Евгений, здравствуйте. Что касается диплома, то он выполнен, работа соответсвует ожидаемому по ТЗ. Вы пропустили момент по структуре:

Требования к структуре и реализации программы:  
2. Для каждого класса должна быть создана пара «заголовочный файл + файл исходного кода».

Он не очень принципиальный, тем более у вас бюджет времени ограничен. Если хотите, исправьте когда будет время, и напишите в дискорд, я посмотрю еще раз. Что касается расположения проектный файлов. У вас сейчас получается так:

12345Race\

Race.sln

Race.vcxproj

dinam\_lib\_Race\

dinam\_lib\_Race.vcxproj

Дополнительно, есть дублирование файлов dinam\_lib\_Race.h, dinam\_lib\_Race.cpp в обоих каталогах. Это запутывает. В хорошем варианте у вас должно быть решение на верхнем уровне, библиотека вложена:

12345678**Race**.sln

**Race**.vcxproj

**main**.cpp

**transports**\

**transports**.vcxproj

**ground\_transport**.h

**air\_transport**.h

...

Подумайте над этим, сейчас получается грязно. Что касается применения ООП, я бы назвал это скорее имитация объектного подхода. Ведь вам удобнее было бы написать так:

123**double** **calc\_camel\_time**(**int** distance);

**double** **calc\_centaur\_time**(**int** distance);

...

Но это уже императивный стиль, а не объектный. При настоящем ООП вы создаёте такой базовый класс, чтобы вы могли использовать его, не заботясь о конкретной реализации транспорта:

1234567891011*// Файл transport.h, с интерфейсом для пользователя, доступен в main*

**enum** **class** **TransportType** {

Ground,

Air

};

**class** **Transport** {

**public**:

**virtual** **const** **char**\* **getName**() **const** = 0; *//название транспорта, для отображения в таблице результатов*

**virtual** TransportType **getType**() **const** = 0; *//тип, чтобы выбирать в какой гонке будет участвовать*

**virtual** **double** **calcTimeRide**(**int** distance) **const** = 0; *//расчёт времени по дистанции, математическая модель из ТЗ*

};

Далее вы могли бы с помощью иерархии классов сделать уже реализацию интерфейсов:

12345678910111213141516171819*//Общий класс наземного транспорта*

**class** GroundTransport : public Transport {

**public**:

TransportType getType() **const** **override** {

**return** TransportType::Ground;

}

*//TODO: добавить общие параметры для наземных транспортов, чтобы не было дублирования в расчётах*

};

*//Соник - наземный транспорт, всегда проходит дистанцию за 1 час*

**class** Sonic : public GroundTransport {

**public**:

**const** char\* getName() **const** {

**return** "Sonic";

}

double calcTimeRide(int distance) **const** **override** {

**return** 1;

}

};

После этого в main вам можно всегда работать с указателем на транспорт, если сделать с помощью фабричной функции, то привязки к конкретным реализациям участников вообще не будет:

123456789101112131415161718192021222324252627282930313233////////////////////////////////////////////////////////////////

// файл transport\_factory.h содержит только `extern Transport\*\* makeTransports(size\_t& total\_transports);`, доступен в main

//фабричная функция для создания транспортов

Transport\*\* makeTransports(size\_t& total\_transports)

{

total\_transports = 2;

Transport\*\* trans\_arr = new Transport\*[total\_transports];

trans\_arr[0] = new Sonic();

trans\_arr[1] = new SonicBrother();

return trans\_arr;

}

int main()

{

//вызываем фабрику для создания списка обьектов

size\_t total\_transports = 0;

Transport\*\* transports = makeTransports(total\_transports);

//выполняем логику - просто показываем имя и за сколько пройдет дистанцию

**for** (int i = 0; i < total\_transports; i++)

{

std::cout << "Name:" << transports[i]->getName() << std::endl;

std::cout << "Time for 100:" << transports[i]->calcTimeRide(100) << std::endl;

}

//освобождаем память

**for** (int i = 0; i < total\_transports; i++)

{

**delete** transports[i];

}

**delete** transports;

return 0;

}

Надеюсь, вам будет полезен такой обзор метода ООП, который противоположен по своей сути императивному стилю. При добавлении нового транспорта в моём случае main даже не измениться. Вы добавляете еще один класс, расширяете фабричную функцию одной строчкой и всё, дело в шляпе. Тем не менее, вы продемонстрировали хорошую работу с моделированием иерархий, также использовали неймспейсы. Успехов!

**1 вопрос и работа готова.**

я получаю непонятные ошибки при подключении динамической библиотеки - подскажите как их устранить, основные моменты я в интернете уже читал но так и не смог разобраться что не так в моей работе.

Написать ответ



**Антон Ласточкин**

Ответ эксперта

Евгений, здравствуйте. У меня выводятся понятные ошибка при подкючении библиотеки. Давайте разберем первую:

1**error** LNK2019: ссылка на неразрешенный внешний символ "public: double \_\_cdecl dinam\_lib\_Race::all\_terrain\_boots::funk\_all\_terrain\_boots(int)" (?funk\_all\_terrain\_boots@all\_terrain\_boots@dinam\_lib\_Race@@QEAANH@Z) в функции main.

* error LNKXXX нам говорит о том, что ругается линковщик, компилятору всё понравилось, это уже хорошо.
* ссылка на неразрешенный внешний символ - значит линковщик видит какую-то функцию, но не может найти для неё объектного кода, который компилятор сделал.
* dinam\_lib\_Race::all\_terrain\_boots::funk\_all\_terrain\_boots - точное название кого он не нашёл
* (?funk\_all\_terrain\_boots@ - это промежуточное представление в объектном коде, нам не нужно. Для супер точного позиционирования.
* в функции main. Ага, значит это маин не смог найти.

Далее, я захожу в файл с длл. Там в начале есть макросы для импорта/экспорта:

12345**#ifdef DINAMLIBRACE\_EXPORTS**

**#define MATHPOWERLIBRARY\_API \_\_declspec(dllexport)**

**#else**

**#define MATHPOWERLIBRARY\_API \_\_declspec(dllimport)**

**#endif**

Отлично, проверяю, для сборки библиотеки определён макрос DINAMLIBRACE\_EXPORTS? Да, всё отлично, значит MATHPOWERLIBRARY\_API будет работать для экспорта функций и классов. Теперь перехожу к методу funk\_all\_terrain\_boots:

123456789**class** all\_terrain\_boots : public ground { *// БОТИНКИ-ВЕЗДЕХОДЫ*

**public**:

all\_terrain\_boots() {

**this**->speed = 6;

**this**->time\_to\_out = 60;

}

double funk\_all\_terrain\_boots(int long\_distance);

};

Ага, тут нет разрешения на экспорт, понятно, почему не видит main. Исправляем:

123456789 **class** all\_terrain\_boots : public ground { *// БОТИНКИ-ВЕЗДЕХОДЫ*

**public**:

MATHPOWERLIBRARY\_API all\_terrain\_boots() {

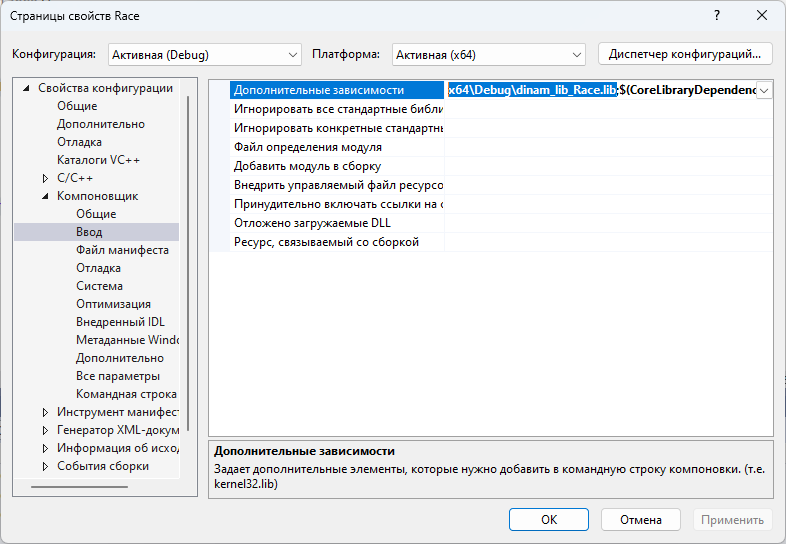
**this**->speed = 6;

**this**->time\_to\_out = 60;

}

double MATHPOWERLIBRARY\_API funk\_all\_terrain\_boots(int long\_distance);

};

Собираем, всё равно линковщик не доволен. Проверяю, подцеплена ли библиотека (lib-файл) вприницпе. Переходим в свойства race, компоновщик и ввод. Там пусто. Добавляем:  
  
Далее, я почитил каталоги и пересобрал. Ошибка пропала. Примерно так надо рассуждать при компоновке библиотек.

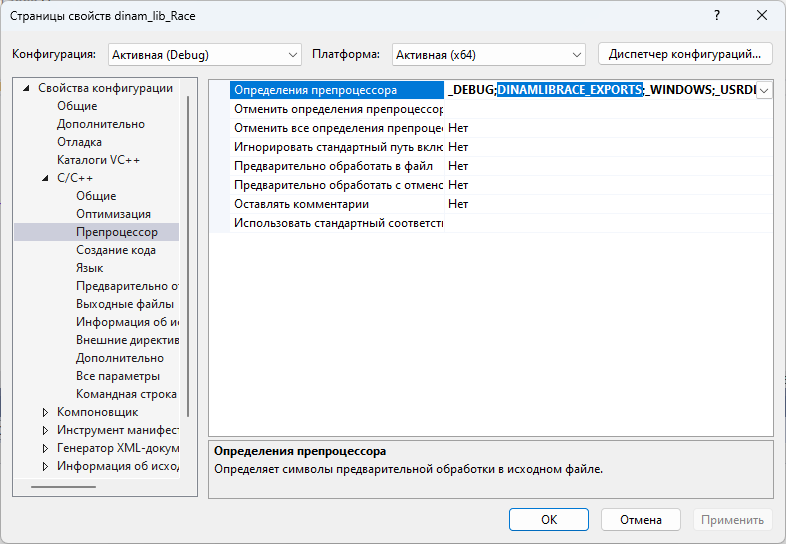
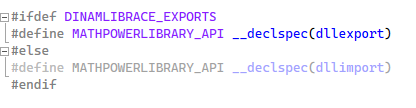
**Антон Ласточкин**

Ответ эксперта

Евгений, признаюсь честно, не очень удобно отвечать в рубрике вопросы и ответы, я бы рекомендовал через форму при отправке курсовой на проверку, там более удобный интерфейс. Тем не менее, отвечаю. Рад, что помог разобраться, хотя бы частично. По автоматической отправке. К сожалению, С++ достаточно низкоуровненвый язык, поэтому само понятие “автоматически” не особо применимо. Когда вы ставите MATHPOWERLIBRARY\_API, то при сборке библиотеки макрос превращается в \_\_declspec(dllexport), который помечает, какие функции доступны компоновщику. При сборке основного модуля макрос превращается в \_\_declspec(dllimport), который подсказывает, что определения надо искать снаружи. Это упрощённо. Рекомендую статью для углубления в предмет: [habr.com...es/150327/](https://habr.com/ru/articles/150327/)  
Макрос для импорта и экспорта просто должен совпадать в исходнике (хедере):

12**#ifdef DINAMLIBRACE\_EXPORTS**

**#define MATHPOWERLIBRARY\_API ...**

И в настройках препроцессора для библиотеки. В обозревателе решений нажимаете правой кнопкой на dynamic\_lib\_Race, затем свойства и переходите к препроцессору:  
  
Если сделано всё правильно, то в редакторе кода подсветиться dllexport:  
  
Есть множество способов подключения библиотеки, я к сожалению, подзабыл как было в презентации. Можете скинуть название презентации и номер слайда? dinam\_lib\_Race.obj - это как раз промежуточный объектный код от компилятора. Не путайте с wavefront obj форматом 3D :)

**ЕС**

**Евгений Супряга**

Ответ студента

1. огромное спасибо за столь подробное разъяснение- для меня это очень полезно.
2. как я понял: пустые экземпляры класса отправляются автоматически ( MATHPOWERLIBRARY\_API)?  
   например:  
   class ground {  
   public:  
   int speed;  
   int time\_to\_out;  
   }; // экземпляр создастся автоматически и автоматически на отпраку.
3. подскажите где можно увидеть название проекта для его указания - DINAMLIBRACE\_EXPORTS. я его указываю наугад, по правилам без пробелов и заглавными.
4. про подцепление библиотеки, я её подцеплял, по презентации - Откройте “Свойства” проекта. Перейдите в пункт меню “C/C++” -> “Общие”. Откройте окно редактирования пункта "Дополнительные каталоги включаемых файлов”.

там у меня было указано “…\dinam\_lib\_Race”.

я попробовал ваш метод дополнительно применить - линковщик начинает ругаться что не находит файл с расширанием …  
LNK1104 не удается открыть файл “…\dinam\_lib\_Race.obj”, и зачем ему 3д файл?!

убрал строку, объявил …\dinam\_lib\_Race по пути презентации - всё заработало.