



# C++: A TAVALYI GYŐZTES KATEGÓRIÁJA

4. forduló

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

## Ismertető a feladathoz

### Fontos információk

A forduló után a megoldások publikálásával együtt iránymutatásként elérhetőek lesznek a **helyezéssel kapcsolatos információk**, látni fogod, hogy a kategóriában a játékosok 20%, 40% vagy 60%-a közé tartozol-e épp.

Felhívjuk figyelmedet, hogy a következő, **5. fordulótól az egyes kategóriák csak a kijelölt napokon lesznek megoldhatóak 7-22 óra között**, érdemes letöltened a naptárat a [Kategóriáim](#) menüpontban.

### Negyedik forduló

Ha a feladatok szövege máshogy nem rendelkezik, a kérdések a C++20 szabványra vonatkoznak.

Felhasznált idő: 02:01/15:00

Elért pontszám: 0/27

## 1. feladat 0/4 pont

Miért lett deprecated, majd került eltávolításra a C++ szabványból az **auto\_ptr**?

### Válasz

- ☐ Az egyéb automatikusan kezelt pointer-típusok (**shared\_ptr**, **weak\_ptr**) bevezetése miatt az **auto\_ptr**-t átnevezték **unique\_ptr**-re.
- ☐ Az **auto\_ptr** nem volt implementálható Windows platformon.
- ☒ Az **auto\_ptr** copy constructora igazából nem másol, hanem mozgat, és ez bugokhoz vezethet.
- ☐ Nem volt specifikálva, hogy az **auto\_ptr::release()** metódus által visszaadott pointert milyen módon kell felszabadítani (**delete** vagy **free**), és a különböző implementációk nem egyeztek ebben.

## Magyarázat

Az `auto_ptr` helyébe a C++11-ben bevezetett `unique_ptr` lépett. A `move` szemantika bevezetése tette lehetővé a helyes válaszban leírt probléma javítását.

## 2. feladat 0/6 pont

Mely állítások igazak?

### Válaszok

- ☒ A `constexpr` függvények automatikusan `inline`-ok.
- ☒ Az `inline` függvényeket a `.h/.hpp` fájlban is definiálhatjuk.
- ☒ Az `inline` függvényeket a `.cpp/.cc` fájlban is definiálhatjuk.
- ☐ Az `inline` függvények hívását a fordító mindig kioptimalizálja.
- ☐ Egy fordítási egységen belül több példányban is definiálhatjuk ugyanazt az `inline` függvényt, de ezeknek a definícióknak azonosnak kell lenniük.
- ☒ Több fordítási egységben is definiálhatjuk ugyanazt az `inline` függvényt, de ezeknek a definícióknak azonosnak kell lenniük.
- ☐ Több fordítási egységben is definiálhatjuk ugyanazt az `inline` függvényt, és ezek a definíciók eltérőek is lehetnek.

## Magyarázat

A "publikus" `inline` függvényeket a `.h`-ben definiáljuk, hiszen elérhető kell, hogy legyen a definíció ott, ahol használjuk. Azonban a csak egy fordítási egységben használt `inline`-okat a `.cpp`-ben is deklarálhatjuk és definiálhatjuk, ebben semmi nem akadályoz meg minket.

Egy `inline` függvénynek egy fordítási egységben egy definíciója lehet, viszont különböző fordítási egységekben is lehet definíciója, amelyeknek azonosnak kell lenniük. Ez elő is fordul, ha több `.cpp` fájlból `include`-oljuk az `inline` függvény definícióját tartalmazó `.h` fájlt. Bár az `inline` hozzáfűti a fordítót abban, hogy egy függvény kódjához hozzáférve eliminálja magát a függvényhívást, valójában nem más, mint egy `linkage specifier`, hiszen a fordítónak az optimalizálásra nézve semmi kötelezettsége nincs.

## 3. feladat 0/5 pont

Két különböző .cpp-ben egy-egy ugyanolyan nevű és szignatúrájú függvényünk van definiálva, de különböző tartalommal. A függvényeket csak az adott .cpp fájlban belül használjuk. Linkelési hibát kapunk. Melyik eszközzel érhető el, hogy a program leforduljon, és azt csinálja, amit akarunk?

### Válaszok

- ☒ static
- ☐ inline
- ☒ névtelen namespace
- ☐ private

### Magyarázat

A static kulcsszó jelentése ebben a kontextusban: "internal linkage", azaz a szimbólum (a függvény) csak ebben a fordítási egységben fog látszani.

A névtelen namespace technikailag más módon, de ugyanazt valósítja meg, ti. a lefordított object fájlban a függvény globálisan egyedi nevet fog kapni, ezáltal más fordítási egységek nem tudnak rá hivatkozni.

Az inline erre nem használható, mert inline módon deklarált függvénynek minden fordítási egységben ugyanaz kell, hogy legyen a definíciója.

A private csak osztálydefiníciókon belül használható, a metódusok láthatóságának biztosítására.

## 4. feladat 0/6 pont

Mi kerüljön a (\*) helyére, ha azt akarjuk, hogy a program outputja garantáltan -1 legyen?

```
#include <bit>
#include <cstring>
#include <iostream>
#include <limits>

int main() {
    unsigned u = std::numeric_limits<unsigned>::max();
    (*)
    std::cout << i << std::endl;
}
```

### Válaszok

- ☐ int i = (int)u;
- ☐ int i = static\_cast<int>(u);
- ☒ int i = std::bit\_cast<int>(u);

- ☒ `int i; memcpy(&i, &u, sizeof(int));`
- ☐ `union { int i; unsigned u2; }; u2 = u;`
- ☒ `int i; reinterpret_cast<unsigned&>(i) = u;`

## Magyarázat

A `memcpy` és a `bit_cast` minden további nélkül működik, a `bit_cast` kifejezetten erre való.

A `reinterpret_cast` is jó, hiszen `int` és `unsigned` ugyanannak az egész típusnak az előjeles ill. előjel nélküli változatai. (ld. [https://en.cppreference.com/w/cpp/language/reinterpret\\_cast](https://en.cppreference.com/w/cpp/language/reinterpret_cast), "Type aliasing")

A `static_cast` és a C-style cast (ami ebben az esetben a `static_cast`-ot valósítja meg) nem jók, mivel előjeles egész túlcsoordulása lép fel, ami C++-ban `undefined behaviour`. Az `int` túlcsoordulás kihasználása nemcsak elméletileg, de gyakorlatban is kerülendő, pl. a fordító az "`i+1 > i`" típusú feltételeket kioptimalizálhatja, mivel feltételezi, hogy nincs túlcsoordulás.

Bár széles körben támogatott, de a `union`-ös trükk szintén nem szabványos, mivel egy uniónak egyszerre csak egy aktív eleme lehet. A legutóbb megkonstruált elem az aktív, és a többi elem olvasása `undefined behaviour`.

Ld. még: [https://www.youtube.com/watch?v=\\_qzMpk-22cc](https://www.youtube.com/watch?v=_qzMpk-22cc)

## 5. feladat 0/6 pont

Az alábbiak közül mely kódrészletek okoznak `undefined behaviour`-t (UB)?

### Válaszok

- ☒ `int x = std::numeric_limits<int>::max(); ++x;`
- ☐ `unsigned x = std::numeric_limits<unsigned>::max(); ++x;`
- ☒ `int x{12}; x /= 0;`
- ☒ `for(;;);`
- ☒ `int *p{nullptr}; std::cout << *p;`

## Magyarázat

Előjeles egész túlcsoordulása, nullával osztás, null pointer által mutatott memória kiolvasása, ez mind `undefined behaviour`.

Az előjel nélküli egészek viszont definíció szerint modulo  $2^N$  aritmetikát valósítanak meg, ott a túlcsoordulás definiált, pl. ebben az esetben a változó értéke 0 lesz.

Mellékhatás nélküli végtelen ciklus szintén UB.

Legfontosabb tudnivalók

Kapcsolat

Versenyszabályzat

Adatvédelem

© 2022 Human Priority Kft.

KÉSZÍTETTE

Megjelenés

☀ Világos ↕