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

implementáció-függő (viszont lásd 79. megjegyzés)
1
8
4

2) Hány byte-on tárol a C++ egy float-ot?

implementáció-függő
6
4
8

3) Hány byte-on tárol a C++ egy short int-et?

implementáció-függő
1
2
8

4) Hány byte-on tárol a C++ egy int-et?

implementáció-függő
1
4
8

5) Hány byte-on tárol a C++ egy double-t?

implementáció-függő
4
6
8

6) Melyik reláció hamis az alábbiak közül?

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

7) Melyik reláció hamis az alábbiak közül?

sizeof(short)<=sizeof(long int)
sizeof(float)<=sizeof(long double)
sizeof(char) == sizeof(signed char)
sizeof(bool) == sizeof(char)

8) Melyik reláció igaz az alábbiak közül?

sizeof(int) <= sizeof(char)
sizeof(unsigned char) < sizeof(char)
sizeof(bool) < sizeof(char)
sizeof(float) <= sizeof(double)

9) Melyik reláció igaz az alábbiak közül?

sizeof(char) <= sizeof(double)
sizeof(signed char) < sizeof(char)
sizeof(unsigned char) < sizeof(char)
sizeof(int) < sizeof(char)

10) Mi a típusa a

0xff

konstansnak?

double
char*
double*
int

11) Mi a típusa a

5e2f

literálnak?

float
ez nem szabályos konstans
double
int

12) 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 4e2 és a 4.2L konstansok típusa megegyezik.
A 4e-1f és a 4.1 konstansok értéke megegyezik.

13) Mennyi a 012 konstans értéke?

10
12
18
0.12

14) Mi a típusa a

"Hello"

literálnak?

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

15) Mennyi a 0x11 konstans értéke?

3
9
17
11

16) Melyik nem preprocesszor direktíva?

#undef
#elif
#else
#while

17) Melyik preprocesszor direktíva?

#undefine
#then

#while
#elif

18) Mi a problemája a preprocesszor használatának?

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

19) Melyik nem preprocesszor direktíva?

#else
#elseif
#define
#elif

20) Melyik kulcsszó nem a tárolási osztályt specifikálja egy deklarációban ill. definícióban?

static
register
public
auto

21) Melyik definíció az alábbiak közül?

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

22) Melyik nem definíció az alábbiak közül?

class Foo { // ... };
struct Foo { // ... };
void f(int i);
int i;

23) Melyik definíció az alábbiak közül?

void* p;
struct X;
int f();

extern int i;

24) Melyik nem definíció az alábbiak közül?

int k;
const int l = 1;
static int i;
extern int j;

25) Melyik igaz az alábbiak közül?

A dinamikus változók a heap-en jönnek létre.
A dinamikus változók a statikus tárterületen jönnek létre.
A dinamikus változók a winchester-en jönnek létre.
A dinamikus változók a stack-en jönnek létre.

26) 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;
   }
}
```

semmit, fordítási hiba keletkezik
2, 1
3, 0
3, 2

27) 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;
   }
}
```

semmit, fordítási hiba keletkezik
4
1
5

28) 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.
Nemdefiniált
0
4

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

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

csak egy int paraméteres konstruktora
copy konstruktora es egy int paraméteres konstruktora
csak copy konstruktora
csak default konstruktora

30) Melyik igaz az alábbiak közül?

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

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

31) Mi a csilagozott sorban meghívott művelet neve?

```
class Foo
{
...
};
Foo f;
Foo g = f; // (*)
```

destruktor
default konstruktor
copy konstruktor
értékadó operátor

32) Milyen konstruktorok hívhatóak az alábbi struct esetében?

```
struct X
{
};
```

copy és default konstruktor
nincsen konstruktora
csak default konstruktor
csak copy konstruktor

33) Melyik deklarációra illeszkedik a csillaggal jelölt sorban meghívott művelet?

```
class Foo
{
...
};
Foo f;
Foo g = f; // (*)
```

Foo::Foo(const Foo& rhs);
Foo::Foo();
Foo& Foo::operator=(const Foo& rhs);
void Foo::operator()();

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

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

0
10
11
Nemdefiniált

38) 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) {}
};
```

Nemdefiniált
5
10
15

39) Melyik konténer asszociatív?

std::queue
std::set
std::vector
std::list

40) Melyik típusnak van push_front tagfüggvénye?

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

41) Melyik konténer szekvenciális?

std::deque
std::set
std::queue
std::map

42) Az alábbiak közül melyiket kötelező inicializálni az inicializáló listában?

az összes adattagot
a referenciákat
az STL-es konténereket
a pointereket

43) Az alábbiak közül melyiket kötelező inicializálni az inicializáló listában?

tömböket
az összes adattagot
semmit sem kötelező inicializálni

a konstansokat

44) Az std::sort algoritmus melyik konténerrel használható?

std::queue
std::vector
std::list
std::set

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

46) 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
1
2

semmi, fordítási hiba keletkezik

47) 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);
```

semmi, fordítási hiba keletkezik
1
2
nem definiált

51) Melyik állítás igaz egy konstans objektum esetében?

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

52) Az alábbi kódban a csillagozott helyen mi this-nek a típusa?

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

Foo*
void*
const Foo*
Foo&

53) 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.c()
foo.b(12);
foo.a(3);
foo.d(foo);

54) Melyik állítás igaz az alábbiak közül?

Az absztrakt osztályból nem lehet objektumot létrehozni.
Az absztrakt osztálynak nem lehet adattagja.
Az absztrakt osztályból nem lehet leszármaztatni.
Az absztrakt osztálynak nem lehet konstruktora.

55) 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

56) Melyik állítás igaz az alábbiak közül?

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

57) Melyik állítás igaz az alábbiak közül?

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

58) Az alábbi típusok közül melyik polimorfikus?

```
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

59) 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.

60) Mitől válik egy osztály absztrakttá?

Van virtuális tagfüggvénye
Van tisztán virtuális tagfüggvénye.
Nincsen adattagja.
Nincsen default konstruktora.

61) Melyik állítás igaz az alábbiak közül?

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.
A C++ engedélyezi a többszörös öröklődést.

62) 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

63) 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

64) Mi nem lehet template paraméter az alábbiak közül?

Lebegőpontos konstans
Típus
Logikai konstans
Külső szerkesztésű objektum címe

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.

68) Melyik állítás igaz az alábbiak közül?

A struct konstrukcióból nem lehet sablont (template-t) írni.
Az enum konstrukcióból lehet sablont (template-t) írni.
Nem lehet sablon (template) tagfüggvénye egy nem-template osztálynak.
A typedef konstrukcióból nem lehet sablont (template-t) írni.
Megjegyzés: A typedef egy alias név adásnak felel meg és nem egy adat konstrukció.

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

```
class Matrix
{
  // ...
```

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

a.operator+(a,b);
operator+(a,b);
Matrix.operator+(a,b);
b.operator+(a);

70) Melyik állítás igaz az alábbiak közül?

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

71) Melyik állítás igaz az alábbiak közül?

A postfix operator++ mindig hatékonyabb, mint a prefix.
Az alaptípusok prefix operator++-nak void a visszatérési érték típusa.
Deklarációban egy plusz paraméterrel tudjuk megkülönböztetni a postfix operator++-t a prefix-től.
A postfix operator++ mindig a megnövelt értéket adja vissza.

72) 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 nem változtathatja meg a mutatott értéket.

Egy
int* const
típusú pointer mutathat változóra.

73) Mikor nevezünk erősen típusosnak egy nyelvet?

Erősen típusos, ha a forditóprogram 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.

74) Melyik értékadás szabályos az alábbi kód után?

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

p *= i;
*p = i;
p = &i;
p = *j;

75) Mit nevezünk funktornak?

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

76) Mit jelent a static kulcsszó az alábbi osztálydefinícióban?

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

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

77) Melyik állítás igaz az alábbiak közül?

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

78) Melyik igaz az alábbiak közül?

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

79) Mennyi az értéke i-nek az alábbi kód végrehajtása után:

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

-1

nem fordul le.

implementáció-függő

255

80) Melyik állítás igaz az alábbiak közül?

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

81) 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);

82) Melyik igaz az alábbiak közül?

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

83) Melyik igaz az alábbiak közül?

A globális változók a stack-en jönnek létre.
A globális változók a statikus tárterületen jönnek létre.
A globális változók a heap-en jönnek létre.
A globális változók a winchester-en jönnek létre.

84) Projektünkben az összes fordítási egység lefordult, de nem jön létre a futtható 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.

85) Lehet-e egy C++ függvényben két azonos nevű változó?

Nem lehet.
Csak akkor, ha különböző blokkban definiálták.
Csak akkor, ha különböző a típusuk.
Csak akkor, ha a láthatóságuk nem esik egybe.

86) Melyik paradigma alapján épül fel a C++ Standard Template Library?

funkcionális
generikus
objektum-orientált
iterator

87) Melyik igaz az alábbiak közül?

```
template
class Foo;
int i;
template
void f(const T& t)
```

```
}
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 fenti kód nem fordul le, mert nem írtuk ki a typename kulcsszót.
A fordítóprogram a fenti kódot úgy elemzi tovább, hogy végeztünk egy szorzást a függvény sablon első sorában.
A fordítóprogramtól függ, hogy a fenti kódban szorzást végzünk vagy egy pointert hozunk létre.

88) Mennyi a 018 konstans értéke?

Nincs ilyen konstans
0.18
24
18

89) Mi a típusa a 5f2e konstansnak?

double
ez nem szabályos konstans
int
float

90) Melyik azonosító szabályos a C++ szabályai szerint?

_18
ures-e
!b
1001_ejszaka

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

Igen, definiálhatunk.
Nem.

Ezt csak a g++ fordítóprogram támogatja.
Csak akkor, ha különböző a típusuk.

93) Adott egy tipus, melynek mérete nem egyezik meg a típus adattagjai méretének 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

(a gyorsabb címszámítás érdekében MINDIG szóhatárra igazít kívéve ha kikapcsoljuk az adott blokkra: ezt a #pragma pack(push, x) és #pragma pack(pop) parancsokkal szabályozhatjuk)

94) 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