Объектно-ориентированное программирование

Object-oriented programming

VI. Полиморфизм

Polymorphism

Что такое OOP?

https://youtube.com/clip/UgkxMjTVfez-AB1gsyvoIpeprL2pc5YxvGjB?si=VIhZaa0uoTkcql6n

leon.brinzan@iis.utm.md

"The evolution of languages from untyped universes to monomorphic and then polymorphic type systems[...], mechanisms for polymorphism such as **overloading**, **coercion**, **subtyping**, and **parameterization**[...], a unifying framework for polymorphic type systems[...] in terms of the typed λ-calculus augmented to include **binding of types by quantification** as well as **binding of values by abstraction**."

L. Cardelli

"On Understanding Types, Data Abstraction, and Polymorphism", 1985

http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf

Виды полиморфизма

- Универсальный
  - параметризация
  - включение (inclusion, sub-typing)
- Специальный (ad-hoc)
  - перегрузка (overloading)
  - приведение (coercion)

leon.brinzan@iis.utm.md

Перегрузка функций

```
int prod(int (&a)[10], int (&b)[10]) {
    int k = 0;
    for(size_t i = 0; i < 10; i++) {
        k += a[i] * b[i];
    }
    return k;
}

std::string prod(std::string (&a)[10], std::string(&b)[10]) {
    std::string k = “”;
    for(size_t i = 0; i < 10; i++) {
        k += a[i] + b[i];
    }
    return k;
}
```

Приведение типов

Явное:

```
int i = (int)abs(negative);
// или
int i = int(abs(negative));
// или
int i = static_cast<int>(abs(negative));
```

Неявное:

```
int *p = malloc(sizeof(int) * 10);
```

Приведение типов и перегрузка

```
3   +   4
3.0 +   4
3   +   4.0
3.0 +   4.0
```

Приведение типов и перегрузка

```
3   + 4
3.0 + 4
3   + 4.0
3.0 + 4.0
```

- оператор “+” перегружен четыре раза
- оператор “+” перегружен два раза: для целых и дробных чисел
- оператор “+” не перегружен, а определен только для дробных чисел

Управление неявным приведением классов в C++

- Создание подходящего **конструктора**, который принимает аргумент заданного типа
- Перегрузка **оператора присваивания**, которая принимает аргумент заданного типа
- Перегрузка **оператора приведения** типа

Управление неявным приведением классов в C++

```
class A { … };

struct B {
    // приведение из типа A (конструктор)
    B(const A &x) {}
    // приведение из типа A (присваивание)
    B& operator=(const A &x) { return *this; }
    // приведение к A
    operator A() { return A(); }
};
```

https://cplusplus.com/doc/tutorial/typecasting/

Супер-типы и под-типы

```
class abstract_data_t {
public:
    iterator find(int) = 0;
    …
};

class vector: public abstract_data_t {
public:
    iterator find(int) override;
    …
private:
    int *p{nullptr};
};
```

Супер-типы и под-типы

```
vector v = {1, 2, 3, 4};
…
list l = {5, 4, 3, 2, 1};
…
abstract_data_t *a = &v;
abstract_data_t *b = &l;
…
a->find(4);
b->find(4);
```

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/polymorphism

Параметризация в C++

```
auto x = prod({1, 2, 3}, {4, 5, 6});

auto y = prod({1.0, 2.0}, {4.5, 5.0});

auto z = prod({"a", "b"}, {"c", "d"});
```

Параметризация в C++

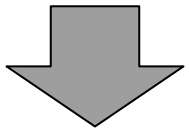
```
template <typename T, size_t n>
auto prod(const T(&a)[n], const T(&b)[n]) {
    T k = 0;
    for(size_t i = 0; i < n; i++) {
        k += a[i] * b[i];
    }
    return k;
}
```

Параметризация в C++

```
template <typename T, size_t n>
auto prod(const T(&a)[n], const T(&b)[n]) {
    T k = T{};
    for(size_t i = 0; i < n; i++) {
        k += a[i] * b[i];
    }
    return k;
}
```

Параметризация в ML

```
fun factorial n =
    if n <= 1 then 1
    else factorial (n-1) * n
```

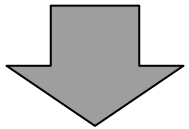


```
val factorial = fn : int -> int
```

Параметризация в ML

```
fun filter pred [] = []
   | filter pred (a::rest) =
   if pred a
   then a::(filter pred rest)
   else (filter pred rest)

filter (fn x => x <> "text") ["text", "notext"]
```

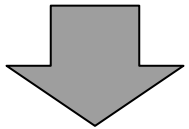


```
val filter = fn : ('a -> bool) -> 'a list -> 'a list
```

Параметризация в ML

```
fun map (f, xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f, xs'))

map (fn x => x + 1) [4, 8, 12, 16]
```



```
val map = fn : ('a -> 'b) * 'a list -> 'b list
```

Выявление типа (type inference)

- Процесс **удовлетворения ограничений** (constraint satisfaction)
- Аппликация функции к аргументу генерирует ограничение, приравнивающее тип домена функции к типу аргумента
- Ограничения разрешаются методом исключения (unification)
  - Если решений нет – ошибка определения типа (overconstrained)
  - Если решений несколько, функция полиморфна, когда есть одно “лучшее” решение (underconstrained)
  - Если строго одно решение – проблема определения типа однозначна (uniquely determined)
- **Перегрузка операторов** создает дополнительные сложности в такой системе

http://www.cs.cmu.edu/~rwh/isml/book.pdf