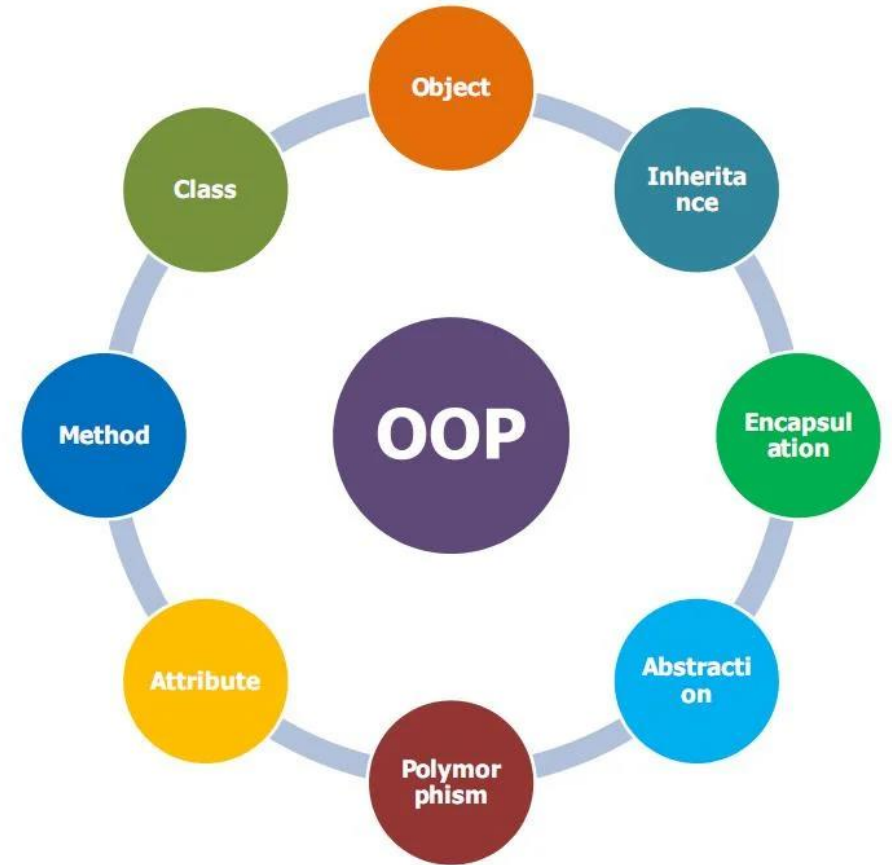


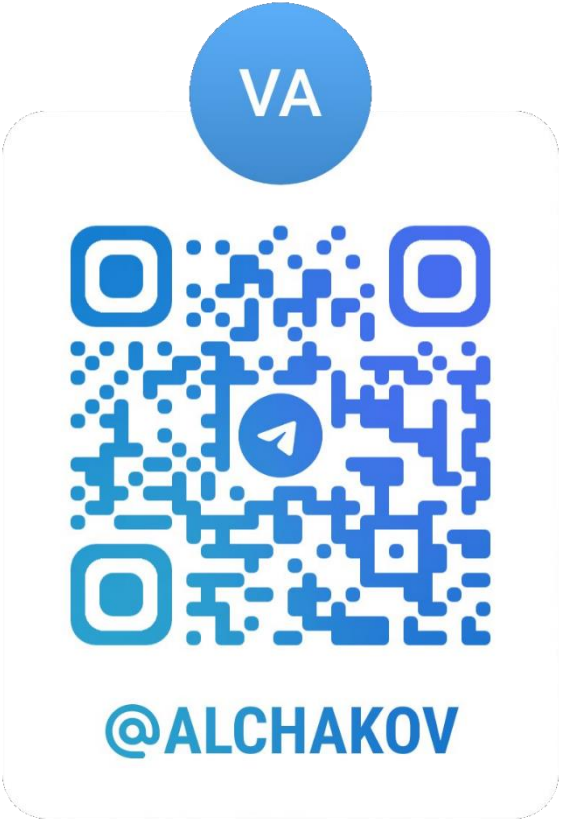
Практика №1

Введение.

Модульный подход к построению программ.



Ведущий преподаватель
АЛЬЧАКОВ Василий Викторович
канд. техн. наук, доцент кафедры ИУТС



Курс	Семестр	Общий объем, ЗЕ (ч)	Контактная работа, ч			Самостоятельная работа, ч	Контроль, ч	РГЗ, контрольная работа	Курсовой проект (курсовая работа)	Зачет (семестр)	Экзамен (семестр)
			Лекции	Практические занятия	Лабораторные занятия						
Очная форма обучения											
3	5	5 (180)	30	36	18	60	36	—	—	—	5

Перечень лабораторных работ

ЛР №1: Основы языка C++. Исследование модульного подхода к созданию программ

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

Задание на работу: создать комплексное решение, объединяющее несколько проектов с помощью IDE Visual Studio и Qt Creator. Реализовать три консольных приложения, в соответствии с вариантом задания. Для первого задания использовать жесткое кодирование входных параметров, для второго задания использовать ввод через консоль, для третьего – ввод исходных параметров и вывод результатов работы в текстовые файлы.

ЛР №2: Создание пользовательского класса на языке C++. Исследование механизмов инкапсуляции

Цель работы: разработать пользовательский класс для выбранной предметной области (или, заданной по номеру варианта). Изучить механизм инкапсуляции.

Задание на работу: реализовать заданную структуру в виде программного кода. Атрибуты класса должны быть закрытыми, доступ к атрибутам осуществляется с помощью *get* и *set* методов. Атрибуты класса должны использовать максимально возможное количество типов данных языка C++ (включая контейнеры или массивы). В основной программе необходимо создать несколько экземпляров разработанного класса, инициализировать заданные атрибуты, вывести информацию об экземпляре класса на экран. Обосновать использования инкапсуляции для разработки пользовательского класса

Перечень лабораторных работ

ЛР №3: Динамическое создание объектов пользовательских классов с помощью операторов new и delete. Использование работы конструкторов и деструкторов. Работа с умными указателями.

Цель работы: изучить особенности работы с экземплярами классов с помощью динамического выделения памяти. Изучить механизм работы конструкторов и деструкторов объектов. Изучить работу с умными указателями.

Задание на работу: доработать класс из ЛР № 2, добавив в него конструкторы (по умолчанию, с параметрами, копирования) и деструктор. Создание и удаление объекта класса выполнить с помощью операторов new и delete соответственно. Проанализировать преимущества работы с динамической памятью. Изучить механизм использования умных указателей. Сравнить работу умных указателей с работой обычных указателей.

Перечень лабораторных работ

ЛР №4: Создание иерархии классов. Исследование механизмов наследования.

Цель работы: научиться создавать базовые классы и классы наследники. Исследовать механизм наследования.

Задание на работу: для классов, созданных в ЛР № 2–3 продумать иерархию объектов. Разработать базовый класс и несколько классов наследников. В базовом классе выделить общие свойства и методы для объектов исследуемой предметной области, в классах наследниках сосредоточить уникальные свойства, описывающие объекты, порождённые от базового класса. Исследовать свойства декомпозиции на примере разработанной иерархии, доказать ее целесообразность. При реализации методов базового класса и классов наследников предусмотреть различные варианты обеспечения доступа к методам и атрибутам базового класса. На примере продемонстрировать механизм наследования методов базового класса, механизм наследования конструкторов. Проиллюстрировать как работает передача параметров конструктора в конструктор класса наследника и базового класса.

Перечень лабораторных работ

ЛР №5: Создание полиморфных функций. Исследование механизмов полиморфизма

Цель работы: научиться создавать полиморфные функции. Изучить механизм полиморфизма.

Задание на работу: для классов, созданных в ЛР № 2–4 модифицировать базовый класс таким образом, чтобы в нем была реализована полиморфная функция. Реализацию выполнить несколькими способами: с помощью виртуальных функций и с помощью чисто виртуальных функций. Создать абстрактный класс. Обосновать необходимость использования механизма полиморфизма для разработанной иерархии классов. Объяснить, что такое динамический полиморфизм.

ЛР №6: Перегрузка операторов

Цель работы: изучить механизм перегрузки операторов для объектов пользовательского класса.

Задание на работу: для классов, разработанных в ЛР № 2–5 реализовать перегрузку операторов (до двух-трех операторов). Проиллюстрировать работу перегруженных операторов на примере.

Отчет по курсу лабораторных работ

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»

ОТЧЕТ
по лабораторному практикуму по дисциплине
«Объектно-ориентированный анализ и проектирование»

№ ЛР	Дата	Оценка	Подпись
1			
2			
3			
4			
5			
6			
7			

Выполнил:
ст. группы УТС/6-22-**X**-о
Фамилия И.О.

Принял:
канд. техн. наук, доцент кафедры ИУТС
Альчаков В.В.

Севастополь
2024

Мэтт Вайсфельд

5-е международное издание

Объектно-ориентированный подход



Рекомендованная литература

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ С ПРИМЕРАМИ ПРИЛОЖЕНИЙ

Третье издание

ГРАДИ БУЧ, РОБЕРТ А. МАКСИМЧУК, МАЙКЛ У. ЭНГЛ, БОББИ ДЖ. ЯНГ, ДЖИМ КОНАЛТЕН, КЕТЛИ А. ХЬУСТОН

ADDISON-WESLEY

BOOCH JACOBSON RUMBAUGH

РЕДАКТОРЫ СЕРИИ

СЕРИЯ ОБЪЕКТНЫЕ ТЕХНОЛОГИИ

ПРОГРАММИРОВАНИЕ

принципы и практика использования C++

Исправленное издание

Бьярне Страуструп

Издательский дом "Вильямс"

Москва • Санкт-Петербург • Киев

2011

Васильев А.Н.

ПРОГРАММИРОВАНИЕ

НА C++

В ПРИМЕРАХ И ЗАДАЧАХ

- Основные сведения о синтаксисе и концепции языка
- От простых объектно-ориентированных программ до многопоточного программирования
- Примеры и задачи для самостоятельного изучения
- Для студентов и самостоятельного изучения

РОССИЙСКИЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР



Труды Шилда представляют программистам об интереснейшей теме: теории, доказательствах, кратко и авторитетно!

ACM Computing Reviews

КЛАССИЧЕСКОЕ ИЗДАНИЕ

C++

Герберт Шилдт

БОНУС!

Матрица кодов доступа к базе данных

КЛАССИКА COMPUTER SCIENCE

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ


В C++

4-Е ИЗДАНИЕ

Р. ЛАФОРЭ

SAMS

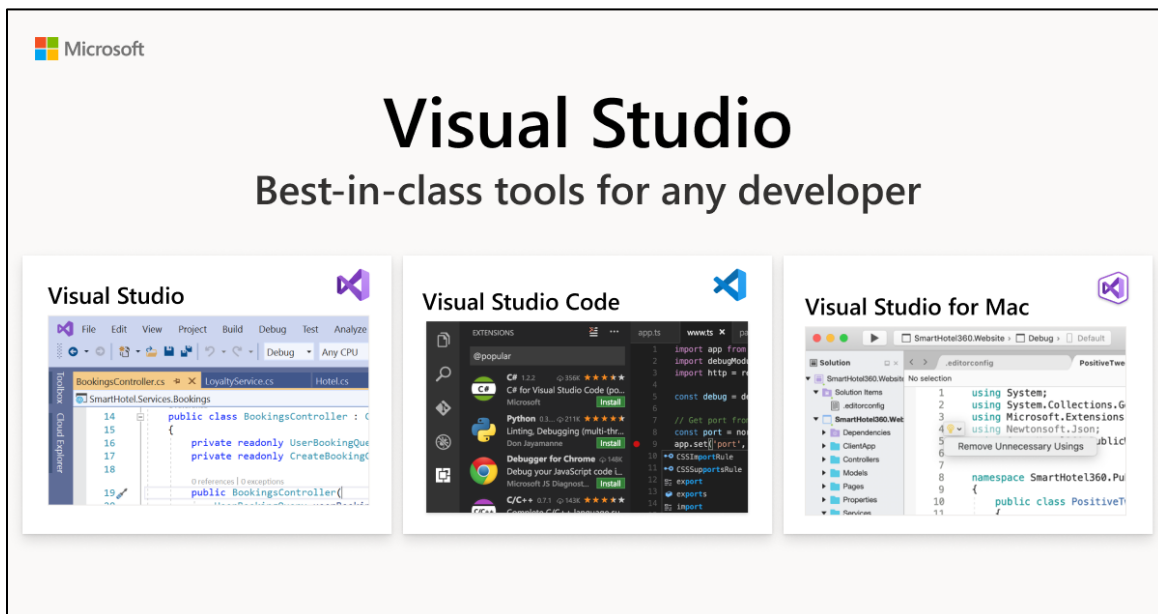
ПИТЕР



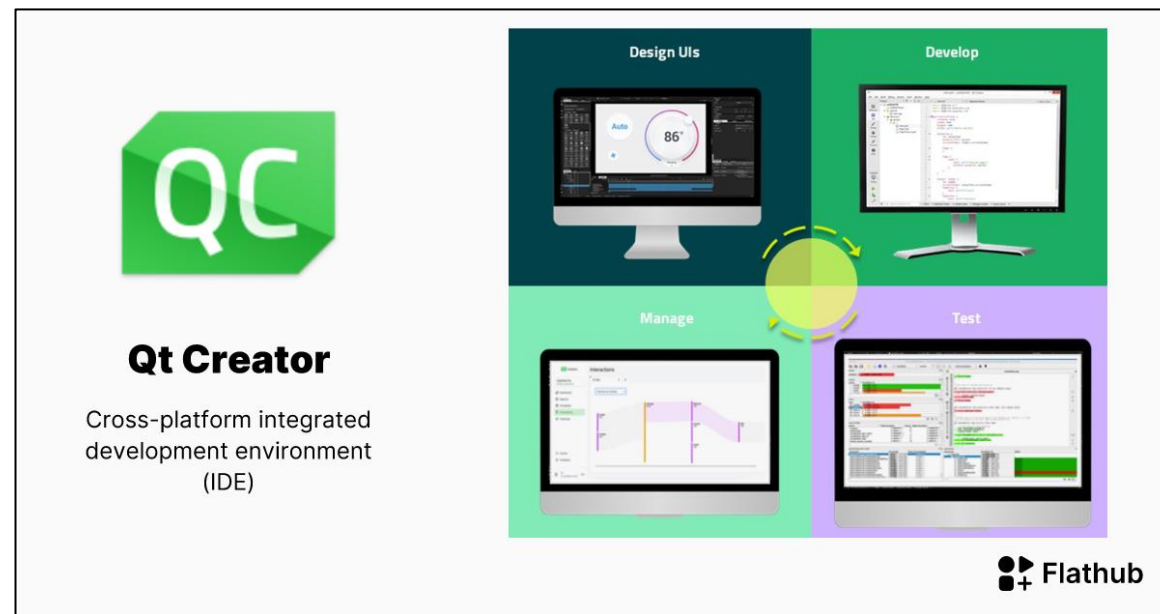
Выбор среды разработки

IDE — Интегрированная среда разработки (англ. Integrated Development Environment) — система программных средств, используемая программистами для разработки программного обеспечения.

Visual Studio Community 2022



Qt Creator

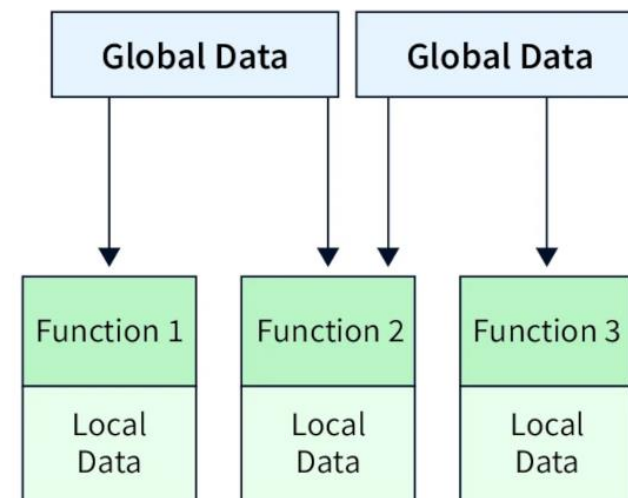


Процедурное и объектно-ориентированное программирование

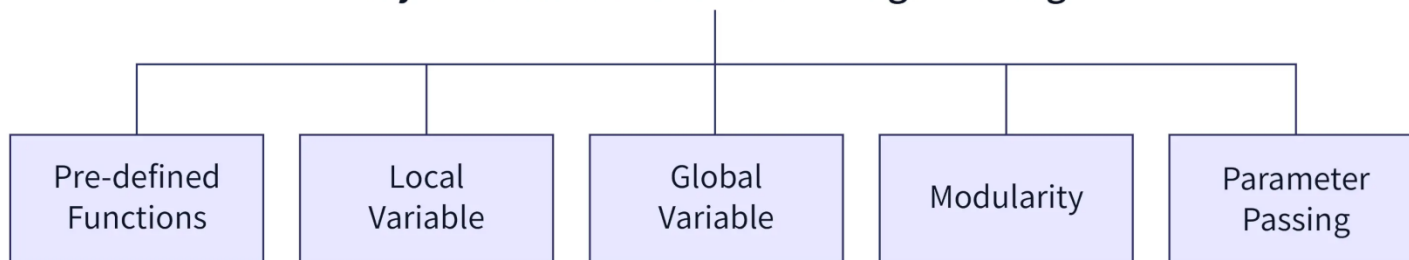
Процедурное программирование (ПП) — парадигма программирования, в которой используется линейный нисходящий подход и рассматривает данные и процедуры как разных объекта.

Недостатки

- Код программы труднее писать, когда используется процедурное программирование
- Процедурный код часто не может быть использован повторно, что может привести к необходимости воссоздания кода, если это необходимо для использования в другом приложении.
- Сложно общаться с реальными объектами
- Важное значение придается операции, а не данным, что может создавать проблемы в некоторых случаях, связанных с данными
- Данные открыты для всей программы, что делает их не очень безопасными

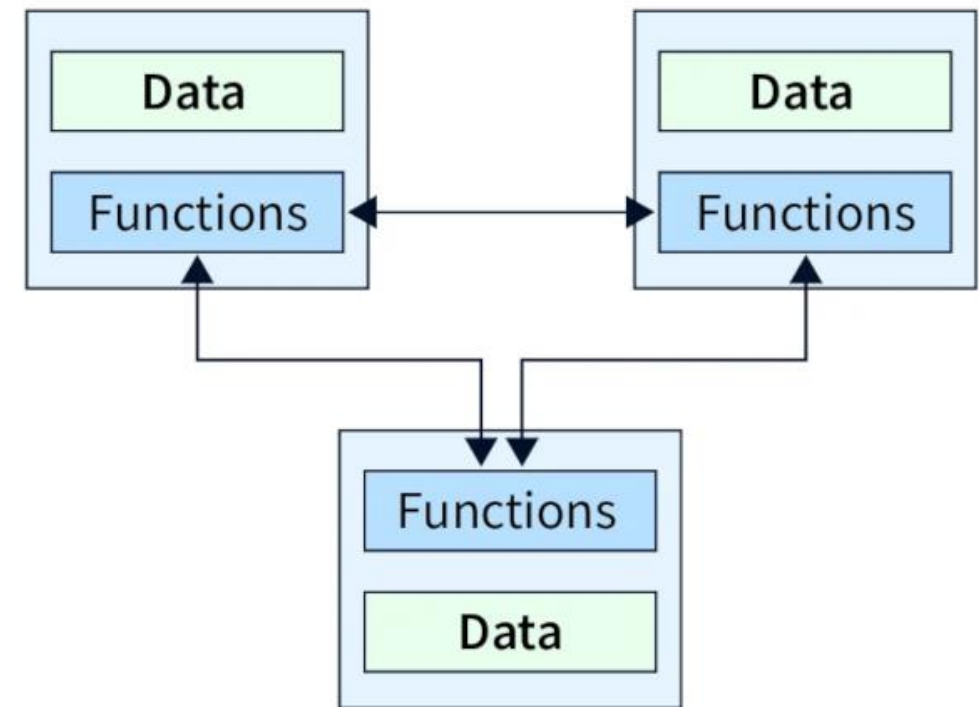


Key Features of Procedural Programming



Процедурное и объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — парадигма программирования, в центральное место занимают объекты и данные, а не процедуры как в ПП. При этом данные являются изменяемыми и могут храниться непосредственно в объектах.



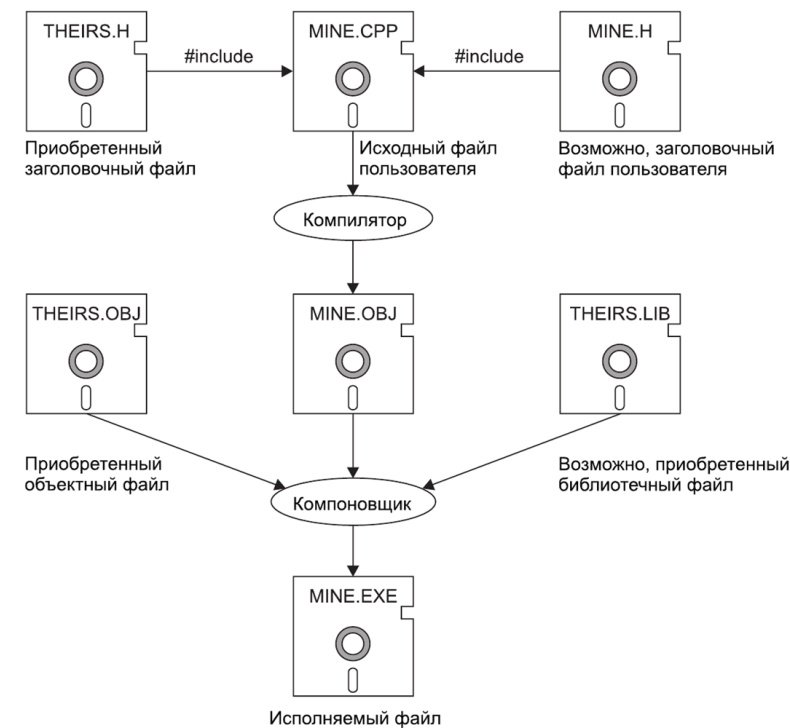
Преимущества

- Благодаря модульности и инкапсуляции, ООП предлагает простоту управления
- ООП имитирует реальный мир, облегчая понимание
- Поскольку объекты являются цельными внутри себя, они могут использоваться в других программах

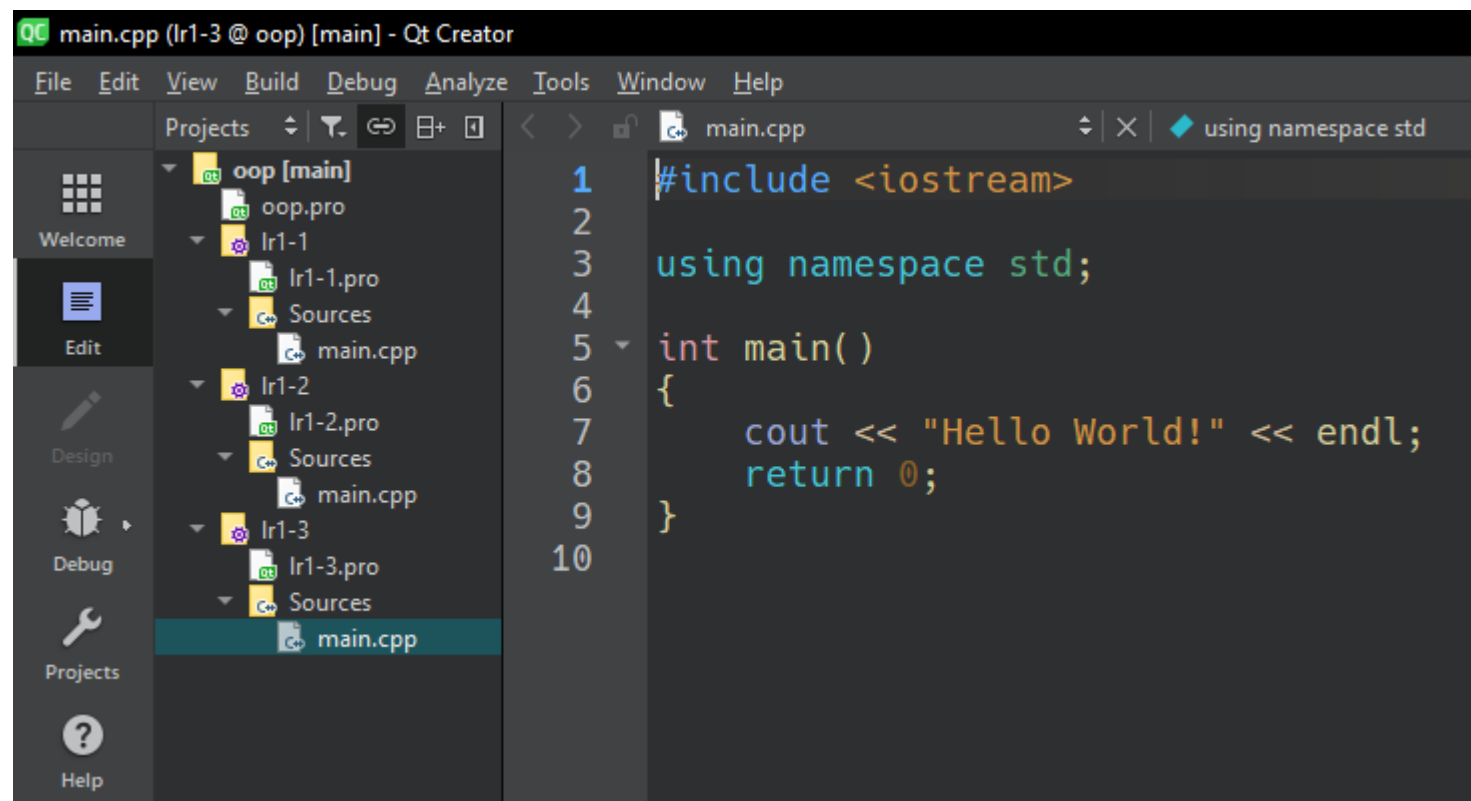
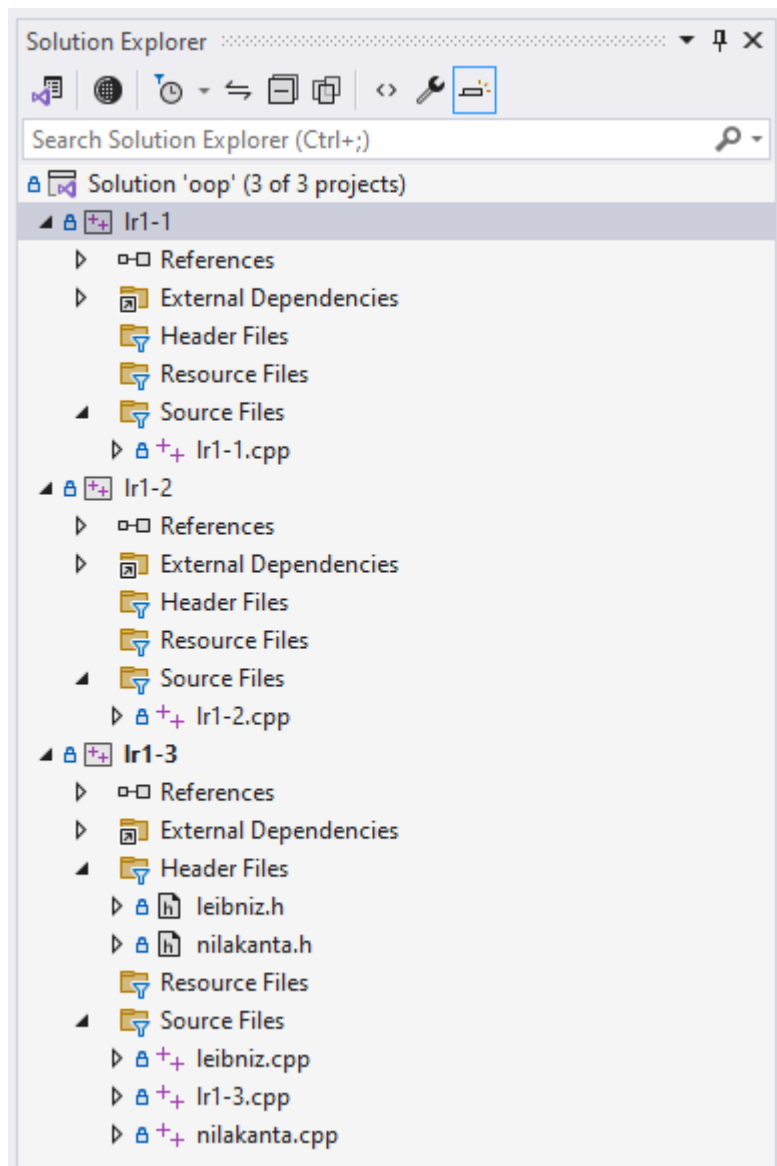
Структура программы. Модульный подход

Модульное программирование — организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.

Модуль можно независимо разрабатывать, тестировать и поддерживать. Каждый модуль выполняет конкретную функцию и имеет определённые входные и выходные данные.



Программное решение (Solution)



```
1 TEMPLATE = subdirs
2
3 SUBDIRS += \
4     lr1-1 \
5     lr1-2 \
6     lr1-3
```


Потоки ввода/вывода в C++

Поток — общее название потока данных. Преимущество потокового ввода/вывода в простоте использования. Поток ввода/вывода может быть связан с файлом, консолью, принтером, строкой. Пользовательские классы также могут поддерживать операции << (вставки) и >> (извлечения) что позволяет работать с этим классами как со стандартными типами.

МОДУЛЬНЫЙ ПОДХОД

```

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
14     return EXIT_SUCCESS;
15 }

```

```

20     std::stringstream stream;
21     stream << std::fixed;
22     stream << std::setprecision(digitsCount) << PI;
23     return stream.str();

```

```

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
22         outFile << "PI = " << setprecision(i) << M_PI << endl;
23     }
24     outFile.close(); // Закрываем файл

```

Использование пространства имен

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.

МОДУЛЬНЫЙ ПОДХОД

```
1  #include <iostream>
2
3  namespace A
4  {
5      int add(int a, int b)
6      {
7          return a + b;
8      }
9  }
10 namespace B
11 {
12     int add(int a, int b)
13     {
14         return a*a + b*b;
15     }
16 }
```

```
18 int main()
19 {
20     int a = 3;
21     int b = 2;
22     std::cout << A::add(a, b) << std::endl;
23     std::cout << B::add(a, b) << std::endl;
24     std::cout << add(a, b) << std::endl;
25 }
```

```
std::cout << add(a, b) << std::endl;
```



identifier "add" is undefined

[Search Online](#)

[Show potential fixes \(Alt+Enter or Ctrl+.\)](#)

```
std::cout << add(a, b) << std::endl;
```



Change 'add' to 'A::add'

Add 'using namespace A'

Change 'add' to 'B::add'

Add 'using namespace B'

...

```
std::cout << B::add(a, b) << std::endl;
```

```
std::cout << add(a, b) << std::endl;
```

```
std::cout << A::add(a, b) << std::endl;
```

```
}
```