

C++: A TAVALYI GYŐZTES KATEGÓRIÁJA

1. forduló

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

Ismertető a feladathoz

Fontos információk:

A kérdésekre **mindig van helyes válasz!** Ha csak egy helyes válasz van az adott kérdésre, radio button-os választási lehetőségeket fogsz látni.

Kérjük, hogy a feladatok szövegeit **ne másold** és a böngésződ fejlesztő eszközét/ konzolját se nyisd meg feladatmegoldás közben! Mindkettő kizárást vonhat maga után.

Minden forduló után a **megoldások csütörtök reggel 8 órakor** lesznek elérhetőek.

A megoldásokkal kapcsolatos esetleges **észrevételeket a megoldások megjelenését követő kedd éjfélig** várjuk.

A több válaszlehetőségű feleletválasztós kérdéseknél járnak **részpontszámok, ha egyik rossz választ sem jelölöd be.**

Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat.

Minden feladatsornak van egy **becsült minimum megoldási ideje** (legalább a feladat elolvasási ideje). Aki ennél rövidebb idő alatt oldja meg, kizárható a versenyből.

Az első kategória után, amelynek a feladatlapját megoldod a fordulóban, kapni fogsz egy 2-3 perc alatt kitölthető **kérdőívet**. Az ezekből összeállított piackutatás legfontosabb eredményeit a díjátadót követően Veled is megosztjuk majd. Formáljuk közösen a piacot!

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

Felhasznált idő: 02:02/15:00

Elért pontszám: 0/14

1. feladat 0/2 pont

Mi a garantáltan helyes módja az alábbi kódban lefoglalt memória felszabadításának?

```
int *a = new int[5];
```

Válasz

- ☐ delete a;
- ☒ delete[] a;
- ☐ delete a[];
- ☐ free(a);
- ☐ Magától felszabadul, ha nincs használatban, mert C++20-ban már van garbage collector.

Magyarázat

Csak a "delete[] a;" helyes. A sima delete a tömbök felszabadítására nem jó. A free() pedig csak a malloc() és hasonló függvények által lefoglalt memóriát tudja felszabadítani.

Miért van külön delete és delete[]? Hatékonysági okokból: pl. a delete esetén biztosan csak 1 destruktort kell meghívni, a delete[]-nél viszont csak futási időben derül ki, hogy hányat, így szükség van egy külön for-ciklusra. C++-alapelv: "Ne fizess azért, amit nem használsz."

Irodalom:

https://en.cppreference.com/w/cpp/memory/new/operator_delete

<https://en.cppreference.com/w/c/memory/free>

2. feladat 0/3 pont

Melyik állítás igaz, ha az adott implementációban 1 byte == 8 bit?

Válaszok

- ☒ sizeof(char) < sizeof(int)
- ☒ sizeof(int) <= sizeof(long)
- ☒ sizeof(int) == sizeof(unsigned)
- ☐ sizeof(void*) == sizeof(void(*)())
- ☐ sizeof(size_t) == sizeof(void*)

Magyarázat

A szabvány garantálja, hogy `1 == sizeof(char)`, és az `int` legalább 16-bites. Emiatt (felhasználva az extra feltételt, hogy 1 byte 8 bitből áll, amit egyébként a szabvány nem követelne meg) `sizeof(char) < sizeof(int)`.

Mindig igaz, hogy a `long` legalább akkora, mint az `int` (és egyébként a `long` legalább 32-bites).

Az `unsigned` önmagában állva `unsigned int`-et jelent. Egy típus `signed` és `unsigned` változatai (pl. `int` és `unsigned`) mindig ugyanakkorák.

A `void(*)()` típus egy argumentum nélküli, `void` visszatérési értékű függvényre mutató pointer. Nem garantált, hogy egy adatpointer (`void*`) és egy függvénypointer (`void(*)()`) ugyanakkorák. Vannak ugyanis architektúrák, ahol az adatot és a kódot külön memóriában tárolják (pl. mikrokontrollerek).

`size_t` csak annyit garantál, hogy bármilyen objektum méretét képes tárolni. Nem biztos, hogy a `size_t` akkora, mint a `void*`, ugyanis nem biztos, hogy az adott platformon egyben le lehet foglalni az egész memóriát. (Pl. DOS szegmentált architektúra: egy objektum bele kell, hogy férjen 64KB-ba, de az egész RAM ennél jóval nagyobb is lehet.) A pointerok egész számmá való konvertálására az `intptr_t` és `uintptr_t` típusok valók.

3. feladat 0/3 pont

A C++ standard szerint melyik állítás igaz?

Válasz

- ☐ A `char` előjeles típus.
- ☐ Az `int` és az `int32_t` ugyanaz a típus.
- ☐ Az `int64_t` mindig definiálva van.
- ☒ Az alábbi definíció esetén `sizeof(A) > 0`:

```
struct A {};
```

- ☐ Az alábbi definíciók esetén `sizeof(B) == sizeof(A) + sizeof(int)`:

```
struct A {};  
struct B : A { int value; };
```

Magyarázat

A `char` lehet előjel nélkül is (pl. AVR mikrokontrollerek). Ugyanezen a platformon az `int` 16-bites, az `int32_t` pedig 32. Az `int`-ről csak annyi van kikötve, hogy legalább 16 bites előjeles egész típus legyen.

A fix bithosszúságú egész típusokat nem kötelező minden implementációnak támogatnia.

Minden objektum mérete pozitív, így az üres structé is, vagyis `sizeof(A) > 0`. Viszont ha egy objektum örököl egy másiktól, akkor az ősobjektum által ténylegesen elfoglalt bájtok számát a fordító 0-ra optimalizálhatja, ezért előfordulhat (és legtöbbször elő is fordul), hogy `sizeof(B) == sizeof(int) < sizeof(A) + sizeof(int)`.

4. feladat 0/3 pont

Adott az alábbi kódrészlet:

```
std::string a = "cica";  
std::string b = a;
```

Mely logikai állítások azok, amelyek mindenképpen igazak?

Válaszok

- ☒ a == b
- ☒ a.c_str() == b
- ☒ a == b.c_str()
- ☐ a.c_str() == b.c_str()

Magyarázat

Ha két `const char*`-ot hasonlítunk össze, csak akkor kapunk `true`-t, ha maguk a pointerok is megegyeznek. Ez két különböző string példány `c_str()` pointerre esetén nincs így.

`std::string` és `const char*` összehasonlításakor a `const char*` implicit konverzióval konvertálódik stringgé. Két `std::string` összehasonlítása pedig a tartalmat hasonlítja össze, így ezekben az esetekben a kifejezések értéke `true` lesz.

5. feladat 0/3 pont

Mit ír ki az alábbi C++20 program?

```
#include <cstdio>  
  
int main() {  
    printf("LOL??!");  
    return 0;  
}
```

Válasz

- ☐ LOL|
- ☒ LOL??!

- ☐ LOL?!
- ☐ A program szintaktikai hibás

Magyarázat

A C++ **trigráfok** olyan 3 karakterekből álló szekvenciák voltak, amiket a fordító egy adott karakterre cserélt ki. Régen egyes billentyűzeteken pl. nem volt | karakter, ennek a helyettesítésére találták ki a ??! trigráfot. A C++17 előtti szabványok szerint tehát LOL|-t kellett hogy kiírjon a program, de C++17-ban és azután már LOL?! a helyes válasz.

[Legfontosabb tudnivalók](#)

[Kapcsolat](#)

[Versenyszabályzat](#)

[Adatvédelem](#)

© 2022 Human Priority Kft.

KÉSZÍTETTE

Megjelenés

 Világos 