




МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДФУ)
ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

СОГЛАСОВАНО
Руководитель ОП

(подпись) Сущенко А.А.
(ФИО)

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись) Неботарев А.Ю.
(ФИО)
«25» марта 2022 г.



РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ
Объектно-ориентированное программирование
Направление подготовки 01.03.02 Прикладная математика и информатика
(Программы бакалавриата «Прикладная математика и компьютерные науки»)
Форма подготовки очная

курс 1 семестр 2
лекции 12 час.
практические занятия 0 час.
лабораторные работы 30 час.
в том числе с использованием МАО лек. 12 /пр. 0 /лаб. 14 час.
всего часов аудиторной нагрузки 42 час.
в том числе с использованием МАО 26 час.
самостоятельная работа 66 час.
в том числе на подготовку к экзамену 36 час.
контрольные работы (количество) 2
курсовая работа / курсовой проект - / -
зачет 2 семестр
экзамен 2 семестр

Рабочая программа составлена в соответствии с требованиями Федерального государственного образовательного стандарта по направлению подготовки 01.03.02 **Прикладная математика и информатика** утвержденного приказом Министерства образования и науки РФ от 10 января 2018 г. № 9 с изменениями и дополнениями.

Рабочая программа обсуждена на заседании департамента, математического и компьютерного моделирования протокол № 6 от «25» марта 2022 г.

Директор департамента

Сущенко А.А.

Составители:

Сущенко А.А.

Оборотная сторона титульного листа РПД

I. Рабочая программа пересмотрена на заседании кафедры/департамента:

Протокол от « _____ » _____ 20__ г. № _____

Заведующий *кафедрой* _____
(подпись) (И.О. Фамилия)

II. Рабочая программа пересмотрена на заседании кафедры/департамента:

Протокол от « _____ » _____ 20__ г. № _____

Заведующий *кафедрой* _____
(подпись) (И.О. Фамилия)

III. Рабочая программа пересмотрена на заседании кафедры/департамента:

Протокол от « _____ » _____ 20__ г. № _____

Заведующий *кафедрой* _____
(подпись) (И.О. Фамилия)

IV. Рабочая программа пересмотрена на заседании кафедры/департамента:

Протокол от « _____ » _____ 20__ г. № _____

Заведующий *кафедрой* _____
(подпись) (И.О. Фамилия)

1. Цели и задачи освоения дисциплины:

Цель:

Изучение базовых основ языка программирования C++ и приобретение навыком объектно-ориентированного программирования.

Задачи:

- ознакомить студентов с языком программирования C++;
- научить основам объектно-ориентированного программирования;
- дать навыки реализации сложных алгоритмов с использованием указанных технологий.

Для успешного изучения дисциплины «Объектно-ориентированное программирование» у обучающихся должны быть сформированы следующие предварительные компетенции:

- УК-2. Способен определять круг задач в рамках поставленной задачи и выбирать эффективные методы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений

- ОПК-1 способен применять теоретические знания, полученные в области математических и естественных наук, и применять их в профессиональной деятельности

- ОПК-4 способен решать задачи профессиональной деятельности с применением существующих информационно-коммуникационных технологий и с учетом основных требований информационной безопасности

Профессиональные компетенции выпускников и индикаторы их достижения:

| Тип задач | Код и наименование профессиональной компетенции (результат освоения) | Код и наименование индикатора достижения компетенции |
|---------------------------------|---|---|
| Производственно-технологический | ПК-6 Способен осуществлять целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" (далее - сеть "Интернет") и в других источниках | ПК-6.1 проводит целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети «Интернет» и в других источниках |
| | | ПК-6.2 работает с базами данных и другими источниками по новейшим решениям и научно-технологическим достижениям |
| | | ПК-6.3 применяет современные информационно-коммуникационные технологии, учитывая требования |

| Тип задач | Код и наименование профессиональной компетенции (результат освоения) | Код и наименование индикатора достижения компетенции |
|-------------------------------|--|--|
| | | информационной безопасности, для решения профессиональных задач |
| Организационно-управленческий | ПК-7 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы | ПК-7.1 использует принципы разработки и отладки программного кода, методы тестирования программного обеспечения |
| | | ПК-7.2 использует методы по выявлению и устранению сбоев и отказов в работе программного обеспечения, ликвидации их последствия и восстановления работоспособности |
| | | ПК-7.3 разрабатывает программный код, проводит его отладку и тестирование, своевременно принимает меры по выявлению и устранению сбоев и отказов в работе программного обеспечения, ликвидации их последствий и восстановлению работоспособности |

| Код и наименование индикатора достижения компетенции | Наименование показателя оценивания (результата обучения по дисциплине) |
|---|--|
| ПК-6.1 проводит целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети «Интернет» и в других источниках | Знает |
| | Умеет |
| | Владеет |
| ПК-6.2 работает с базами данных и другими источниками по новейшим решениям и научно-технологическим достижениям | Знает |
| | Умеет |
| | Владеет |
| ПК-6.3 применяет современные информационно-коммуникационные технологии, учитывая требования информационной безопасности, для решения профессиональных задач | Знает |
| | Умеет |
| | Владеет |
| ПК-7.1 использует принципы разработки и отладки программного кода, методы | Знает |
| | Умеет |
| | Владеет |

| Код и наименование индикатора достижения компетенции | Наименование показателя оценивания (результата обучения по дисциплине) |
|--|--|
| тестирования программного обеспечения | |
| ПК-7.2 использует методы по выявлению и устранению сбоев и отказов в работе программного обеспечения, ликвидации их последствия и восстановления работоспособности | Знает |
| | Умеет |
| | Владеет |
| ПК-7.3 разрабатывает программный код, проводит его отладку и тестирование, своевременно принимает меры по выявлению и устранению сбоев и отказов в работе программного обеспечения, ликвидации их последствий и восстановлению работоспособности | Знает |
| | Умеет |
| | Владеет |

2. Трудоёмкость дисциплины и видов учебных занятий по дисциплине

Общая трудоёмкость дисциплины составляет 3 зачётных единиц (108 академических часов).

(1 зачетная единица соответствует 36 академическим часам)

Видами учебных занятий и работы обучающегося по дисциплине могут являться:

| Обозначение | Виды учебных занятий и работы обучающегося |
|-------------|---|
| Лек | Лекции |
| Лаб | Лабораторные работы |
| Пр | Практические занятия |
| ОК | Онлайн курс |
| СР | Самостоятельная работа обучающегося в период теоретического обучения |
| Контроль | Самостоятельная работа обучающегося и контактная работа обучающегося с преподавателем в период промежуточной аттестации |

Структура дисциплины:

Форма обучения – очная.

| № | Наименование раздела дисциплины | С | Количество часов по видам учебных занятий и работы обучающегося | |
|---|---------------------------------|---|---|--|
| | | е | | |

| | м е с т р | Лек | Лаб | Пр | ОК | СР | Контр оль | Формы промежуточной аттестации |
|---|-----------------------|-----|-----|----|----|----|--------------|--------------------------------------|
| Основы управления памятью | 2 | 2 | 3 | | | 1 | 2 | экзамен 1 зачет 1 |
| Полиморфизм. Friend. Const-correctness Пространства имен. | 2 | 1 | 2 | | | 1 | 2 | экзамен 1 зачет 1 |
| Управление памятью. Аллокаторы. Умные указатели | 2 | 1 | 2 | | | 1 | 2 | экзамен 1 зачет 1 |
| Стандартная библиотека. STL. Functional. Exceptions. | 2 | 1 | 2 | | | 2 | 2 | экзамен 1 зачет 1 |
| Шаблоны классов и функций. Специализация шаблона. Инстанцирование шаблона. SFINAE. Variadic templates. | 2 | 1 | 2 | | | 1 | 2 | экзамен 1 зачет 1 |
| Rvalue, lvalue. Move-семантика. Правила вывода типов в шаблонах, auto, decltype(auto). | 2 | 1 | 2 | | | 3 | 2 | экзамен 1 зачет 1 |
| . Exception-safety. RAII. Basic, strong, nothrow guarantees. | 2 | 1 | 2 | | | 3 | 2 | экзамен 1 зачет 1 |
| OOP. Liskov substitution. Scott's solution. NVI. | 2 | 1 | 2 | | | 3 | 2 | экзамен 1 зачет 1 |
| Exception-safety-2. Uncaught exception/exceptions. ScopeGuard idiom. SCOPE_EXIT, SCOPE_FAIL, SCOPE_SUCCESS. | 2 | 1 | 2 | | | 4 | 2 | экзамен 1 зачет 1 |
| Undefined behavior. | 2 | 1 | 2 | | | 4 | 2 | экзамен 1 зачет 1 |
| Многопоточность. | 2 | 1 | 2 | | | 4 | 2 | экзамен 1 зачет 1 |
| Итого: | 2, 2 | 12 | 30 | 0 | 0 | 30 | 22 | Э(2)Зач(2) |

I. СТРУКТУРА И СОДЕРЖАНИЕ ТЕОРЕТИЧЕСКОЙ ЧАСТИ КУРСА

Лекционные занятия (12 час.)

Тема 1. Основы управления памятью.

Тема 2. Полиморфизм. Friend. Const-correctness Пространства имен.

Тема 3. Управление памятью. Аллокаторы. Умные указатели.

Тема 4. Стандартная библиотека. STL. Functional. Exceptions.

Тема 5. Шаблоны классов и функций. Специализация шаблона. Инстанцирование шаблона. SFINAE. Variadic templates.

Тема 6. Rvalue, lvalue. Move-семантика. Правила вывода типов в шаблонах, auto, decltype(auto).

Тема 7. Exception-safety. RAII. Basic, strong, nothrow guarantees.

Тема 8. OOP. Liskov substitution. Scott's solution. NVI.

Тема 9. Exception-safety-2. Uncaught exception/exceptions. ScopeGuard idiom. SCOPE_EXIT, SCOPE_FAIL, SCOPE_SUCCESS.

Тема 10. Undefined behavior.

Тема 11. Многопоточность.

II. СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА И САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Лабораторные работы (30 часов)

Раздел 1.

Лабораторная работа 1. Необходимо реализовать класс PrimeNumberGenerator — генератор простых чисел. У класса должен быть конструктор, принимающий (int start), и функция int GetNextPrime(), возвращающая ближайшее справа от start-а простое число (включая start). В конструкторе копирования требуется скопировать только значение value, при этом modulo задается равным нулю.

Функция GetNextPrime должна изменять состояние объекта — при повторном ее вызове нужно возвращать уже следующее простое число.

Лабораторная работа 2. Вам необходимо написать .cpp файл с реализацией хедера num.h. В конструкторе Num необходимо сохранять значение value по модулю modulo! По умолчанию modulo равно нулю, в таком случае value сохраняется без взятия по модулю.

Лабораторная работа 3. Реализовать класс Date со следующими методами: Конструктор Date(int year, int month, int day). Метод bool IsLeap() const, возвращающий true в случае, если год является високосным и false в противном случае. Метод std::string ToString() const, возвращающий строковое представление даты в формате dd.mm.yyyy. Метод Date DaysLater(int days) const, возвращающий дату, которая наступит спустя days дней от текущей. Метод int DaysLeft(const Date& date) const, возвращающий разницу между указанной и текущей датой (в днях).

Лабораторная работа 4. Необходимо реализовать Set — класс, в котором реализованы основные операции над множествами:

```
Set Union(const Set&) const,
```

```
Set Intersection(const Set&) const,
```

```
Set Difference(const Set&) const,
```

```
Set SymmetricDifference(const Set&) const.
```

Также необходимо реализовать конструктор Set(const std::vector&) и функции добавления, удаления и проверки наличия элемента во множестве: void Add(int64_t), void Remove(int64_t), bool Contains(int64_t) const.

Для доступа ко всем элементам множества реализовать функцию std::vector Data() const. Предполагается, что класс будет использован для хранения целых чисел типа int64_t. Для хранения элементов следует использовать std::vector с соответствующим параметром шаблона.

Лабораторная работа 5. Вам необходимо написать .cpp файл с реализацией хедера num.h.

Лабораторная работа 6. Требуется реализовать класс BufferedReader со следующим интерфейсом:

```
class BufferedReader {
```

```
public:
```

```
    explicit BufferedReader(PackageStream* stream);
```

```
    int32_t Read(char* output_buffer, int32_t buffer_len);
```

```
};
```

В конструктор BufferedReader передается указатель на объект класса PackageStream (см. описание ниже), с помощью которого будут считываться пакеты некоторой длины. Метод int32_t Read(char* output_buffer, int32_t buffer_len) записывает по указателю output_buffer пакет длины не более buffer_len и возвращает реальный размер записанного пакета (это число может быть меньше, чем заданная длина, если строка закончилась раньше).

Интерфейс класса PackageStream:

```
class PackageStream {
```

```
public:
```

```
    PackageStream(std::string source, int32_t package_len);
```

```
    int32_t PackageLen() const;
```

```
    int32_t ReadPackage(char* output_package);
```

```
};
```

В конструктор PackageStream передается строка source, из которой впоследствии побайтово будут считываться пакеты длины package_len и, собственно, длина пакетов package_len. Метод int32_t PackageLen() возвращает длину пакета (package_len), который считывает метод ReadPackage. Метод int32_t ReadPackage(char* output_package) записывает по указателю output_package пакет длины не более package_len и возвращает реальный размер записанного пакета.

Раздел 2.

Лабораторная работа 1. Конструктор класса PageAllocator принимает размер блока в байтах. Функция Allocate выделяет блок размера, заданного в конструкторе. Данный класс реализовывать не нужно.

```
class PageAllocator {
public:
    explicit PageAllocator(std::uint64_t page_size);
    void* Allocate();
};
```

Необходимо реализовать класс FixedAllocator, у которого должны быть: Конструктор принимающий page_size — размер блока в элементах типа Tr. Функция Allocate возвращающая указатель на следующую свободную память. Если свободной памяти нет функция Allocate получает ее через объект page_allocator_. Функция Deallocate добавляющая указатель обратно в пул свободной памяти. Функция InnerAllocator возвращающая неизменяемую ссылку на объект page_allocator_

```
template<typename Tr>
class FixedAllocator {
    PageAllocator page_allocator_;
public:
    explicit FixedAllocator(std::uint64_t page_size);
    Tr* Allocate();
    void Deallocate(Tr* p);
    const PageAllocator& InnerAllocator() const noexcept;
};
```

Таким образом вы должны выделять минимально возможное кол-во блоков памяти (кол-во вызовов Allocate у объекта page_allocator_). Выделять память можно только с помощью данного объекта.

Лабораторная работа 2. Вам необходимо реализовать класс SmartPointer, интерфейс которого находится в файле SmartPointer.hpp, а также реализовать вспомогательный класс Core. Ограничение: При реализации класса SmartPointer нельзя добавлять новые поля.

Лабораторная работа 3. Реализуйте паттерн проектирования "Фабрика". Фабрика может создавать произвольных потомков базового класса. В нашем случае базовым классом будет Object, а сама фабрика — классом Factory. Определение класса Object должно быть в точности таким:

```
class Object {
public:
    virtual std::string ToString() const = 0;
    virtual ~Object() {}
};
```

Метод ToString является абстрактным. Это означает, что все потомки Object обязаны перегрузить этот метод.

Ваша фабрика должна уметь понимать, потомка какого типа от неё хотят получить в данный момент. Это означает, что у каждого потомка должен быть

некоторый идентификатор. В этом задании будем использовать строковые идентификаторы.

Фабрика поддерживает всего две операции. Одна из них: `Object* Create(const std::string& class_id)` — этот метод класса `Factory` получает на вход идентификатор класса, создает экземпляр этого класса и возвращает указатель на созданный экземпляр. Сразу после конструирования ваша фабрика должна уметь создавать потомков с идентификаторами "apple!", "list" и "yet another identifier". В этом задании все потомки `Object` при вызове `ToString` должны возвращать свои идентификаторы. Например, код

```
Factory factory;
Object* apple_instance_ptr = factory.Create("apple!");
cout << apple_instance_ptr->ToString() << endl;
должен печатать "apple!".
```

Чтобы не было скучно, ваша фабрика должна поддерживать создание любых потомков `Object`. Для этого существует операция регистрации:

`void Register(const std::string& class_id, Object*(*instance_creator)())` — этот метод связывает идентификатор класса `class_id` с порождающей функцией `instance_creator`. Параметр `instance_creator` — это указатель на функцию, которая возвращает указатель на наследника `Object`. Пример использования:

```
Factory factory;
factory.Register("smth", new_smth);
Object* smth_instance_ptr = factory.Create("smth");
cout << smth_instance_ptr->ToString() << endl;
```

Где `new_smth` это функция, объявленная как `Object* new_smth()`; Файл с решением должен содержать только реализацию классов `Factory` и `Object` и вспомогательных классов, если необходимы.

Лабораторная работа 4. Вам заданы классы узлов синтаксического дерева программы, необходимые для описания объявления класса, методов класса и полей класса.

```
class ClassDeclarationNode;
class VarDeclarationNode;
class MethodDeclarationNode;
class BaseNode;
class BaseVisitor {
public:
    virtual void Visit(const BaseNode* node) = 0;
    virtual void Visit(const ClassDeclarationNode* node) = 0;
    virtual void Visit(const VarDeclarationNode* node) = 0;
    virtual void Visit(const MethodDeclarationNode* node) = 0;
};
class BaseNode {
public:
    virtual void Visit(BaseVisitor* visitor) const = 0;
};
class ClassDeclarationNode: public BaseNode {
```

```

public:
    const std::string& ClassName() const;
    const std::vector<BaseNode*>& PublicFields() const;
    const std::vector<BaseNode*>& ProtectedFields() const;
    const std::vector<BaseNode*>& PrivateFields() const;
    void Visit(BaseVisitor* visitor) const override {
        visitor->Visit(this);
    }
};
class VarDeclarationNode: public BaseNode {
public:
    const std::string& VarName() const;
    const std::string& TypeName() const;
    void Visit(BaseVisitor* visitor) const override {
        visitor->Visit(this);
    }
};
class MethodDeclarationNode: public BaseNode {
public:
    const std::string& MethodName() const;
    const std::string& ReturnTypeName() const;
    const std::vector<BaseNode*>& Arguments() const;
    void Visit(BaseVisitor* visitor) const override {
        visitor->Visit(this);
    }
};

```

Требуется реализовать класс FormatVisitor, который будет позволять получать отформатированное представление программы в виде строки, в соответствии с синтаксисом языка C++ и Google Style Guide.

```

class FormatVisitor: public BaseVisitor {
public:
    void Visit(const BaseNode* node) override {
        node->Visit(this);
    }
    void Visit(const ClassDeclarationNode* node) override;
    void Visit(const VarDeclarationNode* node) override;
    void Visit(const MethodDeclarationNode* node) override;
    const std::vector<std::string>& GetFormattedCode() const;
};

```

Лабораторная работа 5. Необходимо реализовать класс GameDatabase со следующим интерфейсом:

```

class GameDatabase
{
public:
    GameDatabase() = default;

```

```

/// вставляет в базу объект с именем [name] и позицией [x, y]
/// если объект с таким id в базе уже есть, заменяет его новым
void Insert(ObjectId id, string name, size_t x, size_t y)
/// удаляет элемент по id
/// если такого элемента нет, ничего не делает
void Remove(ObjectId id);
/// возвращает вектор объектов с именем [name]
/// сортировка по убыванию id
vector<GameObject> DataByName(string name) const;
/// возвращает вектор объектов, находящихся в позиции [x, y]
/// сортировка по убыванию id
vector<GameObject> DataByPosition(size_t x, size_t y) const;
/// возвращает вектор всех объектов из базы
/// сортировка по убыванию id
vector<GameObject> Data() const;
};

```

Код для GameObject и ObjectId указан ниже.

```

using ObjectId = unsigned long long int;
struct GameObject
{
    ObjectId id;
    string name;
    size_t x;
    size_t y;
};

```

Рекомендуется использовать структуры данных: `std::unordered_map`, `std::map`, `std::set`. Отдельная сортировка не потребуется если использовать компаратор для упорядоченных контейнеров (`std::set`, `std::map`). Пример организации данных с компаратором:

```

bool operator>(const GameObject& a, const GameObject& b) {
    return a.id > b.id;
}

```

```

template<class Tp, template<class> class Compare>
class DereferenceCompare {
    Compare<Tp> comp;

```

```

public:
    bool operator()(const Tp* const a, const Tp* const b) const {
        return comp(*a, *b);
    }
};

```

```

/// быстрый доступ по id
std::map<ObjectId, GameObject, std::greater<ObjectId>>

```

```
/// быстрый доступ по позиции
std::map<std::pair<size_t, size_t>, std::set<GameObject*>,
DereferenceCompare<GameObject, std::greater>>>
```

```
/// быстрый доступ по имени
std::unordered_map<string, std::set<GameObject*>,
DereferenceCompare<GameObject, std::greater>>>
```

Раздел 3.

Лабораторная работа 1. Вам необходимо написать функцию `initialize_vector(value, dim1, dim2, ...)`, принимающую значение и размерности, и возвращающую вектор заданных размерностей, заполненный этим значением. Пример использования такой функции может быть следующим:

```
vector<vector<vector<int>>> v = initialize_vector(-1, 100, 50, 30)
```

Для реализации требуется использовать `variadic templates`.

Лабораторная работа 2. Написать функцию: `Iterator Find<T, Iterator>(const T& value, Iterator first, Iterator last)`, которая принимает элемент и итераторы на отсортированную коллекцию и возвращает итератор на требуемый элемент (в случае отсутствия такого элемента, функция должна вернуть `last`). В зависимости от типа итератора, данная функция должна использовать бинарный или линейный поиск (Бинарный поиск, если итератор является `Random Access`).

Лабораторная работа 3. Необходимо реализовать функцию `MergeAssociative`, которая принимает 2 ассоциативных контейнера и добавляет содержимое второго к первому. Возвращает `false` если операцию можно выполнить (см. далее), иначе возвращает `true`

```
template<class C1, class C2>
bool MergeAssociative(C1* c1, const C2& c2)
```

С контейнерами, имеющимися в стандартной библиотеке C++ можно ознакомиться здесь. Операцию можно выполнить если верны 3 условия:

- 1) Оба типа являются ассоциативными контейнерами
- 2) Их типы элементов совпадают, не учитывая `cv qualifiers`
- 3) Первый контейнер является мультиконтейнером или оба ими не являются

Мультиконтейнерами в данном случае названы следующие: `multiset`, `unordered_multiset`, `multimap`, `unordered_multimap`. Примеры пар типов, для которых операцию выполнить можно:

```
multiset<int> + set<int>
map<int, int> + unordered_map<int, int>
multimap<int, const int> + unordered_map<int, volatile int>
```

Для которых нельзя:

```
set<int> + multiset<int>
set<int> + set<double>
int + double
```

Лабораторная работа 4. Реализовать шаблонный класс визитора со следующим интерфейсом для использования в алгоритме поиска в ширину в неориентированном графе.

```
template<Vertex>
class BfsVisitor {
public:
    void ExamineVertex(const Vertex& vertex);
    void DiscoverVertex(const Vertex& vertex);

    size_t DistanceTo(const Vertex& target) const;
    Vertex Parent(const Vertex& vertex) const;
}
```

Объект данного класса будет использован функцией обхода графа в ширину, аналогичной данной. Метод `ExamineVertex` будет вызван в момент извлечения вершины из очереди, метод `DiscoverVertex` будет вызван в момент добавления вершины в очередь. После обхода графа визитор должен хранить кратчайшие расстояния от начальной вершины до всех остальных. Для получения расстояния до вершины будет использован метод `DistanceTo`. Также, в процессе обхода в ширину визитор должен построить соответствующее такому обходу остовное дерево графа. Метод `Parent` будет использован для получения предка каждой вершины в таком графе. Родителем корневой вершины является она сама. Экземпляр визитора передается в функцию по значению, и для эффективного копирования его размер должен быть не больше размера `shared_ptr`.

Лабораторная работа 5. Вам необходимо написать метафункцию `is_customly_convertible<A, B>`, которая проверяет, существует ли специализация структуры `Convert` для типов `A` и `B`. Интерфейс функции должен соответствовать аналогичным функциям из модуля `type_traits`, например `is_same`. Специализация структуры `Convert` может выглядеть следующим образом:

```
template <>
struct Convert<int, float> {
    float operator()(const int& a) {
        return a;
    }
};
```

Также необходимо реализовать 2 структуры: `NoTriviallyConstructible` — структуру без дефолтного конструктора и `NoCopyConstructible` — структуру без конструктора копирования. (Это единственные требования к структурам, все остальное — неважно). Для вышеописанных структур требуется добавить специализацию функтора `Convert`: для `(NoTriviallyConstructible, int)` и `(NoCopyConstructible, NoTriviallyConstructible)` и реализовать ей оператор `()` произвольным образом.

Лабораторная работа 6. Вам необходимо написать преобразование `TupleToVector` и обратное для него `VectorToTuple`

Типы на выходе должны быть без cv qualifiers и ссылок

```
tuple<vector<T1>, vector<T2>, vector<T3>, ...> ==> vector<tuple<T1, T2, T3,
...>>
vector<tuple<T1, T2, T3, ...>> ==> tuple<vector<T1>, vector<T2>, vector<T3>,
...>
```

```
tuple<vector<int>, vector<double>, vector<char>> tpl;
const tuple<const vector<int>, const vector<double>, vector<char>> tpl2;
vector<tuple<int, double, char>> vec;
const vector<tuple<const int, double, const char>> vec2;
static_assert(std::is_same_v<decltype(VectorToTuple(vec)),
decltype(tpl)>);
static_assert(std::is_same_v<decltype(TupleToVector(tpl)),
decltype(vec)>);
static_assert(std::is_same_v<decltype(VectorToTuple(vec2)),
decltype(tpl)>);
static_assert(std::is_same_v<decltype(TupleToVector(tpl2)),
decltype(vec)>);
vector<int> v1, v2, v3;
static_assert(std::is_same_v<decltype(TupleToVector(tuple<const
vector<int>&, const vector<int>&, const vector<int>&>{v1, v2, v3})),
vector<tuple<int, int, int>>>);
```

Если возможно, функции должны возвращать 'удобные' типы вместо std::tuple, а также принимать их в качестве параметра

В частности, tuple размера 2 должен быть преобразован в pair, а размера 1 в тип первого элемента

```
tuple<vector<int>, vector<double>> tpl;
vector<pair<int, double>> vec = TupleToVector(tpl); //
tuple<vector<int>, vector<double>> -> vector<tuple<int, double>> ->
vector<pair<int, double>>
vector<tuple<int, double>> vec2;
pair<vector<int>, vector<double>> tpl2 = VectorToTuple(vec2) //
vector<tuple<int, double>> -> tuple<vector<int>, vector<double>> ->
pair<vector<int>, vector<double>>
// *****
tuple<vector<int>> tpl;
vector<int> vec = TupleToVector(tpl); // tuple<vector<int>> ->
vector<tuple<int>> -> vector<int>
vector<tuple<int>> vec2;
vector<int> tpl2 = VectorToTuple(vec2) // vector<tuple<int>> ->
tuple<vector<int>> -> vector<int>
// *****
vector<int> vec;
vec = VectorToTuple(vec);
vec = TupleToVector(vec);
// *****
```

```
vector<pair<int, char>> vp;  
pair<vector<int>, vector<char>> pv = VectorToTuple(vp);  
vp = TupleToVector(pv);
```

Раздел 4.

Лабораторная работа 1. Необходимо реализовать функцию CaesarEncrypt обрабатывающую шифром Цезаря (правый сдвиг на 3) входную строку в несколько потоков. Гарантируется, что строка будет состоять только из маленьких латинских букв в кодировке ASCII

```
void CaesarEncrypt(std::string* s);
```

Функция должна обрабатывать быстрее (по системному времени), чем следующая:

```
void CaesarEncryptOneThread(std::string* s)  
{  
    for (char& c : *s)  
        c = 'a' + (c + 3 - 'a') % 26;  
}
```

Лабораторная работа 2. Вам дан класс с основными функциями для реализации матриц со следующим интерфейсом.

```
class DenseMat {  
public:  
    DenseMat(int32_t rows = 0, int32_t cols = 0);  
    DenseMat(int32_t rows, int32_t cols, const std::vector<int32_t>& data);  
    int32_t Rows() const;  
    int32_t Cols() const;  
    const int32_t& operator()(int row, int col) const;  
    int32_t& operator()(int row, int col);  
    bool operator==(const DenseMat& other) const;  
    bool operator!=(const DenseMat& other) const;  
};
```

Требуется реализовать функцию DenseMat MatMulParal(const DenseMat& a, const DenseMat& b, int thread_count);, которая выдает результат умножения матрицы a на матрицу b. Функция должна использовать алгоритм параллельного умножения матриц, используя thread_count потоков. При перемножении матриц вычисление каждого i,j-го элемента матрицы-результата не зависит от порядка вычисления остальных элементов, поэтому вы можете вычислять отдельные части матрицы-результата независимо в разных потоках без синхронизации между ними. Тестирующая система будет проверять, что:

1. функция перемножения матриц выдает правильный результат при различных количествах потоков (в т.ч. большем, чем количество ядер на тестирующей машине).

2. с увеличением числа потоков до количества ядер время выполнения уменьшается.

3. наилучшее время выполнения достаточно мало.

Лабораторная работа 3. Вам необходимо реализовать thread-safe очередь со следующими методами:

```
template <typename T>
class Queue {
public:
    T Pop();

    size_t Size();

    template <typename U>
    void Push(??);

    template <typename ... U>
    void Emplace(??);
};
```

Очередь должна уметь работать с объектами без конструктора копирования.

III. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Объектно-ориентированное программирование» включает в себя:

1. план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;
2. характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;
3. требования к представлению и оформлению результатов самостоятельной работы;

План-график выполнения самостоятельной работы по дисциплине

| № п/п | Дата/сроки выполнения | Вид самостоятельной работы | Примерные нормы времени на выполнение | Форма контроля |
|--------------|------------------------------|---|--|--------------------------|
| 1 | Недели 1-2 | Подготовка к лабораторной работе №1-2, раздела 1 | 3 часа | Лабораторная работа №1-2 |
| 2 | Недели 3-4 | Подготовка к лабораторной работе №3-4, раздела 1 | 3 часа | Лабораторная работа №3-4 |
| 3 | Недели 5-6 | Подготовка к лабораторной работе №5-6, раздела 1 | 3 часа | Лабораторная работа №5-6 |
| 4 | Недели 7-8 | Подготовка к лабораторной работе №1-3, раздела 2 | 3 часа | Лабораторная работа №1-3 |
| 5 | Недели 9-10 | Подготовка к лабораторной работе №4-5, раздела 2 | 3 часа | Лабораторная работа №4-5 |
| 6 | Недели 11-12 | Подготовка к лабораторной работе №1-13, раздела 3 | 4 часа | Лабораторная работа №1-3 |
| 7 | Недели 13-14 | Подготовка к лабораторной работе №4-6, раздела 3 | 4 часа | Лабораторная работа №4-6 |

| | | | | |
|---|--------------|---------------------------------------|--------|--------------------------|
| 8 | Недели 15-16 | Подготовка к лабораторной работе №1-3 | 4 часа | Лабораторная работа №1-3 |
|---|--------------|---------------------------------------|--------|--------------------------|

Характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению

Самостоятельная работа студентов состоит из подготовки к лабораторным работам в компьютерном классе, работы над рекомендованной литературой. При подготовке к лабораторным работам необходимо сначала прочитать основные понятия по теме. При выполнении задания нужно сначала понять, что требуется в задаче, какой теоретический материал нужно использовать, наметить план решения задачи. Лабораторные работы выполняются студентами в командах.

Рекомендуется использовать методические указания и материалы по курсу «Объектно-ориентированное программирование», электронные пособия, имеющиеся на сервере Школы естественных наук, библиотеке ДВФУ и в сети Интернет. При подготовке к экзамену нужно освоить теорию: разобрать определения всех понятий и методов, рассмотреть примеры и самостоятельно решить несколько типовых задач из каждой темы. При решении задач всегда необходимо комментировать свои действия и не забывать о содержательной интерпретации.

Требования к представлению и оформлению результатов самостоятельной работы

Результатом самостоятельной работы студентов являются выполненные лабораторные работы. Лабораторные работы предоставляются в виде файлов приложений и сопровождаются пояснительной запиской.

IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

| № п/п | Контролируемые разделы / темы дисциплины | Код и наименование индикатора достижения | | Оценочные средства | |
|-------|--|--|---------|--|---|
| | | | | текущий контроль | промежуточная аттестация |
| 1. | Раздел 1. | ПК-6.1 ПК-7.1 | знает | Устные ответы на практических занятиях по разделу 1. | Письменный ответ на экзамене (программная реализация) |
| | | ПК-6.2 ПК-7.2 | умеет | Практическое задание по разделу 1. | |
| | | ПК-6.3 ПК-7.3 | владеет | Практическое задание по разделу 1 | |
| 2. | Раздел 2. | ПК-6.1 ПК-7.1 | знает | Устные ответы на практических занятиях по разделу 2. | Письменный ответ на экзамене (программная реализация) |
| | | ПК-6.2 ПК-7.2 | умеет | Практическое задание по разделу 2. | |
| | | ПК-6.3 ПК-7.3 | владеет | Практическое задание по разделу 2 | |
| 3. | Раздел 3. | ПК-6.1 ПК-7.1 | знает | Устные ответы на практических занятиях по разделу 3. | Письменный ответ на экзамене (программная реализация) |
| | | ПК-6.2 ПК-7.2 | умеет | Практическое задание по разделу 3. | |
| | | ПК-6.3 ПК-7.3 | владеет | Практическое задание по разделу 3 | |
| 4. | Раздел 4. | ПК-6.1 ПК-7.1 | знает | Устные ответы на практических занятиях по разделу 4. | Письменный ответ на экзамене (программная реализация) |
| | | ПК-6.2 ПК-7.2 | умеет | Практическое задание по разделу 4. | |
| | | ПК-6.3 ПК-7.3 | владеет | Практическое задание по разделу 4 | |

V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература

1. FirebirdSQL Manuals .- Firebird Foundation Inc., 2013.
<http://www.firebirdsql.org/en/documentation/>
2. Кауфман В. Ш. Языки программирования. Концепции и принципы. -- М.: Радио и связь, 1993.
3. Мартыненко Б. К. Языки и трансляции, СПбГУ, 2002
4. Corbett R. Bison manual, © 1998-2002 Free Software Foundation, Inc.
5. Paxson V. Flex, a fast scanner generator. Edition 2.5, © 1998 Free Software Foundation, Inc.
6. Kakde O. G. Algorithms for Compiler Design, © Charles River Media, 2002

7. Leone M. Research Language Overviews, © Carnegie-Mellon University, 2002
8. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework). — ISO/IEC, 2011 — 90 с.
9. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation). — ISO/IEC, 2011 — 1483 с.
10. ISO/IEC 9075-2:2011 Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata). — ISO/IEC, 2011 — 316 с.

Дополнительная литература
(печатные и электронные издания)

1. Gulutzan, Peter and Pelz, Trudy SQL-99 Complete, Really. - 2011.
2. <https://mariadb.com/kb/v/sql-99-complete-really/>
3. MariaDB Documentation. SkySQL Corporation Ab., 2013
4. <https://mariadb.com/kb/en/mariadb-documentation/>
5. PostgreSQL Manuals, version 9.3 .- The PostgreSQL Global Development Group, 2013.
6. <https://www.postgresql.org/docs/manuals/>
7. Калинин А. Г., Мацкевич И. В. Универсальные языки программирования. Семантический подход. -- М.: Радио и связь, 1991
8. Фомичев В. С. Формальные языки, грамматики и автоматы, СПбГЭУ "ЛЭТИ"
9. Kakde O. G. Algorithms for Compiler Design, © Charles River Media, 2002
10. Leone M. Research Language Overviews, © Carnegie-Mellon University, 2002

Перечень ресурсов информационно-телекоммуникационной сети
«Интернет»

1. Материалы сайта [Электронный ресурс]. – Режим доступа: <http://www.cplusplus.com>
2. Материалы сайта [Электронный ресурс]. – Режим доступа: <https://en.cppreference.com/w/>
3. Материалы сайта [Электронный ресурс]. – Режим доступа: [Stroustrup. 4 Edition](#)
4. Материалы сайта [Электронный ресурс]. – Режим доступа: [C++17 Standard](#)

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Рекомендации по планированию и организации времени, необходимого для изучения дисциплины.

Изучение теоретического материала по учебнику – 1 час в неделю.

Подготовка к практическому занятию и работе в компьютерном классе – 1 час.

Тогда общие затраты времени на освоение курса «Объектно-ориентированное программирование» студентами составят около 2 часов в неделю.

Описание последовательности действий студента («сценарий изучения дисциплины»).

При изучении дисциплины «Объектно-ориентированное программирование» следует внимательно слушать и конспектировать материал, излагаемый на аудиторных занятиях. Для его понимания и качественного усвоения рекомендуется следующая последовательность действий:

1. После окончания учебных занятий для закрепления материала просмотреть и обдумать выполненные сегодня практические работы, разобрать рассмотренные примеры (10-15 минут).
2. При подготовке к практической работе следующего дня повторить содержание предыдущей работы, подумать о том, какая может быть следующая тема (10-15 минут).
3. В течение недели выбрать время для работы со специальной литературой в библиотеке и для занятий на компьютере (по 2 часа).
4. При подготовке к практическим занятиям следующего дня необходимо сначала прочитать основные понятия по теме домашнего задания. При выполнении задания нужно сначала понять, что требуется в задаче, какой теоретический материал нужно использовать, наметить план решения задачи. Если это не дало результатов, и Вы сделали задачу «по образцу» аудиторной задачи, или из методического пособия, нужно после решения такой задачи обдумать ход решения и попробовать решить аналогичную задачу самостоятельно.

Рекомендации по работе с литературой

Теоретический материал курса становится более понятным, когда дополнительно изучаются и книги, и Интернет-ресурсы. Полезно использовать несколько учебников, однако легче освоить курс, придерживаясь одного учебника и конспекта. Рекомендуется, кроме «заучивания» материала, добиться понимания изучаемой темы дисциплины. Кроме того, очень полезно мысленно задать себе и попробовать ответить на следующие вопросы: о чем эта глава, какие новые понятия в ней введены.

Советы по подготовке к экзамену

Необходимо пользоваться учебниками. Вместо «заучивания» материала важно добиться понимания изучаемых тем дисциплины. При подготовке к экзамену нужно освоить теорию: разобрать определения всех понятий,

рассмотреть примеры и самостоятельно решить несколько типовых задач из каждой темы. При решении задач всегда необходимо комментировать свои действия и не забывать о содержательной интерпретации.

Указания по организации работы с контрольно-измерительными материалами

При подготовке к лабораторной работе необходимо сначала прочитать теорию по каждой теме. Отвечая на поставленный вопрос, предварительно следует понять, что требуется от Вас в данном случае, какой теоретический материал нужно использовать, наметить общий план решения.

VII. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Планирование и организация времени, отведенного на изучение дисциплины. Приступить к освоению дисциплины следует незамедлительно в самом начале учебного семестра. Рекомендуется изучить структуру и основные положения Рабочей программы дисциплины. Обратит внимание, что кроме аудиторной работы (лекции, лабораторные занятия) планируется самостоятельная работа, итоги которой влияют на окончательную оценку по итогам освоения учебной дисциплины. Все задания (аудиторные и самостоятельные) необходимо выполнять и предоставлять на оценку в соответствии с графиком.

В процессе изучения материалов учебного курса предлагаются следующие формы работ: чтение лекций, лабораторные занятия, задания для самостоятельной работы.

Лекционные занятия ориентированы на освещение вводных тем в каждый раздел курса и призваны ориентировать студентов в предлагаемом материале, заложить научные и методологические основы для дальнейшей самостоятельной работы студентов.

Лабораторные занятия акцентированы на наиболее принципиальных и проблемных вопросах курса и призваны стимулировать выработку практических умений.

Особо значимой для профессиональной подготовки студентов является *самостоятельная работа* по курсу. В ходе этой работы студенты отбирают необходимый материал по изучаемому вопросу и анализируют его. Студентам необходимо ознакомиться с основными источниками, без которых невозможно полноценное понимание проблематики курса.

Освоение курса способствует развитию навыков обоснованных и самостоятельных оценок фактов и концепций. Поэтому во всех формах контроля знаний, особенно при сдаче зачета, внимание обращается на понимание проблематики курса, на умение практически применять знания и делать выводы.

Работа с литературой. Рекомендуется использовать различные возможности работы с литературой: фонды научной библиотеки ДВФУ

и электронные библиотеки (<http://www.dvfu.ru/library/>), а также доступные для использования другие научно-библиотечные системы.

Подготовка к экзамену. К сдаче экзамена допускаются обучающиеся, выполнившие все задания (лабораторные, самостоятельные), предусмотренные учебной программой дисциплины, посетившие не менее 85% аудиторных занятий.

VIII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Образовательный процесс по дисциплине проводится в лекционных и компьютерных аудиториях.

Мультимедийная лекционная аудитория (мультимедийный проектор, настенный экран, документ-камера) о. Русский, кампус ДВФУ, корпус 20(D), ауд. D738, D654/D752, D412/D542, D818, D741, D945, D547, D548, D732

Компьютерные классы: (доска, 15 персональных компьютеров) о. Русский, кампус ДВФУ, корпус 20(D), D733, D733а, D734, D734а, D546, D546а, D549а (Кампус ДВФУ), оснащенные компьютерами класса Pentium и мультимедийными (презентационными) системами, с подключением к общекорпоративной компьютерной сети ДВФУ и сети Интернет.

IX. ФОНДЫ ОЦЕНОЧНЫХ СРЕДСТВ

В соответствии с требованиями ФГОС ВО для аттестации обучающихся на соответствие их персональных достижений планируемым результатам обучения по дисциплине созданы фонды оценочных средств:

| № п/п | Контролируемые разделы дисциплины (результаты по разделам) | Код контролируемой компетенции/планируемые результаты обучения | Наименование оценочного средства |
|-------|--|---|---|
| 1 | Раздел 1. | ПК-6/ Знать информацию о новейших научных и технологических достижениях. Уметь: осуществлять целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках. Владеть: навыками целенаправленного поиска информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках ПК-7/ Знать основы составления планов работы с учетом ресурсов. Уметь: составлять и контролировать план выполняемой работы. Владеть: способностью составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы. | Письменный ответ на экзамене (программная реализация) |

| | | | |
|---|-----------|--|---|
| 2 | Раздел 2. | <p>ПК-6/ Знать информацию о новейших научных и технологических достижениях.</p> <p>Уметь: осуществлять целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках.</p> <p>Владеть: навыками целенаправленного поиска информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках</p> <p>ПК-7/ Знать основы составления планов работы с учетом ресурсов.</p> <p>Уметь: составлять и контролировать план выполняемой работы.</p> <p>Владеть: способностью составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы.</p> | Письменный ответ на экзамене (программная реализация) |
| 3 | Раздел 3 | <p>ПК-6/ Знать информацию о новейших научных и технологических достижениях.</p> <p>Уметь: осуществлять целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках.</p> <p>Владеть: навыками целенаправленного поиска информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках</p> <p>ПК-7/ Знать основы составления планов работы с учетом ресурсов.</p> <p>Уметь: составлять и контролировать план выполняемой работы.</p> <p>Владеть: способностью составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы.</p> | Письменный ответ на экзамене (программная реализация) |
| 4 | Раздел 4. | <p>ПК-6/ Знать информацию о новейших научных и технологических достижениях.</p> <p>Уметь: осуществлять целенаправленный поиск информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках.</p> <p>Владеть: навыками целенаправленного поиска информации о новейших научных и технологических достижениях в информационно-телекоммуникационной сети "Интернет" и в других источниках</p> <p>ПК-7/ Знать основы составления планов работы с учетом ресурсов.</p> <p>Уметь: составлять и контролировать план выполняемой работы.</p> <p>Владеть: способностью составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы.</p> | Письменный ответ на экзамене (программная реализация) |

Описание показателей и критериев оценивания:

| Оценка | Неудовлетворительно | Удовлетворительно | Хорошо | Отлично |
|----------------------------|---------------------|-------------------|-------------------|-----------------|
| Набранная сумма баллов (%) | Менее 3 (Менее 50%) | 3-3,5 (50- 69%) | 3,6 -4,4 (70-84%) | 4,5-5 (85-100%) |

| | | | | |
|---|---------------------|-------------------|--|--|
| выполненных заданий) (макс – 5) | | | | |
| Оценка | Незачет | Зачет | | |
| Набранная сумма баллов (% выполненных заданий) (макс – 5) | Менее 3 (Менее 50%) | 3,1 – 5 (50-100%) | | |

Зачетно-экзаменационные материалы

1. ООП в C++. Структуры и классы. Множественное наследование. Виртуальное наследование. Таблица виртуальных функций. Абстрактные классы. Liskov substitution. NVI
2. Управление памятью в C++. Выравнивание полей структур. Битовые поля в структурах. sizeof. Выделение памяти (operator new, placement new). new vs malloc. POD. Фрагментация памяти. Счетчик ссылок. Умные указатели. Аллокаторы
3. Const correctness. Константные указатели, ссылки, методы, переменные, замыкания. Mutable. Стандартная библиотека C++. Ввод-вывод. Контейнеры. Типы контейнеров. Типы итераторов.
4. Обработка исключений в c++. Модули стандартной библиотеки для работы с исключениями. Exception safety. RAII. Виды гарантий exception safety. Раскрутка стека. std::uncaught_exception (s). Исключения в многопоточном приложении C++.
5. Шаблоны в C++. Вложенные шаблоны. Различия в шаблонах функций и классов. Инстанцирование шаблона. Специализация шаблона. SFINAE.
6. Метапрограммирование. type_traits. Variadic templates. Списки типов. Tag dispatch (by instance, by type, на примере iterator_traits).
7. Функции в C++. Способы передачи функции в качестве аргумента. Указатель на функцию. Функторы. std::function. Замыкания и анонимные функции в C++.
8. Многопоточность в C++. Race condition. Способы синхронизации. Dead lock, live lock. Lock-free. Модуль atomic. Conditional variable. Future. Promise
9. Правила вывода типов в C++. decltype, auto, decltype(auto). Универсальные ссылки. Move-семантика. Perfect forwarding. Реализация std::move, std::forward. Value categories.

X.