Множественное наследование

Мотивация

В процессе проектирования у вас может возникнуть потребность в создании класса, обладающего свойствами сразу нескольких классов.

```
class StackMax { ... };

class StackMin { ... };

class StackMinMax;  // обладает свойствами обоих классов
```

Синтаксис

```
class StackMinMax : public StackMin, public StackMax { ... };
```

Наследоваться можно любым возможным способом (`public`, `private`, `protected`).

При этом каждый модификатор будет распространяться только на тот класс, перед которым он написан.

Синтаксис

Все правила обычного наследования (затенения, срезки и т.д.) остаются прежними, никаких неожиданностей.

```
void StackMinMax::Push(int value) {
  StackMin::Push(value);
  StackMax::Push(value);
}
```

```
StackMinMax smm;
smm.Push(1);  // вызывается StackMinMax::Push
```

Синтаксис

Если в базовых классах есть методы с одним именем (возможно, с разными аргументами), то при попытке вызова одного из них возникнет ошибка.

Такие методы надо вызывать с полным именем (явно указать базовый класс).

```
class StackMinMax : public StackMin, public StackMax { ... };
```

```
StackMinMax smm;

smm.Top();  // CE, если в StackMinMax метод не переопределен

smm.StackMin::Top();  // Ok
smm.StackMax::Top();  // Ok
```

Порядок вызова конструкторов/деструкторов при множественном наследовании

Порядок вызова конструкторов/деструкторов при множественном наследовании

- Базовые классы создаются в соответствии с порядком перечисления в списке базовых классов.

```
class StackMinMax : public StackMin, public StackMax {
 public:
  StackMinMax() : StackMax(3) {  // сначала StackMin(), затем StackMax(3)
      // ...
  }
};
```

- Порядок вызова деструкторов обратный - сначала деструктор производного класса, а затем базовых классов в обратном порядке.

Проблема "ромбовидного" наследования

Проблема "ромбовидного" наследования

Самое время вспомнить, что `StackMin` и `StackMax` унаследованы от `Stack`.

```
class Stack { ... };

class StackMin : public Stack { ... };

class StackMax : public Stack { ... };

class StackMinMax : public StackMin, public StackMax { ... };
```

Сколько всего стеков в классе `StackMinMax`?

Проблема "ромбовидного" наследования

Самое время вспомнить, что `StackMin` и `StackMax` унаследованы от `Stack`.

```
class Stack { ... };

class StackMin : public Stack { ... };

class StackMax : public Stack { ... };

class StackMinMax : public StackMin, public StackMax { ... };
```

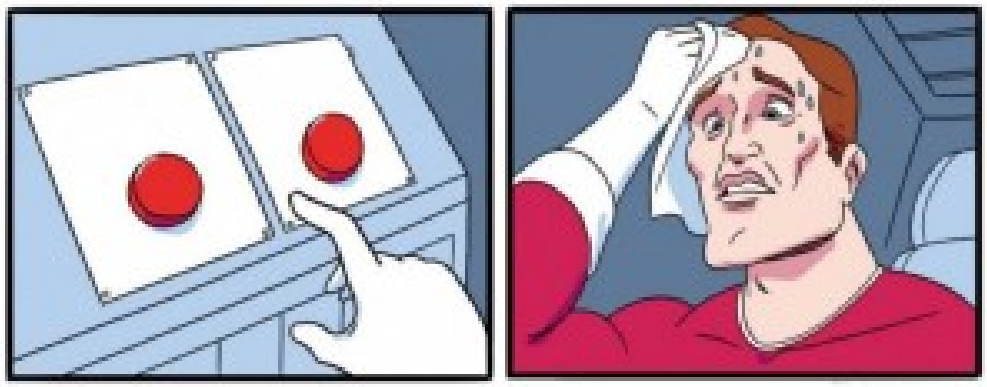
Сколько всего стеков в классе `StackMinMax`?

Ответ: 2

Проблема "ромбовидного" наследования

С точки зрения компилятора `StackMin` и `StackMax` это два отдельных класса, каждый из которых содержит свой экземпляр `Stack`.

```
StackMinMax smm;
Stack s = smm;  // CE, какой стек нужно скопировать?
Stack* s_ptr = &smm;  // CE, на какой стек нужно указывать?
Stack& s_ref = smm;  // CE, на какой стек ссылаться?
smm.StackMin::Stack::Top();  // Для вызова метода Stack::Top
```



Данная проблема называется *проблемой ромбовидного наследования*.

Виртуальное наследование

(не путать с виртуальными методами и полиморфным наследованием!)

Для решения проблемы ромбовидного наследования необходимо виртуально наследовать класс, который потенциально может встречаться несколько раз в иерархии:

```
class Stack { ... };

class StackMin : virtual public Stack { ... };

class StackMax : virtual public Stack { ... };

class StackMinMax : public StackMin, public StackMax { ... };
```

Теперь в классе `StackMinMax` ровно один экземпляр `Stack` (`StackMin` и `StackMax` используют один `Stack` на двоих).

Виртуальное наследование

(не путать с виртуальными методами и полиморфным наследованием!)

В общем случае, каждое виртуальное вхождение базового класса объединяется в одну сущность, а под каждое невиртуальное вхождение выделяется своя часть.

```
class A { ... };

class B : virtual A { ... };

class C : virtual A { ... };

class D : A { ... };

class E : B, C, D { ... };
```

Теперь в классе `E` два экхемпляра класса `A`: одно виртуальное (делится между классами `B`, `C`) и одно невиртуальное (принадлежит классу `D`).

Конструкторы и деструкторы при виртуальном наследовании

Конструкторы и деструкторы

К сожалению, виртуальное наследование привносит новую проблему:

```
class StackMin : virtual public Stack {
 public:
  StackMin() : Stack(1) { ... }
  // ...
};

class StackMax : virtual public Stack {
 public:
  StackMax() : Stack(2) { ... }
  // ...
};

class StackMinMax : public StackMin, public StackMax {
 public:
  StackMinMax() = default;  // Stack(1) или Stack(2)?
  // ...
};
```

Кто создает `Stack`? `StackMin` или `StackMax`?

Конструкторы и деструкторы

Правила вызова конструкторов немного меняются

```
class StackMin : virtual public Stack {
 public:
  StackMin() : Stack(1) { ... }
  // ...
};

class StackMax : virtual public Stack {
 public:
  StackMax() : Stack(2) { ... }
  // ...
};

class StackMinMax : public StackMin, public StackMax {
 public:
  StackMinMax() = default;  // Stack() !
  // ...
};
```

`Stack` создастся по умолчанию, вызовы в `StackMin` и `StackMax` игнорируются!

Конструкторы и деструкторы

Для вызова конкретного конструтора, нужно прописать его явно

```
class StackMin : virtual public Stack {
 public:
  StackMin() : Stack(1) { ... }
  // ...
};

class StackMax : virtual public Stack {
 public:
  StackMax() : Stack(2) { ... }
  // ...
};

class StackMinMax : public StackMin, public StackMax {
 public:
  StackMinMax() : Stack(3) { ... }  // Stack(3), StackMin(), StackMax()
  // ...
};
```

Важно! Это работает только в случае виртуального наследования!