0. otazka:
co je to singleton?

odpoved:
trida, ktera 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. otazka:

```
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. tady je odpoved zrejma - pro char ani nemusime konverzi delat.)

5. otazka:

```
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. otazka:

```
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] neni definovane
[B] 2
[C] 3
[D] 1

odpoved:
C
(jestlize alespon jeden typ je presne urcen, ma tato definice prednost)

7. otazka:

```
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 kterych radcich muze byt vyvolana vyjimka?

odpoved:
4, 6, b
(4 - pro nedostatek pameti. 6 - to, ze jsme zvetsili alokovanou pamet dvakrat jeste neznamena, ze je vetsi nez used; navic opet muze spadnout pro nedostatek pameti (volani copy konstruktoru operatorem =, tedy alokace pameti). b - vubec nezkoumame velikost s; opet alokujeme pamet 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, ktery muze zpusobit behovou chybu

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]. Kdyz 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_ , ktere 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 kostruktory)

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 vracet hodnotou, globalni funkce by nemela byt const,

argumenty by mely byt const, aby funkce mohla zpracovavat i konstantni argumenty)

12. otazka:

```
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. otazka:
Jaky musi mit destruktor abstraktni trida?

[A] virtualni
[B] ciste virtualni
[C] zadny

odpoved:
A

14. otazka:
Jaky je rozdil?

```
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. otazka:

```
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. otazka:

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

Ktery constructor se vyvola pri volani f(T p)?
[A] T::T();
[B] T::T(int);
[C] T::(const T&);

odpoved:
C
(pri predavani hodnoty do funkce se objekt kopiruje, proto copy-konstruktor)

19. otazka:

```
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 kterych radcich muze nastat vyjimka?

odpoved:
D
(po insertu se vsechny iteratory zneplatni)

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

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

B b;
A a=b;
a.f();
```
ktera funkce se zavola?
[A] A::f()
[B] B::f()

odpoved:
A
(virtualni funkce se v potomcich redefinuje. zavola se ta, ktera patri k 'volajicimu' objektu.)
(jiná verze zdůvodnění: dochází ke slicingu, protože se do předka uloží jenom část potomka)

21. otazka:
na kterych radcich muze nastat vyjimka?
```
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);
}
```

odpoved:
C, D
(C - pokud bychom nulovy prvek nenasli, D - pokud by C proslo, po insertu jsou iteratory neplatne)

22. otazka:
kdy se vykonaji konstruktory nestatickych polozek objektu?
[A] pri dynamicke alokaci
[B] po skonceni konstruktoru
[C] az je potreba
[D] pred telem konstruktoru objektu

odpoved:
D
(pozor - nestaticke polozky neznamenaji dynamicke, ale polozky, ktere nejsou uvozene slovem static)

23. otazka:
jaky by mel byt konstruktor abstraktni tridy?
[A] private
[B] public
[C] protected
[D] zadny

odpoved:
C

24. otazka:
```
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 mit x.a?
[A] 4
[B] 3
[C] 5
[D] 1

odpoved:
B
(nejdriv se zavola U(), takze a=**1.** potom konverzni konstruktor z T na U)

25. otazka:
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. otazka:
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. otazka:
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]    --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. otazka:
predpokladejme 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 preteceni dynamicky alokovane pameti
[B] cast objektu y se okopiruje na misto objektu x
[C] prikaz bude kompilatorem oznacen jako chybny
[D] objekt x zanikne a misto nej vznikne kopie y

odpoved:
B

29. otazka:
ktery z techto globalnich operatoru je nejvhodnejsi 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 tezko pouzitelne)

30. otazka:
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 dojit pamet)

31. 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 Complex operator+ (const Complex &) const;
 virtual Complex operator+ (const Real &) const;
}

Real r;
Complex c;
Base b = r;

b = b + c;
```
Ktery 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. otazka:
predpokladejme 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 kompilatorem oznacen jako chybny
[B] objekt x zanikne a misto nej vznikne kopie y
[C] promenna x se presmeruje na objekt y
[D] okopiruje cely objekt y na misto objektu x

odpoved:
A
(nemuzeme 'mensi' (rodice) objekt nakopirovat do 'vetsiho' (potomka))

33. otazka:
(kod je nesmyslny - to neni omyl. skutecne existuje varianta pisemky, ve ktere se vyskytuje)
trida SStk implementuje zasobnik retezcu takto:
```
class SStk
{
public:
 /* ... */
 void push(const std::string & s);
private:
 std::string * p_;
 int allocated_, used_;
};
```
v konzistennim stavu polozka allocated_ urcuje celkovou velikost pole p_ a polozka used_ pocet pouzitych polozek. oznacte radky, ktere mohou zpusobit vyjimku:
```
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 chova pri vyjimce? (predpokladejme, ze pri vstupu do funkce je zasobnik konzistentni)
[A] zanecha zasobnik v nezmenenem stavu
[B] muze zpusobit ztratu (nedostupnost) alokovane pameti
[C] muze zanechat zasobnik ve stavu neodpovidajicim uvedenemu konzistentnimu stavu

odpoved:
4, 6, b, B, C
(4 - pro nedostatek pameti. 6 - to, ze jsme zvetsili alokovanou pamet dvakrat jeste neznamena, ze je vetsi nez used; navic opet muze spadnout pro nedostatek pameti (volani copy konstruktoru operatorem =, tedy alokace pameti). b - vubec nezkoumame velikost s; opet alokujeme pamet v copy konstruktoru; B - ztratime nove alokovanou pamet; C - kvuli te ztrate pameti)

34. 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 Complex operator+ (const Complex &) const;
 virtual Complex operator+ (const Real &) const;
}

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

b = b + c;
```

Ktery 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 pri prekladu, zjistuje se ktera funkce ze vsech moznych se na zaklade parametru zavola. B je base, C je complex, B + C je operator+ becka s parametrem Complex. B je Base a Base ma jedinou moznost 'Base operator+ (const Base &) const'. kdyby v Base bylo treba 'Complex operator+ (const Complex &) const' tak by se volalo Complex a ne Base. proste se koukne a najde nejlevnejsi overloading. tady je jediny mozny.

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