Virtuální funkce

aneb třída jako šablona

- Chceme vytvořit vektorové malovátko, které kreslí různé tvary,
- každý tvar se kreslí jinak, ale všechny chceme dát do spojáku.
- Spoják zajistí třída `nakres`(litelny) a chce také vynutit funkci `sezame_nakresli`.
- `class nakres{ public: void sezame_nakresli();};`
- ```
  class bod:public nakres{
        public: void sezame_nakresli();};
  ```
- `nakres*hlava=new bod();...`
- ... a celé to nebude fungovat, protože metody jsou reprezentovány v prototypu.
- Virtuální funkce jsou reprezentovány ve VMT.
- Tvoří se stejně jako v Pascalu:
  `virtual navr_typ jmeno(parametry)...`
  ```

Virtuální funkce

v příkladu malovátko

```
#include <stdio.h>
class nakres
{    public: virtual void sezame_nakresli()
     {    printf("Nic nekreslim!\n"); }
};
class bod:public nakres
{    public: virtual void sezame_nakresli()
     {    printf("Kreslim bod!\n");}
};
int main()
{    nakres*hlava=new bod();
     hlava->sezame_nakresli();
}
```

Statické prvky

sedí ve třídě a ne v objektech

- Někdy chceme, aby atribut (nebo metoda) existovala jen jednou.
- Takové prvky můžeme definovat, aby příslušely přímo třídě...
- ... zvány statické, definují se modifikátorem `static`. Ten se chová (syntakticky) stejně jako `virtual`, tedy:
- statická proměnná:

```
class t{ public: t()
     {    static int citac1=0; citac1++;}
}
```

Statické prvky

- Statický atribut třídy musíme explicitně definovat (nedefinuje se deklarací statického atributu):

```
class t{ static int citac1;
public: t(){t::citac1++}}
int t::citac1=0;
```

- statická metoda:

```
class t{...
     static int kolikrat()
     {    return citac1;
     }
};
...
printf("Konstruovali jsme %d",t::kolikrat());}
```

Friend-funkce

aneb placený zabiják v roli sheriffa

- Občas chceme, aby funkce nepříslušná třídě měla přístup k privátním atributům.
- V tom případě funkci definujeme mimo třídu – např.
  `void kamarad(t a)`
  `{...}`
  a ve třídě jen deklarujeme, že je zpřátelená:
- `friend void kamarad(t);`
- Friend-funkci využijeme především předáváme-li jako parametr svou vlastní třídu a zpřátelená funkce jí přistupuje k privátním atributům.

Čistě virtuální funkce

jsou funkce, které chceme definovat až v synovských třídách

- Motivace: Chceme napsat ovladač od tiskárny, ale každá tiskárna je jiná.
- Tiskárna obecná neexistuje (existují jehličkové, inkoustové, laserové,...) a každá tiskne jinak.
- Chceme rozhraní, které definuje společné požadavky - například metodu `tiskni`.
- Použijeme dědičnost a metoda `tiskni` bude virtuální.
- V rodiči ji nechceme definovat, tak do ní přiřadíme nulu: virtual int tiskni(char*)=0;
- Funkci definujeme až v synech.
- Takové funkci říkáme *čistě virtuální.*

Příklad

s tiskárnami

```
class tisk
{    public: virtual int tiskni(char*)=0;
};
class ltisk:public tisk
{    public: virtual int tiskni(char*co)
     {    printf("Tisknu laserove: %s\n",co);}
};
class itisk:public tisk
{    public: virtual int tiskni(char*co)
     {    printf("Tisknu inkoustem: %s\n",co);}
};
...
```

Abstraktní třída

byla na předchozím slildu

- Třída, která obsahuje aspoň jednu čistě virtuální funkci, se nazývá abstraktní.
- Nelze vytvořit instanci této třídy,
- že je třída abstraktní, překladač pozná.
- Takové třídy můžeme používat jako tzv. interface,
- tedy řekneme, co musí každá synovská třída definovat.
- Praktické použití: často omílané vektorové malovátko (a typ `drawable`).

Přetěžování operátorů

dělá něco podobného jako přetěžování funkcí

- Chceme implementovat komplexní čísla, uděláme tedy třídu s reálnou a imaginární složkou.
- Čísla chceme sčítat, odčítat,... jako jiné číselné typy (ne `secti(kom a, kom b);`).
- Přetížený operátor definujeme jako funkci s divným jménem, např. `operator=`, `operator+`,...

Příklad

-

```
kom& kom::operator=(const kom A)
{    this->re=A.re;this->im=A.im;
     return *this;
}// pozor na operatory copy- a move-assign...

kom kom::operator+(const kom B)
{    kom C;
     C.re=re+B.re;
     C.im=im+B.im;
     return C;
}
```

Proudy

a využití přetížených operátorů

- Načítáme pomocí `getchar` a vypisujeme pomocí `printf`,
- v C++ je možnost použít streamy (otevřít soubor jako stream)
- Příklad:

```
ofstream vystupni;
vystupni.open("vystup.txt");
...
```

- Standardní vstup/výstup reprezentován streamy `cin, cout`.
- Soubor je pak reprezentován objektem (kterému lze volat metody).

Proudy II

temné

- Načítání a výpis:

```
char data[100];
cin>>data;
cout<<data<<endl;
```

- Ovšem << resp. >> známe, byly to operátory bitového posunu.
- Tady jsou tyto operátory přetížené.

Code snippets

specifikum Visual Studia

- Jsou úkony, které děláme opakovaně (například for-cyklus s cyklící proměnnou `i`).
- Proto stačí napsat `for` a stisknout tabelátor.
- Anebo často definujeme třídy. Třída `MyClass` s (implicitními) konstruktory a destruktorem stisknutím tabelátoru po napsání `class`.
- Podrobnosti s kolegou Holanem (pozor, ač snippety fungují podobně, do jisté míry se liší).