

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Московский физико-технический институт  
(национальный исследовательский университет)»**

**УТВЕРЖДЕНО**

**Директор физтех-школы  
прикладной математики и  
информатики  
А.М. Райгородский**

**Рабочая программа дисциплины (модуля)**

**по дисциплине:** Программирование на C++  
**по направлению:** Прикладная математика и информатика  
**профиль подготовки:** Искусственный интеллект и большие данные  
Сетевое обучение  
кафедра алгоритмов и технологий программирования  
**курс:** 1  
**квалификация:** бакалавр

Семестр, формы промежуточной аттестации: 1 (осенний) - Дифференцированный зачет

Аудиторных часов: 60 всего, в том числе:

лекции: 30 час.

семинары: 30 час.

лабораторные занятия: 0 час.

Самостоятельная работа: 84 час.

Всего часов: 144, всего зач. ед.: 4

Программу составил: И.С. Мещерин, ассистент

Программа обсуждена на заседании кафедры алгоритмов и технологий программирования  
11.06.2021

## Аннотация

Принципы объектно-ориентированного подхода. Объектно-ориентированный анализ и проектирование. Основы объектно-ориентированного программирования. Обобщенные классы и методы. Строго типизированные источники. Управление взаимодействием объектов.

### 1. Цели и задачи

#### Цель дисциплины

- Сформировать представление о разнообразных вычислительных задачах в теории графов и об асимптотических сложностях их решений;
- дать теоретические и практические знания об алгоритмах и структурах данных теории графов с доказательством корректности их работы, о методах оценки сложности алгоритмов.

#### Задачи дисциплины

- Научить формулировать задачи в терминах изученных теорий, выбирать подходящий алгоритм для поставленной задачи;
- научить разрабатывать комбинации алгоритмов для решения поставленных задач, оценивать сложности алгоритмов, их модификаций и комбинаций, в том числе с помощью амортизационного анализа, выбирать подходящие структуры данных для поставленных задач, реализовывать алгоритмы в обобщенной форме на языке программирования C++.

### 2. Перечень формируемых компетенций

Освоение дисциплины направлено на формирование следующих компетенций:

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-2 Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения	ОПК-2.1. Использует знания основных положений и концепций в области программирования, архитектуру языков программирования, основную терминологию и базовые алгоритмы, основные требования информационной безопасности для практического применения
	ОПК-2.2. Анализирует типовые языки программирования, составляет программы
	ОПК-2.3 Применяет на практике опыт решения задач с использованием базовых алгоритмов, анализа типов коммуникаций и интеграции различных типов программного обеспечения

### 3. Перечень планируемых результатов обучения по дисциплине (модулю)

В результате освоения дисциплины обучающиеся должны

#### знать:

- Алгоритмы на графах и структуры данных, связанные с ними;
- оценки сложности стандартных алгоритмов;
- стандартные алгоритмы на графах и используемые структуры данных, подходы к модификации классических алгоритмов;
- разнообразные классические задачи в теории графов и асимптотические сложности их решений.

#### уметь:

- Формулировать задачи в терминах изученных теорий, выбирать подходящий алгоритм для поставленной задачи;
- разрабатывать комбинации алгоритмов для решения поставленной задачи;

- оценивать сложности алгоритмов, их модификаций и комбинаций, в том числе с помощью амортизационного анализа;
- выбирать подходящие структуры данных для конкретной задачи;
- реализовывать алгоритм в обобщенной форме на языке программирования c++;
- реализовывать стандартные алгоритмы на графах и структуры данных на языке программирования C++.

#### **владеть:**

- Методами декомпозиции задач в области информационных технологий и построения единого решения с использованием изученных алгоритмов;
- методами оценки сложности алгоритмов, их модификаций и комбинаций.

#### **4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий**

##### **4.1. Разделы дисциплины (модуля) и трудоемкости по видам учебных занятий**

№	Тема (раздел) дисциплины	Трудоемкость по видам учебных занятий, включая самостоятельную работу, час.			
		Лекции	Семинары	Лаборат. работы	Самост. работа
1	Введение в язык	2	1		5
2	Модификаторы типов	2	2		6
3	Перегрузка операторов	2	2		5
4	Наследование (inheritance)	2	2		5
5	Шаблоны (templates)	2	2		5
6	Исключения (exceptions)	2	2		5
7	Аллокатеры (allocators)	2	2		6
8	Итераторы	2	2		5
9	Move-семантика и rvalue-ссылки	1	2		6
10	Умные указатели	2	2		5
11	Вывод типов	2	2		6
12	Шаблонное метапрограммирование и SFINAE	2	2		5
13	Функциональные объекты и лямбда-функции	2	2		5
14	Некоторые особые полезные типы	2	2		5
15	Введение в ООП	2	1		5
16	Контейнеры (containers)	1	2		5
Итого часов		30	30		84
Подготовка к экзамену		0 час.			
Общая трудоёмкость		144 час., 4 зач.ед.			

##### **4.2. Содержание дисциплины (модуля), структурированное по темам (разделам)**

Семестр: 1 (Осенний)

#### **1. Введение в язык**

Общие слова: место языка C++ среди современных языков, актуальные версии этого языка, ключевые люди, связанные с этим языком, официальный стандарт языка

Структура программы, функция main, понятие области видимости (scope). Ключевые слова. Ввод-вывод (cin, cout).

Объявления (declarations). Идентификаторы. Фундаментальные типы (int, long, long long, float, double, long double, char, bool, модификаторы signed и unsigned). Размеры этих типов, основные операции над ними, неявные преобразования типов между собой. Литералы, литеральные суффиксы для основных типов. Объявления функций, разница между объявлением и определением, one definition rule.

Выражения (expressions). Операторы. Арифметические операторы. Побитовые операторы. Логические операторы, особенности их работы. Оператор присваивания и операторы составного присваивания, особенности его работы. Понятие lvalue и rvalue в C++03. Инкремент и декремент, отличие префиксной версии от постфиксной. Операторы сравнения. Тернарный оператор. Оператор “запятая”. Оператор sizeof.

Инструкции (statements). Конструкции if...else, for, while, do...while, switch, их синтаксис, правила работы. Инструкции break, continue, return, их действие. Инструкция goto и метки.

Понятия ошибки компиляции, ошибки времени выполнения (runtime error), неопределенного поведения (undefined behaviour), отличия между ними, примеры. Виды ошибок компиляции: лексические, синтаксические, семантические. Понятие segmentation fault и stack overflow.

## 2. Модификаторы типов

Указатели, операции над ними. Операция взятия адреса. Автоматическая память (стек). Массивы, операция [] (квадратные скобки), ее принцип работы. Указатель на void и его особенности.

Функции. Перегрузка функций, правила разрешения перегрузки (общая схема, без деталей). Функции с аргументами по умолчанию. Функции с неуказанным количеством аргументов.

Указатели на функции, операции над ними, их особенности.

Динамическая память. Операторы new и new[], их использование (в стандартной форме). Операторы delete и delete[], их использование (в стандартной форме). Проблема утечек памяти. Проблема двойного удаления.

Передача аргументов по значению и по указателю, первая версия функции swap. Дилемма с присваиванием (создавать новое название или копию?). Идея ссылок (references). Отличия ссылок от указателей, правила работы со ссылками, вторая версия функции swap. Проблема, связанная со ссылкой на локальную переменную (“битые ссылки”).

Идея констант, ключевое слово const. Понятие константных и неконстантных операций, особенности работы с константами. Константные и неконстантные ссылки. Константные указатели и указатели на константу. Разрешенные и запрещенные присваивания между всеми вышеупомянутыми типами.

Виды приведений типов: static\_cast, reinterpret\_cast, const\_cast и C-style cast, их особенности, примеры применения и примеры, когда они не сработают.

## 3. Перегрузка операторов

Общая идея перегрузки операторов. Перегрузка арифметических операторов на примере класса BigInteger: бинарные операторы, составные присваивания с ними, правильное выражение одного через другое. Проблема с корректностью выражений вида “x+y=5;”. Проблема в случае левого операнда - не объекта класса (выражения вида “5+x”). Перегрузка операторов << и >> на примере потокового ввода-вывода.

Перегрузка операторов сравнения, правильное выражение одних сравнений через другие.

Перегрузка инкремента и декремента (префиксного и постфиксного).

Перегрузка оператора [] (квадратные скобки). Правильное соблюдение константности при перегрузке оператора [].

Перегрузка оператора “круглые скобки”. Понятие функтора и функционального класса, компаратора. Пример использования в стандартных алгоритмах.

Особенности перегрузки операторов “логическое И”, “логическое ИЛИ” и “запятая”.

Особенности перегрузки операторов “унарная звездочка”, “унарный амперсанд” и “стрелочка”.

Перегрузка операторов приведения типа. Еще одно применение ключевого слова `explicit`.

#### 4. Наследование (inheritance)

Объявление наследования. Модификатор доступа `protected`. Разница между приватным, публичным и защищенным наследованием. Разница между наследованием классов и структур. Поиск имен при наследовании. Соккрытие имен наследником. Явный вызов методов родителя у наследника. Использование `::` и `using`. Проблемы с видимостью названий родителей и их полей у потомков в случае двухуровневого наследования, где первый уровень - приватное наследование. Правила действия слова `friend` в этих случаях.

Порядок вызова конструкторов и деструкторов при наследовании. Проблема с инициализацией родителей при определении конструктора наследника, вновь применение списков инициализации. Правила размещения объектов классов-наследников в памяти.

Множественное наследование, неоднозначности при нем, проблема ромбовидного наследования. Примеры разрешения неоднозначности с помощью приведений типов и оператора `::`, комбинации всего этого с приватным наследованием, сдвиги указателей.

Виртуальное наследование. Особенности комбинации виртуального и неvirtуального наследования.

Приведение типов между родителем и наследником: срезка при копировании, приведение указателей, приведение ссылок. Особенности `static_cast`, `reinterpret_cast` между родителями и наследниками (а также указателями или ссылками на них). Оператор `dynamic_cast`, его отличие от `static_cast`.

Виртуальные функции, их общая идея и отличие от неvirtуальных. Особенности размещения в памяти классов с виртуальными функциями, понятие полиморфизма. Полиморфные классы. Понятие о таблице виртуальных функций.

Виртуальный деструктор и его предназначение.

Абстрактные классы и “чисто виртуальные” (`pure virtual`) функции, их особенности. Чисто виртуальный деструктор. Ошибка “`pure virtual function call`” и ее возникновение.

Ключевые слова `override` и `final` при наследовании, их предназначение.

Механизм RTTI. Оператор `typeid` и динамическое определение типа объекта. Класс `std::type_info`.

Проблема с вызовом виртуальных функций в конструкторах. Проблема с аргументами по умолчанию в виртуальных функциях.

Empty base optimization, примеры.

#### 5. Шаблоны (templates)

Мотивировка и общая идея шаблонов. Шаблоны классов, шаблоны функций, синтаксис объявления, примеры использования, связь шаблонов и полиморфизма, статический полиморфизм.

Специализации шаблонов, принцип “частное предпочтительнее общего” применительно к шаблонам. Частичные и полные специализации. Принцип “лучше точное соответствие, чем приведение типа”. Разница между специализацией и перегрузкой для шаблонных функций.

Правила выбора компилятором кандидатов на специализацию и на перегрузку.

Ключевое слово `typedef`, его предназначение. Шаблонные `typedef`ы, использование слова `using`.

Проблема с обращением к `typedef`ам внутри шаблонных классов. Применение ключевого слова `typename` для решения этой проблемы.

Примеры реализации простейших `type_traits` с помощью шаблонных структур и `typedef`ов внутри них: `remove_const`, `remove_reference`.

Правила вывода типов для шаблонов. Отбрасывание ссылок при выводе типа. Разбор случаев со ссылками и константами.

Параметры шаблонов, не являющиеся типами (пример: массив константной длины).  
Параметры шаблонов, являющиеся шаблонами (“template template parameters”).  
Шаблоны с переменным количеством аргументов (variadic templates). Синтаксис использования. “Откусывание” шаблонных аргументов по одному. Оператор “sizeof...”.  
Функциональные классы и функциональные объекты (функторы), схема использования.  
Компараторы. Пример: компаратор в `std::sort`. Стандартные компараторы (`std::less`, `std::greater`, `std::equal` и т. п.), их реализация.  
Curiously Recurring Template Pattern (CRTP).

## 6. Исключения (exceptions)

Общая идея, мотивировка использования исключений, оператор `throw` и конструкция `try...catch`. Примеры стандартных операторов, генерирующих исключения.  
Разница между исключениями и ошибками времени выполнения. Ошибки, не являющиеся исключениями, и исключения, не являющиеся ошибками.  
Правила ловли и повторного бросания исключений, приведения типов при ловле исключений.  
Ловля всех исключений. Правила выбора блока `catch` компилятором в случае, когда подходят разные блоки.  
Копирование при бросании и ловле исключений, исключения и наследование. Особенности перехвата исключений по значению и по ссылке, по ссылке на базовый класс.  
Спецификации исключений в старом стиле и их проблемы, `unexpected exceptions` (неожиданные исключения). спецификации исключений в стиле C++11, оператор и спецификатор `noexcept`. Условный `noexcept`.  
Исключения в конструкторах и проблема утечки памяти при исключениях.  
Исключения в деструкторах, функция `uncaught_exception`, функции `terminate` и `set_terminate`.  
Гарантии безопасности при исключениях: базовая и строгая.  
Function-try блоки, их особенности.

## 7. Аллокаторы (allocators)

`Placement new`, его синтаксис, действие и отличие от обычного `new`.  
Разница между оператором `new` и функцией `operator new`. Более подробный разбор действия оператора `new`. Перегрузка `new` для отдельных классов. Перегрузка глобального `new`.  
Определение `new` с произвольными параметрами. То же самое для операторов `delete` и `delete[]`.  
Пример, когда компилятор неявно вызывает `delete` с нестандартными параметрами. Поведение `delete` для полиморфных объектов.  
`nothrow` оператор `new`, его синтаксис и особенности.  
Разбор поведения `new` в случае нехватки памяти. Функция `new_handler`, функции `set_new_handler` и `get_new_handler`.  
Понятие аллокатора. Класс `std::allocator`, его основные методы (`allocate`, `deallocate`, `construct`, `destroy`) и их примерная реализация. Особенности реализации конструкторов и оператора присваивания у стандартного аллокатора.  
Класс `std::allocator_traits`, его предназначение и основные методы.  
Пример нестандартного аллокатора (`PoolAllocator`), идея реализации его методов. Проблемы с конструктором копирования и оператором присваивания.

## 8. Итераторы

Общая идея итераторов. Использование итераторов у стандартных контейнеров.  
Виды итераторов: `input`, `output`, `forward`, `bidirectional`, `random access`. Операции, поддерживаемые каждым видом итераторов. Виды итераторов у стандартных контейнеров.  
Константные и `reverse`-итераторы. Методы `cbegin`, `cend`, `rbegin`, `rend`, `crbegin`, `crend` у контейнеров. Реализация класса `std::reverse_iterator`, метод `base`.

Класс `std::iterator`, его предназначение. Класс `std::iterator_traits`, его предназначение. Пример ситуации, когда он необходим (обращение к `value_type`).

Функции `std::distance` и `std::advance`. Различие в поведении этих функций для разных видов итераторов, реализация этого различия.

Стандартная библиотека алгоритмов, использование стандартных алгоритмов над контейнерами с итераторами. Итераторы для вставок: классы `std::insert_iterator`, `std::back_insert_iterator`, их предназначение, реализация. Функции `std::inserter`, `std::back_inserter`, их реализация.

Правила инвалидации итераторов в стандартных контейнерах. Безопасные и небезопасные операции в контейнерах с точки зрения инвалидации итераторов.

## 9. Move-семантика и rvalue-ссылки

Проблемы, приводящие к идее move-семантики: неэффективный `swap`, неэффективный `push_back`, `emplace_back`, `construct`.

Применение магической функции `std::move`. Решение проблемы со `swap`.

Понятие move-конструктора и move-assignment оператора, их реализация, генерация компилятором, “правило пяти”.

Реализация функции `std::move`. Дилемма: что принять в качестве параметра?

Понятие rvalue-ссылок. Особенности инициализации rvalue-ссылок, разрешенные и запрещенные присваивания между ссылками (включая проблемы с константностью). Решение проблемы с `push_back`.

Понятия `glvalue`, `lvalue`, `rvalue`, `prvalue` и `xvalue`. Связи между ними. Примеры выражений, являющихся тем или иным видом `value`.

Понятие универсальных ссылок, отличие их от rvalue-ссылок. Правила вывода типа шаблонов в случае универсальных ссылок, решение проблемы с типом параметра функции `move`. Правила сворачивания ссылок (reference collapsing).

Проблема прямой передачи (perfect forwarding). Функция `std::forward` и ее применение. Решение проблем с `emplace_back` и `construct`.

Реализация `std::forward`, ее обсуждение. Почему типы у принимаемого параметра и возвращаемого значения именно такие?

Новая проблема с `push_back`: безопасность относительно исключений. Функция `std::move_if_noexcept`, решение проблемы с ее помощью.

Return Value Optimization, условия ее возникновения. Примеры, когда RVO точно произойдет и когда может не произойти. Примеры, когда имеет и когда не имеет смысл писать `return std::move(x)` вместо `return x`. Copy Elision, примеры.

Ссылочные квалификаторы. Решение проблемы с запретом оператора присваивания для rvalue у кастомных типов.

Примеры типов, для которых прямая передача работает некорректно.

Особенности поведения универсальных ссылок при разрешении перегрузки. Феномен “поглощения” универсальными ссылками обычных ссылок.

## 10. Умные указатели

Идея и мотивировка умных указателей.

Класс `std::auto_ptr` как первая неудачная попытка реализовать идею.

Класс `std::weak_ptr`, его концепция. Особенности его конструкторов, деструктора и операторов присваивания. Методы `*` и `->`. Специализация `weak_ptr` для массивов.

Класс `std::shared_ptr`, его концепция. Идея реализации счетчика ссылок. Реализация методов.

Потенциальная проблема, связанная с прямым вызовом `new`. Функции `std::make_unique` и `std::make_shared` как способ избежать прямого вызова `new`. Реализация этих функций.

Функция `std::allocate_shared`, ее предназначение и реализация. Исправление реализации конструктора `shared_ptr` для этого.

Проблема закольцованности указателей. Класс `std::weak_ptr` как решение этой проблемы. Реализация методов этого класса. Проблема с реализацией метода `expired()`, модификация класса `shared_ptr` для правильной работы с этим. Кастомные `deleter`'ы для умных указателей, схема использования. Более правильная реализация деструкторов `unique_ptr` и `shared_ptr`. Класс `std::enable_shared_from_this`, его предназначение и реализация. Еще одна модификация конструктора `shared_ptr` (проверка на наследника `enable_shared_from_this`).

## 11. Вывод типов

Проблема с длинными названиями типов. Проблема с возможными ошибками в написании точных названий типов. Ключевое слово `auto` как решение этих проблем. Правила вывода типов для `auto`. Особый случай с типом `initializer_list`. Особенности при `auto&&`. `auto` в качестве возвращаемого типа функции. Ключевое слово `decltype`, правила вывода типов для него. Особенности поведения `decltype` от выражений (случаи `lvalue`, `xvalue`, `rvalue`). Пример с `decltype((x))`. Особенности взятия `decltype` от тернарного оператора. Конструкция `decltype(auto)`. Пример: обертка над обращением к контейнеру по индексу. Трюк для вывода названий выведенных типов на экран (намеренное провоцирование ошибок компиляции).

## 12. Шаблонное метапрограммирование и SFINAE

Имитация `if` через шаблоны. Имитация `for` через шаблоны. Примеры: вычисление чисел Фибоначчи, проверка простоты числа, вывод чисел от 1 до 1000 с помощью шаблонов. Ключевое слово `constexpr` для функций и для переменных. Отличие `constexpr` от `const`. Требования к `constexpr`-функциям. `type_traits`. Структуры `std::is_const` (`pointer`, `reference` etc.), `std::add_const` (`pointer`, `reference` etc.), `std::remove_const` (`pointer`, `reference`, `extent` etc.), их реализации. Структуры `std::is_same`, `std::true_type`, `std::false_type`, `std::conjunction`, `std::disjunction`, `std::conditional`, `std::rank`, их реализации. Идиома SFINAE. Общая идея. Простейший пример: структура `std::enable_if`, ее реализация и применение. Структура `std::is_class`, ее реализация (без реализации `std::is_union`). Реализация метода `std::allocator_traits::construct` (проверка, определен ли у аллокатора метод `construct`). Проблема: невозможность написать `T()` для произвольного типа `T`. Функция `std::declval`, ее особенности. Решение предыдущей проблемы с ее помощью. Структуры `std::is_constructible`, `std::is_convertible`, `std::is_copy_constructible`, `std::is_move_constructible` etc. Их реализации. Реализация `std::is_nothrow_move_constructible`. Реализация `std::move_if_noexcept` через `std::is_nothrow_move_constructible`. Почему принимаемый и возвращаемый типы именно такие? Понятие неполных типов (`incomplete types`). Новая проблема с `declval` (что возвращать), решение проблемы с помощью `rvalue`-ссылки. Реализация `std::is_base_of`. Пример применения: проверка `is_base_of<enable_shared_from_this<T>>` в конструкторе `shared_ptr<T>`. Реализация `std::common_type`.

## 13. Функциональные объекты и лямбда-функции

Лямбда-функции: мотивировка, простой пример (нестандартный компаратор в `std::sort`). Списки захвата в лямбда-функциях. Захват по ссылке и по значению. Особенности захвата `this`. Захват с присваиванием и перемещающий захват в C++14. Слово `mutable` применительно к лямбда-функциям. Явное указание возвращаемого значения.



Захват по умолчанию и проблемы, которые он потенциально порождает. Пример с классом и методом `getFunction()` в нем.

Обобщенные лямбда-функции в C++14. Применение `auto` и `decltype` в лямбда-функциях.

Класс `std::function`, его предназначение и схема использования. Реализация `std::function`.

Функция `std::bind`, ее предназначение и схема использования (без реализации). Placeholder'ы.

Класс `std::is_invocable`. Класс `std::invoke_result` и функция `std::invoke`. (Все без реализации.)

#### 14. Некоторые особые полезные типы

Юнионы (`union`), их основная идея. Отличия от классов и структур. Инициализация полей юниона, активный член юниона и его изменение на другой.

Класс `std::variant`, его предназначение и основные методы. Примерное описание реализации этого класса.

Класс `std::any`, его предназначение и основные методы. Примерная реализация этого класса.

Класс `std::optional`, его предназначение и основные методы.

Неудачная попытка создать `vector<int&>`. Класс `std::reference_wrapper` для решения этой и других проблем. Примерная реализация этого класса.

#### 15. Введение в ООП

Идея ООП. Понятия класса и структуры, членов класса. Поля и методы, понятие инкапсуляции. Модификаторы доступа.

Конструкторы и деструкторы. Конструктор по умолчанию. Перегрузка конструкторов.

Конструктор копирования, его сигнатура и схема реализации. Пример, когда необходим нетривиальный конструктор копирования и оператор присваивания. Правила генерации компилятором конструкторов. Ключевые слова `default` и `delete` в контексте определения функций-членов.

Операторы “точка” и “стрелочка”. Ключевое слово `this` и пример использования.

Оператор присваивания, его сигнатура и схема реализации. “Правило трех”.

Проблема с инициализацией констант и ссылок. Решение: списки инициализации в конструкторах.

Ключевое слово `explicit`. Пример с конструктором `String(int n)`.

Константные и неконстантные методы, примеры.

Ключевое слово `mutable`, пример применения.

Понятие дружественных функций и классов, ключевое слово `friend`.

Проблема вызова конструкторов из других конструкторов. Решение: делегирующие конструкторы.

Статические поля и методы, пример. Локальные статические переменные.

Указатели на члены и указатели на методы. Синтаксис объявления, пример использования.

Операторы “точка со звездочкой” и “стрелочка со звездочкой”.

#### 16. Контейнеры (containers)

Общие слова о контейнерах. Класс `std::vector`, его предназначение, идея реализации, основные методы и их алгоритмическая сложность.

Поля класса `std::vector`. Реализация конструкторов, деструкторов, оператора присваивания с правильным обращением к аллокатору.

Реализация метода `push_back` с правильным обращением к аллокатору.

Реализация оператора `[]` для константных и неконстантных `vector`. Разница между `[]` и методом `at()`.

Метод `emplace_back`, его реализация и отличие от `push_back`.

Методы `size()`, `resize()`, `capacity()`, `reserve()` и `shrink_to_fit()`.

Особенности работы с аллокатором при копировании вектора. Метод `select_on_container_copy_construction`.

Вопросы на понимание: чему равно `sizeof(v)`, где `v` - вектор, и что произойдет при вызове `delete[] &(v[0])`?

Класс `vector<bool>` и его отличие от обычного `vector`, преимущества и недостатки. Внутренний класс `BoolProxy`. Особенности реализации оператора `[]` и оператора присваивания для `vector<bool>` по сравнению с обычным `vector`.

Класс `std::deque`, основные методы и их алгоритмическая сложность. Разница между `deque` и `vector`: методы `deque`, отсутствующие у `vector`; методы `vector`, отсутствующие у `deque`. Адаптеры над контейнерами: `std::stack`, `std::queue` и `std::priority_queue`, их реализации. Компараторы в `priority_queue` и ее специфичные методы.

Класс `std::list`, основные методы и их алгоритмическая сложность. Идея реализации `list`'а. Вставка и удаление из произвольного места. Специфичные для `list`'а методы: `splice`, `sort`, `merge`, `reverse`. Особенности работы `list`'а с аллокатором, метод `rebind` у аллокаторов. Класс `std::forward_list`, его отличия от обычного `list`.

Ассоциативные контейнеры. Класс `std::map`, его предназначение, идея реализации. Описание шаблонных параметров класса `map`. Класс `std::pair` и функция `std::make_pair`. Основные методы `map`'а и их алгоритмическая сложность. Способы поиска в `map`'е. Способы вставки в `map`, особенности работы оператора `[]`. Способы удаления из `map`'а. Классы `std::set`, `std::multimap` и `std::multiset`, их предназначение, отличия от `std::map`.

Класс `std::unordered_map`, сходства и различия с обычным `std::map`. Основные методы и их алгоритмическая сложность. Особые для `unordered_map` шаблонные параметры: `Hasher`, `Equal`. Класс `std::hash` и его специализации. Особые для `unordered_map` методы: `bucket_count`, `load_factor`, `rehash`. Классы `std::unordered_set`, `std::unordered_multimap`, их идея, отличие от `unordered_map`.

## **5. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)**

Учебная аудитория, оснащенная компьютером и мультимедийным оборудованием (проектор, звуковая система).

## **6.Перечень рекомендуемой литературы**

Основная литература  
не предусмотрено

Дополнительная литература  
не предусмотрено

## **7. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)**

Не используются

## **8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень необходимого программного обеспечения и информационных справочных систем (при необходимости)**

ПО для разработки и отладки программ на языке программирования C++.

## **9. Методические указания для обучающихся по освоению дисциплины (модуля)**

Выполнение домашнего задания является одной общей работой студента. Необходимо требовать от студентов качественного и понятного программного кода на ряду и в равной степени с правильной работой программы.

## ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

**по направлению:** Прикладная математика и информатика  
**профиль подготовки:** Искусственный интеллект и большие данные  
 Сетевое обучение  
 кафедра алгоритмов и технологий программирования  
**курс:** 1  
**квалификация:** бакалавр

Семестр, формы промежуточной аттестации: 1 (осенний) - Дифференцированный зачет

**Разработчик:** И.С. Мещерин, ассистент

### 1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-2 Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения	ОПК-2.1. Использует знания основных положений и концепций в области программирования, архитектуру языков программирования, основную терминологию и базовые алгоритмы, основные требования информационной безопасности для практического применения
	ОПК-2.2. Анализирует типовые языки программирования, составляет программы
	ОПК-2.3 Применяет на практике опыт решения задач с использованием базовых алгоритмов, анализа типов коммуникаций и интеграции различных типов программного обеспечения

### 2. Показатели оценивания компетенций

В результате изучения дисциплины «Программирование на C++» обучающийся должен:

**знать:**

- Алгоритмы на графах и структуры данных, связанные с ними;
- оценки сложности стандартных алгоритмов;
- стандартные алгоритмы на графах и используемые структуры данных, подходы к модификации классических алгоритмов;
- разнообразные классические задачи в теории графов и асимптотические сложности их решений.

**уметь:**

- Формулировать задачи в терминах изученных теорий, выбирать подходящий алгоритм для поставленной задачи;
- разрабатывать комбинации алгоритмов для решения поставленной задачи;
- оценивать сложности алгоритмов, их модификаций и комбинаций, в том числе с помощью амортизационного анализа;

- выбирать подходящие структуры данных для конкретной задачи;
- реализовывать алгоритм в обобщенной форме на языке программирования C++;
- реализовывать стандартные алгоритмы на графах и структуры данных на языке программирования C++.

#### **владеть:**

- Методами декомпозиции задач в области информационных технологий и построения единого решения с использованием изученных алгоритмов;
- методами оценки сложности алгоритмов, их модификаций и комбинаций.

### **3. Перечень типовых (примерных) вопросов, заданий, тем для подготовки к текущему контролю**

Виды ошибок компиляции: лексические, синтаксические, семантические.

Понятие segmentation fault и stack overflow.

Реализация метода push\_back с правильным обращением к аллокатору.

Идея и мотивировка умных указателей.

Лямбда-функции: мотивировка, простой пример (нестандартный компаратор в std::sort).

### **4. Перечень типовых (примерных) вопросов и тем для проведения промежуточной аттестации обучающихся**

#### **Теорминимум**

Объясните понятия compilation error, runtime error и undefined behaviour. Приведите по паре примеров того, другого и третьего.

Что такое выражения, операторы? Для чего предназначены и что по стандарту делают следующие операторы: тернарный оператор, оператор “запятая”, унарная звездочка, унарный амперсанд, операторы “точка” и “стрелочка”, двойное двоеточие, префиксный и постфиксный инкремент, бинарные & и &&, операторы простого и составного присваивания, операторы << и >>?

Что такое стековая память и динамическая память? Что такое stack overflow? Зачем нужен оператор new? Что такое утечки памяти? Что такое сборка мусора?

Объясните идею ссылок (references). Для чего они нужны, чем они отличаются от указателей?

Что такое “передача аргументов по ссылке и по значению”? Как в C++03 реализовать функцию swap?

Что такое класс, структура, в чем разница между ними? Что такое поля, методы, модификаторы доступа, инкапсуляция?

Что такое конструкторы, деструкторы, для чего они нужны, каков синтаксис их определения?

Те же вопросы про конструктор копирования и оператор присваивания.

Что такое перегрузка функций? Что такое перегрузка операторов? Приведите примеры операторов, которые можно и нельзя перегружать. Приведите пример перегрузки хоть какого-нибудь оператора (напишите сигнатуру его перегрузки).

Что такое наследование? В чем разница между приватным и публичным наследованием?

Что такое виртуальные функции, чисто виртуальные функции, абстрактные классы? Что такое виртуальное наследование?

Что такое полиморфизм? Приведите пример полиморфизма в C++.

Что такое шаблоны? Что такое инстанцирование шаблонов, специализация шаблонов?

Приведите пример каких-нибудь шаблонов из STL, обладающих специализацией.

Что такое исключения? Как пользоваться механизмом обработки исключений? Как бросить, поймать исключение? Приведите какой-нибудь пример.

Рассмотрим контейнеры std::vector, std::list, std::deque. Каковы основные операции, предоставляемые ими, и скорость работы этих операций?

Рассмотрим контейнеры std::map, std::set, std::unordered\_map, std::unordered\_set. Каковы основные операции, предоставляемые ими, и скорость работы этих операций?

Что такое итераторы? Какие виды итераторов существуют? Зачем они вообще нужны? Какие итераторы поддерживает каждый из контейнеров, упомянутых в предыдущих двух вопросах? Что такое компараторы? Что такое функциональные объекты? Приведите хоть один пример. Для чего нужна move-семантика? Расскажите в общих чертах, что это такое. Как правильно реализовать функцию `swap` в C++11?

Что такое умные указатели? Для решения каких проблем они нужны? Приведите пример использования.

Что такое аллокаторы? Какова общая идея класса `std::allocator`, как и для чего он используется?

Что такое лямбда-выражения, каков их синтаксис? Приведите хоть один пример использования.

Напишите вычисление n-го числа Фибоначчи (в пределах `long long`) на этапе компиляции, не используя слова `constexpr`.

Что такое SFINAE? Приведите хотя бы один пример (на уровне идей, можно без реализации).

## Вопросы к экзамену

### Часть 1.

Расскажите об указателях и ссылках в C++03. В чем их идея, для чего они нужны? В чем разница между указателями и ссылками? Какие операции они поддерживают, чем их можно, а чем нельзя инициализировать? Что такое константные указатели, указатели на константу, константные ссылки, в чем проявляется разница между ними и обычными указателями и ссылками? Расскажите о проблеме “битых ссылок”, приведите пример.

Расскажите о том, что такое классы, объекты, члены классов, поля, методы, модификаторы доступа, инкапсуляция. Расскажите про конструкторы и деструкторы, операторы присваивания, “правило трех” и “правило пяти”, про генерацию компилятором этих методов. Что такое списки инициализации в конструкторах и делегирующие конструкторы, зачем они нужны? Покажите на примере класса `String`, как правильно определять вышеперечисленные методы.

Расскажите о том, что такое классы, объекты, члены классов, поля, методы, модификаторы доступа, инкапсуляция. Расскажите о ключевых словах `static`, `explicit`, `mutable`, `friend` с примерами ситуаций, когда их следует применять.

Расскажите о возможностях перегрузки операторов в C++. Покажите на примере реализации длинной арифметики (класс `BigInteger`), как правильно перегружать бинарный плюс, унарный и бинарный минус, операторы составного присваивания с плюсом и минусом, операторы сравнения, префиксный и постфиксный инкремент.

Расскажите о наследовании в C++. В чем разница между приватным и публичным наследованием? Каковы правила видимости полей и методов родителя в теле наследника, как явно обратиться из наследника к полям и методам родителя? Каковы правила видимости родителей и их полей и методов при двухуровневом наследовании, как действует слово `friend` в этих ситуациях? Что такое “срезка при копировании”? Какие неявные конверсии разрешены между родителями и наследниками, в т.ч. ссылками (указателями) на них?

Как размещаются в памяти объекты классов-наследников? В каком порядке вызываются конструкторы и деструкторы при наследовании, как инициализировать поля родителя при определении конструктора наследника? Что такое множественное наследование, в чем заключается проблема ромбовидного наследования, что такое виртуальное наследование? Что означает и как возникает ошибка “ambiguous base”? Что означает и как возникает “warning: inaccessible base class”? Что происходит при комбинации виртуального и неvirtуального наследования?

Что такое виртуальные функции? В чем разница между виртуальными и неvirtуальными функциями при наследовании? Что такое таблица виртуальных функций? Что такое абстрактные классы и чисто виртуальные функции? Для чего нужен виртуальный деструктор? Что такое полиморфизм и как это понятие в C++ связано с виртуальными функциями?

Расскажите о разновидностях приведений типов в C++. В чем разница между C-style `cast`, `static_cast`, `const_cast`, `dynamic_cast` и `reinterpret_cast`, когда они применимы, а когда нет? Для чего нужен каждый из этих кастов?

Расскажите о шаблонах в C++. Что такое инстанцирование шаблонов, специализация шаблонов? Какие 3 вида шаблонов существуют? Какие 3 вида шаблонных параметров существуют? Как использовать шаблоны с переменным количеством аргументов и оператор `sizeof...`? Что такое Curiously Recurring Template Pattern? Что такое полиморфизм и как это понятие в C++ связано с шаблонами?

Расскажите об исключениях в C++. В чем их идея? Как пользоваться оператором `throw` и конструкцией `try...catch`? В чем разница между исключениями и ошибками времени выполнения? Что такое спецификации исключений, как пользоваться оператором и спецификатором `noexcept`? В чем особенности и проблемы исключений в конструкторах и деструкторах?

Расскажите о контейнере `std::vector`. Предложите реализацию конструкторов, деструктора, операторов присваивания, оператора `[]`, методов `at`, `resize`, `reserve`, `front`, `back` для этого класса (с правильной поддержкой аллокаторов и `move`-семантики).

Расскажите о контейнере `std::vector`. Предложите реализацию методов `push_back` и `pop_back` (с правильной поддержкой аллокаторов, `move`-семантики и гарантий безопасности относительно исключений).

Расскажите про `vector<bool>`. В чем его особенность и отличие от обычного `vector`? Расскажите, каким образом достигается эта особенность с точки зрения реализации (опишите реализацию основных методов).

Расскажите о контейнере `std::list`. Каковы его методы и скорость их работы? Как он устроен изнутри? Каким образом `list<T>` использует аллокатор для типа `T`? Предложите реализацию основных методов `list`'а: конструкторы, операторы присваивания, деструктор, `insert` элемента по итератору и `erase` по итератору.

Расскажите о контейнере `std::forward_list`. Каковы его методы и скорость их работы? Как он устроен изнутри? Каким образом `forward_list<T>` использует аллокатор для типа `T`? Предложите реализацию основных методов `forward_list`'а: конструкторы, операторы присваивания, деструктор, `insert` элемента по итератору и `erase` по итератору.

Расскажите о контейнере `std::deque`. Чем он отличается от `vector`? Расскажите об адаптерах над контейнерами (`std::stack`, `std::queue`, `std::priority_queue`). Предложите реализацию какого-нибудь одного из них.

Расскажите о контейнерах `std::map`, `std::set`, `std::multimap` и `std::multiset`. Для чего они применяются? Какие у них шаблонные параметры и что они означают? Каковы их основные методы, скорость и принцип работы этих методов? С помощью какой структуры данных реализованы эти контейнеры?

Расскажите о контейнерах `std::unordered_map`, `std::unordered_set`, `std::unordered_multimap` и `std::unordered_multiset`. Для чего они применяются? Какие у них шаблонные параметры и что они означают? Каковы их основные методы, скорость и принцип работы этих методов? С помощью какой структуры данных реализованы эти контейнеры?

Что такое итераторы? Какие виды итераторов бывают и какие операции допустимы над каждым из них? Расскажите про функции `std::advance` и `std::distance`. Каким образом достигается их разное поведение в зависимости от вида переданного итератора?

Какие виды итераторов поддерживает каждый из стандартных контейнеров? Предложите реализацию итераторов для какого-нибудь из стандартных контейнеров (на ваш выбор), считая, что в остальном контейнер уже реализован.

Расскажите про класс `std::insert_iterator` и функцию `std::inserter`, предложите их реализацию.

## Часть 2.

Расскажите об операторах `new` и `delete`. Зачем они нужны, как ими пользоваться? Что такое `placement new`? В чем разница между оператором `new` и функцией `operator new`? Что из вышеперечисленного можно перегружать и как это делать? Зачем может быть нужно перегружать кастомный `operator delete`? Расскажите о проблеме утечек памяти.

Что такое аллокаторы, зачем они нужны? Расскажите про класс `std::allocator`, предложите его реализацию.

Расскажите о проблемах, для решения которых была введена move-семантика. Расскажите, как работает функция `std::move` и как ее применять, объясните, как она реализована и почему именно так. Как правильно реализовать функцию `swap` в C++11?

Расскажите о том, что такое “идеальная передача” (perfect forwarding). Расскажите, как работает функция `std::forward` и как она реализована. Приведите пример применения этой функции. Почему в `std::forward` нельзя принять T&& в качестве параметра?

Расскажите об rvalue-ссылках и об универсальных ссылках. Какие присваивания между lvalue-, rvalue-ссылками, а также временными объектами и именованными нессылочными объектами разрешены, а какие запрещены? Что меняется в случае с константными ссылками и объектами? Что является и что не является универсальными ссылками? Приведите примеры, иллюстрирующие все вышесказанное.

Что такое lvalue, rvalue, xvalue, glvalue и prvalue? Приведите примеры. Расскажите о том, что такое reference collapsing (сворачивание ссылок), reference qualifiers (ссылочные квалификаторы), return value optimization (RVO) и copy elision. Приведите примеры, иллюстрирующие все вышеперечисленное.

Расскажите, для решения какой проблемы нужны умные указатели. Расскажите про класс `std::weak_ptr`, предложите реализацию основных методов этого класса (конструкторы, деструктор, операторы присваивания, операторы унарная звездочка и стрелочка).

Расскажите про класс `std::shared_ptr`. Какую проблему он решает и как он устроен? Предложите реализацию его основных методов (конструкторы, деструктор, операторы присваивания, операторы унарная звездочка и стрелочка).

Предложите реализацию функции `std::make_unique`. Зачем нужна эта функция, какую потенциальную проблему она решает? Расскажите о функциях `make_shared` и `allocate_shared` (без реализации). Зачем нужна каждая из них, какие потенциальные проблемы они решают?

Расскажите про класс `std::weak_ptr`. Для решения какой проблемы он нужен? Предложите реализацию его основных методов. Как `weak_ptr` узнает, что объект под ним уже был удален?

Расскажите про класс `std::enable_shared_from_this`. Для решения какой проблемы он нужен и как им пользоваться? Предложите реализацию этого класса. Как нужно изменить код `shared_ptr`, чтобы он работал согласованно с `enable_shared_from_this`?

Расскажите о выводе типов с помощью ключевых слов `auto` и `decltype`. В чем разница между правилами вывода типов для них? В каких контекстах можно использовать `auto`? Как работает `decltype` от идентификаторов и от разных категорий выражений (lvalue, xvalue, prvalue)? Что такое `decltype(auto)` и для решения какой проблемы это нужно?

Расскажите о возможностях compile-time вычислений в C++. Как с помощью шаблонного метапрограммирования имитировать циклы и условия? Покажите на примере проверки целого числа на простоту. Расскажите о ключевом слове `constexpr`: чем `constexpr` отличается от `const`, зачем нужны `constexpr`-функции и переменные, какими возможностями они обладают и какие ограничения на них накладываются?

Что такое `type_traits`? Объясните, как реализованы структуры `std::remove_reference`, `std::is_same`, `std::rank`, `std::conjunction`, `std::conditional`, `std::common_type`.

Что такое SFINAE? Объясните общую идею на примере структуры `std::enable_if`. Предложите реализацию структуры `std::is_class`, объясните, как она работает (проверкой на `is_union` можно пренебречь).

Зачем нужна функция `std::declval`? Почему эта функция возвращает T&&, а не T? Предложите реализацию структуры `std::is_constructible`, объясните, как она работает.

Зачем нужен класс `std::allocator_traits`? Предложите реализацию функции `allocator_traits<Allocator>::construct`. Каким образом эта функция проверяет, реализован ли у аллокатора метод `construct`, и как она себя ведет в зависимости от этого?

Расскажите о функции `std::move_if_noexcept`. В каком месте стандартной библиотеки она используется и почему она там необходима? Предложите ее реализацию (считая, что `type_traits` уже реализованы).

Предложите реализацию структуры `std::is_base_of`. Объясните, как она работает.

Что такое лямбда-функции в C++? Каков синтаксис лямбда-выражений? Что такое списки захвата, что такое захват по умолчанию и в чем его опасность? Расскажите о дополнительных

возможностях лямбда-выражений в C++14 (захват с инициализацией, обобщенные лямбда-выражения). Расскажите (без реализации) про класс `std::function` и функцию `std::bind`, приведите примеры их использования.

Что такое юнионы (`union`), для чего они нужны? В чем сходство и различие между юнионами и классами? Каковы особенности инициализации полей юниона, что такое “активный член” юниона и как его поменять? Что такое `std::variant` и `std::any`, для чего они нужны, каковы их основные методы?

Что такое рефлексия? Что можно сказать о возможностях рефлексии в C++? Реализуйте шаблонную функцию `detect_fields_count<S>`, которая бы позволяла определять количество полей у данной структуры `S`. Объясните, как это работает.

Примеры экзаменц. билетов:

билет №1

Что такое SFINAE? Приведите хотя бы один пример (на уровне идей, можно без реализации). Что такое перегрузка функций? Что такое перегрузка операторов? Приведите примеры операторов, которые можно и нельзя перегружать. Приведите пример перегрузки хоть какого-нибудь оператора (напишите сигнатуру его перегрузки).

билет №2

Что такое рефлексия? Что можно сказать о возможностях рефлексии в C++? Реализуйте шаблонную функцию `detect_fields_count<S>`, которая бы позволяла определять количество полей у данной структуры `S`. Объясните, как это работает.

Расскажите о проблемах, для решения которых была введена move-семантика. Расскажите, как работает функция `std::move` и как ее применять, объясните, как она реализована и почему именно так. Как правильно реализовать функцию `swap` в C++11?

билет №3

Какие виды итераторов поддерживает каждый из стандартных контейнеров? Предложите реализацию итераторов для какого-нибудь из стандартных контейнеров (на ваш выбор), считая, что в остальном контейнер уже реализован.

Расскажите про класс `std::insert_iterator` и функцию `std::inserter`, предложите их реализацию.

Критерии оценивания

отлично

10 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы, код оформлен в едином удобочитаемом стиле

9 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы

8 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач

хорошо

7 Полностью решены все задачи. Допущены несущественные ошибки.

6 Полностью решено большинство задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

5 Полностью решено две трети задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

удовлетворительно

4 Полностью решено более половины задач. В остальных задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.



3 Полностью решено более половины задач.

неудовлетворительно

2 Решено менее половины задач.

1 Не решено ни одной задачи.

## **5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности**

Дифференцированный зачет будет состоять из двух частей: предварительной части и основной части.

Предварительная часть будет проходить письменно. Вам будут выданы 4 вопроса в пределах теорминимума на 15 минут, писать ответы нужно будет на выданной бумаге, ничем нельзя пользоваться. Вопросы выбираются из списка “Теорминимум”, приведенного ниже. Ответы оцениваются бинарно. Чтобы успешно пройти предварительную часть, необходимо правильно ответить не менее чем на 3 вопроса из заданных четырех. Если вы успешно проходите предварительную часть, то переходите к основной части; иначе уходите с экзамена с оценкой неуд(2).

Гарантируется, что на предварительной части не будут задаваться вопросы, выходящие за рамки списка “Теорминимум” (с той оговоркой, что вопрос вида “Что такое X” всегда подразумевает умение привести пример). Гарантируется, что один вопрос на предварительной части будет покрывать не более одного пункта данного списка (но необязательно будет совпадать с пунктом списка дословно).

Примеры возможных вопросов предварительной части и правильных ответов на них:

Что такое виртуальная функция? (Это метод класса, который можно переопределить в классах-наследниках так, что реализация метода для вызова будет выбираться во время исполнения, а не во время компиляции.)

Для решения каких проблем нужна move-семантика? (Например, таких: неэффективный swap двух произвольных объектов, неэффективный push\_back в вектор - с демонстрацией на примере кода.)

Каково действие оператора “запятая”? (Вычислить левый операнд, строго после этого вычислить правый операнд и вернуть результат вычисления правого операнда.)

Основная часть проходит по следующим правилам. Вы тянете билет. Билет будет содержать два вопроса из нижеприведенного списка “Основные вопросы”. Список разбит на две части. Один вопрос в каждом билете будет из первой части, один вопрос - из второй части.

Во время подготовки ответа на билет можно пользоваться своими заранее подготовленными рукописными записями в пределах одного листа А4. Пользоваться записями, сделанными на чем-либо, кроме одного заранее выбранного листа А4, или любой печатной (а не рукописной) информацией, или интернет-ресурсами запрещено. Нарушение этого правила приравнивается к списыванию и ведет к удалению с экзамена. (Смысл данного разрешения в том, чтобы каждый мог записать для себя наиболее трудные для запоминания детали. Не следует воспринимать это как призыв постараться законспектировать материал всего курса на одном листе.) С момента начала ответа на билет пользоваться вышеназванным листом запрещается (но, разумеется, можно пользоваться записями, сделанными на другой бумаге во время подготовки ответа на билет).

В ходе основной части экзамена список “Теорминимум” служит ориентиром на случай, если речь идет об оценке уд(3) и ниже. Если в любой момент экзамена выяснится, что Вы не можете ответить хотя бы на несколько вопросов из списка “Теорминимум”, то, вероятнее всего, Вашей оценкой за экзамен будет неуд(2).

Если вы претендуете на оценку выше отл(8), то, скорее всего, вам будут заданы дополнительные вопросы в рамках общей программы курса, приведенной выше (а не только в рамках списков “Основные вопросы” и “Теорминимум”), за исключением того, что выделено красным цветом. В любом случае, преподаватель, если сочтет нужным, может дополнительно к билету задавать вопросы из любого из трех списков в любом количестве по своему усмотрению.