Princípy tvorby softvéru
Typy, logovanie

Robert Lukoťka
lukotka@dcs.fmph.uniba.sk
www.dcs.fmph.uniba.sk/~lukotka

M-255

Typy

Konverzie typov:

- Silné typovanie
- Slabé typovanie
- Žiadne typovanie

Kedy sa to kontroluje

- (Počas písania)
- Compile time
- Run time

Príklady

C: kill

- *kill(int, int)*
  - *kill(SIGUSR1, pid)* alebo *kill(pid, SIGUSR1)*
  - Oboje sa skompiluje. Chyba sa prejavi divným sposobom.
- *kill(Signal, ProcessId)*
  - *kill(SIGUSR1, pid)* alebo *kill(pid, SIGUSR1)*
  - Chyba sa prejaví pri kompilácii.

Príklady

Reťazce + kodovanie:

- $a =$"$lala\backslash aa$"
- Niekde potrebujeme zapísať $a =$"$lala\backslash\backslash aa$", pretože "$\backslash$" je špeciálny znak.
- Takže môžeme chcieť rôzne typy pre normálny a upravený string.
- Typy sa môžu automaticky castovať pomocou konverzných operácií.

Kontrola typov

Kontrola typov je špeciálne dôležitá ak:

- takéto správanie ide cez veľkú časťou
- rozdiel medzi typmi je malý
- podobných typov je viac

Takéto typové chyby je ťažké výnimky systematicky pokrývať testami - najčastejší argument proti typovaniu - akokoľvek rozumné testy by mali zachytiť typové chyby.

Kontrola typov

Mnohé jazyky poskytujú turingovsky úplny compile time typový systém:

```
template <int N> struct Factorial
{
    enum { val = Factorial<N-1>::val * N };
};

template<>
struct Factorial<0>
{
    enum { val = 1 };
};
```

Kontrola typov

Takže typy *std::array<int, 6>* a *std::array<int, Factorial<3>::val>* sú rovnaké.
Ďalší príklad:

```
template<typename T>
inline bool Graph::contains(T &&p) const
            requires RotationPredicate<T>
                     || GraphLocatable<T> {
    decltype(auto) rp2=get_rp(std::forward<T>(p));
    for(auto it=begin();it!=end();it++)
        if (rp2(*it)) return true;
    return false;
}
```

Funkcionálne jazyky

Čisto funkcionálne majú často veľmi silné typové systémy:

- Maybe Monad
- Maybe Monad in C++ - alternatíva k exceptionám (samozrejme, trošku kostrbaté).
- LinkedList - Hardcore funkcionalny prístup v C++ (samozrejme, veľmi kostrbaté).

Logovanie

Prečo potrebujeme logovanie?

- Ako sa dozvieme o problémoch v produkcii?

Je logovanie architektonicky signifikántné?

- Dotýka sa väčšiny / všetkých častí SW - Áno.

Problémy a výzvy

Čo je ťažké:

- Ako logovať aby logov nebolo neúnosne veľa a napiek tomu boli užitočné?
- Je možné v prípade potreby logovať niektoré aspekty detajlnejšie.
- Logovací kód "špiní" business logic.
- Ako logovať v knižniciach
- ...

Logovanie v Pythone

- Logger
- Handler
- Filter
- Formater

Logovanie v Pythone

- *logger = logging.getLogger(__name__)*
  - logery vznikajú v prirodzenej hierarchickej štruktúre
- Úrovne logovania:
  - *logger.debug(...)*
  - *logger.info(...)*
  - *logger.warning(...)*
  - *logger.error(...)*
  - *logger.critical(...)*

Logovanie v Pythone

Logging flow

Logovanie v Pythone

Logger objekty majú o.i.

- *.propagate*
- *.setLevel*
- *.addFilter*
- *.removeFilter*
- *.addHandler*
- *.removeHandler*

Logovanie v Pythone

Handler objekty majú o.i.

- *.setLevel*
- *.setFormatter*
- *.addFilter*
- *.removeFilter*
- *.flush*

Logovanie v Pythone

Priklady implementovaných handler objektov:

- StreamHandler
- FileHandler
- NullHandler
- RotatingFileHandler
- TimedRotatingFileHandler
- SocketHandler
- DatagramHandler
- SMTPHandler
- HTTPHandler

Logovanie v Pythone

Logging Cookbook