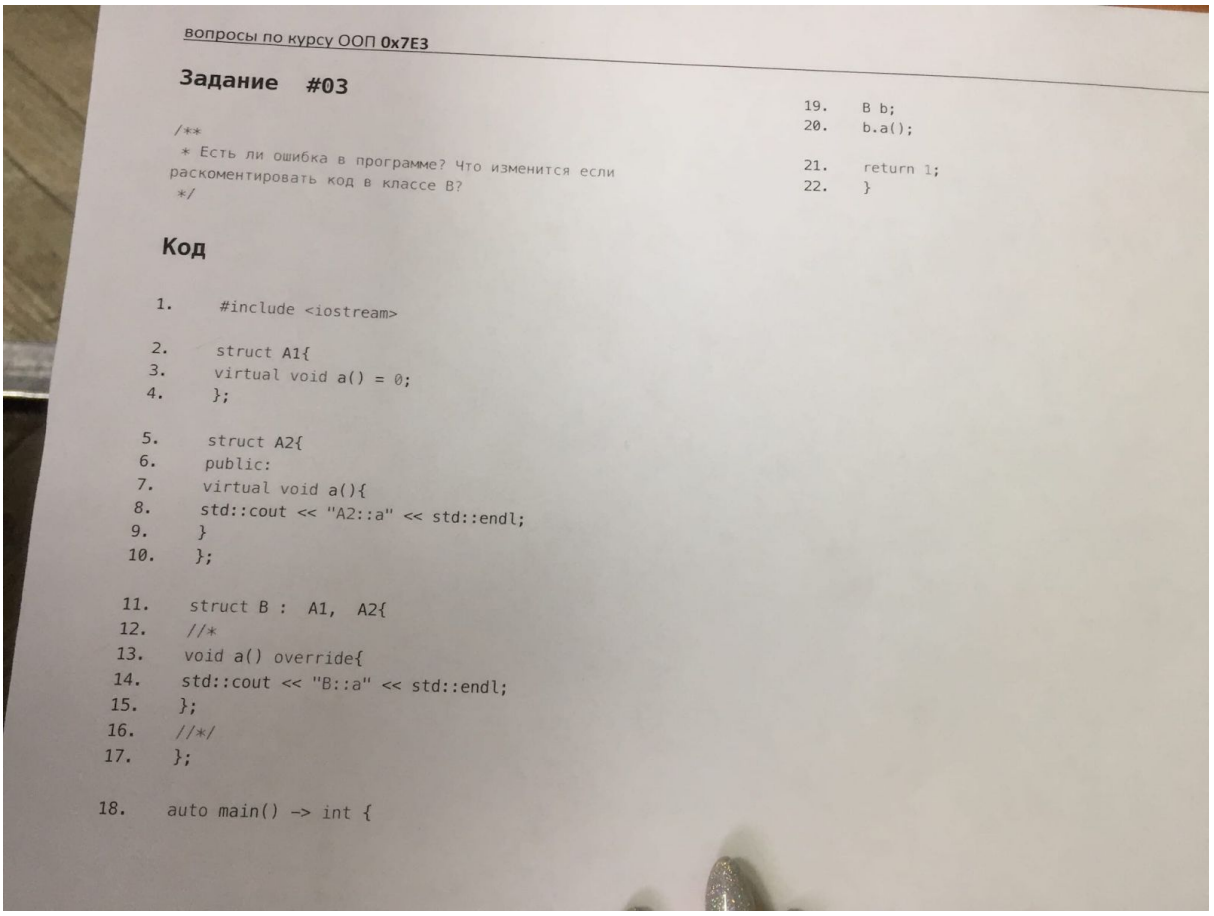
Задача 15: шаблонный стек (реализовать методы push и pop), доп вопрос: что нужно добавить, чтобы он работал в многопоточной программе (в приватное поле добавить мьютекс и локать его в каждом методе)

Задача X: один поток генерирует случайные числа, а другой после завершения генерации выводит их(использовать условные переменные)

Задача 28:
Специализация шаблонов:
Сделайте шаблон, который определяет являются ли два класса эквивалентными.
Решение:
template<class T, class U>
struct is_same : std::false_type {} ;
template<class T>
struct is_same <T, T> : std::true_type {} ;



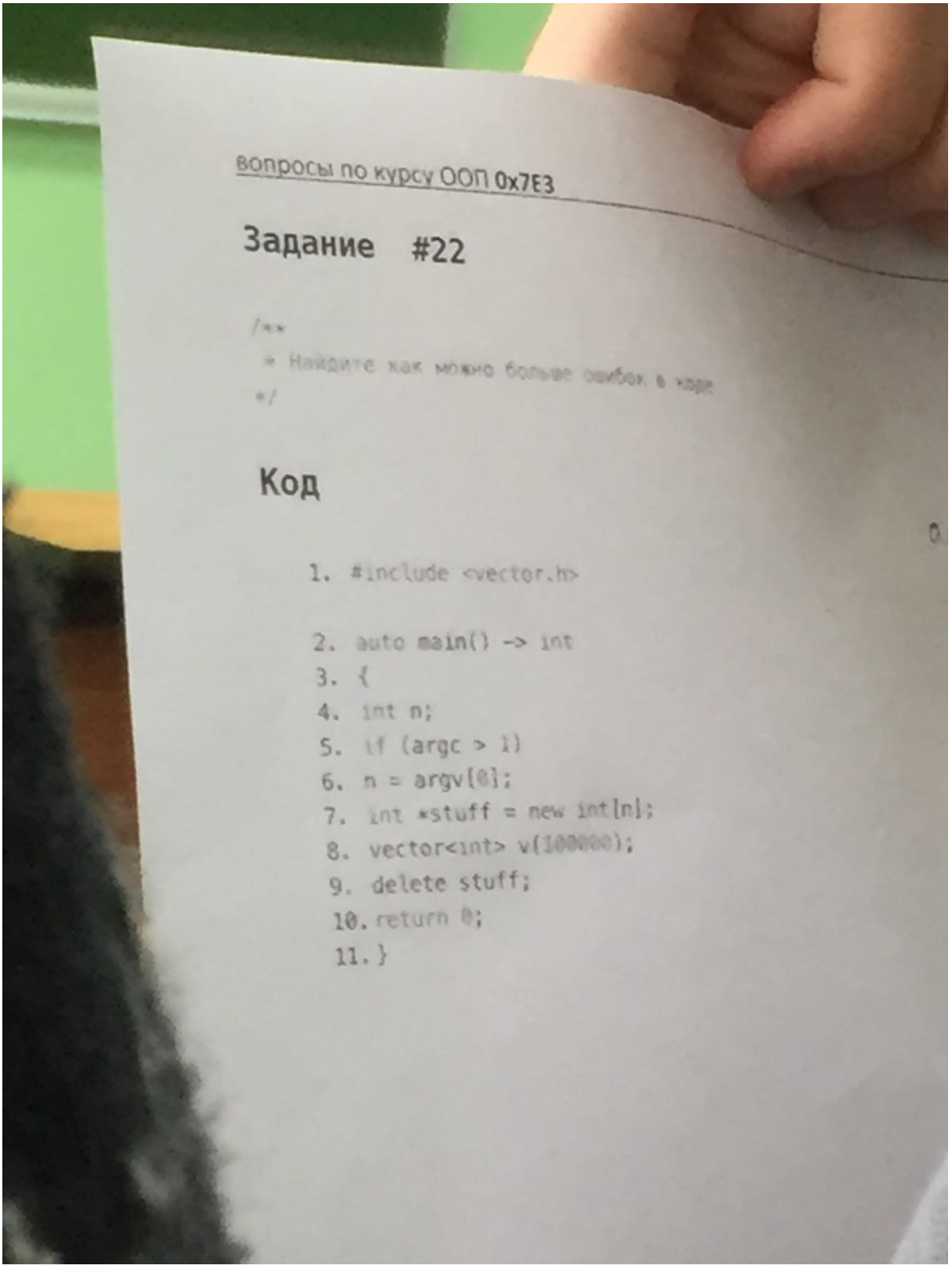
вопросы по курсу ООП 0x7E3

Задание #03

```
/**
 * Есть ли ошибка в программе? Что изменится если
раскоментировать код в классе B?
 */
```

Код

```
#include <iostream>

struct A1{
virtual void a() = 0;
};

struct A2{
public:
virtual void a(){
std::cout << "A2::a" << std::endl;
}
};

struct B :  A1,  A2{
//*
void a() override{
std::cout << "B::a" << std::endl;
};
//*/
};

auto main() -> int {
B b;
b.a();

return 1;
}
```



вопросы по курсу ООП 0x7E3

Задание #22

```
/**
 * Найдите как можно больше ошибок в коде
*/
```

Код

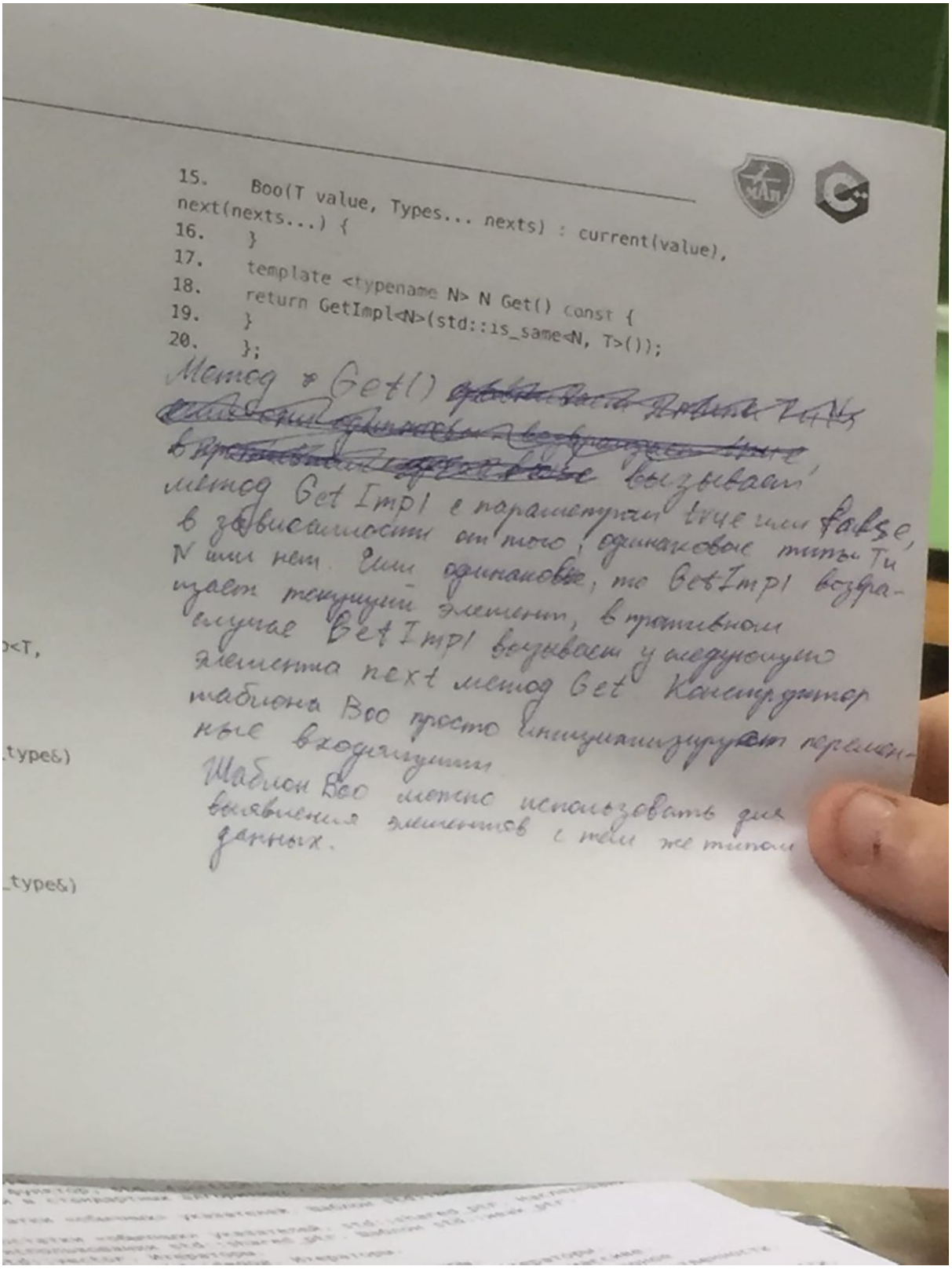
```
1. #include <vector.h>

2. auto main() -> int
3. {
4. int n;
5. if (argc > 1)
6. n = argv[0];
7. int *stuff = new int[n];
8. vector<int> v(100000);
9. delete stuff;
10. return 0;
11. }
```

Задание #06

```
/**
 * Расскажите, что делает шаблон Boo. Как его можно
использовать?
 * Расскажите сколько шаблонов вы видите в программе?
 */
```

Код

```
1.  #include <iostream>
2.  #include <type_traits>

3.  template <typename... Args> class Boo {
4.  };

5.  template <typename T, typename... Types> class Boo<T,
Types...> {
6.  Boo<Types...> next;
7.  T current;
8.  template <typename N> N GetImpl(const std::false_type&)
const {
9.  return next.template Get<N>();
10. }

11. template <typename N> T GetImpl(const std::true_type&)
const {
12. return current;
13. }
14. public:
```



```
15.     Boo(T value, Types... nexts) : current(value),
next(nexts...) {
16.     }
17.     template <typename N> N Get() const {
18.     return GetImpl<N>(std::is_same<N, T>());
19.     }
20.     };
```

Метод Get() ~~[illegible]~~ вызывает метод GetImpl с параметром true или false, в зависимости от того, одинаковые типы T и N или нет. Если одинаковые, то GetImpl возвращает текущий элемент, в противном случае GetImpl вызывает у следующего элемента next метод Get. Конструктор шаблона Boo просто инициализирует переменные входящими.

Шаблон Boo можно использовать для выявления элементов с теми же типами данных.

>T,

type&)

_type&)

Задание #07

```
/**
 * Найдите в данной программе ошибку. Объясните, как ее
исправить.
 */
```

Код

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>

using namespace std::literals::chrono_literals;

struct Data {
Data(const char* n) : name(n) {};
std::string name;
std::vector<unsigned> debitAmounts;
~Data() {std::cout << "dtor" << std::endl;}
};

struct Bar {
std::thread workerThread;
std::vector<std::string> names;

void ProcessData(std::shared_ptr<Data> dataPtr) {
workerThread = std::thread([&] { // change to =
std::this_thread::sleep_for(1s);
std::cout << "start" << std::endl;
for (auto debit : dataPtr->debitAmounts) {
if (debit % 67 == 0) {
std::cout << "Ok";
names.push_back(dataPtr->name);
}
}
});
}
};

auto main() -> int {
Bar b;
b.ProcessData(std::make_shared<Data>("Ivanov"));
b.workerThread.join();
return 1;
}
```

...дание #05

```
/**
 * Найдите ошибку в программе. В чем заключается ошибка?
 * Как нужно дополнить класс Foo что бы ее устранить?
 */
```

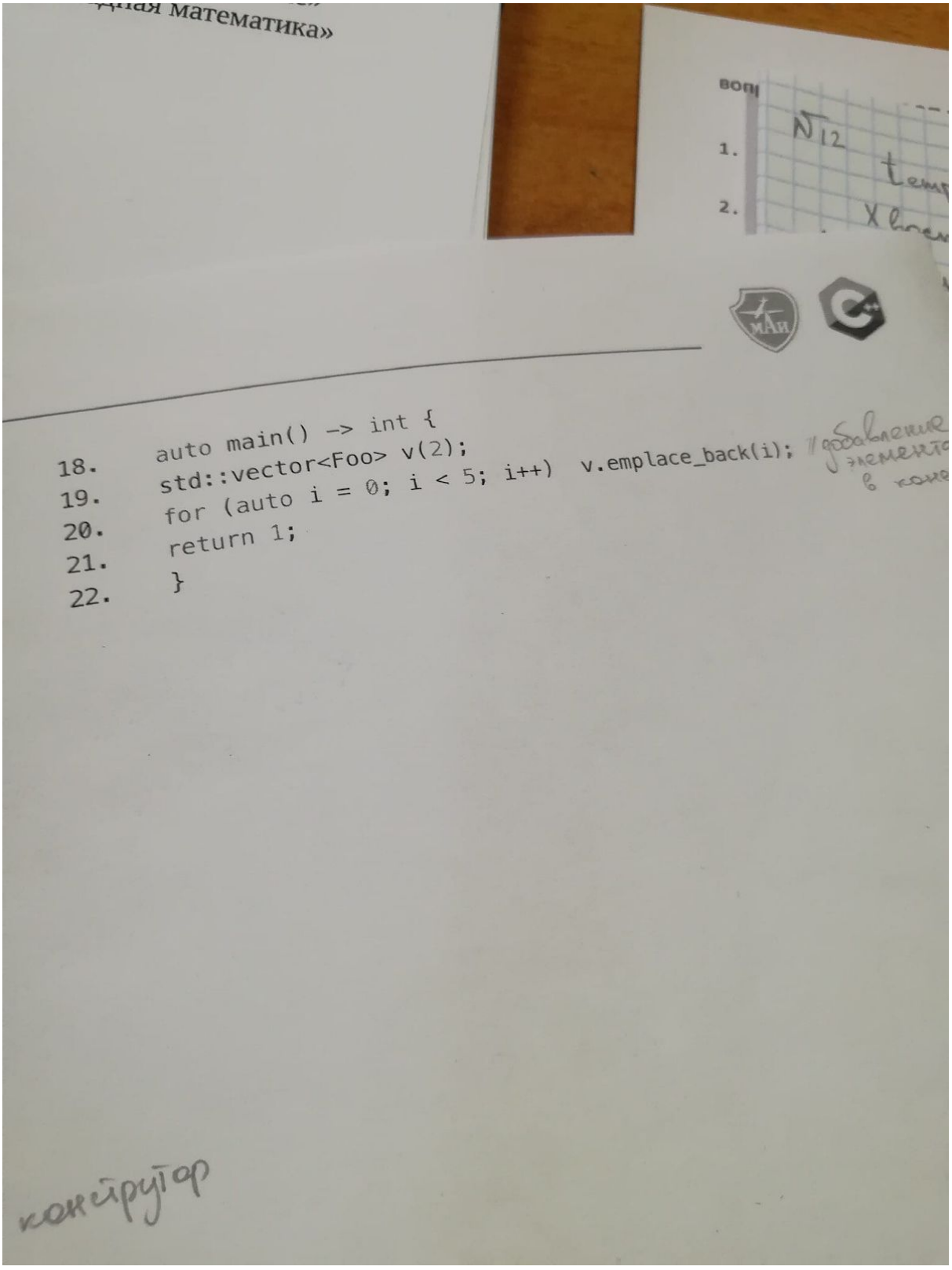
Код

```
#include <iostream>
#include <vector>

class Foo {
int intMember;
char* ptrMember;
public:

Foo() : intMember(0), ptrMember(nullptr) {
std::cout << "Default constructor" << std::endl;
}

Foo(int i) : intMember(i), ptrMember(new char[i]) {
std::cout << "Constructor" << std::endl;
}

~Foo() {
std::cout << "Delete:" << std::endl;
delete[] ptrMember;
```

ошибку в программе. В чем заключается ошибка?
* Как нужно дополнить класс Foo что бы ее устранить?
*/

Код

```
#include <iostream>
#include <vector>

class Foo {
int intMember;
char* ptrMember;
public:

Foo() : intMember(0), ptrMember(nullptr) {
std::cout << "Default constructor" << std::endl;
}

Foo(int i) : intMember(i), ptrMember(new char[i]) {
std::cout << "Constructor" << std::endl;
}

~Foo() {
std::cout << "Delete:" << std::endl;
delete[] ptrMember;
}
};
```



```
18.  auto main() -> int {
19.  std::vector<Foo> v(2);
20.  for (auto i = 0; i < 5; i++)  v.emplace_back(i); // добавление элемента в конец
21.  return 1;
22.  }
```

конструктор