A megoldások között előfordulhatnak hibák! Ezekkel kapcsolatban a blackhawk1990@gmail.com címre várom a visszajelzéseket! Online elérhető a http://elte.bhawk.hu/Programozasi Nyelvek CPP/progcpp.htm oldalon.

1. Hány byte-on tárol a C++ egy karaktert (char)?

- ○ implementáció-függő
- ○ 4
- ◉ 1
- ○ 8

---

2. Melyik reláció **hamis** az alábbiak közül?

- ○ sizeof(float) <= sizeof(double)
- ○ sizeof(short) <= sizeof(int)
- ○ sizeof(unsigned char) == sizeof(char)
- ◉ sizeof(double) < sizeof(long double)

---

3. Melyik állítás igaz az alábbiak közül?

- ◉ A 4e-1 és a 0.4 konstansok értéke megegyezik.
- ○ A 4e-1f és a 4.1 konstansok típusa megegyezik.
- ○ A 4e-1f és a 4.1 konstansok értéke megegyezik.
- ○ A 4e2 és a 4.2L konstansok típusa megegyezik.

---

4. Melyik **nem** preprocesszor direktíva?

- ○ #else
- ○ #elif
- ○ #undef
- ◉ #while

---

5. Melyik definíció az alábbiak közül?

- ○ class MyClass;
- ◉ int a[10];
- ○ extern int i;
- ○ struct MyStruct;

---

6. Melyik igaz az alábbiak közül?

- ◉ Az automatikus változók a stack-en jönnek létre.
- ○ Az automatikus változók a statikus tárterületen jönnek lére.
- ○ Az automatikus változók a winchester-en jönnek létre.
- ○ Az automatikus változók a heap-en jönnek létre.

---

7. Mi a csillagozott sorban meghívott művelet neve?

```
class Foo
{
// ...
};

Foo f;
Foo g = f; // (*)
```

- destruktor
- default konstruktor
- értékadó operátor
- copy konstruktor

---

8. Az alábbi példában a Foo f(5); konstruktor hívása után mennyi lesz f.b értéke?

```
struct Foo
{
    int a, b;
    Foo(int c):a(c*2),b(c*3) {}
};
```

- 10
- nemdefiniált
- 0
- 15

---

9. Melyik típusnak van push_front tagfüggvénye?

- std::list
- std::set
- std::vector
- std::stack

---

10. Adott az alábbi x típus és f függvény. Az f(x) függvény hívásakor az x típus melyik műveletét hajtjuk végre a paraméter átadásához?

```
class X
{
// ...
};

void f(X a)
{
// ...
}

X x;
```

- Az x típus default konstruktorát.
- Az x típus értékadó operátorát.
- Nem hajtunk végre műveletet, mert x hivatkozás szerint adódik át.
- Az x típus copy konstruktorát.

11. Az alábbi kódban a csillagozott helyen mi a this-nek a típusa?

```
struct Foo
{
    void f()
    {
        // (*)
    }
};
```

- ○ const Foo*
- ◉ Foo*
- ○ Foo&
- ○ void*

---

12. Melyik állítás igaz az alábbiak közül?

- ○ Nem származtathatunk az std::string típusból, mert nincs virtuális destruktora.
- ◉ Származtathatunk az std::string típusból.
- ○ Nem származtathatunk az std::string típusból, mert az nem típus, hanem typedef.
- ○ Nem származtathatunk az std::string típusból, mert nincsenek protected adattagjai.

---

13. Mi **nem** lehet template paraméter az alábbiak közül?

- ○ Típus
- ○ Külső szerkesztésű függvény címe
- ○ Egész konstans
- ○ Karakterlánc iterál

---

14. Mennyi az értéke i-nek az alábbi kód végrehajtása után:

```
char ch = 255;
int i = ch;
```

- ◉ -1
- ○ 255
- ○ implementáció-függő
- ○ nem fordul le.

---

15. Melyik értékadás szabályos az alábbi kód után?

```
int i = 10;
const int j = 15;

const int *p = &j;
```

- ○ p = *j;
- ○ *p = i;
- ○ p *= i;

- [x] p = &i;

---

16. Hány byte-on tárol a C++ egy double-t?

- [x] implementáció-függő
- [ ] 4
- [ ] 8
- [ ] 6

---

17. Mi a típusa a "Hello" literálnak?

- [ ] const std::string
- [ ] char*
- [ ] char[5]
- [x] const char[6]

---

18. Melyik kulcsszó **nem** a tárolási osztályt specifikálja egy deklarációban ill. definícióban?

- [x] extern
- [ ] static
- [ ] auto
- [ ] int

---

19. Melyik igaz az alábbiak közül?

- [ ] A globális változók a heap-en jönnek lére.
- [x] A globális változók a statikus tárterületen jönnek létre.
- [ ] A globális változók a winchester-en jönnek létre.
- [ ] A globális változók a stack-en jönnek létre.

---

20. Melyik igaz az alábbiak közül?

```
struct X
{
    X(int i = 0) {}
};
```

- [ ] A fenti struct-nak nincs default konstruktora.
- [ ] A fenti struct-nak csak default konstruktora van.
- [ ] A fenti struct-nak nincs copy konstruktora.
- [x] A fenti struct-nak van default konstruktora.

---

21. Az alábbiak közül melyiket kötelező inicializálni az inicializáló listában?

- [ ] az összes adattagot

- ○ tömböket
- ○ semmit sem kötelező inicializálni
- ◉ a konstansokat

---

22. Melyik konténer asszociatív?

- ○ std::queue
- ◉ std::set
- ○ std::vector
- ○ std::list

---

23. Mi lesz az **a** változó értéke a függvényhívás után?

```
int a = 1, b =2;

void f(int& x, int& y)
{
    int t = x;
    x = y;
    y = t;
}

f(a,b);
```

- ○ nem definiált
- ◉ 2
- ○ semmi, fordítási hiba keletkezik
- ○ 1

---

24. Az alábbi függvény deklarációk alapján melyik tagfüggvény hívható meg **const Foo** objektumon?

```
struct Foo
{
    const int a(int i);
    int b(const int i);
    virtual int c(int i);
    int d(int i) const;
};

const Foo foo;
```

- ◉ foo.d(2);
- ○ foo.c(0);
- ○ foo.a(3);
- ○ foo.b(12);

---

25. Az alábbi típusok közül melyik poliformikus?

```
struct X;
struct A
{
    A(const X& b);
    A(int i);
};
```

```
struct B
{
    static int a;
};

struct Base{};

struct C : public Base
{

};

struct D
{
    virtual ~D();
};
```

- ○ A
- ○ B
- ○ C
- ○ D

---

## 26. Melyik kódrészlet helyes?

- ◉ struct Foo { template <bool f> void bar() const { // ... } }; Foo f; f.bar<true>();
- ○ template <int N> enum A { Elem = N };
- ○ template <typename T> typedef std::set<T, std::greater<T> > GreaterSet;
- ○ template <typename T = int> const T& max(const T& a, const T& b);

---

## 27. Mit jelent a static kulcsszó az alábbi osztálydefinícióban?

```
struct S
{
    static int x;
};
```

- ◉ S-ből nem lehet objektumot létrehozni
- ○ az x változót csak S tagfügvényei érhetik el
- ○ x osztályszintű adattag
- ○ semmit, struct kulcsszóval nem lehet osztályt definiálni

---

## 28. Melyik állítás igaz az alábbiak közül?

- ○ A sizeof(int) == sizeof(int* const) reláció mindig igaz.
- ○ Egy int* const típusú pointer mérete 8 byte.
- ◉ Egy int* const típusú pointer mutathat változóra.
- ○ Egy int* const típusú pointer nem változtathatja meg a mutatott értéket.

---

## 30. Mennyi a 012 konstans értéke?

- ○ 18
- ○ 0.12
- ○ 12

- [x] 10

---

31. Melyik nem definíció az alábbiak közül?

- [ ] struct Foo { // ... };
- [x] void f(int i);
- [ ] int i;
- [ ] class Foo { // ... };

---

32. Melyik deklarációra illeszkedik a csillaggal jelölt sorban meghívott művelet?

```
class Foo
{
// ...
};

Foo f;
Foo g = f; // (*)
```

- [ ] Foo& Foo::operator=(const Foo& rhs);
- [ ] Foo::Foo();
- [x] Foo::Foo(const Foo& rhs);
- [ ] void Foo::operator()();

---

34. Mi lesz az **a** változó értéke a függvényhívás után?

```
int a = 1;

void f(int& x, int& y)
{
    int t = x;
    x = y;
    y = t;
}

f(a,2);
```

- [ ] 2
- [ ] 1
- [x] semmi, fordítási hiba keletkezik
- [ ] nem definiált

---

35. Melyik állítás igaz egy **konstans** objektum esetében?

- [x] Az objektumnak csak a konstans taggfügvényei hívhatóak meg.
- [ ] Az objektumnak csak private adattagja lehet.
- [ ] Az objektumnak csak azok a tagfüggvényei hívhatóak meg, amelyek nem módosítják az adattagjait.
- [ ] Az objektum csak default konstruktorral hozható létre.

---

36. Mitől válik egy osztály absztrakttá?

- ○ Van virtuális destruktora.
- ◉ Van tisztán virtuális tagfüggvénye.
- ○ A tagfüggvényeinek csak a deklarációja ismert.
- ○ Van bázisosztálya.

---

## 37. Mi történik az alábbi függvényhíváskor?

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

max(4.3, 23);
```

- ○ Mindkét paraméter int-té konvertálódik
- ◉ Fordítási hiba keletkezik
- ○ Mindkét paraméter double-lé konvertálódik
- ○ Futás idejű hiba keletkezik

---

## 38. Mit nevezünk funktornak?

- ◉ Azokat az objektumokat, amelyeknek van operator()-a.
- ○ Implementáció függő.
- ○ Azokat az alprogramokat, amelyeknek nem void a visszatérési érték típusa.
- ○ Azokat az alprogramokat, amelyeknek void a visszatérési érték típusa.

---

## 39. Melyik igaz az alábbiak közül?

- ○ A friend kulcsszó több osztály logikai csoportosítására szolgál.
- ◉ Egy friend függvény hozzáférhet az osztály private tagjaihoz.
- ○ A friend kulcsszóval meghatározhatjuk a közelebbi osztályt többszörös öröklődés esetében.
- ○ Egy friend template osztály esetén példányosításkor nem kötelező explicit megadni a template paramétereket.

---

## 40. Hány byte-on tárol a C++ egy float-ot?

- ○ 6
- ○ 8
- ○ 4
- ◉ implementáció-függő

---

## 41. Melyik preprocesszor direktíva?

- ○ #undefine
- ○ #then
- ◉ #elif
- ○ #while

42. Melyik nem definíció az alábbiak közül?

- ○ const int l = 1;
- ○ static int i;
- ◉ extern int j;
- ○ int k;

---

43. Az X::f() függvényhívás során mit ír ki a program?

```
int i = 1;
namespace X
{
    int i = 2;

    void f()
    {
        int a = i + 1;
        int i = ::i - 1;
        std::cout << a << ", " << i << std::endl;
    }
}
```

- ◉ 3, 0
- ○ semmit, fordítási hiba keletkezik
- ○ 3, 2
- ○ 2, 1

---

44. Az alábbi példában a Foo(10); konstruktor hívása után mennyi lesz f.x értéke?

```
struct Foo
{
    int x, y;
    Foo(int i):y(i),x(y++) {}
};
```

- ○ 11
- ○ 10
- ◉ nemdefiniált
- ○ 0

---

46. Melyik állítás igaz az alábbiak közül?

- ○ A dynamic_cast fordítás idejű típuskonverziót végez.
- ○ A dynamic_cast használatához nem lehet statikus adattagja az osztálynak.
- ○ A dynamic_cast soha nem dob kivételt.
- ◉ A dynamic_cast használatához polimorf osztályokra van szükség.

---

47. Melyik állítás igaz az alábbiak közül?

- ○ Az alaptípsuok prefix operator++-nak void a visszatérési érték típusa.
- ◉ Deklarációban egy plusz paraméterrel tudjuk megkülönbzötetni a postfix operator++-t a prefix-től.

- ○ A postfix operator++ mindig hatékonyabb, mint a prefix
- ○ A postfix operator++ mindig a megnövelt értéket adja vissza.

---

48. Melyik állítás igaz az alábbiak közül?

- ○ Polimorf osztályok esetében az összes konstruktornak virtuálisnak kell lennie.
- ○ Nem lehet olyan osztályból származtatni, amelynek nincsen virtuális destruktora.
- ◉ A bázisosztály konstruktorai nem öröklődnek a származtatott típusba.
- ○ A konstruktor közül csak a copy konstruktor lehet virtuális, hogy felüldefiniálható legyen a másolás.

---

49. Mi **nem** lehet template paraméter az alábbiak közül?

- ○ Karakterlánc iterál
- ○ Típus
- ○ Egész konstans
- ○ Külső szerkesztésű függvény címe

---

50. Melyik állítás igaz az alábbiak közül?

- ◉ Paraméterdedukció csak függvények esetében használható.
- ○ Nem lehet származtatni typedef által meghatározott típusból.
- ○ A paraméterdedukció futási időben történik.
- ○ Az objektumok dinamikus típusát ismeri a fordítóprogram.

---

51. Mi a típusa a 5e2f literálnak?

- ◉ float
- ○ int
- ○ double
- ○ ez nem szabályos konstans

---

52. Mi a problémája a preprocesszor használatának?

- ○ A Java programozási nyelv nem támogatja, ezért nem tudjuk együtt használni a C++-t a Java-val.
- ○ A preprocesszor implementáció-specifikus.
- ○ Jelentősen növeli a futási időt.
- ◉ Független a C++ nyelvtől, ezért nincs tekintettel a nyelvi szabályokra.

---

53. Mennyi lesz foo.a értéke?

```
struct Foo
{
    int a;

    Foo(int i):Foo(i, 0)
    {
    }
```

```
    Foo(int i, int j):a(i)
    {
    }
};

Foo foo(4);
```

- (•) Fordítási hibát kapunk.
- ( ) 0
- ( ) Nemdefiniált
- ( ) 4

---

54. Hány byte-on tárol a C++ egy short int-et?

- ( ) 1
- ( ) 2
- ( ) 8
- (•) implementáció-függő

---

55. Melyik definíció az alábbiak közül?

- (•) void* p;
- ( ) struct X;
- ( ) int f();
- ( ) extern int i;

---

56. Milyen konstruktora(i) van(nak) az alábbi struct-nak?

```
struct X
{
    X(int) {...}
};
```

- ( ) csak copy konstruktora
- ( ) csak egy int paraméteres konstruktora
- ( ) csak default konstruktora
- (•) copy konstruktora és egy int paraméteres konstruktora

---

57. Mi **nem** lehet template paraméter az alábbiak közül?

- ( ) Lebegőpontos konstans
- ( ) Külső szerkesztésű objektum címe
- ( ) Logikai konstans
- ( ) Típus

---

58. Mi a típusa a 0xff konstansnak?

- ( ) double

- ○ char*
- ○ double*
- ◉ int

---

59. Melyik kulcsszó **nem** a tárolási osztályt specifikálja egy deklarációban ill. definícióban?

- ○ static
- ○ register
- ◉ public
- ○ auto

---

60. Melyik állítás igaz az alábbiak közül?

- ◉ Absztrakt osztályból nem lehet objektumot létrehozni.
- ○ Absztrakt osztálynak nem lehet adattagja.
- ○ Absztrakt osztályból nem lehet származtatni.
- ○ Absztrakt osztálynak nem lehet konstruktora.

---

61. Melyik vezet fordítási hibához az alábbi osztály template definíciók közül?

```
template <class T>
class A
{
};

template <struct T>
class B
{
};

template <typename T>
class C
{
};

template <int N>
class D
{
};
```

- ○ A
- ◉ B
- ○ C
- ○ D

---

62. Az alábbiak közül melyik függvényhívással lehet ekvivalens az alábbi (csillaggal jelölt) operátorhívás?

```
class Matrix
{
    // ...
};

Matrix a,b;
a + b; // (*)
```

- ○ operator+(a,b);
- ○ a.operator+(a,b);
- ○ Matrix.operator+(a,b);
- ◉ b.operator+(a);

---

63. Melyik reláció igaz az alábbiak közül?

- ○ sizeof(bool) < sizeof(char)
- ○ sizeof(unsigned char) < sizeof(char)
- ○ sizeof(int) <= sizeof(char)
- ◉ sizeof(float) <= sizeof(double)

---

64. Az alábbi függvény deklarációk alapján melyik tagfüggvény hívható meg **const Foo** objektumon?

```
struct Foo
{
    virtual void a(const int i);
    const int& b(const int i);
    void c() const;
    const Foo& d(const Foo& f);
};

Foo foo;
```

- ○ foo.b(12);
- ◉ foo.c();
- ○ foo.d(foo);
- ○ foo.a(3);

---

65. Mi a paraméterdedukció?

- ○ Az az eljárás, amikor referencia-szerinti paraméterátadásra cseréljük az érték-szerintit.
- ◉ Az az eljárás, amikor a fordítóprogram levezeti a template paramétereket a függvényhívásból.
- ○ Az az eljárás, amikor linker feloldja a külső függvényhívások paramétereit.
- ○ Az az eljárás, amikor default paraméterekkel látjuk el a függvény paramétereket.

---

66. Mennyi a 0x11 konstans értéke?

- ○ 11
- ○ 9
- ◉ 17
- ○ 3

---

67. Az alábbiak közül melyiket kötelező inicializálni az inicializáló listában?

- ○ az STL-es konténereket
- ◉ a referenciákat
- ○ a pointereket

- ○ az összes adattagot

---

68. Melyik konténer szekvenciális?

- ○ std::set
- ◉ std::deque
- ○ std::queue
- ○ std::map

---

69. Melyik állítás igaz az alábbiak közül?

- ○ A tömbök és a pointerek mindig ekvivalensek.
- ○ A tömbaritmetika több műveletet képes elvégezni, mint a pointeraritmetika.
- ○ A tömböket mindig void* pointer típusú paraméterként adjuk át a függvényeknek.
- ◉ A tömbök mindig konvertálódnak első elemre mutató pointerré.

---

70. Az X::f() függvényhívás során mit ír ki a program?

```
int i = 1;
namespace X
{
    int i = 2;

    void f()
    {
        int a = i;
        int i = a + X::i + ::i;
        std::cout << i << std::endl;
    }
}
```

- ○ 1
- ○ 4
- ◉ 5
- ○ semmit, fordítási hiba keletkezik

---

71. Az alábbiak közül melyik függvény tisztán virtuális?

```
struct Foo
{
    virtual void a();
    void b() const;
    static void c();
    virtual void d()=0;
};
```

- ○ a
- ○ b
- ○ c
- ◉ d

---

72. Mikor nevezünk erősen típusosnak egy nyelvet?

- ○ Erősen típusos, ha a fordítóprogram ellenőrzi, hogy definiált-e egy objektum vagy alprogram.
- ○ Erősen típusos, ha minden kifejezés és részkifejezés típusa futási időben meghatározott.
- ◉ Erősen típusos, ha minden kifejezés és részkifejezés típusa fordítási időben meghatározott.
- ○ Erősen típusos, ha a futási időben nem keletkezik kivétel.

---

73. Az std::sort algoritmus melyik konténerrel használható?

- ○ std::queue
- ○ std::list
- ○ std::set
- ◉ std::vector

---

74. Melyik állítás igaz az alábbiak közül?

- ○ A typedef konstrukcióból nem lehet sablont (template-t) írni.
- ○ Nem lehet sablon (template) tagfüggvénye egy nem-template osztálynak.
- ○ Az enum konstrukcióból lehet sablont (template-t) írni.
- ◉ A struct konstrukcióból nem lehet sablont (template-t) írni.

---

75. Melyik állítás igaz az alábbiak közül?

- ◉ A C++ engedélyezi a többszörös öröklődést.
- ○ Nem lehet alkalmazni a többszörös öröklődést, ha azonosító ütközés lépne fel.
- ○ Csak akkor használható a többszörös öröklődés, ha az összes bázisosztálynak van virtuális destruktora.
- ○ A C++ tiltja a többszörös öröklődést.

---

76. Milyen konstruktorok hívhatóak az alábbi struct esetében?

```
struct X
{
};
```

- ○ csak copy konstruktor
- ○ nincsen konstruktora
- ◉ copy és default konstruktor
- ○ csak default konstruktor

---

77. Melyik állítás igaz az alábbiak közül?

- ○ A sizeof(int) == sizeof(const int*) reláció mindig igaz.
- ◉ Egy const int* típusú pointer mutathat változóra.
- ○ Egy const int* típusú pointer mérete 4 byte.
- ○ Egy const int* típusú pointer megváltoztathatja a mutatott értéket.

---

78. Mi a típusa az 5f2e konstansnak?

- ○ int
- ◉ ez nem szabályos konstans
- ○ float
- ○ double

---

79. A C++ kódokban lévő makrókat melyik egység dolgozza fel az alábbiak közül?

- ◉ preprocessor
- ○ A szabványos C++-ban nem is írhatunk makrókat (csak C-ben)
- ○ assembler
- ○ linker

---

80. Adott egy típus, melynek mérete nem egyezik meg a típus adattagjai méretéinek összegével. Mi történhetett?

- ○ Megörököltük annak az osztálynak a tagjait is, amelyik minden C++ osztálynak az őse.
- ○ Megfeledkeztünk a header guard-okról és több helyre is be include-oltuk a header filet.
- ○ Találtunk egy bugot a fordítóprogramban.
- ○ A fordítóprogram szóhatárra optimalizálta az adattago(ka)t

---

81. Definiálhatunk-e egy C++ függvény legbelső blokkjában két azonos nevű változót?

- ○ Igen, definiálhatunk.
- ◉ Nem.
- ○ Ezt csak a g++ fordítóprogram támogatja.
- ○ Csa akkor, ha különböző a típusuk.

---

82. Adott egy típus, melynek mérete nem egyezik meg a típus adattagjai méretéinek összegével. Mi történhetett?

- ○ A this pointer miatt nagyobb az osztály mérete.
- ○ Megörököltük annak az osztálynak a tagjait is, amelyik minden C++ osztálynak az őse.
- ○ Az osztályunknak van virtuális függvénye, így létrejött a virtuális tábla pointer.
- ○ Találtunk egy bugot a fordítóprogramban.

---

83. Az std::sort algoritmus melyik konténerrel használható?

- ○ std::set
- ○ std::list
- ○ std::auto_prt
- ◉ std::deque

---

84. Melyik állítás igaz az alábbiak közül?

- ○ A long long típust 8 byte-on ábrázolja a C++.
- ○ A sizeof(long int) <= sizeof(long long) reláció mindig igaz.
- ◉ A szabványos C++ nem definiálja a long long típust.
- ○ A sizeof(long double) == sizeof(long long) reláció mindig igaz.

---

85. Melyik azonosító szabályos a C++ szabályai szerint?

- ○ 101_kiskutya
- ◉ _1
- ○ miez?
- ○ jo!

---

86. Melyik igaz az alábbiak közül?

```
template
class Foo;

int i;

template
void f(const T& t)
{
    typename Foo::N * i;
    // ...
}
```

- ○ A fordítóprogram a fenti kódot úgy elemzi tovább, hogy a függvény sablon első sorában egy i nevű pointerrel elfedtük a globális int i-t.
- ○ A fordítóprogram a fenti kódot úgy elemzi tovább, hogy végeztünk egy szorzást a függvény sablon első sorában.
- ○ Nem fedhetjük el a külső i azonosítót, ezért a fenti kód fordításakor hiba keletkezik.
- ○ A fordítóprogratól függ, hogy a fenti kódban szorzást végzünk vagy egy pointert hozunk létre.

---

87. Melyik konténer asszociatív?

- ○ std::hashmap
- ◉ std::set
- ○ std::vector
- ○ std::list

---

88. Melyik nyelvi konstrukció támogatja párhuzamos programok írását C++-ban?

- ○ polimorfizmus
- ○ template
- ◉ Nincs olyan nyelvi konstrukció, ami támogatja párhuzamos programok írását.
- ○ protected

---

89. Melyik azonosító szabályos a C++ szabályai szerint?

- ○ std::stack
- ◉ vector
- ○ t[i]
- ○ ~dtor

---

90. Mit ír ki a képernyőre az alábbi kódrészlet?

```
template
const T& max(const T& a, const T& b)
{
    return a > b ? a : b;
}

std::cout << max("abc", "sef");
```

- ○ abc
- ○ sef
- ○ Nemdefiniált az eredménye
- ◉ Fordítási hiba keletkezik.

---

91. Melyik állítás igaz az alábbiak közül?

- ○ Nem lehet olyan programot írni C++-ban, amelyik adatbázisszerverhez kapcsolódna.
- ◉ Lehet olyan programot írni C++-ban, amelyik fordítása közben algoritmusokat hajt végre.
- ○ Nem lehet párhuzamos programot írni C++-ban.
- ○ Lehet olyan programot írni C++-ban, amelyik fordítás nélkül is futhat.

---

92. Mennyi a 018 konstans értéke?

- ◉ Nincs ilyen konstans
- ○ 0.18
- ○ 24
- ○ 18

---

93. Melyik paradigma alapján épül fel a C++ Standard Template Library?

- ○ funkcionális
- ◉ generikus
- ○ objektum-orientált
- ○ iterator

---

94. Projektünkben az összes fordítási egység lefordult, de nem jön létre a futtatható állomány a build folyamat végén. Mi lehet a baj?

- ○ A build folyamat közben nem találtuk meg a preprocessor-t.
- ○ A linker nem talált meg egy dinamikus linkelésű library-t.
- ◉ A linker nem talált meg egy statikus linkelésű library-t.

- A virtuális destruktorok hiánya okozta.

---