Motivace C++

C++

- nová klíčová slova a mechanizmy
- rozšiřuje programovací možnosti C (neobjektové vlastnosti)
- jazyk vyšší úrovně
- přidává objektové vlastnosti (program objektově orientován, lze psát i "standardně")
- existují nezávislé normy C a C++ - C (1999, C1X) a C++ (1998/2003, C++0x,C++11)
- přenositelný kód
- C může mít některé vlastnosti navíc, až na výjimky lze říci, že C je podmnožinou C++
- C přijímá některé (neobjektové) vlastnosti z C++
- dědění – znovupoužití a rozšíření/modifikace kódu
- šablony – znovupoužití kódu (pro různé datové typy)
- výjimky – nový způsob ošetření chyb

Neobjektové vlastnosti

- lze použít i samostatně, hlavní využití u práce s objekty (vlastními typy)
- přetěžování funkcí a operátorů
- definice proměnné
- reference
- implicitní parametry
- prostory jmen
- typ bool
- alokace paměti (new, delete) – návaznost na inicializaci a rušení objektu
- typově orientovaný vstup, výstup
- inline funkce
- šablony

Třída

- umožňuje objektové programování – nové možnosti a nový styl programování
- nový složený typ (odvozena od struct)
- spojení dat a funkcí/metod pro práci s nimi + ochrana dat (přístupová práva) - výsledný mechanizmus nazýváme zapouzdřením
- zlepšuje "kulturu" programování – inicializace, ukončení života proměnné, chráněná data …
- umožňuje znovupoužití kódu, tvorbu společných rozhraní (šablony, dědění, polymorfizmus…)
- nejblíže k ní má knihovní celek z jazyka C – rozhraní, data, kód
- výhody: tvorba knihoven, sdílení kódu, údržba programu

- zdrojový kód (nejčastěji) přípona ”cpp”. (hlavičkové soubory bez přípony nebo ”h”, ”hpp”, ”hxx”)

- třída je obdobně jako struktura **popisem** vlastností a velikosti nového typu.
- definujeme-li proměnnou daného typu, je jí rezervováno místo v paměti. U třídy nehovoříme o proměnné ale spíše o objektu, nebo o instanci
- funkce přístupné/nabídnuté k použití uživateli tvoří rozhraní třídy (přes které se s ní komunikuje)

Rozbor úlohy

- formulace (definice) problému – slovní popis
- rozbor problému – vstupní a výstupní data, operace s daty
- návrh dat (vnitřní datové struktury)
- návrh metod (vstupy, výstupy, "výpočty"/operace, vzájemné volání, rozhraní, předávaná data)
- testování (modelové případy, hraniční případy, vadné stavy …)

Rozbor problému – koncepce programu

- konzultace možných řešení, koncepce
- rozhodneme, zda je možné použít stávající třídu, zda je možné upravit stávající třídu (dědění), zda vytvoříme více tříd (buď výsledná třída bude obsahovat jinou třídu jako členská data, nebo vytvoříme hierarchii – připravíme základ, ze kterého se bude dědit – všichni potomci budou mít shodné vlastnosti). (Objekt je prvkem a objekt dědí z …) – relace má (jako prvek) a je (potomkem-typem)
- pohled uživatele (interface), pohled programátora (implementace)

- dědění
- polymorfizmus
- šablony
- výjimky

Formulace problému – konkrétnější specifikace prvků programu

- co má třída dělat – obecně
- určení požadované přesnosti pro vnitřní data
- jak vzniká (->konstruktory)
- jak zaniká (->destruktor)
- jak nastavujeme a vyčítáme hodnoty (->gettery a settery – data jsou soukromá/nedosažitelná pro uživatele)
- jak pracujeme s hodnotami (->metody a operátory)
- vstup a výstup

Návrh datové struktury

- zvolí se data (proměnné a jejich typ) které bude obsahovat, může to být i jiná třída
- během dalšího návrhu nebo až při delší práci se může ukázat jako nevyhovující
- Data jsou (většinou) skrytá

Navrhněte datovou strukturu (členské proměnné/atributy) pro třídu komplexních čísel.

Pro třídu komplexních čísel se nabízí dvě realizace dat:

- Reálná a imaginární složka
- Amplituda a fáze (délka a úhel, …)

První verze je výhodná pro operace jako sčítání, druhá pro násobení.
Budeme více násobit nebo sčítat? Obecně nelze říci -> reprezentace jsou rovnocenné.

Dále ve třídě/objektu můžeme mít datový člen pro signalizaci chybového stavu – minulý výpočet se nezdařil (dělení nulou …)

Návrh metod

- metoda – funkce ve třídě pro práci s daty třídy
- metody vzniku a zániku = konstruktor a destruktor
- metody pro vstup a čtení dat = gettery a settery
- metody pro práci s objektem
- operátory
- vstupy a výstupy
- metody vzniklé implicitně (ošetřit dynamická data)
- vnitřní realizace (implementace) metod - zde se (hlavně) zužitkuje C – algoritmy
- to jak je třída napsaná (jak vypadá ona a metody uvnitř) nazýváme implementací

Navrhněte metodu/funkci, která vrátí reálnou část komplexního čísla (pro obě reprezentace dat)

```
double Real()
{
  return iReal;
}

double Real()
{
  return iAmpl * cos( iUhel );
}
```

V případě, že uživatel nepracuje přímo s datovými atributy třídy (iReal, iAmpl, iUhel), potom při změně (implementace/realizace) vnitřních parametrů uživatel rozdíl nepozná, protože s třídou komunikuje přes metody/funkce, které jsou veřejně přístupné a které tvoří rozhraní/interface mezi atributy třídy a uživatelem

Testování

- na správnost funkce
- kombinace volání
- práce s pamětí (dynamická data)
- vznik a zánik objektů (počet vzniků = počet zániků)
- testovací soubory pro automatické kontroly při změnách kódu

Příklad:
Napište testovací soubor tester.bat pro testování programu progzk.exe tak, že mu předložíte vstupní soubor a jeho výstup porovnáte s výstupem očekávaným. V případě chyby vytiskněte hlášení na konzolu

Část testovacího souboru (**tester.bat** - Windows) pro jedny vstupní parametry

```
progzk.exe  <input1.dat   >output1.dat
fc output1.dat vzor1.dat
if ERRORLEVEL == 1 echo "Program s input1 vratil chybu"
```

nebo pro UNIX/LINUX vložte následující obsah do souboru: **tester.sh**

```
#!/bin/sh
progzk.exe <input1.dat >output1.dat
diff output1.dat vzor1.dat
if [ $? -ne 0 ] ; then
 echo "Program s input1 vratil chybu."
fi
```

Nastavte soubor jako spustitelný…

```
chmod u+x tester.sh
```

A spusťte soubor **tester.sh** z lokálního adresáře

```
./tester.sh
```

Základní pojmy Objektového programování (shrnutí):

– **Třída** *(Class)* - Nový datový celek (datová abstrakce) obsahující: data (atributy / složky) a operace (metody), přístupová práva

– **Instance** *(Instance)* - realizace (výskyt / exemplář) proměnné daného typu.
– **Objekt** *(Object)* - instance třídy (proměnná typu třída).

– **Atribut** (Member attribute / member variable) - data / proměnné definované uvnitř třídy. Často se používá i pojem složka.
– **Metoda** *(Method / Member function)* - funkce definovaná uvnitř třídy. Má vždy přístup k datům třídy a je určena pro práci s nimi.

– **Operátor** *(Operator)* - Metoda umožňující zkrácený zápis svého volání pomocí existujících symbolů. (součet +, podíl /, apod.)

– **Zapouzdření** *(Encapsulation)* – shrnutí logicky souvisejících (součástí programu) do jednoho celku (zde atributy, metody, přístupová práva) – nového datového typu třída. Ve volnější pojetí lze používat i pro funkce a proměnné definované v rámci jednoho .c souboru. Někdy je tímto termínem označováno skrytí přímého přístupu k atributům a některým metodám třídy *(private members)*. Volně dostupné vlastnosti se označují jako veřejné *(public members)*.

– **Životní cyklus objektu** *(Object live cycle)* - Objekt jako každá proměnná má definováno místo vzniku (definice/inicializace) a místo zániku.

– **Konstruktor / Destruktor** *(Constructor / Destructor / c'tor / d'tor)* - metody které jsou v životě objektu volány jako první resp. jako poslední. Slouží k inicializaci atributu resp. k jejich de inicializaci objektu.

– **Rozhraní** *(Interface)* - Seznam metod, které jsou ve třídě definovány jako veřejné a tvoří tak rozhraní mezi vnitřkem a vnějškem třídy.

– **Implementace** *(Implementation)* - Těla funkcí a metod (tj. kód definovaný uvnitř funkce / metody).

– **Dědičnost** *(Inheritance)* - Znovupoužití kódu jedné třídy (předka) k vytvoření kódu třídy nové (potomka). Nová třída (potomek) získá všechny vlastnosti (atributy, metody) z potomka a může definovat libovolnou novou vlastnosti nové.

– **Mnohotvarost** *(polymorfism)* - Třídy se stejným rozhraním, a různou implementací, jednotný přístup k instancím. Mechanismus umožňující volání metod potomka z metod předka.