Warning

This document is not an OOP Exam Program. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an Exam Program.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

МФТИ, ФПМИ, кафедра алгоритмов и технологий программирования
Объектно-ориентированное программирование, 1 курс, сезон 2020-2021
(лектор: Мещерин И.С. a.k.a. mesyarik)

***

Обновления

15.06 Опубликована первая версия программы экзамена для основного потока. Скорее всего, опубликованные списки вопросов существенно уже не поменяются, но незначительные модификации пока еще возможны.

***

Правила экзамена

TODO

***

Списки вопросов

Теорминимум (вопросы на “уд”) (общие для обоих потоков)

Берегите наш язык, наш прекрасный язык C++ – это клад, это достояние, переданное нам нашими предшественниками! Обращайтесь почтительно с этим могущественным орудием; в руках умелых оно в состоянии совершать чудеса.
(И. С. Тургенев)

Нам дан во владение самый богатый, меткий, могучий и поистине волшебный язык C++.
(К. Г. Паустовский)

1. Понятие компилятора. Разница между компилируемыми и интерпретируемыми языками. Примеры компиляторов C++. Запуск компилятора из командной строки. Понятие ошибок компиляции (CE), основные виды ошибок компиляции (лексические, синтаксические, семантические) с примерами.
2. Понятия ошибок времени выполнения (RE) и неопределенного поведения (UB) с примерами. Различные виды RE, отличие RE от CE. Почему не любую RE компилятор может предвидеть? Почему не любое UB приводит к RE?
3. Основные типы: int, long long, size_t, float, double, long double, char, bool, std::string. Допустимые операции над ними. Ввод-вывод с помощью cin и cout.
4. Что делают, что возвращают и в каком порядке вычисляются следующие операторы над основными типами: арифметические и побитовые операторы, логические операторы, операторы присваивания, префиксный и постфиксный инкремент, тернарный оператор, оператор “запятая”.
5. Синтаксис использования конструкций if… else, switch, for, while, do... while. Использование команд break, continue, return.
6. Разница между объявлением и определением. Синтаксис объявления и определения переменных, функций, классов и структур, алиасов для типов (с помощью слова using). Правило одного определения (ODR) для функций и для переменных, объяснение его смысла.
7. Разница между стековой (автоматической) и динамической памятью. Динамическое выделение памяти с помощью оператора new (в стандартной форме). Освобождение динамической памяти. Проблема утечек памяти. Проблема двойного удаления.
8. Понятие области видимости и времени жизни объекта. Конфликты имен переменных, замещение менее локального имени более локальным именем. Пример ситуации неоднозначности при обращении к переменной.
9. Указатели и допустимые операции над ними. Сходства и различия между указателями и массивами.
10. Ссылки. Объяснение концепции. Отличия от указателей. Особенности инициализации ссылок, присваивания ссылкам. Передача аргументов по ссылке и по значению. Проблема висячих ссылок, пример ее возникновения.
11. Константы, константные указатели и указатели на константу. Константные и неконстантные операции. Константные ссылки, их особенности, отличия от обычных ссылок.
12. Неявное приведение типов. Явное приведение типов с помощью оператора static_cast.
13. Понятие перегрузки функций. Что является и что не является частью сигнатуры функции. Основные правила выбора наиболее подходящей функции: частное лучше общего, точное соответствие лучше приведения типа. Пример неоднозначности при обращении к функции.

14. Понятие класса, полей и методов класса, модификаторы доступа public и private, отличие класса от структуры. Применение операторов точка и стрелочка. Применение двойного двоеточия. Применение ключевого слова this. Константные и неконстантные методы.
15. Понятие конструкторов, деструкторов, пример их правильного определения для класса String. В каких контекстах использования от класса требуется наличие конструктора по умолчанию, в каких - конструктора копирования?
16. Понятие перегрузки операторов, синтаксис перегрузки оператора присваивания. В каких контекстах использования от класса требуется наличие оператора присваивания?
17. Условия генерации компилятором конструкторов и оператора присваивания. Использование слов default и delete для их запрета или принудительной генерации компилятором.
18. Списки инициализации в конструкторах. Обязательная инициализация полей перед входом в конструктор. Особенности инициализации полей, являющихся ссылками и константами.
19. Понятие наследования, синтаксис объявления наследника, разница между private и public наследованием.
20. Инициализация родителя наследником и запрет преобразования в обратную сторону. То же самое со ссылками и указателями.
21. Понятие полиморфизма. Понятие виртуальных функций. Разница в поведении виртуальных и невиртуальных функций при наследовании. Предназначение и использование ключевых слов override и final.
22. Понятия шаблонов, инстанцирования шаблонов, специализации шаблонов. Синтаксис определения шаблонов классов и шаблонов функций. Синтаксис определения специализации шаблонов.
23. Понятие исключений. Синтаксис использования throw и конструкции try… catch. Что является и что не является исключениями. Пример ошибки времени выполнения, не являющейся исключением.
24. Наличие/отсутствие и алгоритмическая сложность каждой из операций [], find, insert, erase, push_back, push_front в каждом из контейнеров: vector, list, forward_list, deque, map, set, unordered_map, unordered_set
25. Основные методы контейнера vector и их правильное применение: [], push_back, pop_back, size, resize, capacity, reserve.
26. Основные методы контейнера map и их правильное применение: [], find, insert, erase, lower_bound, upper_bound.
27. Виды итераторов и операции, поддерживаемые каждым из этих видов: forward, bidirectional, random access. Какой вид итераторов поддерживается каждым из контейнеров, упомянутых выше?
28. Использование итераторов для прохода по любому из контейнеров от начала до конца. Использование range-based for.
29. Использование стандартных алгоритмов STL над контейнерами: std::sort, std:find, std::find_if, std::max_element, std::min_element, std::reverse, std::binary_search, std::lower_bound, std::upper_bound, в том числе с произвольной функцией сравнения.
30. Мотивировка move-семантики. Какую проблему она решает? Пример ситуации, когда уместно использовать std::move. Пример правильной реализации move-конструктора и move-оператора присваивания для класса String.
31. Мотивировка умных указателей. Какую проблему они решают? Базовый синтаксис использования shared_ptr (как создать, как пользоваться в простейшем случае).

Вопросы на “хор” (основной поток)

C++ - язык, на котором Бог написал Вселенную.
-- Галилей.

Если Бог есть, Он - великий C++ разработчик.
-- Пол Дирак.

1. Параметры компилятора -Wall, -Wextra и -Werror, объяснение их действия. Примеры предупреждений компилятора, не являющихся ошибками с точки зрения языка. Параметр компилятора -O. Пример неожиданного поведения компилятора при UB с переполнением int и включенном параметре -O. Объяснение этого поведения.
2. Разница между понятиями operator precedence и order of evaluation. Понятие unspecified behaviour, примеры. Несколько примеров выражений, вычисление которых является unspecified behaviour (с объяснением). Примеры выражений, вычисление которых является undefined behaviour.
3. Приведения типов: C-style cast, static_cast, const_cast и reinterpret_cast. В каких ситуациях (кроме наследования) каждый из этих операторов приводит к ошибке компиляции, в каких - к неопределенному поведению, а в каких он уместен?
4. Статические переменные, особенности и примеры их использования. Статические поля и методы классов, статические константы, их особенности и примеры использования. Понятие статической памяти, разница между статической и стековой памятью.
5. Приватные и публичные поля и методы. Особенности выбора версии функции при перегрузке в случае выбора между приватными и публичными версиями (с примерами). Функции-друзья и классы-друзья, синтаксис объявления, пример использования: перегрузка операторов ввода-вывода. Свойства отношения дружбы в C++.
6. Ключевое слово explicit, два возможных контекста его использования. Перегрузка операторов приведения типа для классов, пример. Определение пользовательских литеральных суффиксов для классов, пример.
7. Модификатор доступа protected для полей и методов. Приватное и публичное наследование. Разница между наследованием классов и структур. Разница между видимостью и доступностью. Видимость и доступность различных членов родителя в теле наследника при приватном и публичном наследовании, явное обращение к методам родителя, использование using.
8. Размещение объектов в памяти при наследовании. Порядок вызова конструкторов и деструкторов, а также инициализации полей при наследовании. Наследование конструкторов родителя с помощью using. Обращение к конструкторам родителя из конструкторов наследника. Множественное наследование. Проблема ромбовидного наследования, размещение объекта в памяти при таком наследовании.
9. Шаблонные функции, синтаксис их объявления. Перегрузка и специализация шаблонных функций, разница между этими понятиями (на примере). Эвристические правила разрешения перегрузки: “частное лучше общего”, “точное соответствие лучше приведения типа”. Примеры разрешения перегрузки между шаблонными функциями.
10. Специализация шаблонов. Синтаксис объявления специализации для функций и классов. Разница между перегрузкой и специализацией шаблонных функций на примере. Частичная и полная специализация шаблонов классов. Примеры: реализация hash для нестандартного типа, реализация is_same, реализация remove_reference.

11. Простейшие compile-time вычисления с помощью шаблонной рекурсии. Вычисление чисел Фибоначчи в compile-time с помощью шаблонной рекурсии. Проверка на простоту числа N за O(N) в compile-time с помощью шаблонной рекурсии.
12. Виртуальные функции. Объяснение логики выбора версии метода у наследника в случаях, когда функции виртуальные и когда нет. Логика выбора версий в случае, когда присутствуют как виртуальные, так и невиртуальные методы со слегка отличающимися типами параметров, разной константностью, разной приватностью и т. п.. Логика выбора версий в случае многоуровневого наследования и в случае множественного наследования.
13. Чисто виртуальные функции, синтаксис определения, примеры использования. Понятие абстрактных классов. Виртуальный деструктор, особенности его определения и пример проблемы, возникающей в случае его отсутствия. Понятие RTTI, оператор typeid и особенности его использования.
14. Приведения типов при наследовании: static_cast, dynamic_cast и reinterpret_cast. Особенности использования каждого из этих операторов. Пример ситуации, когда все три этих оператора ведут себя по-разному. Способы приведения “вверх”, “вниз”, и “вбок” по иерархии наследования. Особенности использования данных операторов при приватном наследовании, множественном наследовании.
15. Исключения. Преимущества и недостатки использования исключений. Особенности копирования исключений при их создании и при поимке. Разница между throw без параметров и throw с параметром. Правила приведения типов при поимке исключений. Правила выбора подходящей секции catch.
16. Проблема исключений в конструкторах. Поведение программы при выбросе исключения из конструктора. Идиома RAII. Проблема исключений в деструкторах. Функция uncaught_exception, ее предназначение. Спецификации исключений: спецификатор noexcept и оператор noexcept, синтаксис и пример применения. Исключения в списках инициализации конструкторов, function-try блоки.
17. Понятие аллокатора. Зачем нужны аллокаторы? Реализация основных методов std::allocator. Оператор placement new, его синтаксис использования и отличие от обычного new. Структура std::allocator_traits, ее предназначение. Структура rebind, ее предназначение.
18. Понятие итератора. Пример реализации итераторов для любого из стандартных контейнеров (на ваш выбор). Разница между константными и неконстантными итераторами. Реализация константных итераторов без дублирования кода относительно обычных итераторов, применение метафункции std::conditional.
19. Функции std::advance и std::distance, пример их применения. Реализация функции advance с правильной поддержкой разных видов итераторов, два способа такой реализации: с помощью перегрузки функций (старый способ до C++17) и с помощью if constexpr (новый способ, начиная с C++17).
20. Формальное определение lvalue и rvalue. Объяснение идеи, стоящей за этим определением. Объяснение, как по выражению понять его вид value. Примеры, когда rvalue-выражение допускает присваивание и когда lvalue-выражение не допускает присваивания. Ссылочные квалификаторы (ref-qualifiers), синтаксис и пример использования.
21. Rvalue-ссылки, их сходства и различия с обычными ссылками. Правила инициализации rvalue-ссылок (в том числе константных). Примеры передачи параметров по rvalue-ссылке, перегрузка между обычными и rvalue-ссылками, логика выбора версии функции в случае такой перегрузки. Правила сворачивания ссылок (reference collapsing).
22. Идея универсальных ссылок. Реализация функции std::move. Объяснение действия этой функции. Объяснение, почему принимаемый и возвращаемый типы именно такие.
23. Проблема прямой передачи. Предназначение функции emplace_back, ее преимущество перед push_back (объяснение на примере). Правильное использование функции std::forward (без реализации) для реализации механизма perfect forwarding.

24. Контейнер vector: внутреннее устройство с алгоритмической точки зрения, методы reserve, capacity, shrink_to fit, их действие. Проблема реализации выделения памяти в методе reserve. Реализация методов reserve, resize и push_back с использованием аллокатора (здесь можно без поддержки exception safety). Правила инвалидации итераторов в vector.
25. Контейнер vector<bool>. Отличие от обычного vector. Класс BoolReference, его реализация. Реализация метода [] в vector<bool>.
26. Контейнеры list и forward_list, их внутреннее устройство, примерная реализация основных методов (конструкторы, деструкторы, операторы присваивания, insert, erase, push/pop_back, push/pop_front). Правила инвалидации итераторов в list и forward_list.
27. Контейнеры map и set, их внутреннее устройство с алгоритмической точки зрения: что хранится внутри, какие алгоритмы и структуры данных используются для реализации методов. Понятие компараторов, пример использования нестандартных компараторов. Асимптотика обхода map итератором с объяснением, как он (обход) работает. Правила инвалидации итераторов в map.
28. Мотивировка умных указателей. Класс unique_ptr, его идея и реализация основных методов. Особенности поведения unique_ptr при попытке его скопировать. Функция make_unique, ее реализация, пример использования, преимущество перед обычным конструктором unique_ptr.
29. Идея реализации класса shared_ptr (без поддержки weak_ptr, без поддержки нестандартных аллокаторов и deleter'ов): конструкторы, деструктор, операторы присваивания. Функция make_shared, ее реализация, пример использования, преимущества перед обычным конструктором shared_ptr.
30. Проблемы, приводящие к идее автоматического вывода типов (auto) при объявлениях. Примеры, когда auto необходимо и когда неуместно. Ключевое слово decltype. Особенности поведения auto и decltype от ссылочных типов. Особенности auto в возвращаемом типе функции. Особенности поведения decltype от lvalue, xvalue и prvalue. Конструкция decltype(auto) и пример, когда она нужна.
31. Лямбда-функции и лямбда-выражения, их синтаксис. Пример использования в стандартных алгоритмах, пример использования в качестве компаратора для map. Списки захвата в лямбда-выражениях, их синтаксис, пример использования. Синтаксис захвата по ссылке и по значению.
32. Идиома SFINAE. Простейший пример выбора из двух функций с использованием SFINAE (когда более подходящую функцию пришлось отбросить). Структура enable_if, ее реализация и пример правильного использования.
33. Константные выражения, ключевое слово constexpr. Отличие constexpr от const. Контексты, в которых требуются константные выражения. constexpr-функции и ограничения на их содержимое. Особенности throw в constexpr-функциях, особенности создания объектов в constexpr-функциях. Ключевое слово static_assert и его применение. Пример ситуации, когда static_assert неприменим.

Вопросы на “отл” (основной поток)

Думаю, не ошибусь, если скажу, что никто не понимает move-семантику.
-- Ричард Фейнман

Если тебя реализация type_traits не испугала, значит, ты ничего в ней не понял.
-- Пол Дирак

1. Защищенное наследование. Логика видимости и доступности родителей и прародителей, их членов при двухуровневом наследовании с различными модификаторами доступа. Логика запрета и разрешения обращений к членам родителей при использовании слова friend в разных местах с различными комбинациями приватного и защищенного наследования.
2. Понятие полиморфных объектов. Таблица виртуальных функций для полиморфного объекта, ее содержимое. Размещение в памяти объектов, у которых есть виртуальные функции. Объяснение, как за счет vtable происходит выбор нужной версии функции, а также за счет чего работают dynamic_cast и typeid. Разница между статическим и динамическим выбором версии функции с точки зрения реализации на низком уровне.
3. Виртуальное наследование как решение проблемы ромбовидного наследования. Особенности размещения объектов в памяти при виртуальном наследовании. Почему при наличии виртуальных предков размер объекта увеличивается? Поведение компилятора в случае комбинации виртуального и обычного наследования одного и того же предка. Особенности видимости и доступности родительских методов в этих случаях.
4. Шаблонные шаблонные параметры, синтаксис использования. Пример: класс Stack на основе шаблонного контейнера. Шаблоны с переменным количеством аргументов (variadic templates). Синтаксис использования, пример: функция print. Пакеты аргументов и пакеты параметров, их распаковка. Реализация структуры is_homogeneous. Оператор “sizeof…”.
5. Зависимые имена. Пример неоднозначности между declaration’ом и expression’ом. Применение ключевого слова typename для устранения неоднозначности с зависимым именем. Применение слова template для решения аналогичной проблемы с зависимыми именами шаблонов (иллюстрация примером).
6. Реализуйте класс std::insert_iterator и функцию std::inserter. Объясните, как работает ваша реализация. Приведите пример использования этого класса и этой функции.
7. Реализуйте классы std::istream_iterator и std::ostream_iterator. Объясните, как работает ваша реализация. Приведите пример использования этих классов.
8. Особенности перегрузки операторов new и delete. Разница между оператором new и функцией operator new. Разновидности оператора new, placement new, nothrow new, их правильная перегрузка. Особенности вызова нестандартного operator delete. В каком случае компилятор способен вызвать нестандартный operator delete самостоятельно?
9. Реализуйте стековый аллокатор. Это такой аллокатор, который заводит большой массив в стековой памяти и всю память берет из него, ни разу не обращаясь к new (тем самым давая выигрыш во времени для контейнера, построенного на нем). Он делает это в предположении, что количество запрошенной памяти никогда не превзойдет размер этого массива.
10. Понятие allocator-aware контейнеров. Параметры аллокаторов propagate_on_container_copy_assignment /move_assignment / swap, их предназначение и использование. Функция select_on_container_copy_construction, ее предназначение и использование. Реализация allocator-awareness на примере контейнера vector: правильная работа с аллокатором при копировании/перемещении/присваивании контейнера.

11. Реализация функции std::forward. Объяснение ее действия. Объяснение, почему принимаемый и возвращаемый типы именно такие. Почему в качестве принимаемого типа нельзя написать T&&?
12. Return Value Optimization (RVO), ее идея и пример, когда она возникает. Пример, когда небольшая модификация возвращаемого выражения приводит к исчезновению RVO (оператор += в BigInteger). Примеры, когда надо и когда не надо писать std::move после return. Понятие copy elision, идея этой оптимизации и пример ее возникновения. Условия, при которых она точно происходит, а также при которых может произойти, но не обязана.
13. Контейнер deque: внутреннее устройство с алгоритмической точки зрения, сходства и различия с vector. Что хранится внутри deque, как происходит его расширение? Объяснение асимптотики работы методов push_back/push_front и [], за счет чего она достигается. Как устроены итераторы в deque, что хранят итераторы? Разница между инвалидацией итераторов и инвалидацией ссылок, правила инвалидации итераторов и ссылок в deque с объяснением, почему они такие.
14. Контейнеры unordered_map и unordered_set, их внутреннее устройство с алгоритмической точки зрения: что хранится внутри, какие алгоритмы и структуры данных используются для реализации методов. Что алгоритмически происходит при вставке, при удалении, при rehash? Пример использования нестандартного компаратора, нестандартной хэш-функции в unordered_map. Асимптотика обхода unordered_map итератором с объяснением, как он (обход) работает. Правила инвалидации итераторов и ссылок в unordered_map.
15. Гарантии безопасности контейнеров относительно исключений. Проблемы с гарантиями exception-safety у map и unordered_map. Реализация exception safety на примере vector (с произвольным аллокатором): реализация безопасных конструкторов, операторов присваивания, метода push_back.
16. Проблема циклических shared_ptr. Класс weak_ptr как решение этой проблемы. Реализация основных методов weak_ptr: конструкторы, деструктор, операторы присваивания, методы expired() и lock(). Модификация реализации shared_ptr для поддержки weak_ptr.
17. Класс enable_shared_from_this, описание проблемы, которую он решает. Реализация этого класса.
18. Лямбда-функции. Объекты анонимного типа, сгенерированные из лямбда-функций (замыкания), их внутреннее устройство. Правила генерации компилятором полей и методов этих объектов. Особенности копирования и присваивания этих объектов. Особенности захвата полей класса и указателя this в лямбда-функции. Опасность захвата по умолчанию.
19. Класс std::function, его основные методы и примеры использования. Опишите идею внутреннего устройства std::function: каким образом достигается возможность по ходу работы подменять хранящийся в ней функциональный объект? (Принимается любая работающая идея.)
20. Реализуйте проверку числа N на простоту в compile-time с асимптотикой O(sqrt(n)) с помощью шаблонной рекурсии (без использования constexpr, а также математических функций стандартной библиотеки).
21. Реализуйте метафункцию has_method, позволяющую проверить, присутствует ли у класса T метод с заранее заданным названием от данных типов аргументов. Объясните, как работает ваша реализация. Для чего в ней нужна функция declval и что эта функция из себя представляет? Для чего и в каких местах STL используется метафункция has_method?
22. Реализуйте метафункции is_constructible, is_copy_constructible, is_move_constructible. Объясните, как работают ваши реализации. Для чего в них нужна функция declval и что она из себя представляет?
23. Реализуйте метафункцию is_nothrow_move_constructible и с помощью нее реализуйте функцию move_if_noexcept. Объясните, как работает ваша реализация. Почему нельзя наивно выразить is_nothrow_move_constructible как is_move_constructible_v<...> && noexcept(...)?
24. Реализуйте метафункцию is_base_of<T, U>, позволяющую проверить, является ли класс U наследником класса T (в том числе приватным). Объясните, как работает ваша реализация.

25. Реализуйте метафункцию is_polymorphic<T>, которая позволяет проверить, является ли тип T полиморфным.
26. Реализуйте структуру make_index_sequence<N>, которая бы представляла из себя index_sequence<%последовательность от 0 до N-1%>.
27. Объясните понятие рефлексии в программировании. Реализуйте функцию detect_fields_count<S>, позволяющую для данной структуры S, допускающей агрегатную инициализацию, узнать количество полей в этой структуре.

Доп. вопросы на отл(10)

C++ — единственный всемирный язык, его не надо переводить, на нем душа говорит с душою.
-- Бертольд Ауэрбах

C++ показывает человеку те возможности величия, которые есть в его душе.
-- Ралф Уолдо Эмерсон

TODO