C/C++: Лекция 6

Воробьев Д.В

09.10.2020

Шаблоны

“Точное соответствие лучше приведения типа”

```
template<class T1, class T2>
void foo(T1 a, T2 b) { std::cout << 1; };

template<class T1>
void foo(T1 a, int b) { std::cout << 2; };

int main() {
    int a = 1;
    double b = 1.0;

    // 1
    foo(a, b);
}
```

“Частная версия лучше общей”

```
template<class T1, class T2>
void foo(T1 a, T2 b) { std::cout << 1; };

template<class T1>
void foo(int a, T1 b) { std::cout << 2; };

int main() {
    int a = 1;
    double b = 1.0;

    // 2
    foo(a, b);
    return 0;
}
```

“Специализация для первого сверху primary template”

```
template<class T>
void foo(T) {
    std::cout << 1;
};

template<class T>
void foo(T*) {
    std::cout << 2;
};

template<>
void foo(int*) {
    std::cout << 3;
};

int main() {
    int* a = new int(1);
    // 3
    foo(a);
}
```

```
template<class T>
void foo(T) {
    std::cout << 1;
};

template<>
void foo(int*) {
    std::cout << 3;
};

template<class T>
void foo(T*) {
    std::cout << 2;
};

int main() {
    int* a = new int(1);
    // 2
    foo(a);
}
```

Шаблоны с переменным количеством аргументов

Template parameter pack

Шаблонный параметр, принимающий 0 или более параметров.

variadic template

Шаблон с хотя бы 1 template parameter pack

Примеры

Класс

```
template<typename ... Tail>
struct X {};

int main() {
    X<> x;
    X<int> y;
    X<int, double> z;
}
```

Функция

```
template<typename ... Tail>
void foo()(Tail ... args) {}

int main() {
    foo();
    foo(1);
    foo(1, 1);
}
```

“Откусывание” шаблонных аргументов

```
template<typename Tail>
void Print(Tail tail) {
    std::cout << tail;
}

template<typename Tail, typename ... Head>
void Print(Tail tail, Head ... head) {
    std::cout << tail;
    Print(head...);
}
```

Компаратор в std::sort

std::sort компаратор передается 3 аргументом

```
template<class RandomIt, class Compare>
void sort(RandomIt first, RandomIt last, Compare comp);
```

Стандартные компараторы

std::less

```
template<typename T>
struct less {
    bool operator()(const T& a, const T& b) const {
        return a < b;
    }
};
```

std::greater

```
template<typename T>
struct greater {
    bool operator()(const T& a, const T& b) const {
        return a > b;
    }
};
```

Исключения

Оператор throw и конструкция try...catch

```
int main() {
    try {
        throw 1;
    } catch (int i) {
        std::cout << i;
    }
}
```

Разница между exception и runtime error

Exception

Объект для обработки сценария-отклонения от стандартного сценария исполнения программы и обработки этого сценария-отклонения.

RunTime error

Сценарий отклонения от стандартного сценария исполнения программы , приведший к некорректному заверешнию программы.

Ошибки не являющиеся исключениями

Segmentation fault

```
int main() {
    int x[10];
    try {
        x[100000000000] = 20;
    } catch(...) {
        std::cout << 1;
    }
}
```

Генерация исключения оператором new

new

```
int main() {
    try {
        int* x = new int[100000000000];
    } catch(std::bad_alloc& e) {
        std::cout << e.what();
    }
}
```

Повторное пробрасывание исключений

```
int main() {
    try {
        int* x = new int[100000000000];
    } catch(std::bad_alloc& e) {
        std::cout << e.what();
        // пробросили
        throw e;
    }
}
```

Приведения типов при ловле исключений

Приведение типов (для встроенных типов) не выполняется

```
int main() {
    try {
        throw 1;
    } catch (double x) {
        std::cout << "double";
    } catch (int x) {
        std::cout << "int";
    }
}
```

Выбор catch в сценарии "подходят разные блоки"

Выбирается первый подходящий блок. Далее не идем.

```
int main() {
    try {
        throw 1;
    } catch (int x) {
        std::cout << 1;
    } catch (...) {
        std::cout << 2;
    }
}
```

Исключения и наследовании

При выборе блока приведение типа выполняется

```
int main() {
    try {
        throw Derived();
    } catch (Base& e) {
        // Приходим сюда
    }  catch (Derived& d) {
        std::cout << "d";
    }
}
```

Правила ловли исключений

Правило

catch блоки объявляются от частного к общему (т.к. выполняется первый подходящий catch)

```
int main() {
    try {
        int* x = new int[100000000000];
    } catch (std::bad_alloc& e) {
        // mym
        std::cout << "bad_alloc";
    } catch (std::exception& e) {
        std::cout << "exception";
    } catch (...) {
        std::cout << "all";
    }
}
```

Перехвата исключений по ссылке

```
int main() {
    try {
        std::string s;
        // копия 1
        throw s;
    } catch (std::string& e) {
        // копии нет
    }
}
```

Перехвата исключений по значению

```
int main() {
    try {
        std::string s;
        // копия 1
        throw s;
    } catch (std::string e) {
        // копия 2
    }
}
```