emptyCells* | |

#### 4.4.1.3 display()

```
virtual void Vehicle::display (
            std::ostream & os )  [pure virtual]
```

Implemented in Car.

#### 4.4.1.4 getpos()

```
int Vehicle::getpos ( ) const
```

Gets the position of the vehicle

**Returns**

position

#### 4.4.1.5 getspeed()

```
int Vehicle::getspeed ( ) const
```

Gets the velocity of the vehicle

**Returns**

speed

#### 4.4.1.6 randomDeceleration()

```
void Vehicle::randomDeceleration (
            double p )
```

At a p probability decreases the speed by one

**Parameters**

| | |
|---|---|
| *p* | |

#### 4.4.1.7 setpos()

```
void Vehicle::setpos (
            int pos )
```

Sets the position of the vehicle to pos

**Parameters**

| | |
|---|---|
| *pos* | |

The documentation for this class was generated from the following files:

- Vehicle.h
- Vehicle.cpp

# Chapter 5

# File Documentation

## 5.1 Car.h

```
00001 //
00002 // Created by Balint Kiraly on 2023. 05. 14..
00003 //
00004
00005 #ifndef TRAFFIC_SIMULATION_CAR_H
00006 #define TRAFFIC_SIMULATION_CAR_H
00007
00008 #include "memtrace.h"
00009 #include "Car.h"
00010 #include "Vehicle.h"
00011
00012 class Car : public Vehicle {
00013 public:
00014     Car(int position, int speed) : Vehicle(position, speed, 5) {}
00015
00020     void display(std::ostream& os);
00021     ~Car() {}
00022 };
00023
00024 #endif //TRAFFIC_SIMULATION_CAR_H
```

## 5.2 Highway.h

```
00001 //
00002 // Created by Balint Kiraly on 2023. 05. 14..
00003 //
00004
00005 #ifndef TRAFFIC_SIMULATION_HIGHWAY_H
00006 #define TRAFFIC_SIMULATION_HIGHWAY_H
00007
00008 #include <iostream>
00009 #include <cstdio>
00010
00011 #include "memtrace.h"
00012 #include "Highway.h"
00013 #include "Car.h"
00014
00015 class Highway {
00016     size_t length;
00017     size_t vehicleCount;
00018     Vehicle** vehicles;
00019 public:
00024     explicit Highway(size_t length) : length(length), vehicleCount(0) {
00025         vehicles = new Vehicle*[length];
00026         for (size_t i = 0; i < length; i++) {
00027             vehicles[i] = nullptr;
00028         }
00029     }
00030
00037     void addVehicle(char type, int position, int speed);
00038
00044     void moveVehicle(Vehicle* vehicle, int numOfCells);
00045
00049     void simulate();
```

```
00050
00055     void printState(std::ostream& os);
00056
00060     ~Highway() {
00061         for (size_t i = 0; i < length; ++i) {
00062             delete vehicles[i];
00063         }
00064         delete[] vehicles;
00065     }
00066 };
00067
00068
00069 #endif //TRAFFIC_SIMULATION_HIGHWAY_H
```

## 5.3 memtrace.h

```
00001 /*********************************
00002 Memoriaszivargas-detektor
00003 Keszitette: Peregi Tamas, BME IIT, 2011
00004             petamas@iit.bme.hu
00005 Kanari:     Szeberenyi Imre, 2013.,
00006 VS 2012:    Szeberényi Imre, 2015.,
00007 mem_dump:   2016.
00008 inclue-ok:  2017., 2018., 2019., 2021.
00009 *********************************/
00010
00011 #ifndef MEMTRACE_H
00012 #define MEMTRACE_H
00013
00014 #if defined(MEMTRACE)
00015
00016 /*ha definiálva van, akkor a hibakat ebbe a fajlba írja, egyébkent stderr-re*/
00017 /*#define MEMTRACE_ERRFILE MEMTRACE.ERR*/
00018
00019 /*ha definialva van, akkor futas kozben lancolt listat epit. Javasolt a hasznalata*/
00020 #define MEMTRACE_TO_MEMORY
00021
00022 /*ha definialva van, akkor futas kozben fajlba irja a foglalasokat*/
00023 /*ekkor nincs ellenorzes, csak naplozas*/
00024 /*#define MEMTRACE_TO_FILE*/
00025
00026 /*ha definialva van, akkor a megallaskor automatikus riport keszul */
00027 #define MEMTRACE_AUTO
00028
00029 /*ha definialva van, akkor malloc()/calloc()/realloc()/free() kovetve lesz*/
00030 #define MEMTRACE_C
00031
00032 #ifdef MEMTRACE_C
00033     /*ha definialva van, akkor free(NULL) nem okoz hibat*/
00034     #define ALLOW_FREE_NULL
00035 #endif
00036
00037 #ifdef __cplusplus
00038     /*ha definialva van, akkor new/delete/new[]/delete[] kovetve lesz*/
00039     #define MEMTRACE_CPP
00040 #endif
00041
00042 #if defined(__cplusplus) && defined(MEMTRACE_TO_MEMORY)
00043     /*ha definialva van, akkor atexit helyett objektumot hasznal*/
00044     /*ajanlott bekapcsolni*/
00045     #define USE_ATEXIT_OBJECT
00046 #endif
00047
00048 /*****************************************/
00049 /* INNEN NE MODOSITSD                    */
00050 /*****************************************/
00051 #ifdef NO_MEMTRACE_TO_FILE
00052     #undef MEMTRACE_TO_FILE
00053 #endif
00054
00055 #ifdef NO_MEMTRACE_TO_MEMORY
00056     #undef MEMTRACE_TO_MEMORY
00057 #endif
00058
00059 #ifndef MEMTRACE_AUTO
00060     #undef USE_ATEXIT_OBJECT
00061 #endif
00062
00063 #ifdef __cplusplus
00064     #define START_NAMESPACE namespace memtrace {
00065     #define END_NAMESPACE } /*namespace*/
00066     #define TRACEC(func) memtrace::func
00067     #include <new>
```

```
00068 #else
00069     #define START_NAMESPACE
00070     #define END_NAMESPACE
00071     #define TRACEC(func) func
00072 #endif
00073
00074 // THROW deklaráció változatai
00075 #if defined(_MSC_VER)
00076   // VS rosszul kezeli az __cplusplus makrot
00077   #if _MSC_VER < 1900
00078     // * nem biztos, hogy jó így *
00079     #define THROW_BADALLOC
00080     #define THROW_NOTHING
00081   #else
00082     // C++11 vagy újabb
00083     #define THROW_BADALLOC noexcept(false)
00084     #define THROW_NOTHING noexcept
00085   #endif
00086 #else
00087   #if __cplusplus < 201103L
00088     // C++2003 vagy régebbi
00089     #define THROW_BADALLOC throw (std::bad_alloc)
00090     #define THROW_NOTHING throw ()
00091   #else
00092     // C++11 vagy újabb
00093     #define THROW_BADALLOC noexcept(false)
00094     #define THROW_NOTHING noexcept
00095   #endif
00096 #endif
00097
00098 START_NAMESPACE
00099     int allocated_blocks();
00100 END_NAMESPACE
00101
00102 #if defined(MEMTRACE_TO_MEMORY)
00103 START_NAMESPACE
00104         int mem_check(void);
00105 END_NAMESPACE
00106 #endif
00107
00108 #if defined(MEMTRACE_TO_MEMORY) && defined(USE_ATEXIT_OBJECT)
00109 #include <cstdio>
00110 START_NAMESPACE
00111     class atexit_class {
00112         private:
00113             static int counter;
00114             static int err;
00115         public:
00116             atexit_class() {
00117 #if defined(CPORTA) && !defined(CPORTA_NOSETBUF)
00118                 if (counter == 0) {
00119                         setbuf(stdout, 0);
00120                         setbuf(stderr, 0);
00121                 }
00122 #endif
00123             counter++;
00124             }
00125
00126             int check() {
00127                 if(--counter == 0)
00128                     err = mem_check();
00129                 return err;
00130             }
00131
00132             ~atexit_class() {
00133                 check();
00134             }
00135     };
00136
00137 static atexit_class atexit_obj;
00138
00139 END_NAMESPACE
00140 #endif/*MEMTRACE_TO_MEMORY && USE_ATEXIT_OBJECT*/
00141
00142 /*Innentol csak a "normal" include eseten kell, kulonben osszezavarja a mukodest*/
00143 #ifndef FROM_MEMTRACE_CPP
00144 #include <stdlib.h>
00145 #ifdef __cplusplus
00146     #include <iostream>
00147 /* ide gyűjtjük a nemtrace-vel összeakadó headereket, hogy előbb legyenek */
00148
00149     #include <fstream>  // VS 2013 headerjében van deleted definició
00150     #include <sstream>
00151     #include <vector>
00152     #include <list>
00153     #include <map>
00154     #include <algorithm>
```

```
00155     #include <functional>
00156     #include <memory>
00157     #include <iomanip>
00158     #include <locale>
00159     #include <typeinfo>
00160     #include <ostream>
00161     #include <stdexcept>
00162     #include <ctime>
00163     #if __cplusplus >= 201103L
00164         #include <iterator>
00165         #include <regex>
00166     #endif
00167 #endif
00168 #ifdef MEMTRACE_CPP
00169     namespace std {
00170         typedef void (*new_handler)();
00171 }
00172 #endif
00173
00174 #ifdef MEMTRACE_C
00175 START_NAMESPACE
00176     #undef malloc
00177     #define malloc(size) TRACEC(traced_malloc)(size,#size,__LINE__,__FILE__)
00178     void * traced_malloc(size_t size, const char *size_txt, int line, const char * file);
00179
00180     #undef calloc
00181     #define calloc(count,size) TRACEC(traced_calloc)(count, size, #count","#size,__LINE__,__FILE__)
00182     void * traced_calloc(size_t count, size_t size, const char *size_txt, int line, const char *
      file);
00183
00184     #undef free
00185     #define free(p) TRACEC(traced_free)(p, #p,__LINE__,__FILE__)
00186     void traced_free(void * p, const char *size_txt, int line, const char * file);
00187
00188     #undef realloc
00189     #define realloc(old,size) TRACEC(traced_realloc)(old,size,#size,__LINE__,__FILE__)
00190     void * traced_realloc(void * old, size_t size, const char *size_txt, int line, const char * file);
00191
00192     void mem_dump(void const *mem, size_t size, FILE* fp = stdout);
00193
00194
00195 END_NAMESPACE
00196 #endif/*MEMTRACE_C*/
00197
00198 #ifdef MEMTRACE_CPP
00199 START_NAMESPACE
00200     #undef set_new_handler
00201     #define set_new_handler(f) TRACEC(_set_new_handler)(f)
00202     void _set_new_handler(std::new_handler h);
00203
00204     void set_delete_call(int line, const char * file);
00205 END_NAMESPACE
00206
00207 void * operator new(size_t size, int line, const char * file) THROW_BADALLOC;
00208 void * operator new[](size_t size, int line, const char * file) THROW_BADALLOC;
00209 void * operator new(size_t size) THROW_BADALLOC;
00210 void * operator new[](size_t size) THROW_BADALLOC;
00211 void operator delete(void * p)  THROW_NOTHING;
00212 void operator delete[](void * p) THROW_NOTHING;
00213
00214 #if __cplusplus >= 201402L
00215 // sized delete miatt: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3536.html
00216 void operator delete(void * p, size_t)  THROW_NOTHING;
00217 void operator delete[](void * p, size_t) THROW_NOTHING;
00218 #endif
00219
00220 /* Visual C++ 2012 miatt kell, mert háklis, hogy nincs megfelelő delete, bár senki sem használja */
00221 void operator delete(void *p, int, const char *) THROW_NOTHING;
00222 void operator delete[](void *p, int, const char *) THROW_NOTHING;
00223
00224
00225 #define new new(__LINE__, __FILE__)
00226 #define delete memtrace::set_delete_call(__LINE__, __FILE__),delete
00227
00228 #ifdef CPORTA
00229 #define system(...)  // system(__VA_ARGS__)
00230 #endif
00231
00232 #endif /*MEMTRACE_CPP*/
00233
00234 #endif /*FROM_MEMTRACE_CPP*/
00235 #else
00236 #pragma message ( "MEMTRACE NOT DEFINED" )
00237 #endif /*MEMTRACE*/
00238
00239 #endif /*MEMTRACE_H*/
```

## 5.4 Simulation.h

```
00001 //
00002 // Created by Balint Kiraly on 2023. 05. 14..
00003 //
00004
00005 #ifndef TRAFFIC_SIMULATION_SIMULATION_H
00006 #define TRAFFIC_SIMULATION_SIMULATION_H
00007
00008 #include <cstdio>
00009
00010 #include "memtrace.h"
00011 #include "Simulation.h"
00012 #include "Highway.h"
00013
00014 class Simulation {
00015     size_t cycleCount;
00016     size_t highwayCount;
00017     Highway** highways;
00018 public:
00022     Simulation():cycleCount(0), highwayCount(0), highways(nullptr) {}
00023
00028     void addHighWay(const char* filename);
00029
00034     int simulate();
00035
00039     void printState(std::ostream& os);
00040
00044     ~Simulation() {
00045         for (size_t i = 0; i < highwayCount; ++i) {
00046             delete highways[i];
00047         }
00048         delete[] highways;
00049     }
00050 };
00051
00052
00053 #endif //TRAFFIC_SIMULATION_SIMULATION_H
```

## 5.5 Vehicle.h

```
00001 //
00002 // Created by Balint Kiraly on 2023. 05. 27..
00003 //
00004
00005 #ifndef TRAFFIC_SIMULATION_VEHICLE_H
00006 #define TRAFFIC_SIMULATION_VEHICLE_H
00007
00008 #include <iostream>
00009
00010 #include "memtrace.h"
00011
00012
00013 class Vehicle {
00014 protected:
00015     int position;
00016     int speed;
00017     int maxSpeed;
00018 public:
00019     Vehicle(int position, int speed, int maxSpeed) : position(position), speed(speed),
      maxSpeed(maxSpeed) {}
00020
00025     int getspeed() const;
00026
00031     int getpos() const;
00032
00037     void setpos(int pos);
00038
00042     void accelerate();
00043
00048     void decelerate(int emptyCells);
00049
00054     void randomDeceleration(double p);
00055
00056     virtual void display(std::ostream& os) = 0;
00057
00058     virtual ~Vehicle() {}
00059 };
00060
00061
00062 #endif //TRAFFIC_SIMULATION_VEHICLE_H
```

# Index