C++: A TAVALYI GYŐZTES KATEGÓRIÁJA

7. forduló

RENDELKEZÉSRE ÁLLÓ IDŐ:

10:00

Ismertető a feladathoz

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

Felhasznált idő: 01:57/10:00 **Elért pontszám:** 0/44

1. feladat 0/7 pont

Mit ír ki az alábbi program?

```
#include <iostream>

template <class T>
int f(T) {
  return 1;
}

int f(int) {
  return 2;
}

int f(...) {
  return 3;
}

int main() {
  std::cout << f(2021) << std::endl;
}
```

Válasz

- 1
- 2
- 3
- A program nem fordul le
- Undefined behaviour

Magyarázat

A program szintaktikailag helyes. Innen már nem nehéz a feladat, ugyanis általánosságban az az elv érvényesül, hogy a legjobban meghatározott argumentumtípusú függvény fog nyerni. A legrosszabb helyen az overload resolution szabályok szerint a "..." verzió áll, a nem-template verzió pedig üti a template verziót. Maguk a pontos szabályok már kicsit bonyolultak, nüansznyi különbségek esetén nehéz eldönteni, hogy melyik overload fog nyerni, így a fenti példa a való életben kerülendő:

https://en.cppreference.com/w/cpp/language/overload_resolution

2. feladat 0/10 pont

Milyen kóddal tér vissza az alábbi program?

```
namespace my_app {

class Application {
public:
  int getExitCode() const {
    return 1;
  }
};

int getExitCode(const Application &) {
  return 2;
}

} // namespace my_app

int getExitCode(const my_app::Application &) {
  return 3;
}

int main() {
  return getExitCode(my_app::Application{});
}
```

Válasz

- ○ 1
- ○ 2
- ○ 3
- ○ A program nem fordul le
- ○ Undefined behaviour

Magyarázat

Az ADL (*argument dependent lookup*) miatt a fordító megtalálja az osztállyal közös namespace-ben deklarált függvényt is, de a kívül deklaráltat is, tehát az egyértelműség hiánya miatt fordítási hibát kapunk.

https://en.cppreference.com/w/cpp/language/adl

3. feladat 0/10 pont

Adott az alábbi kód:

```
#include <iostream>

struct A {
  int aValue;
};

struct B {
  int bValue;
};

struct C: A, B {
  int cValue;
};

int main() {
  C c;
  c.bValue = 23;
  B& b = (*);
  std::cout << b.bValue << std::endl;
  return 0;
}
```

Mi kerüljön a (*) helyére, hogy a program outputja garantáltan 23 legyen, azaz szintaktikailag és szemantikailag is helyes kódot kapjunk?

Válaszok

- ☑ static_cast<B&>(c)

- [x] dynamic_cast<B&>(c)
- [ ] const_cast<B&>(c)
- [ ] reinterpret_cast<B&>(c)
- [x] (B&)c
- [x] c

Magyarázat

A példában a C osztálynak B egy publikus, egyértelmű, nem-virtuális ősosztálya. Ebben az egyszerű esetben a static_cast és a dynamic_cast éppúgy jól működik.

A const_cast nem jó, mivel az csak a const/volatile tulajdonságok módosítására szolgál.

A reinterpret_cast bár lefordul, de a kapott referencia használata undefined behaviour-t okoz. Miért? Mert alapszabályként a reinterpret_cast nem generál CPU-utasításokat. Jelen esetben pedig a többszörös öröklődés miatt a B ősobjektum címe eltér C címétől, azaz egy extra összeadás műveletre lenne szükség.

A C-stílusú konverzió jó, ld. az alábbi oldal (1)-es pontját: https://en.cppreference.com/w/cpp/language/explicit_cast. Igen, bár C-ben nincsenek referenciák, de referencia céltípusra is lehet C-stílusú konverziót használni.

Az implicit konverzió is jó.

4. feladat 0/10 pont

Adott az alábbi kód:

```
#include <atomic>
#include <iostream>
#include <thread>
#include <vector>

char parallelSum(const std::vector<char> &v1, const std::vector<char> &v2) {
  (*) sum{};

  std::thread t1([&v1, &sum] {
    for (const char x: v1) { sum += x; }
  });
  std::thread t2([&v2, &sum] {
    for (const char x: v2) { sum += x; }
  });

  t1.join();
  t2.join();

  return sum;
```

```
}

int main() {
  std::cout << static_cast<int>(parallelSum({1, 2, 3}, {4, 5, 6})) << std::endl;
  return 0;
}
```

Mi kerüljön a (*) helyére, hogy a program outputja garantáltan 21 legyen?

Válaszok

- [x] std::atomic<char>
- [ ] volatile char
- [ ] thread_local char
- [ ] char
- [ ] synchronized char
- [x] volatile std::atomic<char>

Magyarázat

A sima char nem jó. A szabvány szerint alapesetben undefined behavior, ha két szál egyike írja, másik meg olvassa ugyanazt a változót. Emiatt a fordító pl. feltételezheti, hogy csak az adott szál írja az adott változót, tehát "tudjuk", mi van benne, és kioptimalizálhatja a változó olvasgatását.

Volatile változó olvasása ezzel szemben mellékhatásnak számít, tehát nem optimalizálható ki. Gondolhatnánk, hogy ezzel meg is vagyunk, hiszen a char 1 bájtos, tehát nem kell tartanunk a részleges olvasástól/írástól sem. Hiába azonban minden, mert a szabvány szerint továbbra is UB, amit csinálunk. (Ennek gyakorlati oka van, pl. lehetséges, hogy egy architektúrán a különböző CPU magokhoz külön cache tartozik, és ezek nem feltétlenül lesznek szinkronban.)

A synchronized kulcsszó nem létezik (ötlet szintjén már igen, viszont máshogy kell majd használni: https://en.cppreference.com/w/cpp/language/transactional_memory).

A thread_local szemantikailag is hibás, ráadásul nem is fordul le, mert a lambda nem tudja elkapni (vajon melyik példányt kapná el? a főszál példányát? vagy egy újonnan inicializált példányt a t1 ill. t2 szálak számára?).

Az std::atomic<char> pontosan erre a célra való.

Persze a volatile std::atomic<char> is jó, de itt a volatile felesleges.

5. feladat 0/7 pont

Mit ír ki az alábbi program?

```
#include <iostream>

class A {
```

```
public:
  A() { std::cout << 'A'; }
  ~A() { std::cout << 'a'; }
};

class B {
public:
  B() { std::cout << 'B'; }
  ~B() { std::cout << 'b'; }
};

void f(int x) {
  if (x >= 0) {
    static A a;
    return;
  }

  static B b;
}

int main() {
  f(-1);
  f(-2);
  f(1);
}
```

Válasz

- ABba
- ABab
- BbBbAa
- BbAa
- BAba
- BAab

- Implementációtól függ
- Undefined behaviour

Magyarázat

Függvényen belüli static változó akkor inicializálódik, amikor a vezérlés először ér az adott sorra.

Ezek a static változók (a globális, valamint az osztályon belül definiált static változókkal együtt) csak a program végén, a main lefutása után semmisülnek meg.

A megsemmisülés sorrendje ellentétes a létrehozás sorrendjével.

https://en.cppreference.com/w/cpp/language/storage_duration#Static_local_variables

https://en.cppreference.com/w/cpp/utility/program/exit

KÉSZÍTETTE

Megjelenés

Világos