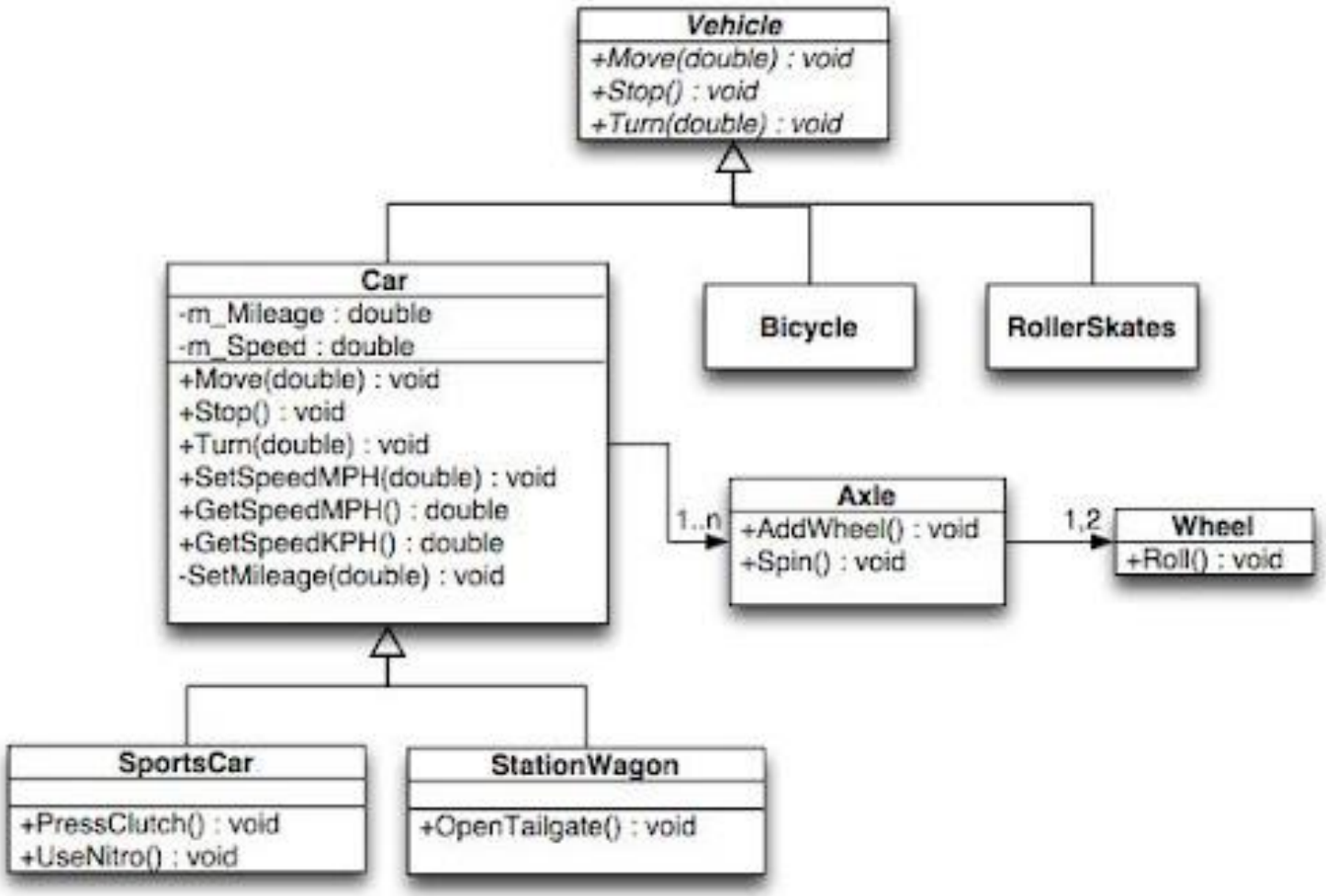
ADATSZERKEZETEK ÉS ALGORITMUSOK

C++ ismétlés

Órai feladat I



Órai feladat I

- `Vehicle` osztály
- Deklarálunk alap változókat
  - `type, kmh, capacity`
- Kell egy konstruktor, ami ezeket feltölti
- Getter-ek
- Absztrakt lesz, tehát kell egy pure **`virtual`** függvény (`moveOn()`)

Órai feladat I

- `Bike` osztály
- Frissítjük a konstruktort
- Implementáljuk a `moveOn()` függvényt, hogy példányosíthassuk
  - Állítja a kilométerórát és kiírja, hogy mennyit mentünk

Órai feladat I

- `Car` osztály
- Frissítjük a konstruktort
- Implementáljuk a `moveOn()` függvényt, hogy példányosíthassuk
- Implementálunk egy üzemanyag menedzsmentet:
  - Van fogyasztása és üzemanyag tartálya az autónak
  - Van az üzemanyagnak ára (osztályváltozó)
  - Az autót fel lehet tankolni és le lehet kérdezni egy út költségét

Órai feladat I

- `Taxi` osztály
- Frissítjük a konstruktort
- Implementálunk egy költség menedzsmentet:
  - A taxisöfőrnek van pénztárcája és óradíja
  - A tankolás pénzbe kerül (ki gondolta volna…)
  - Le szeretnénk tudni osztani /fő az útiköltséget (üzemanyag + óradíj +10%)

Órai feladat I

- `Bus` osztály
- Frissítjük a konstruktort
- Módosítjuk a költség menedzsmentet:
  - Utasok tudnak leszállni/felszállni
  - Az útiköltség fix (buszjegy)
  - Ki akarjuk számolni, hogy maximum hány kilométert mehetünk, amivel még pozitív profitot érünk el

Inicializáló listák – C++11

C++11 kiterjeszti a változók és összetett objektumok inicializációjának lehetőségét – nincs = jel!

```
int a{0};
string s{"hello"};
string s2{s}; //copy construction
vector <string> vs{"alpha", "beta", "gamma"};
map<string, string> stars
    { {"Superman", "+1 (212) 545-7890"},
      {"Batman", "+1 (212) 545-0987"}};
double *pd= new double [3] {0.5, 1.2, 12.99};
```

Oszályokban is

```
class C {
  string s{"abc”};
  double d=2.0;
  char * p {nullptr};
  int y[5] {1,2,3,4};
 public:
  C(double _d) : d(_d) {}
};
C c; //c.d = 2.0
C c2(5.0); //c.d = 5
```

Operátorok

```
class  Complex{
double re = 0.0;
double im = 0.0;
public:
Complex operator+ (const Complex& _other) const;

 };

 Complex Complex::operator+ (const Complex& _other) const{
Complex c;
c.re = this->re + _other.re;
c.im = this->im + _other.im;
return c;    }
```

Mikor van egy Complex a, b és „meghívjuk” a + b túlterhelt operátort, akkor a+b, ahol _a_ lesz a **this** tag, és b lesz az _other tag. Tehát itt a this pointerrel a példányra, mint önmagára hivatkozhatunk.

További info az operátorokról:

http://en.wikibooks.org/wiki/C++_Programming/Operators/Operator_Overloading

http://hu.wikipedia.org/wiki/Oper%C3%A1torok_(C%2B%2B)

Friend

Operátorokat túlterhelhetünk úgy is, hogy az nem az objektum tagfüggvénye. Ekkor szembesülünk azzal a problémával, hogy nem tudjuk elérni az objektum privát tagjait.

Erre jelent megoldást a friend kulcsszó!

Egy osztálynak friend-je lehet egy függvény, egy osztály, vagy egy másik osztály tagfüggvénye, ami így eléri private vagy protected adattagjait.

Tipikus ilyen példa a (<<, >>) operátorok túlterhelése, amikkel streamekre/-ről írhatunk/olvashatunk:

```
friend std::ostream& operator<<(std::ostream& stream, const Car& z);
```

Ezzel túlterheltük, az << operátort, vagyis cout << c << endl értelmezhető, amennyiben c Car típusú változó.

Assignment operator, copy constructor

Mindegyikkel egy objektumot másolunk le.

Az assignment operátor automatikusan lefut értékadásnál, míg a copy constructor érték szerinti paraméter átadásnál (ahol a paraméter nem pointer), ill. függvényből való visszatérésnél (ahol a visszatérési érték nem referencia vagy pointer típusú).

Leggyakrabban azért írjuk meg, mert az osztályban van dinamikusan lefoglalt memóriaterületre hivatkozó mutató

Ha nem írjuk meg, akkor csak ún. shallow copy történik, vagyis csak a változók értéke másolódik. (Tehát pointerek esetén csak a tárolt cím másolódik át, a mutatott memóriaterület tartalma nem.)

Dinamikus memória-kezelés során figyelni kell rá (new-delete párok)!

Assignment operator, copy constructor

Formájuk a következő:

```
class A {
      A (const A& _other);
      A& operator= (const A& _other);
};
```

A “Big Four”

Copy konstruktor, default konstruktor, assignment operátor, destruktor.

Ezek és csak ezek azok a tagok, amelyeket a compiler automatikusan legenerál, ha mi nem írjuk meg (vagy default konstruktor esetén valami más konstruktort).

C++11 óta a move konstruktor és move assigment operátor is ezek közé tartozik

A delete és a default tagfügvények

A fordító egy sor tagfüggvényt hoz létre, ha nem definiáljuk azokat. C++11-ben dönthetünk az automatikusan létrejövő tagfüggvények létrehozásáról (default) vagy elutasításáról (delete).

```
class Car {
public:
Car() = default; // Car c; OK
Car(const Car&) = default; // Car b(c); OK
Car& operator=(const Car&) = delete; // b = a; NEM OK
virtual ~Car() = default;

};
```

A C++ fordító osztályszintű operátor-függvényeket (&, *, ->, new, delete) definiál az osztályokhoz, melyek használatát meg is tilthatjuk.

```
class C {
public:
   void *operator new(size_t) = delete;
   void *operator new[](size_t) = delete;
   };
```

```
int main() {
   C *c = new C; // HIBA
   C *t = new C[3]; // HIBA
   C _c;
   C _t[10];
   }
```

A delete és a default példák

- **delete**:
  - Nem akarjuk, hogy az osztályunk másolható legyen (mert mondjuk költséges és csak a move-ot akarjuk engedni, vagy csak egy pédányt akarunk belőle), ekkor kitörölhetjük a copy és az assignment operátort
- **default**:
  - Átláthatóbbá teszi a kódot, jelzi, hogy gondolták a default verziókra és elfogadod azokat
  - A konstruktort priváttá/protected-é akarjuk tenni (létezik ilyen mintázat), de elfogadjuk a default verziót

Felhasználó által definiált literálok

C++11 –ben lehetőség van saját literálok készítésére:

```
 inline double operator"" _deg (long double degree) {
return degree * 3.14159265 / 180.0;
 }
 double rad = 90.0_deg; // szög = 1.570796325


 unsigned operator"" _Magic(const char* _magic) {
unsigned b = 0;
for(unsigned int i = 0; _magic[i]; ++i)
b = b*2 + (_magic[i] == '1');
return b;
 }
 int mask = 110011_Magic; // maszk = 51
```

Órai feladat II

- Készítsünk egy mátrix osztályt:
- A konstruktor paramétereivel megadhatjuk méreteit, amiket aztán le is lehet kérdezni,
- Legyen assignment operátora és copy constructora,
- Legyen ==, + operátora,
- A () operátorral lehessen elérni az értékeit (amit aztán be is lehet állítani),
- Legyen << operátora, amivel streamre írhatunk,
- Végül legyen az osztálynak egy statikus függvénye, amivel egységmátrixokat lehet gyártani.

Gyakorló feladat G02F03

Írjunk egy iskolát modellező alkalmazást!

- Először is az iskolába járnak emberek, akiknek van nevük, születési évük és nemük.
- Az iskolában tanítanak tantárgyakat, amelyeknek van nevük, leírásuk és hozzá vannak rendelve évfolyamok, amikor ezeket oktatni kell.
- Az iskolába járó emberek között vannak tanulók, akik egy osztályba tartoznak, akiknek van egy tanulmányi átlaguk, és akiknél el kell tárolni a szülő/nevelő elérhetőségeit.
- Az iskolában tanítanak tanárok, akikhez bizonyos tárgyak vannak hozzárendelve. A tanárok között vannak osztályfőnökök, akiknek van osztályuk. A tanároknak van heti óraszámuk, órabérük, ami alapján a havi fizetésük kiszámolható.
- A tanárok között van egy igazgató, akinek a fizetése nem csak az órabér és a heti óraszám alapján képződik (van valamifajta plusz fizetés).
- Végül van az iskola, amiben vannak tanulók, tanárok, igazgató, aminek van egy címe, neve és ahol kiszámolható, hogy havonta összesen mennyi fizetést kell kiosztani.

Gyakorló feladat G02F04

Tervezzünk meg és írjunk meg egy alkalmazott osztályt!

Útmutatás:

- Gondold át, hogy egy alkalmazottat egy tetszőleges vállalatnál milyen adatokkal lehet reprezentálni.
- Ha ez megvan, gondold át azt, hogy mik lehetnek azok az eljárások/műveletek, amiket ehhez az alkalmazott osztályhoz hozzá lehetne kapcsolni. Gondold át, hogy a meglévő adattagokon milyen értelmes műveleteket/lekérdezéseket lehetne végrehajtani.
- Gondold át, hogy a fentiek összességéből melyek azok, amelyek minden alkalmazott esetén ugyanazok/ugyanazt az eredményt/hatást érik el, és melyek azok, amelyek minden alkalmazott esetén különbözőek.
- Majd gondold át, hogy mi az, ami hozzátartozna az osztály publikus interfészéhez, és mi az, amit érdemes elrejteni.
- Ha mindez megvan, a megfelelő nyelvi elemeket felhasználva írd meg az alkalmazott osztályt!

Gyakorló feladat G02F04 (folyt.)

Feladat: A megírt Alkalmazott osztályból leszármaztatva valósítsunk meg egy Főnök osztályt!

Útmutatás:

- Melyek azok a változók, amelyeket célszerű lenne elfedni?
- Milyen új változókat lehetne bevezetni?
- Melyek azok a metódusok, melyeket felül kéne definiálni?
- Milyen új metódusokat kellene megírni?

Majd ezután írj egy olyan main függvényt, melyben kipróbálod a polimorfizmus és a dinamikus kötés lehetőségeit!

Gyakorló feladat G02F06

Egy városban reggel mindenki megy valahova. Szimuláld a városban élők utazását és készíts róla statisztikát.

- A városban van nyugdíjas, felnőtt és diák. Generálj mindből 1 - 80 között valamennyit.
- Legyen koruk is, tanuló 0 - 25, felnőtt 26 - 61, nyugdíjas 62 - 80.
- Utazhatnak vonattal, autóval, taxival és biciklivel. Véletlenszerűen válaszd ki, hogy az egyes emberek melyik tömegközlekedési eszközt használják.
- Egy ember 15 - 45 km-et utazik, ez is véletlenszerű.
- A járműveknek van kapacitása: vonat 60, autó 5, taxi 4, bicikli 1. Ezeket töltsd fel emberekkel úgy, hogy a járművek mindig maximum távra (45 km) mennek. (ha 2 nyugdíjas 20 km-t megy vonattal, és 3 diák 40 km-t vonattal, ők utazhatnak egy járművön)
- A járműveknek van kilométerenkénti díja. Vonat 18 Ft/km, autó 36 Ft/km, taxi 50 Ft/km és biciklinél elégetett kalória van 33 kalória/km.

Gyakorló feladat G02F06

- Továbbá vannak egyedi tulajdonságok.
- Vonat: típus („Stadler Flirt", „Ganz V43", „Siemens Desiro", „Mallard"), pálya szám (1000, 1001, 1002, 1003). Nem kell minden vonatot megalkotni, elég annyit amennyi éppen használatban van. Értsd, ha csak 100 ember utazik vonattal elég 2 vonatot példányosítani. (Az első vonat így „Stadler Flirt" és 1000 a pályaszáma, a második vonat „Ganz V43" és 1001 a pályaszáma stb.)
- Autó: Mikor gyártották (1990 - 2010).
- Taxi: Mikor gyártották (1990 - 2010), és egy száma (1- 100) véletlenszerűen.
- Bicikli: méret (24 - 32).
- Statisztikák, amiket el kell készíteni:
  - Konzolra: mennyit költöttek összesen az emberek autóra, taxira vonatra, és mind háromra összesen, illetve összesen mennyi kalóriát égettek el. Milyen nevű és pályaszámú vonatok mentek, átlag bicikli méret és autók kora, taxik kora, taxik sorszáma.
  - Fájlba: minden emberről statisztikák így: 10. ember, aki 62 éves, 29 km-et utazik és vonattal.

Megjegyzés: a maximum ponthoz mindenképpen használni kell: absztrakt osztály, konstansok, polimorfizmus, dinamikus kötés, referencia, pointer.

Gyakorló feladat G02F07

- Készíts egy Tárgy osztályt. Minden tárgynak van súlya, és kérésre ki tudja írni, hogy mi az a tárgy, és milyen nehéz.
- Készíts három tetszőleges konkrét alosztályt (pl Toll), amelyeknek mind legalább egy további tulajdonsága van (pl. Toll esetében a színe).
- Készíts egy Táskát, mely egy bizonyos mennyiségű – létrehozáskor megadott – súlyt képes befogadni
  - A Táskába véletlen módon tegyünk véletlen tulajdonságú Tárgyakat, amíg az meg nem telik
  - A Táska legyen képes felsorolni, hogy milyen tárgyak vannak benne, ha megtelt a táska, ezt tegyük is meg.

Megjegyzés: a maximum ponthoz mindenképpen használni kell: absztrakt osztály, dinamikus tömb, polimorfizmus, dinamikus kötés, referencia, pointer.

Házifeladat KHF01

Az eddig tanultak alapján készítsük el az ábrán látható osztályokat a szokásos műveletekkel!

Az AbstractArray – mint ahogy neve is sugallja – legyen absztrakt.

FixedArray: ne legyen megváltoztatható a kezdeti mérete.

DynamicArray (Array): ha a jelenleg foglalt memória nem elég, foglaljon le mégegyszer annyit, mint a jelenlegi méret.

PagedArray: ha a jelenleg foglalt memória nem elég, foglaljon le még annyit, mint a kezdeti méret.