PB161 Programování v jazyce C++
Přednáška 1

Organizace
Úvod do OOP v C++

Nikola Beneš

21. září 2015

Cíle předmětu

1. vysvětlit základy OOP
2. seznámit s možnostmi jazyka C++
3. podpořit praktické programátorské schopnosti
4. nadchnout do (nebo alespoň neodradit od) programování

Organizace předmětu

`http://cecko.eu/public/pb161`

Organizace předmětu

Přednášky

- nepovinné, ale snad přínosné a zábavné :-)
- během jedné vnitrosemestrální test
- zvané přednášky ke konci semestru

Cvičení

- povinná, dvě neomluvené neúčasti tolerovány
- aktivní práce na příkladech, konzultace
- průběžné testíky formou odpovědníků
- možnost párového programování
- poslední týden zápočtový příklad

Domácí úkoly

- pět úkolů, jeden navíc
- max 12 bodů za úkol + bonusy
- deadline na stránce úkolu
- odevzdávání do fakultní SVN, max 3 ostré pokusy, odevzdávání nanečisto
- ukázková řešení
- valgrind: chyby nalezené valgrindem snižují hodnocení o dvě pětiny
- podmínka k zápočtu: alespoň 4 domácí úkoly s nenulovým hodnocením

detaily na `http://cecko.eu/public/pb161`

Varování

Neopisujte!

- ubližujete sami sobě
- provádíme automatickou kontrolu opisování
- opisování = 0b za úkol (pro oba)

Nezveřejňujte svá řešení!

- stejný postih jako za opisování
- zákaz zveřejňovat řešení i po termínu domácího úkolu

Neotvírejte odpovědníky jindy než na cvičení!

- cvičící kontrolují, kdo je na cvičení
- hodnoceno -5 body

Bodování

- tvrdé a měkké body
  - měkké se nepočítají do limitu pro zápočet a zkoušku

domácí úkoly	$6 \times 12 = 72$
odpovědníky	$10 \times 3 = 30$
vnitrosemestrální test	20
celkem (tvrdé body)	122

- zkouškový test: 80 bodů
- měkké body:
  - bonusové části domácích úkolů
  - body za aktivitu na cvičeních

Hodnocení

Zápočet

- >= 65 tvrdých bodů, 4 nenulové domácí úkoly, zápočtový příklad

Zkouška

- >= 95 tvrdých bodů, zápočet
- známka podle součtu tvrdých a měkkých bodů:

>= 170 bodů	A
>= 150 bodů	B
>= 130 bodů	C
>= 110 bodů	D
>= 95 bodů	E

Učební materiály

- podklady k přednáškám, slidy, ukázkové zdrojáky
- `http://cecko.eu/public/pb161`
- záznamy přednášek v ISu
- `http://cppreference.com`
- `http://cplusplus.com`

ukázka: online dokumentace

Kontakty

Přednášející: Nikola Beneš `xbenes3@fi.muni.cz`

- konzultační hodiny: pondělí 14.10–15.40 B421

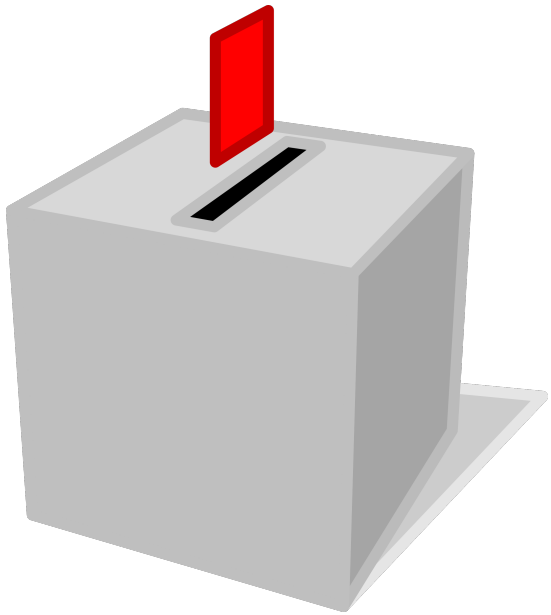
Cvičící: podle rozvrhu

Studentští poradci

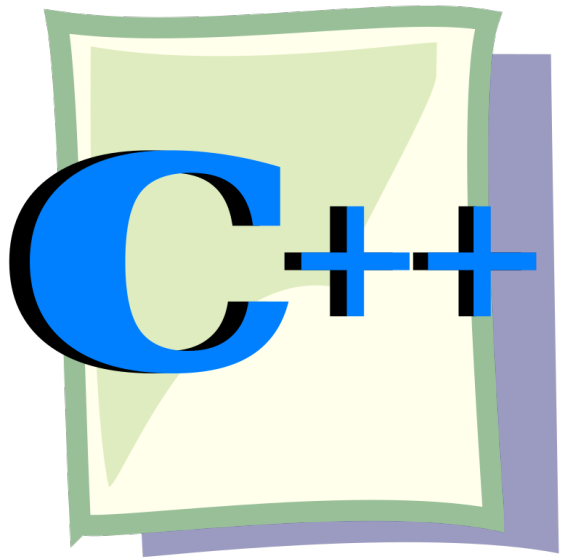
- konzultační hodiny na `http://cecko.eu/public/pb161`
- od druhého týdne semestru

Zpětná vazba

- předmětová anketa v ISu (až na konci semestru)
- občasné dotazníky (obtížnost úloh apod.)
- osobně, e-mailem
- krabice pro anonymní vzkazy

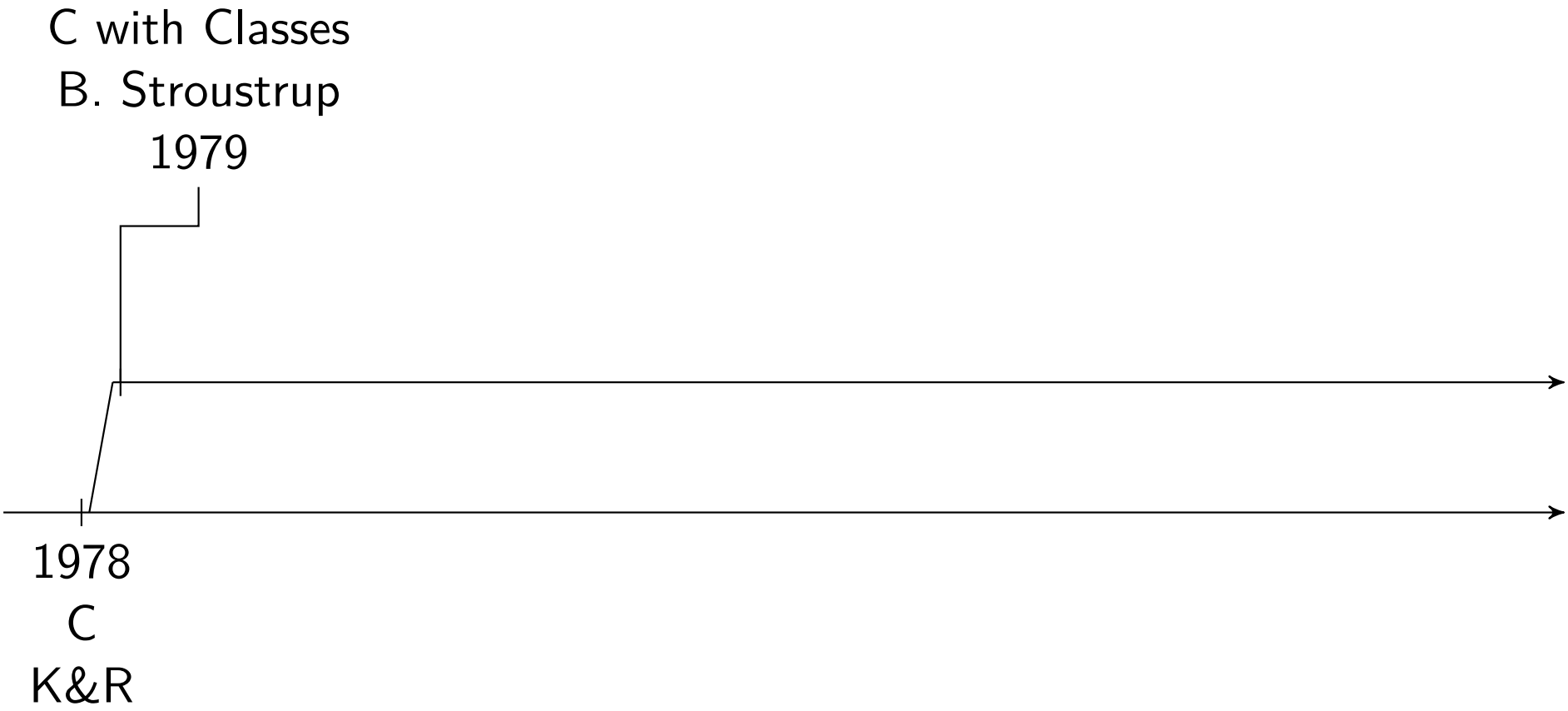


Programovací jazyk C++

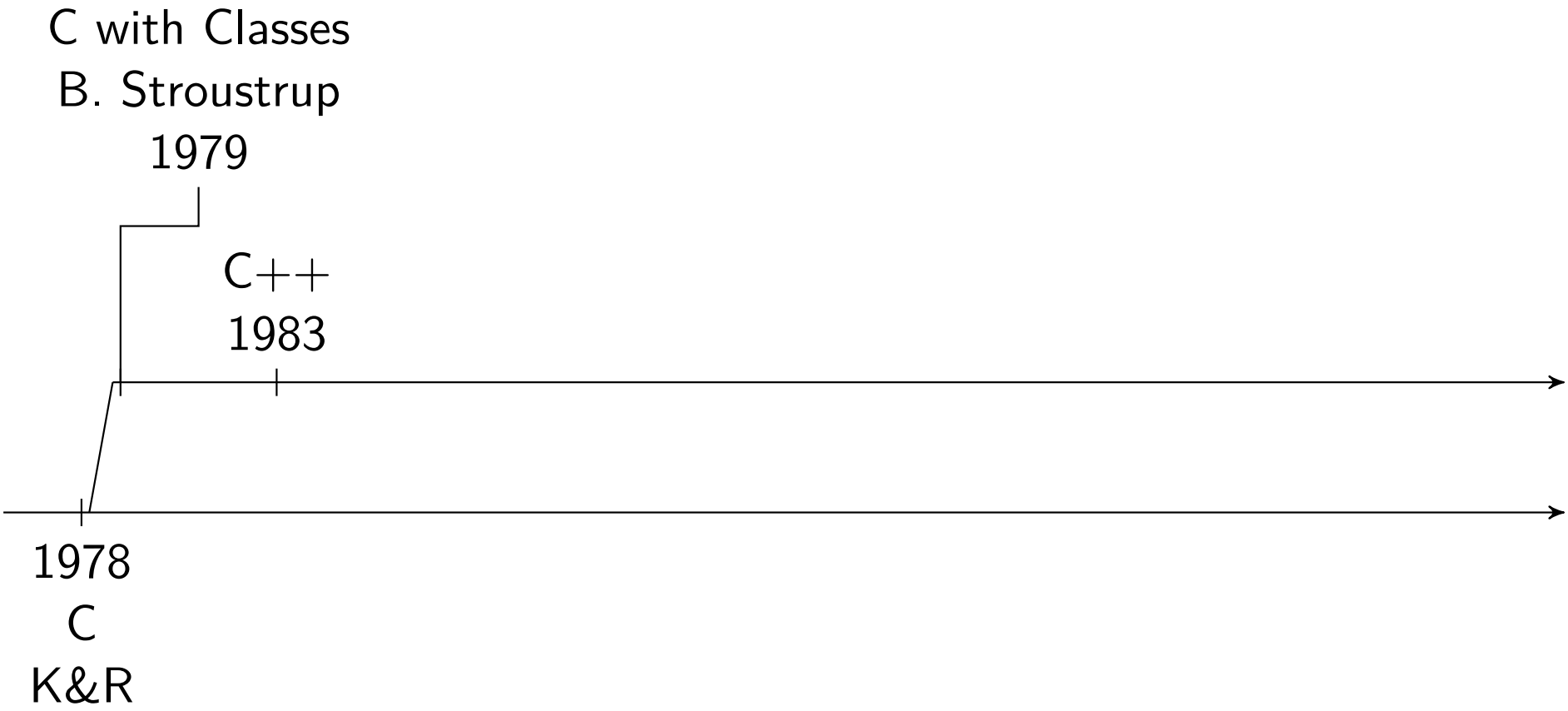


Historie a vývoj

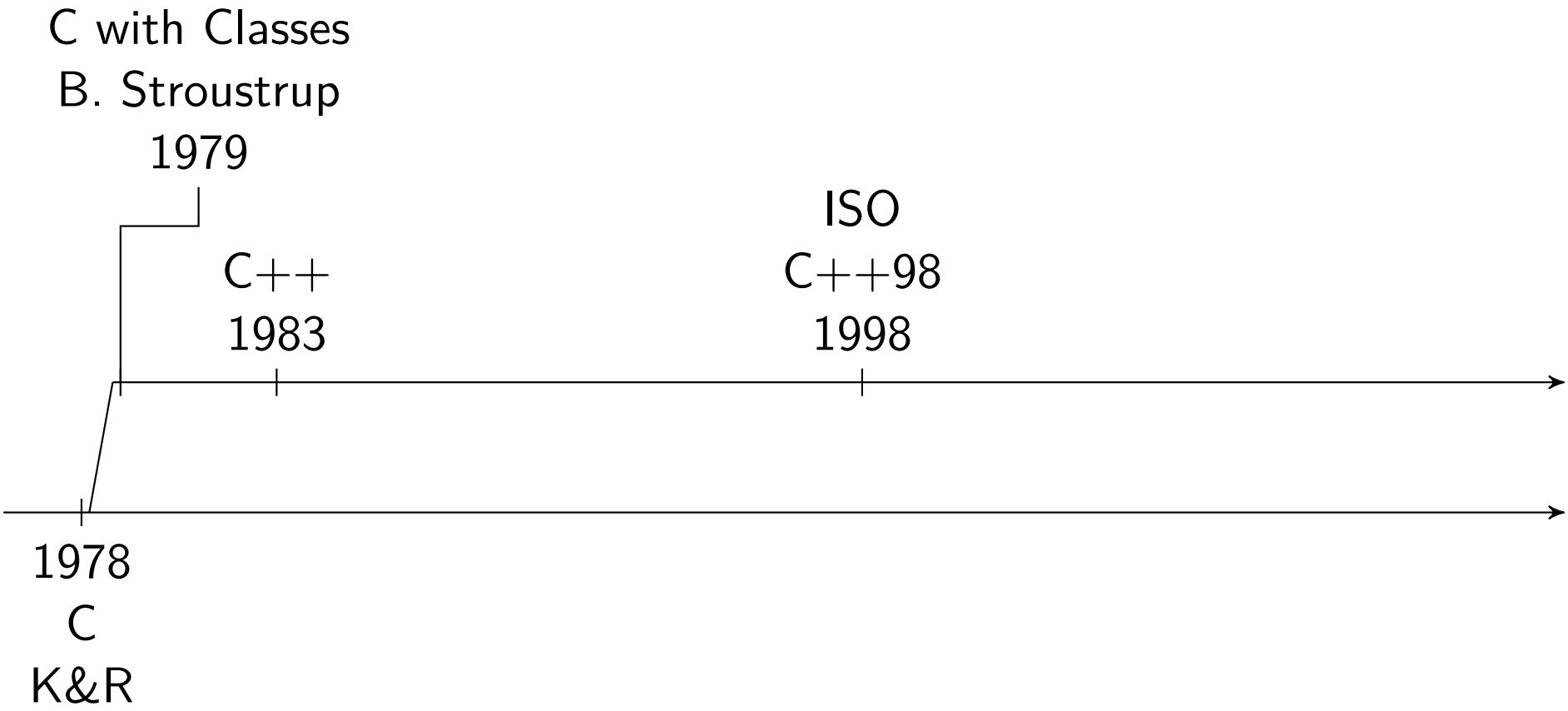
Historie a vývoj



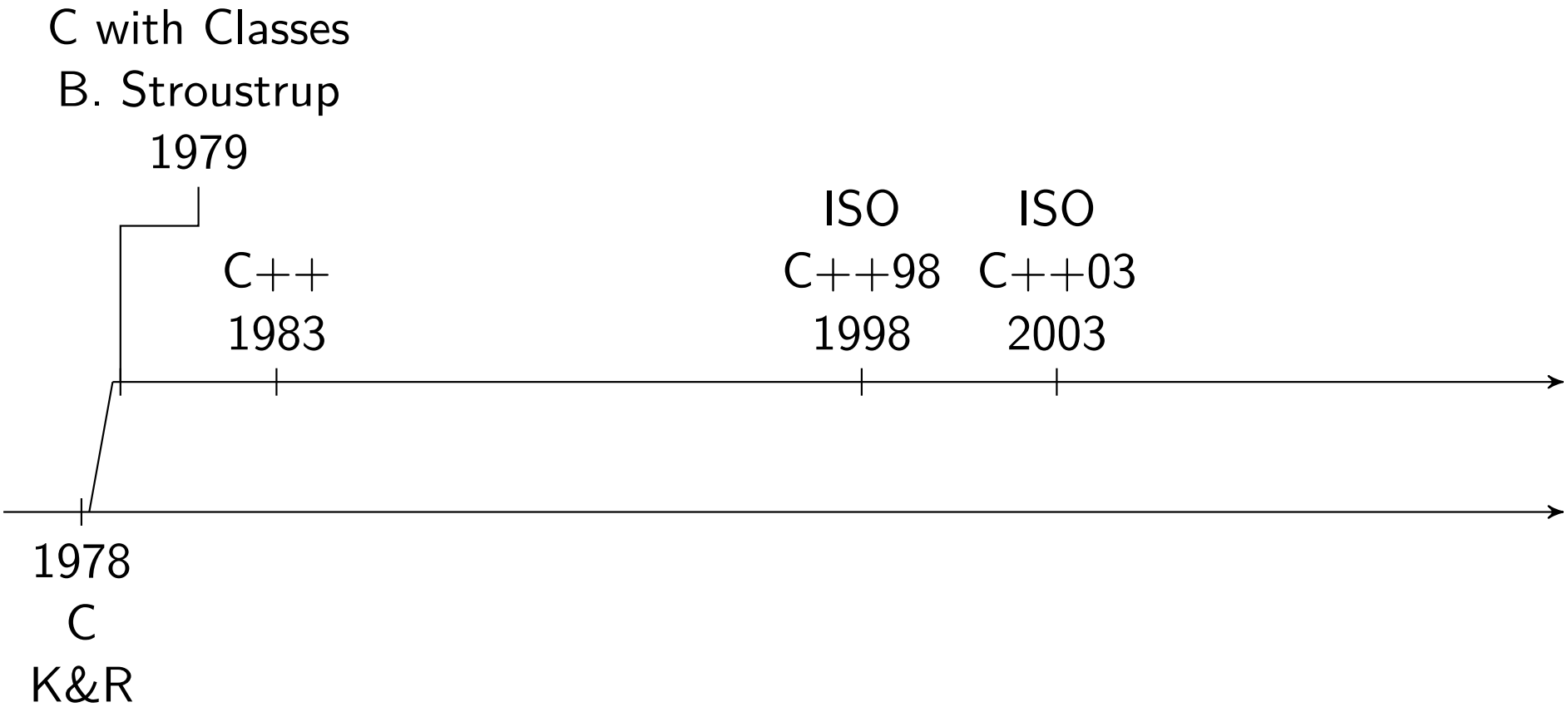
Historie a vývoj



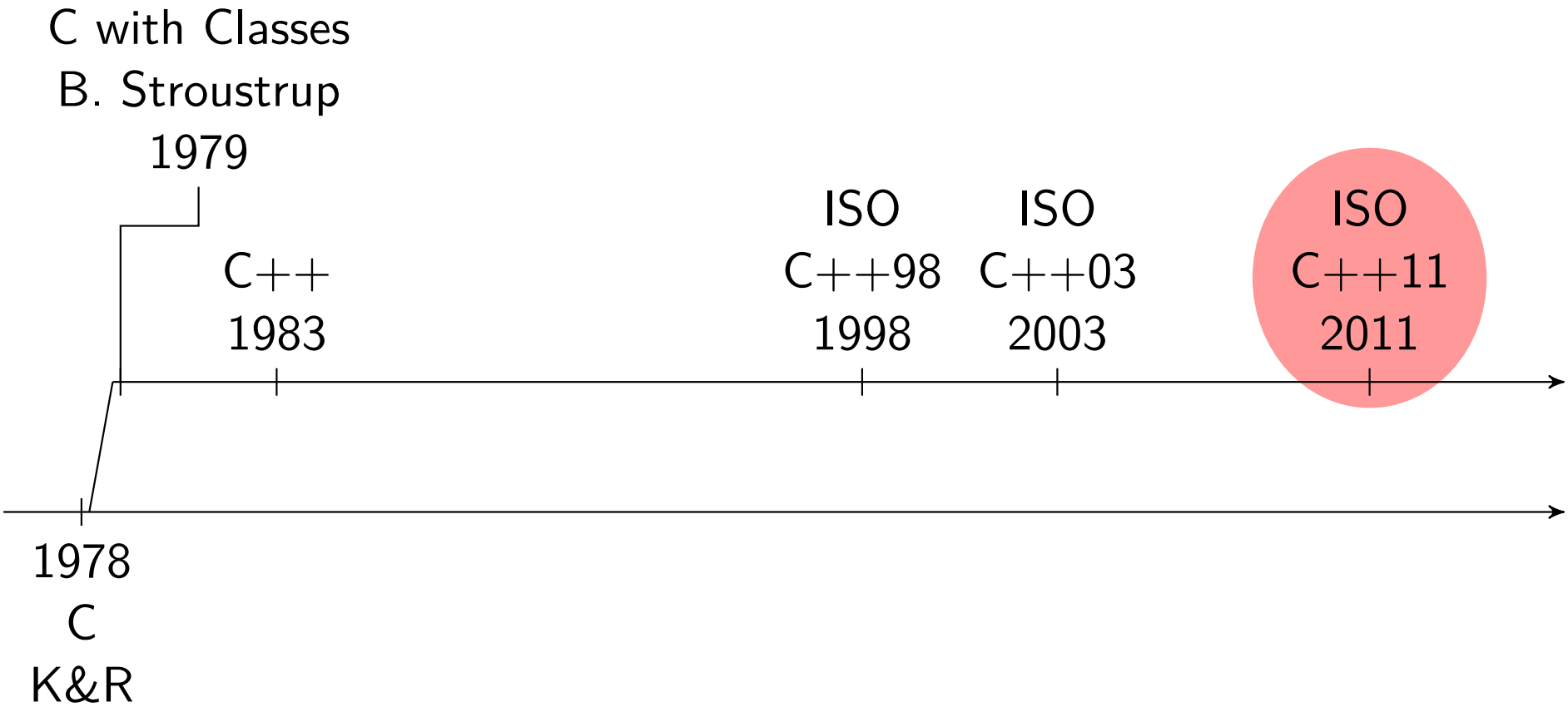
Historie a vývoj



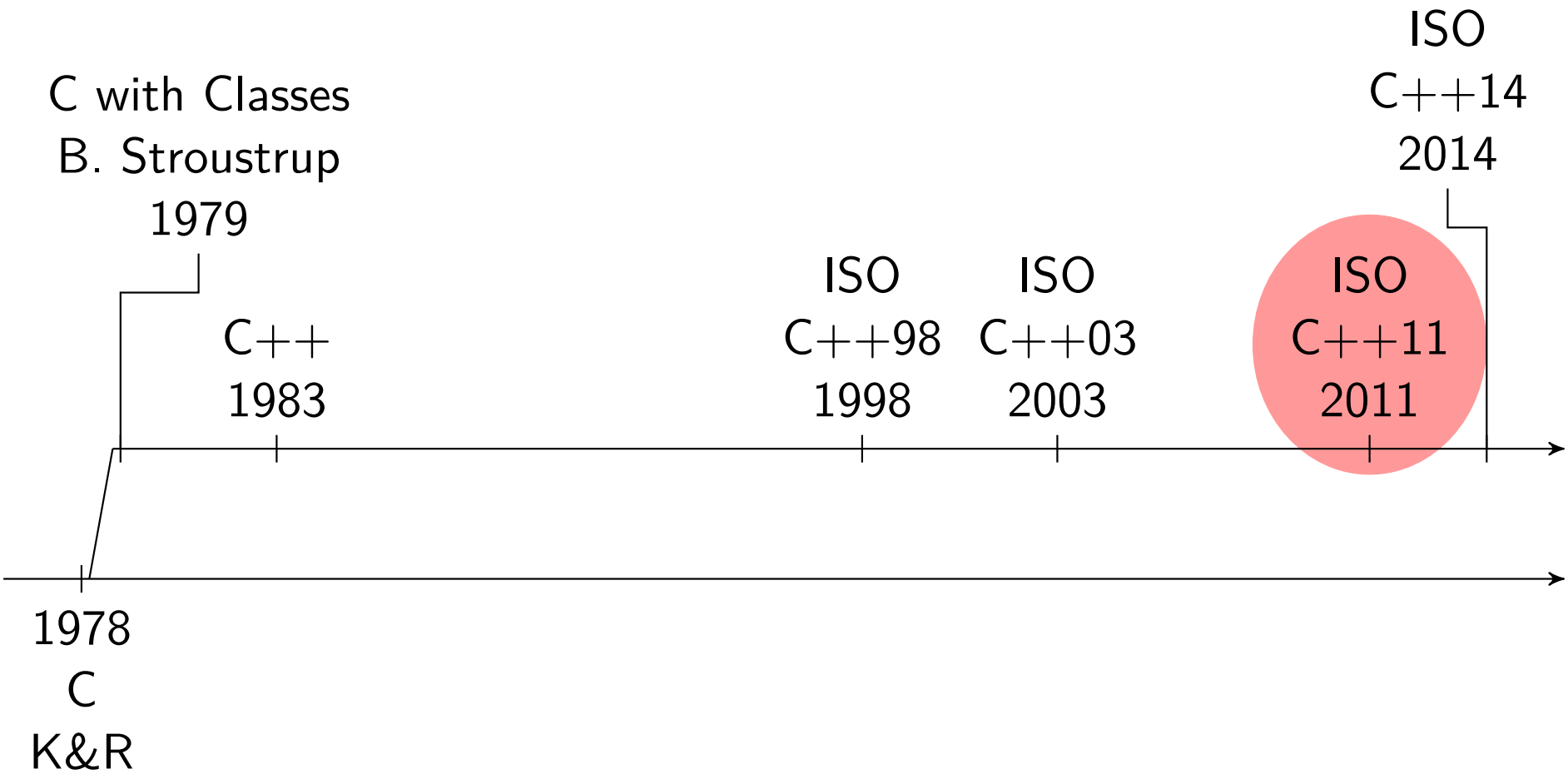
Historie a vývoj



Historie a vývoj



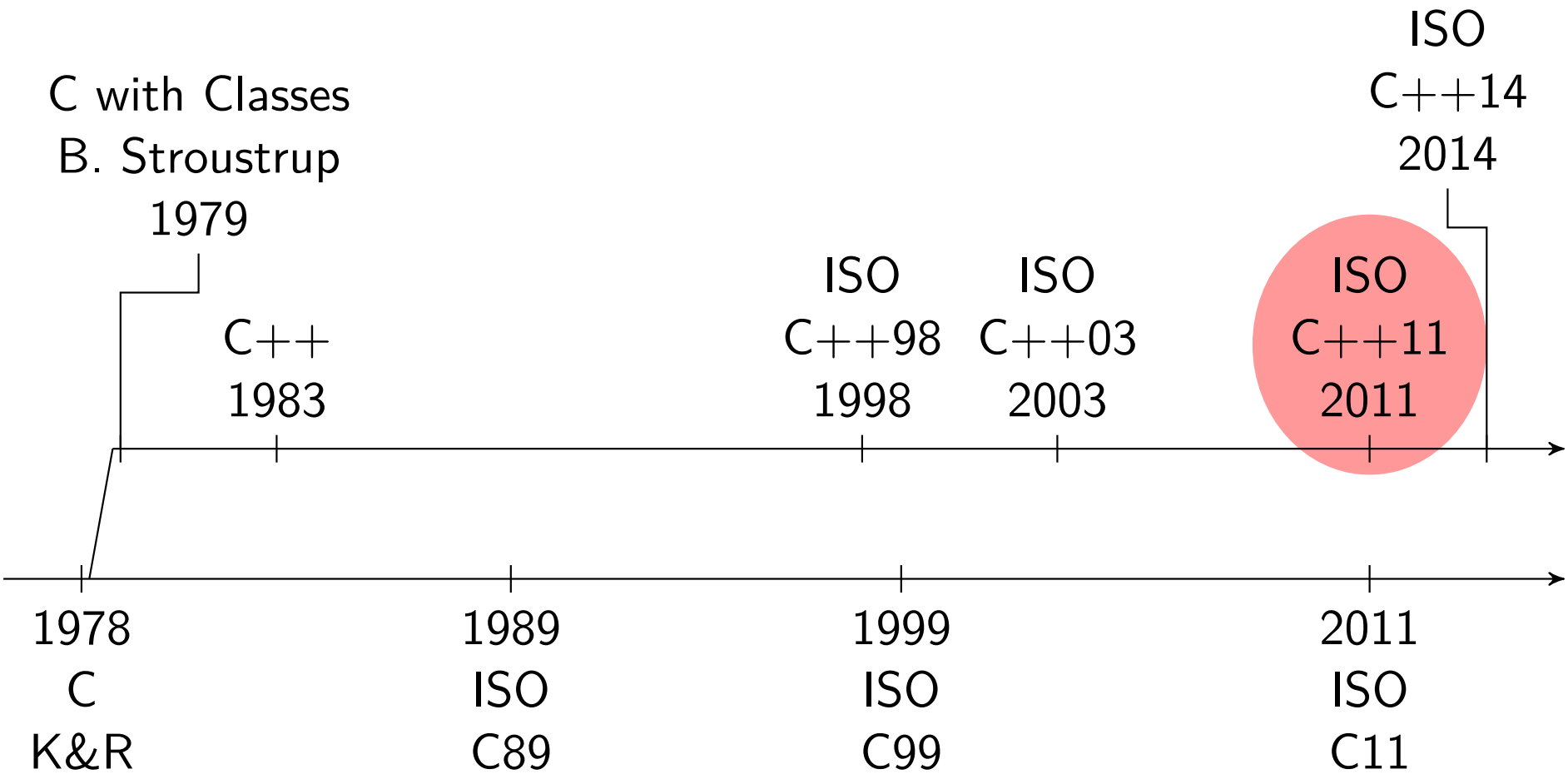
Historie a vývoj



Budoucnost: C++17, ...

Nestandardizovaná rozšíření

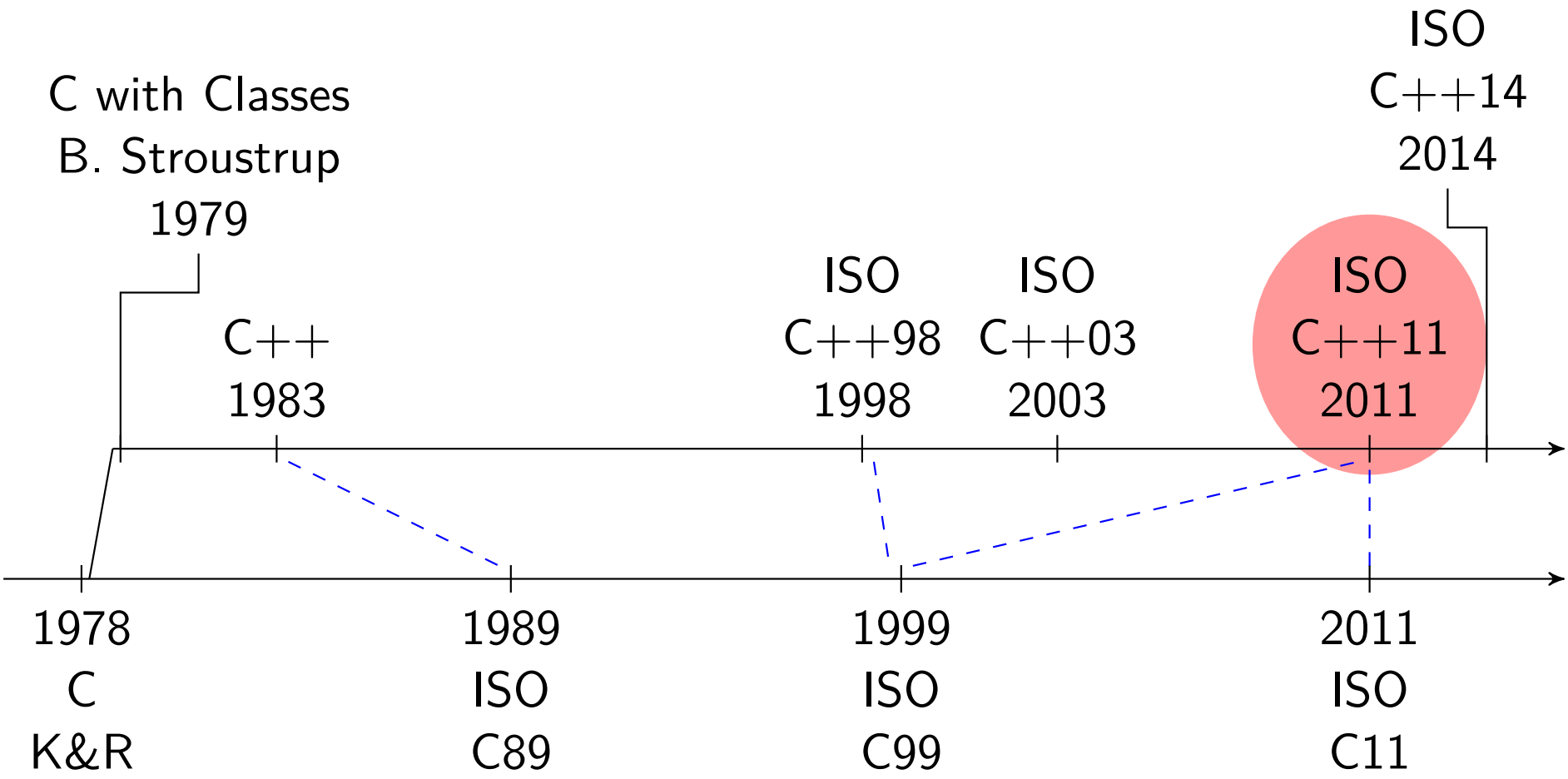
Historie a vývoj



Budoucnost: C++17, ...

Nestandardizovaná rozšíření

Historie a vývoj



Budoucnost: C++17, ...

Nestandardizovaná rozšíření

Charakteristika C++

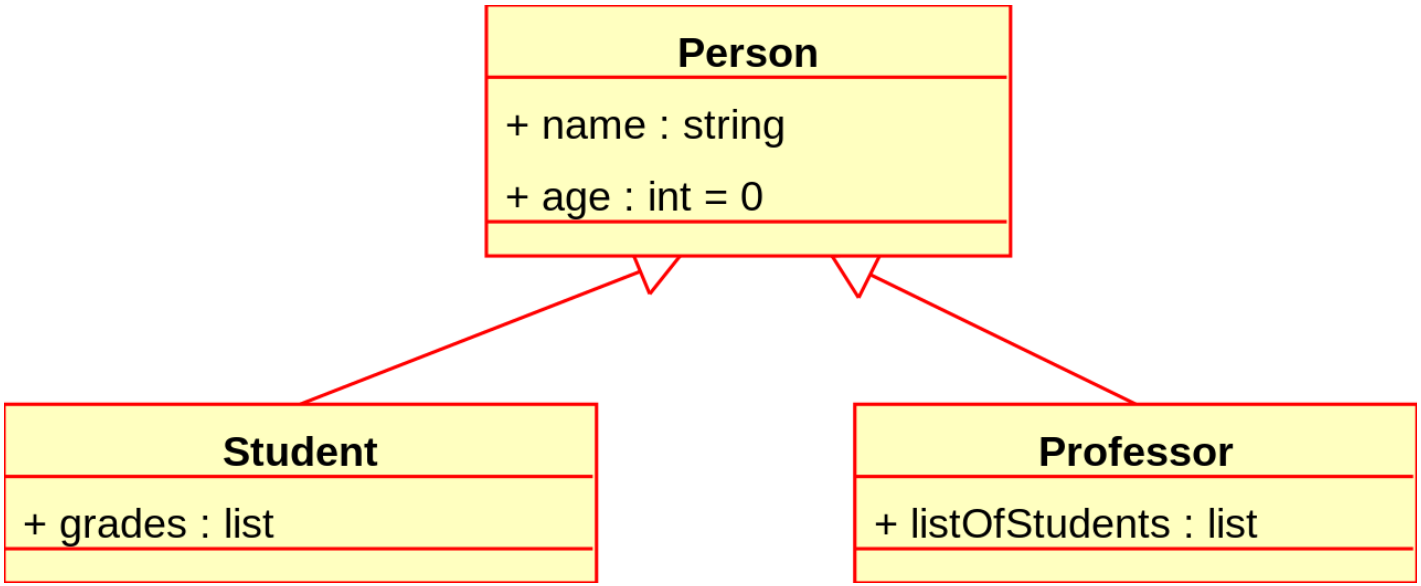
- imperativní, staticky typovaný jazyk
- objektově-orientovaný
- s funkcionálními prvky (zejména od C++11)
- podporuje generické programování (šablony)
- částečně zpětně kompatibilní s C
- rozsáhlá standardní knihovna

Proč používat C++?

- široké rozšíření
- vysoká rychlost kódu
- univerzálnost
- vhodné pro
  - větší projekty
  - systémové aplikace
  - rychlou grafiku
  - embedded zařízení
- spíše nevhodné pro
  - webové aplikace
  - rychlé prototypy

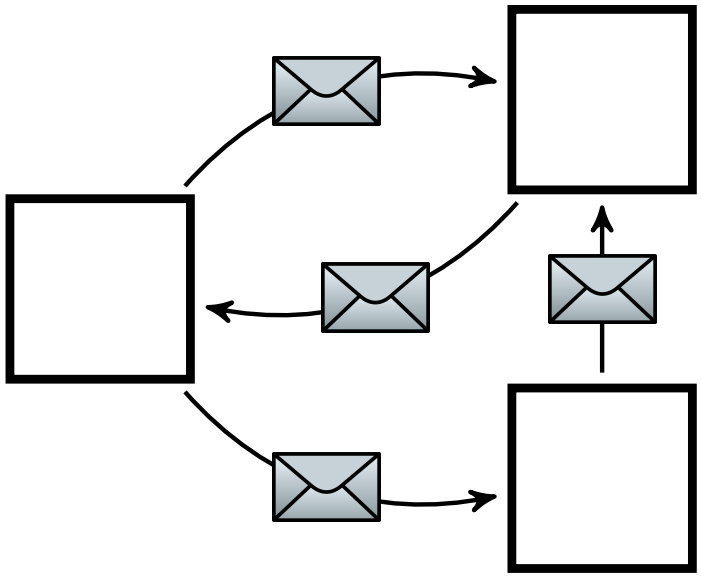
`http://benchmarksgame.alioth.debian.org/`

Objektově orientované programování



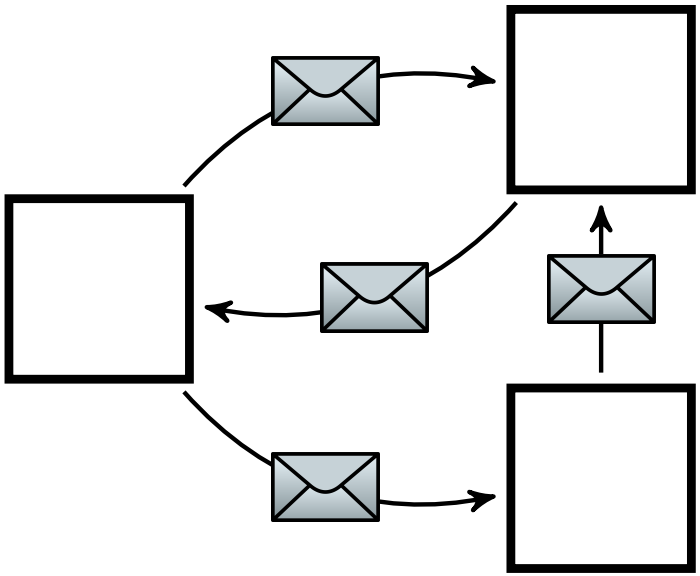
Základní představa OOP

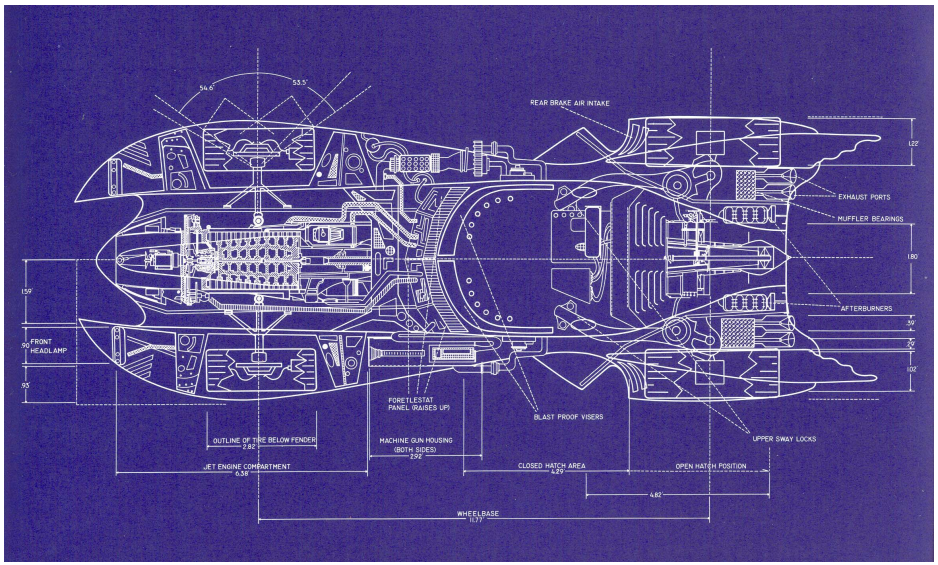
- svět se skládá z objektů
- objekty mají svůj vnitřní stav, který není vidět
- objekty komunikují pomocí zpráv



Základní představa OOP

- svět se skládá z objektů
- objekty mají svůj vnitřní stav, který není vidět
- objekty komunikují pomocí zpráv

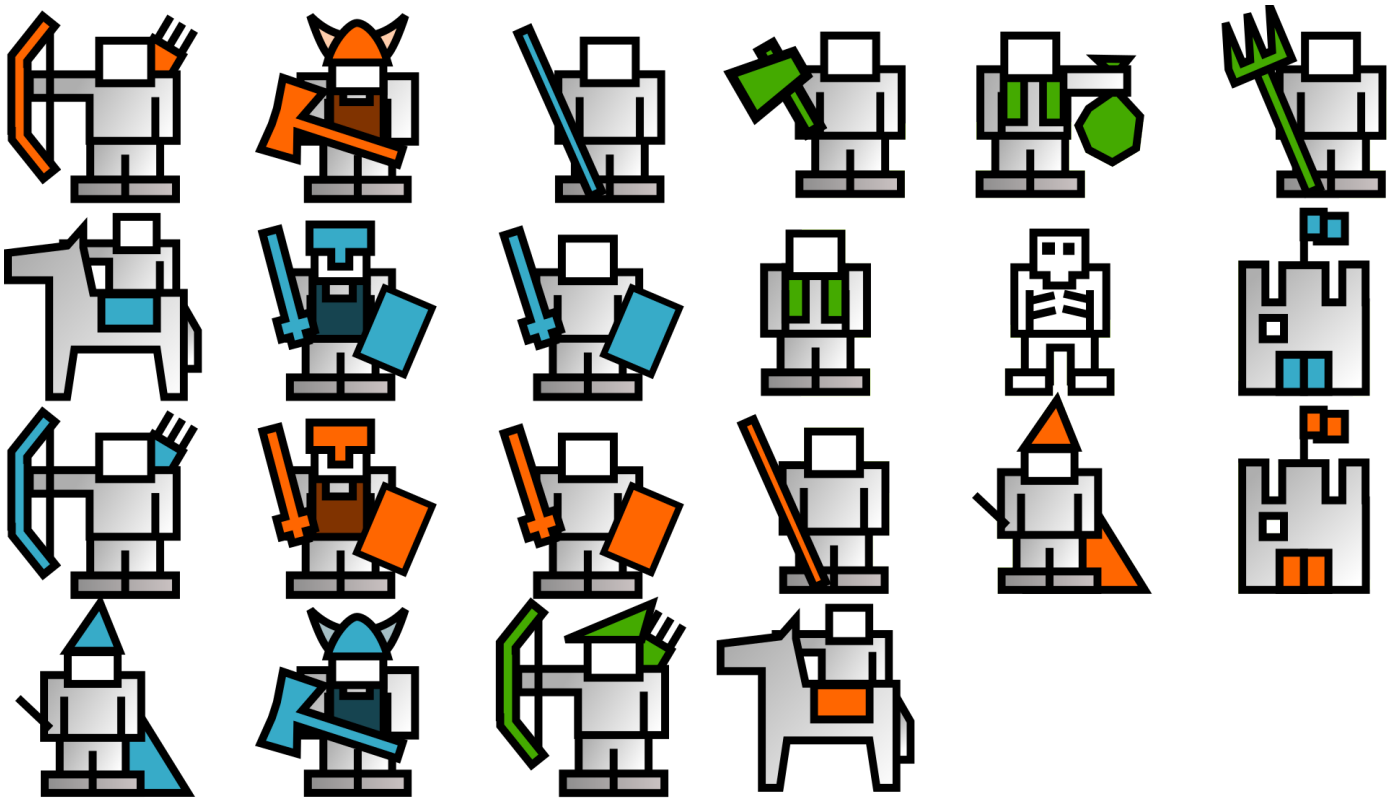




- objekty se často vytvářejí podle vzoru (třída, prototyp, ...)

Příklad

Strategická hra



- co jsou objekty?
- co jsou zprávy?

Principy OOP

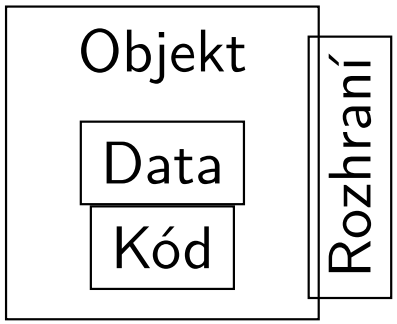
Zapouzdření

Abstrakce

Dědičnost

Polymorfismus

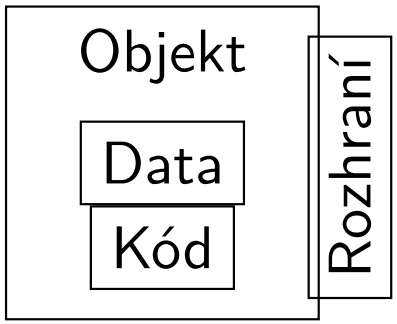
Principy OOP – Zapouzdření



- data i kód na jednom místě
- objekt komunikuje jen skrze rozhraní
- umožňuje skrýt vnitřní reprezentaci dat

Výhody:

Principy OOP – Zapouzdření



- data i kód na jednom místě
- objekt komunikuje jen skrze rozhraní
- umožňuje skrýt vnitřní reprezentaci dat

Výhody:

- možnost změnit vnitřní implementaci
- ochrana proti chybám
- nutí rozvrhnout program do nezávislých částí
- umožňuje další OOP vlastnosti

Principy OOP – Abstrakce

- souvisí se zapouzdřením
- umožňuje pracovat s ideální představou, nezatěžuje programátora implementačními detaily

Datová abstrakce

- použití dat bez znalosti skutečného umístění/reprezentace
- příklad: soubor na disku, na serveru, ...

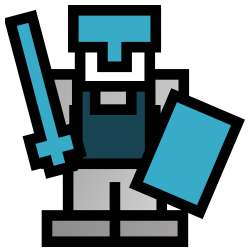
Funkční abstrakce

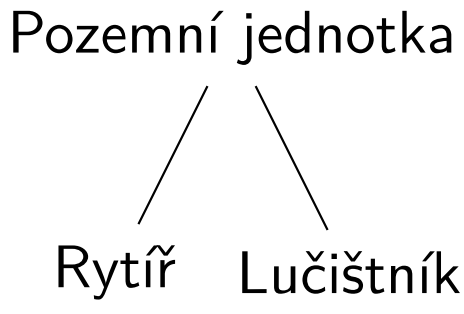
- uživatel nemusí znát detaily implementace
- příklad (strategická hra): zpráva „zaútoč“ pro různé druhy jednotek

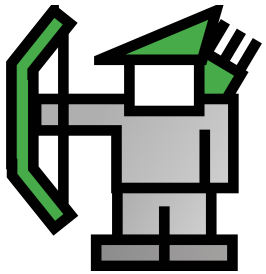
Principy OOP – Dědičnost

- objekt (třída) může dědit od jiného objektu (třídy)
- vztah předek–potomek
- snižuje opakování kódu

Příklad:







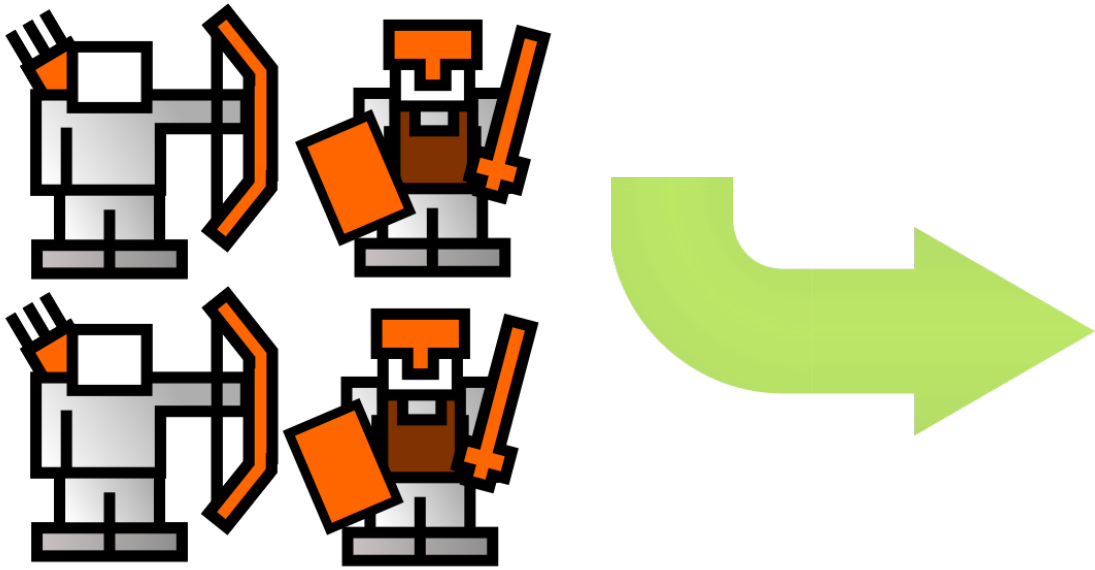
Liskovové princip nahraditelnosti: potomek vždy může zastoupit předka

Vícenásobná dědičnost, kompozice: příště

Principy OOP – Polymorfismus

- souvisí s abstrakcí
- umožňuje psát kód obecně, pouze se znalostí rozhraní
- polymorfismus skrze dědičnost

Příklad: poslat všem pozemním jednotkám ve vybrané oblasti stejný povel



- existují i jiné způsoby realizace polymorfismu (generické programování)

OOP jako styl myšlení

Object-Oriented Design

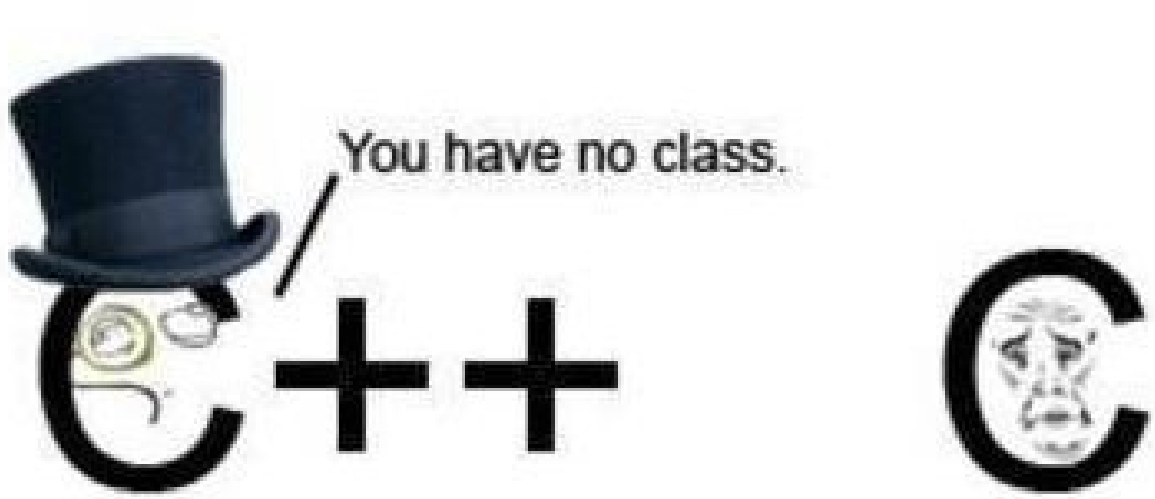
- OOP je přístup k řešení
- objektový návrh: co jsou objekty, jaké mají vztahy
- můžeme programovat „objektově“ i v neobjektových jazycích
- v OOP jazycích ale máme syntaktickou podporu

Příklad v jazyce C

```
struct Archer {
    int hit_points;
    int range;
    int x, y;
};

void moveArcher(struct Archer * archer, int tx, int ty) {
    // ... kód pro pohyb lučištníka
}
```

Realizace OOP v C++



OOP v C++

Přístup založený na třídách: `class`

- rozšíření datového typu `struct` z C
  - v C++ máme `class` i `struct`, liší se v přístupových právech
- kromě atributů (dat) můžou obsahovat i metody (funkce)
- přístupová práva:
  - `public` vidí všichni
  - `protected` vidí jen potomci třídy
  - `private` vidí jen třída sama
  - implicitní práva: u `class` je to `private`, u `struct` je to `public`
- metody realizují princip posílání zpráv
- objekty jsou konkrétní instance tříd

Příklad třídy

```
class LandUnit {
private: // soukromé, není vidět z venku
    int x, y;
public:  // veřejné, je součástí veřejného rozhraní třídy
    void move(int targetX, int targetY); // jen prototyp
};

void LandUnit::move(int targetX, int targetY) {
    // ... kód pro pohyb jednotky ...
    x = targetX;
    y = targetY;
}

int main() {
    LandUnit soldier1; // soldier1 je objekt
    soldier1.move(1, 1);
}
```

Metody – this

- skrytý parametr **this**: ukazatel na konkrétní objekt

```
void LandUnit::move(int, int);
// je jakoby
void LandUnit::move(LandUnit * this, int, int); // není C++!

soldier1.move(1, 1);
// je jakoby
LandUnit::move(&soldier1, 1, 1);                    // není C++!
```

- použití atributů uvnitř metod: místo `x` a `y` jsme mohli psát **this**`->x` a **this**`->y`

Metody – deklarace a definice

- deklarace a definice metod je možno psát
  - odděleně
  - uvnitř deklarace třídy

```
class Person {
private:
    int age;
public:
    void setAge(int age) {
          this->age = age;
    }
    int getAge() {
        return age;
    }
};
```

- typické pro menší metody (getter/setter); jako by byly `inline`

Metody – rozvržení do souborů

Hlavičkový soubor `*.h`

- deklarace třídy
- inline metody

Zdrojový soubor `*.cpp`

- definice (implementace) metod

Zamyšlení: Co všechno patří k třídě?

- What's in a class? The interface principle
  `http://gotw.ca/publications/mill02.html`
- k třídě patří i volné funkce s danou třídou pracující

Konstruktor

- motivace: Jaká bude hodnota atributů při vytvoření objektu?

Konstruktor

- motivace: Jaká bude hodnota atributů při vytvoření objektu?
- Bude neinicializovaná. Jak to řešit?

Konstruktor

- motivace: Jaká bude hodnota atributů při vytvoření objektu?
- Bude neinicializovaná. Jak to řešit?
- neelegantní řešení: metoda pro inicializaci, kterou je třeba zavolat
- lepší řešení: **konstruktor**

```
class LandUnit {
    int x, y;
public:
    LandUnit();
    LandUnit(int, int);
};

LandUnit::LandUnit() {
    x = 0;
    y = 0;
}
```

```
LandUnit::LandUnit(int x, int y) {
    this->x = x;
    this->y = y;
}

int main() {
    LandUnit soldier1;
    LandUnit soldier2(17, 42);
}
```

Konstruktor

- může být definován uvnitř deklarace třídy
- inicializační sekce konstruktoru
  - inicializace atributů
  - předání parametrů konstruktoru rodiče
  - lepší způsob inicializace než přiřazení

```
class LandUnit {
// ...jako předtím...
    LandUnit(int tx, int ty) : x(tx), y(ty) {}
    // všimněte si prázdného těla konstruktoru
};
```

- co když nedefinuju žádný konstruktor?
  - defaultní konstruktor
- kopírovací konstruktor – příště (používá reference)

Destruktor

- úklid objektu poté, co přestal existovat
  - uvolnění paměti
  - zavření souborů, ukončení spojení apod.
- automaticky volán při uvolňování objektu
  - konec platnosti lokální proměnné
  - po zavolání `delete`
- nemá žádné parametry, nelze jej přetěžovat, nevrací žádnou hodnotu
- syntaxe: ~jméno třídy()

```
class LandUnit {
    // ...
    virtual ~LandUnit();
};

LandUnit::~LandUnit() {
        // kód pro uvolnění paměti apod.
}
```

Dědičnost

- třída může dědit od jiné třídy
- v této přednášce jen úplné základy, více si řekneme příště

```
class Archer : public LandUnit {
    int range;
public:
    Archer(int x, int y, int r) : LandUnit(x, y), range(r) {}
    void shootAt(int x, int y);
}
```

- nyní můžeme psát:

```
Archer john(17, 42, 6);
john.move(20, 15);  // používáme zděděnou metodu
john.shootAt(20, 10);
```

Dědičnost (pokr.)

- ukazateli na předka můžeme předat potomka:

```
LandUnit * unit = &john;
unit->move(10, 7);
unit->shootAt(1, 1); // CHYBA! třída LandUnit nemá
                     // metodu shootAt
```

- polymorfismus: dejme tomu, že i třídy `Swordsman` a `Knight` jsou potomky `LandUnit`; potom můžeme psát:

```
Swordsman joe(5, 5);
Knight lancelot(7, 7);
LandUnit * units[3] = { &john, &joe, &lancelot };
for (int i = 0; i < 3; ++i) {
    units[i]->move(i, 15);
}
```

Pozdní vazba – virtual

```
class X {
public:
    void metodaA();
    virtual void metodaB();
};

class Y : public X {
public:
    void metodaA();
    void metodaB();
};

int main() {
    Y y;
    X * px = &y;
    px->metodaA(); // zavolá se X::metodaA(), časná vazba
    px->metodaB(); // zavolá se Y::metodaB(), pozdní vazba
}
```

Abstraktní třída

- abstraktní třída je třída, která neimplementuje nějakou metodu
  - není možno vytvářet objekty abstraktní třídy

```
class LandUnit {
    // ...
    virtual void attack(int x, int y) = 0;
};

void Archer::attack(int x, int y) {
    // kód pro útok
}

int main() {
    LandUnit soldier; // CHYBA!
    LandUnit * unit;  // ukazatel na abstraktní třídu je v pořádku
    Archer john(9, 15);
    unit = &archer;
    unit->attack(10, 7);
}
```

Shrnutí

Objektově-orientované programování

- způsob návrhu řešení problémů pomocí rozdělení na víceméně autonomní celky – objekty
- principy OOP:
  - zapouzdření
  - abstrakce
  - dědičnost
  - polymorfismus

OOP v C++

- třídy – obsahují atributy (data), metody, konstruktory, destruktory,
- přístupová práva – `public`, `protected`, `private`
- metody mají skrytý parametr `this`
- konstruktor slouží k inicializaci objektu
- destruktor slouží k úklidu

Samostudium na příští týden

- najděte si na `http://cppreference.com` dokumentaci k typu `string` a naučte se jej používat
- najděte si na `http://cplusplus.com` tutoriál Basic Input/Output a naučte se základy vstupu a výstupu v C++
- dotazy na použití `string`u mohou být součástí testíku na druhém cvičení, stejně tak i vstup a výstup pomocí `cin` a `cout`