

o. otazka:

co je to singleton?

odpoved:

trida, která ma pouze jednu instanci

1. otazka:

Predpokladejme tyto deklarace :

```
class T
{
public:
    virtual void f();
}
```

```
class U: public T
{
public:
    virtual void f();
}
```

```
U x;
T y=x;
```

Ktera funkce se pouzije, pokud napisu:

```
y.f()
```

odpoved:

```
T::f()
```

(y je instance (ani pointer, ani reference) T, tzn. uz pri prekladu se urci, ze se vola T::f(). sice se zkopiruji polozky potomka do predka, ale VMT se nekopiruje)

2. otazka:

Predpokladejme tyto deklarace:

```
class Base
{
public:
    virtual Base operator+ (const Base &) const;
}

class Complex;

class Real:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Real operator+ (const Real &) const;
    virtual Complex operator+ (const Complex &) const;
}

class Complex:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Real operator+ (const Real &) const;
    virtual Complex operator+ (const Complex &) const;
}

Real r;
Complex c;
Base &b = r;

b = b + c;
```

Ktery operator se zavola?

- [A] Real::operator+ (const Base &) const
- [B] Complex::operator+ (const Complex &) const
- [C] Complex::operator+ (const Base &) const
- [D] Complex::operator+ (const Real &) const

odpoved:

A

(protoze pred b je v deklaraci &, je odkaz na cast objektu typu real, proto se pouzije funkce definovana ve tride real.)

3. otazka:

Kdy se vola konstruktor prvku dynamicky alokovaneho pole?

- [A] tesne pred prvnim pristupem k prvku
- [B] na prvky pole sa nevolaji konstruktory
- [C] prvky pole nesmi mit konstruktory
- [D] automaticky po naalokovani pole

odpoved:

D

4. otázka:

```
namespace T
{
    double f(char x);
};

namespace U
{
    char f(double d);
};

int f(int i);
using namespace T;
using namespace U;
```

```
char x = f('A');
```

Ktera funkce se zavola?

[A] T::f(char)

[B] U::f(int)

[C] nelegalni volani

[D] U::f(double)

odpoved:

A

(pri pretezovani funkci se vybere ta funkce, pro kterou je konverze nejlevnejsi. tedy je odpoved zrejma - pro char ani nemusime konverzi delat.)

5. otázka:

```
class T
{
public:
    T();
    T(const T &);
    T(int);
}
```

```
void f(const T & p) {}
```

Kterym konstruktorem se inicializuje p?

[A] T::T()

[B] T::T(const T &)

[C] zadnym

[D] T::T(int)

odpoved:

C

(predavame odkaz - zadny objekt nevznika ani se nekopiruje)

podotazka:

Ktera metoda neni vhodna pro tridu T?

[A] T & operator= (T &) const

[B] T()

[C] T & operator= (const T &)

[D] T(const T &)

pododpoved:

A

(operator = by nemel byt konstantni, aby mohl prirazovat hodnotu)

6. otázka:

```
template struct T
{
    enum { v = 1 };
}
```

```
template struct T
{
    enum { v = 2 };
}
```

```
template struct T
{
    enum { v = 3 };
}
```

```
char a[ T::v];
```

Jakou velikost bude mi pole a?

[A] není definováno

[B] 2

[C] 3

[D] 1

odpověď:

C

(jestliže alespoň jeden typ je přesně určen, má tato definice přednost)

7. otázka:

```
class SStk
{
public:
    /* ... */
    void push(const std::string & s);
private:
    std::string * p_;
    int allocated_, used_;
};

void SStk::push (const std::string & s)
{
[1]  if (used_ >= allocated)
[2]  {
[3]      int n2 = (allocated_ + 1) * 2;
[4]      std::string * p2 = new string[n2];
[5]      for(int i = 0; i < used_; ++i)
[6]          p2[i] = p_[i];
[7]      delete[] p_;
[8]      p_ = p2;
[9]      allocated_ = n2;
[a]  }
[b]  p_[used_] = s;
[c]  ++used_;
}
```

Na kterých řádcích může být vyvolána výjimka?

odpověď:

4, 6, b

(4 - pro nedostatek paměti. 6 - to, že jsme zvětšili alokovanou paměť dvakrát ještě neznamená, že je větší než used; navíc opět může spadnout pro nedostatek paměti (volání copy konstruktoru operátorem =, tedy alokace paměti). b - vůbec nezkoumáme velikost s; opět alokujeme paměť v copy konstruktoru)

otazka:

Pokud nastane vyjimka:

[A] zanecha zasobnik v nezmenenem stavu

[B] muze zpusobit ztratu (nedostupnost) alokovane pameti

[C] muze zanechat zasobnik v nekonzistentnim stavu, který muze zpusobit behovou chybou

odpoved:

B

(B - Nekonzistentni stav je takovy stav, ve kterem je cast prvku v zasobniku v poradku (tedy platne ukazatele) a cast ukazatelu ukazuje na nejake nahodne misto v pameti. A ted proc se zasobnik nemuze dostat do nekonzistentniho stavu? Cely trik je skryty v radku [8]. Kdys vznikne vyjimka na radku [6], nachazi se pole p2 v nekonzistentnim stavu ... avsak radek [8] se uz nevykona. To znamena, ze v zasobniku zustanou stare prvky p_ , které jsou konzistentni. -> zaslal Peto Cerno)

8. otazka:

Kdy se vyhodnoti konstruktor globalni promenne?

odpoved:

pred vstupem do main

9. otazka:

v jakem poradi se volaji konstruktory tridy? (datove polozky, predci tridy, trida)

odpoved:

predci tridy, datove polozky, trida samotna

10. otazka:

v jakem poradi se volaji destruktory tridy? (datove polozky, predci tridy, trida)

odpoved:

trida samotna, datove polozky, predci tridy

(v opacnem, nez konstruktory)

11. otazka:

ktera z techto funkci se nejvic hodi pro implementaci scitani dvou komplexnich cisel:

[A] Complex operator+(Complex a, Complex b);

[B] Complex& operator+(const Complex& a,const Complex& b);

[C] Complex operator+(Complex& a, Complex& b)const;

[D] Complex operator+(const Complex& a, const Complex& b);

odpoved:

D

(vhodne je vratet hodnotou, globalni funkce by nemela byt const, argumenty by mely byt const, aby funkce mohla zpracovavat i konstantni argumenty)

12. otázka:

```
class T
{
public:
    double f(char c)
    {
        printf("double char");
        return c;
    }
};

class U:public T
{
    int f(int i)
    {
        printf("int int");
        return i;
    }
public:
    char f(double d)
    {
        printf("char double");
        return (int)d;
    }
};

void main (void)
{
    U z;
    char x = z.f('A');
}
```

Ktera funkce se zavola?

odpoved:

tento kod skonci s chybou "cannot acces private member declared in class 'U'"

(pri pretezovani funkci se neberou v uvahu funkce zdedene, nejlevnejsi konverze z char je na int, ale ta je privatni)

13. otázka:

Jaky musi mit destruktork abstraktni trida?

[A] virtualni

[B] ciste virtualni

[C] zadny

odpoved:

A

14. otázka:

Jaky je rozdíl?

```
class T1
{
public:
    int a;
    void f();
}
```

```
struct T2
{
    int a;
    void f();
}
```

[A] T1 nelze prirazovat

[B] T2 nelze prirazovat

[C] zadny

odpoved:

C

(struct je synonymum pro class-public)

15. otázka:

```
class base
{
public:
    virtual base operator+(base &);
}
```

```
class real: public base
{
public:
    virtual base operator+(base &);
    real operator+(real &);
}
```

```
class complex: public base
{
public:
    virtual base operator+(base &);
    complex operator+(complex &);
}
```

```
real r;
complex c;
base b=r;
b=b+c;
```

ktery operator se zavola?

[A] complex complex::operator+(complex &);

[B] real real::operator+(real &);

[C] base complex::operator+(base &);

[D] base base::operator+(base &);

odpoved:

D

(b je typu base, protoze to neni odkaz, ani ukazatel)

16. otazka:

```
class Complex
{
public:
    double Re, Im;
    Complex(Re=0.0, Im=0.0);
    Complex operator+(Complex &);
}

Complex Complex::operator+(Complex & b)
{
    double r = Re + b.Re;
    double i = Im + b.Im;
    ???;
}
```

co se dosadi za ??? ?

[A] return Complex(r) + Complex(i);

[B] return * this;

[C] return Complex(r, i);

[D] neni legalni

odpoved:

C

17. otazka:

```
class Complex
{
public:
    double Re, Im;
    Complex(Re=0.0, Im=0.0);
    Complex operator+(Complex &) const;
}

Complex Complex::operator+(Complex & b) const
{
    Re = Re + b.Re;
    Im = Im + b.Im;
    ???;
}
```

co se dosadi za ??? ?

[A] return Complex(r) + Complex(i);

[B] return * this;

[C] return Complex(r, i);

[D] neni legalni

odpoved:

D

(funkce je konstantni, nemuze menit data v objektu)

18. otázka:

```
class T
{
    T();
    T(int);
    T(const T &);
}
```

Který constructor se vyvolá při volání `f(T p)`?

- [A] `T::T()`;
- [B] `T::T(int)`;
- [C] `T::T(const T&)`;

odpoved:

C

(při předávání hodnoty do funkce se objekt kopíruje, proto copy-konstruktor)

19. otázka:

```
f(std::vector v)
{
    [A] std::vector::iterator it;
    [B] it=std::find(v.begin(), v.end(), 0);
    [C] v.insert(it, -1);
    [D] v.insert(it+1, 1);
}
```

na kterých řádcích může nastat výjimka?

odpoved:

D

(po insertu se všechny iteratory zneplatní)

20. otázka:

```
class A
{
public:
    virtual f();
}

class B: public A
{
public:
    virtual f();
}
```

```
B b;
A a=b;
a.f();
```

která funkce se zavolá?

- [A] `A::f()`
- [B] `B::f()`

odpoved:

A

(virtuální funkce se v potomcích redefinuje. zavolá se ta, která patří k 'volajícímu' objektu.)

(jiná verze zdůvodnění: dochází ke slicing, protože se do předka uloží jenom část potomka)

21. otázka:

na kterých radcích může nastat výjimka?

```
f(std::vector v)
{
[A]  std::vector::iterator it;
[B]  it=std::find(v.begin(), v.end(), 0);
[C]  v.insert(it+1, 1);
[D]  v.insert(it, -1);
}
```

odpověď:

C, D

(C - pokud bychom nulový prvek nenasli, D - pokud by C prošlo, po insertu jsou iterátory neplatné)

22. otázka:

kdy se vykonají konstruktory nestatických položek objektu?

- [A] při dynamické alokaci
- [B] po skončení konstruktoru
- [C] až je potřeba
- [D] před telem konstruktoru objektu

odpověď:

D

(pozor - nestatické položky neznaménají dynamické, ale položky, které nejsou uvozené slovem static)

23. otázka:

jaký by měl být konstruktor abstraktní třídy?

- [A] private
- [B] public
- [C] protected
- [D] žádný

odpověď:

C

24. otázka:

```
Class U
{
    U() {a = 1}
    U(int i) {a= i + 2}
    int a;
}

Class T: public U
{
    T() : U(2) {}
    T(int i) {a = a + 3}
    T(const U &p) : U(p.a) {}
}
```

```
T x = U();
```

Jakou hodnotu bude mít x.a?

- [A] 4
- [B] 3
- [C] 5
- [D] 1

odpověď:

B

(nejdříve se zavolá U(), takže a=1. potom konverzní konstruktor z T na U)

25. otázka:

co je konzistentni stav?

odpoved:

nedostupna data byla korektne destruovana a odalokovana
ukazatele nemiri na odalokovana data
plati dalsi invarianty dane logikou aplikace
(ze slidu)

26. otázka:

jaky je nejvhodnejsi operator+= pro Complex

- [A] Complex operator+=(const & Complex)
- [B] Complex& operator+=(const & Complex)
- [C] Complex& operator+=(const & Complex, const & Complex)
- [D] Complex& operator+=(const & Complex, const & Complex) const

odpoved:

B

27. otázka:

trida SStk implementuje zasobnik retezcu takto:

```
class SStk
{
public:
    /* ... */
    void pop(std::string & out);
private:
    std::string * p_;
    int allocated_, used_;
}
```

V konzistentnim stavu polozka allocated_ urcuje celkovou velikost pole p_ a polozka used_ pocet pouzitych polozek. oznacte radky, které mohou zpusobit vyjimku:

```
void SStk::pop(std::string & out)
{
[a]   if(used_)
    {
[b]       --used_;
[c]       out=p_[used_];
    }
}
```

jak se tato funkce chova pri vyjimce (predpokladejme, ze pri vstupu do funkce je zasobnik konzistentni)

- [A] zanecha zasobnik v nezmenenem stavu
- [B] zanecha zasobnik ve zmenenem, avsak konzistennim stavu
- [C] zanecha zasobnik v nekonzistentnim stavu

odpoved:

c, B

(c - muze dojit pamet, B - used_ se jiz odecetlo, coz je spravne. pri vyjimce se nic nevrati, ale stav zasobniku bude zmeneny a v podstate stejny, jako by k vyjimce nedoslo)

28. otázka:

predpokladajme tyto deklarace:

```
class T
{
public:
    int a;
};

class U : public T
{
public:
    int b;
};
```

T x;

U y;

jakou operaci provede nasledujici prikaz?

x=y;

- [A] dojde k pretečení dynamicky alokovane pameti
- [B] cast objektu y se okopiruje na misto objektu x
- [C] prikaz bude kompilátorem oznacen jako chybný
- [D] objekt x zanikne a misto nej vznikne kopie y

odpoved:

B

29. otázka:

ktery z techto globalnich operatoru je nejvhodnejši jako implementace scitani komplexnich cisel?

- [A] Complex operator+ (Complex &, Complex &)
- [B] Complex operator+ (const Complex &, const Complex &)
- [C] Complex operator+ (const Complex &, const Complex &) const
- [D] const Complex operator+ (const Complex &, const Complex &)

odpoved:

B

(A - kdyby byly argumenty const, tak je nevezme. C,D - ty const z toho delaji nepouzitelne funkce - nebo hodne teško pouzitelne)

30. otázka:

trida SStk implementuje zasobnik retezcu takto:

```
class SStk
{
public:
    /* ... */
    void pop(std::string & out);
private:
    std::string * p_;
    int allocated_, used_;
}
```

V konzistentnim stavu polozka allocated_ urcuje celkovou velikost pole p_ a polozka used_ pocet pouzitych polozek. oznacte radky, ktere mohou zpusobit vyjimku:

```
void SStk::pop(std::string & out)
{
[a]   if(used_)
      {
[b]       out=p_[used_-1];
[c]       --used_;
      }
}
```

jak se tato funkce chova pri vyjimce (predpokladejme, ze pri vstupu do funkce je zasobnik konzistentni)

[A] zanecha zasobnik v nezmenenem stavu

[B] zanecha zasobnik ve zmenenem, avsak konzistennim stavu

[C] zanecha zasobnik v nekonzistentnim stavu

odpoved:

b, A

(b - muze dojít pamet)

31. otázka:

Předpokládejme tyto deklarace:

```
class Base
{
public:
    virtual Base operator+ (const Base &) const;
}

class Complex;

class Real:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Real operator+ (const Real &) const;
    virtual Complex operator+ (const Complex &) const;
}

class Complex:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Complex operator+ (const Complex &) const;
    virtual Complex operator+ (const Real &) const;
}

Real r;
Complex c;
Base b = r;

b = b + c;
```

Který operator se zavola?

- [A] Real::operator+ (const Real &) const
- [B] Real::operator+ (const Base &) const
- [C] Complex::operator+ (const Real &) const
- [D] Base::operator+ (const Base &) const

odpoved:

D

(b je typu base)

32. otázka:

predpokladajme tyto deklarace:

```
class T
{
public:
    int a;
};

class U : public T
{
public:
    int b;
};
```

U x;

T y;

jakou operaci provede nasledujici prikaz?

x=y;

[A] prikaz bude kompilátorem oznacen jako chybný

[B] objekt x zanikne a místo něj vznikne kopie y

[C] promenna x se přesmeruje na objekt y

[D] okopíruje celý objekt y na místo objektu x

odpoved:

A

(nemuzeme 'mensi' (rodice) objekt nakopírovat do 'vetsiho' (potomka))

33. otázka:

(kod je nesmyslný - to není omyl. skutečně existuje varianta písemky, ve které se vyskytuje)

trída SStk implementuje zásobník řetězců takto:

```
class SStk
{
public:
    /* ... */
    void push(const std::string & s);
private:
    std::string * p_;
    int allocated_, used_;
};
```

v konzistentním stavu položka allocated_ určuje celkovou velikost pole p_ a položka used_ počet použitých položek. označte řádky, které mohou způsobit výjimku:

```
void SStk::push (const std::string & s)
{
[1]  if (used_ >= allocated)
[2]  {
[3]      int n2 = (allocated_ +1) * 2;
[4]      std::string * p2 = new string[n2];
[5]      for(int i = 0; i < used_; i++)
[6]          p_[i] = p2[i];
[7]      delete[] p2;
[8]      p_ = p2;
[9]      allocated_ = n2;
[a]  }
[b]  p_[used_] = s;
[c]  ++used_;
}
```

jak se tato funkce chová při výjimce? (předpokládejme, že při vstupu do funkce je zásobník konzistentní)

[A] zanechá zásobník v nezměněném stavu

[B] může způsobit ztrátu (nedostupnost) alokované paměti

[C] může zanechat zásobník ve stavu neodpovídajícím uvedenému konzistentnímu stavu

odpověď:

4, 6, b, B, C

(4 - pro nedostatek paměti. 6 - to, že jsme zvětšili alokovanou paměť dvakrát ještě neznamena, že je větší než used; navíc opět může spadnout pro nedostatek paměti (volání copy konstruktoru operátorem =, tedy alokace paměti). b - vůbec nezkoumáme velikost s; opět alokujeme paměť v copy konstruktoru; B - ztratíme nově alokovanou paměť; C - kvůli té ztrátě paměti)

34. otázka:

Předpokládejme tyto deklarace:

```
class Base
{
public:
    virtual Base operator+ (const Base &) const;
}

class Complex;

class Real:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Real operator+ (const Real &) const;
    virtual Complex operator+ (const Complex &) const;
}

class Complex:public Base
{
public:
    virtual Base operator+ (const Base &) const;
    virtual Complex operator+ (const Complex &) const;
    virtual Complex operator+ (const Real &) const;
}

Real r;
Complex c;
Base & b = r;

b = b + c;
```

Který operator se zavola?

- [A] Base::operator+ (const Base &) const
- [B] Real::operator+ (const Base &) const
- [C] Real::operator+ (const Complex &) const
- [D] Complex::operator+ (const Real &) const

odpoved:

B

(jsou dve faze:

- 1) overloading
- 2) virtual methods

1) overloading je při prekladu, zjistuje se která funkce ze všech možných se na základě parametru zavola. B je base, C je complex, B + C je operator+ becka s parametrem Complex. B je Base a Base má jedinou možnost 'Base operator+ (const Base &) const'. kdyby v Base bylo třeba 'Complex operator+ (const Complex &) const' tak by se volalo Complex a ne Base. prostě se koukne a najde nejlevnější overloading. tady je jediný možný.

2) za behu: za behu se pers * a & používá VMT, takže v b je Real r, takže se použije VMT a koukne se do Real::operator+ (const Base &) const)