2. Подход с явным инстанцированием

Если известны все типы, которые будут использоваться:

cpp Copy Download

```cpp
// my_template.hpp
#pragma once

template<typename T>
class MyTemplate {
public:
    void foo(T value);
};

// Объявляем явные инстанциации
extern template class MyTemplate<int>;
extern template class MyTemplate<double>;
```

cpp Copy Download

```cpp
// my_template.cpp
#include "my_template.hpp"

// Реализация
template<typename T>
void MyTemplate<T>::foo(T value) {
    // ...
}

// Явное инстанцирование
template class MyTemplate<int>;
template class MyTemplate<double>;
```

Как использовать:

cpp Copy Download

```cpp
// main.cpp
#include "my_template.hpp"

int main() {
    MyTemplate<int> obj1;    // Работает
    MyTemplate<double> obj2; // Работает
    // MyTemplate<char> obj3; // Ошибка линковки - нет явного инстанцирования
}
```

Преимущества:

- Уменьшение времени компиляции
- Сокрытие реализации
- Контроль над допустимыми типами

3. Альтернативный вариант (разделение через `.ipp`)

Для лучшей организации кода:

cpp Copy Download

```cpp
// my_template.hpp
#pragma once

template<typename T>
class MyTemplate {
public:
    void foo(T value);
};

#include "my_template.ipp" // Подключаем реализацию
```

cpp Copy Download

```cpp
// my_template.ipp
template<typename T>
void MyTemplate<T>::foo(T value) {
    // Реализация
}
```

Почему `.ipp`?

Это условное расширение для "implementation of template", чтобы отличать от обычных `.cpp`.