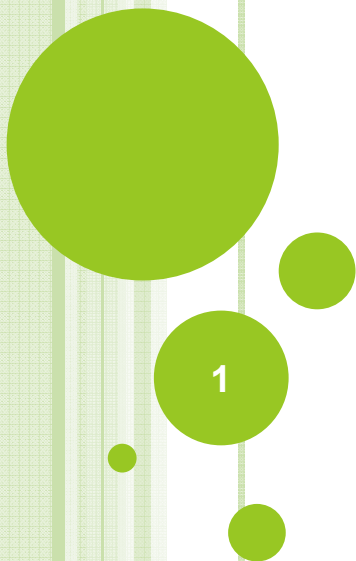


PB161 – PROGRAMOVÁNÍ V JAZYCE C++ OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ



1

Úvod, organizace, myšlenka OOP, nástroje

PB161 – Programování v jazyce C++

Objektově Orientované Programování

Úvod, organizace, myšlenka OOP,
nástroje, string

Cíle předmětu

1. Vysvětlit základy objektově orientovaného programování
2. Seznámit s možnostmi jazyka C++
3. Zavést a podpořit praktické programátorské schopnosti
4. Trochu nadchnout (nebo alespoň úplně neodradit) od programování 😊

Co nás dnes čeká

- Organizační
- OOP a C++ v kontextu
- Ukázka příkladu objektově orientovaného programování (OOP)
- (Překladače, IDE, verzovací nástroje...)

Organizační



I THOUGHT PB071 WILL BE EASY

**Well, PB161 will not
be easy either 😊**

NEVER HAVE I BEEN SO WRONG

MEMECREATOR.EU

Organizační (1)

● Přednášky

- nepovinné, ale snad přínosné a zábavné ☺
- jedna na [vnitrosemestrální test \(28.10.? – upřesníme příště dle vás\)](#)
- zvané přednášky (ke konci semestru, bude upřesněno)
- rozcestník <http://cecko.eu/public/pb161>

● Cvičení

- povinné, dvouhodinové, dvě neúčasti tolerovány
- aktivní práce na příkladech a domácích úkolech, konzultace
- průběžné testíky (přímo na hodině, za 3 body max.)
- http://cecko.eu/public/pb161_cviceni

● Ukončení předmětu

- [zápočet](#) – úkoly + průběžný test + testíky + bonusy, zisk alespoň 65 bodů + úspěšné vypracování zápočtového příkladu na hodině
- [zkouška](#) – zápočet + zkouškový test, zisk alespoň 95 bodů

Organizační (2)

● Materiály

- slidy, ukázkové zdrojáky
- http://cecko.eu/public/pb161_cviceni
- záznam přednášek v ISu (cca 2 denní zpoždění)

● Domácí úkoly

- 5+1 za semestr, zadávány průběžně (na webu cvičení)
- body za funkčnost, body za správné odevzdání
- deadline pro odevzdání (na stránce úkolu, 2 týdny)
- budou zveřejňována ukázková řešení

● Odevzdání/testování

- možnost odevzdání nanečisto (detaily na cvičení)
- odevzdání do fakultního SVN, spuštění notifikačního skriptu
- 12 bodů max. + bonusy (poměr 9 funkčnost, 3 odevzdání)
- strhávání fixních bodů při nalezení chyby
- max. 3 pokusy na odevzdání

Kontakt

- Přednášející

- Petr Švenda, svenda@fi.muni.cz
- Konzultační hodiny: Pondělí 15-15:50 A406
 - Centre for Research on Cryptography and Security
 - (Laboratoř bezpečnosti a aplikované kryptografie)

- Cvičící

- na hodinách – využívejte hojně

- Studentští konzultanti

- dodatečné konzultace nezávisle na skupině
- v definované konzultační hodiny (viz. hlavní web)

Neopisujte

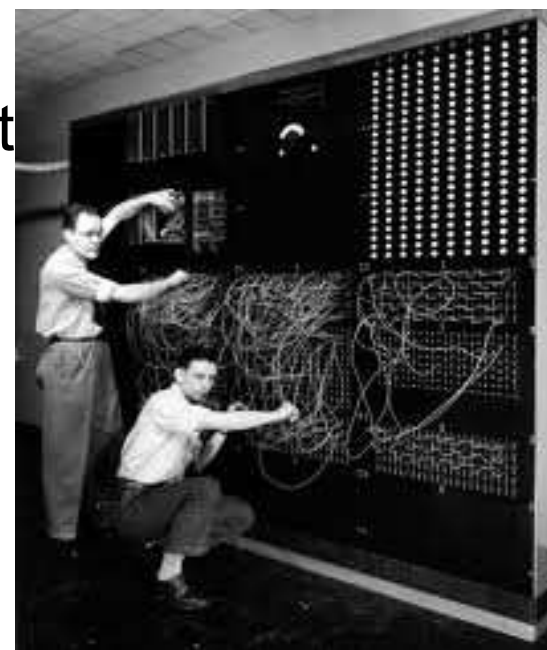


- Škodíte především sami sobě
 - začněte pracovat včas, ať máte čas na řešení "záseků"
- Provádíme automatickou kontrolu opisu u všech odevzdaných příkladů
 - každý s každým
 - každý s řešeními z minulých let (pokud je podobný příklad)
 - u podezřelých příkladů probíhá manuální kontrola
- V případě opsání jsou potrestáni oba účastníci
 - 0 bodů, -5 bodů jako potrestání

Novinky

- Nově připravené podklady pro cvičení
 - Piště prosím veškeré připomínky
- Možnost párového programování na cvičení
 - dva studenti sdílí jeden počítač
 - nepovinné, záleží na vaší chuti
 - je to především zábava a zkušenost
- Zveřejňování slidů i v ppt
 - kvůli animacím
 - zveřejnění předběžných slidů

Harvard Mark 1



Seznam předmětů s programováním na FI

- Seznam předmětů s programováním na FI
 - <http://www.cecko.eu/public/code@fimu>
- Seznam předmětů, kde se alespoň částečně programuje
- Rozděleno dle jarního a podzimního semestru
- Plán:
 - doplnění relevantních MOOC kurzů
 - Coursera, EdX...
 - absolvovali jste nějaký dobrý? Pište prosím na svenda@fi.muni.cz

Sběr zpětné vazby

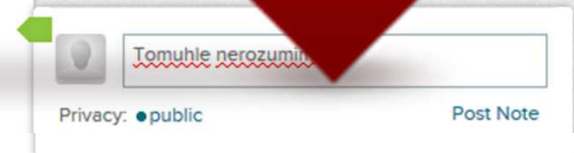
- Předmětová anketa
 - vyhodnocení PB071 jaro 2014:
http://cecko.eu/public/pb071_hodnoceni_jaro2014
- Velké poděkování všem za množství poznámek a námětů!
- Občasné dotazníčky (obtížnost úloh, porozumění...)
- Samozřejmě možné osobně

Demo - Scripd

- Snadné vkládání poznámek od studentů do slidů
 - login přes Facebook n
- Typo nebo chyby
- Co vám není jasné, chcete v
 - čím více tagů, tím větší šance
- ...

DEMO - SCRIPD

- Snadné vkládání poznámek od studentů do slidů
- Co vám není jasné, chcete více vysvětlit
- Typo nebo chyby
- Náměty na rozšíření
- ...



Úvod do C++, 16.9.2013

Proč C++?

15

C++ v kontextu

- Historie C++
 - Bjarne Stroustrup, 1979 (C with classes)
 - ANSI/ISO C++ standard, 1998
 - <http://flinflon.brandonu.ca/dueck/1997/62285/stroustrup.html> :)
- Imperativní, staticky typovaný jazyk
 - zdrojový kód kompilovaný do nativního kódu platformy
- Orientací mezi C a Java/C#
- Až na drobné výjimky je C podmnožina C++
 - C kód je typicky validní C++ kód
- Srovnání C++ s dalšími jazyky podrobněji na poslední přednášce

Vhodnost použití C++

- Proč používat?
 - široké rozšíření (standardizační komise byla zaskočena ☺)
 - typicky vysoká rychlost kódu (shootout.alioth.debian.org)
 - objektově orientovaný jazyk, generické programování (šablony)
- Vhodné využití pro:
 - větší projekty
 - systémové aplikace
 - rychlá grafika, rychlost obecně
 - nové problémy (jazyk příliš neomezuje)
- Spíše nevhodné pro
 - webové aplikace (Python, PHP, C#...)
 - rychlé prototypy (ale nutno znát dobře jiný jazyk)

17

Srovnání rychlostí – práce s poli

reverse-complement benchmark ~240MB N=25,000,000

This table shows 5 measurements - CPU Time, Elapsed Time, Memory, Code and ~ CPU Load.

		sort	sort	sort	sort	
×	Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	~ CPU Load
1.0	C++ GNU g++ #4	1.12	1.12	245,432	2275	1% 0% 1% 100%
1.1	ATS	1.18	1.19	122,628	2077	1% 0% 1% 99%
1.2	C++ GNU g++ #2	1.35	1.35	245,080	1098	0% 0% 1% 100%
1.2	C GNU gcc #4	1.38	1.38	125,188	722	0% 0% 2% 100%
1.5	Ada 2005 GNAT #2	1.68	1.70	197,584	3132	1% 0% 1% 99%
1.6	C++ GNU g++ #3	1.75	1.75	125,272	810	0% 0% 1% 100%
2.1	Scala #4	2.37	2.39	400,184	505	0% 0% 0% 99%
2.1	Pascal Free Pascal #2	2.39	2.39	123,816	751	0% 0% 0% 100%
2.6	Java 6 -server #4	2.86	2.90	473,280	592	0% 1% 0% 99%
2.7	C# Mono	3.06	3.05	161,548	1099	0% 0% 0% 100%
3.6	Haskell GHC #2	4.02	4.02	618,032	913	0% 0% 0% 100%
3.8	C++ GNU g++	4.30	4.30	245,388	571	3% 6% 1% 100%
3.9	Lisp SBCL	4.40	4.41	222,396	896	0% 0% 0% 100%
4.3	OCaml #2	4.78	4.78	168,920	394	0% 0% 0% 100%
5.2	Perl #4	5.80	5.80	124,036	237	0% 0% 1% 100%
6.3	PHP #2	7.00	7.00	444,456	343	0% 0% 0% 100%

Srovnání rychlostí – matematické operace

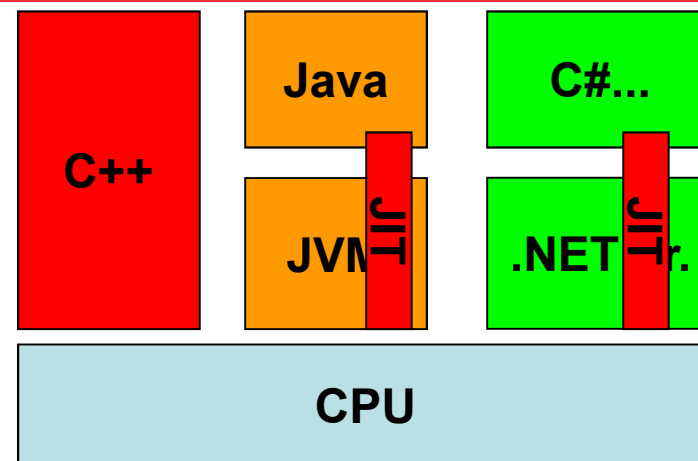
spectral-norm benchmark N=5,500

This table shows 5 *measurements* - CPU Time, Elapsed Time, Memory, Code and ~ CPU Load.

		sort	sort	sort	sort	
×	Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	~ CPU Load
1.0	C GNU gcc #4	11.87	2.99	772	1139	99% 100% 99% 99%
1.0	C++ GNU g++ #7	11.89	2.99	1,196	1114	100% 99% 99% 99%
1.3	Ada 2005 GNAT #3	15.69	3.98	2,528	1702	98% 99% 98% 99%
1.3	Fortran Intel	15.93	4.00	1,276	568	99% 99% 100% 100%
1.4	Haskell GHC	16.02	4.11	2,260	869	96% 99% 96% 99%
1.4	Java 6 steady state #2	17.14	4.31	24,620	1027	99% 99% 99% 99%
1.5	Java 6 -server #2	17.33	4.51	15,208	950	98% 95% 94% 97%
1.5	Scala #2	17.66	4.56	20,720	720	96% 96% 97% 98%
1.6	Ada 2005 GNAT #2	18.75	4.74	3,012	1464	99% 99% 99% 98%
1.9	C# Mono #2	22.31	5.63	5,104	1063	99% 99% 99% 99%
2.0	ATS #2	22.06	5.96	1,656	2339	92% 92% 93% 93%
2.0	Lisp SBCL #3	22.22	6.01	7,476	883	92% 92% 93% 93%
2.1	OCaml #3	20.02	6.21	3,304	907	79% 79% 81% 81%
2.2	Go 6g 8g #2	26.40	6.63	6,672	545	99% 100% 100% 100%
4.1	Erlang HiPE #2	47.70	12.18	13,348	747	98% 98% 97% 98%

C++ a další

- C++
 - překlad přímo do strojového kódu
 - překlad nutný zvlášť pro každou platformu
- Další imperativní: Java, C#...
 - překlad do mezi jazyku bytecode/CIL, jedna binárka pro všechny platformy
 - (Java Virtual Machine) JVM pro velké množství platforem
 - bytecode interpretovaný, ale JIT (Just-In-Time) kompilátor
- Skriptovací imperativní: Perl, Python...
 - nemusí překlad, přímo se interpretuje, platformově nezávislé (interpret)
- Funcionální: Haskell, LISP...
 - jiné paradigma: NE jak dosáhnout výsledku postupnou změnou proměnných, ale matematický zápis odvození z počátečních hodnot
- Logické programování: Prolog...
 - jiné paradigma: JAK má výsledek vypadat, ne jak se k němu dostat



Normy, standardy a rozšíření

- Kniha The C Programming Language (1978)
 - neformální norma pro C
- Bjarne Soustrup, práce na 'C with Classes' (1979)
- B.Soustrup, kniha The C++ Programming Language (1985)
- ISO/IEC 14882:1998 (C++98, -std=c++98 ~ -ansi)
 - g++ -ansi
 - budeme využívat jako default při psaní
- ISO/IEC 14882:2003 (drobné rozšíření, C++03)
- ISO/IEC 14882:2011 (C++11, -std=c++11)
 - <http://en.wikipedia.org/wiki/C++11>
 - bude věnována jedna zvaná přednáška
- ISO/IEC 14882:2014 (nejnovější, drobné rozšíření)
 - <http://en.wikipedia.org/wiki/C%2B%2B14>
 - <https://isocpp.org/blog/2014/08/we-have-cpp14>

Nestandardizovaná rozšíření

- Nestandardizované rozšíření
 - užitečné prvky jazyka dosud neobsažené v normě
 - specificky označeny a dokumentovány
- Problém: využívání vede k omezení přenositelnosti
 - pro jinou platformu nelze překompilovat bez změny kódu
 - omezuje dostupnost programu
 - zvyšuje cenu přechodu na jinou platformu (customer lock-in)
- Proč psát programy v souladu s normou?
 - lze přímo kompilovat pro jiné platformy - svoboda volby platformy
 - svoboda volby kompilátoru a odolnost vůči jeho změnám
 - větší potenciální využití kódu (i jiné projekty/překladače)
 - norma může omezit problematické jazykové konstrukce (nižší chybovost)

Proč C++ - strukturované programování

- Příklad s prioritní frontou procesů z PB071
 - spojovaný seznam v C, vkládání, třídění podle priority

C verze

ukazatele na sousední položky, přepojování, složité procházení...

C++ verze

```
typedef struct priority_queue_item { value_type value; uint remaining_time; } item;
bool compareAsc(item first, item second) {
    return first.remaining_time < second.remaining_time;
}
typedef struct process_queue {
    list<item> processQueue;
    void push(item newEntry) {
        processQueue.push_back(newEntry);
    }
    void sortByPriority() {
        processQueue.sort(compareAsc);
    }
} process_queue;
```



**Lze ještě snáze s využitím
STL priority_queue**

23

Proč C++ - Příklad s generováním SVG

- Formátování XML tokenů

C verze

```
char shapeTag[strlen(targetShapeName) + strlen("< ") + 1];  
memset(shapeTag, 0, sizeof(shapeTag));  
sprintf(shapeTag, "<%s", targetShapeName);
```

C++ verze

```
string shapeTag = "<" + targetShapeName;
```

- Výrazně kratší a přehlednější kód
- Základní knihovna výrazně rozšířena
 - stačí chápat základní princip a umět dohledat detaily

24

O co jde v Objektově Orientovaném Programování?

25

Motivace pro OOP

- V našich programech obvykle modelujeme svět
- Svět je objektový
- Modelovat objekty **objekty** je tedy přirozené

Svět je opravdu objektový!



27

Základní představa OOP

- Všechno je objekt
- Objekty komunikují pomocí zpráv
- Objekty mají interní stav



www.ubergizmo.com

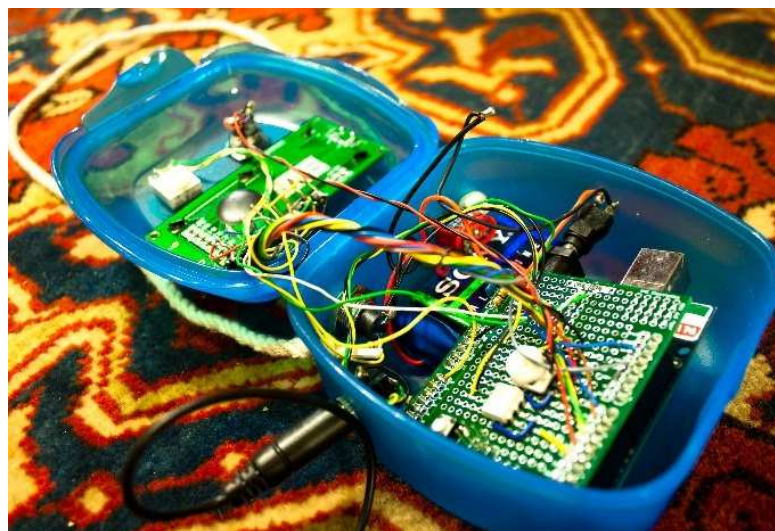


28

Principy OOP - Zapouzdření



- Objekt působí navenek jako „black box“
- Zaměření na to co objekt dělá
- Uživatel by neměl znát „vnitřnosti“



Zapouzdření – způsob komunikace



- Jedinou povolenou komunikací jsou zprávy
- I zpráva je objekt (může mít parametry)
- Pokud objekt zprávě rozumí, musí ji přijmout
- Pokud nerozumí musí ji odmítnout
- Množina zpráv, kterým objekt rozumí, definuje rozhraní
- Pořadí zpráv definuje protokol

Hello World v C++

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello world!";
    cout << 12345;
    cout << 8-);
    return 0;
}
```

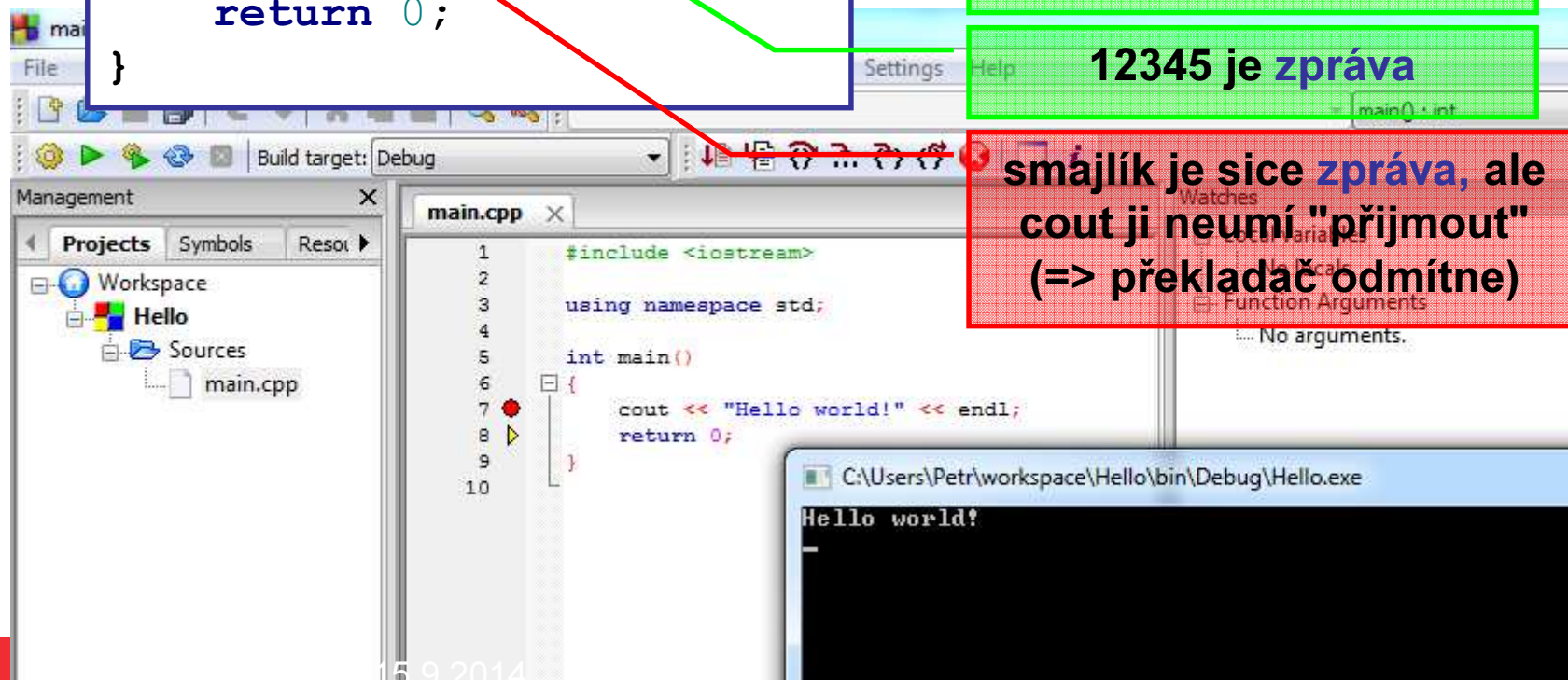
cout je **objekt**
standardního výstupu

"Hello world!" je **zpráva**

<< je operátor zaslání
zprávy (funkční volání)

12345 je **zpráva**

smajlík je sice **zpráva**, ale
cout ji neumí "**přijmout**"
(=> překladač odmítne)



Principy OOP - Polymorfismus



- Různé objekty mohou reagovat na tu samou zprávu
- Jejich reakce ale mohou být různé
- Z pohledu odesilatele jsou přijímací objekty zaměnitelné



32

V čistě OOP programovacím jazyku

- Všechno je objekt
- Objekty komunikují pomocí zpráv
- Každý objekt má předka
- Smalltalk
 - see <http://www.objs.com/x3h7/smalltalk.htm>
- (*C++ není čistě OOP*)

Objektově orientované myšlení

- OOP je především způsob přístupu k řešení problému
- Cílem je zvětšit flexibilitu kódu a produkovat robustnější kód
- OOP můžeme použít i v neobjektových jazycích
- Objektově orientované jazyky ale nabízejí výraznou syntaktickou podporu

Zapouzdření
Abstrakce
Dědičnost
Polymorfismus

Zapouzdření

- Zapouzdření je styl programování
 - snaží se minimalizovat viditelnost proměnných/funkcí
 - uživatel používá jen vybranou podmnožinu funkcí
 - aby nedocházelo k nezáměrným/nevhodným změnám
 - aby nebyl uživatel svázán implementačními detaily
- Kombinace několika vlastností
 - **abstrakce dat/metod** (abstraction)
 - **skrytí dat/metod** (hiding)
 - **zapouzdření dat** (data encapsulation)

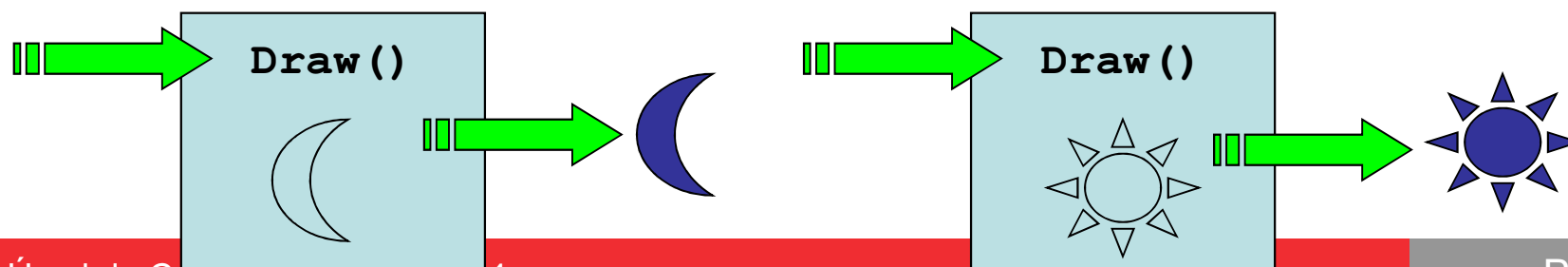
Abstrakce dat/metod (abstraction)

● Datová abstrakce

- data mohou být použita, aniž by uživatel znal způsob jejich reprezentace v paměti
- např. databáze jako soubor na disku nebo vzdáleném serveru (sqlite vs. MySQL database)

● Funkční abstrakce

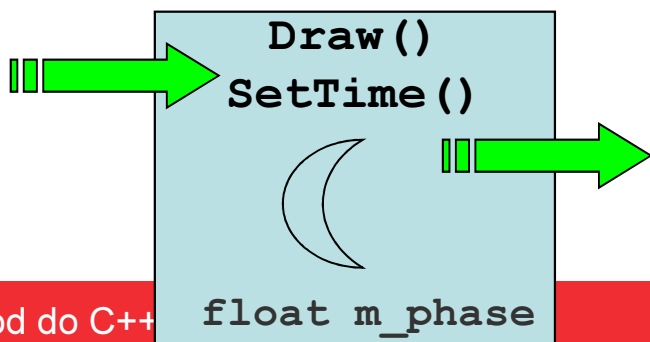
- metoda může být použita, aniž by uživatel znal způsob její implementace
- např. metoda pro vykreslení objektu Draw()



37

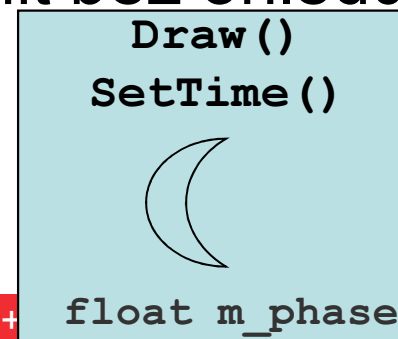
Skrytí dat/metod (hiding)

- Pro okolí jsou vnitřní data (atributy) třídy skryta
 - atribut třídy může být pro okolí nepřístupný
- Nemusejí být ani metody pro přímé získání a nastavení hodnoty atributu
 - atribut se může projevovat jen vlivem na chování ostatních funkcí
- Pro okolí jsou interní metody skryty
 - nejsou součástí veřejného "rozhraní" třídy



Zapouzdření dat (encapsulation)

- S daty lze pracovat jen prostřednictvím **obalujících funkcí**
 - nelze přímo číst/nastavit hodnotu datového atributu
 - např. dostupná pouze `Object::SetTime()`
- Obalující metody mohou kontrolovat
 - validitu argumentů, konzistentnost vnitřního stavu před změnou... (Např. je nastavovaný čas korektní?)
 - metoda může nadstavovat více vnitřních atributů zároveň (Např. daný čas i fázi Měsíce)
- Uživatel třídy není omezován detaily vnitřní logiky
- Implementátor třídy může změnit bez ohledu na uživatele třídy
 - (pokud dodrží rozhraní)



Výhody zapouzdření

- Implicitní tlak na **dělitelnost a nezávislost** kódu
- Implementace třídy se může vyvíjet bez nutnosti změny okolního kódu
 - rozhraní zůstává neměnné
 - např. metoda Draw() přidá detailnější vykreslení
- Vnitřnosti třídy jsou lépe chráněny vůči chybám
 - programátor je omezen a kontrolován překladačem
 - (nastavení `m_phase` mimo objekt je syntakt. chyba)
- Příprava pro další OOP vlastnosti

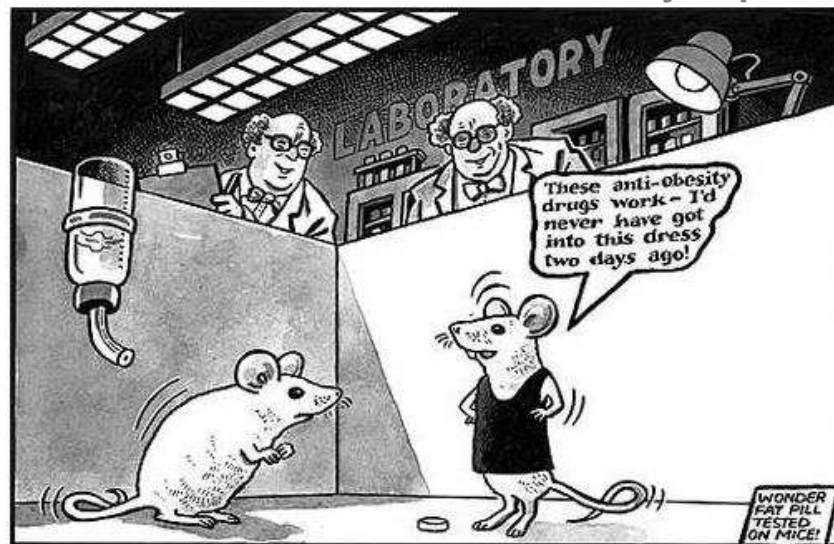
Ukázka příkladu objektově orientovaného programování

41

Laboratoř - zadání

V laboratoři jsou pěstované různé druhy zvířat. V tuto chvíli pouze myši a pavouci. Zvířata se pohybují po ohraničeném prostoru, ve kterém se nachází potrava. Pokud se k ní dostanou, sežerou ji a vyrostou v závislosti na druhu zvířete.

Paul Thomas in The Daily Express



42

Laboratoř - zadání

V laboratoři jsou pěstované různé druhy zvířat. V tuto chvíli pouze myši a pavouci. Zvířata se pohybují po ohraničeném prostoru, ve kterém se nachází potrava. Pokud se k ní dostanou, sežerou ji a vyrostou v závislosti na druhu zvířete.

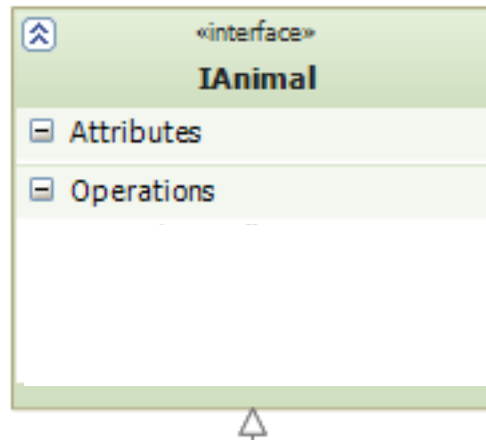
Laboratoř – řešení v C

- Struktura *struct* pro každý druh zvířete a potravy
- Dynamická alokace struktury pro každého jedince
- Zřetězený seznam jedinců v teráriu
- Funkce pro pohyb, nakrmení...
 - jedinec jako parametr funkce

Laboratoř – řešení pomocí OOP v C++

- Abstraktní představa zvířete (IAnimal)
 - obecný předek všech zvířat
 - předpokládáme, že budou společné vlastnosti
 - (v Javě interface, v C++ abstraktní třída)
- Nová třída *class* pro každý druh zvířete
 - potomek IAnimal, např. CMouse a CSpider
- Metody pro pohyb, nakrmení, interakci...
 - součást přímo třídy, ne jako samostatné funkce
- Obecnější chování je součástí předka
 - např. pohyb a detekci potravy lze do předka

45

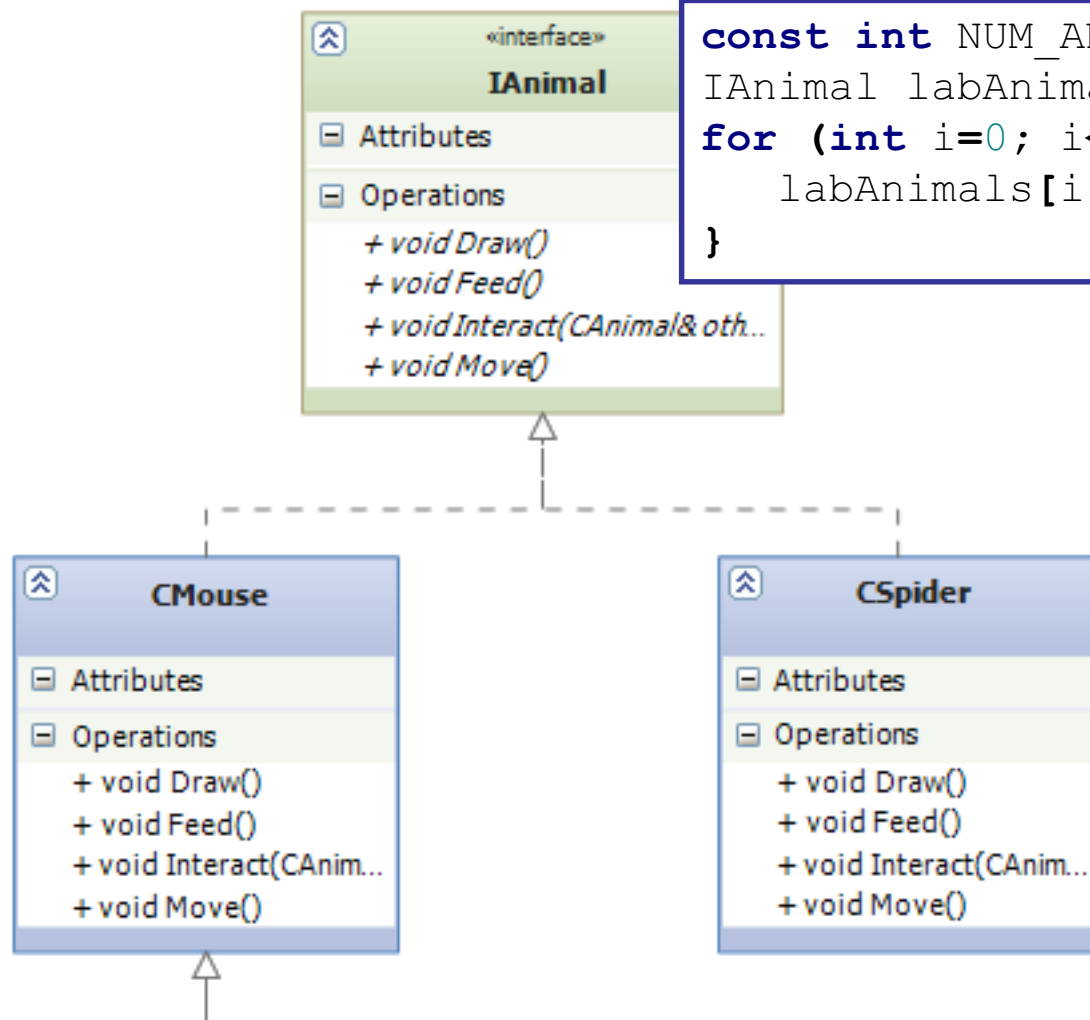


```
const int NUM_ANIMALS = 100;
IAnimal labAnimals[NUM_ANIMALS];
for (int i=0; i<NUM_ANIMALS; i++) {
    labAnimals[i].Draw();
}
```

Jak byste řešili v C?

Laboratoř – dodatečná změna zadání

- Zadavatel rozšíří původní zadání:
 - máme dva druhy myši – domácí (větší) a polní (menší)
- Řešení v C
 - tvorba nových struktur houseMouse a fieldMouse
 - většina chování (== kódu) se u nich opakuje
 - provedeme refactoring manipulačních funkcí
- Řešení v C++
 - vytvoříme dvě nové třídy CHouseMouse a CFieldMouse
 - obě potomci třídy CMouse
 - `IAAnimal->CMouse->CHouseMouse`
 - společné chování necháme v předkovi CMouse
 - do CHouseMouse (CFieldMouse) dáme jen odlišnosti od CMouse

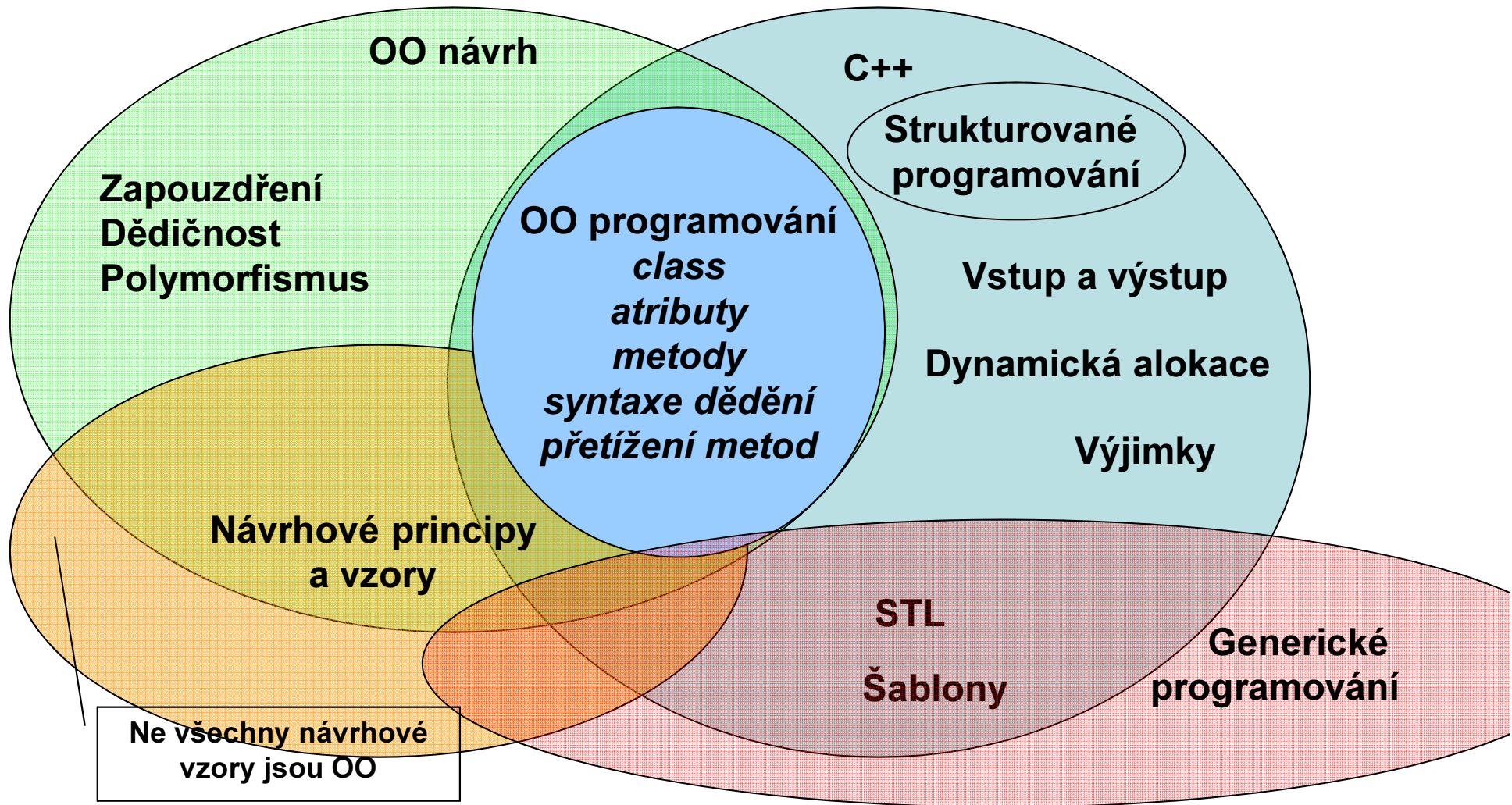


```
const int NUM_ANIMALS = 100;
IAnimal labAnimals[NUM_ANIMALS];
for (int i=0; i<NUM_ANIMALS; i++) {
    labAnimals[i].Draw();
}
```


Laboratoř – dokončení

```
const int NUM_ANIMALS = 100;
IAntimal labAnimals[NUA_NUMANIMALS];
for (int i=0; i<NUM_ANIMALS; i++) {
    labAnimals[i].Draw();
}
```

- OOP umožňuje snazší rozšiřitelnost kódu
 - pokud je dobře navrženo!
- Nemusíme modifikovat existující kód
 - je odzkoušený – nemusíme jej znovu testovat
 - mohou ho používat i jiné programy – přestane jim fungovat
 - přidáváme potomky, kteří upravují původní funkčnost
- Máme možnost přidat chování pro všechny potomky
 - např. přidání funkce pro páření v CMouse je zároveň dostupné v CMouseHouse i CFieldMouse
- Výhody OOP se projevují především ve větších a potenciálně rozšiřovaných programech



Nástroje
Kompilátor (gcc), IDE (QT, Visual Studio, NB..), Debugger, Verzování (SVN, GIT...), Dokumentace (Doxygen), Testy (QTest, CxxTest)

50

`std::string`

51

Motivace pro chytřejší řetězce

- Práce s C řetězcí není nejsnazší
 - musíme používat speciální funkce pro manipulaci
 - musíme hlídat koncovou nulu
 - musíme hlídat celkovou velikost pole
- Chtěli bychom
 - nestarat se o velikost (zvětšení, zmenšení)
 - používat přirozené operátory pro manipulaci
 - mít snadnou inicializaci, kopírování

STL `std::string`

- C++ nabízí ve standardní knihovně třídu `std::string`
 - přesněji, jde o instanci šablony třídy (viz. později)
- Umístěno v hlavičkovém souboru `<string>`
 - pozor, `<string.h>` je hlavička pro C řetězec, ne pro string
- Jedná se o kontejner obsahující sekvenci znaků
 - smysl použití stejný jako pro C řetězec (`char []`)
 - máme ale k dispozici řadu užitečných funkcí a operátorů
 - nemusíme se starat o velikost pole
 - automatické zvětšení/zmenšení
- <http://www.cplusplus.com/reference/string/string/>

STL `std::string` – základní použití

- Deklarace řetězce
 - `string s1; string s2("Hello");`
- Přiřazení hodnoty
 - `s3 = "world";`
- Spojení řetězců
 - `s1 = s2 + s3; // "Helloworld"`
 - `s1 = s2 + " " + s3; // "Hello world"`
- Spojení z jednotlivým znakem
 - `s1 = s2 + 'o';`
- Přístup k jednotlivým znakům
 - `char a = s1[10]; // 10th character - No bounds checking!`
 - `char a = s1.at(10); // 10th character - with bounds checking`
- Zjištění délky řetězce
 - `int len = s1.length(); // length without ending zero`

```

#include <iostream>
#include <string>
using std::cout;
using std::cin;
using std::endl;
using std::string;

int main() {
    string s1;           // empty string
    string s2("Hello");
    string s3;
    s3 = "world";
    s1 = s2 + s3;        // "Helloworld"
    s1 = s2 + " " + s3;  // "Helloworld"

    cout << s1;
    cout << s2 + s3;

    cout << s1[10];      // 10th character - No bounds checking!
    cout << s1.at(10);   // 10th character - with bounds checking

    //cout << s1[100];    // 100th character - No bounds checking!
    //cout << s1.at(100); // 100th character - exception

    s1 = s2 + 'o';       // Append single character

    cout << "Length is " << s1.length();

    return 0;
}

```

STL `std::string` – konstruktory

- Řada konstruktorů pro počáteční inicializaci
- `string()` ... prázdný
- `string(const string& str)` ... kopírovací
- `string(const string& str, int start, int end)` ... podřetězec
- `string(const char* s)` ... z céčkového řetězce
- A další
 - <http://www.cplusplus.com/reference/string/string/string>

STL `std::string` – porovnávací operátory

- K dispozici jsou běžné porovnávací operátory
 - `>`, `>=`, `<`, `<=`, `==`
 - význam stejný jako u céčkové funkce `strcmp()`
- Porovnává se na základě lexikografického uspořádání
 - `"ahoj" < "zlato"`
 - `"ahoj" == "ahoj"`
 - `"ahoj" > "achoj"`
- K dispozici přetížená metoda `compare()`
 - lze namísto operátorů
 - umožňuje i porovnání podčástí apod.
 - <http://www.cplusplus.com/reference/string/string/compare/>

STL std::string – další užitečné metody

- Vyhledávání v řetězci pomocí find()
 - `s1 = "Hello world";`
 - `int posWorld = s1.find("world"); // return 6`
 - <http://www.cplusplus.com/reference/string/string/find/>
- Nahrazení v řetězci pomocí replace()
 - `s1 = "Hello world";`
 - `s1.replace(s1.find("world"), s1.length(), "dolly"); // "Hello dolly"`
 - <http://www.cplusplus.com/reference/string/string/replace>
- Vložení podřetězce na pozici pomocí insert()
 - `s1 = "Hello world";`
 - `s1.insert(6, "bloody "); // "Hello bloody world"`
 - <http://www.cplusplus.com/reference/string/string/insert>

STL `std::string` – konverze na C-řetězec

- Lze konvertovat na Céčkový řetězec
- Metoda `c_str()`
 - `const char* c_str () const;`
 - včetně koncové nuly `\0`
- Metoda `data()`
 - `const char* data () const;`
 - bez koncové nuly!

STL `std::string` – další později

- `std::string` je STL kontejner
- Vlastnosti a interní chování probereme u STL

Shrnutí

- Organizační – vše na <http://cecko.eu/public/pb161>
- C++ je nadstavbou C, ale výrazné rozšíření
 - C lze kompilovat, pozor na míchání syntaxe
- Objektově orientované programování
 - jiný způsob analýzy a dekompozice problému
 - zlepšuje robustnost a rozšiřitelnost kódu
- Používejte nástroje
- Ptejte se!

Bonus 😊



The Codeless Code

An illustrated collection of (sometimes violent) fables
concerning the Art and Philosophy of software development,
written in the spirit of Zen kōans



- <http://thecodelesscode.com/>
- Ilustrované, někdy morbidní, vývojářské bajky
- Inspirované zenovými koany

Case 3

moderately geeky ●●○

Encapsulation

A young monk, new to the temple, not only declared all his methods public but all his instance variables as well. The head abbot grew weary of rebuking the monk and asked the Java master for advice.

The next day, the master summoned the monk to take lunch with him in his private office.

The monk entered to find a sumptuous meal laid out upon the conference table. The master then bade the monk to lie upon the floor.

The monk did as commanded, whereupon the master opened the monk's robes and drew a large knife. He pressed the naked point firmly into the monk's chest until a ruby droplet welled up around the blade. The monk cried out in terror and asked the master what his intention was.

"To slit open your belly," explained the master, "so that I may spoon the rice and pour the tea inside. My schedule is quite full, and I find this method of feeding guests to be extremely efficient."

Afterward the monk required no more correction.



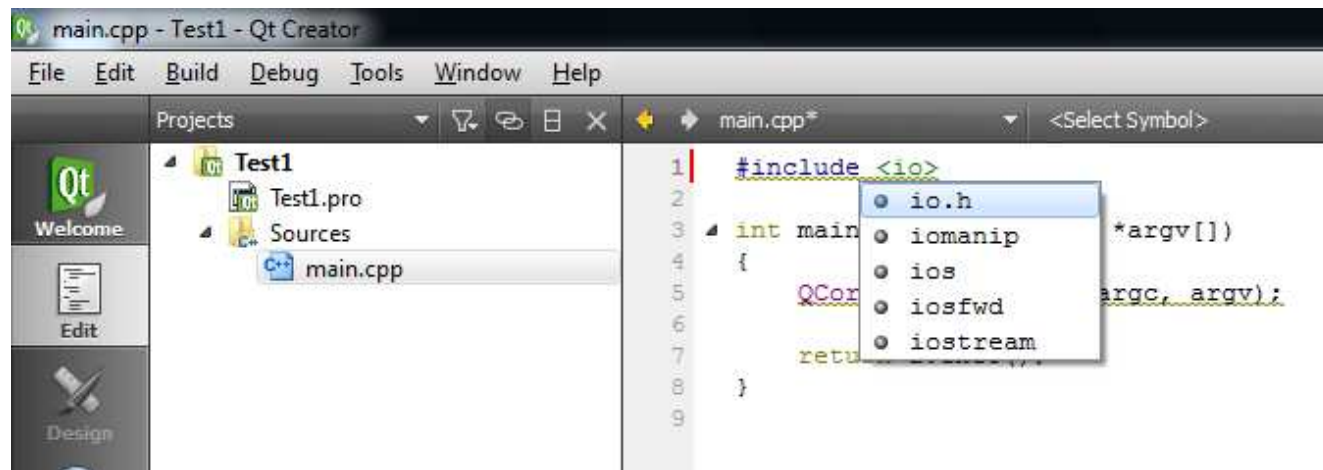
Nástroje



66

Editor

- Samostatný program (vim, pico, joe...)
- Nebo integrovaný v IDE
 - všechny mají
 - zvýraznění syntaxe, lokalizace chyb, kontextová nápověda...
 - např. QT



67

System souborů pro třídy

- Stejný systém jak v C
 - dělení na *.h (*.hpp)
 - použití pro deklaraci třídy, atributy a implementaci krátkých funkcí
 - *.cpp (*.cc) soubory pro těla metod
 - většinou jeden soubor pro každou třídu
- Standardní cesty, prohledávání
 - `#include "header.h"`
 - `#include <header.h>`
 - `#include "../to_include/header.h"`
- Používejte ochranné makra proti násobnému vkládání!

```
#ifndef JMENOTRIDY_H
#define JMENOTRIDY_H
class JmenoTridy {
...
};
#endif
```

Kompilace Aisa

- GNU GCC / g++
 - přepínače (-c, -g, -Wall, -Wextra, -o ...)
 - <http://gcc.gnu.org/onlinedocs/gcc-4.4.1/gcc/Overall-Options.htm>
- Překlad přímo do výsledné binárky
 - g++ -ansi -pedantic -Wall -o hello hello.cpp
 - (mezivýsledky jsou smazány)
- Spuštění programu
 - ./hello
- (Kompilace s C++
 - -std=gnu++0x

```
/2010/p1:  
/2010/p1:  
/2010/p1: g++ -ansi -pedantic -Wall -o hello hello.cpp  
/2010/p1: ./hello  
Hello world!  
/2010/p1:  
/2010/p1:  
/2010/p1:  
/2010/p1:
```

Překlad po částech (prakticky na cvičení)

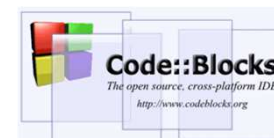
1. **Preprocessing** "g++ -E hello.cpp > hello.i,"
 - rozvinutí maker, expanze include...
2. **Kompilace** "g++ -S hello.i,"
 - syntaktická kontrola kódu, typicky chybová hlášení
3. **Sestavení** "as hello.s -o hello.o,"
 - assembly do strojového kódu
4. **Linkování** "g++ hello.o,"
 - nahrazení relativních adres absolutními

```
: g++ -E hello.cpp > hello.i
: g++ -S hello.i
: as hello.s -o hello.o
: g++ hello.o
: ./hello
Hello world!
```

70

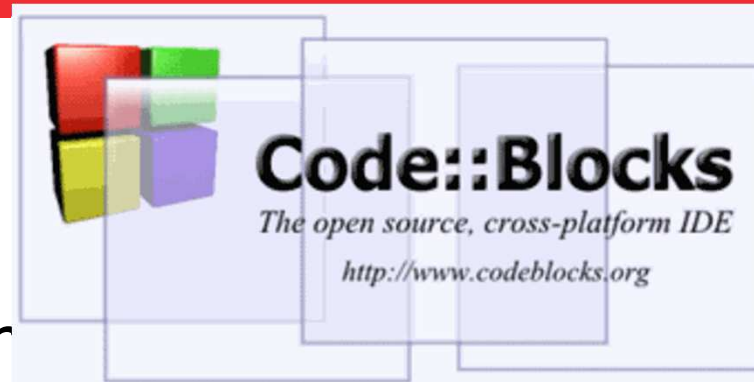
Integrated Development Environment (IDE)

- Integrovaný soubor nástrojů pro podporu vývoje
 - typicky s grafickým rozhraním
 - Code::Blocks, Eclipse, Netbeans, Visual Studio, QT Creator, Dev-C++ a mnoho dalších
- Obsahuje typicky:
 - Způsob vytváření a kompilace celých projektů
 - Editor se zvýrazňováním syntaxe
 - WISIWIG GUI editor
 - Pokročilý debugger
 - Profilační a optimalizační nástroj
 - Podporu týmové spolupráce...



71

Code::Blocks



- IDE spustitelné na běžných
 - (Windows, Linux, MacOS)
 - Podpora různých jazyků, pro nás C/C++
 - Kombinace s TDM-MinGW (gcc 4.5.1)
- Budeme využívat jako defaultní IDE
 - pokud ale ovládáte dobře jiné, klidně jej použijte
 - např. QT Creator, Visual Studio...
 - vhodné použít překlad pomocí gcc
 - hlídá dodržení standardu, kontrola domácích úkolů
- Tutoriál na <http://www.youtube.com/watch?v=uVz0lalh8TM>

72

QT Creator



- IDE spustitelné na běžných OS
- Verze 2.3.0 nainstalována na školních strojích
- POZOR: QT není jen IDE, ale i celé API
 - pro zajištění přenositelnosti nestandardizovaných operací poskytuje mezivrstvu QT API (Qxxx objekty)
 - (přenositelnost zdrojového, nikoli spustitelného kódu)
- QT API nebudeme využívat
 - budeme psát a překládat v čistém C++
- Tutoriál na <http://cecko.eu/public/qtcreator>

Automatizace překladu - Make

- Make, CMake...
 - nástroje pro provedení skriptu automatizující překlad
- Makefile
 - jazykově nezávislý skript definující způsob překladu
 - cíle překladu (targets), nejběžnější -all, -clean, -install
- Jednoduchý Makefile

```
# Simple makefile
all:
    g++ -ansi -pedantic -Wall -o hello hello.cpp
```

- Rozšířený Makefile

```
ccc=g++
CFLAGS=-ansi -pedantic -Wall

all: hello
hello:
    $(CCC) $(CFLAGS) -o hello hello.cpp
clean:
    rm hello
```

74

Doxygen



- Nástroj obdobný jako JavaDoc pro Javu
 - umožňuje generovat dokumentaci z poznámek přímo v kódu
 - html, latex...
- Odevzdávané domácí úkoly musí dokumentaci obsahovat
- Tutoriál na <http://cecko.eu/public/doxygen>

```
/**
 * Display sizes of basic data types
 *
 * @param arraySize  size of dynamically allocated array
 * @return           nothing
 */
void demoDataSizes(int arraySize) {
    #define          ARRAY_SIZE      100
    char             array[ARRAY_SIZE];           // Fixed size array
```

75

Subversion (SVN)



- Nástroj pro verzování kódu a podporu spolupráce v týmu
- V repozitáři (na „serveru“) jsou udržovány všechny provedené změny
 - lze se vrátet zpět na funkční verzi, vytvářet oddělené větve...
 - kód z SVN by měl jít vždy kompilovat
- Checkout, Commit, dokumentace verzí
- Lze vytvářet vlastní repozitáře
 - např. BitBucket, fakultní SVN
 - nebo vlastní server (např. VisualSVN Server)
- Domácí úkoly budou zadávány přes SVN
- Tutoriál na <http://cecko.eu/public/svn>

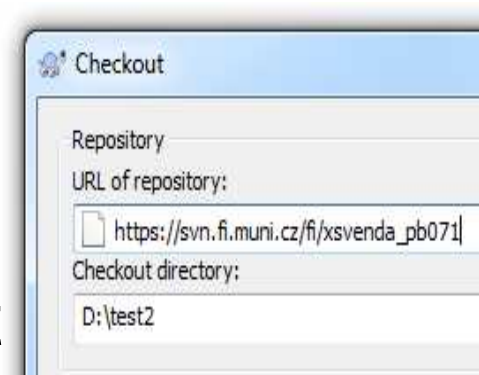
76

Výhody použití verzovacího nástroje

- Při používání jednotlivcem
 1. Záloha práce mimo svůj počítač
 2. Práce na více počítačích (*update*, na konci *commit*)
 3. Návrat zpět na starší verzi (která fungovala)
- Při používání ve skupině
 1. Souběžné práce nad stejnými zdrojáky
 2. Práce vždy nad aktuálními zdrojáky
 3. Možnost práce “offline”
 4. Vytváření nezávislých vývojových větví

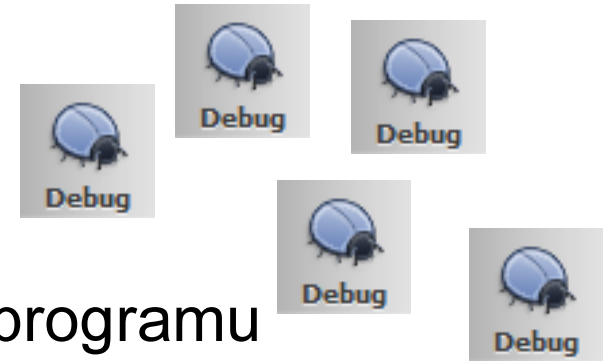
Využití fakultního SVN serveru

- <https://fadmin.fi.muni.cz/auth/>
- Počítačová síť → Subversion účet
- Přidat nový repozitář: `login_pb161`
- RClick → SVN Checkout
 - `https://svn.fi.muni.cz/fi/login_pb161`
- Zřídte si alespoň dva repozitáře
 - `login_pb161` na odevzdávání příkladů
 - `login_cokoli` na svoje průběžné kódy
- Repozitář neodstraňujte, stačí odstranit soubory!



78

Debugging



- Proces hledání a odstraňování chyb v programu
 - ladící výpisy, studium výstupních souborů...
 - často využíván tzv. debugger
- Debugger
 - gdb nebo součást IDE
 - speciální způsob přeložení a spouštění aplikace
 - tak, aby bylo možné provádět jednotlivé operace programu
 - (typicky řádky zdrojového kódu)
- Základní termíny (více později)
 - nastavení prostředí, breakpoints, watch/locals
 - step over, step into, step out, run to cursor
 - změna hodnoty proměnných za běhu, asserts

