

# X36PJC

## 3. přednáška

Datové typy - dokončení  
Reference, typedef, class

# Minulá přednáška


- Primitivní datové typy
- Deklarace
- Operátory

# Reference

- Reference je:
  - alternativní pojmenování objektu
  - složený typ - vždy odkazuje na jiný typ
  - `<datový typ> & <identifikátor> = <objekt>;`
- Nelze definovat referenci na referenci!
- Reference musí být vždy inicializována!
- Musí vždy odkazovat na objekt!

# Reference (2)

- Příklady

```
int hod = 42;  incializa  
ce
```

```
int &refHod = hod; // ok: refHod  
odkazuje na hod
```

```
int &refHod2; // chyba!!! Proč?
```

```
int &refHod3 = 10; // chyba!!! Proč?
```

# Reference (3)

- Příklady

```
int hod = 42; int &refHod = hod;  
  refHod += 2;  
int hod2 = 50;  
refHod = hod2; ← přiřazení
```

- Jaká je hodnota hod, hod2, refHod?

# Definice více referencí

Pozor !!!

- `int i = 1024, i2 = 2048;`
- `int &r = i, r2 = i2; // jakého typu je r, r2?`
- `int i3 = 1024, &ri = i3; // jakého typu je i3, ri?`
- `int &r3 = i3, &r4 = i2; // jakého typu je r3, r4?`

# const Reference

- const Reference může odkazovat na const objekt.
- Zpřístupnění objektu/proměnné pouze pro čtení.
- Výhodné při předávání parametrů.
- `const <typ> & <ident> = <rval| var>;`
- var musí být const

# const Reference (2)

## Příklady

- `const int ival = 1024;`
- `const int &refVal = ival; // ok: obojí je const`
- `int &ref2 = ival; // error: nekonstantní reference na konstantní objekt`
- `int xVal = 50;`
- `const int &refVal2 = xval; // ok`
- `refVal2 += 3; //error`



# const Reference (3)

## Příklady

- `int i = 42;`
- `// možné pouze pro const Reference`
- `const int &r = 42;`
- `const int &r2 = r + i;`

# const Reference (4)

- Jak se zachová překladač v tomto případě?

```
double dval = 3.14;  
const int &ri = dval;
```

- Půjde tento úryvek přeložit?
  - Pokud ano, na co odkazuje ri?

# const Reference (5)

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {  
    double dval = 3.14;  
    const int &refHod = dval;  
    dval += 1.0;  
    cout << "Hod: " << dval << " refHod: " << refHod  
    << endl;  
    return 0;  
}
```

- **Jaký bude výstup tohoto programu?**

# typedef

- Umožňuje definovat synonymum pro typ
- `typedef <datový typ> <identifikátor>;`

## Příklady

- `typedef double wages;`
- `wages salary;`

# typedef (2)

- Nezavádí nový datový typ, ale pouze synonymum!
- Toto synonymum lze použít všude místo označení datového typu.
- Možnost definovat “synonyma synonym”
  - *typedef double wages;*
  - *typedef wages salary;*
  - *salary bob\_salary; // stejné jako double bob\_salary*
- K čemu to je vlastně dobré?

# typedef (3)

- Umožňuje skrýt implementaci datového typu
  - typedef double wages;
  - wages salary;
- Usnadňuje definici složitějších datových typů

# Výčtový typ

- Způsob jak definovat typ, který může nabývat omezený počet hodnot
- Příklad typu definujícího módy otevření souboru

```
// input is 0, output is 1, and append is 2  
enum open_modes {input, output, append};
```

# Výčtový typ (2)

- Může být inicializován const hodnotou  
// shape is 1, sphere is 2, cylinder is 3,  
polygon is 4  
enum Forms {shape = 1, sphere, cylinder,  
polygon};
- Hodnoty nemusí být unikátní  
// point2d is 2, point2w is 3, point3d is 3,  
point3w is 4  
enum Points { point2d = 2, point2w,  
point3d = 3, point3w };



# Výčtový typ (3)

- Každý výčtový typ je unikátním datovým typem
- Příklad (definice typů viz předchozí slide)

Points pt3d = point3d; // ok: point3d je typu Points

Points pt2w = 3; // chyba: pt2w nelze inicializovat pomocí int, je to jiný datový typ!!!

pt2w = polygon; // chyba: polygon není typu Points

pt2w = pt3d; // ok: oba objekty jsou stejného typu

# Třídy

- Uživatelem definované datové typy (UDT).
- 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).

# Třídy (2)

- Třída se skládá z rozhraní a implementace.
- Rozhraní sestává z operací, které třída poskytuje.
- Implementace obsahuje funkce a data nutné k realizaci rozhraní.

# Návrh Třídy

- Při návrhu postupujeme od specifikace rozhraní/operací, které bude třída poskytovat.
- Na základě rozhraní/operací identifikujeme potřebné datové položky, které si musí třída pamatovat.

# Příklad třídy v C++

```
class Book  
{  
    public:  
        Book(int id, double price) { _ident=id;  
        _price=price; }  
        ~Book(void) { }  
        double price(void) {return _price;}  
        int id(void) {return _ident;}  
    private:  
        int _ident; double _price;  
};  
Book b(23, 1.0);
```

# 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 (přednáška 7).

# Deklarace třídy v C++

- 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**.
- **struct** – z důvodu kompatibility s C
- Rozdíl mezi **class** a **struct**:
  - **class**            - implicitní viditelnost je **private**,
  - **struct**           - implicitní viditelnost je **public**.

# Deklarace třídy v C++

```
class Book {  
    .....  
};
```

- Středník za deklarácí třídy je povinný!!!
- Pokud není středník za deklarácí uveden, kompilátor hlásí nesrozumitelné chyby, těžko se hledá příčina. Vyzkoušejte si!

Dotazy?

Děkuji za pozornost!