**C++:**

1. **Конструкторы и деструкторы. Конструкторы копирования.**

Конструкторы нужны в нескольких случаях. Первый случай: если мы создаём и хотим сразу обратиться к полям этого объекта. В таком случае, если поля не были инициализированы, то в них будет храниться «мусор». Второй случай: чтобы не нарушать принципы инкапсуляции. Поля нашего объекта должны быть private. После создания объекта мы не сможем обратиться к полям этого объекта. В таком случае нам на помощь приходят конструкторы. С помощью конструкторов мы можем проинициализировать поля нашего объекта при его создании. Деструктор нужен, чтобы корректно удалять объекты, которые в полях класса содержат указатель на какой-нибудь тип. Конструктор копирования нужен, чтобы из существующего объекта создать копию этого объекта. У конструктора копирования есть задекларированная конструкция: T(const T&), где T – наш класс.

1. **Создание объектов в стеке и в свободной памяти. Деструкторы.**

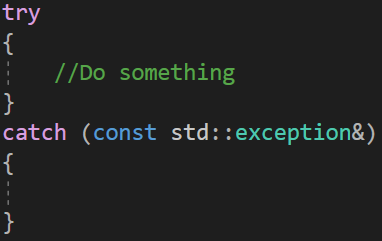
Объекты, которые были созданы с помощью оператора «new», создаются в динамической памяти, остальные – создаются на стеке. Если объект создан в динамической памяти, то есть с помощью оператора «new», то мы должны вызвать его деструктор путём вызова оператора «delete».

1. **Разработка методов классов. Операции класса и их перегрузка.**
2. **Статические компоненты класса. Статические методы.**

Статические метода – это методы, которые относятся не к конкретному объекту, а ко всему классу в целом. В статических методах нельзя использовать поля класса и указатель *this.*

1. **Исключительные ситуации и их обработка.**

Когда объект имеет какие-либо неподходящие параметры или переданные в функцию параметры не являются валидными для данной функции, мы можешь бросить исключение (или ошибку) с помощью оператора *throw*. Тип исключение определяется типом объекта, которого бросает оператор *throw*. Если это исключение не будет обработано, то программа завершает свою работу. Обработать исключение можно с помощью следующей конструкции:



В блоке *try* мы вызваем функции, которые могут бросить исключение. Блок *catch* определяет поведени программы, если в блоке *try* было брошено исключение. Блок *catch* может ловить только исключения одного типа, но может быть записано несколько блоков *catch,* будет выбран самый подходящий по типу блок. Тип указывается как параметр в круглых скобках.

1. **Включение и делегирование.**

Включение – это использование объекта существующего класса, как поля в создаваемом классе. Делегирование – это вызов методов включенного класса, через методы создаваемого класса.

1. **Наследование. Базовый и производный классы. Доступ к наследуемым компонентам.**

Наследование представляет один из ключевых аспектов объектно-ориентированного программирования, который позволяет наследовать функциональность одного класса или базового класса в другом - производном классе. Наследование нужно, чтобы инкапсулировать общие методы для нескольких классов в один родительский класс. Доступ к наследуемым компонентам зависит от типа доступа в родительском классе и типа доступа наследования. Есть 3 основных типа доступа: **private**, **public**, **protected.**

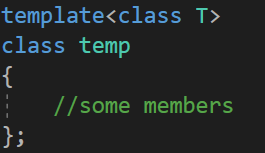
1. **Виртуальные функции и их вызов. Абстрактные классы.**

Если мы создаём дочерний класс через указатель на родительский и определяем метод, сигнатура которого есть в родительском классе, то при вызове этого метода, будет вызван метод родительского класса. Чтобы избежать подобной ситуации, мы должны использовать ключевое слово *virtual.* Это слово помечает функцию как виртуальную, и теперь, если мы определим в дочернем классе функции с сигнатурой родительского класса, то мы переопределим существующую функцию вместо создания новой. И теперь при вызове этой функции через указатель на родительский класс, у нас будет корректно вызываться функция. Абстрактный класс – это класс, который имеет хотя бы одну чисто виртуальную функцию. Чисто виртуальная функция не имеет реализации в родительском классе, её реализацией занимаются дочерние класса, пока эта функция не будет реализована, классы будут считаться абстрактными. Сигнатура чисто виртуальной функции:

возвращаемый\_тип имя\_функции(параеметры) = 0;

1. **Классы-шаблоны. Общие операции.**

Класс-шаблон – это класс, объекты которого могу работать с разными типа. Класс-шаблон определяется следующей конструкцией:



Про общие операции хз что сказать.

1. **Библиотека STL. Контейнеры.**

STL – стандартная библиотека шаблонов. В этой библиотеке собраны контейнеры, функции и алгоритмы для работы с контейнеры. В STL есть много контейнеров: std::vector, std::deque, std::set, std::map и тд.

1. **Итераторы STL.**

Итераторы нужны для доступа к элементам непоследовательных контейнеров, так как элементы в этих контейнерах располагаются не последовательно, следовательно к ним мы не может обратиться через обычный указатель или по индексу.

1. **Общие операции для контейнеров. Функция emplace.**

Для всех контейнеров есть общие методы:

1. begin() – возвращает итератор на 1й элемент
2. end() – возвращает итератор на элемент, который находится за последни
3. emptu() – true, если контейнер пуст
4. size() – количество элементов в контейнере

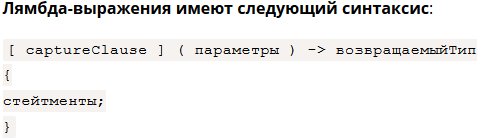
Метод emplace() передаёт параметры напрямую в конструктор объекта и создаёт его на месте.

1. **Последовательные контейнеры. Выбор типа контейнера.**

Последовательные контейнеры – это контейнеры элементы в которых хранятся последовательно и доступ к ним можно осуществить по индексу то есть за O(1). Если нам нужен быстрый доступ к элементу контейнера, то лучше выбирать последовательные контейнеры.

1. **Алгоритмы для последовательных контейнеров**
2. **Функциональные объекты и лямбда-выражения.**

Функциональные объекты – это объекты класса у которого перегружен оператор «operator()». Лямбда-выражение в программировании позволяет определить анонимную функцию внутри другой функции. Возможность сделать функцию вложенной является очень важным преимуществом, так как позволяет избегать как захламления пространства имен лишними объектами, так и определить функцию как можно ближе к месту её первого использования.



Поля captureClause и параметры могут быть пустыми, если они не требуются программисту. Поле возвращаемыйТип является опциональным, и, если его нет, то будет использоваться вывод типа с помощью ключевого слова auto.

1. **Ассоциативные контейнеры. Пары. Словарь.**

В ассоциативных контейнерах элементы хранятся в парах ключ-значение и сам контейнер отсортирован, что позволяет делать поиск в среднем за O(log2(N)).

1. **«Умные» указатели.**

Умные указатели нужны для того, чтобы не беспокоиться за динамическое выделение памяти. «Умный» указатель в своём деструкторе удаляет указатель который он хранит. Из этого можно сделать вывод, что указатели нельзя копировать, их можно только перемещать.