

Výčtové datové typy

alias enumy

- jsou podobné jako v Pascalu,
- definují se klíčovým slovem enum.
- Příklad: `enum boolean {false,true};`
`enum boolean a=true,b;`
- Můžeme též přiřadit hodnoty:
`enum boolean{false=0,true=1};`

C++

je objektový jazyk

- C++ nás omezuje více než C (např. přetypování),
- nabízí také plno dalších věcí (objekty, přetěžování,...),
- v C++ se programuje podobně jako v C (opatrnost),
- ale jde o jiný jazyk!
- Syntax je konzervativním rozšířením C (sémantika ne).

Objekty

už byly v Pascalu a v C++ jsou podobné

- také se chovají jako struktury s funkcemi,
- také podporují virtuální metody, dědičnost, konstruktory a destruktory,
- ale také statické atributy/metody, čistě virtuální funkce, výjimky, šablony, přetěžování operátorů...
- Až na syntax jsou základy podobné Pascalu.

Definice třídy

objekty se definují stejně jako proměnné

- vypadá jako definice struktury, jen `struct` \Rightarrow `class`.

- Příklad:

```
class spojak{int hod;
spojak*next;};
```

- Lze definovat/deklarovat i funkce:

```
class spojak{
    int hod; spojak*next;
    spojak*pridej(co);
    void uber(int&hod,spojak**nhlava);
};
```

Deklarované funkce definujeme pomocí operátoru čtyřtečky

```
void*spojak::uber(int&vysl,spojak**nhlava)
{
    spojak*pom=this; //pointer na nas j. self
    vysl=hod;
    *nhlava=next;
    delete pom;
}
```

Příklad

jak lze zapsat třídu

```
#include<stdio.h>
class spojak{int hod; spojak*next;
    public:
    spojak*pridej(int hod)
    {    spojak*pom=new spojak();
        pom->hod=hod;
        pom->next=this;
        return pom;
    }
    int uber(int&,spojak**);
};
```

Problém s ochranou paměti

tady totiž narozdíl od Pascalu funguje tak, jak má

- Cokoliv definujeme ve třídě, je privátní.
- Jinak musíme použít přepínač `public` nebo `protected`,
- klademe za ně dvojtečku a jsou to přepínače, tedy zahajují sekci takových atributů/metod.
- Příklad:

```
class spojak{
    int hod; spojak*next;
    public: spojak*pridej(int hod);
           int uber(int&,spojak**);
};
```

Přetěžování funkcí

- V C je funkce určena názvem,
- v C++ je funkce určena názvem a strukturou parametrů,
- `int secti(int,int);` a `int secti(int,int,int);` jsou v C++ dvě různé funkce.
- Hlavní využití je u konstruktorů.

Konstruktory

konstruují, tedy inicializují objekt

- v Pascalu byly dobrovolné, v C++ jsou povinné,
- konstruktor je funkce jmenující se jako dotyčná třída:

```
■ class spojak{...  
    spojak(int,spojak*);  
    spojak(int);//udela 1. prvek  
    spojak(spojak*);//prida nulu  
    ...  
};
```

Konstruktory

podruhé

- Konstruktor je při tvorbě objektu třeba vždy zavolat,
- nedefinujeme-li žádný, vygeneruje se implicitní konstruktory (bez parametrů a copy-konstruktor),
- copy konstruktor má hlavičku `trida(const trida&);`
- používá se implicitně třeba předáme-li objekt hodnotou,
- často se plete s operátorem přiřazení.

Destruktor

je jen jeden, nebere parametry a nic nevrací

- `~trida();...`
`trida::~~trida()`
`{...},`
- volá se, chceme-li objekt zrušit, hodí se, měl-li objekt něco naalokováno.
- Implicitní nedělá nic viditelného.

Alokace a dealokace

objektů

- Z C známe `malloc/free`.
- V C++ (pro objekty) používáme operátory `new` a `delete`.
- Příklad: `hlava=new spojak(5,NULL);`
- do kulatých závorek dáváme parametry konstruktoru.
- Destrukce: `delete hlava;`
- Alokace pole objektů:
`spojak**hlava=new spojak*[10];`
- Dealokace: `delete[] hlava;`
- Pozor, takto nenaalokujeme jednotlivé prvky, je tedy nutné ještě volat jednotlivě:
`hlava[0]=new spojak(0,NULL);...`

Dědičnost

tu také známe z prváku

- `class zivahmota{...};`
- `class clovek:zivahmota{char*jmeno,...};`
- `class clovek:public zivahmota{...};` modifikátor říká, zda mají být zděděné prvky vidět nebo ne. V předchozím příkladu se nedostaneme ani k veřejným prvkům v rodiči. K privátním se nedostaneme nikdy (přístup se při dědění jen omezuje)!
- Implicitní je privátní dědění `private`.

Vícenásobná dědičnost

a diamantový problém – já bych mu řekl jinak a byl by malér

- Dědí se často proto, abychom mohli implementovat různé ovladače událostí,
- někdy chceme jeden ovladač pro více událostí,
- pak dědíme od více tříd:
`class ovladac:ovl_sony,ovl_sharp,ovl_tesla{...}`

- Diamantový problém:
`class A{...}`
`class B:public A{...}`
`class C:public A{...}`
`class D:B,C{...}`
... D zdědí třídu A dvakrát.