Autópálya

Generated by Doxygen 1.8.20

Chapter 1

Autópálya forgalma dokumentáció

1.1 A 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!

A fenti szöveg a tárgy honlapjáról származik, ez a feladatkiírás, melynek pontjaira a dokumentáció során sokszor hivatkozom.

1.2 A feladatspecifikáció

A program induláskor két számot kér a felhasználótól: Az autópálya hosszát cellákban (L), majd az autók számát (N). Amennyiben **N >= L, N < 2, vagy L < 3**, a program figyelmezteti a felhasználót és új adatokat kér, amíg megfelelőt nem kap. Az autók eleinte véletlenszerűen vannak elszórva az L cellában, valamint sebességük kezdetben 0. Az autópályát úgy tekintjük, mintha egy kör lenne, vagyis amikor egy autó kilép az utolsó cellából, akkor az első cellába kerül, mintha az autópálya végtelen lenne. Ezután a program elkezdi a szimulációt: minden ciklusban, miután a fenti műveleteket elvégezte, kirajzolja az autópályát. Az autópálya teljes hosszában ír egy sort csupa „-” karakterből, a következő sorban jön az autópálya: üres cella esetén szóköz, autó esetén pedig egy jellegzetes karakter. Ezután egy újabb elválasztó sor zárja egy ciklus „kirajzolását”. A program egymás után öt ilyen ciklust rajzol ki, majd egy számot vár inputként:

• 1: Folytatás. Újabb öt ciklus fut le, ugyanazokkal a paraméterekkel.

• 2: Autópálya hosszának változtatása. Ezután a program ismét inputot vár, az autópálya új hosszát. Erre is vonatkoznak a korábbi feltételek. Miután megfelelő input érkezett, visszalépünk az előző választáshoz.

• 3: Autók számának változtatása. Ezután a program ismét inputot vár, az autók új számát. Erre is vonatkoznak a korábbi feltételek. Miután megfelelő input érkezett, visszalépünk az előző választáshoz.

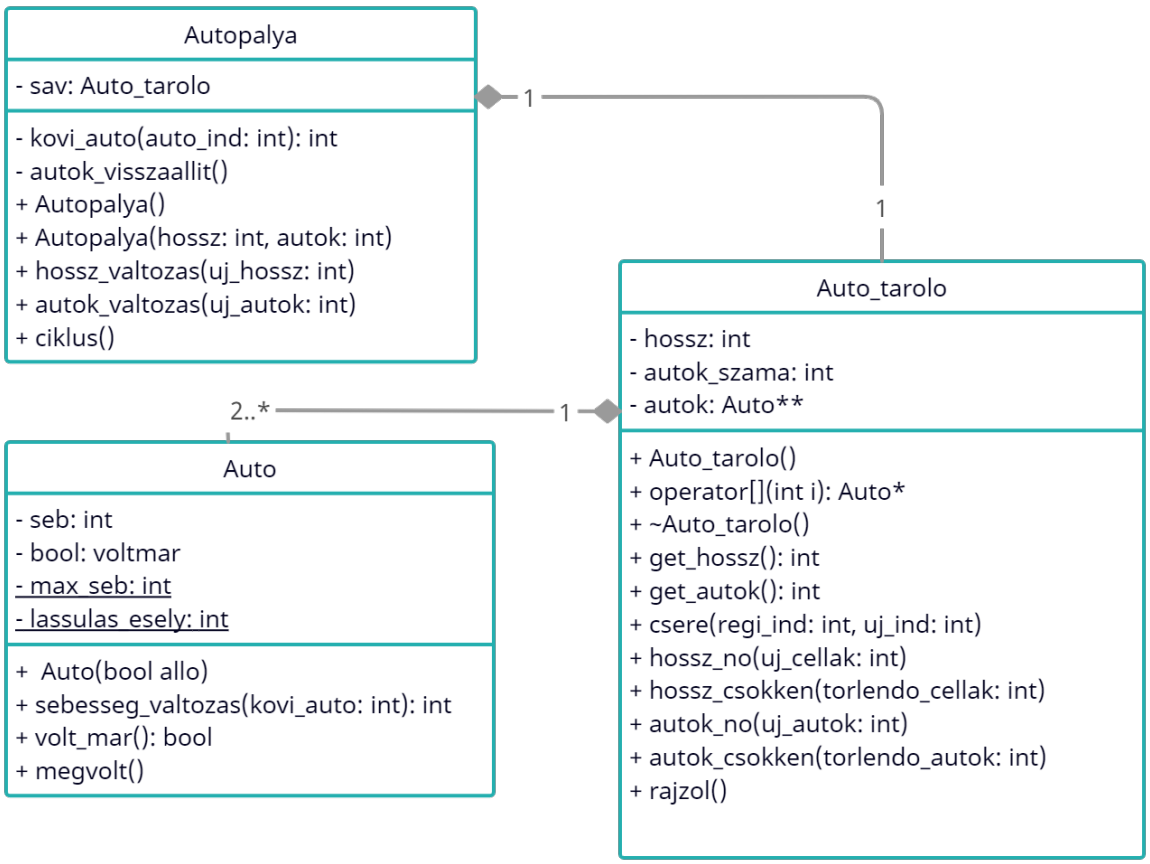
• 4: Kilépés a programból.

Paraméterek változása esetén nem egy új szimuláció indul, hanem a meglévő változik meg, azaz az autók nem álló helyzetből indulnak, és nem feltétlen véletlen vannak elszórva. Az újonnan megjelent autók sebessége kezdetben a maximális sebesség fele, és véletlenszerűen lesznek elosztva a korábbi autók között, amik megtartják eddigi sebességüket. Ha az autópálya hossza megnő, akkor az eddigi cellák után keletkeznek új cellák, ahol kezdetben nem lesznek autók. Ha az autópálya hossza csökken, az egyszerűség kedvéért bizonyos autók törlődnek, majd helyettük újak keletkeznek a már kisebb szakaszon. Az autók számának csökkenése esetén véletlenszerűen kiválasztott autók törlődnek a szimulációból.

A speifikáció elején megfogalmazott feltélekre (**félkövérben**) is sokszor vissza fogok utalni a dokumentáció során.

1.3 Tervezés

A feladat megoldásához mindössze három osztályt használok, amik közül valójában csak egy lesz látható a felhasználó számára. A három osztály kapcsolata az alábbi UML diagramon látszik.



1.3.1 Autopalya osztály

Ez a felhasználó felé is publikus osztály. Ha egy autópályát szeretne modellezni a programjában, akkor egy Autopalya objektumot kell készíteni. A másik két osztályt ez az osztály tartalmazza, és csak a belső működéshez szükségesek. A feladata tehát a feladatban megadott ciklusok végrehajtása és a másik két osztály kezelése. Ennek az osztálynak két konstruktora is van: egy paraméter nélküli és egy kétparaméteres. Ha a paraméter nélkülit használjuk, akkor a szimuláció előtt még a hossz_valtozas() és autok_valtozas() függvénnyel be kell állítani az autópálya paramétereit, ha érdembeli szimulációt szeretnénk (a default kontruktor 0-ra állítja mindkét paramétert, így nem lesz túl érdekes a szimuláció). A két paraméteres konstruktor kivételt dob hibás paraméterek esetén, azonban ha ezt egy try blokkban próbáljuk kezelni, akkor a try blokkon kívül nem lesz elérhető az objektum. Ezért használható a default konstruktor is.

1.3.2 Auto_tarolo osztály

Ennek az osztálynak a feladata az Auto objektumokat tarolni. A konstruktorában mindent nullára állít. A tároló hosszát a hossz_no() és hossz_csokken(), míg az autók számát az autok_no() és autok_csokken() függvényekkel lehet állítani. Az Auto objektumokat az autok_no() függvényben foglalja, és a destruktorban szabadítja fel őket, valamint magát a tömböt is. A tömb valójában Auto∗-eket tárol, és mérete az autópálya hosszával egyezik meg, az üres cellákat NULL pointer jelzi. Így könnyen eldönthető, hogy a tömb adott indexű cellája üres-e, és amikor az autók mozognak, nem kell az objektumokat másolni, csak a pointereket cserélni.

1.3.3 Auto osztály

Az autópálya egy autóját modellező osztály. Feladata az autó sebességét tárolni, és a sebességét változtatni a feladatban leírt első három pont alapján.

1.3.4 Indoklás, bővítés

Első ránézésre az Autopalya osztály akár feleslegesnek is tűnhet, a tagfüggvényei hasonlítanak az Auto_tarolo függvényeire. Az Auto_tarolo azonban egy tarolo objektum, ezért fontosnak gondoltam a felhasználóval és az autókkal való kommunikálást külön objektumba tenni, és ez az Autopalya. Így az Auto_tarolo függvényei mind a tömbre és annak tartalmára vonatkoznak, míg az Autopálya függvényei használják fel igazából a tömböt és végzik el az objektummodel tényleges feladatát.

Ha a modellt bővíteni szeretnénk, a legkézenfekvőbb talán az lenne, ha különböző járműveket tennénk az autópályára. Ezt könnyen megtehetnénk, ha az Auto osztály helyére felvennénk egy absztrakt Jarmu osztályt, és az Auto, valamint a további járművek ennek a leszármazottai lennének. Továbbra is célszerű lenne az üres cellákat NULL pointerként tárolni, hiszen a szimuláció szemppontjából a járműveket az üres cellától szeretnénk megkülönböztetni, nem egymástól, valamint a cellák cseréje ugyanolyan egyszerű maradna, mint most. Az Auto_tarolo osztályt lényegében nem kéne megváltoztatni, csak az Auto osztály helyett mindenhol Jarmu-re kellene hivatkoznia, de mivel nem volt feladat több járművet kezelni, csak az autónál maradtam.

1.4 Fontosabb algoritmusok

A dokumentáció következő fejezeteiben minden függvény szerepét és alapvető működését leírom, de a programban szerepel pár bonyolultabb algoritmus, ezeket ide kiemeltem.

1.4.1 Fisher-Yates

A fisher_yates(int hossz) függvény hossz hosszúságú int tömböt foglal, és feltölti számokkal 0-tól hossz-1-ig, majd a Fisher-Yates algoritmus szerint véletlenszerűen összekeveri az elemeket. Visszatér a tömb címével, amit a hívónak (pl. Auto_tarolo osztály tagfüggvényei) kell felszabadítania. A Fisher-Yates algoritmus itt használt változatának pszeudókódja (a tömb neve legyen tomb):

```
ciklus i = hossz-1-től 1-ig:

    j = véletlen egész úgy, hogy 0 <= j <= i

    tomb[j] és tomb[i] cseréje
```

1.4.2 A Fisher-Yates felhasználása

Az előbbi függvény által generált tömböt két helyen használom a programban: az Auto_tarolo autok_no() és autok↩_csokken() függvényeiben.

1.4.2.1 autok_no()

Itt a függvényt a szabad cellák számával hívom meg, hiszen az a célom, hogy az üres cellákban véletleszerűen osszam el az új autókat. A fisher_yates() függvény összekeveri a számokat, és ebből én leválasztom az első N darabot, ahol N = új autók száma. Ez az N hely lesz, ahova új autót rakok, de ez még nem a tömbbeli index, hanem az üres cellák indexe. A függvény által visszaadott tömböt (vagyis az első N elemét) növekvő sorrendbe rendezem, majd végigmegyek tömbön, közben számon tartva az üres cellák számát, és amikor egy olyan indexűhöz érek, ami szerepel a tömb első N elemében, akkor oda egy új autót generálok.

1.4.2.2 autok_csokken()

Hasonló az előző függvényhez, de itt az autók számát adom meg paraméternek, és a tömb első M elemét választom le, ahol M = törlendő autók száma. Miközben végigmegyek a cellákon, az autókat számolom, és ha egy olyan autóhoz érek, ami szerepel a tömbben, akkor azt törlöm a rendszerből.

1.5 A Tesztprogram

A feladatkiírás szerint a készíteni kell egy tesztprogramot is a modell bemutatására. Ez található az autopalya_main.cpp fájlban. Ez a tesztprogram létrehoz egy Autopalya objektumot paraméter nélkül, majd a felhasználótól kér be a paramétereket. Ezeket a hossz_valtozas() és az autok_valtozas() függvényekkel továbbítja az Autopalya objektumnak, ami rossz paraméterek esetén kivételt dob, amiket a tesztprogram kezel, és ilyenkor új adatokat kér a felhasználótól. Ezután a tesztprogram lefutatt öt ciklust, majd felajánlja a lehetőséget a felhasználónak, hogy megváltoztassa az autópálya paramétereit. Ezeket szintén a fent említett függvényekkel továbbítja az objektumnak. Kilépés esetén az Autopalya destruktora automatikusan lefut, ami lefuttatja az auto_tarolo destrukorát is, felszabadítva minden lefoglalt memóriát.

1.6 Tesztelés

A függvények és a program helyességét a gtest_lite.h segítségével teszteltem. Sajnos ezt a programot meglehetősen nehéz tesztelni, egyrészt mert a programban többször is előfordulnak direkt véletlenszerű viselkedések. Ezt úgy oldom meg, hogy a tesztesetek csak akkor futnak le, ha a CPORTA makró definiálva van minden projektfájlban. Ilyenkor az autók sebessége sosem csökken figyelmetlenség miatt (feladatkiírás 3. pontja), valamint az autok_no() az első üres cellákba teszi az új autókat, nem lesznek véletlenszerűen elosztva. Így a program minden eredménye előre látható és számítható, ami megkönnyíti a tesztelést. A másik probléma a teszteléssel, hogy a program eredménye csak a szabványos kimeneten jelenik meg, amit a gtest_lite-tal nem igen lehet ellenőrizni. Ezért a tesztesetek csak a függvények megfelelő működését ellenőrzik, a program egészét nem.

1.6.1 Auto_tarolo tesztesetei

1. Ellenőrzi, hogy a konstruktor valóban nullára inicializálja-e az objektum tagváltozóit.
2. Ellenőrzi, hogy a hossz_no() és az autok_no() függvények növelik-e a tagváltozók értékét, valamint hogy ténylegesen keletkeznek-e új autók.
3. Ellenőrzi, hogy a csere() függvény a megfelelő esetekben dob-e kivételt, valamint hogyha nem dobott kivételt, akkor tényleg megtörtént-e a csere.
4. Ellenőrzi, hogy a hossz_csokken() és az autok_csokken() függvények csökkentik-e a tagváltozók értékét, valamint hogy ténylegesen törlődnek-e (a megfelelő) autók.

1.6.2 Auto tesztesetei

1. Ellenőrzi, hogy az Auto konstruktora helyesen állítja be a kezdősebességet az allo paraméter alapján.
2. Ellenőrzi, hogy az autók sebesseg_valtozas() függvénye a várt eredményt adja kettő jellegzetes esetben. (kevesebb, illetve több üres cella van előtte, mint a jelenlegi sebessége)
3. Ellenőrzi, hogy az objektum voltmar boolean tagváltozója megváltozott-e a sebbesség változást követően.

1.6.3 Autopalya paraméter nélküli konstruktorához kapcsolódó tesztesetek

Ellenőrzi, hogy az Autopálya osztály hossz_valtozas() és autok_valtozas() függvényei helyes esetekben dobnak-e kivételt.

1.6.4 Autopalya kétparaméteres konstruktorához kapcsolódó tesztesetek

Ellenőrzi, hogy az Autopalya kétparaméteres konstruktora a megfelelő esetekben dob kivételt.

1.6.5 A memóriaszivárgás ellenőrzése

A program fejlesztés közben folyamatosan egy projektben volt a memtrace.h fájlal, így bármilyen fellépő problémát kijavítottam. Emellett WSL-en keresztül a Valgrind memóriakezelés-ellenörzővel is teszteltem a programot. A kész program futása során nem sikeült előidéznem memóriakezelési hibát sem a memtrace.h szerint, sem a Valgrind szerint.

1.7 Konklúzió

Végül szeretnék pár szót írni a programról felhasználói szemmel. Amikor kiválasztottam ezt a feladatot, nagyon szívesen álltam neki a munkának, mert nagyon kíváncsi voltam, hogy fog kinézni, valamint hogy milyen eredményt fogok kapni. Sajnos kicsit csalódnom kellett, hiszen a program konzolos felületen nagyon nehezen használható, azaz nagyon nehéz bármilyen következtetést levonni a szimulációból, hiszen annak változásait nehéz követni. Az talán egyértelműen látszik, hogy sok ciklus után az autók az autópálya vége felé feltorlódnak. Ez amiatt van, hogy a szimuláció hossza valójában nem végtelen, az autópálya utolsó autója előtt egy olyan autó van, ami szintén előre fog mozdulni a másik autóval együtt, de ezt a számolásban nem lehet figyelembe venni, hiszen valahonnan mégis el kell kezdenünk a ciklust.

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 files with brief descriptions:

Chapter 4

Class Documentation

4.1 Auto Class Reference

`#include <auto.h>`

Public Member Functions

- Auto (bool allo)
- bool volt_mar () const
- void megvolt ()
- int sebesseg_valtozas (int kovi_auto)

Private Attributes

- int seb
- bool voltmar

Static Private Attributes

- static const int max_seb = 5
- static const int lassulas_esely = 15

4.1.1 Detailed Description

Az autópálya egy autóját modellező osztály. Feladata az autó sebességét tárolni, és a sebességét változtatni a feladatkiírásban leírt első három pont alapján.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Auto()

```
Auto::Auto (
            bool allo )
```

Az Auto osztály konstruktora. Álló helyzetben 0-ra, egyébként max_seb/2-re inicializálja a sebsséget.

Parameters

allo	Igaz, ha álló helyzetből szeretnénk indítani az autót.

4.1.3 Member Function Documentation

4.1.3.1 megvolt()

```
void Auto::megvolt ( )  [inline]
```

Megváltoztatja az autó állapotát arra, hogy még nem volt ebben a ciklusban. A voltmar tagváltozó setter függvénye, ami mindig hamisra állítja az értékét, hiszen kívülről csak ebben az irányban használt. (Új ciklus kezdetén használandó)

4.1.3.2 sebesseg_valtozas()

```
int Auto::sebesseg_valtozas (
            int kovi_auto )
```

A feladatkiírásban megadott első három lépést végzi el, vagyis beállítja az autó új sebességét. Végül megváltoztatja az autó állapotát arra, hogy már volt ebben a ciklusban. Ha a CPORTA makró definiálva van, a 3. lépést (véletlenszerű lassulás) kihagyja.

Parameters

kovi_auto	Hány üres cella van a következő autó előtt, azaz mennyi lehet az autó maximális sebessége a függvény végén.

Returns

Az autó új sebessége (ennyi cellát kell előre mozgatni)

4.1.3.3 volt_mar()

```
bool Auto::volt_mar ( ) const  [inline]
```

Volt-e már ez az autó az adott ciklusban? A voltmar tagváltozó getter függvénye.

Returns

Igaz, ha már volt ebben a ciklusban.

4.1.4 Member Data Documentation

4.1.4.1 lassulas_esely

```
const int Auto::lassulas_esely = 15  [static], [private]
```

Annak a valószínűsége∗100, hogy egy autó egy ciklusban egy egységet lassul, feladatkiírás szerint 15.

4.1.4.2 max_seb

```
const int Auto::max_seb = 5  [static], [private]
```

Az autók maximális sebessége, feladatkiírás szerint 5.

4.1.4.3 seb

```
int Auto::seb  [private]
```

Az autó jelenlegi sebessége

4.1.4.4 voltmar

```
bool Auto::voltmar  [private]
```

Volt-e már vizsgálva az autó az adott ciklusban?

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

- auto.h
- auto.cpp

4.2 Auto_tarolo Class Reference

```
#include <auto_tarolo.h>
```

Public Member Functions

- Auto_tarolo ()
- Auto ∗ operator[ ] (int i) const
- ∼Auto_tarolo ()
- int get_hossz () const
- int get_autok () const
- void csere (int regi_ind, int uj_ind)
- void hossz_no (int uj_cellak)
- void hossz_csokken (int torlendo_cellak)
- void autok_no (int uj_autok)
- void autok_csokken (int torlendo_autok)
- void rajzol () const

Private Attributes

- int hossz
- int autok_szama
- Auto ** autok

4.2.1 Detailed Description

Auto objektumokat tároló osztály.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Auto_tarolo()

```
Auto_tarolo::Auto_tarolo ( )  [inline]
```

Az Auto_tarolo osztály konstruktora, ami az autópálya hosszát és az autók számát nullára állítja, az autókat tároló tömböt pedig NULL-ra. A szimuláció elkezdéséhez be kell állítani a két paraméter értékét a hossz_no() és az autok_no() függvényekkel.

4.2.2.2 ~Auto_tarolo()

```
Auto_tarolo::~Auto_tarolo ( )
```

Destruktor, ami felszabadítja a rendszer összes autóját, majd magát a tömböt is

4.2.3 Member Function Documentation

4.2.3.1 autok_csokken()

```
void Auto_tarolo::autok_csokken (
            int torlendo_autok )
```

A paraméterben kapott értékkel lecsökkenti a rendszerben lévő autók számát. Véletlenszerűen választja ki a törlendő autókat. A paraméter értékét nem ellenőrzi, az a hívó dolga.

Parameters

torlendo_autok	Ennyi autót töröl a rendszerből.

4.2.3.2 autok_no()

```
void Auto_tarolo::autok_no (
            int uj_autok )
```

A paraméterben kapott értékkel megnöveli a rendszerben lévő autók számát. Ha még nincs autó a rendszerben, vagyis a szimuláció még nem kezdődött el, akkor álló helyzetből indítja az új autókat. Az új autókat véletlenszerűen szórja el a már létező autók között. Ezzel a függvénnyel kell az autók számát inicializálni, hiszen a konstruktor azt 0-ra állítja.

Parameters

uj_autok	Ennyi autót vesz fel a rendszerbe

4.2.3.3 csere()

```
void Auto_tarolo::csere (
            int regi_ind,
            int uj_ind )
```

Megcseréli a két index által mutatott címet.

Parameters

regi_ind	Első index, itt Auto∗-nek kell szerepelnie
uj_ind	Második index, itt NULL-nak kell szerepelnie

Exceptions

std::invalid_argument	Ha valamelyik index nem jó adatra mutat

4.2.3.4 get_autok()

```
int Auto_tarolo::get_autok ( ) const  [inline]
```

Az Auto_tarolo autok_szama tagváltozójának getter függvénye.

Returns

A jelenleg tárolt autók száma

4.2.3.5 get_hossz()

```
int Auto_tarolo::get_hossz ( ) const  [inline]
```

Az Auto_tarolo hossz tagváltozójának getter függvénye.

Returns

Az autópálya jelenlegi hossza.

4.2.3.6 hossz_csokken()

```
void Auto_tarolo::hossz_csokken (
            int torlendo_cellak )
```

A paraméterben kapott értékkel lecsökkenti a tároló hosszát, azaz az utolsó torlendo_cellak cellát törli a rendszerből. Ha ezeken a mezőkön voltak autók, azokat is törli, majd meghívja az autok_no() függvényt a törölt autók számával, hogy visszaállítsa az autók számát az eredetire. A paraméter értékét nem ellenőrzi, az a hívó dolga.

Parameters

torlendo_cellak	Hány cellát szeretnénk törölni.

4.2.3.7 hossz_no()

```
void Auto_tarolo::hossz_no (
            int uj_cellak )
```

Megnöveli a tároló, és így az autópálya hosszát a paraméterben kapott értékkel. Ehhez új tömböt foglal, ahol az új cellákat mind NULL-ra állítja, majd a régi tömböt törli. Ezzel a függvénnyel kell az autópálya hosszát inicializálni, hiszen a konstruktor azt 0-ra állítja.

Parameters

uj_cellak	Ennyi cellával növeljük meg a tömböt

4.2.3.8 operator[]()

```
Auto * Auto_tarolo::operator[] (
            int i ) const
```

Indexelő operátor

Parameters

i	index

Returns

A tömb i-edik Auto∗-e

4.2.3.9 rajzol()

```
void Auto_tarolo::rajzol ( ) const
```

Kirajzolja a képernyőre az autópálya jelen állását. Elválasztásként az autópálya teljes hosszában ír egy sort "-" karakterből, majd minden cellába " "-t ír, ha üres a cella, "o"-t egyébként. Végül egy újabb "-" sorral zárja.

4.2.4 Member Data Documentation

4.2.4.1 autok

```
Auto** Auto_tarolo::autok  [private]
```

Az Auto∗-eket tartalmazó tömb

4.2.4.2 autok_szama

```
int Auto_tarolo::autok_szama  [private]
```

Autók száma az autópályán, azaz hány nem üres cella van a rendszerben

4.2.4.3 hossz

```
int Auto_tarolo::hossz  [private]
```

A tömb, azaz az autópálya hossza (cellákban)

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

- auto_tarolo.h
- auto_tarolo.cpp

4.3 Autopalya Class Reference

`#include <autopalya.h>`

Public Member Functions

- Autopalya ()
- Autopalya (int hossz, int autok)
- void ciklus ()
- void hossz_valtozas (int uj_hossz)
- void autok_valtozas (int uj_autok)

Private Member Functions

- int kovi_auto (int auto_ind) const
- void autok_visszaallit () const

Private Attributes

- Auto_tarolo sav

4.3.1 Detailed Description

Ez a felhasználó felé is publikus osztály. Ha egy autópályát szeretne modellezni a programjában, akkor egy Autopalya objektumot kell készíteni. A másik két osztályt ez az osztály tartalmazza, és csak a belső működéshez szükségesek. Az osztály feladata a feladatkiírásbanban megadott ciklusok végrehajtása és a másik két osztály kezelése.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Autopalya() [1/2]

```
Autopalya::Autopalya ( )
```

Az Autopalya osztály default konstruktora, ami a tárolót inicializálja (annak a default konstruktorával). Ha ezt a konstruktort használjuk, akkor szükség van a hossz_valtozas() és az autok_valtozas() függvényekre, az autópálya paramétereinek beállításához.

4.3.2.2 Autopalya() [2/2]

```
Autopalya::Autopalya (
            int hossz,
            int autok )
```

Az Autopalya osztály kétparaméteres konstruktora. Inicializálja az Auto_tarolo-t, ellenőrzi majd inicializálja a paramétereket.

Parameters

hossz	Az autópálya hossza cellákban
autok	Az autók száma az autópályán

Exceptions

std::invalid_argument	Ha autok $<$ 2 $||$ autok $>=$ hossz.

4.3.3 Member Function Documentation

4.3.3.1 autok_valtozas()

```
void Autopalya::autok_valtozas (
            int uj_autok )
```

Ellenőrzi a paraméterként adott új autók számát, és a tároló megfelelő függvényét meghívja.

Parameters

uj_autok	Erre az értékre állítja az autók számát.

Exceptions

std::invalid_argument	Ha nem teljesülnek a feltételek a paraméterre.

4.3.3.2 autok_visszaallit()

```
void Autopalya::autok_visszaallit ( ) const  [private]
```

A tömb minden autóját visszaállítja abba az állapotba, hogy lehessen rajtuk a ciklust végezni.

4.3.3.3 ciklus()

```
void Autopalya::ciklus ( )
```

A feladatleírásban megadott 4 pontot hajtja végre a segédosztályokon, majd kirajzolja az autópálya jelenlegi állását. Az autópálya végétől kezdve, visszafele minden autóra megvizsgálja, hogy hány üres cella van előtte a kovi_auto() függvénnyel, meghatározza az autó új sebességét az Auto osztály sebesseg_valtozas() függvényével, végül előremozgatja az autót az új sebességével az Auto_tarolo osztály csere() függvényével. Közben figyel arra, hogy minden autót csak egyszer vizsgáljon.

4.3.3.4 hossz_valtozas()

```
void Autopalya::hossz_valtozas (
            int uj_hossz )
```

Ellenőrzi a paraméterként adott új hossz értékét, és a tároló megfelelő függvényét meghívja. Végül kirajzolja a jelenlegi állapotot (kivéve, ha először állítjuk be a hosszt).

Parameters

uj_hossz	Erre az értékre állítja az autópálya hosszát

Exceptions

std::invalid_argument	Ha nem teljesülnek a feltételek a paraméterre.

4.3.3.5 kovi_auto()

```
int Autopalya::kovi_auto (
            int auto_ind ) const  [private]
```

Megadja, hogy az adott indexű autó előtt hány üres cella van a következő autóig. Ha a sorban az utolsó autót vizsgáljuk, akkor a következő autó a sorban első autóra vonatkozik.

Parameters

auto_ind	A vizsgált autó indexe a sav tömbben

Returns

Üres cellák száma a következő autóig

4.3.4 Member Data Documentation

4.3.4.1 sav

```
Auto_tarolo Autopalya::sav  [private]
```

Az autópálya egyetlen sávja, ami egy Auto_tarolo objektum, ebben tárolódnak az autók.

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

- autopalya.h
- autopalya.cpp

Chapter 5

File Documentation

5.1 auto.cpp File Reference

Az Auto osztály privát és publikus függvéneit tartalmazó fájl.

```
#include "auto.h"
#include <cstdlib>
```

5.1.1 Detailed Description

Az Auto osztály privát és publikus függvéneit tartalmazó fájl.

Ide tartozik a tkonstruktor és a sebesség változás logikája.

5.2 auto.h File Reference

Classes

- class Auto

5.3 auto_tarolo.cpp File Reference

Az Auto_tarolo osztály privát és publikus függvéneit tartalmazó fájl.

```
#include "auto_tarolo.h"
#include <cstdlib>
#include <ctime>
```

Functions

- int int_hasonlit (const void *p1, const void *p2)
- int * fisher_yates (int hossz)

5.3.1 Detailed Description

Az Auto_tarolo osztály privát és publikus függvéneit tartalmazó fájl.

Ide tartozik a tömb inicializálása, feltöltése adatokkal, adatok vagy méret változtatása.

5.3.2 Function Documentation

5.3.2.1 fisher_yates()

```
int* fisher_yates (
            int hossz )
```

hossz hosszúságú int tömböt foglal, és feltölti számokkal 0-tól hossz-1-ig, majd a Fisher-Yates algoritmus szerint véletlenszerűen összekeveri az elemeket.

Parameters

hossz	0-tól hossz-1-ig fogja összekeverni a számokat

Returns

Dinamikusan foglalt int*, ami véletlenszerű sorrendben tartalmazza az egész számokat 0-tól hossz-1-ig

5.3.2.2 int_hasonlit()

```
int int_hasonlit (
            const void * p1,
            const void * p2 )
```

Generikus rendező algoritmus által használt összehasonlító függvény. Két int-et hasonlít, növekvő sorrendben.

Parameters

p1	
p2	

Returns

5.4 auto_tarolo.h File Reference

```
#include <iostream>
#include "auto.h"
```

Classes

- class Auto_tarolo

Functions

- int * fisher_yates (int hossz)

5.4.1 Function Documentation

5.4.1.1 fisher_yates()

```
int* fisher_yates (
            int hossz )
```

hossz hosszúságú int tömböt foglal, és feltölti számokkal 0-tól hossz-1-ig, majd a Fisher-Yates algoritmus szerint véletlenszerűen összekeveri az elemeket.

Parameters

hossz	0-tól hossz-1-ig fogja összekeverni a számokat

Returns

Dinamikusan foglalt int*, ami véletlenszerű sorrendben tartalmazza az egész számokat 0-tól hossz-1-ig

5.5 autopalya.cpp File Reference

Az Autopalya osztály privát és publikus függvéneit tartalmazó fájl.

```
#include "autopalya.h"
```

5.5.1 Detailed Description

Az Autopalya osztály privát és publikus függvéneit tartalmazó fájl.

Ide tartoznak a szimuláció ciklusainak vezérlése és a felhasználói input ellenőrzése.

5.6 autopalya.h File Reference

```
#include "auto_tarolo.h"
#include <cstdlib>
#include <ctime>
```

Classes

- class Autopalya

5.7 autopalya_main.cpp File Reference

Az autópályamodell használatát bemutató tesztprogram.

```
#include <iostream>
#include "autopalya.h"
```

Functions

- int szam_beolvas (int min, int max)
- int ensure_int_input ()
- int main ()

5.7.1 Detailed Description

Az autópályamodell használatát bemutató tesztprogram.

5.7.2 Function Documentation

5.7.2.1 ensure_int_input()

```
int ensure_int_input ( )
```

Addig kér inputot a felhasználótól, amíg egész számot nem kap. Ha az inputban nem számot talál, az input további részét eldobja.

Returns

Felhasználó inputja, garantáltan int.

5.7.2.2 main()

```
int main ( )
```

Az Autópálya modellt bemutató tesztprogram, amiben a felhasználó dönti el, milyen hosszú legyen az autópálya, valamint hogy hány autó legyen rajta. Egymás után 5 ciklust lefuttat, majd lehetőséget ad a felhasználónak, hogy megváltoztassa az autópálya paramétereit.

Returns

Exit code.

5.7.2.3 szam_beolvas()

```
int szam_beolvas (
            int min,
            int max )
```

Kiírja a menü opcióit, majd választást kér a felhasználótól. Ha a felhasználói input nincs [min, max]-ban, újat kér.

Parameters

min	Legkisebb elfogadható felhasználói input
max	Legnagyobb elfogadható felhasználói input

Returns

A felhasználó választása, garantáltan [min, max]-ból.

5.8 autopalya_teszt.cpp File Reference

gtest_lite.h segítségével írt tesztek a programhoz. A CPORTA makró definiálása (projekt szinten) esetén futnak le.

```
#include "gtest_lite.h"
#include "autopalya_teszt.h"
#include "autopalya.h"
```

5.8.1 Detailed Description

gtest_lite.h segítségével írt tesztek a programhoz. A CPORTA makró definiálása (projekt szinten) esetén futnak le.

5.9 autopalya_teszt.h File Reference

Functions

- void autopalya_teszt ()

5.9.1 Function Documentation

5.9.1.1 autopalya_teszt()

```
void autopalya_teszt ( )
```

Index