Лекция 8. const, mutable, static, inline.

Евгений Линский

const в C

Назначение:

- избавить себя от описок
- сделать код более понятным для читающего

Из прошлых лекций:

```
const double pi = 3.14159;
char const * ptr1; //pointer to const
char * const ptr2; //const pointer
```

```
size_t strlen(const char *s);
```

const в C

```
void f(const char* str) {
  char* s1 = (char*)str;
  s1[0] = 'x';
}
```

```
char s1[] = "Hello, world!";
const char* s2 = "Hello, world!"
f(s1); //will work
f(s2); //run-time error
```

s2 — строковая константа (литерал):

- Хранится в отдельном области памяти (рядом с глобальными переменными)
- Память, выделенная под такую константу, может быть помечена как read-only

const в C++: поле, параметр метода

const у поля: нужно инициализировать в конструкторе

```
class Array {
private:
    const size_t size;
    int *data;
public:
    Array(size_t s);
    ~Array();
    Array(Array& a);
    Array& operator=(Array &a);
    ...
};
Array::Array(size_t s): size(s) {...}
```

- Модификатор const у параметра метода: метод не меняет параметр.
- Куда стоит вписать?

const в C++: параметр метода

```
class Array {
private:
    const size_t size;
    int *data;
public:
    Array(size_t s);
    Array(const Array& a);
    Array& operator=(const Array &a);
    ~Array();
    ...
};
```

const в C++: у метода

const у метода: метод не меняет полей класса

```
class Array {
private:
    const size_t size;
    int *data;
public:
    Array(int s);
    ...
    int get(int i) const;
    void set(int i, int v);
    size_t get_size();
};
```

```
void print(const Array& a) {
  for(int i = 0; i < a.get_size(); i++)
    printf("%d", a.get(i));
}
```

1. Почему не компилируется print?

const в C++: у метода

const у метода: метод не меняет полей класса

```
class Array {
private:
    const size_t size;
    int *data;
public:
    Array(int s);
    ...
    int get(int i) const;
    void set(int i, int v);
    size_t get_size();
};
```

```
void print(const Array& a) {
  for(int i = 0; i < a.get_size(); i++)
    printf("%d", a.get(i));
}
```

1. Почему не компилируется print?
2. Должно быть: size_t get_size() const;

mutable

```
class Matrix {
...
  double get(size_t i, size_t j) const;
  void set( size_t i, size_t j, double value);
  double determinant() const;
};
```

- deteminant() — вычислительно сложная операция; лучше сохранить в поле, чтобы каждый раз не вычислять заново.
- Поля:
  1. double det;
  2. bool det_up_to_date; — сбрасывается в set, устанавливается в determinant();
- Что не так?

mutable

```
class Matrix {
...
  double get(size_t i, size_t j) const;
  void set( size_t i, size_t j, double value);
  double determinant() const;
};
```

- deteminant() — вычислительно сложная операция; лучше сохранить в поле, чтобы каждый раз не вычислять заново.
- Поля:
  1. double det;
  2. bool det_up_to_date; — сбрасывается в set, устанавливается в determinant();
- Что не так?
- determinant() — это const метод, ему нельзя изменять поля det и det_up_to_date

mutable

- Решение 1: убрать const у determinant()

mutable

- Решение 1: убрать const у determinant()
- Проблема: не получится вызвать determinant() внутри print(const Matrix& m)
- Решение 2:
  mutable double det;
  mutable bool det_up_to_date;
  Основное применение mutable — кэширование данных.

static у полей

Из прошлых лекций: локальная static переменная сохраняет свое значение между вызовами функции (хранится в области глобальных переменных).

```
void func() {
  static int num_func_calls = 0;
  printf("%d", num_func_calls++);
}
```

Статическое поле: одно на все объекты

```
//person.h
class Person {
private:
  static int last_id;
  int id;
  char name[256];
  int age;
  ...
public:
  Person(const char *name, int age);
};
```

static у полей

```
//person.cpp
int Person::last_id = 0; //init of static field

Person::Person(const char *name, int age) {
  ...
  id = last_id++;
}
```

```
main() {
  // Person.last_id = 0
  Person p1("Vasya", 12); // p1.id = 0, Person.last_id = 1
  Person p2("Masha", 11); // p2.id = 1, Person.last_id = 2
}
```

static метод

- У static метода нет this => не может получить доступ к нестатическим полям.
- Может быть вызыван через имя класса

```
class Person {
  ...
  static int get_next_id() { return last_id; }
}

main() {
  int i = Person::get_next_id();
}
```

```
class Point {
...
  static int distance(const Point& p1, const Point& p2);
};
main() {
  Point p1(1, 3); Point p2(2, 3);
  Point::distance(p1, p2); // вместо p1.distance(p2);
}
```

static метод

error: double definition:

```
//tree.cpp
tree_t node;
void fill_nodes() { tree_t t; ... }
//list.cpp
list_t node;
void fill_nodes() { list_t l;... }
```

static и линковка

static у глобальной переменной или функции — идентификатор используется только для разрешения имен в рамках одного файла

```
//tree.cpp
static tree_t node;
static void fill_nodes() { tree_t t; ... }
//list.cpp
static list_t node;
static void fill_nodes() { list_t l;... }
```

inline и линковка

“Заинлайнить” — оптимизация компилятора.

До:

```
int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
main( ) {
  int c = ...;
  int b = ...;
  int d = max(c, b);
}
```

inline и линковка

После:

```
main( ) {
  int c = ...;
  int b = ...;
  if(c > b)
    d =  c;
  else
    d = b;

}
```

Компилятор принимает такие решения самостоятельно. Можно дать совет -O0, -O1, -O2, -O3 (уровень оптимизации).

inline и линковка

a.cpp

```
int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
```

b.cpp

```
f( ) {
  int c = ...;
  int b = ...;
  int d = max(c, b);
}
```

- Сможет ли компилятор заинлайнить?

inline и линковка

a.cpp

```
int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
```

b.cpp

```
f( ) {
  int c = ...;
  int b = ...;
  int d = max(c, b);
}
```

- Сможет ли компилятор заинлайнить?
- Нет. На стадии компиляции определение (definition) функции max недоступно при комиляции b.cpp

inline и линковка

a.h

```
int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
```

b.cpp

```
#include "a.h"
f( ) { int c = ...; int b = ...; int d = max(c, b); }
```

c.cpp

```
#include "a.h"
g( ) { int e = ...; int f = ...; int g = max(e, f); }
```

- Сможет ли компилятор заинлайнить?

inline и линковка

a.h

```
int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
```

b.cpp

```
#include "a.h"
f( ) { int c = ...; int b = ...; int d = max(c, b); }
```

c.cpp

```
#include "a.h"
g( ) { int e = ...; int f = ...; int g = max(e, f); }
```

- Сможет ли компилятор заинлайнить?
- Сможет, но будет ошибка double definition на линковке.

inline и линковка

a.h

```
inline int max(int a, int b) {
  if(a > b)
    return a;
  else
    return b;
}
```

b.cpp

```
#include "a.h"
f( ) { int c = ...; int b = ...; int d = max(c, b); }
```

c.cpp

```
#include "a.h"
g( ) { int c = ...; int b = ...; int d = max(c, b); }
```

inline и методы

```
class Array {
  ...
  size_t get_size() { return size; }
};
```

или

```
class Array {
  ...
  size_t get_size();
};
inline size_t Array::get_size() { return size; }
```

Еще про staic: Singletone

- Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software.
- Шаблон проектирования или паттерн (англ. design pattern) — повторяемая архитектурная конструкция (решение проблемы проектирования).
- Singletone — один из паттернов:
  - Надо спроектировать класс, у которого в программе будет только один объект. Попытка создания второго объекта должна вызывать ошибку компиляции.
  - Примеры: class Settings (хранение настроек), class Logger (журнал событий в программе).

Singletone

Settings.h

```
class Settings {
private:
  static Settings *p;
  Settings();
public:
  static Settings* getInstance() {
    if (p == NULL) p = new Settings();
    return p;
  }
  void load();
};
```

Setting.cpp

```
Settings* Settings::p = NULL;
```

main.cpp

```
Settings *s = Settings::getInstance();
s->load();
```