Программирование на языке C++

Шаблов Анатолий
anatoliishablov@gmail.com

ИТМО, весенний семестр 2024

ITMOre than a UNIVERSITY

Специальные методы классов

Конструктор

Конструктор — это специальная не статическая функция-элемент класса, которая используется для инициализации объектов своего классового типа.

Имя функции совпадает с именем класса.

При создании объекта класса всегда вызывается один конструктор.

Конструктор нельзя вызвать явно.

У класса может быть произвольное число конструкторов (в том числе ноль).

Деструктор

Деструктор — это специальная функция-элемент, которая вызывается, когда заканчивается время жизни объекта. Цель деструктора освободить ресурсы, которые объект мог получить за время своего существования (или сделать какую-то финализацию).

Имя функции `~ClassName`.

У деструктора не может быть аргументов. По умолчанию `noexcept`.

При любом удалении объекта класса вызывается его деструктор.

У класса может быть только один деструктор.

Деструктор можно вызвать явно.

Особенности конструкторов и деструкторов

- Нет возвращаемого значения.
- Нет cv-квалификаторов.
- Нет ref-квалификаторов.

Пример конструкторов и деструктора

```
struct S {
  S();              // 1
  S(int);           // 2
  S(char, double);  // 3
  S(const S &);     // 4
  ~S();
};

S get_s();

int main() {
  S s1;      // 1st constructor is called
  S s2(10);  // 2nd constructor is called
  {
    const S &s_ref = get_s();
    S s3('a', 0);    // 3rd constructor is called
    S s4 = get_s();  // probably, 4th constructor is called
                     // then S::~S()
  }                  // S::~S() for s3, s4 and tmp object is called
}  // S::~S() for s1 and s2 is called
```

Список инициализации членов

```
struct S {
  S(int x) : a(x), b(a * 2) {}

  int a;
  int b = 10;
};

struct SS : S {
  SS() : S(20), c(b) {}

  int c;
};
```

Варианты инициализации в списке

```
struct A {
  A() {}
  int a = 10;
};
struct B {
  B(int x) : b(x) {}  // direct initialization
  int b;
};
template <class... Ts>
struct C : A, B, Ts... {
  C(int x, int y, const Ts &...ts)
      : B{x}  // list initialization
        ,
        Ts(ts)...  // pack expansion
        ,
        c(y)  // direct
  {}
  int c;
};
```

Порядок инициализации объекта класса

1. Базовые классы инициализируются в том порядке, в каком они перечислены в списке наследования.

2. Не статические поля класса инициализируются в том порядке, в каком они объявлены в определении класса.

3. Выполняется тело конструктора.

Порядок уничтожения объекта класса

1. Тело деструктора.

2. Не статические поля класса разрушаются в порядке, обратном порядку их объявления в классе.

3. Для базовых классов деструкторы вызываются в порядке, обратном их порядку в списке наследования.

Некорректное завершение выполнения конструктора

```
struct S {
  S() : s{"Hello, world"} {
    v.resize(100);
    next = new S(1);
    throw 1;
  }

  std::string s;
  std::vector<int> v;
  S* next{nullptr};
};

S s;
```

Делегирование конструкторов

```
class Foo {
 public:
  Foo(char x, int y) {}
  Foo(int y) : Foo('a', y) {}  // Foo(int) делегирует Foo(char,int)
};
```

У делегирующего конструктора список инициализации членов должен состоять из одного элемента – вызова целевого конструктора.

Делегирующий конструктор не может быть рекурсивным.

Наследование конструкторов

```
struct A {
  A(int, double, char);
};

struct B : A {};

struct C : A {
  using A::A;
};

int main() {
  B b(10, 0.5, 'a');   // error
  C c(10, 0.5, '\n');  // OK
}
```

explicit конструкторы

```
struct A {
  A(int);
};

struct B {
  explicit B(int);
};

A f(A const &) {
  return 10;  // OK, A::A(int) is called
}

B g(B const &) {
  return 10;     // error
  return B(10);  // OK, B::B(int) is called
}

int main() {
  f(10);     // OK, A::A(int) is called
  g(10);     // error
  g(B(10));  // OK B::B(int) is called
}
```

Конструкторы приведения типа

Любые не-explicit конструкторы.

Могут участвовать в последовательности неявного приведения типа.

```
struct A {
  A(int);
};

void f(A);

f(10);
```

Удалённые и автоматически создаваемые методы

Некоторые методы могут быть созданы автоматически – как неявно, так и явно.

Запретить неявную автоматическую генерацию можно “удалив” метод.

```
struct A {};  // A::A() is implicitly-defined

struct B {
  B() = default;  // B::B() implicit definition is forced
};

struct C {
  C() = delete;  // C::C() is not defined
};
```

Некоторые методы могут быть удалены неявно.

Методы, создаваемые автоматически

- Конструктор по умолчанию
- Конструктор копирования
- Конструктор перемещения
- Деструктор
- Операторы присваивания

Конструктор по умолчанию

```
struct S {
  S();  // default constructor
};

S s;  // default constructor is called

struct A {
  A();
};
struct B : A {
  B();
};

B b;  // A::A() and B::B() are called
```

Автоматическое создание конструктора по умолчанию

```
struct A {};

struct B {
  B() {}
};
```

Удалённый конструктор по умолчанию

- Наличие ссылочного поля без инициализатора.
- Наличие не статического константного поля без явно определённого конструктора по умолчанию или инициализатора.
- Наличие не статического поля без инициализатора, имеющего удалённый либо недоступный конструктор по умолчанию.
- Наличие прямого или виртуального предка с удалённым либо недоступным конструктором по умолчанию.
- Наличие прямого или виртуального предка с удалённым либо недоступным деструктором.

Удалённый деструктор

```
struct S {
  ~S() = delete;
};

static S ss;              // error
S s;                      // error
S* ps = new S;            // Ok
```

Копирование и перемещение

Конструктор копирования

Это не шаблонный конструктор, принимающий первым аргументом lvalue ссылку на объект того же типа, что и класс конструктора, который может быть вызван с одним аргументом.

Конструктор копирования вызывается всегда, когда объект класса инициализируется из lvalue выражения того же типа (класса).

Пример конструкторов копирования

```
struct S {
  S(int);
  S(const S &);  // copy constructor 1
  S(S &);        // copy constructor 2
};

S f(S s) {
  return s;  // copy constructor ? is called
}

const S s1(10);
S s2 = s1, s3(s1);  // copy constructor ? is called
f(S(10));           // copy constructor ? is called
```

ITMO more than a UNIVERSITY

Автоматическое создание конструктора копирования

- `C(const C &)`, если
  - Все прямые или виртуальные предки имеют конструктор копирования `B(const B &)`.
  - Все нестатические поля имеют конструктор копирования `M(const M &)`.
- `C(C &)` в противном случае.

Такой конструктор выполняет копирование всех подобъектов создаваемого объекта, в обычном порядке инициализации.

Удалённый конструктор копирования

- Наличие не копируемого нестатического поля.
- Наличие не копируемого прямого или виртуального предка.
- Наличие прямого или виртуального предка с удалённым или недоступным деструктором.
- Наличие rvalue ссылочного не статического поля.

Наличие пользовательского конструктора перемещения или перемещающего оператора присваивания ≡ отсутствие автоматически созданного конструктора копирования.

Копирующий оператор присваивания

Это не шаблонный оператор присваивания, принимающий ровно один аргумент, имеющий тип *T*, *T&* или *T const &* (если *T* – это данный класс).

```
struct S {
  S(const S &);
  S &operator=(const S &);
};

S s1, s2 = s1;  // copy constructor is called
s1 = s2;        // copy assignment is called
```

Предполагаемое поведение такого оператора – копирование значения одного объекта данного класса в другой объект этого же класса.

Автоматическое создание копирующего оператора присваивания

- `C& operator=(C const &)`, если
  - Все прямые или виртуальные предки имеют оператор копирования `B& operator=(B const &)`.
  - Все нестатические поля имеют оператор копирования `M& operator=(M const &)`.
- `C& operator=(C &)` в противном случае.

Такой оператор выполняет копирование всех подобъектов создаваемого объекта, в обычном порядке инициализации.

Удалённый копирующий оператор присваивания

- Наличие не присваиваемого не статического поля.
- Наличие не присваиваемого прямого или виртуального предка.
- Наличие ссылочного не статического поля.
- Наличие константного не статического поля, чей тип не является классом.

Наличие пользовательского конструктора перемещения или перемещающего оператора присваивания ≡ отсутствие автоматически созданного копирующего оператора присваивания.

Конструктор перемещения

Это не шаблонный конструктор, принимающий первым аргументом rvalue ссылку на объект того же типа, что и класс конструктора, может быть вызван с одним аргументом.

Конструктор перемещения вызывается всегда, когда объект класса инициализируется из xvalue выражения того же типа (класса).

Предназначение перемещающего конструктора – забрать содержимое у аргумента - временного объекта и, по возможности, избежать затрат на копирование этого содержимого.

При этом аргумент должен остаться в валидном, но не обязательно определённом, состоянии.

Пример конструкторов копирования и перемещения

```
struct S {
  S() = default;
  S(const S &);  // copy constructor
  S(S &&);       // move constructor
};

void f(S s);

int main() {
  S s1;
  f(s1);            // copy constructor is called
  f(S{});           // move constructor is called
  f(std::move(s1)); // move constructor is called
}
```

Автоматическое создание конструктора перемещения

Создаётся, если

- Нет ни одного пользовательского конструктора перемещения.
- Нет ни одного пользовательского конструктора копирования.
- Нет ни одного пользовательского оператора присваивания (копирующего или перемещающего).
- Нет пользовательского деструктора.

Такой конструктор выполняет инициализацию всех подобъектов создаваемого объекта, в обычном порядке инициализации, из xvalue выражения соответствующего объекта-донора.

Удалённый конструктор перемещения

- Наличие не статического поля, для которого нельзя вызвать конструктор перемещения.
- Наличие прямого или виртуального предка, для которого нельзя вызвать конструктор перемещения.
- Наличие прямого или виртуального предка, для которого нельзя вызвать деструктор.

Удалённый конструктор перемещения не участвует в разрешении перегрузки.

```
struct S {
  S(const S &);
  S(S &&) = delete;
};

S get_s();

S s = get_s();  // OK
```

Перемещающий оператор присваивания

Это не шаблонный оператор присваивания, принимающий ровно один аргумент, имеющий тип *T&&* или *T const &&*.

Предполагаемое поведение перемещающего оператор присваивания – перемещение значения одного объекта данного класса в другой объект данного класса.

```
struct S {
  S &operator=(S &&);
};

S s1, s2;
s2 = S();             // move assignment is called
s1 = std::move(s2);   // move assignment is called
```

Автоматическое создание перемещающего оператора присваивания

Создаётся, если

- Нет ни одного пользовательского конструктора перемещения.
- Нет ни одного пользовательского конструктора копирования.
- Нет ни одного пользовательского копирующего оператора присваивания.
- Нет пользовательского деструктора.

Такой оператор выполняет перемещающее присваивание всех подобъектов левого объекта из соответствующих подобъектов правого, в обычном порядке инициализации.

Удалённый перемещающий оператор присваивания

- Наличие не статического константного поля.
- Наличие не статического ссылочного поля.
- Наличие не статического поля, для которого нельзя вызвать перемещающее присваивание.
- Наличие прямого или виртуального предка, для которого нельзя вызвать перемещающее присваивание.

Удалённый оператор перемещения не участвует в разрешении перегрузки.

Copy elision

Материализация

prvalue → xvalue

```
T f();

int a = 1 + 2;
int && b = 1 + 2;
T x = f();
M const & m = f().member;
```

Copy elision

Отсутствие материализации:

- Инструкция return, когда её операнд – prvalue того же типа, что и тип возвращаемого значения функции.
- Инициализация переменной, когда выражение инициализации – prvalue того же типа.

```
T f() { return T(); }
T x = T(T(T(f())));  // only default constructor of T is called
```

NRVO

Named return value optimization: если в инструкции return операнд обозначает локальную переменную, которая:

- Не является параметром функции.
- Объект которой не является volatile.
- Объект которой имеет автоматический тип размещения.
- Тип объекта которой совпадает с типом возвращаемого значения функции.

То компилятор может не генерировать код вызова конструктора копирования/перемещения, как если бы это была copy elision.

```
T f() {
  T t;
  return t;
}
T x = f();  // only default constructor of T is called
```