TÍPUSOK

•A C++ objektumainak mérete mindig a char méretének többszöröse, így a char mérete 1. Egy objektum méretét a sizeof operátorral kapjuk meg.

1≡sizeof(char)≤sizeof(short)≤sizeof(int)≤sizeof(long)
1≤sizeof(bool) ≤sizeof(long)
sizeof(char) ≤sizeof(wchar_t) ≤sizeof(long)
sizeof(float) ≤sizeof(double) ≤sizeof(long double)
sizeof(char)<=sizeof(int)
sizeof(bool)<=sizeof(long)
sizeof(N)≡sizeof(signed N)≡sizeof(unsigned N) (N lehet char, short int, int vagy long int)

-int: 0, 2012, -1 – decimális (10-es alapú) egész számok
0, 012 (10), 015 (13) – oktális (8-as alapú) számok, első jegye 0
0, 0x1 (1), 0xD (13) – hexadecimális, 0x ill. 0X előtag
0xFFFFu, 65535U – unsigned egész, u vagy U suffix
19871207L, 0x12f35e7l – long egész, l vagy L suffix
3087007744UL, 0xB8000000LU – unsigned long egész
-2., 100.45, 2e-3, 11E2, -3.14 – lebegőpontos konsans, előjeles valós számot reprezentál, egész részből, tizedes tört (. tizedespont) részből és kitevőből (10 hatványkitevője, e vagy E után) áll
3.1415F, 2.7182f – float lebegőpontos konstans
literál: 5e2f, 7e-3f – float
int: 'x'+'y' , 4/7,0xbeac
1011b,5f2e - nem szabályos
0xbalu – unsigned long
16ul/4ul – unsigned long
4e-1 = 0.4
„Hello” – const char[6]
„Hello”+2 – const char*
0x11 = 17

---

definíció:

- int a[10];
- class Foo { // ... };
- struct Foo { // ... };
- int i;
- static int i;
- const int l = 1;
- void* p;
- std::string msg = "Hello";

deklaráció:

- extern int i;
- struct Myclass;
- class Myclass;
- void f(int i);
- std::string a();
- int* p(int);

Mi a problemája a preprocesszor használatának? - Független a C++ nyelvtől, ezért nincs tekintettel a nyelvi szabályokra.

Melyik azonosító található az STL-ben? basic_string

Melyik operátort használjuk egy változó címének lekéréséhez? - operator&

tárolási osztályt specifikál: extern, static, auto,register

kötelező inicializálni az inicializáló listában : referenciák, konstansok

Melyik típusnak van push_front tagfüggvénye? – std::list

Az std::sort algoritmus melyik konténerrel használható? - std::vector

asszociatív konténer: set,map

szekvenciális konténer: deque,list,vector

tisztán virtuális függvény: *virtual void d()=0;*

extern int a; - Az a nevű változót egy másik fordítási egység definiálja, de az a.cpp-ből is szeretnénk használni.

Mitől válik egy osztály absztrakttá? - Van tisztán virtuális tagfüggvénye.

nem lehet template paraméter: Lebegőpontos konstans, Karakterlánc literál

Milyen típusú hibához vezet, ha egy olyan típussal példányosítunk egy template-et, amelyen hiányzik valamely elvárt művelet? - Fordítási hibát okoz.

Milyen típusú azonosítókat használnak a szabványos könyvtár implementálásához? - Aláhúzással kezdődő azonosítókat

Mi a paraméterdedukció - Az az eljárás, amikor a fordítóprogram levezeti a template paramétereket a függvényhívásból.

Mi alapján lehet egy osztályban megkülönböztetni a prefix- és postfix operator++-t? - A postfix operator++-nak van egy rejtett plusz int paramétere.

Mikor nevezünk erősen típusosnak egy nyelvet? - Erősen típusos, ha minden kifejezés és részkifejezés típusa fordítási időben meghatározott.

Mit nevezünk funktornak? - Azokat az objektumokat, amelyek van operator()-a.

Melyik memóriaallokáció helyes az alábbiak közül? char* a = new char[20];

Mit jelent a static kulcsszó az alábbi osztálydefinícióban?
struct S{ static int x; };
x osztályszintű adattag – minden példányban ugyanez az egy létezik

Projektünkben az összes fordítási egység lefordult, de nem jön létre a futtható állomány a build folyamat végén. Mi lehet a baj? - A linker nem talált meg egy statikus linkelésű library-t

Melyik paradigma alapján épül fel a C++ Standard Template Library? – generikus

Melyik azonosító szabályos a C++ szabályai szerint? _18, if, std

Melyik konstrukciót nem a C++11 vezette be? - funktor

Az std::string size metódusa mit ad vissza? - Hány karakter van a string-ben.

IGAZ ÁLLÍTÁSOK:

- A char típus előjelessége fordító-függő.
- A dinamikus változók a heap-en jönnek létre.
- Az automatikus változók a stack-en jönnek létre.
- A globális változók a statikus tárterületen jönnek létre.
- Az inicializáló listában végrehajtott utasítások a konstruktor törzsének végrehajtása előtt befejeződnek
- Egy konstans objektumnak csak a konstans tagfüggvényei hívhatóak meg.
- Az absztrakt osztályból nem lehet objektumot létrehozni.
- Polimorfikus objektumok esetén a virtuális tábla pointer definiálja, hogy melyik metódust hívjuk futási időben.
- Polimorfikus típusok objektumainak van egy rejtett virtuális tábla pointer tagja
- A bázisosztály konstruktorai nem öröklődnek a származtatott típusba.
- A virtuális függvények virtualitása megszüntethető az altípusokban.
- A dynamic_cast használatához polimorf osztályokra van szükség.
- Származtathatunk az std::string típusból.
- A C++ engedélyezi a többszörös öröklődést.
- A névtelen (anonim) névtérben definiált azonosítók nem láthatóak más fordítási egységekben.
- A typedef konstrukcióból nem lehet sablont (template-t) írni.
- Egy const int* típusú pointer mutathat változóra.
- Deklarációban egy plusz paraméterrel tudjuk megkülönböztetni a postfix operator++-t a prefix-től.
- A tömbök mindig konvertálódnak első elemre mutató pointerré.
- Egy friend függvény hozzáférhet az osztály private tagjaihoz.
- Paraméterdedukció csak függvények esetében használható.
- Lehet olyan programot írni C++-ban, amelyik fordítása közben algoritmusokat hajt végre.
- A szabványos C++ nem definiálja a long long típust.

helyes kódrészlet:

```
struct Foo {
        template <bool f> void bar() const { // ... }};
Foo f; f.bar<true>();
```

FUNKTOR:

```
struct X{ bool operator()(int a, int b) const{ return a > b;};
std::vector<int> v;
```

- helyes használat: std::multiset<int, X> s;

- funktortípus: std::less < double >
- Concatenate funktort az STL-ben hol használhatunk fel? - Az std::accumulate paramétereként.

POLIMORFIKUS:

```
struct Base{};
struct C : public Base
{…};
```

---

int i = 10;
const int j = 15;
const int *p = &j;

- p = &i;

Mi történik az alábbi függvényhíváskor?

```
template <typename T>
T max(const T& a, const T& b);
max<double>(6.72, 157);
```

- Mindkét paraméter double-lé konvertálódik

```
template < typename T >
struct List
{
  // ...
public:
  T& front();
  const T& front() const;
  // ...
};
```

- A fenti kód helyes, mert a front tagfüggvényt a this const-ságán túl lehet terhelni.

Mi a hiba az alábbi kódrészletben?

```
template< class T>
void  f(const T& t)
{  std::list::const_iterator i;
  // ...
}
```

- Nem jeleztük, hogy a const_iterator egy template paramétertől függő típusnév.