FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ústav teoretické a experimentální elektrotechniky

# Seminář C++

7. přednáška

Autor: doc. Ing. Jan Mikulka, Ph.D.

2023

# Obsah přednášky

- Generické programování
  - Šablony
  - Šablony funkcí
  - Šablony objektových typů
  - Specializované a částečně specializované šablony

# Generické programování

- Základní myšlenkou generického programování je rozdělení kódu programu na algoritmus a datové typy.
- Kód lze následně chápat jako obecný, bez ohledu nad jakými datovými typy pracuje.
- Konkrétní kód algoritmu se z něj stává dosazením datového typu.
- Typickým příkladem může být objekt reprezentující 1D pole. Bez generického programování je datový typ jednotlivých prvků pole znám předem a je pevně daný. S využitím generického programování lze typ prvku pole definovat až ve vlastním programu.

# Šablony

- Šablona je nástrojem generického programování.
- Šablona je realizována pomocí klíčového slova **template**.
- Šablona definuje tzv. formální datový typ, nad kterým se implementuje obecný kód. Tento formální datový typ je následně definován konkrétním datovým typem při překladu programu.

# Šablony

- Deklarace šablony vypadá následovně:

**template** <seznam parametrů> deklarace

- „**template**“ je klíčové slovo
- „seznam parametrů“ je seznam formálních parametrů/typů
- „deklarace“ je deklarace funkce, metody, objektu, členské proměnné, …

# Šablony

- Definice šablonových, formálních parametrů:

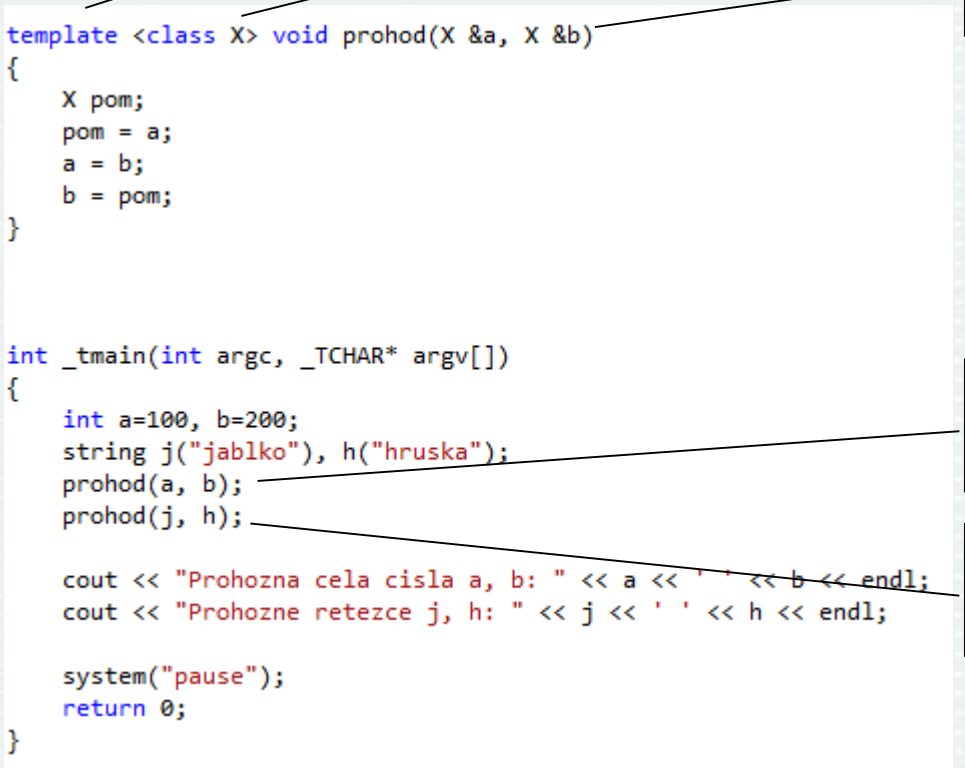
**template<class** mujtyp1**, class** mujtyp2**> class** objekt

**template**<**typename** mujtyp1, **typename** mujtyp2> **class** objekt

# Šablony funkcí

- Šablonové funkce mají velký význam v případě, kdy potřebujeme vytvořit několik funkcí se stejným významem, ale pro různé datové typy.
- Typickým příkladem může být funkce, která „prohodí" hodnoty dvou zadaných parametrů.
- Bez generického programování bychom tuto funkci přetížili pro všechny typy, které v parametrech předpokládáme.

# Šablony funkcí, příklad



```
template <class X> void prohod(X &a, X &b)
{
    X pom;
    pom = a;
    a = b;
    b = pom;
}


int _tmain(int argc, _TCHAR* argv[])
{
    int a=100, b=200;
    string j("jablko"), h("hruska");
    prohod(a, b);
    prohod(j, h);

    cout << "Prohozna cela cisla a, b: " << a << ' ' << b << endl;
    cout << "Prohozne retezce j, h: " << j << ' ' << h << endl;

    system("pause");
    return 0;
}
```

Šablona

Formální datový typ X

Dva parametry typu reference na X

Volání šablonové funkce pro typ int a int

Volání šablonové funkce pro typ string a string

# Šablony objektových typů

- Deklarace šablonové třídy vypadá následovně:

```
template <seznam parametrů> klíč název
{
};
```

- kde **klíč** je buď **class**, **struct** nebo **union**

# Šablony objektů, příklad

```
template<class T> class komplex {
    T re, im;

public:
    komplex(T re, T im) {
        this->re = re;
        this->im = im;
    }

    template<class P> friend ostream& operator<<(ostream &out, komplex<P> &k);
};

template<class P> ostream& operator<<(ostream &out, komplex<P> &k) {
    return out << noshowpos << k.re << showpos << k.im << "i" << noshowpos;
}


int _tmain(int argc, _TCHAR* argv[])
{
    komplex<int> a(1, 2);
    komplex<double> b(3.5, 4);

    cout << a << endl << b << endl << endl;

    system("pause");
    return 0;
}
```

Deklarace formálního typu

Zapouzdřené proměnné formálního typu

Konstruktor s parametry formálního typu

Spřátelená šablonová funkce

Instance třídy komplex pro celočíselný typ re a im

Instance třídy komplex pro reacionální typ re a im

# Specializace šablon

- Explicitní specializací se rozumí případ, kdy šabloně přiřadíme novou definici pro konkrétní hodnoty parametrů.
- V případě, kdy zúžíme množinu parametrů, hovoříme o částečné specializaci. V případě, kdy předem definujeme množinu parametrů šablony, hovoříme o úplné specializaci.
- Při deklaraci specializace zapisujeme do lomených závorek hodnoty parametrů nebo konkrétní typy, které chceme při specializaci použít.
- Při vytváření instancí potom překladač zvolí specializaci, která nejlépe odpovídá skutečným parametrům.

# Specializace šablon, příklad

Primární šablona

```
template<class T> class pole {
    T *p;
    int s;

public:
    pole(int n) {
        p = new T[this->s = n];
        for (int i = 0; i < s; i++) p[i] = i;
    }

    ~pole() {
        delete [] p;
    }

    template<class P> friend ostream& operator<<(ostream &out, pole<P> &k);
};

template<class P> ostream& operator<<(ostream &out, pole<P> &p) {
    for (int i = 0; i < p.s; i++) out << p.p[i] << ' ';
    return out;
}
```

Specializace na pole pointerů

```
template<class T> class pole<T*> {
    T *p;
    int s;

public:
    pole(int n) {
        p = new T[this->s = n];
        for (int i = 0; i < s; i++) p[i] = i;
    }

    ~pole() {
        delete [] p;
    }

    template<class P> friend ostream& operator<<(ostream &out, pole<P*> &k);
};

template<class P> ostream& operator<<(ostream &out, pole<P*> &p) {
    for (int i = 0; i < p.s; i++) out << p.p[i] << ' ';
    return out;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    pole<int> A(10);
    pole<int*> B(10);

    cout << A << endl << B << endl << endl;

    system("pause");
    return 0;
}
```

# Specializace šablon, příklad

```
int _tmain(int argc, _TCHAR* argv[])
{
    pole<int> A(10);
    pole<bool> B(10);

    cout << A << endl << B << endl << endl;

    system("pause");
    return 0;
}
```

Primární šablona

```
template<class T> class pole {
    T *p;
    int s;

public:
    pole(int n) {
        p = new T[this->s = n];
        for (int i = 0; i < s; i++) p[i] = i;
    }

    ~pole() {
        delete [] p;
    }

    template<class P> friend ostream& operator<<(ostream &out, pole<P> &k);
};
```

Specializace na pole prvků bool

```
template<> class pole<bool> {
    bool *p;
    int s;

public:
    pole(int n) {
        p = new bool[this->s = n];
        for (int i = 0; i < s; i++) rand()%2==1?p[i] = true:p[i]=false;
    }

    ~pole() {
        delete [] p;
    }

    template<class P> friend ostream& operator<<(ostream &out, pole<P> &k);
};

template<class P> ostream& operator<<(ostream &out, pole<P> &p) {
    for (int i = 0; i < p.s; i++) out << p.p[i] << ' ';
    return out;
}
```

# Specializace šablon, příklad

```
int _tmain(int argc, _TCHAR* argv[])
{
    pole<int, 10> A;
    pole<double, 2> B;

    cout << A << endl << B << endl << endl;

    system("pause");
    return 0;
}
```

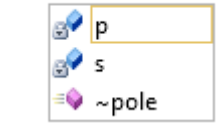
Primární šablona

```
template<class T, int n> class pole {
    T *p;
    int s;

public:
    pole() {
        p = new T[this->s = n];
        for (int i = 0; i < s; i++) p[i] = i;
    }

    ~pole() {
        delete [] p;
    }

    template<class P, int n> friend ostream& operator<<(ostream &out, pole<P, n> &k);
};
```

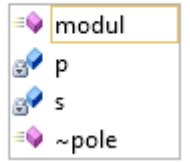
```
template<> class pole<double, 2> {
    double *p;
    int s;

public:
    pole() {
        p = new double[this->s = 2];
        for (int i = 0; i < s; i++) p[i] = i;
    }

    double modul(void) {
        return sqrt(p[0]*p[0]+p[1]*p[1]);
    }

    ~pole() {
        delete [] p;
    }

    template<class P, int n> friend ostream& operator<<(ostream &out, pole<P, 2> &k);
};

template<class P, int n> ostream& operator<<(ostream &out, pole<P, n> &p) {
    for (int i = 0; i < p.s; i++) out << p.p[i] << ' ';
    return out;
}
```

Specializace na dvouprvkové pole typu double

A.




B.



# Děkuji Vám
# za pozornost.

mikulka@vut.cz
www.utee.fekt.vut.cz