# 12. előadás

## Jövő héttől vizsgaidőszak

Előadás nem lesz.

## Hány órás lesz a ZH: 1+3

Beugró: 1 óra

Gyakorlati rész: 3 óra

Összesen: 4 óra

CodeBlocks helyett csak parancssort lehet használni fordításhoz.

## Template mindig a headerben legyen kifejtve, ne másik fájlban

Különben nem működik.

## Öröklődés

### Alap lista

|  |  |
| --- | --- |
| $$list1.h$$ | $$list1.cpp$$ |
| #ifndef LIST\_H#define LIST\_H#include <iostream>class list**{**public**:** list**();** **~**list**();** list **\***remove**();** void append**(**list **\***lp**);** void insert**(**list **\***lp**);** list **\***get\_next**()** const **{** **return** next**;** **}** list **\***get\_prev**()** const **{** **return** prev**;** **}** void print**(** std**::**ostream**&** os**)** const**;** static int get\_nid**()** **{** **return** nid**;** **}**private**:** const int id**;** list **\***next**;** list **\***prev**;** static int nid**;** // ADA: private limited list**(** const list **&**rhs**);** list **operator=(** const list **\***rhs**);****};**std**::**ostream**&** **operator<<(** std**::**ostream**&,** const list**&);**#endif /\* LIST\_H \*/ | #include <iostream>#include "list1.h"**using** **namespace** std**;**// statikus membert deklarálni kell!int list**::**nid **=** 0**;**// inicializáló lista// warning-ot ad, mert a mezők deklarációs sorrendje// a list.h-ban mást volt. A list.h az elsődleges!list**::**list**()** **:** id**(**nid**),** prev**(**0**),** next**(**0**){** **++**nid**;****}**list**::~**list**(){** remove**();****}**list **\***list**::**remove**(){** **if** **(** prev **)** prev**->**next **=** next**;** **if** **(** next **)** next**->**prev **=** prev**;** next **=** prev **=** 0**;** **return** **this;****}**void list**::**append**(** list **\***lp**){** next **=** lp**->**next**;** lp**->**next **=** **this;** prev **=** lp**;** **if** **(** next **)** next**->**prev **=** **this;****}**void list**::**insert**(** list **\***lp**){** prev **=** lp**->**prev**;** lp**->**prev **=** **this;** next **=** lp**;** **if** **(** prev **)** prev**->**next **=** **this;****}**void list**::**print**(** ostream**&** os**)** const**{** os **<<** "[ id = " **<<** id **<<** " ]"**;****}**ostream**&** **operator<<(** ostream**&** os**,** const list **&**l**){** l**.**print**(**os**);** **return** os**;****}** |

### Alap napló

|  |  |
| --- | --- |
| $$diary1.h$$ | $$diary2.cpp$$ |
| // Még nem polimorfikus#ifndef DIARY\_H#define DIARY\_H#include <iostream>#include <string>#include "list1.h"#include "date.h"class diary **:** public list**,** public date**{**// lehetséges lenne örökölni az std::string-ből is \*public**:** diary**(** const char **\***e**,** int y**=**2000**,** int m**=**1**,** int d**=**1**);** diary**(** const char **\***e**,** const char **\***d**);** diary**(** const char **\***e**,** const date**&** dd**);** // ebben a verzióban még nem virtuális void print**(** std**::**ostream**&** os**)** const**;**private**:** std**::**string event**;****};**std**::**ostream**&** **operator<<(** std**::**ostream**&** os**,** const diary**&** d**);**#endif /\* DIARY\_H \*/ | #include <iostream>#include "diary1.h"**using** **namespace** std**;**diary**::**diary**(**const char **\***e**,**int y**,**int m**,**int d**)** **:** date**(**y**,**m**,**d**),**event**(**e**)** **{** **}**diary**::**diary**(**const char **\***e**,** const char **\***d**)** **:** date**(**d**),** event**(**e**)** **{** **}**diary**::**diary**(**const char **\***e**,** const date**&** dd **)** **:** date**(**dd**),** event**(**e**)** **{** **}**void diary**::**print**(** ostream**&** os**)** const**{** list**::**print**(**os**);** os **<<** ", "**;** date**::**print**(**os**);** os **<<** ", "**;** os **<<** event**;****}**ostream**&** **operator<<(** ostream**&** os**,** const diary**&** d**){** // no cpy constr d**.**print**(**os**);** **return** os**;****}** |

Többszörös öröklődés: class diary **:** public list**,** public date

\* Lehetséges lenne örökölni az $std : :string$-ből is. Probléma: 50-60 fölösleges metódus meg adattag jönne vele. Nem akarjuk a string összes szolgáltatását.

|  |  |  |
| --- | --- | --- |
| Hibás main 1 | Hibás main 2 | Hibás main 3 |
| #include <iostream>#include "diary1.h"**using** **namespace** std**;**int main**(){** diary d1**(** "Tanfolyam kezdete"**,** 2000**,** 6**,** 19**);** diary d2**(** "Tanfolyam vege"**,** 2000**,** 6**,** 27**);** diary d3**(** "Pihenes"**,** 2000**,** 6**,** 28**);** d2**.**append**(&**d1**);** d3**.**append**(&**d2**);**// for ( diary \*dp = &d1; dp; dp = dp->get\_next() )// syntax error! **for** **(** list **\***lp **=** **&**d1**;** lp**;** lp **=** lp**->**get\_next**()** **){** cout **<<** **\***lp**;** cout **<<** endl**;** **}** **return** 0**;****}** | #include <iostream>#include "diary1.h"**using** **namespace** std**;**int main**(){** diary d1**(** "Tanfolyam kezdete"**,** 2000**,** 6**,** 19**);** diary d2**(** "Tanfolyam vege"**,** 2000**,** 6**,** 27**);** diary d3**(** "Pihenes"**,** 2000**,** 6**,** 28**);** d2**.**append**(&**d1**);** d3**.**append**(&**d2**);** **for** **(** diary **\***dp **=** **&**d1**;** dp**;** dp **=** **(**diary**\*)**dp**->**get\_next**()** **){**//for ( list \*lp = &d1; lp; lp = lp->get\_next() )  cout **<<** **\***dp**;** // most már jó! ...vagy mégsem? cout **<<** endl**;** **}** **return** 0**;****}** | #include <iostream>#include "diary1.h"**using** **namespace** std**;**int main**(){** diary d1**(** "Tanfolyam kezdete"**,** 2000**,** 6**,** 19**);** diary d2**(** "Tanfolyam vege"**,** 2000**,** 6**,** 27**);** diary d3**(** "Pihenes"**,** 2000**,** 6**,** 28**);** // orvul közbelép a felhasználó: list l**;** d2**.**append**(&**d1**);** d3**.**append**(&**d2**);** l**.**append**(&**d2**);** **for** **(** diary **\***dp **=** **&**d1**;** dp**;** dp **=** **(**diary**\*)**dp**->**get\_next**()** **){**//for ( list \*lp = &d1; lp; lp = lp->get\_next() )  cout **<<** **\***dp**;** // runtime error!  cout **<<** endl**;** **}** **return** 0**;****}** |

Jó main:

|  |
| --- |
| $$main2.cpp$$ |
| #include <iostream>#include "diary2.h"**using** **namespace** std**;**int main**(){** diary d1**(** "Tanfolyam kezdete"**,** 2000**,** 6**,** 19**);** diary d2**(** "Tanfolyam vege"**,** 2000**,** 6**,** 27**);** diary d3**(** "Pihenes"**,** 2000**,** 6**,** 28**);** d2**.**append**(&**d1**);** d3**.**append**(&**d2**);** list l**;** l**.**insert**(&**d2**);** **for** **(** list **\***lp **=** **&**d1**;** lp**;** lp **=** lp**->**get\_next**()** **){** **if** **(** diary **\***dp **=** **dynamic\_cast<**diary**\*>(**lp**)** **){** cout **<<** **\***dp**;** **}else{** cout **<<** **\***lp**;** **}** cout **<<** endl**;** **}** **return** 0**;****}** |

Dinamikus cast​olás: null-t ad vissza, ha nem sikerül. Ezt használja ki a fenti kód.

És $diary2.h$-t használ:

|  |  |
| --- | --- |
| $$diary2.h$$ | $$diary2.cpp$$ |
| // polimorfikus: mert egyik õse polimorfikus#ifndef DIARY\_H#define DIARY\_H#include <iostream>#include <string>#include "list2.h"#include "date.h"class diary **:** public list**,** public date**{**public**:** diary**(** const char **\***e**,** int y**=**2000**,** int m**=**1**,** int d**=**1**);** diary**(** const char **\***e**,** const char **\***d**);** diary**(** const char **\***e**,** const date**&** dd**);** /\* virtual \*/ void print**(** std**::**ostream**&** os**)** const**;**private**:** std**::**string event**;****};**std**::**ostream**&** **operator<<(** std**::**ostream**&** os**,** const diary**&** d**);**#endif /\* DIARY\_H \*/ | #include <iostream>#include "diary2.h"**using** **namespace** std**;**diary**::**diary**(**const char **\***e**,**int y**,**int m**,**int d**)** **:** date**(**y**,**m**,**d**),**event**(**e**)** **{** **}**diary**::**diary**(**const char **\***e**,** const char **\***d**)** **:** date**(**d**),** event**(**e**)** **{** **}**diary**::**diary**(**const char **\***e**,** const date**&** dd **)** **:** date**(**dd**),** event**(**e**)** **{** **}**void diary**::**print**(** ostream**&** os**)** const**{** list**::**print**(**os**);** os **<<** ", "**;** date**::**print**(**os**);** os **<<** ", "**;** os **<<** event**;****}**ostream**&** **operator<<(** ostream**&** os**,** const diary**&** d**){** // no cpy constr d**.**print**(**os**);** **return** os**;****}** |

Virtuális függvény: futásidőben derül ki, melyik verziója fut le. Emiatt nem optimalizálható.

$diary2.h$ pedig $list2.h$-t használ:

|  |  |
| --- | --- |
| $$list2.h$$ | $$list2.cpp$$ |
| #ifndef LIST\_H#define LIST\_H#include <iostream>class list**{**public**:** list**();**// ökölszabály: polimorfikus osztálynak legyen// virtuális destruktora! virtual**~**list**();** list **\***remove**();** voidappend**(**list **\***lp**);** voidinsert**(**list **\***lp**);** list **\***get\_next**()** const **{** **return** next**;** **}** list **\***get\_prev**()** const **{** **return** prev**;** **}**// virtuális függvény virtual void print**(** std**::**ostream**&** os**)** const**;** static int get\_nid**()** **{** **return** nid**;** **}**private**:** const intid**;** list **\***prev**;** list **\***next**;** static int nid**;** list**(** const list **&**rhs**);** list **operator=(** const list **\***rhs**);****};**std**::**ostream**&** **operator<<(** std**::**ostream**&,** const list**&);**#endif /\* LIST\_H \*/ | #include <iostream>#include "list2.h"**using** **namespace** std**;**// statikus membert deklarálni kell!!int list**::**nid **=** 0**;**list**::**list**()** **:** id**(**nid**),** prev**(**0**),** next**(**0**){** **++**nid**;****}**list**::~**list**(){** remove**();****}**list **\***list**::**remove**(){** **if** **(** prev **)** prev**->**next **=** next**;** **if** **(** next **)** next**->**prev **=** prev**;** next **=** prev **=** 0**;** **return** **this;****}**void list**::**append**(** list **\***lp**){** next **=** lp**->**next**;** lp**->**next **=** **this;** prev **=** lp**;** **if** **(** next **)** next**->**prev **=** **this;****}**void list**::**insert**(** list **\***lp**){** prev **=** lp**->**prev**;** lp**->**prev **=** **this;** next **=** lp**;** **if** **(** prev **)** prev**->**next **=** **this;****}**void list**::**print**(** ostream**&** os**)** const**{** os **<<** "[ id = " **<<** id **<<** " ]"**;****}**ostream**&** **operator<<(** ostream**&** os**,** const list **&**l**){** l**.**print**(**os**);** **return** os**;****}** |

Legjobb main:

|  |
| --- |
| $$main3.cpp$$ |
| #include <iostream>#include "diary2.h"**using** **namespace** std**;**int main**(){** diary d1**(** "Tanfolyam kezdete"**,** 2000**,** 6**,** 19**);** diary d2**(** "Tanfolyam vege"**,** 2000**,** 6**,** 27**);** diary d3**(** "Pihenes"**,** 2000**,** 6**,** 28**);** d2**.**append**(&**d1**);** d3**.**append**(&**d2**);** list l**;** l**.**insert**(&**d2**);** **for** **(** list **\***lp **=** **&**d1**;** lp**;** lp **=** lp**->**get\_next**()** **){** cout **<<** **\***lp **<<** endl**;** **}** **return** 0**;****}** |

Virtuális lett a $print$, ezért polimorfikusan, többalakúan működik. Az objektum dinamikus típusa alapján, futásidőben dől el, melyik $print$ fut le.

## Polimorfizmus

Mindig pointeren vagy referencián keresztül adjunk át. Polimorfizmus megőrzéséhez elkerülendő a "szeletelődés".

Mi az a szeletelődés: $slicing$. Az öröklő típus extra adatokkal egészítheti ki a bázist, de ez levágódik, ha érték szerint, másoló konstruktorral adjuk át.

|  |
| --- |
| $$main.h$$ |
| #include <iostream>#include <list>#include <algorithm>#include "vehicle.h"#include "car.h"#include "bus.h"#include "truck.h"**using** **namespace** std**;**void global\_print**(**vehicle **\***vp**){** vp**->**print**();****}**int main**(){** list**<**vehicle**\*>** vl**;** vl**.**push\_back**(** **new** car**(**"abc123"**)** **);** vl**.**push\_back**(** **new** bus**(**"bbb123"**,** 45**)** **);** vl**.**push\_back**(** **new** truck**(**"eee765"**,** 4.5**)** **);** vl**.**push\_back**(** **new** truck**(**"fff098"**,** 3.5**)** **);** vl**.**push\_back**(** **new** bus**(**"ccc456"**,** 55**)** **);** vl**.**push\_back**(** **new** car**(**"bcd2"**)** **);**/\* for ( list<vehicle\*>::iterator vi = vl.begin(); vi != vl.end(); ++vi){ global\_print(\*vi);} \*/ for\_each**(** vl**.**begin**(),** vl**.**end**(),** global\_print**);** **return** 0**;****}** |

Memóriaszemét: Meg kell hívni a **delete**-et. A virtuális destruktor fog lefutni a típusoknak megfelelően. (Minek, a program úgyis kilép.)

### Busz

|  |
| --- |
| $$bus.h$$ |
| #ifndef BUS\_H#define BUS\_H#include <string>#include "vehicle.h"class bus **:** public vehicle**{**public**:** bus**(**std**::**string s**,** int per**)** **:** vehicle**(**s**),** persons**(**per**)** **{}** /\* virtual \*/ bool mv**()** **{** **return** persons **<** 50**;** **}**private**:** int persons**;****};**#endif /\* BUS\_H \*/ |

### Kamion

|  |
| --- |
| $$truck.h$$ |
| #ifndef TRUCK\_H#define TRUCK\_H#include <string>#include "vehicle.h"class truck **:** public vehicle**{**public**:** truck**(**std**::**string s**,** double w**)** **:** vehicle**(**s**),** weight**(**w**)** **{}** /\* virtual \*/ bool mv**()** **{** **return** vehicle**::**mv**()** **&&** weight **<** 4.0**;** **}**private**:** double weight**;****};**#endif /\* TRUCK\_H \*/ |

### Autó

|  |
| --- |
| $$car.h$$ |
| #ifndef CAR\_H#define CAR\_H#include <string>#include "vehicle.h"class car **:** public vehicle**{**public**:** car**(**std**::**string s**)** **:** vehicle**(**s**)** **{}** // virtual ~car() {} automatikusan létrejön /\* virtual \*/ bool mv**()** **{** **return** get\_reg**().**length**()** **>** 5**;** **}****};**#endif /\* CAR\_H \*/ |

### Jármű

|  |
| --- |
| $$vehicle.h$$ |
| #ifndef VEHICLE\_H#define VEHICLE\_H#include <string>#include <typeinfo>// polimorfikus osztály, mert van virtuális függvénye,// és absztrakt osztály, mert van legalább egy pure virtual függvényeclass vehicle**{**public**:** vehicle**(**std**::**string s**)** **:** reg**(**s**)** **{}** virtual **~**vehicle**()** **{** **}** // virtuális destruktor! // ez nem virtual bool print**(){** std**::**cout **<<** **typeid(\*this).**name **<<** " with reg = " **<<** get\_reg**()** **<<** " : " **<<** mv**()** **<<** std**::**endl**;** **return** mv**();** **}** std**::**string get\_reg**()** const **{** **return** reg**;** **}** // pure virtual virtual bool mv**()** **=** 0**;**private**:** std**::**string reg**;****};**inline bool vehicle**::**mv**()** **{** **return** reg **!=** ""**;** **}**#endif /\* VEHICLE\_H \*/ |

(mv = műszaki vizsga)

$pure virtual$ – Mindenkinek kötelező felüldefiniálni, de mégis lehet implementációja. Miért jó ez? Lehet a bázisra hivatkozni altípusokból, így nem kell lemásolni több helyre ugyan azt a kódot.
Pl.: bool mv**()** **{** **return** vehicle**::**mv**()** **&&** weight **<** 4.0**;** **}** ellenőrzi a rendszám létezését és a súlyt is.

Mindig írjuk ki a megörökölt virtuális destruktort, stb. Nehogy valaki félreértse a kódunkat.

## Inicializálás

Sorrend fontos, compiler​nek jogában áll megcserélni.

Még valamit akart mondani tanár úr, de elfelejtette.

Előadás vége