PB161: PB161 vnitro 14:00

Jméno a příjmení – pište do okénka	*UČO*	*Číslo zadání*
		1

Za otázku se všemi správně odpovězenými možnostmi jsou 2 body. Za část správných odpovědí je poměrný počet bodů. Za každou špatnou odpověď je -1 bod. Otázka může mít více správných odpovědí. Není povoleno používat dodatečné materiály.

1

```
#include <iostream>

class A {
public:
    _S1 void foo1() {std::cout << "1";}
    _S2 void foo2() {std::cout << "2";}
    _S3 ~A() {std::cout << "3";}
};

class B : public A {
public:
    _S4 void foo1() {std::cout << "4";}
    _S5 void foo2() {std::cout << "5";}
    _S6 ~B() {std::cout << "6";}
};

int main() {
    A* object1 = new B;
    B* object2 = new B;

    object1->foo1();
    object1->foo2();
    object2->foo1();
    object2->foo2();

    delete object1;
    delete object2;

    return 0;
}
```

Vyberte korektní možnosti hodnot specifikátorů _S u metod tak, aby došlo k vypsání textu "45456363"

A _S1=, _S2=virtual, _S3=virtual, _S4=virtual, _S5=, _S6=

B _S1=virtual, _S2=virtual, _S3=virtual, _S4=virtual, _S5=, _S6=

C _S1=virtual, _S2=, _S3=virtual, _S4=, _S5=, _S6=virtual

D _S1=, _S2=virtual, _S3=virtual, _S4=, _S5=, _S6=virtual

E _S1=virtual, _S2=, _S3=virtual, _S4=virtual, _S5=, _S6=

F Nelze doplnit tak, aby došlo k vypsání požadovaného výstupu.

2

```
#include <iostream>
#include <list>
class A {
public: void foo() {std::cout << "A"; }
};
int main() {
  std::list<A*> seznam;
  std::list<A*>::iterator i;
  seznam.push_back(new A);
  seznam.push_back(new A);
  seznam.push_back(new A);
     for (i= seznam.begin(); i!= seznam.end(); i++) {
      i->foo();
  }
  return 0;
}
```

Kolega vám ukazuje uvedený kód s tím, že jej nelze přeložit s chybou error: request for member 'foo' in '* iter.std::_List_iterator<_Tp>::operator-> [with _Tp = A*]()', which is of non-class type 'A*'

Jak mu odpovíte?

A Uvedená chyba nemohla být překladačem vypsána.

B Chybná deklarace kontejneru seznam. Pokud by byl kontejner deklarován jako std::list<A> (jediná změna), tak by bylo možné program bez chyb zkompilovat.

C Chybně deklarovaný iterátor i. Pokud by byl iterator deklarován jako std::list<A>::iterator (jediná změna), tak by bylo možné program bez chyb zkompilovat.

D Chybně vkládané prvky do kontejneru seznam. Pokud by byly prvky vkládány jako seznam.push_back(new A()); (jediná změna), tak by bylo možné program bez chyb zkompilovat

E Chybně vkládané prvky do kontejneru seznam. Pokud by byly prvky vkládány jako seznam.push_back(A); (jediná změna), tak by bylo možné program bez chyb zkompilovat.

F Využití iterátoru bez derefence.

3

```
class A {
protected:
  void release() {}
public:
  virtual ~A() { release(); }
};

class B : public A {
  int* m_array;
public:
  B(int length) {m_array = new int[length];}
  void release() {
    if (m_array)delete[] m_array;
  }
};

int main() {
    A* object1 = new B(10);
    B* object2 = new B(10);

    delete object1;
    delete object2;

    return 0;
}
```

Která z uvedených tvrzení jsou pravdivá?

A Uvedený kód způsobí memory leak dvou polí, každé z nich alokované jako new int[10].
B Pokud bychom deklarovali metodu release() třídy B jako virtuální, tak by uvedený kód nezpůsobil žádný memory leak.
C Uvedený kód způsobí memory leak jednoho pole o velikosti new int[10].
D Uvedený kód nelze přeložit, neboť třída A nenabízí veřejný konstruktor.
E Uvedený kód způsobí memory leak právě jednoho pole o velikosti 10 bajtů.
F Uvedený kód lze přeložit a nezpůsobí žádný memory leak.

4

```
int main() {
    B* obj1 = new A;
    B* obj2 = new C;
    C* obj3 = new A;
    return 0;
}
```

Který z uvedených vztahů dědičnosti mezi třídami A, B, C platí v případě, že uvedený kód lze zkompilovat?

A C není potomek B, A není potomek C, A není potomek B
B pro uvedený kód není možné takové vztahy najít
C A je potomek B, A je potomek C, B není potomek C
D B je potomek A, C je potomek A, B je potomek C
E žádná z ostatních odpovědí není správná

5

```
#include <iostream>
#include <list>
class A {
public:
    A() {  std::cout<<"A"; }
    ~A() {  std::cout<<"~A"; }
};
int main() {
    std::list<A*> seznam;
    for (int i = 0; i < 3; i++) {
       seznam.push_back(new A);
    }
    seznam.clear();
    seznam.clear();
    return 0;
}
```

Pro uvedený kód platí:

A vypíše 'AAA~A~A~A'
B všechna dynamicky alokovaná paměť je korektně uvolněna
C vypíše 'AAA'
D vypíše 'AAA~A~A~A~A~A~A'
E žádná z ostatních možností není správná
F dynamicky alokovaná pamět není korektně uvolněna

6

```
#include <iostream>
void print() { std::cout<< "x"; }
namespace MyNamespace {
  void print() {std::cout<< "y";}
}
namespace MyNamespace2 {
  void print() {std::cout<< "z";}
}
using namespace MyNamespace;
int main() {
    print();
    return 0;
}
```

Pro uvedený kód platí:

A vypíše 'yz'
B nelze přeložit
C vypíše 'y'
D vypíše 'z'
E vypíše 'x'
F žádná z ostatních možností není správná

7

```
#include <iostream>
class A {
public:
    A(int value) : m_value(value) {}
    void set(int value) { m_value = value; }
    int get() const { return m_value; }
private:
    int m_value;
};

void foo(A v1, A& v2, A* v3) {
    v1.set(v2.get());
    v2.set(v3->get());
    v3->set(v1.get());
}
int main() {
    A x(1);
    A y(2);
    A* z = new A(3);

    foo(x, y, z);
    foo(x, y, z);
    std::cout<<x.get()<<" "<<y.get();
    std::cout<<" "<<z->get();
    delete z;
    return 0;
}
```

Pro uvedený kód platí:

A vypíše '1 2 3'
B vypíše '2 2 2'
C žádná z ostatních možností není správná
D vypíše '3 3 3'
E vypíše '1 2 2'
F vypíše '3 2 3'

8

```
class A {
_PRAVO1_
  A(int value) : m_value(value) {}
public:
  virtual int GetValue() const = 0;
  virtual void SetValue(int value) {
    m_value = value;
  }
_PRAVO2_
  int m_value;
};

class B : _PRAVO3_ A {
_PRAVO4_
    B(int value) : A(value) {}
    int GetValue() const { return m_value; }
};

int main() {
  B test(10);
  test.SetValue(11);
  int value = test.GetValue();
  return 0;
}
```

Doplňte správné hodnoty práv namísto označení _PRAVO1_, _PRAVO2_, _PRAVO3_ a _PRAVO4_ tak, aby bylo možné kód zkompilovat a zároveň dodržoval pravidla zapouzdření.

A _PRAVO1_ = public:, _PRAVO2_ = private:, _PRAVO3_ = public, _PRAVO4_ = public:
B _PRAVO1_ = private:, _PRAVO2_ = protected:, _PRAVO3_ = public, _PRAVO4_ = public:
C _PRAVO1_ = protected:, _PRAVO2_ = private:, _PRAVO3_ = protected, _PRAVO4_ = protected:
D _PRAVO1_ = private:, _PRAVO2_ = private:, _PRAVO3_ = private, _PRAVO4_ = private:
E _PRAVO1_ = public:, _PRAVO2_ = private:, _PRAVO3_ = protected, _PRAVO4_ = public:
F _PRAVO1_ = protected:, _PRAVO2_ = protected:, _PRAVO3_ = public, _PRAVO4_ = public:

---

9 Která z uvedených tvrzení jsou pro jazyk C++ pravdivá?

A Metoda deklarovaná s klíčovým slovem const může vždy měnit obsah vnitřních atributů třídy
B Pokus o změnu parametru předávaného konstantní referencí upozorní překladač už v době překladu
C Metoda deklarovaná s klíčovým slovem const může být volána včasnou i pozdní vazbou (podle toho, jak je deklarována)
D Proměnná typu konstantní reference může být měněna pouze v metodě deklarované s klíčovým slovem const
E Metoda deklarovaná s klíčovým slovem const musí mít všechny parametry předávané konstantní referencí

10 Která z uvedených tvrzení jsou korektní?

A STL kontejner map poskytuje rychlejší přístup k danému prvku než kontejner vector, pokud známe index daného prvku.

B STL kontejner vector má menší paměťové nároky pro uložení stejného množství prvků než kontejner list.

C STL kontejner list poskytuje rychlejší přístup k hodnotě prvku, pokud známe index daného prvku, než vector.

D STL kontejner list se typicky používá pro datové struktury, které vyžadují rychlé vkládání nebo ubírání prvků ve středu kontejneru.

E STL kontejner vector se typicky používá pro datové struktury, které vyžadují rychlé vkládání nebo ubírání prvků ve středu kontejneru.