Tények, érdekességek az informatika világából

Magyar közmondások, ha leprogramoznánk C/C++-ban ezeket

💻 Egyszer volt Budán kutyavásár!

- `#define NR_OF_DOG_MARKET_IN_BUDA 1`
- A **#define** makró tipikusan konstansok és ismétlődő értékek megadására szolgál C/C++-ban.
- `const int NR_OF_DOG_MARKET_IN_BUDA = 1;`
- C++-ban: ez azért jobb, mert a **#define** csak egy szöveges helyettesítés a preprocessorban, míg a **const** / **constexpr** típusbiztos, és bekerül a fordító ellenőrzési rendszerébe is.
- `constexpr int NR_OF_DOG_MARKET_IN_BUDA = 1;`
- C++11 óta megjelent a **constexpr** kulcsszó.
- `bool visitedBuda = false;`

```
string dogMarket(string city = „Buda") {
    if (visitedBuda && city == „Buda") return „";
    visitedBuda = true;
    return „vásár";
}
```

- Írhatunk egy függvényt is, amely csak akkor téríti vissza a „vásár" stringet, ha a **visitedBuda** változó **false**.
- `#include <iostream>`

```
#include <memory>
#include <mutex>

class DogMarket {
private:
    static std::unique_ptr<DogMarket> instance;
    static std::once_flag initFlag;

    // privát konstruktor, hogy kívülről ne lehessen
  példányosítani
    DogMarket() {
        std::cout << „Egyszer volt Budán kutyavásár." <<
  std::endl;
    }

public:
    // másolás és értékadás tiltása
```

```
    DogMarket(const DogMarket&) = delete;
    DogMarket& operator=(const DogMarket&) = delete;

    static DogMarket& getInstance() {
        std::call_once(initFlag, []() {
            instance.reset(new DogMarket());
        });
        return *instance;
    }

    void trade() {
        std::cout << „Kutyát árulunk Budán!” <<
   std::endl;
    }
};

// statikus adattagok definiálása
std::unique_ptr<DogMarket> DogMarket::instance;
std::once_flag DogMarket::initFlag;

// --- Használat ---
int main() {
    DogMarket& market1 = DogMarket::getInstance();
    market1.trade();

    DogMarket& market2 = DogMarket::getInstance();
    market2.trade();

    return 0;
}
```

- Az „Egyszer volt Budán kutyavásár!” közmondás singletonnal szinte tökéletesen modellezhető, mert a kutyavásár csak egyszer jöhet létre. Fent egy egyszerű C++ megoldást láthatunk Singleton mintával. A `DogMarket` osztályt csak egyszer lehet példányosítani, a konstruktor privát. A `getInstance()` biztosítja, hogy csak egyszer jöjjön létre a példány (Budán csak egy kutyavásár volt). A `std::once_flag` és `std::call_once` garantálja, hogy szálbiztos is legyen. Ha többször hívjuk a `getInstance()`-t, mindig ugyanazt az egyetlen példányt kapjuk vissza.
-
```
#include <iostream>
#include <memory>
#include <mutex>
```

```
#include <stdexcept>

class DogMarket {
private:
    static std::unique_ptr<DogMarket> instance;
    static std::once_flag initFlag;
    static bool alreadyCreated;

    DogMarket() {
        if (alreadyCreated) {
            throw std::runtime_error("Egyszer volt Budán
  kutyavásár!");
        }
        alreadyCreated = true;
        std::cout << „Kutyavásár megnyitva Budán.” <<
  std::endl;
    }

public:
    // másolás és értékadás tiltása
    DogMarket(const DogMarket&) = delete;
    DogMarket& operator=(const DogMarket&) = delete;

    static DogMarket& getInstance() {
        std::call_once(initFlag, []() {
            instance.reset(new DogMarket());
        });
        return *instance;
    }

    void trade() {
        std::cout << „Kutyát árulunk Budán!” <<
  std::endl;
    }
};

// statikus adattagok definiálása
std::unique_ptr<DogMarket> DogMarket::instance;
std::once_flag DogMarket::initFlag;
bool DogMarket::alreadyCreated = false;

// --- Használat ---
```

```
int main() {
    try {
        DogMarket& market1 = DogMarket::getInstance();
        market1.trade();

        // Második „vásár” kísérlet Budán
        DogMarket& market2 = DogMarket::getInstance();
        market2.trade();
    }
    catch (const std::runtime_error& e) {
        std::cerr << „Exception: „ << e.what() <<
   std::endl;
    }

    return 0;
}
```

- A fenti programot kiegészíthetjük kivételkezeléssel is. Amikor az első példány létrejön, kiírja, hogy „Kutyavásár megnyitva Budán.” Ha bárki újra próbálkozna, a konstruktor kivételt vált ki, s a képernyőn megjelenik, hogy „Egyszer volt Budán kutyavásár!”

Ki korán kel, aranyat lel.

- `if (wakeUpTime < sunrise) ++gold;`
- Ha korábban kelünk fel, mint a Nap, akkor növelődik az aranyaink száma.
- `gold = (wakeUpTime < sunrise) ? ++gold : gold;`
- Kifejezésben is használhatjuk, ha átfogalmazzuk a `? :` operátor segítségével.

Ki mint vet, úgy arat.

- `harvest = sow(seeds);`
- Ez egy egyszerű megfogalmazás.
- `harvest = std::transform_reduce(seeds.begin(), seeds.end(), 0, std::plus<>(), plant);`
- STL-stílusban, szójátékkal.
- `auto harvest = [&](){ return sow(seeds); }();`
- Lambda-kifejezással.
- `auto harvest = std::for_each(seeds.begin(), seeds.end(), plant);`
- Függvény túlterheléssel.

Kétszer ad, aki gyorsan ad.

- ```
  int give(bool fast) {
    return fast ? 2 : 1;
  ```
  ```

```
}
```

💻 Amilyen az adjonisten, olyan a fogadjisten.

```
string greet(string input) return input;
```

💻 Lassan járj, tovább érsz.

```
if (speed == „slow") reach = farther;
```

💻 Aki másnak vermet ás, maga esik bele.

```
void digPit(Target target) {
  if (target == nullptr) target = self;
  trap(target);
}
```

💻 Jobb félni, mint megijedni.

```
if (cautious) danger.avoid();
else surprise.scare();
```

💻 Sok lúd disznót győz.

```
if (geese.size() > pig.strength) pig.defeated = true;
```

💻 Addig jár a korsó a kútra, míg el nem törik.

```
while (!pitcher.broken()) {
  pitcher.walkTo(well);
  pitcher.fill(well);
}
```

- Ez egy előltesztelős ciklus.

```
do {
  pitcher.walkTo(well);
  pitcher.fill(well);
} while (!pitcher.broken());
```

- Ez pedig hátultesztelős ciklussal. Itt legalább egyszer járt a korsó a kúton.

💻 Addig nyújtózkodj, ameddig a takaród ér.

```
if (wish > blanket.length()) wish = blanket.length();
```

💻 Sok bába közt elvész a gyerek.

```
if (midwives.size() > MANY) child.lost = true;
```

💻 Amelyik kutya ugat, az nem harap.

```
if (dog.barks()) dog.bites = false;
```

💻 Kutyából nem lesz szalonna.

```
if (animal == „dog") product != „bacon";
```

💻 Nem esik messze az alma a fájától.

```
apple.distance = tree.position + ran-
dom_small_offset();
```

K.L.