

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
**Федеральное государственное автономное образовательное
учреждение высшего образования
«Севастопольский государственный университет»**

Институт радиоэлектроники и интеллектуальных технических систем
Кафедра «Информатика и управление в технических системах»



МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторной работы
**«Исследование модульного подхода
к созданию программ»**

по дисциплине
«Объектно-ориентированный анализ и проектирование»

*для студентов очной формы обучения направления 27.03.04
«Управление в технических системах» (профиль подготовки
«Интеллектуальные робототехнические системы»)*

Версия 30082024

Севастополь — 2024

УДК 004.6

Методические указания к выполнению лабораторной работы «Исследование модульного подхода к созданию программ» по дисциплине «Объектно-ориентированный анализ и проектирование» для студентов очной формы обучения направления 27.03.04 «Управление в технических системах» (профиль подготовки «Интеллектуальные робототехнические системы») / Сост. Альчаков В.В. — Севастополь: Изд-во ФГАОУ ВО «Севастопольский государственный университет», 2024. — с. 46.

Методические указания:

рассмотрены и рекомендованы к изданию решением кафедры «Информатика и управление в технических системах», протокол № 1 от 30.08.2024 г.;

допущены учебно-методическим центром ФГАОУ ВО «Севастопольский государственный университет» в качестве методических указаний.

Рецензент:

Крамарь В.А., д-р техн. наук., профессор, профессор кафедры «Информатика и управление в технических системах» СевГУ.

© СевГУ, 2024

© Альчаков В.В., 2024

Содержание

Цель работы.....	4
Порядок выполнения и задание на работу.....	4
Требования к отчёту по лабораторной работе.....	4
Основные теоретические сведения.....	6
Создание программного решения.....	6
Структура программы.....	23
Потоковый ввод-вывод.....	27
Файловый ввод-вывод.....	29
Использование пространства имен.....	32
Варианты задания.....	33
Пример выполнения работы.....	37
Контрольные вопросы.....	43
Список использованных источников.....	43
Справочная информация.....	44
Полезные ссылки.....	45

Цель работы

Актуализировать знания о разработке программ на языке C++. Изучить модульный подход к разработке программ. Работа в пакетах Qt Creator и Visual Studio.

Порядок выполнения и задание на работу

1. Актуализировать знания по основам программирования на языке C++.
2. Изучить модульный подход к разработке программ.
3. Изучить потоковый ввод-вывод в файлы.
4. Изучить возможности IDE Microsoft Visual Studio и Qt Creator для разработки и отладки программ.
5. Создать комплексное решение, объединяющее несколько проектов.
6. Написать программу в соответствии с заданием части I своего варианта задания. Допускается задание входных данных непосредственно в теле программы.
7. Написать программу в соответствии с заданием части II своего варианта задания. Входные данные должны быть заданы через консоль. Выходные данные должны быть записаны в файл.
8. Написать программу в соответствии с заданием части III своего варианта задания. Описание функции для вычисления заданного выражения должно быть сделано в отдельном модуле. Структура модуля должна содержать заголовочный файл .h и файл реализации .cpp. Ввод исходных данных и вывод результатов осуществить с помощью файлов. Предусмотреть форматированный вывод вещественных чисел (аргумент функции два знака после запятой, значение функции четыре знака после запятой).
9. Подготовить индивидуальный отчёт о проделанной работе.

Требования к отчёту по лабораторной работе

Отчёт о выполненной лабораторной работе должен содержать:

- титульный лист;
- цель лабораторной работы;
- основные положения;
- ход работы;
- выводы по работе;
- приложения (листинги исходных кодов программ).

Отчёт составляется каждым обучающимся индивидуально и должен соответствовать варианту задания, назначенного преподавателем.

В отчёте к данной лабораторной работе необходимо привести краткие теоретические сведения о использованных синтаксических конструкциях языка C++, использование потокового файлового ввода и вывода, правила использования модульного подхода к разработке программ. В конце отчета привести краткие выводы по работе.

Основные теоретические сведения

Создание программного решения

Разработка современного программного обеспечения невозможна без использования специализированной IDE (интегрированной среды разработки англ. Integrated Development Environment). Наиболее распространенными в настоящее время IDE для разработке на языке C++ являются: Microsoft Visual Studio, Microsoft Visual Studio Code, Qt Creator.

Первые две IDE заточены под разработку программ для операционной системы Windows, IDE Qt Creator поддерживает кросс платформенную разработку и свою расширенную библиотеку классов Qt. Выполнять задания в рамках курса можно используя любую из указанных сред разработки.

Объемы кода реальных приложений занимают тысячи, десятки и даже сотни тысяч строк кода. Сам по себе процесс разработки программных продуктов в настоящее время характеризуется как работа со сложной системой, в которой в качестве объекта выступает программный код. Для упрощения работы со сложными системами принято использовать принцип декомпозиции — процесс разбиения одной сложной задачи на несколько более простых задач. Тот же принцип следует применять и к разработке программного обеспечения. Этот принцип называется модульным подходом. При модульном подходе исходный код программы делится на отдельные модули, хранящиеся в отдельных файлах. Модули могут быть как полностью независимыми, так и использующими кодовую базу других модулей (такие модули называются зависимыми). Использование модульного подхода дает преимущество с точки зрения удобства разработки: небольшие по объему модули проще отлаживать и сопровождать, разработку модулей можно распределить между членами команды, созданные модули можно повторно использовать в других приложениях.

Исходя из важности модульного подхода, в рамках выполнения лабораторной работы, необходимо освоить модульный подход к разработке программ. Для этого, необходимо изучить возможности IDE предназначенные для создания модулей и ведения нескольких проектов в рамках одного программного решения.

В рамках данной лабораторной работы необходимо реализовать три отдельных консольных приложения и объединить их в одно программное решение.

Рассмотрим процесс создания программного решения в IDE Visual Studio Community Edition 2022.

Шаг 1. Запускаем IDE Visual Studio Community Edition 2022.

Шаг 2. В появившемся окне мастера выбираем последнюю опцию **Create a new project**.

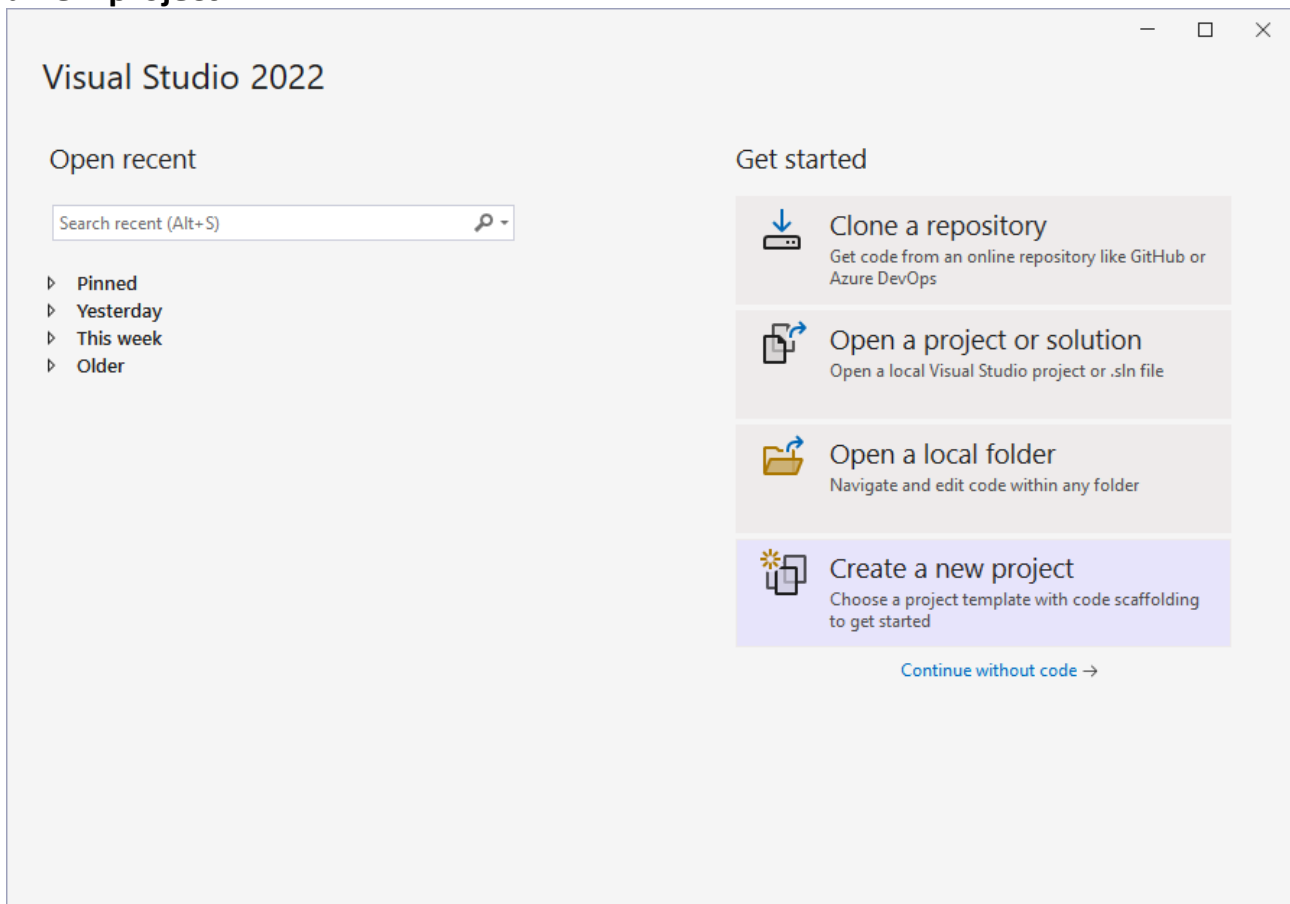


Рис. 1 — Окно мастера Visual Studio

Шаг 3. На следующем шаге выбираем **Blank Solution** (предварительно выставив в выпадающих списках опции All languages, All platforms, Other, как это показано на рисунке).

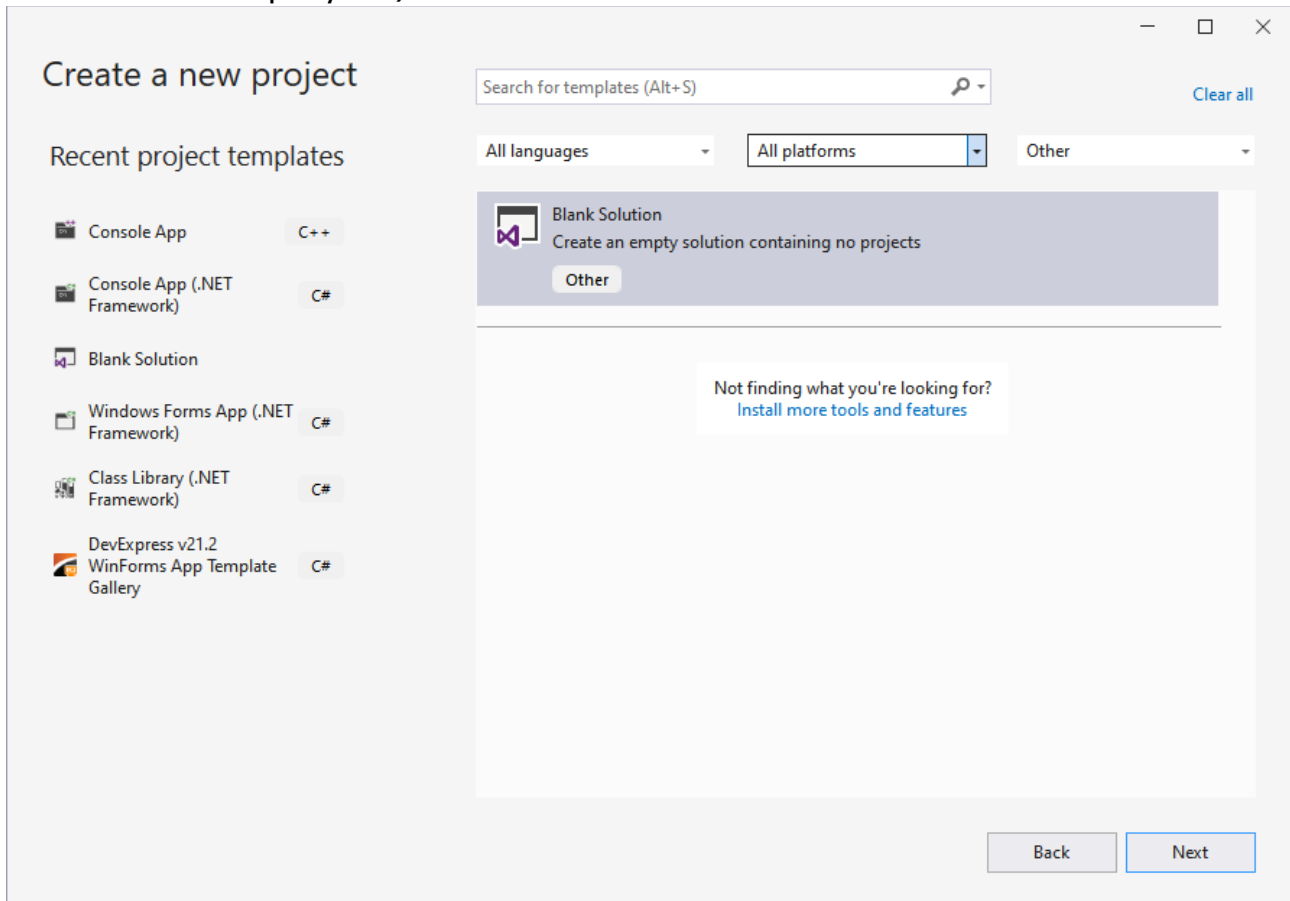
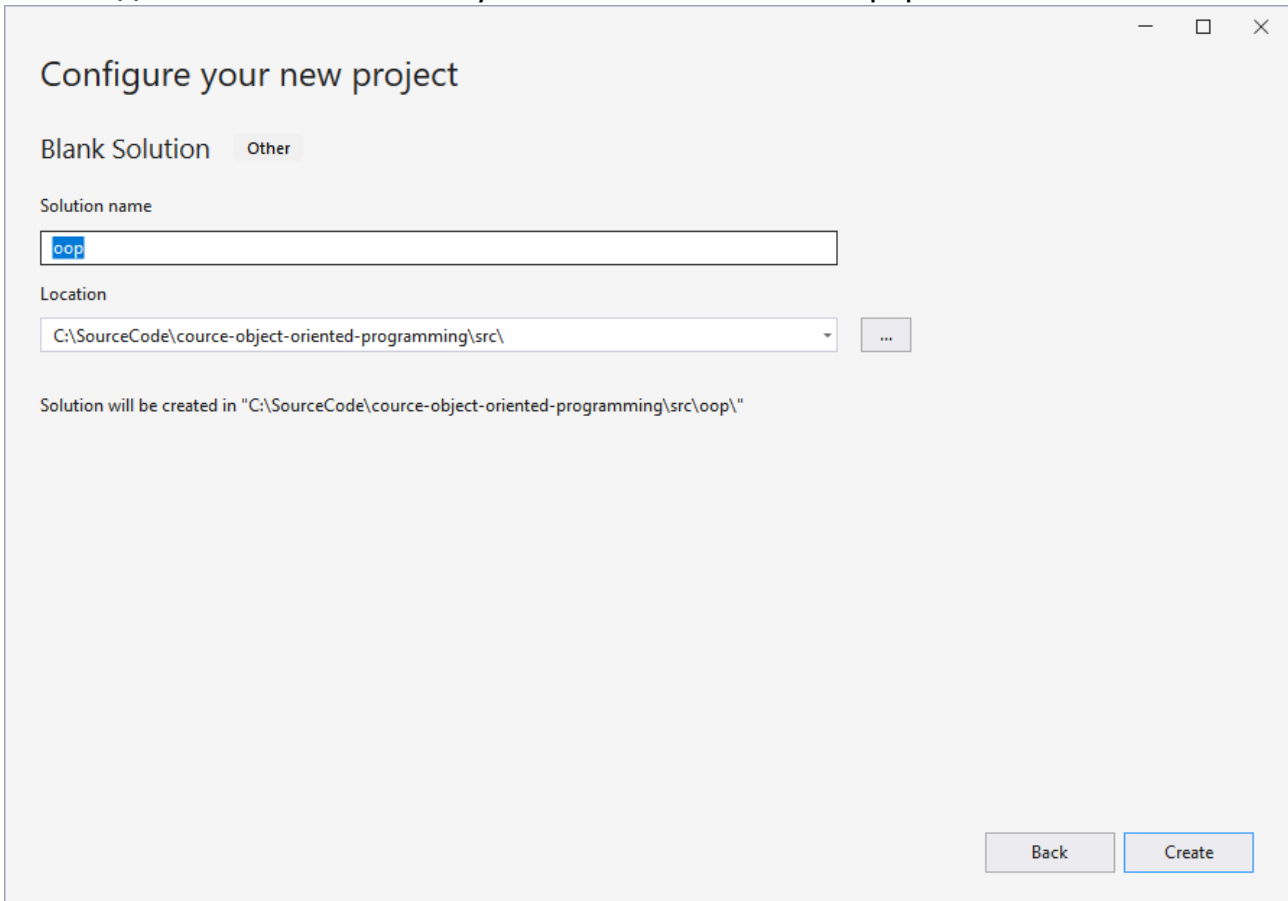


Рис. 2 — Создание пустого решения

Шаг 4. На последнем шаге мастера остается указать имя программного решения (может быть произвольным, в данном случае используется *Solution name*: оор) и путь, по которому будут храниться исходные коды данного решения - *Location*. После ввода данной информации в форму мастера необходимо нажать на кнопку *Create* в нижней части формы.



Configure your new project

Blank Solution Other

Solution name

оор

Location

C:\SourceCode\course-object-oriented-programming\src\

Solution will be created in "C:\SourceCode\course-object-oriented-programming\src\оор\"

Back Create

Рис. 3 — Задание имени решения и пути к исходным кодам

В результате работы мастера создания решения будет открыт редактор Visual Studio как это показано на Рис. 4. В окне *Solution Explorer* (обозреватель решений) можно увидеть только что созданное решение, которое пока не содержит ни одного проекта.

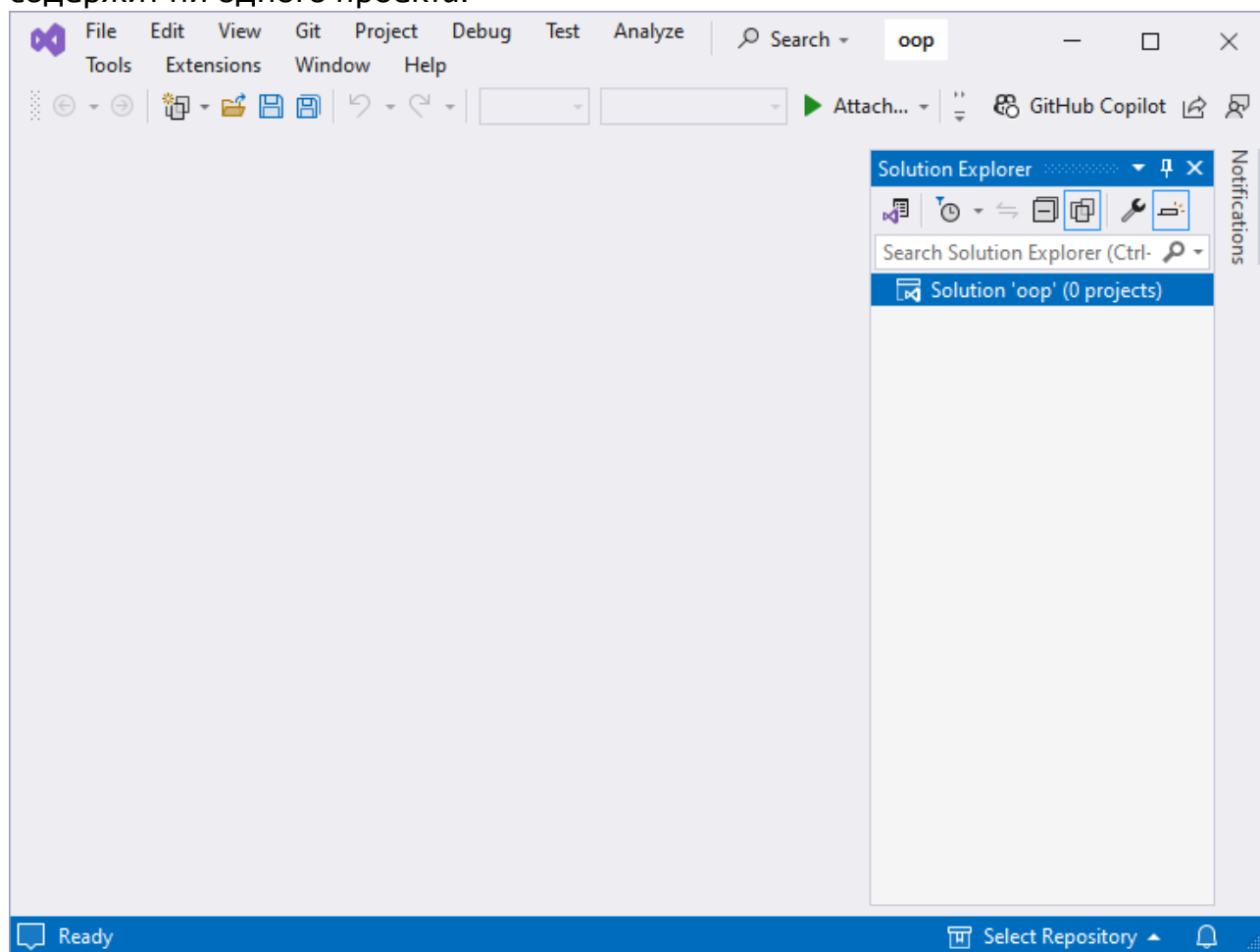


Рис. 4 — Окно IDE Visual Studio с пустым решением

Несмотря на то, что созданное решение пустое, в папке решения будут созданы служебные файлы в виде

c:\SourceCode\course-object-oriented-programming\src\oop*.*				
Name	Ext	Size	Date	Attr
[..]		<DIR>	08/14/2024 17:43	----
[.vs]		<DIR>	08/14/2024 17:43	--h-
oop	sln	421	08/14/2024 17:43	-a--

Рис. 5 — Структура каталога с файлами решения

Далее, в пустое решение нужно добавить проекты. В нашем случае мы добавим три пустых проекта консольных приложений: lr1-1, lr1-2 и lr1-3.

Впоследствии в каждом из этих проектов будет реализовано соответствующее задание.

Для добавление нового проекта в существующее программное решение необходимо переместить указатель мыши в окно Solution Explorer и вызвать контекстное меню над элементом *Solution 'oop'*

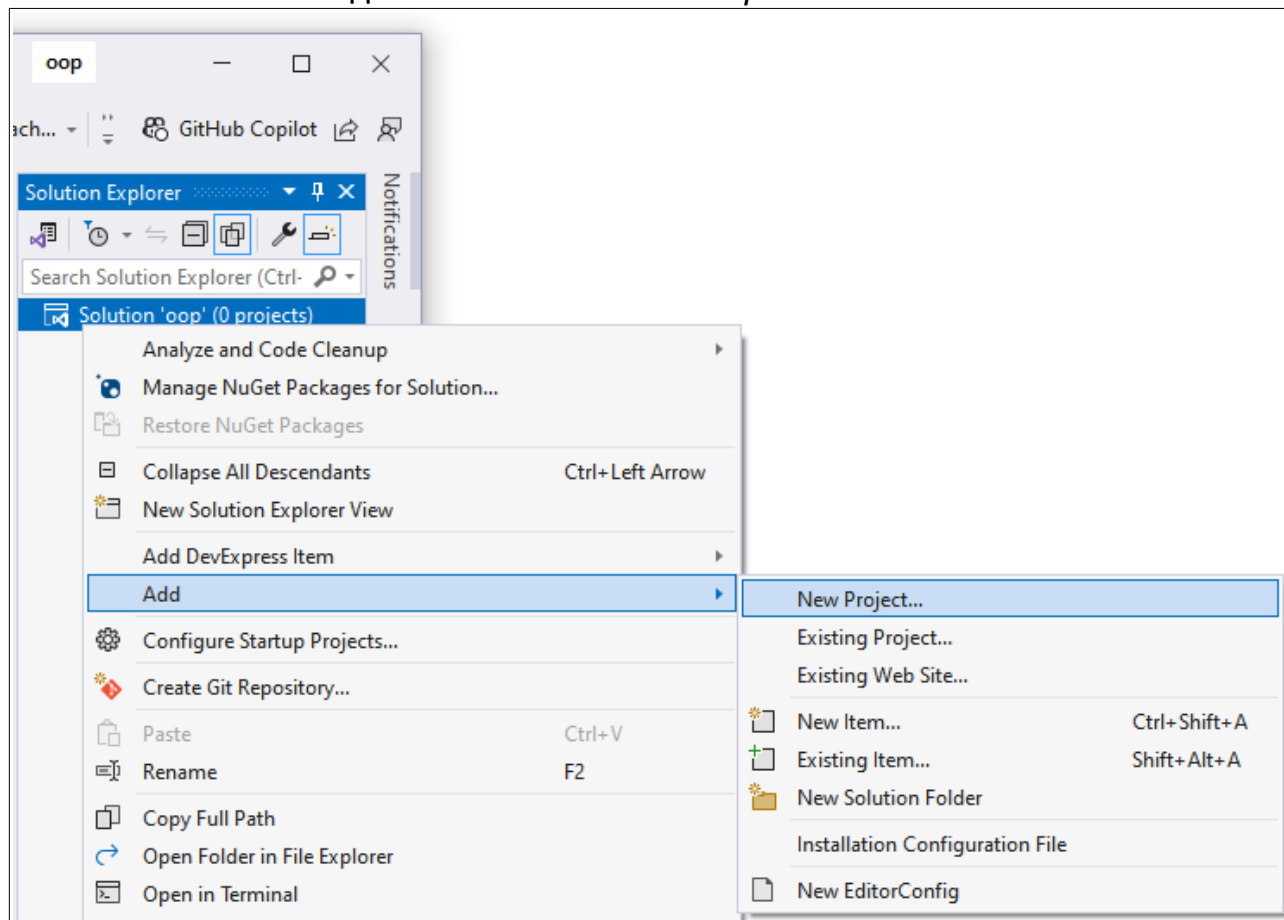


Рис. 6 — Добавление нового проекта в существующее решение

В появившемся контекстном меню нужно выбрать пункт **Add⇒New Project...** после чего запустится мастер добавления нового проекта. В окне мастера необходимо установить фильтры, определяющие тип проекта, который вы собираетесь добавить. Необходимо указать язык *C++*, платформу *Windows* и тип приложения *Console*.

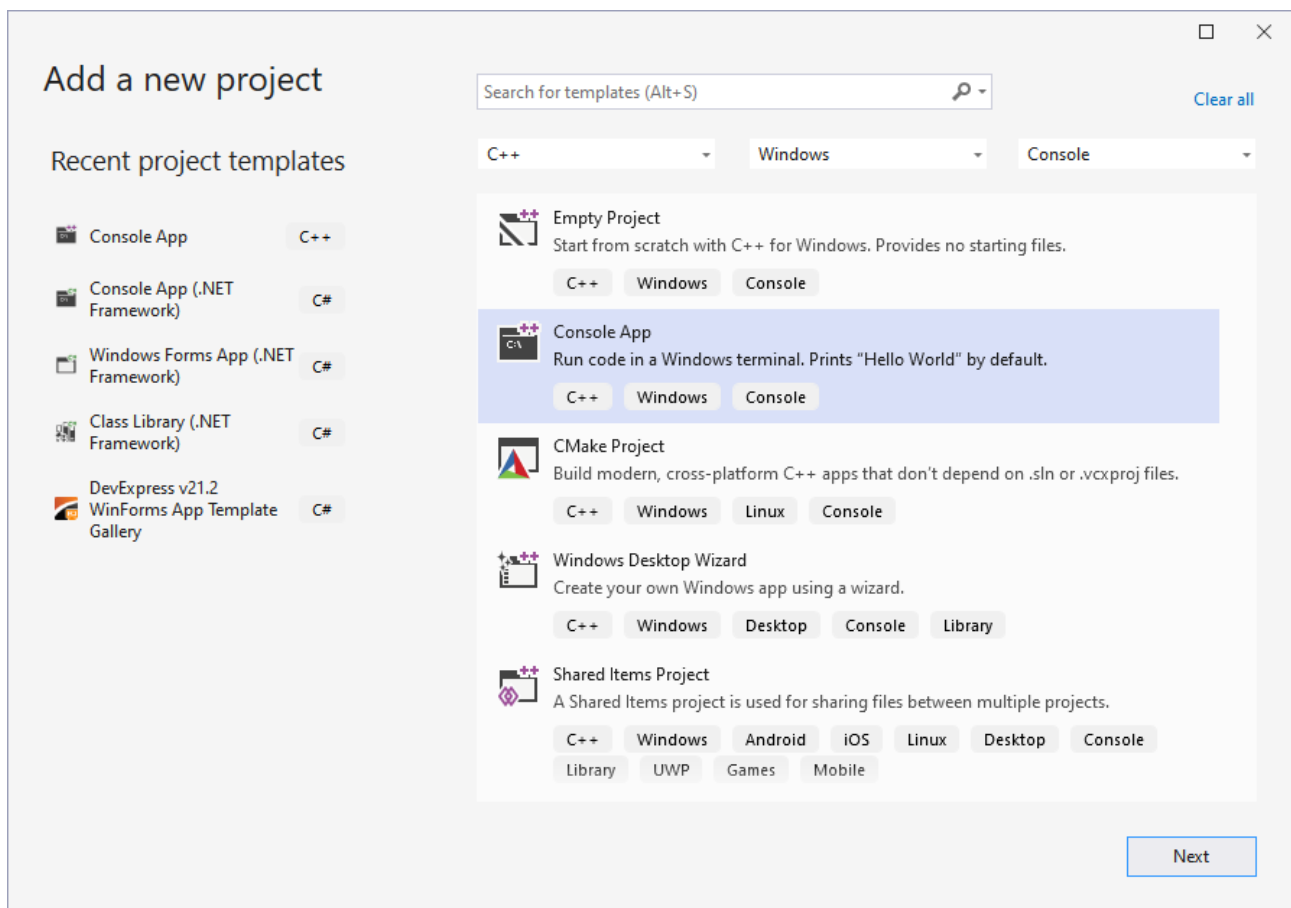


Рис. 7 — Добавление консольного проекта для Windows на языке C++

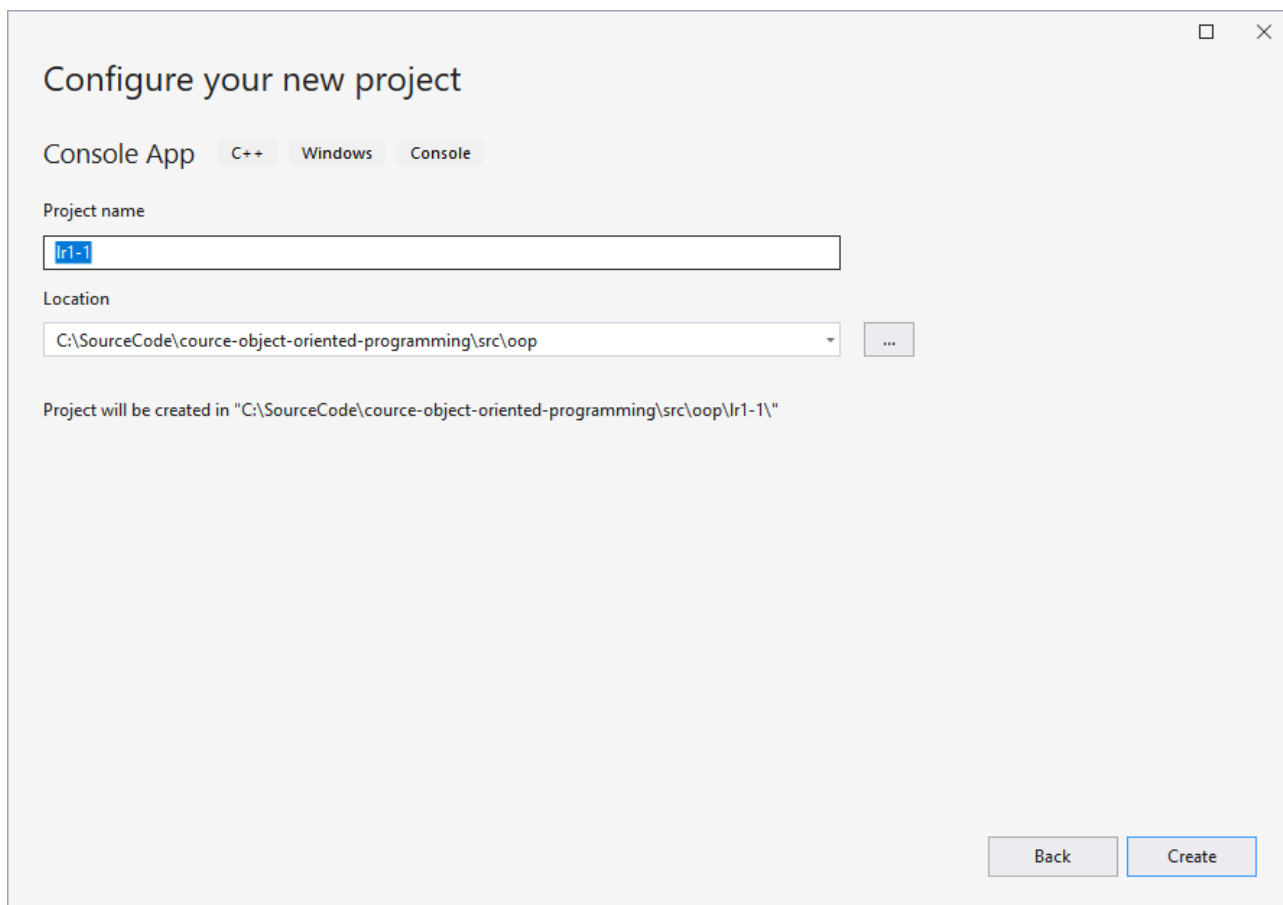


Рис. 8 — Задание имени проекта

В окне мастера заполняется поле Project name (lr1-1) после чего необходимо нажать кнопку **Create**.

После указанных операций в программное решение будет добавлен шаблон консольного приложения как это показано на Рис. 9. Закомментированный код можно удалить. Аналогично в программное решение необходимо добавить еще два консольных проекта lr1-2 и lr1-3, после чего Solution Explorer примет вид как на Рис. 10.

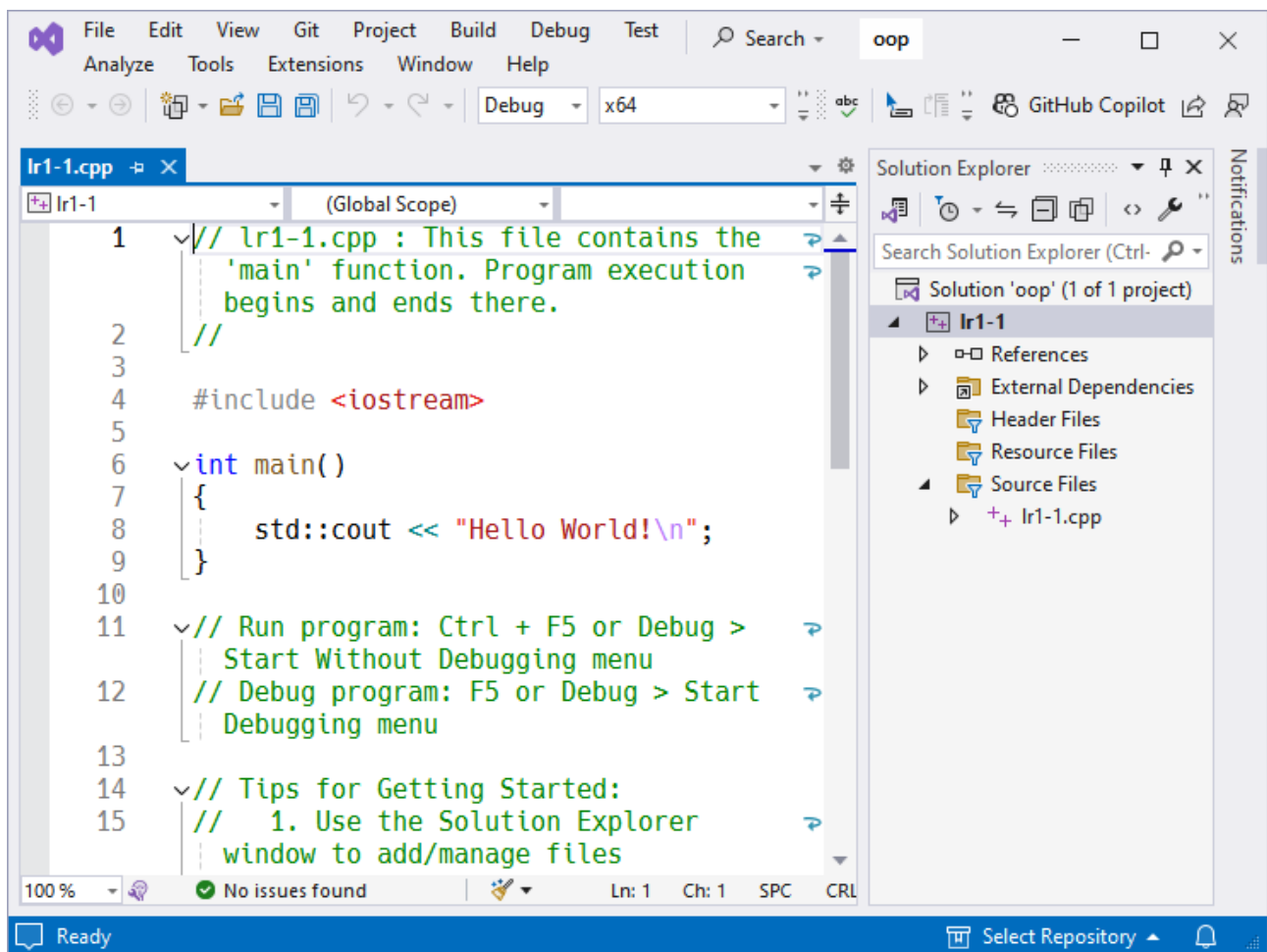


Рис. 9 — Окно IDE после добавления нового проекта в решение

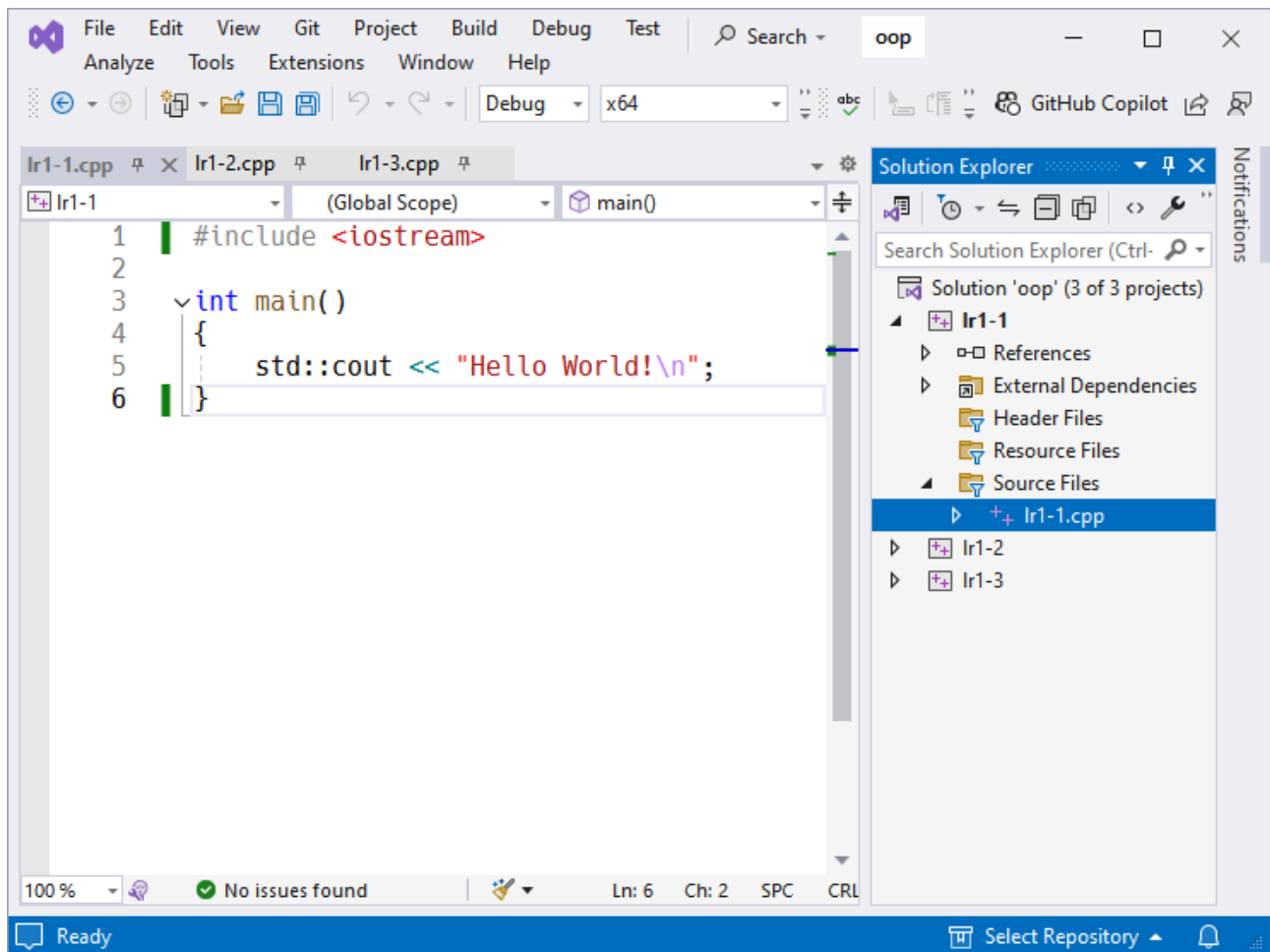


Рис. 10 — Окно IDE после добавления трех проекта в решение

Один из проектов в решении является активным. Имя этого проекта в обозревателе решений выделено жирным шрифтом, в нашем случае это проект **lr1-1**. Именно этот проект будет запущен, если выполнить команду *F5*. Проект по умолчанию может быть изменен, для этого необходимо вызвать контекстное меню над именем проекта и выбрать опцию *Set as Startup Project*.

Процесс создания программного решения с несколькими проектами в среде Qt Creator во многом похож на создание решения в Visual Studio Community Edition 2022.

Шаг 1. Запускаем Qt Creator.

Шаг 2. В открывшемся окне переходим в раздел **Welcome** и нажимаем кнопку **Create Project...** либо выбираем пункт меню **File** ⇒ **New Project...**

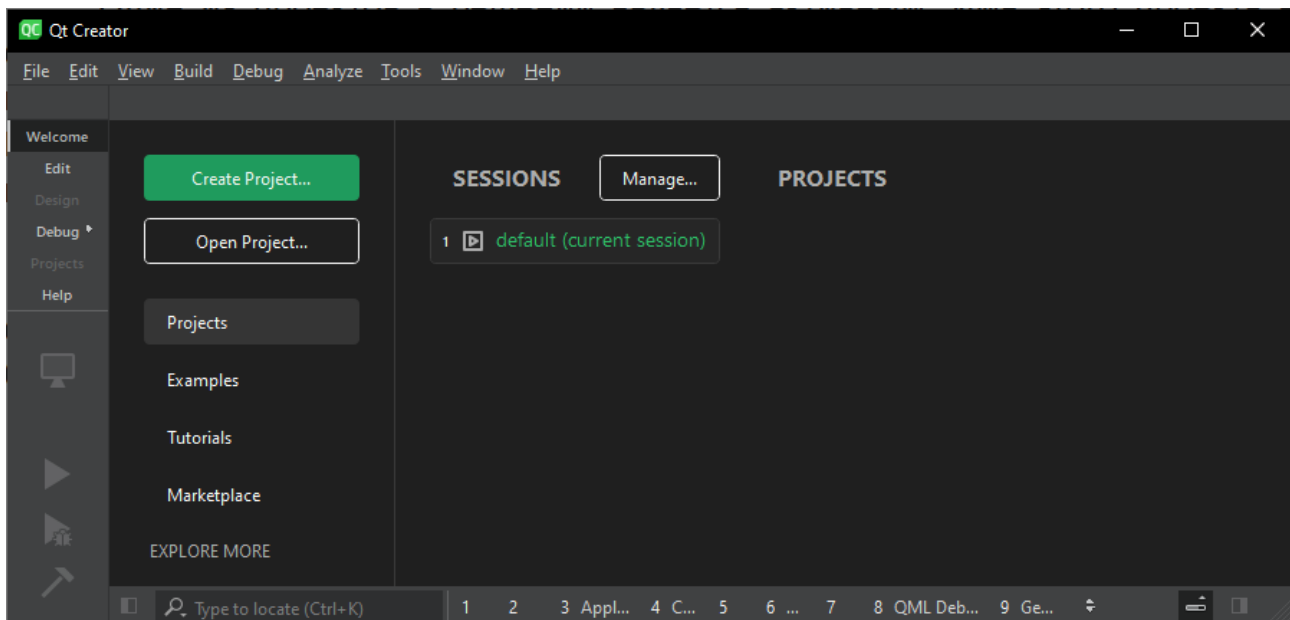


Рис. 11 — Создание программного решения в Qt Creator

Шаг 3. Заполняем формы мастера создания нового проекта. В открывшемся окне выбираем тип шаблона *Other Project*, *Subdirs Project*

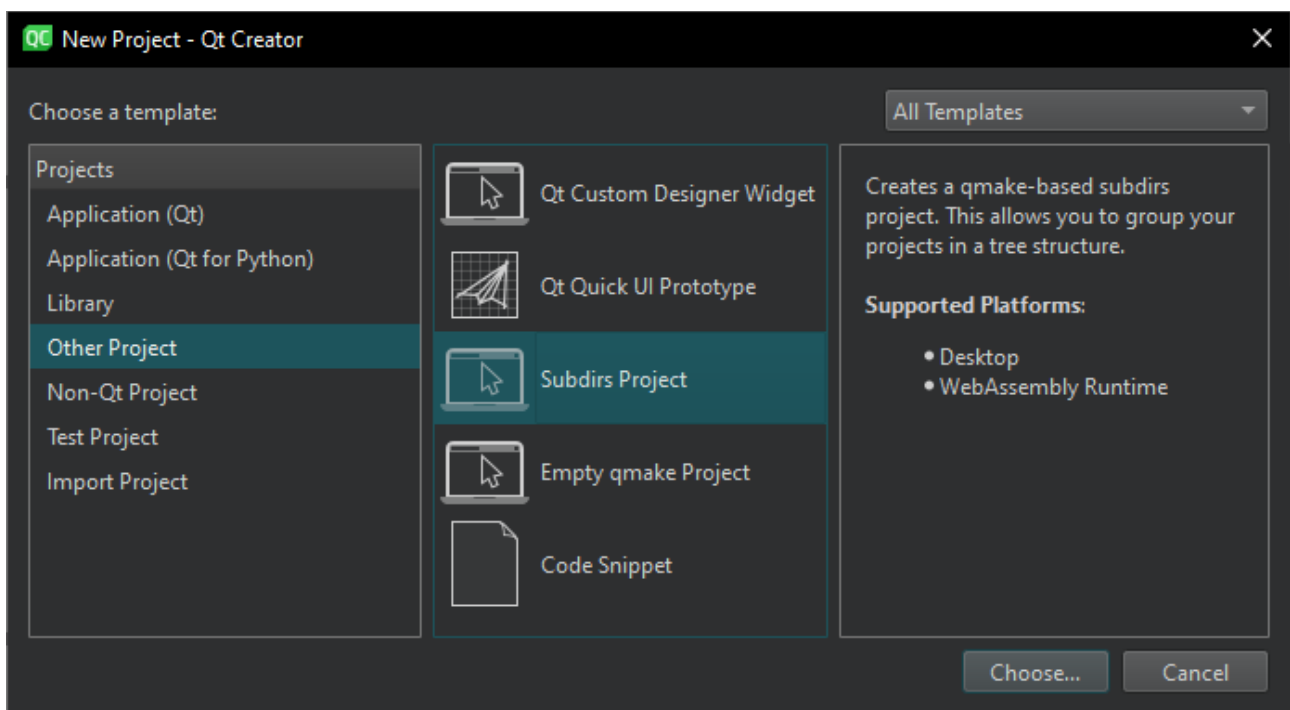


Рис. 12 — Выбор шаблона проекта

В следующем окне заполняем поле имя проекта *Name: oop* и указываем путь к исходным файлам проекта *Create in:*

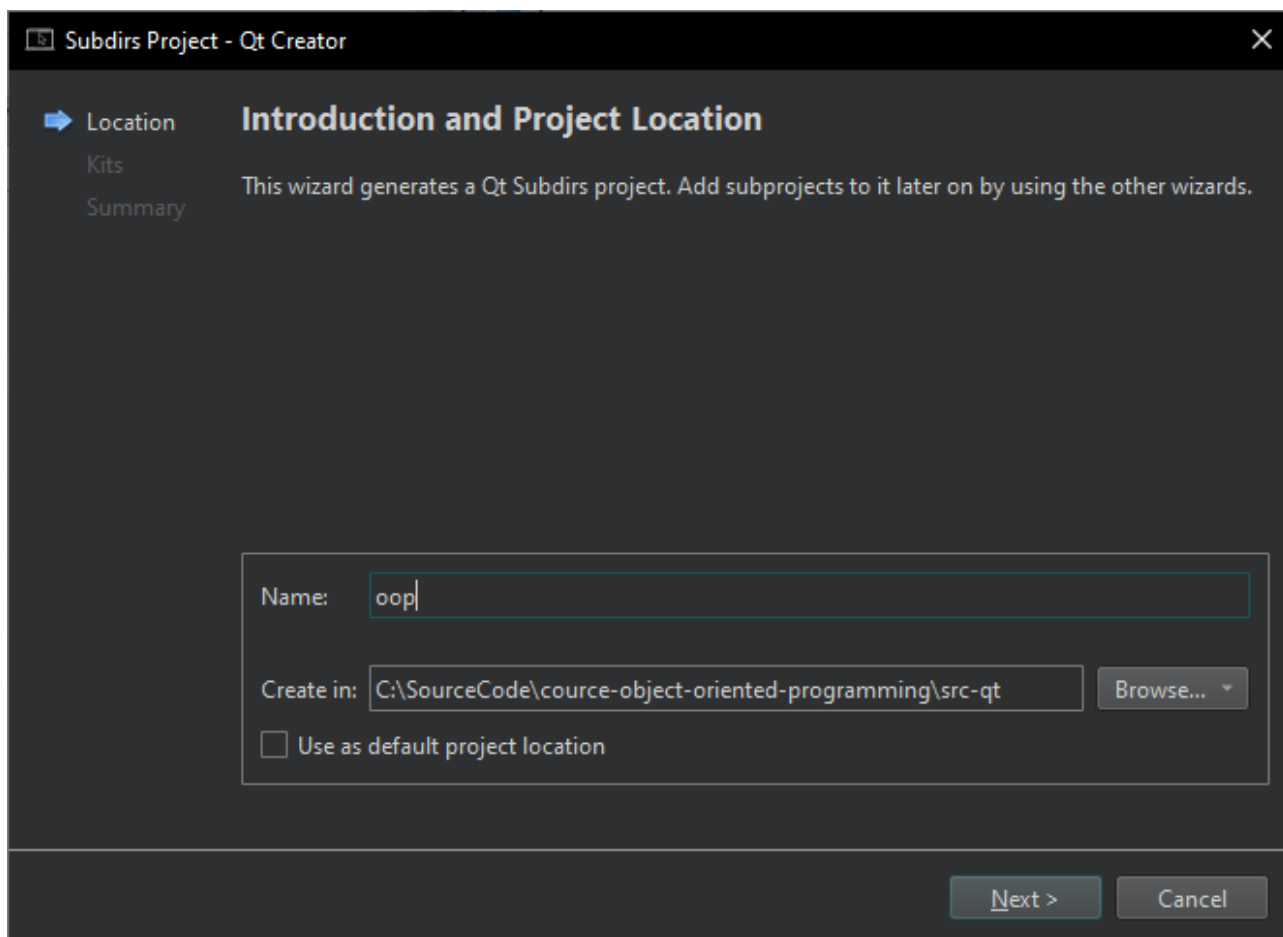


Рис. 13 — Задание имени проекта и расположения исходных кодов

Далее необходимо выбрать набор инструментов (компиляторов) которые будут использоваться для сборки проекта. Рекомендуется использовать Desktop Qt 6.7.0 MinGW 64-bit или Desktop Qt 6.7.0 MSVC2019 64-bit (в зависимости от установленного программного обеспечения и установленной версии Qt Creator названия опций могут отличаться).

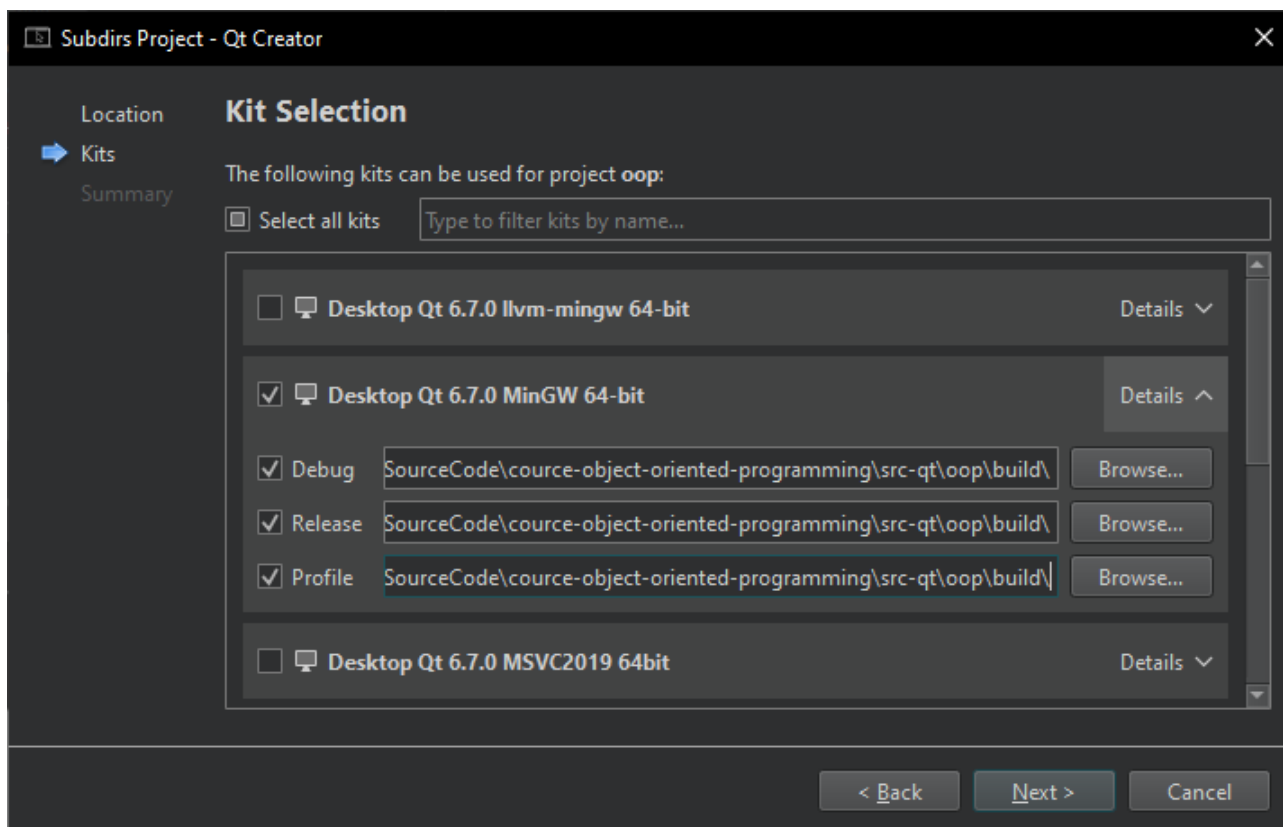


Рис. 14 — Выбор инструментов для сборки проекта

В следующем окне выставляем опцию *Add to version control: None*. Нажимаем кнопку *Finish & Add Subproject*.

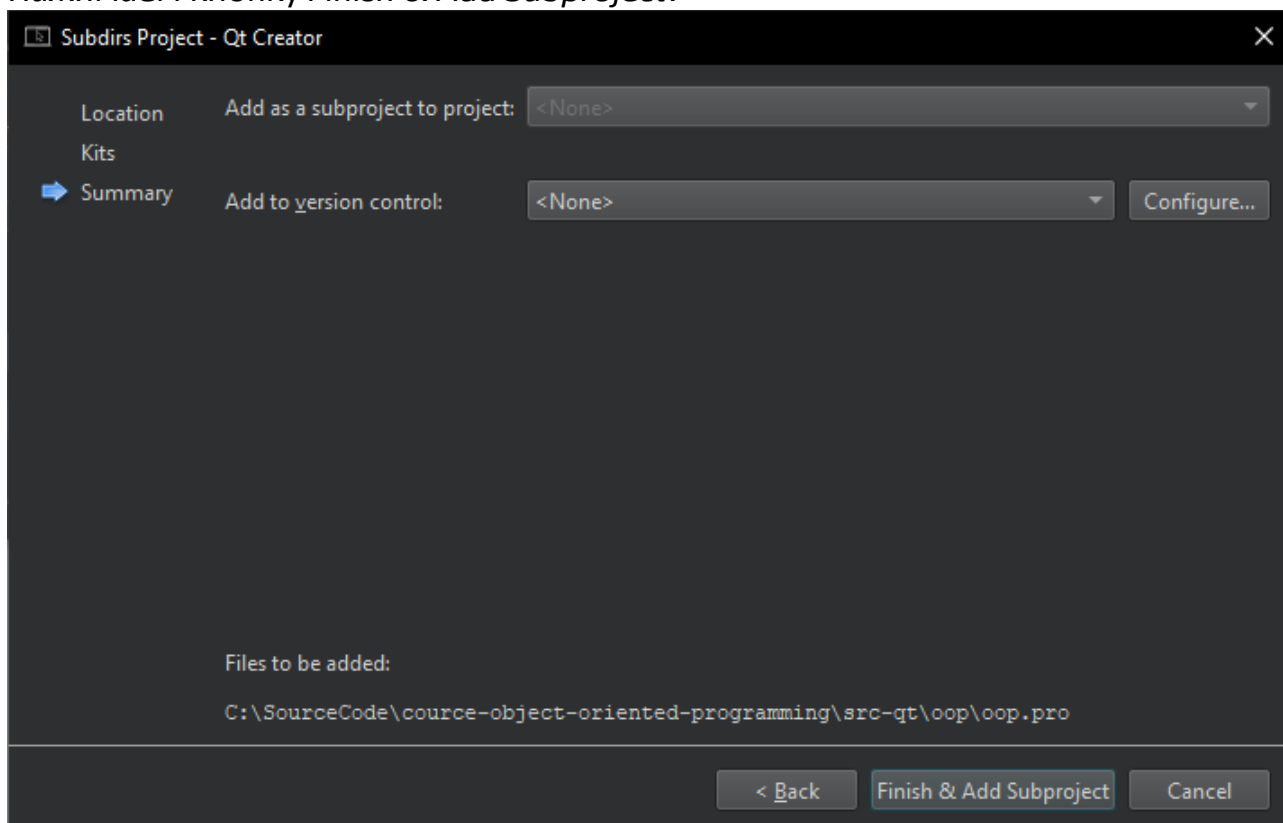


Рис. 15 — Окно мастера с настройками контроля версий

Шаг 4. Добавляем подпроекты в программное решение. Для этого вызываем контекстное меню для решения *oop* и выбираем опцию *New Subproject...*

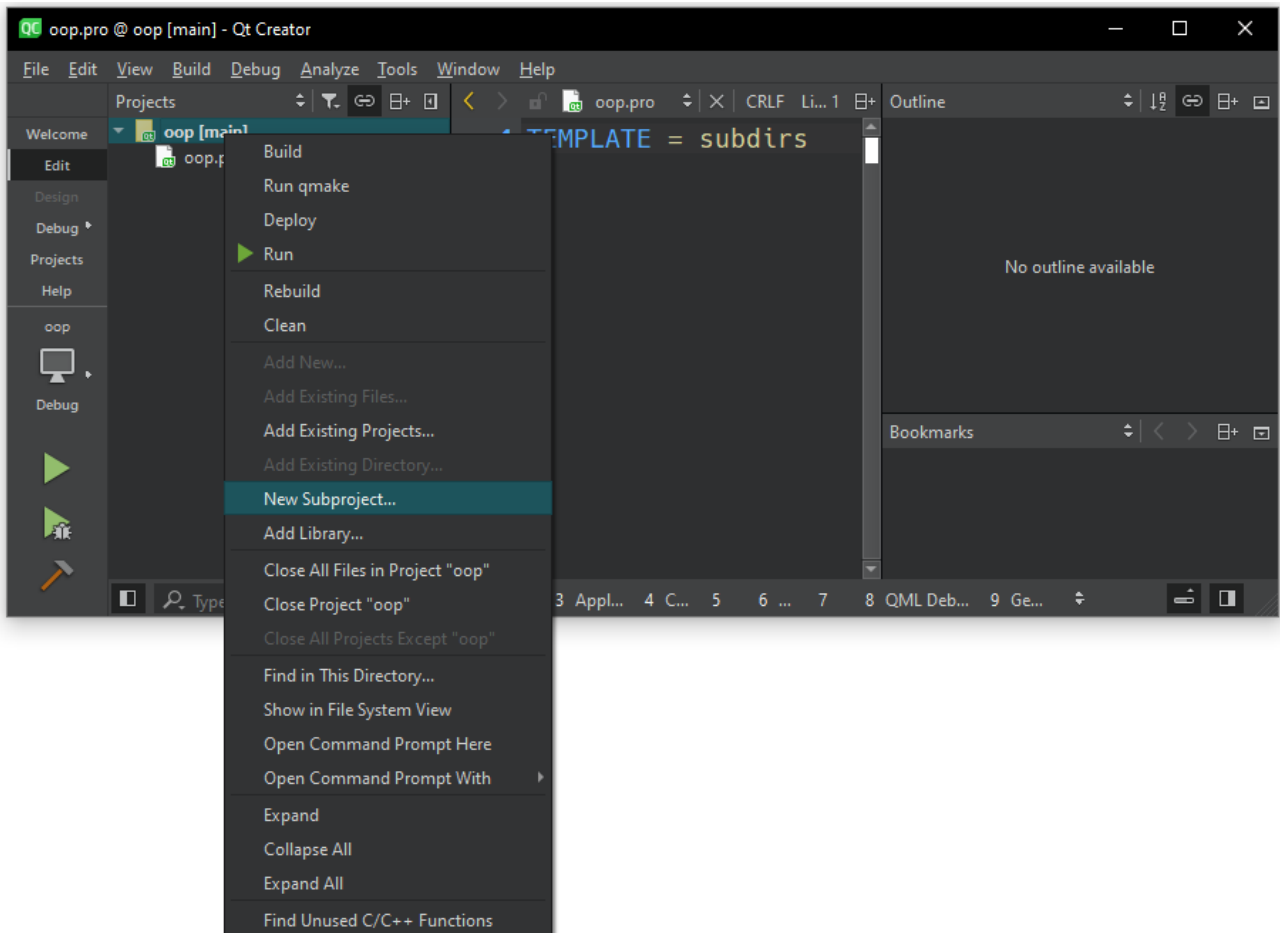


Рис. 16 — Добавление подпроекта в существующее решение

В следующем окне выбираем опцию *Non-Qt Project, Plain C++ Application*

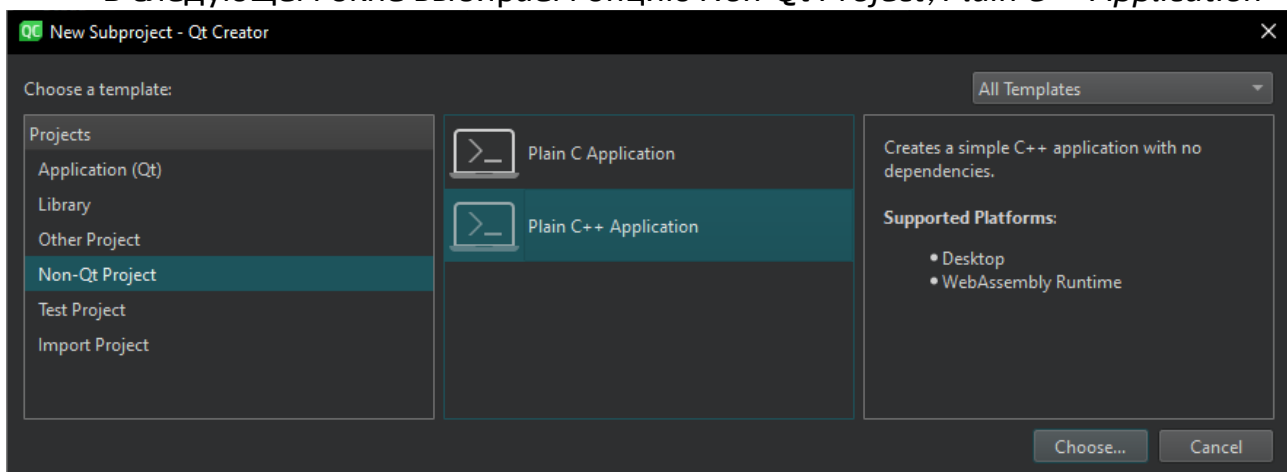


Рис. 17 — Задание типа подпроекта

Задаем имя подпроекта *Name: lr1-1*

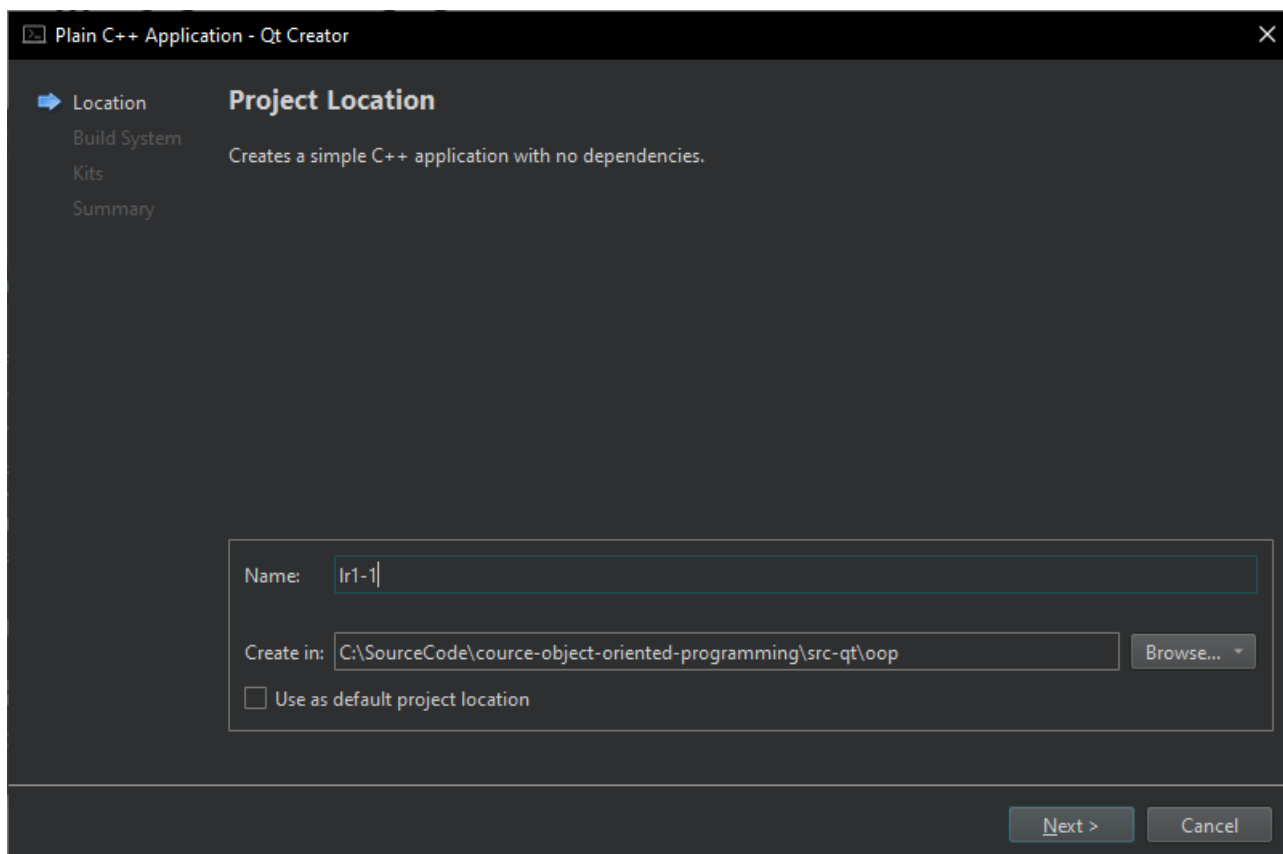


Рис. 18 — Задание имени подпроекта
Систему сборки оставляем без изменения.

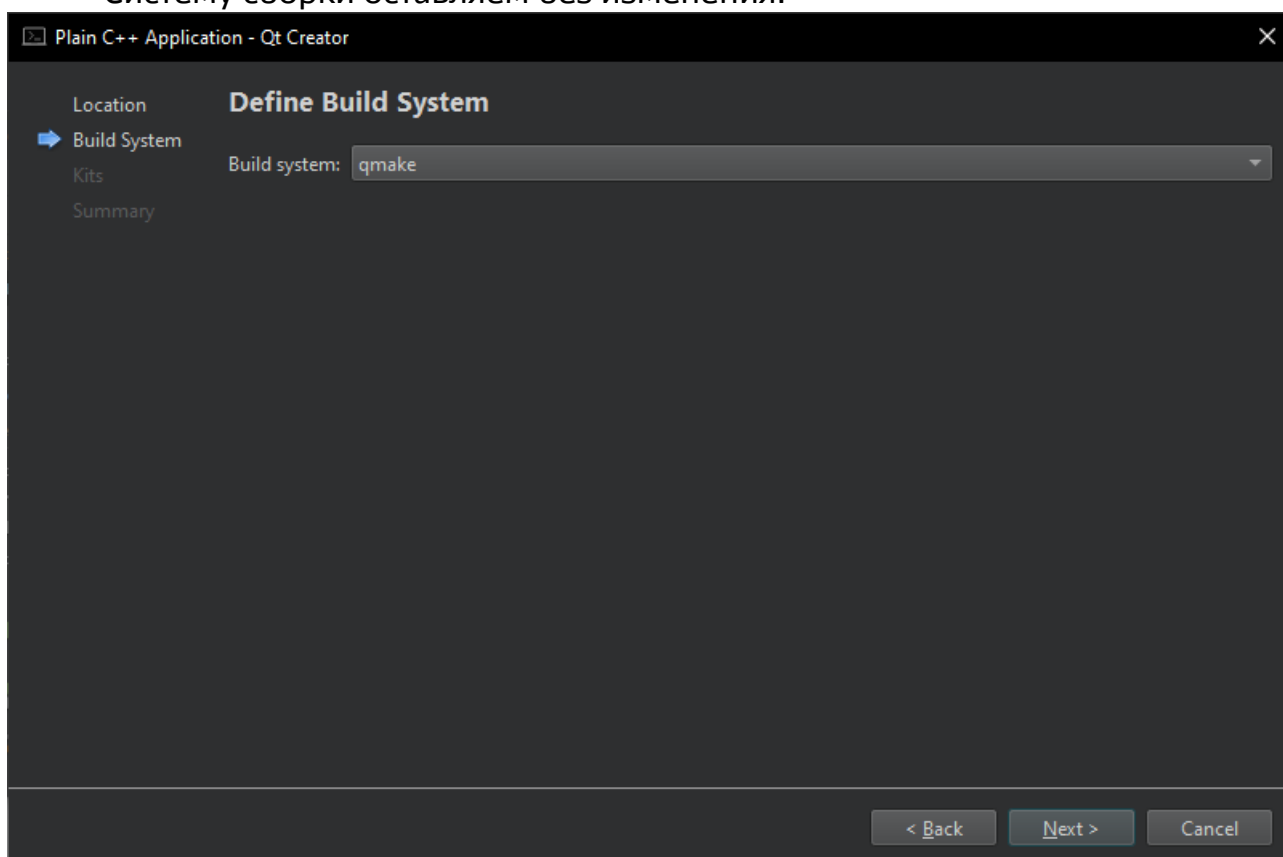


Рис. 19 — Задание системы сборки

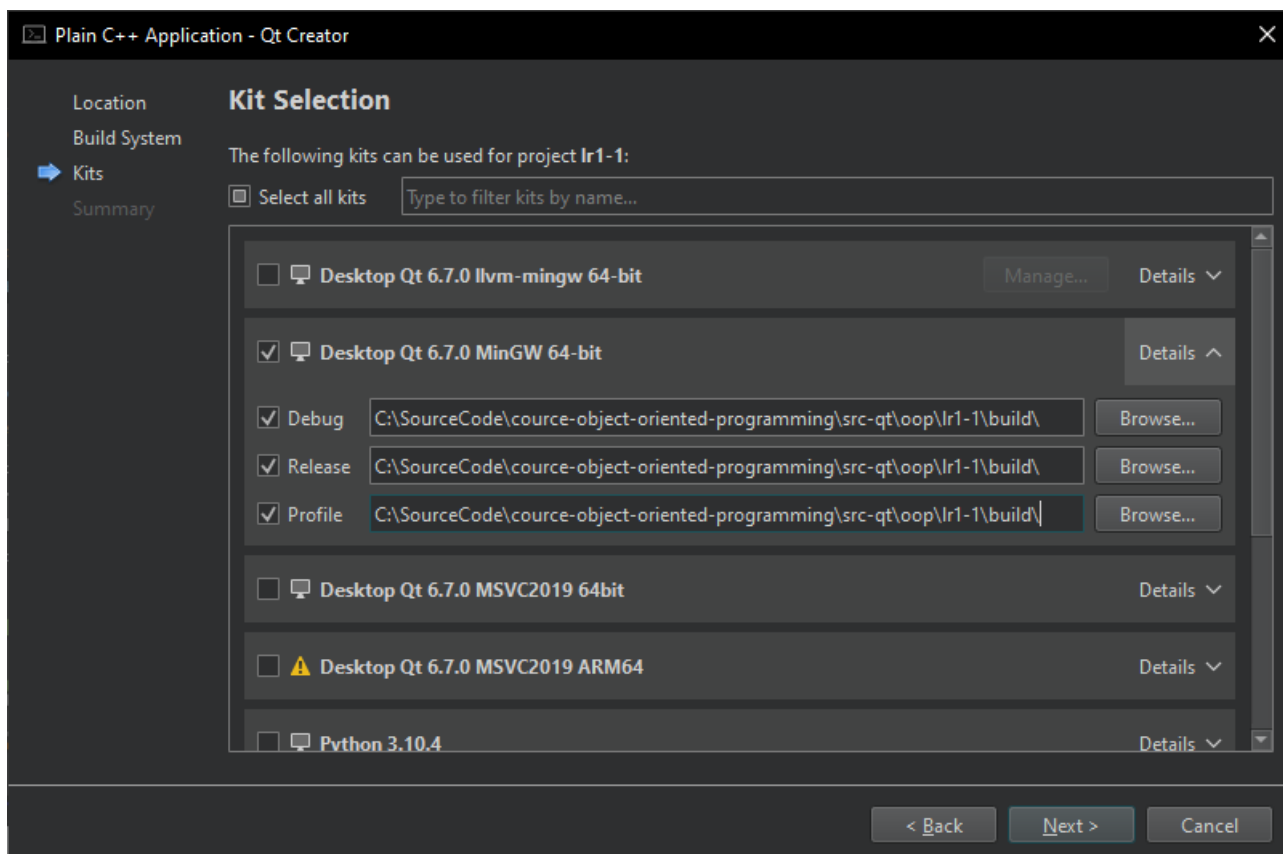


Рис. 20 — Задание системы сборки

Отключаем настройки системы контроля версий и нажимаем *Finish*.

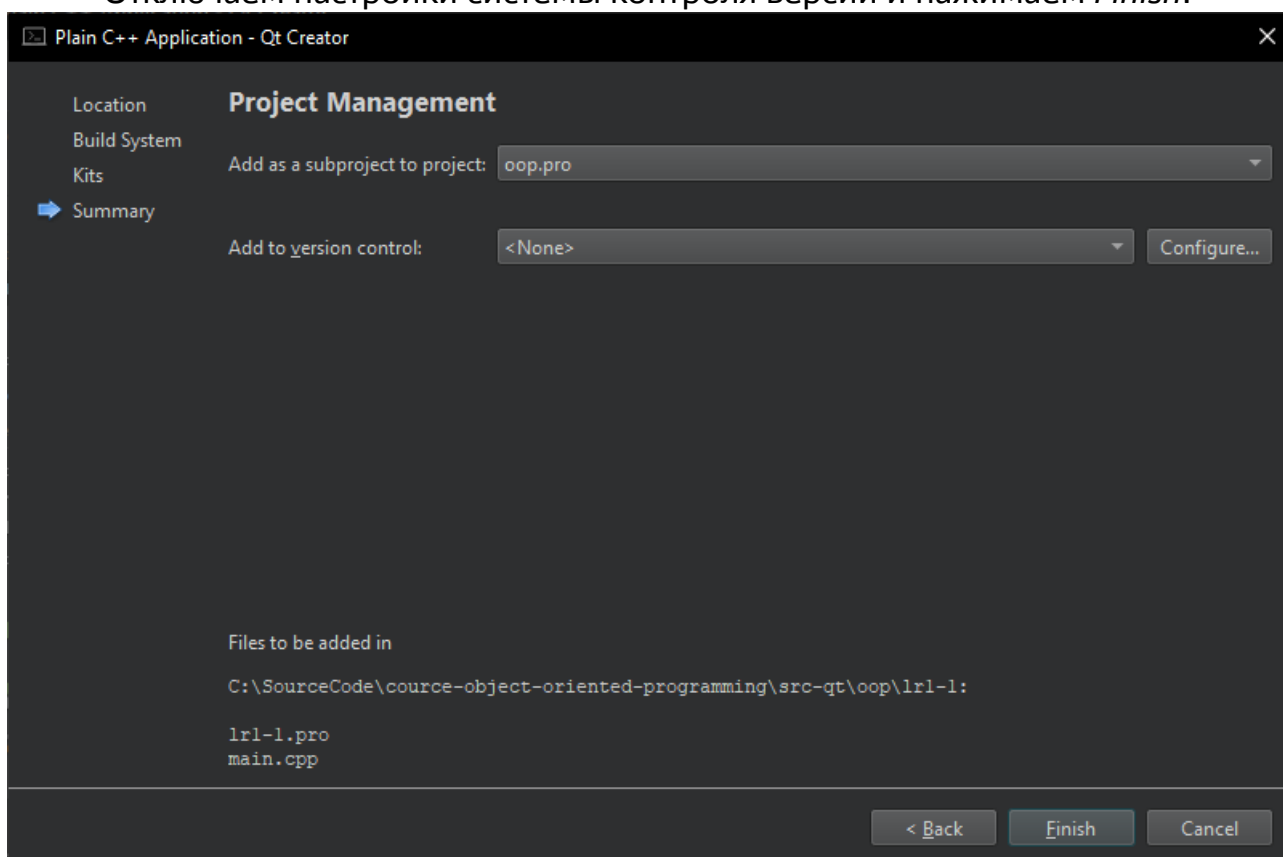


Рис. 21 — Настройка системы контроля версий

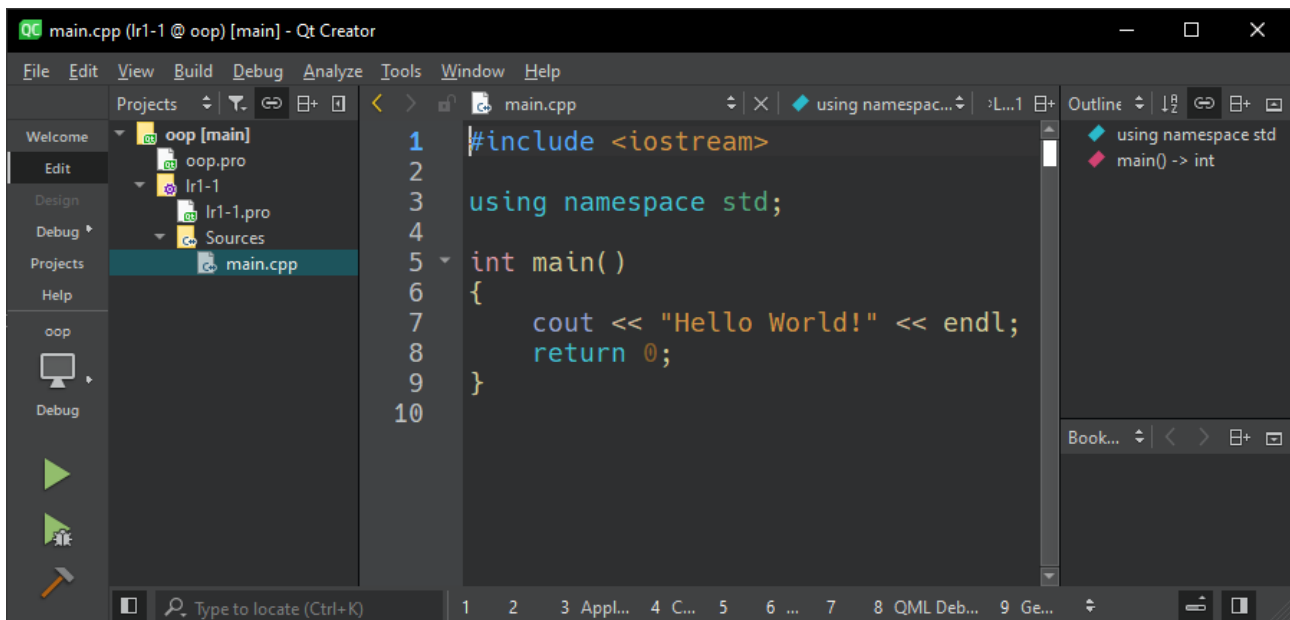


Рис. 21 — Главное окно IDE Qt Creator после добавления подпроекта

Аналогично в программное решение необходимо добавить еще два консольных проекта lr1-2 и lr1-3, после чего главное окно IDE Qt Creator примет вид как на Рис. 22.

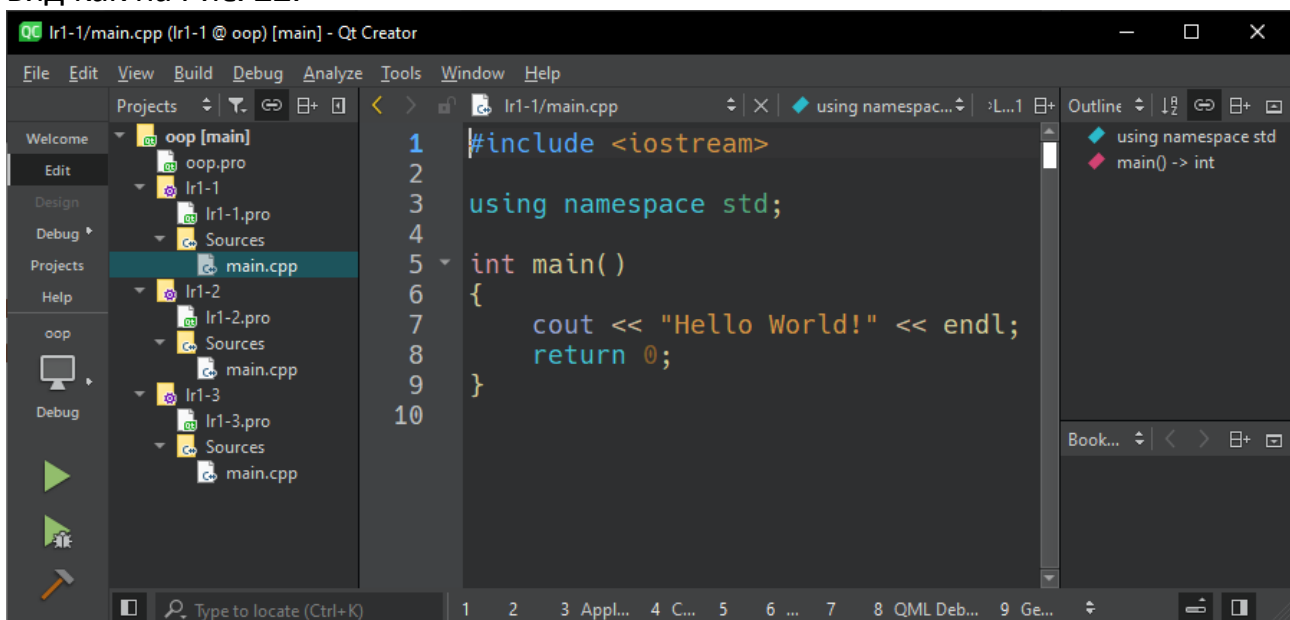


Рис. 22 — Главное окно IDE Qt Creator после добавления подпроектов lr1-1, lr1-2 и lr1-3

Структура программы

Любая реальная система состоит из разных компонентов, таких как исходные файлы, заголовочные файлы и библиотеки. Многие системы содержат ресурсы, включая изображения, звуки и конфигурационные файлы. Разделение программы на логические компоненты меньшего размера является рекомендуемым подходом к проектированию программного обеспечения, поскольку эти компоненты легче обслуживать по сравнению с одним большим файлом.

Принципы компонентного представления. Ничто не мешает вам написать всю программу целиком внутри функции *main* одного исходного файла. Но по мере увеличения размера данной функции за ней становится сложно уследить. В связи с этим программу имеет смысл разбить на компоненты, которые обмениваются информацией через общую границу (*интерфейс*). В исходном коде, состоящем из компонентов, легче разобраться и его можно использовать сразу в нескольких местах программы или даже в других проектах.

Обычно понимание того, как лучше всего разделить программу, требует опыта. Многие свои решения программисты принимают с оглядкой на производительность. Например, у вас может возникнуть необходимость минимизации взаимодействия по интерфейсу с высокой латентностью. Или вы можете возложить проверку полей ввода на клиентский код, чтобы данные не нужно было отправлять на сервер и обратно. Рассмотрим некоторые принципы проектирования программного обеспечения на основе компонентов (*модулей*).

Связность и зацепление. Помимо производительности, хорошо структурированная программа должна иметь и другие полезные свойства, такие как слабое зацепление и высокая связность. *Связность (cohesion)* — это показатель того, что у элементов программного интерфейса общая цель. Представьте, к примеру, что заголовочный файл предоставляет функции для определения длины строки, вычисления тангенса заданного входного значения и создания потока выполнения. Такой заголовочный файл имеет низкую связность, поскольку доступные в нем функции не имеют никакого отношения друг к другу. А вот заголовочный файл с функциями для определения длины, соединения двух строк в одну и поиска подстроки имеет высокую связность, поскольку вся его функциональность имеет одну и ту же направленность. Следовательно, если вам нужно работать со строками, то достаточно подключить один подходящий заголовочный файл. Точно так же взаимосвязанные определения функций и типов, которые составляют публичный интерфейс, должны предоставляться одним и тем же заголовочным

файлом, чтобы сделать этот интерфейс сильно связным и ограниченным по функциональности.

Зацепление (coupling) определяет, насколько созависимы программные интерфейсы. Например, заголовочный файл с жестким зацеплением нельзя подключить к программе сам по себе, он должен подключаться вместе с другими заголовочными файлами и в определенном порядке. Интерфейсы могут быть зацепленными по целому ряду причин, таких как совместное использование определенных структур данных, взаимная зависимость между функциями или работа с разделяемым глобальным состоянием. Однако сильное зацепление интерфейсов затрудняет модификацию логики программы, поскольку изменения могут распространяться по всей системе. Всегда старайтесь делать компоненты слабо зацепленными, независимо от того, принадлежат они к публичному интерфейсу или являются подробностями реализации программы.

Повторное использование кода. *Повторное использование кода* — это методика, согласно которой функциональность реализуется только раз и затем применяется на разных участках программы без дублирования. Дублирование кода может приводить к тонким и неожиданным расхождениям в поведении системы, чрезмерному увеличению размера исполняемых файлов и повышению расходов на сопровождение кода. Самая низкоуровневая единица функциональности, пригодная для повторного использования, — *функция*. В нее можно инкапсулировать любую логику, которая должна повторяться. Если функциональность имеет несущественные вариации, то функцию зачастую можно параметризовать, чтобы она имела многоцелевое назначение. Каждая функция должна выполнять работу, которая не дублируется никакой другой функцией. Их можно объединять, чтобы решать все более сложные задачи.

Абстракция данных. *Абстракция данных* — любой программный компонент, рассчитанный на повторное использование и соблюдающий четкое разделение между публичным интерфейсом абстракции и подробностями ее реализации. Публичный интерфейс любой абстракции данных состоит из определений типов, объявлений функций и определений констант, необходимых пользователю этой абстракции; он размещается в заголовочном файле. Подробности реализации того, как абстракция воплощена в коде, а также приватные вспомогательные функции размещаются в файле с исходным кодом или в отдельном от публичного интерфейса заголовочном файле.

Подобное разделение публичного интерфейса и приватной реализации позволяет изменять последнюю, не нарушая работу кода, зависящего от вашего компонента.

Заголовочные файлы обычно содержат объявления функций и определения типов компонента. Например, `<string.h>` из стандартной библиотеки C предоставляет публичный интерфейс для действий, связанных со строками, а в `<threads.h>` находятся служебные функции для работы с потоками выполнения. Такое логическое разделение позволяет добиться слабого зацепления и высокой связности. Этот подход позволяет обращаться только к тем компонентам, которые вам нужны, сокращая время компиляции и снижая вероятность конфликтов имен. Например, если вы хотите воспользоваться функцией `strlen`, то вам не нужно ничего знать об API для работы с потоками.

Еще один вопрос, о котором нужно подумать, состоит в том, нужно ли явным образом подключить заголовочные файлы, необходимые вашему заголовочному файлу, или пусть этим занимаются сами пользователи. С точки зрения абстрагирования данных заголовочные файлы лучше сделать самодостаточными и подключить к ним любые файлы, которые они задействуют. Если этого не сделать, то пользователям абстракции придется выполнять дополнительные действия, что чревато утечкой подробностей ее реализации.

Исходные файлы реализуют функциональность, объявленную заданным заголовочным файлом или программную логику, ориентированную на определенные задачи и применяемую для выполнения необходимых действий. Например, если у вас есть заголовочный файл `network.h`, описывающий публичный интерфейс для сетевого взаимодействия, то у вас может быть исходный файл `network.cpp`, который реализует его логику.

Подробности реализации можно разделять между двумя исходными файлами, используя общий заголовочный файл, но он должен храниться отдельно от публичного интерфейса, чтобы случайно не раскрыть эти подробности.

Библиотеки классов.

В традиционном процедурно-ориентированном программировании долгое время среди разработчиков ПО было принято предоставлять программистам библиотеки функций. С помощью комбинирования этих библиотек и добавления некоторых собственных процедур и функций получалась программа — продукт, представляемый конечному пользователю.

Библиотеки обычно содержат очень широкий спектр готовых функций, пригодных для различного применения. Например, разработчик может предложить библиотеку функций для статистической обработки данных или оптимизации работы с памятью компьютера.

Поскольку C++ построен на классах, а не на функциях, неудивительно, что библиотеки для программ на этом языке состоят из классов. Удивляет то, насколько библиотека классов лучше и прогрессивней старомодных библиотек

функций. Поскольку классы представляют собой совокупность данных и функций для их обработки, а также из-за того, что они лучше моделируют реальную жизнь, интерфейс между библиотеками классов и приложениями, которые их используют, гораздо понятнее, чем между библиотеками функций и приложениями.

Поэтому библиотеки классов являются более важным предметом при программировании на C++, чем библиотеки функций при традиционном программировании. Использование этих библиотек освобождает программиста от очень многих забот. При разработке приложений оказывается, что если доступны необходимые библиотеки, то для создания полнофункционального продукта необходим минимум ручной работы по программированию. К тому же необходимо учитывать, что создается все больше и больше библиотек, а значит, все больший и больший спектр различного программного обеспечения можно создавать без лишних проблем.

Крайне важный пример библиотеки классов, это *Стандартная библиотека C++ (STL)*.

Обычно библиотека классов состоит из интерфейса (*interface*) и реализации (*implementation*). Это две основные части, которые присутствуют практически всегда.

Интерфейс. Для того чтобы библиотеку можно было использовать, программисту необходим доступ к различным определениям, включая объявления классов. Они представляют собой общедоступную часть библиотеки и обычно поставляются в виде исходного кода в составе заголовочного файла с расширением *.h*. Обычно этот файл включается в текст программы с помощью директивы *#include*.

Эти определения называются интерфейсом, потому что это та часть библиотеки, которую программист видит и с помощью которой он взаимодействует с классами. При этом ему нет никакой нужды вникать в содержимое другой части библиотеки под названием реализация (*implementation*).

Реализация. Если интерфейсную часть можно назвать фасадом здания, то реализация представляет собой его внутренности. В контексте библиотеки классов реализация — это содержимое классов. Как туристу, любующемуся архитектурными памятниками, вовсе не обязательно знать, что находится внутри зданий, так и программисту не нужно вникать в то, как происходит внутренняя работа библиотечных классов. К тому же разработчики библиотек в большинстве случаев просто не предоставляют клиентам исходных кодов, так как они могут быть нелегально использованы или изменены. Поэтому методы классов чаще всего поставляются в виде объектных (*.OBJ*) или библиотечных (*.LIB*) файлов.

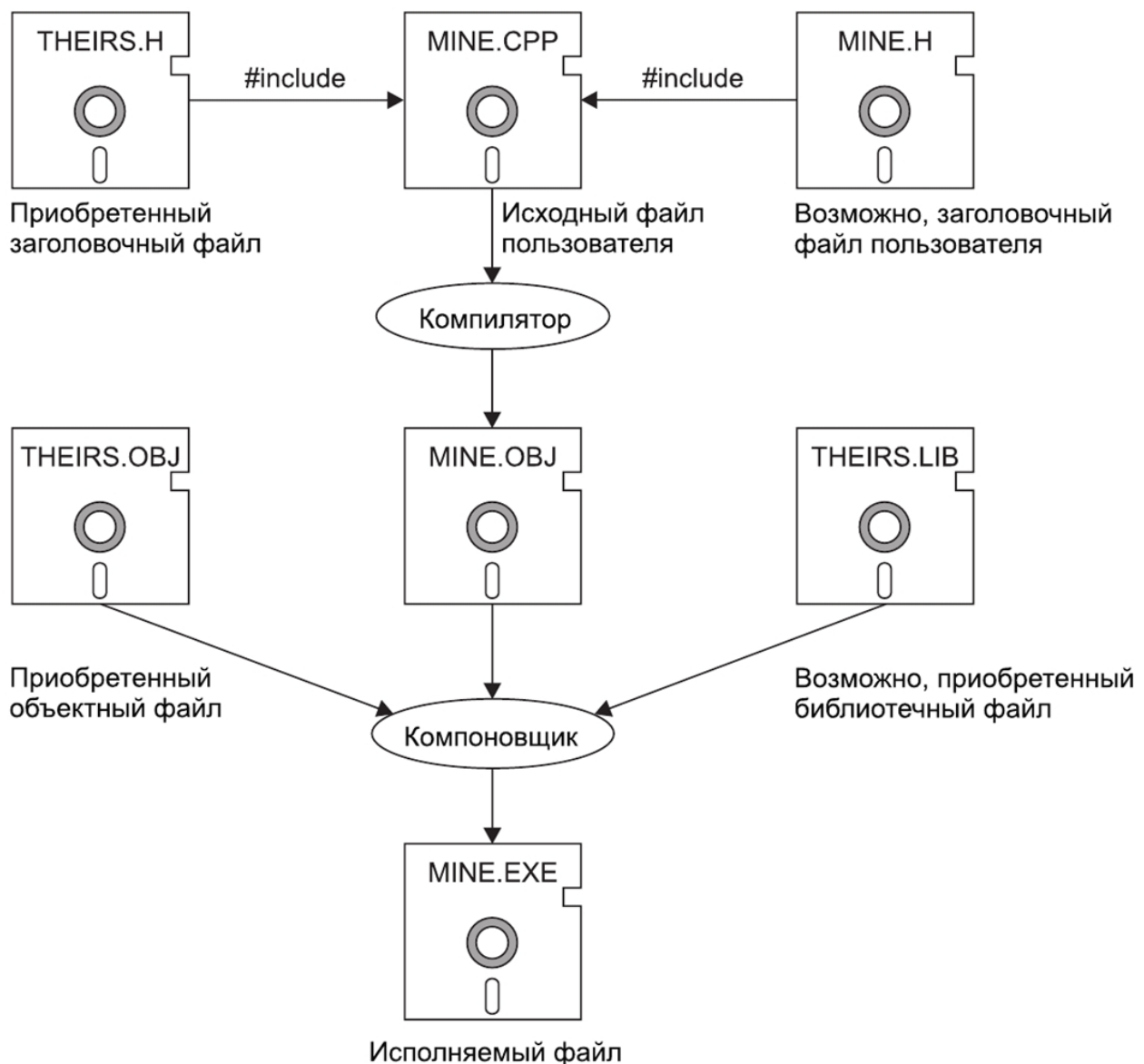


Рис. 23 — Структура многофайлового приложения

Потоковый ввод-вывод

В языке C++ предусмотрено две системы ввода-вывода. Первая система унаследована от языка C. Вторая, объектно-ориентированная система, присуща только языку C++. Хотя функции *printf()* и *scanf()* по-прежнему доступны, C++ обеспечивает иной, лучший способ выполнения операций ввода и вывода в консоль. Следует отметить, что в C++ ввод/вывод выполняется с использованием операций, а не функций ввода/вывода. Кроме того, система ввода-вывода языка C++ оперирует потоками – логическими устройствами, получающими или передающими информацию. Поток связан с физическим устройством ввода-вывода. Все потоки функционируют одинокого, хотя физические устройства могут быть разными.

Консольный ввод-вывод. Для работы с консолью (экран + клавиатура) необходимо подключить заголовочный файл `<iostream>`. Данный заголовочный файл определяет:

- объекты ввода `cin` и вывода `cout`, связанные с консолью;
- операцию `>>`, которая используется для извлечения данных из входного потока;
- операцию `<<`, которая используется для помещения данных в выходной поток;
- операции форматированного ввода-вывода.

Пример: Написать программу для вывода числа π на экран с использованием потокового вывода C++. Вывести число π с 10, 5, 4, 3 знаками после запятой.

Для задания точности вывода и формата вывода числа, необходимо помимо заголовочного файла `<iostream>` использовать `<iomanip>`.

Листинг программы:

```
1  #include <iostream>
2  #include <iomanip>
3  #include <math.h>
4
5  using namespace std;
6
7  int main()
8  {
9      cout << fixed;
10     // Вывод числа Пи с кол-вом знаков по умолчанию
11     cout << "PI = " << M_PI << endl;
12     // Вывод числа Пи с 10 знаками после запятой
13     cout << "PI = " << setprecision(10) << M_PI << endl;
14     // Вывод числа Пи с 4 знаками после запятой
15     cout << "PI = " << setprecision(4) << M_PI << endl;
16     // Вывод числа Пи с 3 знаками после запятой
17     cout << "PI = " << setprecision(3) << M_PI << endl;
18     return 0;
19 }
20
```

Модифицируем программу и выполним ввод с консоли максимального числа знаков после запятой и выполним вывод в цикле.

```

1  #include <iostream>
2  #include <iomanip>
3  #include <math.h>
4
5  using namespace std;
6
7  int main()
8  {
9      int N;
10     // Ввод с консоли
11     cout << "N = ";
12     cin >> N;
13     // Вывод на консоль
14     cout << fixed;
15     for (int i = 1; i < N + 1; i++)
16         cout << "PI = " << setprecision(i) << M_PI << endl;
17
18     return 0;
19 }
20

```

Результаты работы программы:

```

N = 10
PI = 3.1
PI = 3.14
PI = 3.142
PI = 3.1416
PI = 3.14159
PI = 3.141593
PI = 3.1415927
PI = 3.14159265
PI = 3.141592654
PI = 3.1415926536

```

Файловый ввод-вывод

Как было сказано ранее, ввод-вывод в С++ осуществляется с помощью потоков. Различают три вида потоков:

- стандартные;
- файловые;
- строковые.

Работу со стандартным потоком мы уже рассмотрели, перейдем к рассмотрению принципов работы с файловыми потоками, которые предназначены для обмена информацией с файлами на внешних носителях.

В С++ реализованы три класса для работы с файлами: *ifstream* — класс входных файловых потоков; *ofstream* — класс выходных файловых потоков; *fstream* — класс двунаправленных файловых потоков. Эти классы являются

производными от классов *istream*, *ostream* и *iostream* соответственно, поэтому они наследуют перегруженные операции << и >>, а также манипуляторы для управления форматированием данных.

Работу с файлами через потоки ввода-вывода можно осуществить, подключив к программе заголовочный файл <fstream>.

В общем случае, при программном использовании файлов предполагаются следующие последовательные этапы: создание потока нужного типа; открытие потока и связывание с ним файла; обмен данных (ввод-вывод); закрытие файла.

Модифицируем код последней программы для вывода результатов расчета числа π в выходной файл данных.

Листинг программы:

```
1  #define _USE_MATH_DEFINES // Для работы с константой M_PI
2
3  #include <fstream> // Для работа с файловыми потоками
4  #include <iomanip> // Для работы с форматом данных
5  #include <cmath> // Для работы с математическими функциями
6
7  int main()
8  {
9      std::ifstream inFile; // Входной файл
10     inFile.open("pi-in.txt", std::ios::in); // Открываем файл
11     int N = 5; // Значение по умолчанию
12     inFile >> N; // Читаем одну строку данных из файла
13     inFile.close(); // Закрываем файл
14
15     std::ofstream outFile("pi-out.txt"); // Выходной файл
16     outFile << std::fixed; // Задаем формат вывода
17     for (int i = 1; i < N + 1; i++)
18     {
19         // Задаем кол-во знаков после запятой и выводим число Пи
20         outFile << "PI = " << std::setprecision(i) << M_PI << std::endl;
21     }
22     outFile.close(); // Закрываем файл
23
24     return EXIT_SUCCESS;
25 }
26
```

В приведенном листинге программы осуществляется ввод максимального количества знаков числа π из файла pi-in.txt. Для этого используется переменная inFile типа std::ifstream. Открытие файла осуществляется с помощью метода open, которые принимает два параметра: 1 — имя файла, 2 — флаг, определяющий режим доступа. Возможные режимы доступа к файлу приведены в Табл. 1.

Табл. 1 — Возможные значения флага `std::ios` для задания режима открытия файла

Constant	Explanation
app	seek to the end of stream before each write
binary	open in binary mode
in	open for reading
out	open for writing
trunc	discard the contents of the stream when opening
ate	seek to the end of stream immediately after open
noreplace (C++23)	open in exclusive mode

Далее, во второй части листинга программы, осуществляется создание выходного файла с данными, для этого используется переменная `outFile` типа `std::ofstream`. При определении переменной в круглых скобках указывается параметр конструктора класса (понятие *конструктор класса* будет изучено в дальнейшем в рамках данного курса). В качестве параметра используется имя файла, в который будут записаны результаты работы программы.

Обратите внимание, что после того, как работа с файлом закончена, для переменной, связанной с файлом, необходимо вызвать метод `close()` который закроет файл и освободит дескриптор, связанный с файлом.

Модифицируем предыдущий листинг программы для того, чтобы результаты работы программы дописывались в выходной файл, а не создавали его снова. Также, воспользуемся синтаксической конструкцией, которая позволит задать пространство имен, чтобы избавиться от префикса `std::` перед некоторыми операторами в листинге программы. Соответствующая конструкция добавлена в строке 7

```
using namespace std;
```

После использования этой конструкции, все объявления классов и функций стандартной библиотеки могут быть сокращены. Например, объявления переменной `std::ifstream inFile;` можно будет сократить до `ifstream inFile;` Вызов метода вывода данных в консоль `std::cout <<` сократить до `cout <<` ... и т. д.

Листинг программы:

```
1  #define _USE_MATH_DEFINES // Для работы с константой M_PI
2
3  #include <fstream> // Для работа с файловыми потоками
4  #include <iomanip> // Для работы с форматом данных
5  #include <cmath> // Для работы с математическими функциями
6
7  using namespace std;
8
9  int main()
10 {
11     ifstream inFile; // Входной файл
12     inFile.open("pi-in.txt", ios::in); // Открываем файл
13     int N = 5; // Значение по умолчанию
14     inFile >> N; // Читаем одну строку данных из файла
15     inFile.close(); // Закрываем файл
16
17     ofstream outFile("pi-out.txt", ios::app); // Выходной файл
18     outFile << fixed; // Задаем формат вывода
19     for (int i = 1; i < N + 1; i++)
20     {
21         // Задаем кол-во знаков после запятой и выводим число Пи
22         outFile << "PI = " << setprecision(i) << M_PI << endl;
23     }
24     outFile.close(); // Закрываем файл
25
26     return EXIT_SUCCESS;
27 }
28
```

Использование пространства имен

В соответствии со спецификацией Microsoft *пространство имен* — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.). Пространства имен используются для организации кода в виде логических групп и с целью избежания конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек. Все идентификаторы в пределах пространства имен доступны друг другу без уточнения. Идентификаторы за пределами пространства имен могут получить доступ к членам с помощью полного имени для каждого идентификатора (`std::vector<std::string> vec;`) или директивы `using` для всех идентификаторов в пространстве имен (`using namespace std;`).

Использование пространства имен проиллюстрировано в примере выполнения задания (Часть III).

Варианты задания

В рамках лабораторной работы необходимо реализовать три отдельных проекта, организованных в одно программное решение.

Часть I – Написать консольное приложение. Ввод исходных данных допускается непосредственно в теле программы.

1. Даны катеты прямоугольного треугольника a и b . Найти гипотенузу c и углы треугольника α , β .
2. Известна гипотенуза c и прилежащий угол α прямоугольного треугольника. Найти площадь треугольника S и угол β .
3. Известна диагональ квадрата d . Вычислить площадь S и периметр P квадрата.
4. Дан диаметр окружности d . Найти длину окружности L и площадь круга S .
5. Даны три числа — a , b , c . Найти среднее арифметическое и среднее геометрическое заданных чисел.
6. Даны катеты прямоугольного треугольника a и b . Найти гипотенузу c и периметр P .
7. Дана длина окружности L . Найти радиус окружности R и площадь круга S .
8. Даны два ненулевых числа a и b . Найти сумму S , разность R , произведение P и частное d квадратов заданных чисел.
9. Поменять местами содержимое переменных A и B и вывести новые значения A и B .
10. Точки A и B заданы координатами на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$. Найти длину отрезка AB .
11. Заданы два катета прямоугольного треугольника a и b . Вычислить площадь S и периметр P .
12. Даны переменные A , B , C . Изменить их значения, переместив содержимое A в B , B — в C , C — в A , и вывести новые значения переменных A , B , C .
13. Известна диагональ ромба d . Вычислить площадь S и периметр P .
14. Найти значение функции $y = 4 \cdot (x + 1)^3 + 5 \cdot (x - 1)^5 + 2$ и её производной при заданном значении x .
15. Даны два ненулевых числа a и b . Найти сумму S , разность R , произведение P и частное D модулей заданных чисел.

16. Известны координаты вершин квадрата $ABCD$: $A(x_1, y_1)$ и $C(x_2, y_2)$. Найти площадь S и периметр P .
17. Даны длины сторон прямоугольника a и b . Найти площадь S и периметр P .
18. Известно значение периметра P равностороннего треугольника. Вычислить площадь S .
19. Задан периметр квадрата P . Вычислить сторону квадрата a , диагональ d и площадь S .
20. Дана сторона квадрата a . Вычислить периметр квадрата P , его площадь S и длину диагонали d .
21. Три точки заданы координатами на плоскости: $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$. Найти длины отрезков AB и BC .
22. Даны переменные A , B , C . Изменить их значения, переместив содержимое A в C , C — в B , B — в A , и вывести новые значения переменных A , B , C .
23. Даны числа — a_1, a_2, a_3, a_4, a_5 . Найти их среднее арифметическое и среднее геометрическое значения.
24. Найти значение функции $y = \frac{3}{2} \cdot (x + 3)^4 - \frac{1}{5} \cdot (x - 1)^5$ и её производной при заданном значении x .
25. Точки A и B заданы координатами в пространстве: $A(x_1, y_1, z_1)$, $B(x_2, y_2, z_2)$. Найти длину отрезка AB .

 JeSuisUnGeek

Часть II – Написать консольное приложение. Ввод исходных данных осуществлять через консоль, вывод результатов программы осуществить в текстовый файл.

1. Расстояние L задано в сантиметрах. Найти количество полных метров в нём и остаток в сантиметрах.
2. Масса M задана в килограммах. Найти количество полных тонн в ней и остаток в килограммах.
3. Размер файла B дан в байтах. Найти количество полных килобайтов, которые занимает данный файл и остаток в байтах.
4. Дано двузначное число. Вывести на экран количество десятков и единиц в нём.
5. Дано двузначное число. Найти сумму его цифр.
6. Дано двузначное число. Найти произведение его цифр.
7. Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.
8. Дано трехзначное число. Определить, сколько в нём единиц, десятков и сотен.
9. Дано трехзначное число. Найти сумму его цифр.
10. Дано трехзначное число. Найти произведение его цифр.
11. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и десятков исходного числа.

 JeSuisUnGeek

12. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и единиц исходного числа.
13. Дано трехзначное число. Вывести число, полученное при перестановке цифр десятков и единиц исходного числа.
14. С начала суток прошло N секунд. Найти количество полных минут, прошедших с начала суток и остаток в секундах.
15. С начала суток прошло N секунд. Найти количество полных часов, прошедших с начала суток и остаток в секундах.
16. Дано двузначное число. Найти сумму квадратов его цифр.
17. Дано двузначное число. Найти квадрат разности его цифр.
18. Расстояние L задано в метрах. Найти количество полных километров в нём и остаток в метрах.
19. Масса M задана в граммах. Найти количество полных килограммов в ней и остаток в граммах.
20. Размер файла B дан в килобайтах. Найти количество полных мегабайтов, которые занимает данный файл и остаток в килобайтах.
21. Расстояние L задано в дециметрах. Найти количество полных метров в нём и остаток в сантиметрах.
22. С начала года прошло K дней. Найти количество полных недель, прошедших с начала года и остаток в днях.
23. С начала года прошло K часов. Найти количество полных дней, прошедших с начала года и остаток в часах.
24. Дано трехзначное число. Найти сумму квадратов его цифр.
25. Дано трехзначное число. Найти квадрат суммы его цифр.



Часть III — Написать консольное приложение. Ввод и вывод данных осуществлять с помощью текстовых файлов.

В соответствии с вариантом задания вычислить значение математической функции. Начальное и конечное значения аргумента, а также шаг изменения аргумента использовать в соответствии со значениями, представленными в табл. 5. Для каждой функции определить максимальное и минимальное значения. Результат табулирования функции вывести в текстовый файл, при выводе вещественных значений ограничиться двумя знаками после запятой для значения аргумента и четырьмя знаками для значения функции.

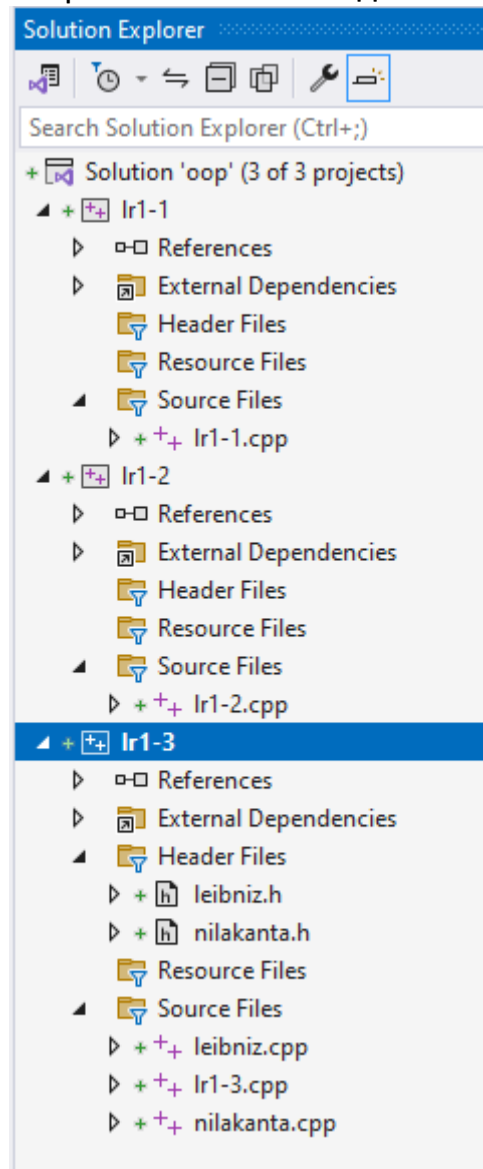
Табл. 2 — Варианты задания

№	Функция	x_0	x_f	h_x
1	$y(x) = 3x^3 - 15x^2 - 12x + 8$	-5	15	0,01
2	$y(x) = x^2 \sin^3(3x)$	-3	2,5	0,001
3	$y(x) = x^3 \sin^5(4x)$	-1,6	3,2	0,001
4	$y(x) = (1 + 6 \sin^2(x))^{1/2} \cos(3x)$	-5	12	0,001
5	$y(x) = 2x^4 - 12x^3 - 4x^2 + 2$	-5	10	0,01
6	$y(x) = \cos(2,5x) \sin(3x) - 0,2$	-7	7	0,01
7	$y(x) = 4^{-(x-1)^2} \cos(2x)$	-10	20	0,1
8	$y(x) = \cos(2,5x) \sin(3x) - 0,2$	-3,2	8	0,001
9	$y(x) = x(\sin(3x) + \cos(2x))$	-3,5	4,5	0,01
10	$y(x) = x^2(\sin^2(3x) - \cos^2(2x))$	-3,5	5,5	0,1

Пример выполнения работы

Реализация выполнена в IDE Microsoft Visual Studio Community 2022.

Структура программного решения имеет вид



Файлы реализации *.cpp находятся в папке *Source Files*, заголовочные файлы *.h папке *Header Files*.

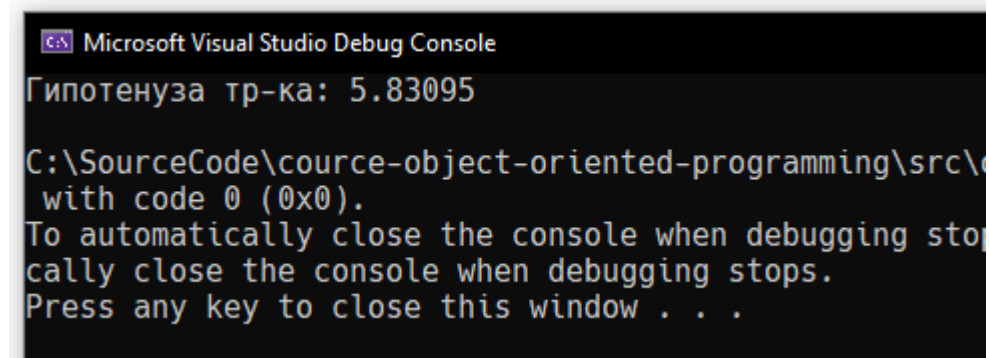
Часть I – Написать консольное приложение для вычисления гипотенузы c прямоугольного треугольника ABC со сторонами $a=3$ и $b=5$

Листинг программы:

lr1-1.cpp

```
1  #include <iostream>
2  #include <math.h>
3
4  int main()
5  {
6      // Для вывода кириллицы в консоль
7      setlocale(LC_ALL, "Russian");
8
9      double a = 3; // Катет 1
10     double b = 5; // Катет 2
11     double c = std::sqrt(a * a + b * b); // Гипотенуза
12     std::cout << "Гипотенуза тр-ка: " << c << std::endl;
13     return EXIT_SUCCESS;
14 }
```

Результаты работы:



Microsoft Visual Studio Debug Console

Гипотенуза тр-ка: 5.83095

C:\SourceCode\course-object-oriented-programming\src\
with code 0 (0x0).

To automatically close the console when debugging stops,
manually close the console when debugging stops.

Press any key to close this window . . .

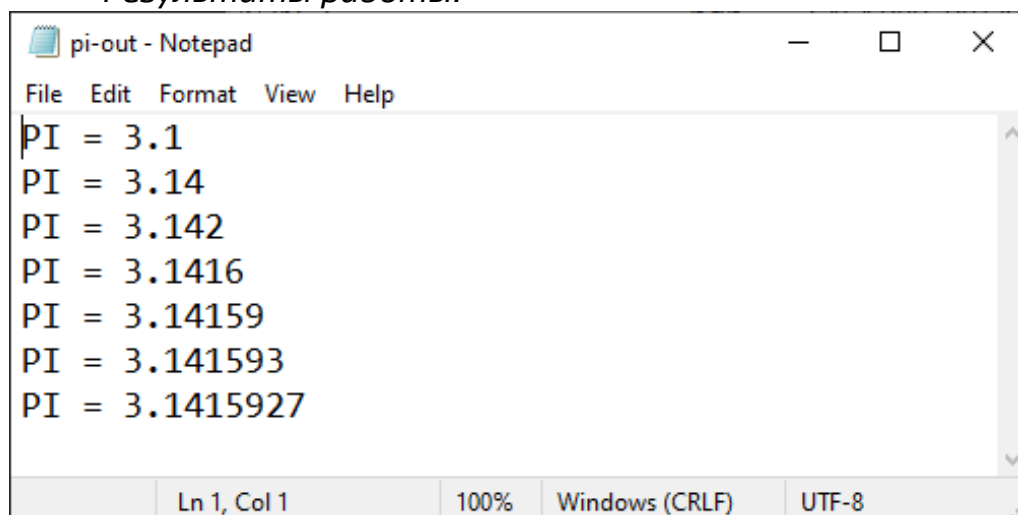
Часть II – Написать консольное приложение для вывода числа Пи с заданным числом знаков после запятой. Вывод осуществить в текстовый файл pi-out.txt

Листинг программы:

lr1-2.cpp

```
1  #define _USE_MATH_DEFINES // Для работы с константой M_PI
2
3  #include <fstream> // Для работа с файловыми потоками
4  #include <iomanip> // Для работы с форматом данных
5  #include <cmath> // Для работы с математическими функциями
6
7  using namespace std;
8
9  int main()
10 {
11     ifstream inFile; // Входной файл
12     inFile.open("pi-in.txt", ios::in); // Открываем файл
13     int N = 5; // Значение по умолчанию
14     inFile >> N; // Читаем одну строку данных из файла
15     inFile.close(); // Закрываем файл
16
17     ofstream outFile("pi-out.txt", ios::app); // Выходной файл
18     outFile << fixed; // Задаем формат вывода
19     for (int i = 1; i < N + 1; i++)
20     {
21         // Задаем кол-во знаков после запятой и выводим число Пи
22         outFile << "PI = " << setprecision(i) << M_PI << endl;
23     }
24     outFile.close(); // Закрываем файл
25
26     return EXIT_SUCCESS;
27 }
```

Результаты работы:



```
pi-out - Notepad
File Edit Format View Help
PI = 3.1
PI = 3.14
PI = 3.142
PI = 3.1416
PI = 3.14159
PI = 3.141593
PI = 3.1415927
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```


Часть III – Написать консольное приложение для вычисления числа Пи с помощью ряда Лейбница: $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$ и с помощью ряда Нилаканты:

$$\pi = 3 + \frac{4}{2*3*4} - \frac{4}{4*5*6} + \frac{4}{6*7*8} - \frac{4}{8*9*10} + \frac{4}{10*11*12} - \dots$$

Расчет числа Пи реализовать в отдельных модулях, используя заголовочные файлы *.h и файлы реализации *.cpp. Входные параметры функции (elementsCount – число элементов ряда, digitsCount – количество знаков после запятой в результирующем числе)

Листинг программы:

leibniz.h

```
1  #pragma once
2  #include <iomanip>
3  #include <iostream>
4  #include <sstream>
5  #include <string>
6
7  namespace lb
8  {
9      std::string calculatePI(int elementsCount, int digitsCount);
10 }
```

leibniz.cpp

```
1  #include "leibniz.h"
2
3  namespace lb
4  {
5      std::string calculatePI(int elementsCount, int digitsCount)
6      {
7          double PI = 3.00;
8          short sign = 1;
9          int n = 2;
10
11          double tmp = 0;
12
13          for (int i = 0; i < elementsCount; i++)
14          {
15              PI += sign * (4.00 / ((n) * (n + 1) * (n + 2)));
16              sign *= -1;
17              n += 2;
18          }
19
20          std::stringstream stream;
21          stream << std::fixed;
22          stream << std::setprecision(digitsCount) << PI;
23          return stream.str();
24      }
25 }
```


nilakanta.h

```
1  #pragma once
2  #include <iomanip>
3  #include <iostream>
4  #include <sstream>
5  #include <string>
6
7  namespace nl
8  {
9      std::string calculatePI(int elementsCount, int digitsCount);
10 }
```

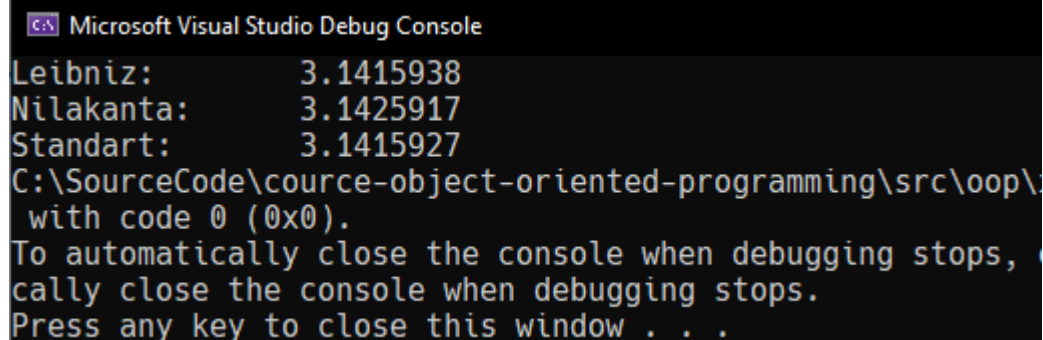
nilakanta.cpp

```
1  #include "nilakanta.h"
2
3  namespace nl
4  {
5      std::string calculatePI(int elementsCount, int digitsCount)
6      {
7          double PI = 1.00;
8          short sign = -1;
9          int n = 3;
10
11          double tmp = 0;
12
13          for (int i = 0; i < elementsCount; i++)
14          {
15              PI += sign * (1.00 / n);
16              sign *= -1;
17              n += 2;
18          }
19          PI *= 4;
20
21          std::stringstream stream;
22          stream << std::fixed;
23          stream << std::setprecision(digitsCount) << PI;
24          return stream.str();
25      }
26 }
```

lr1-3.cpp

```
1  #define _USE_MATH_DEFINES // Для работы с константой M_PI
2
3  #include <iostream> // Для работы с потоками ввода / вывода
4  #include <iomanip> // Для работы с форматом данных
5  #include <cmath> // Для работы с математическими функциями
6
7  #include "leibniz.h"
8  #include "nilakanta.h"
9
10 int main()
11 {
12     int digitsCount = 7; // Количество знаков после запятой
13     // Вызов функции из пространства имен Leibniz
14     std::cout << "Leibniz:\t" << lb::calculatePI(1000, digitsCount) << std::endl;
15     // Вызов функции из пространства имен Nilakanta
16     std::cout << "Nilakanta:\t" << nl::calculatePI(1000, digitsCount) << std::endl;
17     // Сравнение результатов с константой модуль math
18     std::cout << std::fixed;
19     std::cout << "Standart:\t" << std::setprecision(digitsCount) << M_PI;
20
21     return EXIT_SUCCESS;
22 }
```

Результаты работы:



Microsoft Visual Studio Debug Console

```
Leibniz:      3.1415938
Nilakanta:    3.1425917
Standart:     3.1415927
C:\SourceCode\course-object-oriented-programming\src\oop\
with code 0 (0x0).
To automatically close the console when debugging stops,
call close the console when debugging stops.
Press any key to close this window . . .
```

Контрольные вопросы

1. Обоснуйте необходимость представление программного кода в виде отдельных модулей.
2. Что из себя представляет программный модуль, написанный на C++?
3. Расскажите о дополнительных библиотеках C++. Приведите примеры использования модулей.
4. Что такое расширение пространства имен, в каких случаях его нужно использовать. Какие варианты использования пространства имен вы знаете?
5. Опишите правила работы с файлами в C++. Чтение из файла.
6. Опишите правила работы с файлами в C++. Запись в файл.
7. Как реализуется форматированный вывод в C++?
8. Как создать программное решение, состоящее из нескольких проектов в IDE Qt Creator.
9. Как создать программное решение, состоящее из нескольких проектов в IDE Microsoft Visual Studio.
10. Сравните IDE Microsoft Visual Studio и IDE Qt Creator.

Список использованных источников

1. **Лафоре Р. Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е издание / Р. Лафоре — СПб.: Питер, 2021. — 928 с.**
2. **Шилдт Г. C++: полное руководство, классическое издание Python для сложных задач: наука о данных и машинное обучение / Г. Шилдт — Пер. с англ. — СПб.: ООО «Диалектика», 2020. — 800 с.**
3. **Дейтел П. C++20 для программистов / П. Дейтел — СПб.: Питер, 2024. — 1056 с.**
4. **Вайсфельдт М. Объектно-ориентированный подход. 5-е межд. Изд. / М. Вайсфельдт — СПб.: Питер, 2020. — 256 с.**
5. **Нобак М. Объекты. Стильное ООП. / М. Нобак — СПб.: Питер, 2023. — 304 с.**
6. **Васильев А. Программирование на C++ в примерах и задачах / А. Васильев — М.: Эксмо, 2021. — 368 с.**

Справочная информация

Стандартные функции для вычисления математических операций

Обозначение	Действие
abs(x)	Модуль целого числа x

Обозначение	Действие
fabs(x)	Модуль вещественного числа x
sin(x)	Синус числа x
cos(x)	Косинус числа x
tan(x)	Тангенс числа x
atan(x)	Арктангенс числа x , $x \in (-\frac{\pi}{2}; \frac{\pi}{2})$
acos(x)	Аркосинус числа x
asin(x)	Арсинус числа x
exp(x)	Экспонента, e^x
log(x)	Натуральный логарифм, ($x > 0$)
log10(x)	Десятичный логарифм, ($x > 0$)
sqrt(x)	Корень квадратный, ($x > 0$)
pow(x,y)	Возведение числа x в степень y
ceil(x)	Округление числа x до ближайшего большего целого
floor(x)	Округление числа x до ближайшего меньшего целого

Пример записи математических выражений с использованием стандартных функций

Математическая запись	Запись на языке C++
$\sqrt[3]{(a+b)^2}$	pow((a+b)*(a+b),1./3) ИЛИ pow(pow(a+b,2),1./3)
$\cos^4(x)$	pow(cos(x), 4)
e^{2x}	exp(2*x)
$e^{5 \sin(\frac{x}{2})}$	exp(5*sin(x/2))
$\sin^2(\sqrt{x})$	pow(sin(sqrt(x)),2)
$\ln(x-2)$	log(fabs(x-2))
$\log_b a$	log(a)/log(b)
$\frac{\lg(x^2+1)}{\lg(4)}$	log10(x*x+1)/log10(4)
$\sin(x^2+y^2) + \cos \frac{(x^2+y^2)}{2 \cdot y} + \sqrt{x^2+y^2}$	z=x*x+y*y; sin(z)+cos(z/(2*y))+sqrt(z);

Полезные ссылки

1. <https://ru.cppreference.com/w/> — Справочник по языку C++ на русском языке.
2. <https://cplusplus.com/> — Справочник по языку C++ на английском языке.
3. <https://visualstudio.microsoft.com/vs/community/> — Страница загрузки Microsoft Visual Studio Community Edition.
4. <https://www.qt.io/download-open-source> — Страница Qt Creator Community Edition.

Учебное издание

Василий Викторович Альчаков

ИССЛЕДОВАНИЕ МОДУЛЬНОГО ПОДХОДА К СОЗДАНИЮ ПРОГРАММ

Методические указания к выполнению лабораторной работы

Оригинал-макет и вёрстка В.В. Альчаков

© СевГУ, 2024

© Альчаков В.В., 2024