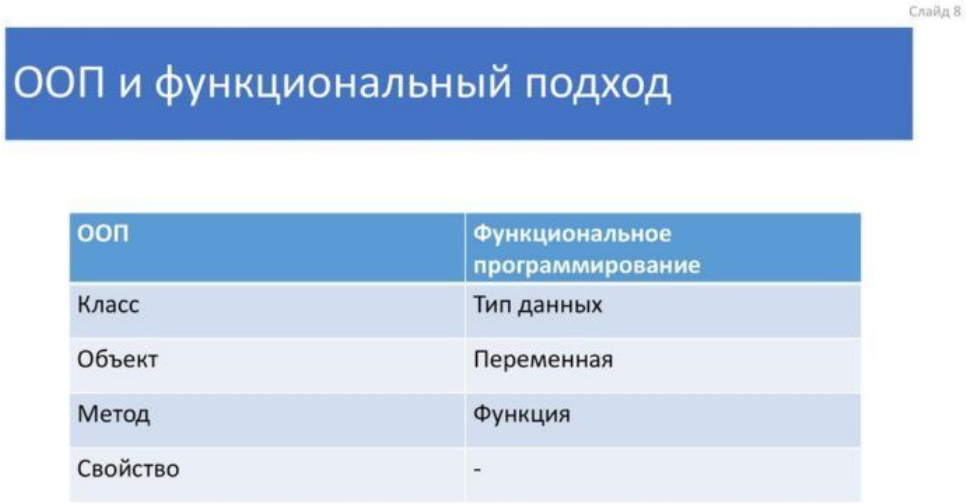
Функциональное программирование

ФП. Описание.

Функциональное программирование (ФП) — это парадигма программирования, в которой программа рассматривается как вычисление математических функций и избегает изменяемого состояния и мутабельных данных.

Функциональное программирование подразумевает использование функций как основного строительного блока программы. В отличие от императивного программирования, где важны последовательность команд и изменение состояний переменных, ФП ориентировано на объявление и применение функций, которые не изменяют своих аргументов и не имеют побочных эффектов.

ООП и ФП.


Слайд 8
ООП и функциональный подход
ООП
Функциональное программирование
Класс
Тип данных
Объект
Переменная
Метод
Функция
Свойство
-

ФП. Концепции.

Функции первого класса (First-Class Functions):

- Функции в ФП рассматриваются как объекты первого класса, что означает, что они могут быть переданы как аргументы другим функциям, возвращены из функций как результат их работы и присвоены переменным.

Чистые функции (Pure Functions):

- Чистая функция возвращает результат, зависящий только от своих аргументов и не имеющий побочных эффектов, таких как изменение глобальных переменных или ввод-вывод.

Неизменяемость (Immutability):

- Данные в ФП считаются неизменяемыми, то есть после создания не могут быть изменены. Это способствует предсказуемости программы и упрощает параллельное программирование.

Рекурсия (Recursion):

- Рекурсивные вызовы являются естественным способом итерации в функциональном программировании, так как они позволяют избегать изменяемых циклов.

Плюсы и минусы ФП;

Преимущества функционального программирования:

1. **Чистые функции и отсутствие побочных эффектов:**
2. **Иммутабельные (неизменяемые) данные:**
3. **Функции высшего порядка и лямбда-выражения:** Функции высшего порядка позволяют передавать функции как аргументы другим функциям или возвращать их как результаты. Это способствует созданию более гибких и модульных программ.
4. **Рекурсия вместо циклов:**
5. **Декларативный стиль программирования:** ФП поддерживает декларативный стиль программирования, который описывает, что должно быть сделано, вместо того, как это должно быть сделано. Это позволяет программистам сосредоточиться на бизнес-логике, а не на деталях реализации.

Недостатки функционального программирования:

1. **Сложность взаимодействия с изменяемым состоянием:** В функциональном программировании изменяемое состояние обычно рассматривается как нечистое источник побочных эффектов. Это может сделать сложным взаимодействие с внешними системами или изменяемыми данными, такими как базы данных или GUI.
2. **Ограниченная поддержка в некоторых областях:** Некоторые задачи, такие как манипуляции с мутабельными данными, могут требовать от программистов сильно настроенные техники или библиотеки, чтобы обойти ограничения функционального программирования.
3. **Подходит не для всех типов задач:** Функциональное программирование не всегда является лучшим выбором для всех типов задач. В некоторых случаях императивный или объектно-ориентированный стиль может быть более естественным и эффективным.
4. **Сложность в понимании рекурсивных алгоритмов:** Рекурсия, хотя и мощный инструмент в функциональном программировании, может быть сложной для понимания и отладки, особенно для новичков.
5. **Производительность:** В некоторых случаях функциональное программирование может не быть самым эффективным с точки зрения производительности из-за большего количества вызовов функций и управления стеком вызовов.

Функциональное или Параллельное

Функциональное программирование:

- **Преимущества:**
  1. **Чистота функций:** Использование чистых функций без побочных эффектов делает код более предсказуемым, легче отлаживаемым и устойчивым к ошибкам.
  2. **Иммутабельные данные:** Неизменяемость данных упрощает параллельное программирование и уменьшает необходимость в синхронизации.
  3. **Декларативный стиль:** Функциональное программирование поддерживает декларативный подход к описанию логики, что упрощает понимание и сопровождение кода.
- **Ограничения:**
  1. **Сложность работы с изменяемыми данными:** Взаимодействие с изменяемыми данными или внешними системами может потребовать дополнительных усилий или настроек.
  2. **Производительность:** Некоторые задачи могут быть менее эффективны в функциональном стиле из-за увеличения числа вызовов функций и управления стеком вызовов.

Параллельное программирование:

- **Преимущества:**
  1. **Улучшенная производительность:** Параллельное программирование позволяет использовать многопоточность или распределенные вычисления для ускорения выполнения задач.
  2. **Эффективное использование ресурсов:** Можно достигнуть более высокой производительности за счет распараллеливания задач, особенно на многоядерных системах или в распределенных средах.
  3. **Решение сложных задач:** Параллельное программирование подходит для решения задач, требующих обработки больших объемов данных или вычислений.
- **Ограничения:**
  1. **Сложность синхронизации:** Работа с разделяемыми ресурсами может привести к проблемам с синхронизацией доступа и потенциальным гонкам данных.
  2. **Сложность отладки:** Параллельное программирование может усложнить отладку из-за потенциальных проблем с конкуренцией и синхронизацией потоков.
  3. **Сложность проектирования:** Не все задачи могут быть легко распараллеливаемыми или требуют сложного проектирования алгоритмов для достижения эффективности.

Выбор подхода;

Выбор между функциональным и параллельным программированием зависит от конкретных требований вашего проекта:

- **Если ваша задача требует чистоты функций, устойчивости к ошибкам и легкости тестирования,** функциональное программирование может быть предпочтительным выбором.
- **Если вам необходимо достичь высокой производительности, эффективно использовать многоядерные ресурсы или обрабатывать большие объемы данных,** параллельное программирование может быть более подходящим вариантом.

ФП. Пример.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

// Пример чистой функции, вычисляющей квадрат числа
int square(int x) {return x * x;}

// Пример использования функции высшего  порядка (функции, принимающей функцию в качестве аргумента )
void apply_to_each(const std::vector<int>& data, int(*func)(int)) {
   for (auto& num : data)
       std::cout << func(num) << " ";
   std::cout << std::endl;
}

int main() {
   std::vector<int> numbers = {1, 2, 3, 4, 5};
   std::cout << "Squares:";
   apply_to_each(numbers, square); // Передаем функцию square как аргумент
   std::cout << "Cubics:";
   apply_to_each(numbers, [](int x) { return x * x * x; });
   return 0;
}
```