Bevezetés a programozásba 2

6. Előadás: A `const`

ISMÉTLÉS

Interface - Implementation

```
struct Particle {
    int x,y;
    unsigned char r,g,b;
    void rajzol();
};
```

```
void Particle::rajzol() {
    gout << move_to(x, y)
    << color (r, g, b)
    << dot;
}
```

ISMÉTLÉS

Láthatóság

```
struct Particle {
    Particle(int X, int Y);
    virtual void mozog( … );
    virtual void rajzol( … );
protected:
    double x,y;
    unsigned char r,g,b;
};
```

ISMÉTLÉS

Osztály

```
class Particle {
public:
    Particle(int X, int Y);
    virtual void mozog( … );
    virtual void rajzol( … );
protected:
    double x,y;
    unsigned char r,g,b;
};
```

ISMÉTLÉS

Implementáció elrejtése

particle.hpp

```
#ifndef PARTICLE_HPP
#define PARTICLE_HPP

struct Particle {
    int x,y;
    unsigned char
r,g,b;
    void rajzol();
};

#endif
```

particle.cpp

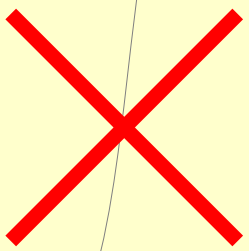
```
#include "particle.hpp"

void Particle::rajzol() {
    gout << move_to(x, y)
    << color (r, g, b)
    << dot;
}
```

main.cpp

```
#include "particle.hpp"

int main() {
 …
    Particle p;
    p.rajzol();
}
```



ISMÉTLÉS

A fordítás menete

- Preprocesszor
- Forráskódok értelmezése, típusok feltérképezése (pl. méret miatt).
- Forrásfájlonként egy tárgykód (object, .o) fájl létrehozása. Ebben gépi kódú részletek vannak, előkészítve a csatlakozási pontokat
- A linker összeköti a tárgykódokat
  - “undefined reference”: egyik tárgykódban sincs feloldva egy csatlakozási pont, pl függvényhíváshoz nem található függvény implementáció

ISMÉTLÉS

Fordítási egység

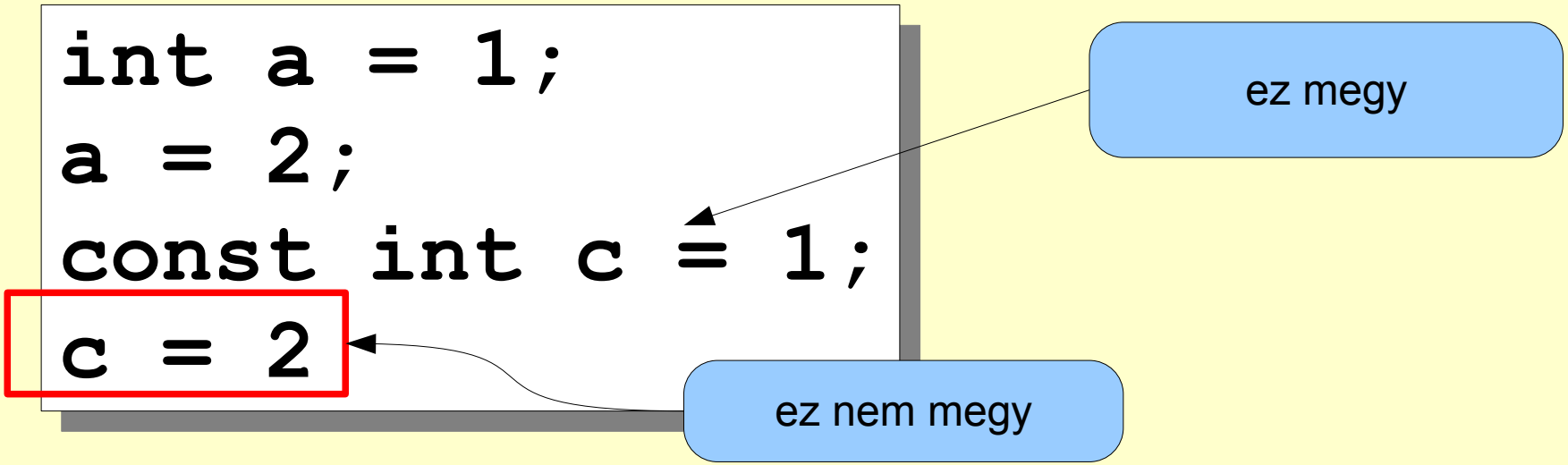
- Fordítási egység az, ami önállóan fordítható
- A C++ nyelvben minden .cpp fájl önálló fordítási egység
- Moduláris programozás
  - Felgyorsítja a fordítást: csak a megváltozott forráskódokhoz tartozó egységek fordulnak újra
  - A személyi felelősség értelmezhető a rendszer alkotóelemeire: önállóan tesztelhető modulok
  - Strukturális / moduláris programozás

Konstansok

- Értékek
  - 5, vagy „Hello world”
  - Literál
- Változónak látszó, azonosítóval ellátott konstans
  - `const double pi = 3.14159;`
  - Konstans
- Globális változót nem illik csinálni, globális konstanssal nincs probléma.

Konstansok

- kizárólag kezdeti értékkel lehet deklarálni:
  - definiálni
- Értékadás nincs értelmezve
- Fordítási időben ismert az értéke



Konstans referencia

- `const T & r;`
- Gyakran használt paraméterezési lehetőség
- Használandó:
  - Másoló konstruktor
  - operátorok
  - …
- Gyors, biztonságos
  - C nyelvben nincs ilyen, ott mutatót használnak, és odafigyelnek, hogy ne állítsák el

A `const` használata

- Ha valami **`const`**, akkor nem végezhető vele olyan művelet, ami az értéket megváltoztat***hat***ja
- A változatlanságra garanciát ad a fordító
  - paraméterként csak akkor adható át, ha az adott függvény ott konstans paramétert vár, így a lokális változó is **`const`**
  - még mutatót sem lehet rá állítani, csak olyan mutatót, ami konstansokra tud csak mutatni
- Honnan tudható, hogy tagfüggvény hívható-e?

Konstans tagfüggvények

```
int main() {
  const string s = ”Hello”;
  cout << s.length();
}
```

- Ez működik. A fordítóprogram valahonnan tudja, hogy a **`length()`** nem fogja megváltoztatni az értéket

Konstans tagfüggvények

```
int main() {
  const string s = ”Hello”;
  cout << s.length();
}
```



```
size_type string::length() const
{
  …
}
```

Konstans tagfüggvények

```
class Particle {
public:
    Particle(int X, int Y);
    virtual void mozog( … );
    virtual void rajzol( … ) const;
protected:
    double x,y;
    unsigned char r,g,b;
};
```

Konstansság

- Mezei változónak konstans értékül adható
  - úgyis másolat készül, az eredeti nincs veszélyben
  - mutatót nem lehet rá állítani
- Konstansnak pályafutása kezdetén mezei változó értékül adható
  - paraméterként kapott lokális változók
- Ez lehetőséget ad arra, hogy a változó által tárolt értéknek időszakosan, egy-egy függvényhívás idejére védettséget biztosítsunk

ISMÉTLÉS

A `this`

- Kizárólag tagfüggvényben használható
- Mutató, ami a meghívott objektumra mutat
  - más szavakkal maga az implicit paraméter
- Tipikus használata
  - ha az objektum tagfüggvényén belül hív olyan függvényt, ahol paraméterként saját magát kell átadnia
  - visszatérési értékként értékadás operátorban, és egyéb olyan operátorokban, függvényekben, ahol a kifejezés hatása a visszatérési értékben is hasznos

A this

```
struct koord {
    double x,y;
    koord operator=(const koord& masik) {
        x=masik.x; y=masik.y;
        return *this;
    }
};

int main() {
    koord a,b,c;
    c=b=a;
    return 0;
}
```

Konstans tagfüggvények

- A **`this`** típusa C class egy sima tagfüggvényében: **`C *`**
- Konstans tagfüggvényben pedig **`const C *`**
- Tehát a konstans tagfüggvényen belül a mezőkre nem hívható nem konstans művelet
- Ez a technikai háttere a konstansság ellenőrzésének konstans tagfüggvényekben

Áttekintés

- A **`const`** használata az eddigiek alapján egy már tartalmas programra láncreakciót okoz
  - Ha egy tagfüggvényt megjelölünk, hogy konstans, a bennelevő meg nem jelölt tagfüggvényekre hivatkozás, illetve paraméterátadás nem const paraméterként nem fordul le
  - Ha paraméternél jelöljük a konstansságot, nem hívhatjuk meg az objektum nem megjelölt tagfüggvényeit
- Érdemes idejekorán ezt elintézni
- Később sem vészes, a fordító keresgél helyettünk

Kiskapuk

- Néha előfordul, hogy a szigorú konstansság követelmények bajt okoznak
  - Meglevő, nem módosítható könyvtár készítői nem gondoltak a konstansok helyes kezelésére
- **`const_cast`**
- **`mutable`**

A `mutable`

- A **`mutable`** egy osztály mezőjének módosítója lehet
- Azt jelenti, hogy ez a mező akár egy konstans objektumban is megváltozhat
- Tipikus használata:
  - olvasásszám nyilvántartás
  - cache
- Jó tervezésnél a reprezentációt nem deklaráljuk mutable-ként

`const_cast`

- Általában nem állíthatunk mutatót konstansra, ha mégis kell*, akkor használjuk a **`const_cast`**-ot
- **`const T* cm =`** .. a konstansra mutató **`T* m = const_cast<T*>(cm);`**
- Ezzel most **`m`** változtatható módon mutat bele a konstansba, és ez veszélyes
- Nem definiált következmény!

* nagyon ritkán kell. Ha kellett, akkor máris van miért újratervezni az egészet.

Biztonságos programozás és a `const`

- Fordítási időben kiderül a probléma
  - konstans tagfüggvények használatával a lehetséges módosítások köre szűkül
  - következetesség a fordítóprogram támogatásával
- Ismétlődő literálok elkerülése (magic numbers)
  - a literálok típusai bizonytalanok (1/2 vs 0.5)

Egyéb érdekességek

- Mutatók is lehetnek konstansok:
  - const int * m
  - int const * m
  - int * const m
  - const int * const m
- Az első két írásforma felcserélhető
  - mutató, ami konstans int-re mutat
- Ha a * után van a const, a mutató konstans, tehát nem mutathat később máshová
- Jobbról balra érdemes olvasni