

Y36PJC Programování v jazyce C/C++

# Třídy a objekty I

Ladislav Vagner

# Dnešní přednáška

- Deklarace třídy.
- Konstruktor, destruktory, metody.
- Zapouzdření.
- Třídní a instanční proměnné a metody.
- Třídy v C++ a struktury v C.

## Minulá přednáška

- Dynamická alokace paměti.
- Ukazatele na funkce a pole.
- Práce s řetězcí v C.
- Buffer-overflow chyby.
- Práce s řetězcí v C++.

# Třídy a objekty

- Třídy:
  - Reprezentace modelované skutečnosti v programu.
  - Soustředí související údaje do jednoho celku.
  - Uzavření před vnějším světem (omezení přístupu).
  - Rozhraní pro práci (metody).
  - Využitelné pro polymorfismus a dědění.
  - Pouze v C++.
  - V C třídy nejsou, pouze struktury (bez metod a řízení přístupu).
- Objekty:
  - instance nějaké (právě jedné) třídy,
  - třída má typicky mnoho instancí.

## Příklad třídy v C++

```
class CCplx
{
    public:
        CCplx(double r, double i=0) {re=r; im=i;}
        CCplx(const char * str) { ... }
        ~CCplx(void) { }
        double Re    (void) {return re;}
        double Im    (void) {return im;}
        double Abs   (void) {return sqrt (re*re+im*im);}
        double Phi   (void) {return atan2 (im, re);}
    private:
        double re, im;
};

CCplx  a ( 3, 4 );
cout << a.Re()   << "+" << a.Im () << "i = " <<
      a.Abs() << "<" << a.Phi() << endl;
```

# Deklarace třídy v C++

- Konstruktor:
  - jméno stejné jako třída,
  - inicializace členských proměnných,
  - volán vždy při vzniku instance,
  - konstruktorů může být ve třídě více,
  - rozlišení – viz pravidla pro přetěžování funkcí.
- Destruktor:
  - jméno stejné jako třída s ~,
  - maximálně jeden,
  - volán vždy, když je odstraňována instance,
  - úklid členských proměnných.
- Existují v Javě destruktory?

## Deklarace třídy v C++

- Řízení přístupu:
  - **public:** – neomezený přístup,
  - **protected:** – pouze pro metody třídy a metody potomků,
  - **private:** – pouze pro metody třídy.
- Viditelnost je daná blokově, podle poslední použité direktivy **public/protected/private**.

# Deklarace třídy v C++

- Klíčové slovo `class`:
  - zahajuje deklaraci třídy,
  - stejný význam má i klíčové slovo `struct`.
- Rozdíl mezi `class` a `struct`:
  - `class` – implicitní viditelnost je `private`,
  - `struct` – implicitní viditelnost je `public`.

```
class ABC
{ public:
    ...
};
struct ABC
{ ... };
```

```
struct XYZ
{ private:
    ...
};
class XYZ
{ ... };
```



# Deklarace třídy v C++

- Deklarace metod – v deklaraci třídy.
- Definice metod (těla metod):
  - v deklaraci třídy (jako v Javě),
  - v odděleném souboru.
- Definice metod v deklaraci – inline metody:
  - doporučení kompilátoru, že má metody rozepsat na místě použití,
  - podobné klíčovému slovu **inline** u funkcí.
- Doporučení inline nemusí překladač respektovat:
  - dlouhé metody,
  - rekurzivní metody,
  - ukazatel na metodu.

# Deklarace třídy v C++

```
// inline způsob
class CCplx
{
    public:
        CCplx(double r, double i=0) {re=r; im=i;}
        CCplx(const char * str) { ... }
        ~CCplx(void) { }
        double Re    (void) {return re;}
        double Im    (void) {return im;}
        double Abs   (void) {return sqrt (re*re+im*im);}
        double Phi   (void) {return atan2 (im, re);}
    private:
        double re, im;
};
```

# Deklarace třídy v C++

```
// .h soubor - deklarace, ne tela metod
class CCplx
{
    public:
        CCplx(double r, double i=0);
        CCplx(const char * str)
        ~CCplx(void);
        double Re    (void);
        double Im    (void);
        double Abs    (void);
        double Phi    (void);
    private:
        double re, im;
};
```

# Deklarace třídy v C++

```
// .cpp soubor - definice tel metod
#include "CCplx.h"

    CCplx::CCplx(double r, double i)
{ // uz bez implicitnich hodnot parametru
    re = r;
    im = i;
}

double CCplx::Re    (void)
{
    return re;
}

...
```

# Konstruktory v C++

- Inicializace členských proměnných při vzniku instance.
- Výběr konstruktoru parametry v závorce při vytváření instance:

```
CCplx  a ( 10, 20 );  
CCplx  b ( "10 + 2i" );
```

- Pravidla pro přetěžování funkcí.
- Implicitní hodnoty parametrů – snížení počtu přetížených konstruktorů.

## Destruktory v C++

- Volán při zániku instance.
- Uvolnění prostředků, které objekt vlastnil:
  - dynamicky alokovaná paměť,
  - prostředky OS (soubory, sokety, thready, semaforey, mutexy, ...).
- Není povinný, systém si "domyslí" prázdný destruktory, pokud neexistuje jiný.
- V ukázkovém příkladu nebyl potřeba.

## Instance vytvořené staticky

```
void foo1 ( CCplx & x )
{
    cout << x . Abs ();
}
void foo2 ( CCplx * x )
{
    cout << x -> Abs ();
}
void foo3 ( void )
{
    CCplx a ( 10, 20 );    // konstruktor
    CCplx b ( "1 + 2i" ); // konstruktor

    foo1 ( a );
    foo2 ( &a );
    cout << a . Re ();    // pristup k metode pomoci .
} // destruktory a, destruktory b
```

# Instance vytvořené dynamicky

```
void foo1 ( CCplx * x )  
{  
    ...  
    cout << x -> Abs ();  
}
```

```
void foo2 ( void )  
{  
    CCplx * a;  
    CCplx b = new CCplx ( "1 + 2i" ); // konstruktor  
  
    a = new CCplx ( 10, 20 ); // konstruktor  
    foo1 ( a );  
    cout << a -> Re (); // pristup k metode pomoci ->  
    delete a; // destruktork a  
    delete b; // destruktork b  
}
```



# Operátory . a ->

- Oba operátory slouží pro:
  - přístup ke členským proměnným,
  - volání metod.
- Operátor . se použije pro:
  - práci přímo s instancí,
  - práci s referencí.
- Operátor -> se použije pro práci s ukazatelem na instanci.
- Platí:  
$$X . \text{foo} () \iff (\&X) -> \text{foo} () ;$$
$$Y -> \text{foo} () \iff (*Y) . \text{foo} () ;$$

# Konstantní metody

```
class CCplx
{
    public:
        ...
        double Re    ( void ) const { return re; }
        double Im    ( void )      { return im; }
        void  Inc    ( void ) const {re += 1;} // !!
        ...
};
```

```
CCplx a ( 1, 2 );
const CCplx b ( 3, 4 );
cout << a . Re (); // ok
cout << a . Im (); // ok
cout << b . Re (); // ok
cout << b . Im (); // Im není const
```

# Třídni metody

- Instanční metoda:
  - spuštěna nad konkrétní instancí,
  - má přístup k jejím členským proměnným.
- Třídni metoda:
  - spuštěna bez konkrétní instance,
  - má přístup pouze ke svým parametrům a ke globálním proměnným.
- Deklarace třídni metody – klíčové slovo **static**.
- Volání třídni metody – čtyřtečková notace.

# Třídni metody

```
class CCplx
{
    public:
        ...
        static CCplx Add ( const CCplx & a,
                           const CCplx & b )
        {
            return CCplx ( a.re + b.re, a.im + b.im );
        }
        ...
};
```

```
CCplx u ( 2,3 ), v (4, 5), w;
```

```
w = CCplx::Add ( u, v );
```

# Třídní proměnné

- Instanční proměnné:
  - každý objekt má vlastní sadu instančních proměnných,
  - instanční proměnné různých objektů se vzájemně "nevidí".
- Třídní proměnné:
  - vázané na třídu,
  - v systému existuje právě 1x, bez ohledu na počet aktivních instancí.
- Deklarace třídní proměnné – klíčové slovo **static**.
- Zpřístupnění třídní proměnné – čtyřtečková notace.
- Využití – hlavně pro tabulky předpočítaných hodnot.

# Třídní proměnné

```
// hlavickovy soubor - deklarace
class CCplx
{
    public:
        CCplx (double r, double i);
        ~CCplx ( void );

        ...
    protected:
        double re, im;
        static int instCnt;

        ...
};
```

## Třídní proměnné

```
// soubor .cpp - definice
#include "CCplx.h"

int CCplx::instCnt = 0; // definice

    CCplx::CCplx (double r, double i)
{
    re = r; im = i; instCnt ++;
}

    CCplx::~~CCplx ( void )
{
    instCnt --;
}
```

# Struktury v C

- Podobné třídám v C++:
  - členské proměnné – složky struktury,
  - neobsahovaly metody, konstruktor a destruktork.
- Použití:
  - v C docela běžné,
  - základ C++ tříd.



# Struktury v C

```
struct TTime  
{  
    int h, m, s;  
};
```

```
struct TTime a;    // C zapis  
a . h = 12;  
a . m = 0;  
a . s = 0;
```

## Unie v C

- Podobné strukturám:
  - složky unie jsou skládané "přes" sebe,
  - variantní záznam – platí jen jedna z nich,
  - šetří paměť.
- Příklad:

```
union UAngle
{
    struct
    {
        int deg, min, sec;
    } degrees;
    double radians;
};
```

## Unie v C

- Problém – která složka platí?
  - Informace musí být součástí unie.
- Vyžaduje další (vnější strukturu - nepraktické).
- Nepřehledný přístup ke složkám.
- Řešení třídami a dědičností je zde mnohem přehlednější.
- Použití – výjimečné, pro volání některých služeb OS (typicky `ioctl`).

## Unie v C

```
struct TAngle
{ enum { DEG, RAD } eType;
  union
  {
    struct
    { int deg, min, sec; } degrees;
    double radians;
  } uVal;
};
```

```
struct TAngle x;
x . eType = DEG;
x . uVal . degrees . deg = 90;
...
```

# Třídy jako datové struktury

- Abstraktní datové typy:
  - definované rozhraní pro práci,
  - definované axiomy popisující chování.
- Příklady:
  - zásobník,
  - fronta,
  - tabulka,
  - prioritní fronta,
  - seznam.
- Implementace:
  - třída (pro konkrétní datový typ),
  - třída polymorfní,
  - třída generická.

# Třídy jako datové struktury

- Příklad – zásobník celých čísel:
  - vznik (konstruktor),
  - zánik (destruktor),
  - vložení (push),
  - čtení z vrcholu (read),
  - čtení a vyzvednutí (pop),
  - test prázdnoty (isEmpty).
- Operace se promítanou do veřejného rozhraní (metody, konstruktor, destruktor).
- Vnější svět je izolován od implementace:
  - pole fixní velikosti,
  - pole realokované podle potřeby,
  - spojový seznam.

# Zásobník jako třída

```
class CStack
{
    public:
        CStack      ( int maxSize=100 );
        ~CStack     ( void );
        int         Push      ( int X );
        int         Read      ( int & X ) const;
        int         Pop       ( int & X );
        int         IsEmpty   ( void ) const;
    protected:
        int         dataNr;    // first free index
        int         dataMax;   // max. size
        int         * data;    // dyn. alloc data
};
```

# Zásobník jako třída

```
CStack::CStack    ( int maxSize )
{
    dataNr = 0;
    dataMax = maxSize;
    data    = new int [maxSize];
}

CStack::~~CStack  ( void )
{
    delete [] data;
}

int CStack::Push   ( int X )
{
    if ( dataNr >= dataMax ) return ( 0 );
    // full - failed
    data[dataNr++] = X;
    return ( 1 ); // success
}
```



## Zásobník jako třída

```
int CStack::Read      ( int & X ) const
{
    if ( IsEmpty () ) return ( 0 ); // empty - failed
    X = data[dataNr - 1];
    return ( 1 );
}
int CStack::Pop       ( int & X )
{
    if ( Read ( X ) )
    {
        dataNr --;
        return ( 1 );
    }
    return ( 0 );
}
int CStack::IsEmpty   ( void ) const
{ return ( dataNr == 0 ); }
```

# Zásobník jako třída

```
int main ( int argc, char * argv [] )
{
    CStack x;
    int y;

    x . Push ( 10 );
    x . Push ( 20 );
    x . Push ( 30 );

    while ( ! x . IsEmpty () )
    {
        x . Pop ( y );
        cout << y << endl;
    }
    return ( 0 );
}
```

# Třídy jako datové struktury

- Nevýhoda implementace:
  - shora omezená velikost,
  - struktura se nedokáže přizpůsobit požadavkům.
- Řešení:
  - realokace pole podle potřeby:
    - režie na kopírování.
  - spojový seznam:
    - režie – správa paměti.
  - kombinace.

# Zásobník jako třída

```
class CStack
{
    public:
        CStack      ( void );
        ~CStack     ( void );
        int         Push      ( int X );
        int         Read      ( int & X ) const;
        int         Pop       ( int & X );
        int         IsEmpty   ( void ) const;
    protected:
        struct TItem
        {
            TItem * Next;
            int   Data;
        };
        TItem * top;    // linked list
};
```

# Zásobník jako třída

```
CStack::CStack    ( void )
{
    top = NULL;
}
CStack::~~CStack  ( void )
{
    TItem * tmp;

    while ( top )
    {
        tmp = top -> Next;  // !! uložit kopii
        delete top;
        top = tmp;
    }
}
```

# Zásobník jako třída

```
int CStack::Push      ( int X )
{
    TItem * n;

    n          = new TItem;
    n -> Data = X;
    n -> Next = top;
    top = n;
    return ( 1 ); // success
}

int CStack::Read      ( int & X ) const
{
    if ( IsEmpty () ) return ( 0 ); // empty - failed
    X = top -> Data;
    return ( 1 );
}
```

# Zásobník jako třída

```
int CStack::Pop      ( int & X )
{
    if ( Read ( X ) )
    {
        TItem * tmp = top -> Next;
        delete top;
        top = tmp;
        return ( 1 );
    }
    return ( 0 );
}

int CStack::IsEmpty  ( void ) const
{
    return ( top == NULL );
}
```

# Zásobník jako třída

```
int main ( int argc, char * argv [] )
{
    CStack x;
    int y;

    x . Push ( 10 );
    x . Push ( 20 );
    x . Push ( 30 );

    while ( ! x . IsEmpty () )
    {
        x . Pop ( y );
        cout << y << endl;
    }
    return ( 0 );
}
```



Dotazy...

Děkuji za pozornost.