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

УТВЕРЖДЕНО
Директор института нано-, био-, информационных, когнитивных и социогуманитарных наук и технологий
Т.Е. Григорьев

Рабочая программа дисциплины (модуля)

по дисциплине: Технологии программирования
по направлению: Прикладные математика и физика
профиль подготовки: Термоядерная энергетика и плазменные технологии
Физтех-школа природоподобных, плазменных и ядерных технологий им. И.В. Курчатова
кафедра информатики и вычислительных сетей
курс: 3
квалификация: бакалавр

Семестры, формы промежуточной аттестации:
5 (осенний) - Экзамен
6 (весенний) - Дифференцированный зачет

Аудиторных часов: 90 всего, в том числе:
лекции: 30 час.
семинары: 0 час.
лабораторные занятия: 60 час.

Самостоятельная работа: 105 час.

Подготовка к экзамену: 30 час.

Всего часов: 225, всего зач. ед.: 5

Программу составил: П.С. Сорокоумов, старший преподаватель

Программа обсуждена на заседании кафедры информатики и вычислительных сетей 20.03.2024

Аннотация

Цель данного курса - ознакомление с современными технологиями программирования, позволяющими давать адекватное масштабируемое описание систем высокой сложности. Акцент при изложении материала сделан на тех особенностях языков программирования, которые обеспечивают создание гибких настраиваемых интерфейсов, то есть высокую модульность получаемых приложений.

Объектно-ориентированное программирование описано на примере двух возникших в разное время подходов - как непосредственное развитие ранее известных низкоуровневых процедурных средств в языке C++ и как набор высокоуровневых, рассчитанных на широкие круги программистов средств в языке Python. Это позволяет одновременно и обучать использованию высокоуровневых средств объектно-ориентированного программирования (виртуальных функций, кодогенерации, менеджеров ресурсов), и анализировать их устройство, возможности и ограничения. В рамках практических занятий отрабатывается не только разработка программ отдельно на C++ и Python, но и их совместное использование путём создания библиотеки динамического связывания, что соответствует сложившейся практике применения данных языков. В курсе рассмотрены также некоторые теоретические проблемы объектной модели, описанные в терминах языка Smalltalk.

Высокоуровневые подходы к описанию предметной области – функциональный и логический - позволяют создавать короткие и ёмкие модели, отличающиеся широтой охвата и глубиной детализации. Традиционно применяемый в логическом программировании язык Prolog до сих пор активно используется в задачах представления знаний и анализа данных и, кроме того, составляет базу для изучения многих других методов этих предметных областей. Изучение языка Haskell даёт большой опыт применения высокоуровневых средств (типизации, ленивых вычислений, функций высших порядков) для решения таких низкоуровневых задач, как оптимизация приложений или их адаптация к ограничениям вычислительной системы. Это позволяет затронуть всегда актуальные вопросы сложности алгоритмов и переносимости программ. Практические занятия позволяют учащимся закрепить полученные знания и выработать навык самостоятельной разработки программ высокой сложности.

1. Цели и задачи

Цель дисциплины

- изучение общих принципов программирования сложных систем, методов выбора типа программной модели системы, получение навыков программирования при создании сложных гетерогенных программных комплексов.

Задачи дисциплины

- изучение основных принципов организации сложных программных систем;
- освоение распространённых приемов и методов программирования, а также методов их сочетания и решения возникающих при этом проблем;
- изучение технологий создания программ на разных языках программирования с использованием разных представлений предметных областей;
- получение навыков решения научных и практических задач с использованием методов императивного, функционального, объектно-ориентированного программирования.

2. Перечень формируемых компетенций

Освоение дисциплины направлено на формирование следующих компетенций:

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-2 Способен использовать современные информационные технологии и программные средства при решении задач профессиональной деятельности, соблюдая требования информационной безопасности	ОПК-2.1 Способен применять современные вычислительную технику и сервисы сети Интернет в области (сфере) профессиональной деятельности
	ОПК-2.2 Знает и умеет применять численные математические методы и прикладное программное обеспечение для решения научных задач в профессиональной области
	ОПК-2.3 Знает основные требования информационной безопасности

3. Перечень планируемых результатов обучения по дисциплине (модулю)

В результате освоения дисциплины обучающиеся должны

знать:

- основные принципы организации сложных программных систем.

уметь:

- применять приемы и методы программирования в своей практической деятельности;
- создавать сложные программные системы;
- обоснованно выбирать наиболее подходящие типы моделей разрабатываемых систем.

владеть:

- методологией и навыками решения научных и практических задач с использованием объектно-ориентированного, функционального и логического программирования.

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	Основные понятия ООП	5		15	30
2	Модульная организация программ	10		15	30
3	Логическое программирование	5		15	20
4	Функциональное программирование	10		15	25
Итого часов		30		60	105
Подготовка к экзамену		30 час.			
Общая трудоёмкость		225 час., 5 зач.ед.			

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

Семестр: 5 (Осенний)

1. Основные понятия ООП

Основы языков C++ и Python: базовый синтаксис, модульность, системы типов, трансляция. Автоматизация тестирования, таймирования, проверки стиля программ.

Основные понятия объектно-ориентированного программирования. Классы и объекты в C++ и Python.

Шаблоны в C++: понятие статического полиморфизма, реализация, стандартные шаблоны из STL. Стандартные контейнеры в Python.

Наследование в C++ и Python: определение, виртуальные функции, множественное наследование, метаклассы в Python. Обработка ошибок.

Работа с ресурсами: ручное управление ресурсами в C++, умные указатели, алгоритмы сборки мусора.

2. Модульная организация программ

Паттерны проектирования в ООП: типы, применение, примеры (итератор, действие, стратегия, фасад, фабрика, синглтон, посетитель, адаптер).

Ввод-вывод: средства стандартных библиотек, обработка ошибок, асинхронный ввод-вывод.

Пользовательский интерфейс: виды, требования, методы обработки событий.

История развития ООП: предпосылки появления, язык Smalltalk. Перспективы ООП.

Семестр: 6 (Весенний)

3. Логическое программирование

Основы логического программирования на языке Prolog: базовый синтаксис фактов, правил, запросов; алгоритм обработки запросов.

Организация программ на языке Prolog: отладка программ, генерация и обработка ошибок; специфика рекурсии в логических языках; разбор примеров.

4. Функциональное программирование

Теоретические концепции функционального программирования: лямбда-исчисление, основы теории типов.

Основы языка Haskell: базовый синтаксис, типы, организация программных модулей, сборка и отладка программ.

Исполнение программы в Haskell: понятие ленивых вычислений, бесконечные структуры данных, рекурсия.

Типизация в функциональных языках: вывод типов, составные типы данных и их обработка, виды полиморфизма, классы типов.

Обработка данных в функциональных языках: преобразование списков с помощью map и fold, генерация списков.

Ввод-вывод в функциональных языках: состояние, побочные эффекты, монады.

Введение в теорию категорий: определение категории, коммутативные диаграммы, морфизмы, функторы. Применение теории категорий в анализе методов обработки информации.

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

1. Язык программирования C++ (стандарт C++11) : Краткий курс [Текст] = A Tour of C++, [учеб. пособие для вузов] /Бьерн Страуструп ; пер. с англ. под ред. Н. Н. Мартынова. -М., БИНОМ, 2017

Фонд литературы кафедры

2. Алгоритмы искусственного интеллекта на языке PROLOG /И. Братко , пер. с англ. и ред. К. А. Птицина, PROLOG Programming for Artificial Intelligence, -М. ; СПб. ; Киев, Вильямс, 2004

3. У. Курт. Программируй на Haskell / пер. с англ. Я. О. Касюлевича, А. А. Романовского и С. Д. Степаненко; под ред. В. Н. Брагилевского. – М.: ДМК Пресс, 2019. — 648 с.: ил. ISBN 978-5-97060-694-0

Дополнительная литература

Фонд литературы кафедры

1. Программирование на Python /М. Лутц , -СПб., Символ-Плюс, 2002
2. Стренг Г., Фикс Дж. Теория метода конечных элементов. Под редакцией Марчука Г.И., М.: «МИР», 1977, 350 с.
3. Strang G. Computational Science and Engineering, Wellesley-Cambridge Press, 2007.
4. D.S. Watkins. Fundamentals of Matrix Computations. John Wiley and Sons Inc., 2002

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

. http://lib.mipt.ru– электронная библиотека Физтеха.

2. http://www.Sci-lib.com – Большая научная библиотека.

3. ocw.mit.edu – Собрание он-лайн курсов по различным дисциплинам, читаемых в MIT.

4. http://arXiv.org– CornellUniversityLibrary – Библиотека Корнельского Университета, электронный ресурс arXiv.

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

На лекционных занятиях демонстрируются презентации с помощью мультимедийных технологий.

В процессе самостоятельной работы обучающиеся могут использовать программные средства:

1. Bjarne Stroustrup's FAQ: http://www.stroustrup.com/bs_faq.html
2. C++ FQA: https://yosefk.com/c++fqa/
3. Документация языка Python 3: https://docs.python.org/3/
4. Haskell wiki: https://wiki.haskell.org/Haskell

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

Для успешного освоения курса, помимо посещения лекций, от студентов требуется самостоятельная работа в объеме не менее чем те часы, которые указаны для каждого раздела программы. В основном, это время отводится на самостоятельное решение задач, входящих в два задания. Студенты, успешно прошедшие все формы промежуточного контроля, допускаются к сдаче экзамена в 5 семестре и дифференцированного зачета в 6 семестре по дисциплине.

ПРИЛОЖЕНИЕ

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

по направлению: Прикладные математика и физика
профиль подготовки: Термоядерная энергетика и плазменные технологии
Физтех-школа природоподобных, плазменных и ядерных технологий им. И.В. Курчатова
кафедра информатики и вычислительных сетей
курс: 3
квалификация: бакалавр

Семестры, формы промежуточной аттестации:
5 (осенний) - Экзамен
6 (весенний) - Дифференцированный зачет

Разработчик: П.С. Сорокоумов, старший преподаватель

1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
ОПК-2 Способен использовать современные информационные технологии и программные средства при решении задач профессиональной деятельности, соблюдая требования информационной безопасности	ОПК-2.1 Способен применять современные вычислительную технику и сервисы сети Интернет в области (сфере) профессиональной деятельности
	ОПК-2.2 Знает и умеет применять численные математические методы и прикладное программное обеспечение для решения научных задач в профессиональной области
	ОПК-2.3 Знает основные требования информационной безопасности

2. Показатели оценивания компетенций

В результате изучения дисциплины «Технологии программирования» обучающийся должен:

знать:

- основные принципы организации сложных программных систем.

уметь:

- применять приемы и методы программирования в своей практической деятельности;
- создавать сложные программные системы;
- обоснованно выбирать наиболее подходящие типы моделей разрабатываемых систем.

владеть:

- методологией и навыками решения научных и практических задач с использованием объектно-ориентированного, функционального и логического программирования.

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

В целях текущего контроля успеваемости предусмотрен краткий опрос по темам предыдущих занятий по теме прошлой лекции или в конце занятия по пройденной теме.

3. Перечень типовых контрольных заданий, используемых для оценки знаний, умений, навыков

Промежуточная аттестация по дисциплине «Технологии программирования» осуществляется в форме экзамена в 5 семестре и в форме дифференцированного зачета в 6 семестре. **Экзамен и дифференцированный зачет** проводятся в устной форме.

Пример задания на лабораторные занятия

Вариант 1. Карта окрестностей робота

Задание

1. Спроектировать, реализовать, протестировать и отладить первый из указанных необходимых классов отдельно от остальных.
2. Спроектировать иерархию классов.
3. Спроектировать, реализовать и протестировать контейнерный класс и классы иерархии, выполнив при необходимости рефакторинг ранее реализованного класса.
4. Провести рефакторинг, заменив (один или несколько пунктов по согласованию с преподавателем):

- часть реализованных вручную контейнеров на имеющиеся в STL;
- часть использованных алгоритмов на реализации из стандартной библиотеки;
- часть реализованных вручную контейнеров на самостоятельно выполненные шаблонные реализации

Описание приложения

Приложение используется для хранения информации об окрестностях автономного робота. Карта содержит описания объектов-целей исследования для этого робота и пунктов подзарядки для него.

Необходимые классы

Цель исследования: объект содержит название цели (строка), её координаты (2 дробных числа), срок устаревания (целое число минут). Необходимо регистрировать перенос цели в другое место и менять её срок устаревания.

Пункт подзарядки: объект содержит напряжение питания (целое), состояние (включен-выключен) и координаты. Необходимо включать/выключать пункт подзарядки.

Карта: kd-дерево, содержащее объекты карты и текущие координаты робота. Должна допускать регистрацию новых объектов, их перемещение, получение списка ближайших к роботу объектов и позиции ближайшего к нему включенного источника питания с заданным напряжением.

Перечень контрольных вопросов для сдачи экзамена в 5 семестре:

1. Транслятор C++ - компилятор или интерпретатор? А транслятор Python?
2. Приведите три примера ошибок, возникающих на этапе компиляции программы в C++.
3. Приведите три примера ошибок, возникающих на этапе связывания (linking) программы.
4. Зачем нужно охватывать текст заголовочных файлов C++ директивами #ifndef / #define / #endif или (не всегда переносимо) #pragma once?
5. Что произойдёт, если два заголовочных файла C++ вставить с помощью #include друг в друга?
6. Что произойдёт, если два файла Python попытаться импортировать друг в друга?
7. Пусть нам хочется объявить в заголовочном файле a.h структуру struct a, использующую struct b*, а в заголовочном файле b.h --- struct b, использующую struct a*. Возможно ли это?

8. Может ли конструктор класса в C++ не иметь параметров? А в Python?
9. Может ли конструктор класса в C++ быть private? Если да, то зачем такое может понадобиться?
10. Может ли у класса в C++ быть два конструктора? А два деструктора?
11. Может ли у класса в Python быть два конструктора?
12. Может ли конструктор некоторой переменной в C++ запуститься до функции main?
13. Зачем нужно скрывать часть класса с помощью private?
14. Обеспечивает ли C++ защиту информации, хранимой в private-полях класса, от несанкционированного чтения злоумышленником?
15. В каких методах класса C++ невозможно использовать указатель this?
16. Можно ли в C++ перегрузить сложение для двух переменных типа int? Для двух переменных разных типов?
17. Можно ли в Python перегрузить сложение для двух объектов разных классов? А индексирование ("a[i]")?
18. Является ли класс в C++ объектом? Если да, то какого класса?
19. Является ли класс в Python объектом? Если да, то какого класса?
20. Может ли класс быть наследником самого себя?
21. Может ли класс быть наследником не класса, а встроенного типа данных (int, char и т.д.)?
22. Можно ли запретить наследование от определённого класса?
23. Может ли наследник не иметь новых полей класса (по сравнению с базовым классом), а только новые методы?
24. Почему создание protected полей класса считается плохой практикой (в отличие от protected методов)?
25. Могут ли у базового класса быть какие-либо public-методы, которые не являются public-методами наследника? Почему?
26. Может ли наследник поменять область видимости перегружаемой им функции (например, вместо protected сделать её public)?
27. Может ли перегруженный метод возвращать значение иного типа, чем метод базового класса?
28. Можно ли запретить перегрузку метода?
29. Пусть некоторый метод базового класса перегружен в наследнике. Можно ли вызвать этот метод базового класса для объекта класса-наследника?
30. Почему вызов виртуальной функции выполняется медленнее, чем невиртуальной?
31. Наследуются ли конструкторы и деструкторы?
32. Наследуются ли перегруженные операторы (присваивание, сравнение)?
33. Зачем деструктор при построении иерархии классов делать виртуальным?
34. Нужно ли делать деструктор базового класса иерархии виртуальным, если никакие классы иерархии не работают с ресурсами (динамически выделяемой памятью, файлами и т. п.)?
35. Почему по указателю на объект базового класса может находиться объект производного класса, но не наоборот?
36. Может ли класс-наследник не перегружать виртуальные функции базового класса? А чисто виртуальные функции базового класса?
37. Есть ли у генерации кода с помощью шаблона преимущества перед «генерацией кода» с помощью макроопределений в стиле C (#define)?
38. Если шаблон функции ни разу не был вызван в коде, будет ли сгенерирован код по нему?
39. Если один из методов шаблона класса ни разу не был вызван в коде, будет ли сгенерирован код по нему?

40. Можно ли передать шаблон в качестве параметра шаблона?

41. Может ли класс быть наследником шаблона класса? Может ли шаблон класса быть наследником класса?

42. Какой стандартный контейнер STL предназначен для хранения упорядоченных элементов, не допуская их дублирования (т.е. в нём не могут встречаться совпадающие элементы)?

43. Какой стандартный контейнер STL предназначен для хранения данных с обеспечением доступа по ним по дисциплине FIFO? LIFO? Любой из этих дисциплин по выбору?

44. Что произойдёт при попытке доступа к элементу vector по индексу, выходящему за пределы вектора?

45. Что произойдёт при попытке доступа к элементу map по индексу, не встречающемуся в map?

46. Может ли существовать vector < vector <int> > ? map < int , vector <int> > ? map < Rectangle , int > (Rectangle - класс прямоугольников)?

47. Если функция принимает как параметр значение типа const vector <const int* >, можно ли передать ей значение типа vector < int* > и почему?

48. Может ли итератор, указывающий на элемент списка list, стать невалидным (т.е. перестать указывать на элемент контейнера) в результате вызова какого-либо метода контейнера, не удаляющего этот элемент?

49. Может ли итератор, указывающий на элемент массива vector, стать невалидным (т.е. перестать указывать на элемент контейнера) в результате вызова какого-либо метода контейнера, не удаляющего этот элемент?

50. Является ли Python интерпретируемым или компилируемым языком?

51. Можно ли преобразовать код на Python в код на компилируемом языке? Зачем такое может понадобиться?

52. Почему в C++ есть ограничение на размер целого числа встроенных типов, а в Python --- нет?

53. Как распознать переполнение переменной при выполнении целочисленных арифметических операций в C? В C++? В Python?

54. Почему интерпретатор Python может автоматически освобождать ненужные ресурсы, а в C++ об этом должен заботиться программист?

55. Что произойдёт, если обратиться к отсутствующему элементу списка Python? К элементу с отрицательным индексом?

56. Что произойдёт, если в одной программе Python будут использованы разные типы отступов для одного уровня вложенности конструкций?

57. На какие сообщения должны реагировать объекты-классы Smalltalk?

58. Можно ли в Smalltalk добавлять новые управляющие конструкции? А в C? В C++? В Python?

59. Можно ли в Smalltalk создать собственный метакласс и что для этого нужно?

60. Зачем статические поля класса в C++ нужно не только объявлять в определении класса, но и отдельно определять в какой-либо одной единице трансляции?

61. Выполняется ли dynamic_cast быстрее или медленнее преобразования типов в стиле C?

62. Является ли с точки зрения ООП круг разновидностью эллипса?

63. Является ли с точки зрения ООП эллипс разновидностью круга?

64. Чем отличается виртуальное множественное наследование от невиртуального?

65. Можно ли по исключению C++ определить, в каком месте кода оно было брошено?

66. Почему при использовании исключений в C++ могут возникнуть утечки памяти?

67. Зачем может понадобиться использовать не стандартный аллокатор std:allocator, а какой-то другой? Как указать, какой аллокатор использовать в создаваемом vector?

68. Какие проблемы могут возникнуть при присваивании объекта самому себе (a = a;)? Как их можно избежать?

69. Какие шаблоны проектирования в ООП позволяют дополнять интерфейс класса новыми возможностями без явной модификации собственного кода класса?

70. Почему стратегии в C++ часто реализуют в виде наследования от параметров шаблона, а не обычного наследования?

Пример экзаменационных билетов

Билет №1

1. Является ли с точки зрения ООП круг разновидностью эллипса?
2. Приведите три примера ошибок, возникающих на этапе связывания (linking) программы.

Билет №2

1. Какой стандартный контейнер STL предназначен для хранения данных с обеспечением доступа по ним по дисциплине FIFO? LIFO? Любой из этих дисциплин по выбору?
2. Если функция принимает как параметр значение типа const vector <const int* >, можно ли передать ей значение типа vector < int* > и почему?

Перечень контрольных вопросов для сдачи дифференцированного зачета в 6 семестре:

1. Как можно вывести информацию о стеке вызовов функций Prolog на экран?
2. Какие методы избежания зацикливания в Prolog могут в свою очередь привести к зацикливанию?
3. Является ли система типов Prolog строгой или нестрогой? Явной или неявной?
4. Имеется ли наследование в Prolog?
5. Что такое хвостовая рекурсия? Как можно использовать её для оптимизации программы?
6. Какие типы тестов выделяют по ширине охвата компонентов тестируемой системы?
7. В чём отличие тестирования от отладки?
8. Что произойдёт, если malloc не может выделить запрошенное количество памяти? Как проверить при вызове, выделена память или нет?
9. Что произойдёт, если realloc не может выделить запрошенное количество памяти? Как проверить при вызове, выделена память или нет?
10. Что произойдёт, если вызвать free или delete с нулевым параметром (NULL)? А если передать им указатель, который не указывает на ранее выделенную память из кучи?
11. Что произойдёт, если new не может выделить запрошенное количество памяти? Как проверить при вызове, выделена память или нет?
12. Как распознать проблемы при работе с памятью в Prolog? В Haskell?
13. Как средствами Haskell реализовать поведение, аналогичное наследованию от класса-интерфейса в C++?

14. Чем в плане оптимизации отличаются прямая и косвенная рекурсия? Какой дополнительный механизм должен быть добавлен в язык для развёртывания косвенных рекурсивных вызовов в циклы?
15. Можно ли средствами C++ запретить копирование объекта? Присваивание объекта?
16. Почему монада в Haskell не определяется явно как состояние вычислительного процесса?
17. Как в трансляторе Prolog решены проблемы копирования объектов, типичные для C++?
18. Какой механизм используется в языке Haskell для представления состояния?
19. Почему в Haskell чистые функции выделяются среди прочих?
20. Может ли доступная программе память исчерпаться из-за слишком большого стека вызовов?
21. Пусть в системе свободно 50Мб оперативной памяти. Может ли malloc, пытающийся выделить 10Мб, закончиться неудачей?
22. Почему явное обращение к RTTI (dynamic_cast, typeid) считается плохой практикой?
23. Какие средства языка обеспечивают рефлексию в Python? В C++? В Haskell? В Prolog?
24. Есль ли преимущества у функции, возвращающей генератор значений, перед функцией, возвращающей список значений?
25. Какой механизм обычно используется в языке Prolog вместо присваивания?
26. Что означает RAII применительно к оперативной памяти?
27. Как применить концепцию RAII к другим видам ресурсов, помимо памяти?
28. Как связаны применение RAII и безопасность обработки исключений?
29. Почему при использовании нетривиальных конструкторов либо нетривиальных перегруженных операторов необходимо применять RAII?
30. Приведите примеры преобразований структур данных, обладающих свойствами катаморфизма, анаморфизма, хиломорфизма, параморфизма.
31. Приведите примеры функций Haskell, обладающих свойствами катаморфизма, анаморфизма, хиломорфизма, параморфизма.
32. Почему в C++ нет единого вида умного указателя, а есть много разных его вариантов? Как Python, Prolog и Haskell обходятся без умных указателей?
33. Почему исключения в Python рекомендуют использовать всем и всегда, а в C++ это гораздо более сложный вопрос?
34. Как можно реализовать в Prolog поведение, аналогичное исключениям?
35. Нуждаются ли Prolog или Haskell в умных указателях для управления ресурсами?
36. Как ограничить глубину стека вызовов в C++? в Python? в Prolog? в Haskell?
37. Можно ли указать функции Python значение параметра по умолчанию? Можно ли передать параметры не в порядке указания? А в C++?
38. Чем отличаются типизации в C++ и Python, Prolog, Haskell?
39. Как Haskell, Smalltalk, Python и Prolog обходятся без явно выделенного типа указателей?
40. Можно ли поместить по умному указателю unique_ptr массив объектов? а по shared_ptr?
41. Как weak_ptr позволяет избежать зацикливания цепочек shared_ptr?
42. Какие проблемы могут возникнуть при использовании умных указателей в многопоточной среде?
43. Почему при работе с бесконечными структурами данных Haskell не происходит переполнение памяти?

44. Какие сложности возникают при попытке использовать сборщик мусора в C++?

45. Почему при сборке мусора на основе подсчёта ссылок могут возникнуть утечки памяти?

46. Можно ли в C++ на этапе компиляции определить, имеются ли в программе утечки памяти или нет? Есть ли в других языках методы, позволяющие это сделать?

47. Можно ли создать в C++ указатель, не указывающий ни на какой выделенный ранее участок памяти?

48. Можно ли создать в C++ ссылку, не ссылающуюся ни на какую выделенную ранее переменную?

49. Можно ли в Python создать указатель на участок памяти? А ссылку на переменную?

50. Можно ли в Prolog или Haskell создать невалидную ссылку на участок памяти? Какой модуль обеспечивает небезопасные операции с памятью в Haskell?

51. Что произойдёт, если в C или C++ приравнять указатель произвольному числу и записать в память, находящуюся по этому указателю, какое-либо значение?

52. Чем сходны и чем отличаются возможности классов типов Haskell и наследования ООП в отношении повторного использования кода?

53. vtable - по сути просто глобальная переменная, своя для каждого класса иерархии - точно так же, как и статическое поле класса. Компилятор создаёт vtable автоматически и неявно для программиста. Почему он не может точно так же автоматически создать статическое поле класса, а требует явно определять его после определения класса?

54. Может ли vtable какого-либо класса не содержать указатели на методы этого класса, а только на методы предков? А на методы потомков?

55. Используются ли vtable в Haskell? В Prolog?

56. Почему механизм наследования в базовой версии Prolog не реализован?

57. Какие средства поддержки полиморфизма присутствуют в C++? в Python? В Prolog? В Haskell?

4. Критерии оценивания

Оценка	Баллы	Критерии
отлично	10	Выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины, проявляющему интерес к данной предметной области, продемонстрировавшему умение уверенно и творчески применять их на практике при решении конкретных задач, свободное и правильное обоснование принятых решений.
	9	Выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины и умение уверенно применять их на практике при решении конкретных задач, свободное и правильное обоснование принятых решений.
	8	Выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы дисциплины и умение уверенно применять их на практике при

		решении конкретных задач, правильное обоснование принятых решений, с некоторыми недочетами.
хорошо	7	Выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но недостаточно грамотно обосновывает полученные результаты.
	6	Выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но допускает в ответе или в решении задач некоторые неточности.
	5	Выставляется студенту, если он в основном знает материал, грамотно и по существу излагает его, умеет применять полученные знания на практике, но допускает в ответе или в решении задач достаточно большое количество неточностей.
удовлетворительно	4	Выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, недостаточно правильные формулировки базовых понятий, нарушения логической последовательности в изложении программного материала, но при этом он освоил основные разделы учебной программы, необходимые для дальнейшего обучения, и может применять полученные знания по образцу в стандартной ситуации.
	3	Выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, допускающему ошибки в формулировках базовых понятий, нарушения логической последовательности в изложении программного материала, слабо владеет основными разделами учебной программы, необходимыми для дальнейшего обучения и с трудом применяет полученные знания даже в стандартной ситуации.
неудовлетворительно	2	Выставляется студенту, который не знает большей части основного содержания учебной программы дисциплины, допускает грубые ошибки в формулировках основных принципов и не умеет использовать полученные знания при решении типовых задач.
	1	Выставляется студенту, который не знает основного содержания учебной программы дисциплины, допускает грубейшие ошибки в формулировках базовых понятий дисциплины и

		вообще не имеет навыков решения типовых практических задач.

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

При проведении экзамена обучающемуся предоставляется 40 минут на подготовку. Опрос обучающегося по билету на экзамене не должен превышать двух астрономических часов.

Во время проведения экзамена обучающиеся могут пользоваться программой дисциплины, конспектами лекций, а также любой справочной литературой.

При проведении дифференцированного зачета обучающемуся предоставляется не менее 40 минут на подготовку. Опрос по билету и ответы на дополнительные вопросы не должны превышать двух астрономических часов. По завершении отведенного на опрос времени, экзаменатор должен выставить обучающемуся экзаменационную оценку.