Семестр 2. Лекция 8. C++11. Шаблоны. STL.

Евгений Линский

5 Мая 2017

Templates in C++11

- Templates aliases (“typedef для темплейтов”)

```
template<typename T>
using fast_vector = std::vector<T,
                        big_pool_allocator<T>>;

fast_vector<int> v;
```

  big_pool_allocator — например, один раз выделяет большой массив, а потом “выдает” из него последовательные участки

- Исправлена проблема с угловыми скобками: `T<U<int>>`.

Variadic templates

```
template <typename T>
T sum(T n) { return n; }

template <typename T, typename ...Args>
T sum(T n, Args... rest) { return n + sum(rest...); }

double d = sum(3, (double)4.3, 5);
```

- "..." отщипывает один аргумент; затем тот же шаблон применяется еще раз, пока не дойдем до базы "шаблонной рекурсии" с одним аргументом (*T sum(T n)*).
- для этого примера компилятор сгенерирует три функции

```
//получено с помощью макроса __PRETTY_FUNCTION__
T sum(T, Args ...) [with T = int; Args = {double, int}]
T sum(T, Args ...) [with T = double; Args = {int}]
T sum(T) [with T = int]
```

Переменное число аргументов в C (printf)

va_start, va_arg, va_end — макросы.

```
void simple_printf(const char* fmt, ...) {
  va_list args;
  //записать в args адрес следующего за fmt параметра на стеке
  va_start(args, fmt);
  while(*fmt != '\0' {
    if(*fmt=='d') {
      //достать со стека переменную типа int
      int i = va_arg(args, int)
      // здесь должен быть код, который
      // выводит int на экран с помощью putc
    }
    fmt++;
  }
  va_end(args);
}
//Труднообнаруживаемые ошибки
printf("%s", 5);
print("%d %d", 4);
printf("%d", 4, 5);
```

Переменное число аргументов в C++11 (printf)

```
void printf(const char *s) {
    while (*s) {
        if (*s == '%' && *(++s) != '%')
            throw std::runtime_error("invalid format");
        std::cout << *s++;
    }
}
template<typename T, typename... Args>
void printf(const char *s, T value, Args... rest) {
    while (*s) {
        if (*s == '%' && *(++s) != '%') {
            std::cout << value;
            printf(++s, rest...);
            return;
        }
        std::cout << *s++;
    }
    throw std::logic_error("extra arguments
      provided to printf");
}
```

unordered_set, unordered_map

```
template<class Key,
         class Hash = std::hash<Key>,
         class KeyEqual = std::equal_to<Key>,
         class Allocator = std::allocator<Key>>
    class unordered_set;
```

- Типовая реализация: карманы и списки
- Для своего класса нужно реализовать сравнение на равенство и хэш функцию
- *find* is O(1) but O(n) worst case (все элементы оказались в одном кармане)
- *insert* is O(1) but O(n) worst case (может вызвать рехэширование — “увеличение числа карманов”)

unordered_set, unordered_map

```
class Point {
private:
    int x, y;
public:
    bool operator == (const Point& rhs) const{
        return x == rhs.x && y == rhs.y;
    }
};
namespace std {
    template <>
    struct hash<Point> {
        size_t operator()(Point const & p) const {
            return (std::hash<int>()(p.getX()) * 51
                + std::hash<int>()(p.getY()));
        }
    };
}
std::unordered_set<Point> points;
//Можно реализовать функторы PointComparator, PointHasher
std::unordered_set<Point, PointComparator, PointHasher> ps;
```

unique_ptr, shared_ptr

```
struct Foo {
    ...
    void bar() { std::cout << "Foo::bar\n";  }
};

void f(const Foo &) { std::cout << "f(const Foo&)\n"; }

int main() {
    std::unique_ptr<Foo> p1(new Foo);  // p1 owns Foo
    if (p1) p1->bar();

    // now p2 owns Foo
    std::unique_ptr<Foo> p2(std::move(p1));
    f(*p2);

    p1 = std::move(p2);  // ownership returns to p1
    if (p1) p1->bar();

    // p1 = p2; // error, assign operator is =delete
}
```

other

- контейнер (т.е. есть итераторы) template<class T, std::size_t N> struct array;

```
std::array<int, 3> a2 = {1, 2, 3};
```

- std::tuple — "как pair", но только с произвольным числом полей (реализован на variadic templates)

```
auto t = std::make_tuple("String", 5.2, 1);
std::cout << std::get<0>(t) << ' '
          << std::get<1>(t) << ' '
          << std::get<2>(t) << '\n';
```

- regex, regex_search, regex_match

```
regex reg("[a-zA-Z_][a-zA-Z_0-9]*\\.[a-zA-Z0-9]+");
regex_search("Print readme.txt", reg);
```

std::function

std::function — "шаблонная обертка" для всего, что можно вызвать (callable): функция, функтор, лямбда.

```
void execute(const vector<function<void ()>>& fs){
    for (auto& f : fs) f();
}
void plain_old_func() {
    cout << "old plain function" << endl;
}
struct functor {
    void operator()() const{
        cout << "functor" << endl;
    }
};
int main() {
    vector<function<void ()>> x;
    x.push_back(plain_old_func);
    functor functor_instance;
    x.push_back(functor_instance);
    x.push_back([] () { cout << "lambda" << endl; });
    execute(x);
}
```

std::bind

std::bind — позволяет создать “обертку” над функцией, уменьшив число ее параметров (“удаленным” параметрам задаются конкретные значения).

```
void show_text(const string& t) {
    cout << "TEXT: " << t << endl;
}

int main() {
    vector<function<void ()>> x;
    // void f() { show_text("Hello"); }
    function <void ()> f = bind(show_text, "Hello");
    x.push_back(f);
    execute(x);
}
```

std::bind, placeholder

```
using namespace std::placeholders;
int multiply(int a, int b) { return a * b; }
int main() {
  //_1 -- placeholder;
  // 1ый параметр функции f подставляется
  // на второе место в multiply
  // int f(int _1) { return multiply(5, _1); }
  auto f = bind(multiply, 5, _1);
  cout << out: << f(6); // out:  30
}
```

std::thread

```
void f1(int n){
    std::cout << "f1: " << n << std::endl;
}

void f2(int& n){
    n++;
}

int main(){
    int m = 45;
    std::thread t1(f1, m);
    // std::ref нужен, чтобы у функции, выполняемой
    // потоком была тип void f(int &x)
    std::thread t2(f2, std::ref(m));
    t1.join();
    t2.join();
    cout << m;
}
```

std::thread

Примерой с лямбдой.

```
std::vector<std::thread> threads;

for(int i = 0; i < 5; ++i){
    threads.push_back(std::thread([](){
        std::cout << std::this_thread::get_id()
        << std::endl;
    }));
}

for(auto& thread : threads){
    thread.join();
}
```

А можно передать фунтор и сохранить в нем результат исполнения.