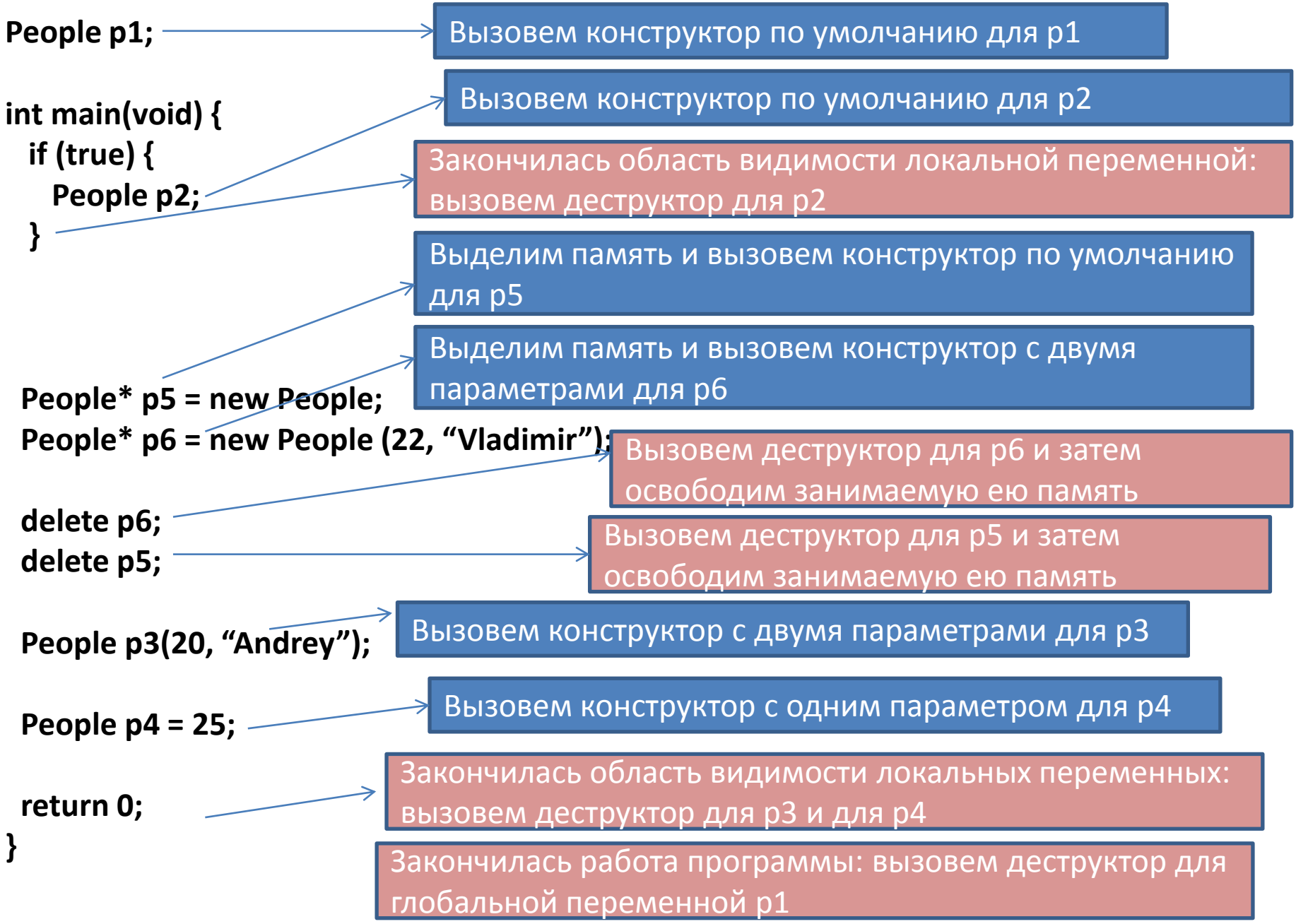

People p1;
Вызовем конструктор по умолчанию для p1
int main(void) {
if (true) {
People p2;
}
Вызовем конструктор по умолчанию для p2
Закончилась область видимости локальной переменной: вызовем деструктор для p2
Выделим память и вызовем конструктор по умолчанию для p5
Выделим память и вызовем конструктор с двумя параметрами для p6
People* p5 = new People;
People* p6 = new People (22, “Vladimir”);
Вызовем деструктор для p6 и затем освободим занимаемую ею память
delete p6;
delete p5;
Вызовем деструктор для p5 и затем освободим занимаемую ею память
People p3(20, “Andrey”);
Вызовем конструктор с двумя параметрами для p3
People p4 = 25;
Вызовем конструктор с одним параметром для p4
return 0;
}
Закончилась область видимости локальных переменных: вызовем деструктор для p3 и для p4
Закончилась работа программы: вызовем деструктор для глобальной переменной p1

People array[5];

Создадим массив из пяти объектов. Для каждого из них вызовем конструктор по умолчанию. Если конструктора по умолчанию нету – то программа не скомпилируется

People arr[2] = {People(17), People (19, “Anna”)};

Создадим массив из двух объектов. Для каждого из них вызовем нужный нам конструктор

Закончилась область видимости переменных array и arr: вызовем деструктор для каждого из содержащихся в них объектов

```
class People {
public:
  int age;
  char name[100];

  People(){};
  People(int age, const char* nm)  {
    this->age = age;
    strcpy(this->name, nm);
  };
  ~People(){};
}

void Show(People value) {
  …
}

People p1(22, “Inga”);
Show(p1);
```

Вызовем конструктор с двумя параметрами для p1

Вызовем функцию Show, передав значение p1 как параметр

Для создания объекта value на основе данных p1 должен быть применен конструктор копирования, но в классе People его нету. Поэтому производится побитовое копирование всех данных из p1 в value.
В данном случае нас это устраивает.

Закончилась область видимости параметра value: вызовем для него деструктор. У нас он пустой.

Закончилась область видимости переменной p1: вызовем для нее деструктор. У нас он пустой.

```
class People {
public:
  int age;
  char* name;

  People() {name = new char[100];};
  People(int age, const char* name) {
    this->age = age;
    this->name = new char[strlen(name) + 1];
    strcpy(this->name, name);
  };
  ~People() {delete name;};
}

void Show(People value) {
  …
}

People p1(22, “Inga”);
Show(p1);
```

Вызовем конструктор с двумя параметрами для p1

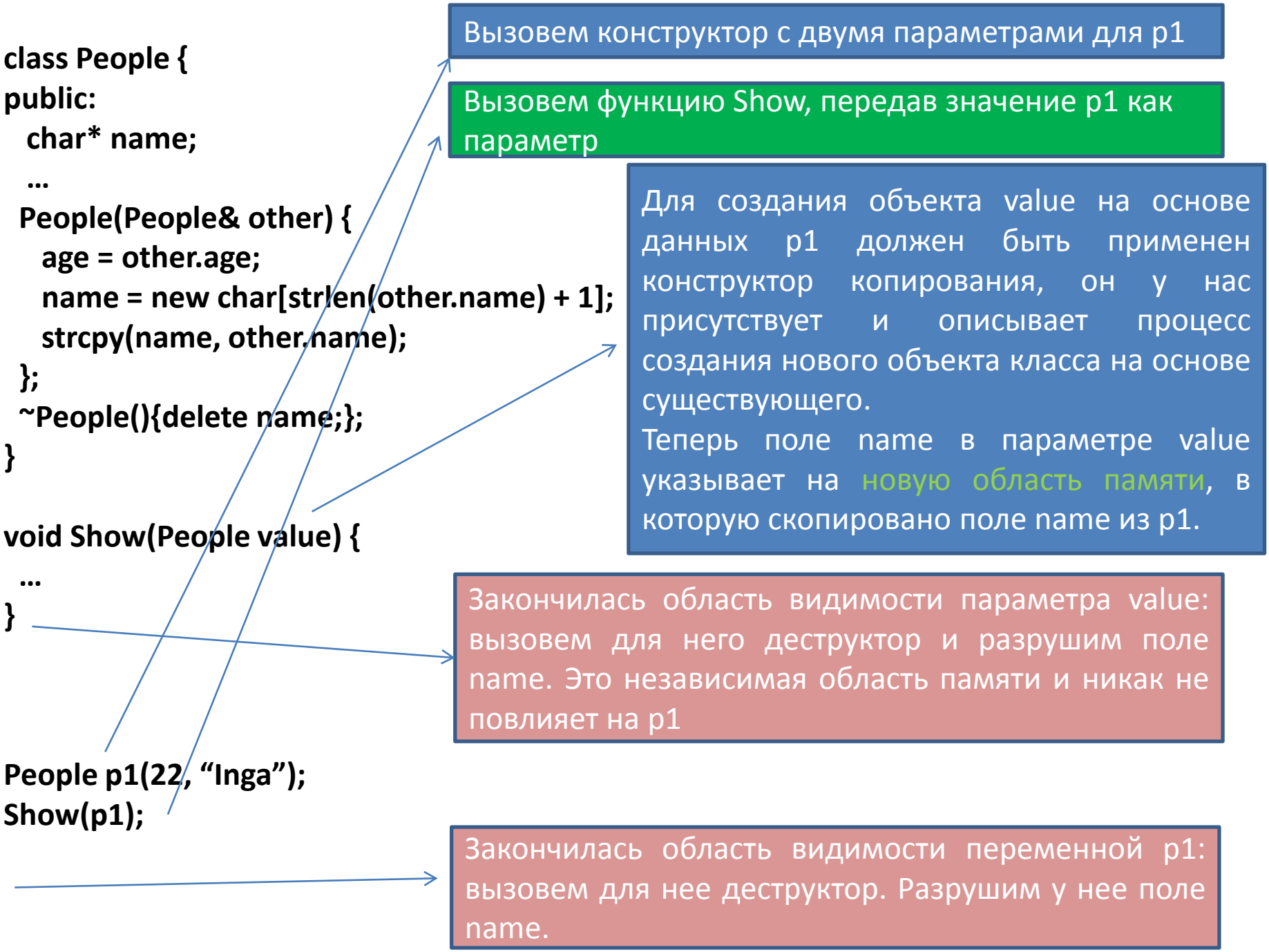
Вызовем функцию Show, передав значение p1 как параметр

Для создания объекта value на основе данных p1 должен быть применен конструктор копирования, но в классе People его нету. Поэтому производится побитовое копирование всех данных из p1 в value.
Теперь поле name в параметре value указывает на ту же область памяти, что и поле name в p1. Это в дальнейшем вызовет проблемы.

Закончилась область видимости параметра value: вызовем для него деструктор. Разрушая поле name у value мы разрушим его и у p1.

Поле name у p1 теперь невалидно

Закончилась область видимости переменной p1: вызовем для нее деструктор. Пытаемся разрушить уже удаленное поле name


class People {
public:
char* name;
…
People(People& other) {
age = other.age;
name = new char[strlen(other.name) + 1];
strcpy(name, other.name);
};
~People(){delete name;};
}
void Show(People value) {
…
}
People p1(22, “Inga”);
Show(p1);
Вызовем конструктор с двумя параметрами для p1
Вызовем функцию Show, передав значение p1 как параметр
Для создания объекта value на основе данных p1 должен быть применен конструктор копирования, он у нас присутствует и описывает процесс создания нового объекта класса на основе существующего.
Теперь поле name в параметре value указывает на новую область памяти, в которую скопировано поле name из p1.
Закончилась область видимости параметра value: вызовем для него деструктор и разрушим поле name. Это независимая область памяти и никак не повлияет на p1
Закончилась область видимости переменной p1: вызовем для нее деструктор. Разрушим у нее поле name.

```
class People {
public:
  int age;
  char* name;

  People(){name = new char[100];};
  People(int age, const char* name) {
    this->age = age;
    this->name = new char[strlen(name) + 1];
    strcpy(this->name, name);
  };
  ~People(){delete name;};
}

void Show(People& value) {
  …
}

People p1(22, “Inga”);
Show(p1);
```

Вызовем конструктор с двумя параметрами для p1

Вызовем функцию Show, передав ссылку на p1 как параметр

value является тем же объектом, что и p1. Передается только указатель на него. Никакого нового объекта не создается. Изменение value вызовет и изменение p1.

Закончилась область видимости параметра value. Однако он является на самом деле указателем (ссылкой) и поэтому сам объект не будет разрушаться.

Закончилась область видимости переменной p1: вызовем для нее деструктор. Разрушаем у нее поле name.