



1920

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Филиал федерального государственного бюджетного образовательного учреждения высшего образования

«Кубанский государственный университет» в г. Геленджике

УТВЕРЖДАЮ:

И. о. проректора по учебной
работе, качеству образования –
первого проректора

Т. А. Хагуров

17 мая 2025 г.



Рабочая программа дисциплины

МДК.01.01 «Разработка программных модулей»

09.02.07 «Информационные системы и программирование»

2025

Рабочая программа МДК.01.01 «Разработка программных модулей» разработана на основе федерального государственного образовательного стандарта среднего профессионального образования (ФГОС СПО) по специальности 09.02.07 «Информационные системы и программирование», утвержденного приказом Министерства образования и науки от 9 декабря 2016 года № 1547 (зарегистрирован Министерством юстиции Российской Федерации 26 декабря 2016г., регистрационный №44936) (далее – ФГОС СПО).

Дисциплина МДК.01.01 «Разработка программных модулей»

Форма обучения	очная
Учебный год	2025-2026
2,3 курсы	4, 5, 6 семестр
Лекции	174 час.
Практические занятия	150 час.
Консультации	7 час.
Экзамен	12 час.
форма итогового контроля	дифференцированный зачёт, экзамен

Составитель: преподаватель

Поддубная Е.А.

Утверждена на заседании предметной (цикловой) комиссии профессиональных дисциплин программирования в компьютерных системах

Протокол № 10 от 27 мая 2025 г.

Председатель предметной (цикловой) комиссии профессиональных дисциплин специальности **09.02.07 «Информационные системы и программирование» и 09.02.03 «Программирование в компьютерных системах»**



Л.А. Благова

подпись

Рецензенты:

Директор ООО «Современные
информационные технологии»



А.В.Сметанин

Заместитель директора по научной работе филиала ФГБОУ ВО «КубГУ» в г.
Геленджике



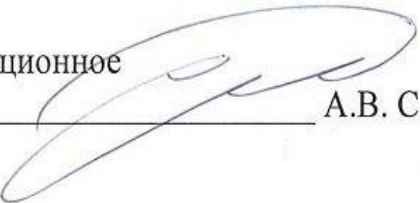
Л.В. Галицкая, к.т.н.

ЛИСТ
согласования рабочей учебной программы по дисциплине
МДК.01.01 «Разработка программных модулей»
Специальность среднего профессионального образования:
09.02.03 Программирование в компьютерных системах

СОГЛАСОВАНО:

Заместитель директора по УР филиала _____  Т.А. Резуненко

Заведующая сектором библиотеки филиала _____  Л.Г. Соколова

Инженер-электроник (программно-информационное
обеспечение образовательной программы) _____  А.В. Сметанин

СОДЕРЖАНИЕ

1. ПАСПОРТ РАБОЧЕЙ ПРОГРАММЫ УЧЕБНОЙ ДИСЦИПЛИНЫ	4
1.1. Область применения программы	4
1.2. Место учебной дисциплины в структуре программы подготовки специалистов среднего звена	4
1.3. Цели и задачи учебной дисциплины - требования к результатам освоения дисциплины	4
1.4. Перечень планируемых результатов обучения по дисциплине (перечень формируемых компетенций)	5
2. СТРУКТУРА И СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ	9
2.1. Объем учебной дисциплины и виды учебной работы	9
2.2. Структура дисциплины:	9
2.3. Тематический план и содержание учебной дисциплины	10
2.4. Содержание разделов дисциплины	12
2.4.1. Занятия лекционного типа	12
2.4.2. Занятия семинарского типа	12
2.4.3. Практические (лабораторные) занятия	12
2.4.4. Содержание самостоятельной работы	13
2.4.5. Содержание учебно-методического обеспечения для самостоятельной работы	14
3. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ	16
3.1. Образовательные технологии при проведении лекция	16
3.2. Образовательные технологии при проведении практических занятий	16
4. УСЛОВИЯ РЕАЛИЗАЦИИ ПРОГРАММЫ ДИСЦИПЛИНЫ	18
4.1. Материально-техническая база, необходимая для осуществления образовательного процесса по дисциплине	18
4.2. Перечень необходимого программного обеспечения	18
5. ПЕРЕЧЕНЬ ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ УЧЕБНОЙ ЛИТЕРАТУРЫ, НЕОБХОДИМОЙ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ	21
5.1. Основная литература	21
5.2. Дополнительная литература.....	22
5.3. Периодические издания.....	23
5.4. Перечень ресурсов информационно-телекоммуникационной сети Интернет, необходимых для освоения дисциплины	
6. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ	25
7. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ КОНТРОЛЯ УСПЕВАЕМОСТИ	30
7.1. Паспорт фонда оценочных средств	30
7.2. Критерии оценки знаний	30
7.3. Оценочные средства для проведения текущей аттестации	31
7.4. Оценочные средства для проведения промежуточной аттестации	32
7.4.1. Примерные вопросы для проведения промежуточной аттестации	32
7.4.2. Примерные задачи для проведения промежуточной аттестации	40
8. ДОПОЛНИТЕЛЬНОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ	51

1 ПАСПОРТ РАБОЧЕЙ ПРОГРАММЫ УЧЕБНОЙ ДИСЦИПЛИНЫ

МДК.01.02 «Поддержка и тестирование программных модулей»

1.1. Область применения программы

Рабочая программа **МДК.01.01 «Разработка программных модулей»** является частью программы подготовки специалистов среднего звена в соответствии с ФГОС СПО по специальности 09.02.07. «Информационные системы и программирование»

1.2. Место учебной дисциплины в структуре программы подготовки специалистов среднего звена

Учебная дисциплина **МДК.01.01 «Разработка программных модулей»** принадлежит профессиональному модулю ПМ.01. **«Разработка модулей программного обеспечения для компьютерных систем»**. Она обеспечивает профессиональный уровень подготовки специалиста и соответствует развитию их профессионально значимых качеств.

Для освоения дисциплины обучающиеся используют знания, умения и навыки, сформированные при изучении МДК.01.01 «Разработка программных модулей» и ОП.04 «Основы алгоритмизации и программирования».

1.3 Цель и планируемые результаты освоения дисциплины:

Для основного вида деятельности **«Разработка модулей программного обеспечения для компьютерных систем»** осваиваются частично:

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием

ПК.1.3. Выполнять отладку программных модулей с использованием специализированных программных средств

ПК 1.4. Выполнять тестирование программных модулей

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода

«Осуществление интеграции программных модулей» осваиваются частично компетенции

ПК 2.1. «Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент»,

ПК 2.3. «Выполнять отладку программного модуля с использованием специализированных программных средств»,

ПК 4.2. Осуществлять измерения эксплуатационных характеристик программного обеспечения компьютерных систем

Максимальная учебная нагрузка обучающегося – 343 часа, в том числе

- обязательная аудиторная нагрузка – 324 часа;
- самостоятельная работа обучающегося – не предусмотрено
- консультации – 7 часа;
- экзамен – 12 часов

1.4. Перечень планируемых результатов обучения по дисциплине (перечень формируемых компетенций)

В результате освоения учебной дисциплины обучающийся должен

уметь:

- применять логические операции, формулы логики, законы алгебры логики;
- формулировать задачи логического характера и применять средства математической логики для их решения.

В результате освоения учебной дисциплины обучающийся должен

знать:

- основные принципы математической логики, теории множеств и теории алгоритмов;
- формулы алгебры высказываний;
- методы минимизации алгебраических преобразований;
- основы языка и алгебры предикатов;
- основные принципы теории множеств.

В результате освоения учебной дисциплины обучающийся осваивает элементы общих компетенций.

1.4.1. Перечень общих компетенция

Код компетенции	Формулировка компетенции	Знания, умения ⁴
ОК 01	Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам	Умения: распознавать задачу и/или проблему в профессиональном и/или социальном контексте; анализировать задачу и/или проблему и выделять её составные части; определять этапы решения задачи; выявлять и эффективно искать информацию, необходимую для решения задачи и/или проблемы; составить план действия; определить необходимые ресурсы; владеть актуальными методами работы в

		профессиональной и смежных сферах; реализовать составленный план; оценивать результат и последствия своих действий (самостоятельно или с помощью наставника) Знания: актуальный профессиональный и социальный контекст, в котором приходится работать и жить; Основные источники информации и ресурсы для решения задач и проблем в профессиональном и/или социальном контексте; алгоритмы выполнения работ в профессиональной и смежных областях; методы работы в профессиональной и смежных сферах; структуру плана для решения задач; деятельности
ОК 02	Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности	Умения: определять задачи для поиска информации; определять необходимые источники информации; планировать процесс поиска; структурировать получаемую информацию; выделять наиболее значимое в перечне информации; оценивать практическую значимость результатов поиска; оформлять результаты поиска Знания: номенклатура информационных источников, Применяемых в профессиональной деятельности; Приемы структурирования информации; формат оформления результатов поиска информации

ОК 03	Планировать и реализовывать собственное профессиональное и личностное развитие.	Умения: определять актуальность нормативно-правовой документации в профессиональной деятельности; применять современную научную профессиональную терминологию; определять и выстраивать траектории профессионального развития и самообразования Знания: содержание актуальной нормативно-правовой документации; современная научная и профессиональная профессионального развития и самообразования
ОК 04	Работать в коллективе и команде, эффективно взаимодействовать с коллегами, Руководством, клиентов	Умения: организовывать работу коллектива и команды; Взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности Знания: психологические основы деятельности коллектива, психологические особенности личности; основы проектной деятельности

	и	
ОК 05	Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.	Умения: грамотно излагать свои мысли и оформлять документы по профессиональной тематике на государственном языке, проявлять толерантность в рабочем коллективе Знания: особенности социального и культурного контекста; правила оформления документов и построения устных сообщений.
ОК 09	Использовать информационные технологии в профессиональной деятельности	Умения: применять средства информационных технологий для решения профессиональных задач; использовать современное программное обеспечение Знания: современные средства и устройства информатизации; порядок их применения и программное обеспечение в профессиональной деятельности

1.4.2. Перечень профессиональных компетенций

В результате освоения учебной дисциплины обучающийся осваивает элементы профессиональных компетенций: ПК 1.1-1.5, 2.1, 2.3, 2.4, 4.2.

Перечень профессиональных компетенций, элементы которых формируются в рамках учебной дисциплины

Код	Наименование видов деятельности и профессиональных компетенций
ОВД 1	Разработка модулей программного обеспечения для компьютерных систем
ПК 1.1	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием
ПК 1.2	Разрабатывать программные модули в соответствии с техническим заданием
ПК 1.3	Выполнять отладку программных модулей с использованием специализированных программных средств
ПК 1.4	Выполнять тестирование программных модулей
ПК 1.5	Осуществлять рефакторинг и оптимизацию программного кода

В результате освоения МДК 01.01 студент должен:

Иметь практический опыт	Разрабатывать алгоритм решения поставленной задачи и реализовывать его средствами автоматизированного проектирования Разрабатывать код программного продукта на основе готовой спецификации на уровне модуля. Использовать инструментальные средства на этапе отладки программного продукта. Проводить тестирование программного модуля по определенному сценарию Анализировать алгоритмы, в том числе с применением инструментальных средств. Осуществлять рефакторинг и оптимизацию программного кода. Интегрировать модули в программное обеспечение. Отлаживать программные модули. Инспектировать разработанные программные модули на предмет соответствия стандартам кодирования.
уметь	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием. Оформлять документацию на программные средства Создавать программу по разработанному алгоритму как отдельный модуль. Оформлять документацию на программные средства. Выполнять отладку и тестирование программы на уровне модуля. Оформлять документацию на программные средства Выполнять оптимизацию и рефакторинг программного кода. Работать с системой контроля версий.
знать	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием. Оформлять документацию на программные средства Создавать программу по разработанному алгоритму как отдельный модуль. Оформлять документацию на программные средства. Выполнять отладку и тестирование программы на уровне модуля. Оформлять документацию на программные средства Выполнять оптимизацию и рефакторинг программного кода. Работать с системой контроля версий. Основные виды и принципы тестирования программных продуктов Основные этапы разработки программного обеспечения. Основные принципы технологии структурного и объектноориентированного программирования.

В результате освоения учебной дисциплины обучающийся осваивает элементы основных видов деятельности:

Основной вид деятельности	Требования к знаниям, умениям, практическим действиям
ОВД 1. «Разработка модулей программного обеспечения для компьютерных систем»	Указаны выше
ОВД 2. «Осуществление интеграции программных модулей»	Указаны выше
ОВД 4 «Сопровождение иобслуживание программного обеспечения компьютерных систем»	Указаны выше

Технологии формирования ОК

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

Предоставить обучающемуся возможность на практических занятиях для самостоятельного выбора способа решения задачи

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

Предоставить обучающемуся возможность на практических занятиях для самостоятельного выбора способа подготовки информации для решения задачи

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

Предоставить обучающемуся возможность на лекциях и практических занятиях для выражать свою заинтересованность в обучении и реализовывать ее

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

Предоставить обучающемуся возможность на практических занятиях, на учебной и производственной практиках для решения задач в коллективе, помогать в случае затруднений

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

Предоставить обучающемуся возможность на лекциях практических занятиях для общения на государственном языке

ОК 09. Использовать информационные технологии в профессиональной деятельности.

Предоставить обучающимся возможности на практических занятиях выполнять задания средствами ИТ. Предоставлять студентам возможность самостоятельно осуществлять поиск, анализ и оценку информации при выполнении практической и самостоятельной работы

Изучать профессиональные стандарты на государственном языке (русском) и знакомить обучающихся с международными стандартами на английском языке

Технологии формирования ПК

Изучение теории и практики разработки программных модулей, сред разработки, отладки и тестирования и закрепления их на опыте.

2. СТРУКТУРА И СОДЕРЖАНИЕ УЧЕБНОЙ ДИСЦИПЛИНЫ

2.1. Объем учебной дисциплины и виды учебной работы

Вид учебной работы	Объем в часах
Объем образовательной программы	343
в том числе:	
теоретическое обучение (+ текущая аттестация – зачёт)	174
практические занятия	150
консультации	7
Промежуточная аттестация (экзамен)	12

2.2. Структура дисциплины

Наименование разделов и тем	Всего	Количество аудиторных часов		Самостоятельная работа обучающегося (час)
		Теоретическое обучение	Практические и лабораторные занятия	
Раздел 1. Отладка и тестирование программного обеспечения	324	174	150	-
Тема 1.1. Жизненный цикл ПО	2	2	-	-
Тема 1.2. Структурное программирование	50	24	26	-
Тема 1.3. Объектно-ориентированное программирование	104	54	50	-
Тема 1.4. Паттерны проектирования	22	16	6	-
Тема 1.5. Событийно управляемое программирование	80	30	50	-
Тема 1.6. Оптимизация и рефакторинг кода	8	6	2	-
Тема 1.7. Разработка пользовательского интерфейса	8	6	2	-
Тема 1.8. Основы ADO.Net	50	36	14	-
Всего по дисциплине	324	174	150	-

2.3. Тематический план и содержание учебной дисциплины МДК.01.01 «Разработка программных модулей»

Наименование разделов и тем профессионального модуля	Содержание учебного материала, лабораторные работы и практические занятия, самостоятельная учебная работа обучающихся, курсовая работа (проект)	Уровень освоения	ПК
1	2		
Раздел 1. Разработка программных модулей			

МДК.01.01 Разработка программных модулей			
Тема 1.1. Жизненный цикл ПО	Содержание		
	Л1. Понятие ЖЦ ПО. Этапы ЖЦ ПО. Модели ЖЦ ПО: каскадная, инкрементная, спиральная модели [Понятие алгоритма. Виды алгоритмов. Способы записи алгоритмов)	2-3	ПК.1.1 – ПК.1.5
	В том числе, практических занятий и лабораторных работ		
Тема 1.2. Структурное программирование	Содержание		
	Л12. Технология структурного программирования. Основные этапы решения задач на ЭВМ. Структурное программирование и алгоритмизация. Теорема структуризации. [Система программирования: состав и функции]	2-3	ПК.1.1 – ПК.1.5
	Л13. Разработка сверху вниз и снизу вверх. Основные принципы технологии структурного программирования. Методы и приемы алгоритмизации поставленных задач [Понятие переменной. Виды данных. Алфавит ЯП С#]	2-3	ПК.1.1 – ПК.1.5
	Л14. Инструментальные средства оформления и документирования алгоритмов программ. Определение процессов предметной области. Процессы управления проектами. Технология быстрой разработки приложений. [Операторы ЯП С#]	2-3	ПК.1.1 – ПК.1.5
	Л15. Методология, технология и инструментальные средства разработки прикладного программного обеспечения. RAD и CASE-средства. Задачи и функции инструментального программного обеспечения [Простые алгоритмы. Массивы]	2-3	ПК.1.1 – ПК.1.5
	Л16. Простые методы сортировки. [Алгоритмы решения типовых задач, области и способы их применения.]	2-3	ПК.1.1 – ПК.1.5
	Л17. Сортировка вставками, выбором, быстрая	2-3	ПК.1.1 –
	Л18. Алгоритмы сложной сортировки. Сортировка пузырьком. Сортировка перемешиванием (шейкерная сортировка). Сортировка расчёской.	2-3	ПК.1.1 – ПК.1.5
	Л19. Алгоритмы решения типовых задач, области и способы их применения	2-3	ПК.1.1 – ПК.1.5
	Л110. Алгоритмы поиска данных. Последовательный поиск. Индексно-последовательный поиск. Бинарный поиск.	2-3	ПК.1.1 – ПК.1.5
	Л111. Рекурсивные и эвристические алгоритмы. Понятие рекурсии и рекурсивной функции. Эвристика. Применение эвристического алгоритма. Пример оценки эвристического решения	2-3	ПК.1.1 – ПК.1.5
	Л112. Алгоритмы решения типовых задач области и способы их применения	2-3	ПК.1.1 – ПК.1.5
	Л113. Оценка сложности алгоритма. Классификация, классы алгоритмов, неразрешимые задачи. Методы и приемы формализации задач	2-3	ПК.1.1 – ПК.1.5
	В том числе, практических и лабораторных занятий по отработке умений: - использовать методы и приемы алгоритмизации поставленных задач;		
	<i>Пр 1. Применение циклических алгоритмов</i>	3	ПК.1.1 – ПК.1.5

Пр 2. Оценка сложности циклических алгоритмов	3	ПК.1.1 –
Пр 3. Применение вложенных циклов	3	ПК.1.1 –
Пр 4. Оценка сложности вложенных циклов	3	ПК.1.1 –
Пр 5. Применение алгоритмов сортировки в программе	3	ПК.1.1 –
Пр 6. Оценка сложности алгоритмов сортировки	3	ПК.1.1 –
Пр 7. Применение алгоритмов поиска данных в программе	3	ПК.1.1 – ПК.1.5
Пр 8. Разработка алгоритмов с использованием подпрограмм	3	ПК.1.1 – ПК.1.5

	Пр 9. Оценка сложности алгоритмов поиска	3	ПК.1.1 – ПК.1.5
	Пр 10. Применение рекурсивных алгоритмов в программе	3	ПК.1.1 – ПК.1.5
	Пр 11. Оценка сложности рекурсивных алгоритмов	3	ПК.1.1 – ПК.1.5
	Пр 12. Применение эвристических алгоритмов в программе	3	ПК.1.1 – ПК.1.5
	Пр 13. Оценка сложности эвристических алгоритмов	3	ПК.1.1 – ПК.1.5
Тема 1.3. Объектно-ориентированное программирование	Содержание		
	Л14. Основные этапы и принципы процесса разработки программного обеспечения. Процесс разработки. Цикл разработки.	2-3	ПК.1.1 – ПК.1.5
	Л 15. Языки программирования и среды разработки, средства пакетного выполнения процедур	2-3	ПК.1.1 – ПК.1.5
	Л16. Основы программирования на объектно-ориентированном языке C#. Вид программы на языке C#. Простые типы данных. Оператор присваивания. Понятие арифметического выражения. Ввод данных с клавиатуры и вывод на экран. Форматный вывод.	2-3	ПК.1.1 – ПК.1.5
	Л17. Основные структуры данных. Синтаксис языка программирования C# , особенности программирования на этом языке, стандартные библиотеки языка программирования	2-3	ПК.1.1 – ПК.1.5
	Л18. Особенности использования данных целого и вещественного типа. Правила согласования типов. Проблема переноса программы с одного языка программирования на другой.	2-3	ПК.1.1 – ПК.1.5
	Л19. Операторы разветвления алгоритма. Логические выражения. Условный оператор if. Условная операция. Оператор выбора (варианта, переключения) switch. Оператор разрыва break. Оператор безусловного перехода goto.	2-3	ПК.1.1 – ПК.1.5
	Л20. Операторы циклов. Оператор цикла с параметром (for). Оператор цикла с предусловием (while). Оператор цикла с постусловием (do while). Вспомогательные операторы, управляющие работой цикла: оператор разрыва break и оператор продолжения continue.	2-3	ПК.1.1 – ПК.1.5

Л21. Одномерные массивы: определение и описание массива, формирование массивов исходных данных, особенности вывода одномерных массивов. Символьный массив	2-3	ПК.1.1 – ПК.1.5
Л22. Двумерные (многомерные) массивы: определение двумерных массивов, особенности формирования двумерных массивов, особенности вывода и обработки двумерных массивов.	2-3	ПК.1.1 – ПК.1.5
Л23. Использование подпрограмм. Понятие подпрограммы. Структура процедур и функций. Обращение к подпрограмме. Механизм передачи параметров. Рекурсивные процедуры и функции.	2-3	ПК.1.1 – ПК.1.5
Л24. Строковые данные. Функция для работы со строками. Перенос символов между строкой и символьным массивом.	2-3	ПК.1.1 – ПК.1.5
Л25. Другие способы организации данных. Коллекции. Списки. Понятие коллекции и списка. Способы определения и описания коллекции и списка.	2-3	ПК.1.1 – ПК.1.5
Л26. Структурированные данные (структуры, struct).	2-3	ПК.1.1 –
Л27. Запись и чтение данных из файла. Методы организации файловых систем. Запись и чтение строк. Прямой доступ к файлу.	2-3	ПК.1.1 – ПК.1.5
Л28. Понятие классов. Классы: основные понятия. Применение классов.	2-3	ПК.1.1 – ПК.1.5
Л29. Принципы ООП: наследование, инкапсуляция, полиморфизм	2-3	ПК.1.1 – ПК.1.5
Л30. Перегрузка методов. Понятие метода и сигнатуры. Способы применения перегрузки методов	2-3	ПК.1.1 – ПК.1.5

Л31. Операции класса. Понятие операции класса. Секции операции. Параметры и выражения. Кванторы видимости	2-3	ПК.1.1 – ПК.1.5
Л32. Иерархия классов. Базовые и производные классы. Модификаторы доступа. Конструкторы и деструкторы. Примеры реализации классов	2-3	ПК.1.1 – ПК.1.5
Л33. Синтаксис интерфейсов. Определение интерфейса. Применение интерфейсов. Явная реализация и реализация в базовых и производных классах.	2-3	ПК.1.1 – ПК.1.5
Экзамен за 4 семестр		
5 семестр		
Л34. Интерфейсы и наследование. Наследование интерфейсов. Ковариантность и контравариантность обобщенных интерфейсов	2-3	ПК.1.1 – ПК.1.5
Л35. Структуры. Определение структуры. Конструкторы структуры. Применение структур в программе	2-3	ПК.1.1 – ПК.1.5
Л36. Делегаты. Определение делегатов. Присвоение ссылки на метод. Соответствие методов делегату. Добавление методов в делегат. Объединение делегатов. Вызов делегата. Делегаты как параметры методов. Обобщенные делегаты. Применение делегатов	2-3	ПК.1.1 – ПК.1.5

Л37. Регулярные выражения. Классы StringBuilder и String. Параметр RegexOptions. Синтаксис регулярных выражений. Проверка на соответствие строки формату. Замена и метод Replace	2-3	ПК.1.1 – ПК.1.5
Л38. Коллекции. Параметризованные классы. Понятие коллекции. ArrayList. Список List. Двухсвязный список LinkedList. Очередь Queue. Стек Stack. Словарь Dictionary. Класс ObservableCollection. Интерфейсы Enumerable и Enumerator. Итераторы и оператор yield	2-3	ПК.1.1 – ПК.1.5
Л39. Указатели. Определение указателя. Операции с указателями. Арифметика указателей. Константы и указатели. Указатели и массивы. Указатели в параметрах функции. Массивы в параметрах функции. Указатели на функции. Указатели на функции как параметры. Указатель на функцию как возвращаемое значение. Динамические объекты и массивы	2-3	ПК.1.1 – ПК.1.5
Л40. Операции со списками. Класс List. Двухсвязный список. Добавление и удаление элементов. Получение индекса элемента и сортировка списков	2-3	ПК.1.1 – ПК.1.5
В том числе, практических и лабораторных занятий по отработке умений: - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных;	2-3	
Пр 14. Изучение интерфейса среды разработки программ	3	ПК.1.1 – ПК.1.5
Пр 15. Особенности визуального компонента Форма	3	ПК.1.1 – ПК.1.5
Пр 16. Особенности использования визуальных и невидимых компонентов	3	ПК.1.1 – ПК.1.5
Пр 17. Проектирование простого консольного приложения с использованием выбранной среды программирования Разработка простого консольного приложения с использованием выбранной среды программирования	3	ПК.1.1 – ПК.1.5
Пр18. Разработка простого консольного приложения с использованием выбранной среды программирования	3	ПК.1.1 – ПК.1.5
Пр 19. Проектирование консольных приложений и применение операторов ветвления с применением стандартных алгоритмов в соответствующих областях	3	ПК.1.1 – ПК.1.5
Пр 20. Разработка консольных приложений и применение операторов ветвления с применением стандартных алгоритмов в соответствующих областях	3	ПК.1.1 – ПК.1.5
Пр 21. Разработка консольных приложений и применение циклических операторов	3	ПК.1.1 – ПК.1.5
Пр 22. Разработка консольных приложений по работе с одномерными массивами	3	ПК.1.1 – ПК.1.5

	ПЗ 23. Разработка консольных приложений по работе с двумерными массивами	3	ПК.1.1 – ПК.1.5
	ПЗ 24. Разработка консольных приложений и применение подпрограмм	3	ПК.1.1 – ПК.1.5
	ПЗ 25. Работа со строками в консольных приложениях с применением стандартных алгоритмов в соответствующих областях	3	ПК.1.1 – ПК.1.5
	ПЗ 26. Работа с коллекциями, списками и структурами в консольных приложениях	3	ПК.1.1 – ПК.1.5
	ПЗ 27. Запись и чтение данных из файла с помощью консольных приложений	3	ПК.1.1 – ПК.1.5
	ПЗ 28. Работа с классами	3	ПК.1.1 – ПК.1.5
	ПЗ 29. Работа с методами	3	ПК.1.1 – ПК.1.5
	ПЗ 30. Перегрузка методов	3	ПК.1.1 – ПК.1.5
	ПЗ 31. Определение операций в классе	3	ПК.1.1 – ПК.1.5
	ПЗ 32. Создание наследованных классов	3	ПК.1.1 – ПК.1.5
	ПЗ 33. Работа с объектами через интерфейсы	3	ПК.1.1 – ПК.1.5
	ПЗ 34. Использование стандартных интерфейсов	3	ПК.1.1 – ПК.1.5
	ПЗ 35. Работа с типом данных структура	3	ПК.1.1 – ПК.1.5
	ПЗ 36. Коллекции. Параметризованные классы	3	ПК.1.1 – ПК.1.5
	ПЗ 37. Использование регулярных выражений	3	ПК.1.1 – ПК.1.5
	ПЗ 38. Операции со списками	3	ПК.1.1 – ПК.1.5
Тема 1.4. Паттерны проектирования	Содержание		
	Л41. Назначение и виды паттернов. Основы паттернов проектирования. Введение в паттерны проектирования.	2-3	ПК.1.1 – ПК.1.5
	Л42. Отношения между классами и объектами. Интерфейсы или абстрактные классы	2-3	ПК.1.1 – ПК.1.5
	Л43. Основные шаблоны. Взаимодействие между классами или объектами. Идиомы. Шаблон делегирования. Шаблон функционального дизайна. Неизменяемый интерфейс. Интерфейс-маркер. Контейнер свойств. Канал событий	2-3	ПК.1.1 – ПК.1.5
	Л44. Порождающие шаблоны. Фабричный метод (Factory Method). Абстрактная фабрика (Abstract Factory). Одиночка (Singleton). Прототип (Prototype). Строитель (Builder)	2-3	ПК.1.1 – ПК.1.5
	Л45. Структурные шаблоны. Декоратор (Decorator). Адаптер (Adapter). Фасад (Facade). Компонировщик (Composite). Заместитель (Прокси). Мост (Bridge). Приспособленец (Flyweight)	2-3	ПК.1.1 – ПК.1.5
	Л46. Поведенческие шаблоны. Стратегия (Strategy). Наблюдатель (Observer). Команда (Command).	2-3	ПК.1.1 – ПК.1.5
	Л47. Шаблонный метод (Template Method). Итератор (Iterator). Состояние (State). Цепочка Обязанностей (Chain of responsibility). Интерпретатор (Interpreter). Посредник (Mediator). Хранитель (Memento). Посетитель (Visitor)	2-3	ПК.1.1 – ПК.1.5
	Л48. Принцип единственной обязанности. Принципы SOLID. Принцип единственной обязанности. Принцип открытости/закрытости. Принцип подстановки Лисков. Принцип разделения интерфейсов. Принцип инверсии зависимостей	2-3	ПК.1.1 – ПК.1.5

	В том числе, практических и лабораторных занятий по отработке умений: - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных;		
--	---	--	--

	- применять выбранные языки программирования для написания программного кода; - применять стандартные алгоритмы в соответствующих областях; применять языки программирования, определенные в техническом задании на разработку системы управления базами данных, для написания программного кода		
	<i>Пр 39. Использование основных и порождающих шаблонов</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 40. Использование структурных шаблонов</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 41. Разработка приложения с использованием шаблонов</i>	3	ПК.1.1 – ПК.1.5
Тема 1.5.	Содержание		
Событийно-управляемое программирование	Л49. Событийно-управляемое программирование. Основные характеристики языка. Структура программы. Визуальные компоненты .NET Framework.	2-3	ПК.1.1 – ПК.1.5
	Л50. Методы и средства сборки модулей и компонент программного обеспечения	2-3	ПК.1.1 – ПК.1.5
	Л51. Технологии разработки приложений WPF (Windows Presentation Foundation) Создание приложений при помощи визуальных компонентов. Создание формы. Компонент MainWindow как основной контейнер для всех визуальных компонентов.	2-3	ПК.1.1 – ПК.1.5
	Л52. События и свойства. Основные визуальные компоненты программирования: MainWindow, Button, Label, TextBox, ComboBox.	2-3	ПК.1.1 – ПК.1.5
	Л53. Элементы управления. Диалоговые окна. Обработчики событий. Присваивание и вывод результатов при помощи компонента Label. Ввод данных и вывод результатов. Условный операор. Условный оператор с двумя ветвями. Условный оператор с операторными скобками. Вложенные условные операторы.	2-3	ПК.1.1 – ПК.1.5
	6 семестр		
	Л54. Свойства объекта MainWindow	2-3	ПК.1.1 – ПК.1.5
	Л55. Способы создания групп компонентов. Создание группы «радиокнопок» (radioButton) и «чек-кнопок» (checkBox). Компонент TrackBar. Компонент ComboBox. Компонент ListBox. Компонент GroupBox. Перезапись строк из разных компонентов. Изменение свойств текста. Оператор переключения (выбора). Выбор, содержащий более одного оператора.	2-3	ПК.1.1 – ПК.1.5
	Л56. Методы формирования меню. Компонент menuStrip. Использование компонента contextMenuStrip. Оформление меню.	2-3	ПК.1.1 – ПК.1.5
	Л57. Обработка дополнительных форм. Добавление дополнительных окон и страниц в проект. Обработка диалоговых окон. Окна сообщений (MessageBox). Вызов диалоговых окон.	2-3	ПК.1.1 – ПК.1.5

Л58. Применение вкладок на форме. Компонент tabControl. Разработка программы с помощью вкладок. Отдельная вкладка как индивидуальный контейнер компонентов.	2-3	ПК.1.1 – ПК.1.5
Л59. Компоненты диапазона значений. Компонент trackBar. Компонент progressBar. Настройка и применение компонентов trackBar и progressBar.	2-3	ПК.1.1 – ПК.1.5
Л60. Работа с компонентом вывода табличных данных. Вывод	2-3	ПК.1.1 – ПК.1.5
Л61. Создание базы данных и привязка для последующего вывода и редактирования БД с помощью компонента dataGridView. Компонент DataSet	2-3	ПК.1.1 – ПК.1.5
Л62. Введение в графику. Рисование простых графических объектов. Изменение масштаба. Вращение в плоскости. Рисование трехмерных фигур.	2-3	ПК.1.1 – ПК.1.5
Л63. Применение в программировании прочих компонентов. Компонент панели (panel). Применение таймера в визуальном программировании (timer). Применение компонента ListView. Ввод «по маске», в том числе паролей, с использованием MaskedTextBox. Использование календаря MonthCalendar. Встраивание компонента WebBrowser.	2-3	ПК.1.1 – ПК.1.5

В том числе, практических и лабораторных занятий по отработке умений:36 - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных; - применять выбранные языки программирования для		
<i>Пр 42. Основы языка ZAML и его применение для среды разработки Visual Studio</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 43. Изучение свойств базовых инструментов технологии Wpf</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 44. Создание событий для базовых инструментов</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 45. Проектирование приложения с использованием текстовых компонентов</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 46. Разработка приложения с использованием текстовых компонентов</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 47. Проектирование приложения с несколькими формами</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 48. Разработка приложения с несколькими формами</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 49. Проектирование приложения с не визуальными компонентами</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 50. Разработка приложения с не визуальными компонентами</i>	2	ПК.1.1 – ПК.1.5
<i>Пр 51. Разработка алгоритма игрового приложения</i>	2	ПК.1.1 – ПК.1.5
<i>Пр 52. Разработка интерфейса игрового приложения</i>	2	ПК.1.1 – ПК.1.5
<i>Пр 53. Разработка кода игрового приложения</i>	2	ПК.1.1 – ПК.1.5
<i>Пр 54. Разработка алгоритма, интерфейса и кода приложения с анимацией</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 55. Создание приложения Windows Forms и организация ввода/вывода</i>	3	ПК.1.1 – ПК.1.5

	<i>Пр 56. Работа с радиокнопками, чек-кнопками и компонентом GroupBox</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 57. Работа с компонентами ComboBox и ListBox</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 58. Разработка главного меню программы с помощью компонента MenuStrip</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 59. Работа с контекстным меню (contextMenuStrip)</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 60. Работа с диалоговыми формами и вывод сообщений</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 61. Работа с вкладками tabControl</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 62. Работа с компонентами trackBar и ProgressBar</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 63. Работа с базами данных на форме с помощью компонента dataGridView</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 64. Рисование простых графических объектов на форме</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 65. Рисование трехмерных объектов на форме</i>	3	ПК.1.1 – ПК.1.5
	<i>Пр 66. Применение в программе прочих визуальных компонентов</i>	3	ПК.1.1 – ПК.1.5
Тема 1.6. Оптимизация и рефакторинг кода	Содержание		
	Л64. Методы оптимизации программного кода. Основные принципы оптимизации: естественность, производительность, время. Участки кода, которые не оптимизируются. Настройка окружения. Избавление от ненужного функционала. Мемоизация. Кеширование.	2-3	ПК.1.1 – ПК.1.5
	Л65. Распараллеливание программ. Способы оптимизации	2-3	ПК.1.1 – ПК.1.5
	Л66. Цели и методы рефакторинга. Понятие рефакторинга (перепроектирования) кода. Причины применения рефакторинга. Признаки плохого кода. Методы и приемы рефакторинга. Проблемы, возникающие при применении рефакторинга. Средства автоматизации рефакторинга.	2-3	ПК.1.1 – ПК.1.5
	В том числе, практических и лабораторных занятий по отработке умений:		

	<i>Пр 67. Оптимизация и рефакторинг кода</i>		ПК.1.1 –
Тема 1.7. Разработка пользовательского интерфейса	Содержание	8	
	Л67. Принципы разработки графического интерфейса пользователя. Понятие интерфейса пользователя. GUI. UI-дизайн. Необходимость разработки UI-дизайна. Разработка UI-интерфейсов.	2-3	ПК.1.1 – ПК.1.5
	Л68. Проектирование. Создание дизайн-макета. Карта экранов (UFD). Утверждение структуры и согласование стиля	2-3	ПК.1.1 – ПК.1.5
	Л69. Правила разработки интерфейсов пользователя. Главная задача интерфейса пользователя. Ключевое назначение интерфейсов. Акценты и их расстановка. Взаимодействия пользователя с интерфейсом. Основные принципы и правила разработки пользовательских интерфейсов.	2-3	ПК.1.1 – ПК.1.5
	В том числе, практических и лабораторных занятий по отработке умений: - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных.	2-3	ПК.1.1 – ПК.1.5
	<i>Пр №68. Разработка интерфейса пользователя</i>		ПК.1.1 – ПК.1.5
Тема 1.8.	Содержание		

Л70. Основы работы с MS SQL Server. Установка MS SQL Server. Установка SQL Server Management Studio. Установка LocalDB.	2-3	ПК.1.1 – ПК.1.5
Л71. Начало работы с MS SQL Server. Создание базы данных. Создание таблиц. Первый запрос на T-SQL	2-3	ПК.1.1 – ПК.1.5
Л72. Работа с базами данных. Понятие и структура базы данных (БД) Основы интерфейса взаимодействия с базами данных в ADO.NET. Провайдеры данных	2-3	ПК.1.1 – ПК.1.5
Л73. Доступ к данным. Создание базы данных. Строка подключения. Создание подключения. Пул подключений. Методы и средства миграции и преобразования данных	2-3	ПК.1.1 – ПК.1.5
Л74. Операторы DDL. Создание и удаление базы данных. Создание и удаление таблиц. Типы данных T-SQL. Атрибуты и ограничения столбцов и таблиц. Внешние ключи. Изменение таблицы. Пакеты. Команда GO	2-3	ПК.1.1 – ПК.1.5
Л75. Создание таблицы, работа с записями. Команды создания и удаления таблиц. Обновление, добавление, изменение и удаление записей в таблицах БД.	2-3	ПК.1.1 – ПК.1.5
Л76. Операторы DML. Добавление данных. Команда INSERT. Выборка данных. Команда SELECT.	2-3	ПК.1.1 – ПК.1.5
Л 77. Сортировка. ORDER BY. Извлечение диапазона строк. Фильтрация. WHERE. Операторы фильтрации. Обновление данных. Команда UPDATE. Удаление данных. Команда DELETE	2-3	ПК.1.1 – ПК.1.5
Л78. Способы создания команд. Выполнение команд и SqlCommand. Чтение результатов запроса и SqlDataReader. Типизация результатов SqlDataReader. Получение скалярных значений.	2-3	ПК.1.1 – ПК.1.5
Л79. Параметризация запросов. Выходные параметры	2-3	ПК.1.1 – ПК.1.5
Л80. Хранимые процедуры и транзакции. Работа с хранимыми процедурами. Выходные параметры хранимых процедур.	2-3	ПК.1.1 – ПК.1.5
Л 81. Транзакции. Сохранение и извлечение файлов из базы данных	2-3	ПК.1.1 – ПК.1.5
Л82. Методы группировки и соединения таблиц. Агрегатные функции. Операторы GROUP BY и HAVING. Расширения SQL Server для группировки.	2-3	ПК.1.1 – ПК.1.5
Л83. Неявное соединение таблиц. Inner Join. Outer Join. Группировка в соединениях. UNION. EXCEPT. INTERSECT	2-3	ПК.1.1 – ПК.1.5
Л84. Принципы работы с компонентами SqlDataAdapter и DataSet. SqlDataAdapter и DataSet. Постраничный просмотр в SqlDataAdapter. SqlCommandBuilder и сохранение изменений DataSet в базе данных.	2-3	ПК.1.1 – ПК.1.5

Л85. Обновление БД из DataSet вручную. Все операции с БД в графическом приложении. DataSet и DataTable. Отношения между таблицами в DataSet. LINQ to DataSet. DataSet и XML	2-3	ПК.1.1 – ПК.1.5
--	-----	-----------------

Л86. Методы преобразования LINQ в SQL. Определение контекста данных и моделей. Операции с данными в LINQ to SQL. Изменение объектов в LINQ to SQL. Добавление в LINQ to SQL. Удаление в LINQ to SQL.	2-3	ПК.1.1 – ПК.1.5
Л 87. Методы ExecuteCommand и ExecuteQuery. Хранимые процедуры	2-3	ПК.1.1 – ПК.1.5
В том числе, практических и лабораторных занятий по отработке умений:6 - использовать выбранную среду программирования и средства системы управления базами данных - осуществлять разработку кода программного модуля на языках низкого и высокого уровней.		
<i>Пр 69. Разработка алгоритма приложения с БД с использованием выбранных средств системы управления базами данных</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 70. Создание интерфейса и кода приложения с БД</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 71. Создание приложения с БД</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 72. Создание простых запросов к БД</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 73. Создание сложных запросов к БД</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 74. Создание отчётов о состоянии БД</i>	3	ПК.1.1 – ПК.1.5
<i>Пр 75. Создание хранимых процедур</i>	3	ПК.1.1 – ПК.1.5

2.4. Содержание разделов дисциплины

2.4.1. Занятия лекционного типа

		Уровень освоения	ПК
Раздел 1. Разработка программных модулей			
МДК.01.01 Разработка программных модулей			
	Л1. Понятие ЖЦ ПО. Этапы ЖЦ ПО. Модели ЖЦ ПО:	2-3	ПК.1.1-ПК.1.5
Тема 1.2. Структурное программирование	Л2. Технология структурного программирования. Основные этапы решения задач на ЭВМ. Структурное программирование	2-3	ПК.1.1-ПК.1.5
	Л3. Разработка сверху вниз и снизу вверх. Основные принципы технологии структурного программирования.	2-3	ПК.1.1-ПК.1.5
	Л4. Инструментальные средства оформления и документирования алгоритмов программ. Определение	2-3	ПК.1.1-ПК.1.5
	Л5. Методология, технология и инструментальные средства разработки прикладного программного обеспечения. RAD и	2-3	ПК.1.1-ПК.1.5
	Л6. Простые методы сортировки	2-3	ПК.1.1-ПК.1.5
	Л7. Сортировка вставками, выбором, быстрая сортировка,	2-3	ПК.1.1-ПК.1.5
	Л8. Алгоритмы сложной сортировки. Сортировка пузырьком. Сортировка перемешиванием (шейкерная сортировка).	2-3	ПК.1.1-ПК.1.5
	Л9. Алгоритмы решения типовых задач, области и способы	2-3	ПК.1.1-ПК.1.5
	Л10. Алгоритмы поиска данных. Последовательный поиск.	2-3	ПК.1.1-ПК.1.5
	Л11. Рекурсивные и эвристические алгоритмы. Понятие рекурсии и рекурсивной функции. Эвристика. Применение	2-3	ПК.1.1-ПК.1.5
	Л12. Алгоритмы решения типовых задач области и способы	2-3	ПК.1.1-ПК.1.5
	Л13. Оценка сложности алгоритма. Классификация, классы алгоритмов, неразрешимые задачи. Методы и приемы	2-3	ПК.1.1-ПК.1.5
Тема 1.3. Объектно-ориентированное программирование	Содержание		
	Л14. Основные этапы и принципы процесса разработки программного обеспечения. Процесс разработки. Цикл	2-3	ПК.1.1-ПК.1.5
	Л15. Языки программирования и среды разработки, средства пакетного выполнения процедур	2-3	ПК.1.1-ПК.1.5
	Л16. Основы программирования на объектно-ориентированном языке C#. Вид программы на языке C#.	2-3	ПК.1.1-ПК.1.5
	Л17. Основные структуры данных. Синтаксис языка программирования C#, особенности программирования на	2-3	ПК.1.1-ПК.1.5
	Л18. Особенности использования данных целого и вещественного типа. Правила согласования типов. Проблема	2-3	ПК.1.1-ПК.1.5
	Л19. Операторы разветвления алгоритма. Логические выражения. Условный оператор if. Условная операция.	2-3	ПК.1.1-ПК.1.5
	Л20. Операторы циклов. Оператор цикла с параметром (for). Оператор цикла с предусловием (while). Оператор цикла с	2-3	ПК.1.1-ПК.1.5
	Л21. Одномерные массивы: определение и описание массива, формирование массивов исходных данных, особенности вывода	2-3	ПК.1.1-ПК.1.5
	Л22. Двумерные (многомерные) массивы: определение двумерных массивов, особенности формирования двумерных	2-3	ПК.1.1-ПК.1.5
	Л23. Использование подпрограмм. Понятие подпрограммы. Структура процедур и функций. Обращение к подпрограмме.	2-3	ПК.1.1-ПК.1.5
	Л24. Строковые данные. Функция для работы со строками. Перенос символов между строкой и символьным массивом.	2-3	ПК.1.1-ПК.1.5
	Л25. Другие способы организации данных. Коллекции. Списки. Понятие коллекции и списка. Способы определения и	2-3	ПК.1.1-ПК.1.5

	Л26. Структурированные данные (структуры, struct).	2-3	ПК.1.1-ПК.1.5
	Л27. Запись и чтение данных из файла. Методы организации файловых систем. Запись и чтение строк. Прямой доступ к	2-3	ПК.1.1-ПК.1.5
	Л28. Понятие классов. Классы: основные понятия. Применение классов.	2-3	ПК.1.1-ПК.1.5
	Л29. Принципы ООП: наследование, инкапсуляция, полиморфизм	2-3	ПК.1.1-ПК.1.5
	Л30. Перегрузка методов. Понятие метода и сигнатуры. Способы применения перегрузки методов	2-3	ПК.1.1-ПК.1.5
	Л31. Операции класса. Понятие операции класса. Секции операции. Параметры и выражения. Кванторы видимости	2-3	ПК.1.1-ПК.1.5
	Л32. Иерархия классов. Базовые и производные классы. Модификаторы доступа. Конструкторы и деструкторы.	2-3	ПК.1.1-ПК.1.5
	Л33. Синтаксис интерфейсов. Определение интерфейса. Применение интерфейсов. Явная реализация и реализация в	2-3	ПК.1.1-ПК.1.5
	Л34. Интерфейсы и наследование. Наследование интерфейсов. Ковариантность и контравариантность	2-3	ПК.1.1-ПК.1.5
	Л35. Структуры. Определение структуры. Конструкторы структуры. Применение структур в программе	2-3	ПК.1.1-ПК.1.5
	Л36. Делегаты. Определение делегатов. Присвоение ссылки на метод. Соответствие методов делегату. Добавление методов в	2-3	ПК.1.1-ПК.1.5
	Л37. Регулярные выражения. Классы StringBuilder и String. Параметр RegexOptions. Синтаксис регулярных выражений.	2-3	ПК.1.1-ПК.1.5
	Л38. Коллекции. Параметризованные классы. Понятие коллекции. ArrayList. Список List.	2-3	ПК.1.1-ПК.1.5
	Л39. Указатели. Определение указателя. Операции с указателями. Арифметика указателей. Константы и указатели.	2-3	ПК.1.1-ПК.1.5
	Л40. Операции со списками. Класс List. Двухсвязный список. Добавление и удаление элементов. Получение индекса элемента	2-3	ПК.1.1-ПК.1.5
	Л41. Назначение и виды паттернов. Основы паттернов проектирования. Введение в паттерны проектирования.	2-3	ПК.1.1-ПК.1.5
	Л42. Отношения между классами и объектами. Интерфейсы или абстрактные классы	2-3	ПК.1.1-ПК.1.5
	Л43. Основные шаблоны. Взаимодействие между классами или объектами. Идиомы. Шаблон делегирования. Шаблон	2-3	ПК.1.1-ПК.1.5
	Л44. Порождающие шаблоны. Фабричный метод (Factory Method). Абстрактная фабрика (Abstract Factory). Одиночка	2-3	ПК.1.1-ПК.1.5
	Л45. Структурные шаблоны. Декоратор (Decorator). Адаптер (Adapter). Фасад (Facade). Компонировщик (Composite).	2-3	ПК.1.1-ПК.1.5
	Л46. Поведенческие шаблоны. Стратегия (Strategy). Наблюдатель (Observer). Команда (Command).	2-3	ПК.1.1-ПК.1.5
	Л47. Шаблонный метод (Template Method). Итератор (Iterator). Состояние (State). Цепочка Обязанностей (Chain of	2-3	ПК.1.1-ПК.1.5
Тема 1.4. Паттерны проектирования	Содержание		
	Л41. Назначение и виды паттернов. Основы паттернов проектирования. Введение в паттерны проектирования.	2-3	ПК.1.1-ПК.1.5
	Л42. Отношения между классами и объектами. Интерфейсы или абстрактные классы	2-3	ПК.1.1-ПК.1.5
	Л43. Основные шаблоны. Взаимодействие между классами или объектами. Идиомы. Шаблон делегирования. Шаблон	2-3	ПК.1.1-ПК.1.5
	Л44. Порождающие шаблоны. Фабричный метод (Factory Method). Абстрактная фабрика (Abstract Factory). Одиночка	2-3	ПК.1.1-ПК.1.5

	Л45. Структурные шаблоны. Декоратор (Decorator). Адаптер (Adapter). Фасад (Facade). Компонент (Composite).	2-3	ПК.1.1-ПК.1.5
	Л46. Поведенческие шаблоны. Стратегия (Strategy). Наблюдатель (Observer). Команда (Command).	2-3	ПК.1.1-ПК.1.5
	Л47. Шаблонный метод (Template Method). Итератор (Iterator). Состояние (State). Цепочка Обязанностей (Chain of	2-3	ПК.1.1-ПК.1.5
	Л48. Принцип единственной обязанности. Принципы SOLID. Принцип единственной обязанности. Принцип	2-3	ПК.1.1-ПК.1.5
Тема 1.5. Событийно-управляемое программирование	Содержание		
	Л49. Событийно-управляемое программирование. Основные характеристики языка. Структура программы. Визуальные	2-3	ПК.1.1-ПК.1.5
	Л50. Методы и средства сборки модулей и компонент программного обеспечения	2-3	ПК.1.1-ПК.1.5
	Л51. Технологии разработки приложений Windows Forms. Создание приложений при помощи визуальных компонентов.	2-3	ПК.1.1-ПК.1.5
	Л52. События и свойства. Основные визуальные компоненты программирования: Form, Button, Label, TextBox, ComboBox.	2-3	ПК.1.1-ПК.1.5
	Л53. Элементы управления. Диалоговые окна. Обработчики событий. Присваивание и вывод результатов при помощи	2-3	ПК.1.1-ПК.1.5
	Л54. Свойства объекта Форма	2-3	ПК.1.1-ПК.1.5
	Л55. Способы создания групп компонентов. Создание группы «радиокнопок» (radioButton) и «чек-кнопок» (checkBox). Компонент TrackBar. Компонент ComboBox. Компонент	2-3	ПК.1.1-ПК.1.5
	Л56. Методы формирования меню. Компонент menuStrip. Использование компонента contextMenuStrip. Оформление	2-3	ПК.1.1-ПК.1.5
	Л57. Обработка дополнительных форм. Добавление дополнительных форм Additional в проект. Обработка	2-3	ПК.1.1-ПК.1.5
	Л58. Применение вкладок на форме. Компонент tabControl. Разработка программы с помощью вкладок. Отдельная вкладка	2-3	ПК.1.1-ПК.1.5
	Л59. Компоненты диапазона значений. Компонент trackBar. Компонент progressBar. Настройка и применение компонентов	2-3	ПК.1.1-ПК.1.5
	Л60. Работа с компонентом вывода табличных данных. Вывод данных с помощью компонента dataGridView.	2-3	ПК.1.1-ПК.1.5
	Л61. Создание базы данных и привязка для последующего вывода и редактирования БД с помощью компонента	2-3	ПК.1.1-ПК.1.5
	Л62. Введение в графику. Рисование простых графических объектов. Изменение масштаба. Вращение в плоскости.	2-3	ПК.1.1-ПК.1.5
	Л63. Применение в программировании прочих компонентов. Компонент панели (panel). Применение таймера	2-3	ПК.1.1-ПК.1.5
Тема 1.6. Оптимизация и рефакторинг кода	Содержание		
	Л64. Методы оптимизации программного кода. Основные принципы оптимизации: естественность, производительность,	2-3	ПК.1.1-ПК.1.5
	Л65. Распараллеливание программ. Способы оптимизации	2-3	ПК.1.1-ПК.1.5
	Л66. Цели и методы рефакторинга. Понятие рефакторинга (перепроектирования) кода. Причины применения	2-3	ПК.1.1-ПК.1.5
Тема 1.7. Разработка	Содержание		
	Л67. Принципы разработки графического интерфейса пользователя. Понятие интерфейса пользователя. GUI. UI-	2-3	ПК.1.1-ПК.1.5

пользовательского интерфейса	Л68. Проектирование. Создание дизайн-макета. Карта экранов (UFD). Утверждение структуры и согласование стиля	2-3	ПК.1.1-ПК.1.5
	Л69. Правила разработки интерфейсов пользователя. Главная задача интерфейса пользователя. Ключевое назначение		ПК.1.1-ПК.1.5
Тема 1.8. Основы ADO.Net	Содержание		
	Л70. Основы работы с MS SQL Server. Установка MS SQL Server. Установка SQL Server Management Studio. Установка	2-3	ПК.1.1-ПК.1.5
	Л71. Начало работы с MS SQL Server. Создание базы данных. Создание таблиц. Первый запрос на T-SQL	2-3	ПК.1.1-ПК.1.5
	Л72. Работа с базами данных. Понятие и структура базы данных (БД)	2-3	ПК.1.1-ПК.1.5
	Л73. Доступ к данным. Создание базы данных. Строка подключения. Создание подключения. Пул подключений.	2-3	ПК.1.1-ПК.1.5
	Л74. Операторы DDL. Создание и удаление базы данных. Создание и удаление таблиц. Типы данных T-SQL. Атрибуты и	2-3	ПК.1.1-ПК.1.5
	Л75. Создание таблицы, работа с записями. Команды создания и удаления таблиц. Обновление, добавление,	2-3	ПК.1.1-ПК.1.5
	Л76. Операторы DML. Добавление данных. Команда INSERT. Выборка данных. Команда SELECT.	2-3	ПК.1.1-ПК.1.5
	Л 77. Сортировка. ORDER BY. Извлечение диапазона строк. Фильтрация. WHERE. Операторы фильтрации. Обновление	2-3	ПК.1.1-ПК.1.5
	Л78. Способы создания команд. Выполнение команд и SqlCommand. Чтение результатов запроса и SqlDataReader.	2-3	ПК.1.1-ПК.1.5
	Л79. Параметризация запросов. Выходные параметры запросов	2-3	ПК.1.1-ПК.1.5
	Л80. Хранимые процедуры и транзакции. Работа с хранимыми процедурами. Выходные параметры хранимых	2-3	ПК.1.1-ПК.1.5
	Л 81. Транзакции. Сохранение и извлечение файлов из базы данных	2-3	ПК.1.1-ПК.1.5
	Л82. Методы группировки и соединения таблиц. Агрегатные функции. Операторы GROUP BY и HAVING. Расширения SQL	2-3	ПК.1.1-ПК.1.5
	Л83. Неявное соединение таблиц. Inner Join. Outer Join. Группировка в соединениях. UNION. EXCEPT. INTERSECT	2-3	ПК.1.1-ПК.1.5
	Л84. Принципы работы с компонентами SqlDataAdapter и DataSet. SqlDataAdapter и DataSet.	2-3	ПК.1.1-ПК.1.5
	Л85. Обновление БД из DataSet вручную. Все операции с БД в графическом приложении. DataSet и DataTable. Отношения	2-3	ПК.1.1-ПК.1.5
	Л86. Методы преобразования LINQ в SQL. Определение контекста данных и моделей. Операции с данными в LINQ to	2-3	ПК.1.1-ПК.1.5
	Л 87. Методы ExecuteCommand и ExecuteQuery. Хранимые процедуры	2-3	ПК.1.1-ПК.1.5

2.4.2. Занятия семинарского типа

Не предусмотрено

2.4.3. Практические занятия

Наименование разделов и тем профессионального модуля	Содержание учебного материала, лабораторные работы и практические занятия, самостоятельная учебная работа обучающихся, курсовая работа (проект)	Объем в часах	
1	2	3	
Раздел 1. Разработка программных модулей		324	

МДК.01.01 Разработка программных модулей		324	
	В том числе, практических и лабораторных занятий по отработке умений: - использовать методы и приемы	26	
	<i>Пр 1. Применение циклических алгоритмов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 2. Оценка сложности циклических алгоритмов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 3. Применение вложенных циклов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 4. Оценка сложности вложенных циклов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 5. Применение алгоритмов сортировки в программе</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 6. Оценка сложности алгоритмов сортировки</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 7. Применение алгоритмов поиска данных в программе</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 8. Разработка алгоритмов с использованием подпрограмм</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 9. Оценка сложности алгоритмов поиска</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 10. Применение рекурсивных алгоритмов в программе</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 11. Оценка сложности рекурсивных алгоритмов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 12. Применение эвристических алгоритмов в программе</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 13. Оценка сложности эвристических алгоритмов</i>	3	ПК.1.1-ПК.1.5
	Экзамен за 4 семестр		
	5 семестр		

Тема 1.3. Объектно-ориентированное	В том числе, практических и лабораторных занятий по отработке умений: - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных; ~~- применять различные языки программирования для~~		
	<i>Пр 14. Изучение интерфейса среды разработки программ</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 15. Особенности визуального компонента Форма</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 16. Особенности использования визуальных и</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 17. Проектирование простого консольного приложения с</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 18. Разработка простого консольного приложения с</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 19. Проектирование консольных приложений и</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 20. Разработка консольных приложений и применение</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 21. Разработка консольных приложений и применение</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 22. Разработка консольных приложений по работе с</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 23. Разработка консольных приложений по работе с</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 24. Разработка консольных приложений и применение</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 25. Работа со строками в консольных приложениях с</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 26. Работа с коллекциями, списками и структурами в</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 27. Запись и чтение данных из файла с помощью</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 28. Работа с классами</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 29. Работа с методами</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 30. Перегрузка методов</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 31. Определение операций в классе</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 32. Создание наследованных классов</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 33. Работа с объектами через интерфейсы</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 34. Использование стандартных интерфейсов</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 35. Работа с типом данных структура</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 36. Коллекции. Параметризованные классы</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 37. Использование регулярных выражений</i>	3	ПК.1.1-ПК.1.5
	<i>ПЗ 38. Операции со списками</i>	3	ПК.1.1-ПК.1.5
Тема 1.4. Паттерны проектирования	Содержание В том числе, практических и лабораторных занятий по отработке умений: - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; использовать выбранную среду программирования и средства системы управления		
	<i>Пр 39. Использование основных и порождающих шаблонов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 40. Использование структурных шаблонов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 41. Разработка приложения с использованием шаблонов</i>	3	ПК.1.1-ПК.1.5
Тема 1.5.	Содержание		
	6 семестр		

Событийно-управляемое программирование	В том числе, практических и лабораторных занятий по отработке умени - создавать программу по разработанному алгоритму как отдельный модуль; - осуществлять разработку кода программного модуля на современных языках программирования; - использовать выбранную среду программирования и средства системы управления базами данных; - применять выбранные языки программирования для написания программного кода; - применять стандартные алгоритмы в соответствующих областях; - применять языки программирования, определенные в техническом задании на разработку системы управления базами данных, для написания программного кода		
	<i>Пр 42. Основы языка ZAML и его применение для среды</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 43. Изучение свойств базовых инструментов технологии</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 44. Создание событий для базовых инструментов</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 45. Проектирование приложения с использованием</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 46. Разработка приложения с использованием</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 47. Проектирование приложения с несколькими</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 48. Разработка приложения с несколькими формами</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 49. Проектирование приложения с не визуальными</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 50. Разработка приложения с не визуальными</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 51. Разработка алгоритма игрового приложения</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 52. Разработка интерфейса игрового приложения</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 53. Разработка кода игрового приложения</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 54. Разработка алгоритма, интерфейса и кода</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 55. Создание приложения Windows Forms и организация</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 56. Работа с радиокнопками, чек-кнопками и</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 57. Работа с компонентами ComboBox и ListBox</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 58. Разработка главного меню программы с помощью</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 59. Работа с контекстным меню (contextMenuStrip)</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 60. Работа с диалоговыми формами и вывод сообщений</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 61. Работа с вкладками tabControl</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 62. Работа с компонентами trackBar и ProgressBar</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 63. Работа с базами данных на форме с помощью</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 64. Рисование простых графических объектов на форме</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 65. Рисование трехмерных объектов на форме</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 66. Применение в программе прочих визуальных</i>	3	ПК.1.1-ПК.1.5
Тема 1.6. Оптимизация и рефакторинг кода	Содержание	8	
	В том числе, практических и лабораторных занятий по	2	
	<i>Пр 67. Оптимизация и рефакторинг кода</i>	3	ПК.1.1-ПК.1.5
Тема 1.7. Разработка пользовательского интерфейса	Содержание		
	В том числе, практических и лабораторных занятий по	3	ПК.1.1-ПК.1.5
	<i>Пр №68. Разработка интерфейса пользователя</i>	3	ПК.1.1-ПК.1.5
Тема 1.8.	Содержание		

Основы ADO.Net	В том числе, практических и лабораторных занятий по	14	
	<i>Пр 69. Разработка алгоритма приложения с БД с</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 70. Создание интерфейса и кода приложения с БД</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 71. Создание приложения с БД</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 72. Создание простых запросов к БД</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 73. Создание сложных запросов к БД</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 74. Создание отчётов о состоянии БД</i>	3	ПК.1.1-ПК.1.5
	<i>Пр 75. Создание хранимых процедур</i>	3	ПК.1.1-ПК.1.5

2.4.4. Содержание самостоятельной работы

Не предусмотрено учебным планом

3. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

3.1. Образовательные технологии при проведении лекций

Для реализации компетентностного подхода предусматривается использование в учебном процессе активных и интерактивных форм проведения аудиторных и внеаудиторных занятий с целью формирования и развития профессиональных навыков обучающихся.

Преподавание дисциплины организовано по модульно-блочному принципу.

В процессе преподавания применяются образовательные технологии развития абстрактного, логического и критического мышления. Обязательны компьютерные практикумы по разделам дисциплины, тестирование, тематические презентации, интерактивные технологии.

В учебном процессе наряду с традиционными образовательными технологиями используются компьютерное мультимедийное оборудование и интернет

№	Тема	Виды применяемых образовательных технологий	Кол-во час
1	2	3	4
1	Раздел 1. Разработка программных модулей	МБТ, ИКТ	174
2	Тема 1.1. Жизненный цикл ПО	МБТ, ИКТ	2
3	Тема 1.2. Структурное программирование	МБТ, ИКТ	24
4	Тема 1.3. Объектно-ориентированное программирование	МБТ, ИКТ	54
5	Тема 1.4. Паттерны проектирования	МБТ, ИКТ	16
6	Тема 1.5. Событийно управляемое программирование	МБТ, ИКТ	30
7	Тема 1.6. Оптимизация и рефакторинг кода	МБТ, ИКТ	6
8	Тема 1.7. Разработка пользовательского интерфейса	МБТ, ИКТ	6
9	Тема 1.8. Основы ADO.Net	МБТ, ИКТ	36
Итого по курсу			174
в том числе интерактивное обучение при необходимости*			

3.2. Образовательные технологии при проведении практических занятий (лабораторных работ)

№	Тема занятия	Виды технологий	Часы
1	Раздел 1. Разработка программных модулей	ИКТ	150
2	Тема 1.1. Жизненный цикл ПО	ИКТ	-
3	Тема 1.2. Структурное программирование	ИКТ	26
4	Тема 1.3. Объектноориентированное программирование	ИКТ	50
5	Тема 1.4. Паттерны проектирования	ИКТ	6

6	Тема 1.5. Событийно управляемое программирование	ИКТ	50
7	Тема 1.6. Оптимизация и рефакторинг кода	ИКТ	2
8	Тема 1.7. Разработка пользовательского интерфейса	ИКТ	2
9	Тема 1.8. Основы ADO.Net	ИКТ	14
10	Всего по дисциплине	ИКТ	150
	Примечание: Т – тестирование, Р – написание реферата, У – устный опрос, КР – контрольная работа		

4. УСЛОВИЯ РЕАЛИЗАЦИИ ПРОГРАММЫ УЧЕБНОЙ ДИСЦИПЛИНЫ

«МДК.01.01 Разработка программных модулей»

4.1. Материально-техническая база, необходимая для осуществления образовательного процесса по дисциплине

Для реализации программы учебной дисциплины должны быть предусмотрены следующие специальные помещения:

Кабинет для преподавания дисциплины «Основы разработки баз данных», оснащенный оборудованием и техническими средствами обучения:

- посадочные места по количеству обучающихся;
- рабочее место преподавателя;
- необходимая для проведения практических занятий методическая и справочная литература (в т.ч. в электронном виде).
- компьютеры;
- выход в интернет;
- мультимедийный проектор, экран;
- мультимедийные презентации.

4.2 Перечень необходимого программного обеспечения

1. 7-zip; (лицензия на англ. <http://www.7-zip.org/license.txt>)
2. Adobe Acrobat Reader; (лицензия - <https://get.adobe.com/reader/?loc=ru&promoid=KLXME>)
3. Adobe Flash Player; (лицензия - <https://get.adobe.com/reader/?loc=ru&promoid=KLXME>)
4. Microsoft Office 2003; (лицензия - <http://www.openoffice.org/license.html>)
5. FreeCommander; (лицензия - <https://freecommander.com/ru/%d0%bb%d0%b8%d1%86%d0%b5%d0%bd%d0%b7%d0%b8%d1%8f/>)
6. Mozilla Firefox.(лицензия - <https://www.mozilla.org/en-US/MPL/2.0/>)
7. Visual Studio
8. MySQL Server
9. Workbench

5. ПЕРЕЧЕНЬ ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ УЧЕБНОЙ ЛИТЕРАТУРЫ

Для реализации программы библиотечный фонд образовательной организации должен иметь печатные и/или электронные образовательные и информационные ресурсы, рекомендуемых для использования в образовательном процессе

Основная литература

1. Гниденко, И. Г. Технология разработки программного обеспечения : учебник для среднего профессионального образования / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 248 с. — (Профессиональное образование). — ISBN 978-5-534-18131-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/563151>
2. Федорова, Г. Н. Разработка модулей программного обеспечения для компьютерных систем : учебник для студентов учреждений среднего профессионального образования / Г. Н. Федорова. - 4-е изд., перераб. - Москва : Академия, 2020. - 383 с. : ил. - (Профессиональное образование. ТОП-50). - Словарь терминов: с. 372-377. - Библиогр.: с. 378. - ISBN 978-5-4468-8692-0. - Текст : непосредственный. (25)
3. Федорова, Г. Н. Разработка модулей программного обеспечения для компьютерных систем : учебник для студентов учреждений среднего профессионального образования / Г. Н. Федорова. - 5-е изд., стер. - Москва : Академия, 2023. - 383 с. : ил. - (Профессиональное образование. ТОП-50). - Словарь терминов: с. 372-377. - Библиогр.: с. 378. - ISBN 978-5-0054-0485-5 - Текст : непосредственный. 25 экз.

Дополнительная литература

1. Акопов, А. С. Компьютерное моделирование : учебник и практикум для среднего профессионального образования / А. С. Акопов. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 426 с. — (Профессиональное образование). — ISBN 978-5-534-18369-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/566073>
2. Богатырев, В. А. Информационные системы и технологии. Теория надежности : учебное пособие для вузов / В. А. Богатырев. — 2-е изд. — Москва : Издательство Юрайт, 2024. — 366 с. — (Высшее образование). — ISBN 978-5-534-15951-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/510320>
3. Проектирование информационных систем : учебник и практикум для среднего профессионального образования / Д. В. Чистов, П. П. Мельников, А. В. Золотарюк, Н. Б. Ничепорук. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 273 с. — (Профессиональное образование). — ISBN 978-5-534-20362-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/562355>

4. Советов, Б. Я. Информационные технологии : учебник для среднего профессионального образования / Б. Я. Советов, В. В. Цехановский. — 8-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 414 с. — (Профессиональное образование). — ISBN 978-5-534-20053-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/560670>
5. Тузовский, А. Ф. Объектно-ориентированное программирование : учебник для вузов / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2025. — 213 с. — (Высшее образование). — ISBN 978-5-534-16316-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://urait.ru/bcode/561394>

Периодические издания

1. Открытые системы.- URL: <http://biblioclub.ru/index.php?page=journal&jid=436083>
2. Информатика в школе .- URL: <http://dlib.eastview.com/browse/publication/18988>
3. Программные продукты и системы.- URL: <http://dlib.eastview.com/browse/publication/64086>
4. Информатика и образование.- URL: <http://dlib.eastview.com/browse/publication/18946>
5. Системный администратор.- URL: <http://dlib.eastview.com/browse/publication/66751>
6. Computerword Россия.- URL: <http://dlib.eastview.com/browse/publication/64081>
7. Мир ПК.- URL: <http://dlib.eastview.com/browse/publication/64067>
8. Информационно-управляющие системы.- URL: <http://dlib.eastview.com/browse/publication/71235>
9. Журнал сетевых решений LAN.- URL: <http://dlib.eastview.com/browse/publication/64078>
10. Информатика и образование.- URL: <http://dlib.eastview.com/browse/publication/18946>
11. Windows IT Pro/ Re.- URL: <http://biblioclub.ru/index.php?page=journal&jid=138741>
12. Прикладная информатика.- URL: http://elibrary.ru/title_about.asp?id=25599

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

Электронная библиотека Научной библиотеки КубГУ
<http://megapro.kubsu.ru/MegaPro/Web>

Электронно-библиотечные системы (ЭБС)

1. ЭБС «Университетская библиотека ONLINE»: сайт. — URL: <http://biblioclub.ru>
2. ЭБС Издательства «Лань»: сайт. — URL: <http://e.lanbook.com>
3. ЭБС «Юрайт»: сайт. — URL: <https://urait.ru/>
4. ЭБС «BOOK.ru»: сайт. — URL: <https://www.book.ru>
5. ЭБС «ZNANIUM.COM»: сайт. — URL: <https://www.znanium.com>
6. ЭБС ОИЦ «Академия» <https://academio-moscow.ru/elibrary/>

Профессиональные базы данных

1. Виртуальный читальный зал Российской государственной библиотеки (РГБ)
<https://Idis.rsl.ru/>

2. Национальная электронная библиотека <https://rusneb.ru/>
3. Базы данных компании «ИВИС» <https://eivis.ru/>
4. Научная электронная библиотека «eLibrary.ru» (НЭБ) <http://elibrary.ru/>
5. МИАН. Полнотекстовая коллекция математических журналов <https://www.mathnet.ru/>
6. Полнотекстовая коллекция журналов на платформе РЦНИ (Электронные версии научных журналов РАН) <https://journals.rcsi.science/>
7. Президентская библиотека им. Б.Н. Ельцина <https://www.prilib.ru/>
8. Электронная библиотечная система социо-гуманитарного знания «SOCHUM» <https://sochum.ru/>

Информационные справочные системы

Консультант плюс - справочная система (доступ по локальной сети с компьютеров Библиотеки)

Базы данных открытого доступа

1. КиберЛенинка <https://cuberleninka.ru/>
2. Лекториум ТВ – видеокolleкции ведущих лекторов России <https://www.lektorium.tv/>
3. Федеральный портал «Российское образование» <https://www.edu.ru/>
4. Портал «Образование на русском: Проект Государственного института русского языка им.А.С.Пушкина <https://pushkininstitute.ru/>
5. Справочно-информационный портал «Грамота .ru» <https://gramota.ru/>
6. Словари и энциклопедии на Академике <https://dic.academic.ru/>

Базы данных КубГУ

1. Открытая среда модульного динамического обучения КубГУ <https://openedu.kubsu.ru/>
2. База учебных планов, учебно-методических комплексов, публикаций и конференций <https://infoneeds.kubsu.ru/>
3. Электронный архив документов КубГУ <https://docspace.kubsu.ru/>

6. КОНТРОЛЬ И ОЦЕНКА РЕЗУЛЬТАТОВ ОСВОЕНИЯ УЧЕБНОЙ ДИСЦИПЛИНЫ МДК.01.01 «Разработка программных модулей»

6.1. Критерии оценивания практических работ

Раздел модуля ПМ.01: МДК01.01 Разработка программных модулей		
ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием	Оценка «отлично» - техническое задание проанализировано, алгоритм разработан, соответствует техническому заданию и оформлен в соответствии со стандартами, пояснены его основные структуры. Дополнительно для квалификаций "Программист" и "Технический писатель": указаны использованные стандарты в области документирования; выполнена оценка сложности алгоритма Оценка «хорошо» -алгоритм разработан, оформлен в соответствии со стандартами и соответствует заданию, пояснены его основные структуры. Дополнительно для квалификаций "Программист" и "Технический писатель": выполнена оценка сложности алгоритма Оценка «удовлетворительно» - алгоритм разработан и соответствует заданию.	Экзамен/зачет в форме собеседования: практическое задание по построению алгоритма в соответствии с техническим заданием Защита отчетов по практическим и лабораторным работам
ПК 1.2 Разрабатывать программные модули в соответствии с техническим заданием	Оценка «отлично» - программный модуль (для квалификации "Специалист по тестированию в области информационных технологий": тестовый модуль) разработан по имеющемуся алгоритму в среде разработки (Дополнительно для квалификаций "Программист" на указанном языке программирования) методами объектно- ориентированного/ структурного программирования и полностью соответствует техническому заданию, соблюдены и пояснены основные этапы разработки; документация на модуль оформлена и соответствует стандартам. Оценка «хорошо» - программный модуль (для квалификации "Специалист по тестированию в области информационных технологий") разработан по имеющемуся алгоритму в среде разработки (Дополнительно для квалификаций "Программист" на указанном языке программирования) методами объектно- ориентированного/ структурного программирования и полностью соответствует техническому заданию, соблюдены и пояснены основные этапы разработки; документация на модуль оформлена и соответствует стандартам.	Экзамен/зачет в форме собеседования: практическое задание по разработке программного модуля в соответствии с техническим заданием Защита отчетов по практическим и лабораторным работам Интерпретация результатов

	технологий": тестовый модуль) разработан по имеющемуся алгоритму в среде разработки (Дополнительно для квалификаций "Программист" на указанном языке программирования) методами объектно- ориентированного/ структурного программирования и практически соответствует техническому заданию с незначительными отклонениями, пояснены основные этапы разработки; документация на модуль оформлена и соответствует стандартам. Оценка «удовлетворительно» - программный модуль (для квалификации "Специалист по тестированию в области информационных технологий": тестовый модуль) разработан по имеющемуся алгоритму в среде разработки (Дополнительно для квалификаций "Программист" на указанном языке программирования) методами объектно- ориентированного/ структурного программирования и соответствует техническому заданию; документация на модуль оформлена без существенных отклонений от стандартов.	наблюдений за деятельностью обучающегося в процессе практики
ПК 1.3 Выполнять отладку программных модулей с использованием специализированных программных средств	Оценка «отлично» - выполнена отладка модуля (Дополнительно для квалификаций "Программист" и "Специалист по тестированию в области информационных технологий": с использованием инструментария среды проектирования); с пояснением особенностей отладочных классов; сохранены и представлены результаты отладки. Оценка «хорошо» - выполнена отладка модуля (Дополнительно для квалификаций "Программист" и "Специалист по тестированию в области информационных технологий": с использованием инструментария среды проектирования); сохранены и представлены результаты отладки.	Экзамен/зачет в форме собеседования: практическое задание по выполнению отладки предложенного программного модуля Защита отчетов по практическим и лабораторным работам Интерпретация результатов наблюдений за деятельностью

	Оценка «удовлетворительно» - выполнена отладка модуля, пояснены ее результаты.	обучающегося в процессе практики
ПК 1.5 Осуществлять рефакторинг и оптимизацию программного кода	Оценка «отлично» - определены качественные характеристики программного кода с помощью инструментальных средств; выявлены фрагменты некачественного кода; выполнен рефакторинг на уровнях переменных, функций, классов, алгоритмических структур; проведена оптимизация и подтверждено повышение качества программного кода. Оценка «хорошо» - определены качественные характеристики программного кода с помощью инструментальных средств; выявлены фрагменты некачественного кода; выполнен рефакторинг на нескольких уровнях; проведена оптимизация и выполнена оценка качества полученного программного кода. Оценка «удовлетворительно» - определены качественные характеристики программного кода частично с помощью инструментальных средств; выявлено несколько фрагментов некачественного кода; выполнен рефакторинг на нескольких уровнях; проведена оптимизация и выполнена оценка качества полученного программного кода.	Экзамен/зачет в форме собеседования: практическое задание по оценке качества кода предложенного программного модуля, поиску некачественного программного кода, его анализу, оптимизации методами рефакторинга. Защита отчетов по практическим и лабораторным работам Интерпретация результатов наблюдений за деятельностью обучающегося в процессе практики

Отметка «5» ставится, если:

- работа выполнена полностью, в логических рассуждениях и обосновании решения нет пробелов и ошибок;
- в решении нет ошибок (возможна одна неточность, описка, не являющаяся следствием незнания или непонимания учебного материала).

Отметка «4» ставится, если:

- работа выполнена полностью, но в обосновании шагов решения недостаточны;
- допущена 1-2 ошибки или 1 ошибка и два-три недочета в выкладках.
- Отметка «3» ставится, если:
- допущены 3 ошибки или 2 ошибки и более двух-трех недочетов в выкладках, но обучающийся владеет обязательными умениями по проверяемой теме.

- Отметка «2» ставится, если:
- допущены существенные ошибки, показавшие, что обучающийся не владеет обязательными умениями по данной теме в полной мере

– 5.2. Критерии оценивания конспектов

- Отметка «5» ставится, если:
- работа содержит полные ответы на все теоретические вопросы для составления конспекта;

Отметка «4» ставится, если:

- работа содержит неполный ответ хотя бы на один теоретический вопрос для составления конспекта;

Отметка «3» ставится, если:

- работа содержит неполные ответы на 2 теоретических вопроса для составления конспекта.

Отметка «2» ставится, если:

- работа содержит неполные ответы на 2 и более теоретических вопроса для составления конспекта.

5.3. Критерии оценивания презентаций

Оценка	5	4	3	2
Содержа- ние	Работа полностью завершена	Почти полностью сделаны наиболее важные компоненты работы	Не все важнейшие компоненты работы выполнены	Работа сделана фрагментарно и с помощью преподавателя
	Работа демонстрирует глубокое понимание описываемых процессов	Работа демонстрирует понимание основных моментов, хотя некоторые детали не уточняются	Работа демонстрирует понимание, но неполное	Работа демонстрирует минимальное понимание
	Грамотно используется научная лексика	Научная лексика используется, но иногда не корректно	Научная терминология или используется мало или используется некорректно.	Минимум научных терминов

Грамотность	Нет ошибок: ни грамматических, ни синтаксических	Минимальное количество ошибок	Есть ошибки, мешающие восприятию	Много ошибок, делающих материал трудночитаемым
Дизайн	Имеются постоянные элементы дизайна. Дизайн подчеркивает содержание	Имеются постоянные элементы дизайна. Дизайн соответствует содержанию.	Нет постоянных элементов дизайна. Дизайн может и не соответствовать содержанию.	Элементы дизайна мешают содержанию, накладываясь на него.
	Все параметры шрифта хорошо подобраны (текст хорошо читается)	Параметры шрифта подобраны. Шрифт читаем	Параметры шрифта недостаточно хорошо подобраны, могут мешать восприятию	Параметры не подобраны. Делают текст трудночитаемым

7. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Обучающиеся для полноценного освоения учебного курса должны составлять конспекты как при прослушивании его теоретической (лекционной) части, так и при подготовке к практическим (семинарским) занятиям. Желательно, чтобы конспекты лекций и семинаров записывались в логической последовательности изучения курса и содержались в одной тетради. Это обеспечит более полную подготовку, как к текущим учебным занятиям, так и сессионному контролю знаний.

Самостоятельная работа является важнейшей формой учебно-познавательного процесса. Цель заданий для самостоятельной работы – закрепить и расширить знания, умения, навыки, приобретенные в результате изучения дисциплины; овладеть умением использовать полученные знания в практической работе; получить первичные навыки профессиональной деятельности.

Началом организации любой самостоятельной работы должно быть привитие навыков и умений грамотной работы с учебной и научной литературой. Этот процесс, в первую очередь, связан с нахождением необходимой для успешного овладения учебным материалом литературой. Обучающийся должен изучить список нормативно-правовых актов и экономической литературы, рекомендуемый по учебной дисциплине; уметь пользоваться фондами библиотек и справочно-библиографическими изданиями.

Задания для самостоятельной работы выполняются в письменном виде во внеаудиторное время. Работа должна носить творческий характер, при ее оценке преподаватель в первую очередь оценивает обоснованность и оригинальность выводов. В письменной работе по теме задания обучающийся должен полно и всесторонне рассмотреть все аспекты темы, четко сформулировать и аргументировать свою позицию по исследуемым вопросам. Выбор конкретного задания для самостоятельной работы проводит преподаватель, ведущий практические занятия в соответствии с перечнем, указанным в планах практических занятий.

Обучение осуществляется по модульно-блочной технологии (лекции, практики) с включением инновационных элементов.

С точки зрения используемых методов лекции подразделяются следующим образом: информационно-объяснительная лекция, повествовательная, лекция-беседа, проблемная лекция и т. д.

Устное изложение учебного материала на лекции должно конспектироваться. Слушать лекцию нужно уметь – поддерживать своё внимание, понять и запомнить услышанное, уловить паузы. В процессе изложения преподавателем лекции студент должен выяснить все непонятные вопросы. Записывать содержание лекции нужно обязательно – записи помогают поддерживать внимание, способствуют пониманию и запоминанию услышанного, приводят знание в систему, служат опорой для перехода к более глубокому самостоятельному изучению предмета.

Методические рекомендации по конспектированию лекций:

- запись должна быть системной, представлять собой сокращённый вариант лекции преподавателя. Необходимо слушать, обдумывать и записывать одновременно;
- запись ведётся очень быстро, чётко, по возможности короткими выражениями;
- не прекращая слушать преподавателя, нужно записывать то, что необходимо усвоить. Нельзя записывать сразу же высказанную мысль преподавателя, следует её понять и после этого кратко записать своими словами или словами преподавателя. Важно, чтобы в ней не был потерян основной смысл сказанного;

- имена, даты, названия, выводы, определения записываются точно;
- следует обратить внимание на оформление записи лекции. Для каждого предмета заводится общая тетрадь. Отличным от остального цвета следует выделять отдельные мысли и заголовки, сокращать отдельные слова и предложения, использовать условные знаки, буквы латинского и греческого алфавитов, а также некоторые приёмы стенографического сокращения слов.

Практические занятия по дисциплине «Стандартизация, сертификация и техническое документирование» проводятся в основном по схеме:

- устный опрос по теории в начале занятия (обсуждение теоретических проблемных вопросов по теме);
- работа в группах по разрешению различных ситуаций по теме занятия;
- решение практических задач индивидуально;
- подведение итогов занятия (или рефлексия);
- индивидуальные задания для подготовки к следующим практическим занятиям.

Цель практического занятия - научить студентов применять теоретические знания при решении практических задач на основе реальных данных.

На практических занятиях преобладают следующие методы:

- вербальные (преобладающим методом должно быть объяснение);
- практические (письменные задания, групповые задания и т. п.).

Важным для студента является умение рационально подбирать необходимую учебную литературу. Основными литературными источниками являются:

- библиотечные фонды филиала КубГУ в г. Геленджике;
- электронная библиотечная система «Университетская библиотека онлайн»;
- электронная библиотечная система Издательства «Лань».

Поиск книг в библиотеке необходимо начинать с изучения предметного каталога и создания списка книг, пособий, методических материалов по теме изучения.

Просмотр книги начинается с титульного листа, следующего после обложки. На нём обычно помещаются все основные данные, характеризующие книгу: название, автор, выходные данные, данные о переиздании и т.д. На обороте титульного листа даётся аннотация, в которой указывается тематика вопросов, освещённых в книге, определяется круг читателей, на который она рассчитана. Большое значение имеет предисловие книги, которое знакомит читателя с личностью автора, историей создания книги, раскрывает содержание.

Прочитав предисловие и получив общее представление о книге, следует обратиться к оглавлению. Оглавление книги знакомит обучаемого с содержанием и логической структурой книги, позволяет выбрать нужный материал для изучения. Год издания книги позволяет судить о новизне материала. В книге могут быть примечания, которые содержат различные дополнительные сведения. Они печатаются вне основного текста и разъясняют отдельные вопросы. Предметные и алфавитные указатели значительно облегчают повторение изложенного в книге материала. В конце книги может располагаться вспомогательный материал. К нему обычно относятся инструкции, приложения, схемы, ситуационные задачи, вопросы для самоконтроля и т.д.

Для лучшего представления и запоминания материала целесообразно вести записи и конспекты различного содержания, а именно:

- пометки, замечания, выделение главного;
- план, тезисы, выписки, цитаты;
- конспект, рабочая запись, реферат, доклад, лекция и т.д.

Читать учебник необходимо вдумчиво, внимательно, не пропуская текста, стараясь понять каждую фразу, одновременно разбирая примеры, схемы, таблицы, рисунки, приведённые в учебнике.

Одним из важнейших средств, способствующих закреплению знаний, является краткая запись прочитанного материала – составление конспекта. Конспект – это краткое связное изложение содержания темы, учебника или его части, без подробностей и

второстепенных деталей. По своей структуре и последовательности конспект должен соответствовать плану учебника. Поэтому важно сначала составить план, а потом писать конспект в виде ответа на вопросы плана. Если учебник разделён на небольшие озаглавленные части, то заголовки можно рассматривать как пункты плана, а из текста каждой части следует записать те мысли, которые раскрывают смысл заголовка.

Требования к конспекту:

- краткость, сжатость, целесообразность каждого записываемого слова;
- содержательность записи - записываемые мысли следует формулировать кратко, но без ущерба для смысла. Объём конспекта, как правило, меньше изучаемого текста в 7-15 раз;
- конспект может быть, как простым, так и сложным по структуре – это зависит от содержания книги и цели её изучения.

Методические рекомендации по конспектированию:

- прежде чем начать составлять конспект, нужно ознакомиться с книгой, прочитать её сначала до конца, понять прочитанное;
- на обложке тетради записываются название конспектируемой книги и имя автора, составляется план конспектируемого текста;
- записи лучше делать при прочтении не одного-двух абзацев, а целого параграфа или главы;
- конспектирование ведётся не с целью иметь определённые записи, а для более полного овладения содержанием изучаемого текста, поэтому в записях отмечается и выделяется всё то новое, интересное и нужное, что особенно привлекло внимание;
- после того, как сделана запись содержания параграфа, главы, следует перечитать её, затем снова обращаться к тексту и проверить себя, правильно ли изложено содержание.

Техника конспектирования:

- конспектируя книгу большого объёма, запись следует вести в общей тетради;
- на каждой странице слева оставляют поля шириной 25-30 мм для записи коротких подзаголовков, кратких замечаний, вопросов;
- каждая страница тетради нумеруется;
- для повышения читаемости записи оставляют интервалы между строками, абзацами, новую мысль начинают с «красной» строки;
- при конспектировании широко используют различные сокращения и условные знаки, но не в ущерб смыслу записанного. Рекомендуется применять

общеупотребительные сокращения, например: м.б. – может быть; гос. – государственный; д.б. – должно быть и т.д.

- не следует сокращать имена и названия, кроме очень часто повторяющихся;
- в конспекте не должно быть механического переписывания текста без продумывания его содержания и смыслового анализа.

Для написания реферата необходимо выбрать тему, согласовать ее с преподавателем, подобрать несколько источников по теме, выполнить анализ источников по решению проблемы, обосновать свою точку зрения на решение проблемы.

8. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ КОНТРОЛЯ УСПЕВАЕМОСТИ

8.1. Паспорт фонда оценочных средств

№ п/п	Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или ее части)	Наименование оценочного средства
Код	Наименование видов деятельности и профессиональных компетенций	ОК 1, 2,4,5,9,10 ПК.1.1 – ПК.1.5	
ОВД 1	Разработка модулей программного обеспечения для компьютерных систем	ОК 1, 2,4,5,9,10	Тестирование, анализ выполнения практических работ
ПК 1.1	Выполнять отладку программных модулей с использованием специализированных программных средств	ПК.1.1 – ПК.1.5	Тестирование, анализ выполнения практических работ
ПК 1.2	Разрабатывать программные модули в соответствии с техническим заданием	ПК.1.1 – ПК.1.5	Тестирование, анализ выполнения практических работ
ПК 1.3	Выполнять отладку программных модулей с использованием специализированных программных средств	ПК.1.1 – ПК.1.5	Тестирование, анализ выполнения практических работ, зачёт
ПК 1.4	Выполнять тестирование программных модулей	ПК.1.1 – ПК.1.5	Тестирование, анализ выполнения практических работ, зачёт
ПК 1.5	Осуществлять рефакторинг и оптимизацию программного кода	ПК.1.1 – ПК.1.5	Тестирование, анализ выполнения практических работ, зачёт

8.2. Критерии оценки знаний

Контроль и оценка результатов освоения модуля осуществляется преподавателем в процессе проведения практических занятий, тестирования, а также выполнения студентами индивидуальных самостоятельных заданий.

Тест. Система стандартизированных заданий, позволяющая автоматизировать процедуру измерения уровня знаний и умений обучающегося. Тест оценивается по количеству правильных ответов (не менее 50%).

Критерии оценки знаний студентов в целом по дисциплине:

«отлично» - выставляется студенту, показавшему всесторонние, систематизированные, глубокие знания учебной программы модуля и умение уверенно применять их на практике при решении конкретных задач, свободное и правильное обоснование принятых решений;

«хорошо» - 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;

«удовлетворительно» - выставляется студенту, показавшему фрагментарный, разрозненный характер знаний, недостаточно правильные формулировки базовых понятий, нарушения логической последовательности в изложении программного материала, но при этом он владеет основными разделами учебной программы, необходимыми для дальнейшего обучения и может применять полученные знания по образцу в стандартной ситуации;

«неудовлетворительно» - выставляется студенту, который не знает большей части основного содержания учебной программы модуля, допускает грубые ошибки в формулировках основных понятий дисциплины и не умеет использовать полученные знания при решении типовых практических задач.

8.3. Оценочные средств для проведения текущей аттестации

8.4 Оценочные средства для проведения промежуточной аттестации

7.4.1. Билеты к экзамену за 4 семестр

Билеты к экзамену по дисциплине
МДК.01.01 «Разработка программных модулей»
для студентов 2 курса
специальности 09.02.07 «Информационные системы и программирование»
за 4 семестр

Билет 1.

1.Понятие ЖЦ ПО. Этапы ЖЦ ПО. Модели ЖЦ ПО: каскадная, инкрементная, спиральная модели. В отчёте: Технология структурного программирования

2. Практическое задание

Билет 2.

1.Технология структурного программирования. Основные этапы решения задач на ЭВМ. Структурное программирование и алгоритмизация. Теорема структуризации.

2. Практическое задание

Билет 3.

1. **Разработка сверху вниз и снизу вверх.** Основные принципы технологии структурного программирования. Методы и приемы алгоритмизации поставленных задач
2. Практическое задание

Билет 4.

1. **Инструментальные средства оформления и документирования** алгоритмов программ. Определение процессов предметной области. Процессы управления проектами. Технология быстрой разработки приложений.
2. Практическое задание

Билет 5.

1. **Методология, технология и инструментальные средства разработки прикладного программного обеспечения.** RAD и CASE-средства. Задачи и функции инструментального программного обеспечения
2. Практическое задание

Билет 6.

1. **Простые методы сортировки.** Алгоритмы решения типовых задач, области и способы их применения
2. Практическое задание

Билет 7.

1. **Сортировка вставками, выбором, быстрая сортировка, сортировка слиянием, пирамидальная сортировка**
2. Практическое задание

Билет 8.

1. **Алгоритмы сложной сортировки.** Сортировка пузырьком. Сортировка перемешиванием (шейкерная сортировка). Сортировка расчёской
2. Практическое задание

Билет 9.

1. **Алгоритмы решения типовых задач, области и способы их применения**
2. Практическое задание

Билет 10.

1. **Алгоритмы поиска данных.** Последовательный поиск. Индексно-последовательный поиск. Бинарный поиск
2. Практическое задание

Билет 11.

1. **Рекурсивные и эвристические алгоритмы.** Понятие рекурсии и рекурсивной функции. Эвристика. Применение эвристического алгоритма. Пример оценки эвристического решения

2. Практическое задание

Билет 12.

1. Алгоритмы решения типовых задач области и способы их применения

2. Практическое задание

Билет 13.

1. Оценка сложности алгоритма. Классификация, классы алгоритмов, неразрешимые задачи. Методы и приемы формализации задач

2. Практическое задание

Билет 14.

1. Основные этапы и принципы процесса разработки программного обеспечения.

Процесс разработки. Цикл разработки

2. Практическое задание

Билет 15

1. Языки программирования и среды разработки, средства пакетного выполнения процедур

2. Практическое задание

Билет 16.

1. Основы программирования на объектно-ориентированном языке C#. Вид программы на языке C#. Простые типы данных. Оператор присваивания. Понятие арифметического выражения. Ввод данных с клавиатуры и вывод на экран. Форматный вывод

2. Практическое задание

Билет 17.

1. Основные структуры данных. Синтаксис языка программирования C#, особенности программирования на этом языке, стандартные библиотеки языка программирования

2. Практическое задание

Билет 18.

1. Особенности использования данных целого и вещественного типа. Правила согласования типов. Проблема переноса программы с одного языка программирования на другой

2. Практическое задание

Билет 19.

1. Операторы разветвления алгоритма. Логические выражения. Условный оператор if. Условная операция. Оператор выбора (варианта, переключения) switch. Оператор разрыва break. Оператор безусловного перехода goto

2. Практическое задание

Билет 20.

1.Операторы циклов. Оператор цикла с параметром (for). Оператор цикла с предусловием (while). Оператор цикла с постусловием (do while). Вспомогательные операторы, управляющие работой цикла: оператор разрыва break и оператор продолжения continue

2. Практическое задание

Билет 21.

1. Одномерные массивы: определение и описание массива, формирование массивов исходных данных, особенности вывода одномерных массивов. Символьный массив

2. Практическое задание

Билет 22.

1.Двумерные (многомерные) массивы: определение двумерных массивов, особенности формирования двумерных массивов, особенности вывода и обработки двумерных массивов

2. Практическое задание

Билет 23.

1.Использование подпрограмм. Понятие подпрограммы. Структура процедур и функций. Обращение к подпрограмме. Механизм передачи параметров. Рекурсивные процедуры и функции

2. Практическое задание

Билет 24.

1. Строковые данные. Функция для работы со строками. Перенос символов между строкой и символьным массивом

2. Практическое задание

Билет 25.

1.Другие способы организации данных. Коллекции. Списки. Понятие коллекции и списка. Способы определения и описания коллекции и списка

2. Практическое задание

Билет 26.

1.Структурированные данные (структуры, struct)

2. Практическое задание

Билет 27.

1.Запись и чтение данных из файла. Методы организации файловых систем. Запись и чтение строк. Прямой доступ к файлу

2. Практическое задание

Билет 28.

1. **Понятие классов.** Классы: основные понятия. Применение классов
2. Практическое задание

Билет 29.

1. **Принципы ООП: наследование, инкапсуляция, полиморфизм**
2. Практическое задание

Билет 30.

1. **Перегрузка методов.** Понятие метода и сигнатуры. Способы применения перегрузки методов
2. Практическое задание

Билет 31.

1. **Операции класса.** Понятие операции класса. Секции операции. Параметры и выражения. Кванторы видимости
2. Практическое задание

Билет 32.

1. **Иерархия классов.** Базовые и производные классы. Модификаторы доступа. Конструкторы и деструкторы. Примеры реализации классов
2. Практическое задание

Билет 33.

1. **Синтаксис интерфейсов.** Определение интерфейса. Применение интерфейсов. Явная реализация и реализация в базовых и производных классах
2. Практическое задание

Практические задания к экзамену

по МДК.01.01 «Разработка программных модулей» за 4 семестр
для студентов 2 курса специальности 09.03.07 «Информационные системы и программирование»

Разработать приложение WpfApp на C# для решения перечисленных ниже задач:

№ по списку		Вопрос
1	12	Суммирование всех чисел последовательности
2	13	Суммирование чисел последовательности, удовлетворяющих некоторому условию
3	14	Подсчет количества чисел последовательности, удовлетворяющих некоторому условию. Например, найти сумму элементов, кратных трём

4	15	Определение среднего арифметического чисел последовательности, удовлетворяющих некоторому условию
5	16	Определение порядкового номера некоторого значения в заданной последовательности
6	17	Определение максимального значения в последовательности чисел
7	18	Определение порядкового номера максимального значения в последовательности чисел
8	19	Определение максимального значения чисел последовательности, удовлетворяющих некоторому условию
9	20	Нахождение количества введенных чисел с заданными свойствами.
10	21	Нахождение среднего арифметического элементов массива с заданными свойствами
11	22	Изменение значений элементов массива с заданными свойствами

8.4.2 Вопросы к зачёту за 5 семестр

Вопросы к зачёту за 5 семестр по дисциплине

МДК .01.01 «Разработки программных модулей» для студентов 3 курса

1. **Интерфейсы и наследование.** Наследование интерфейсов. Ковариантность и контрвариантность обобщенных интерфейсов
2. **Структуры.** Определение структуры. Конструкторы структуры. Применение структур в программе
3. **Делегаты.** Определение делегатов. Присвоение ссылки на метод. Соответствие методов делегату. Добавление методов в делегат. Объединение делегатов. Вызов делегата. Делегаты как параметры методов. Обобщенные делегаты. Применение делегатов
4. **Регулярные выражения.** Классы StringBuilder и String. Параметр RegexOptions. Синтаксис регулярных выражений. Проверка на соответствие строки формату. Замена и метод Replace
5. **Коллекции. Параметризованные классы.** Понятие коллекции. ArrayList. Список List. Двухсвязный список LinkedList. Очередь Queue. Стек Stack. Словарь Dictionary. Класс ObservableCollection. Интерфейсы Enumerable и Enumerator. Итераторы и оператор yield
6. **Указатели.** Определение указателя. Операции с указателями. Арифметика указателей. Константы и указатели. Указатели и массивы. Указатели в параметрах функции. Массивы в параметрах функции. Указатели на функции. Указатели на функции как параметры. Указатель на функцию как возвращаемое значение. Динамические объекты и массивы
7. **Операции со списками.** Класс List. Двухсвязный список. Добавление и удаление элементов. Получение индекса элемента и сортировка списков
8. **Назначение и виды паттернов.** Основы паттернов проектирования. Введение в паттерны проектирования
9. **Отношения между классами и объектами.** Интерфейсы или абстрактные классы
10. **Основные шаблоны.** Взаимодействие между классами или объектами. Идиомы. Шаблон делегирования. Шаблон функционального дизайна. Неизменяемый интерфейс. Интерфейс-маркер. Контейнер свойств. Канал событий

11. **Порождающие шаблоны.** Фабричный метод (Factory Method). Абстрактная фабрика (Abstract Factory). Одиночка (Singleton). Прототип (Prototype). Строитель (Builder)
12. **Структурные шаблоны.** Декоратор (Decorator). Адаптер (Adapter). Фасад (Facade). Компоновщик (Composite). Заместитель (Прокси). Мост (Bridge). Приспособленец (Flyweight)
13. **Поведенческие шаблоны.** Стратегия (Strategy). Наблюдатель (Observer). Команда (Command)
14. **Шаблонный метод** (Template Method). Итератор (Iterator). Состояние (State). Цепочка Обязанностей (Chain of responsibility). Интерпретатор (Interpreter). Посредник (Mediator). Хранитель (Memento). Посетитель (Visitor)
15. **Принцип единственной обязанности.** Принципы SOLID. Принцип единственной обязанности. Принцип открытости/закрытости. Принцип подстановки Лисков. Принцип разделения интерфейсов. Принцип инверсии зависимостей
16. **Событийно-управляемое программирование.** Основные характеристики языка. Структура программы. Визуальные компоненты .NET.
17. **Методы и средства сборки модулей** и компонент программного обеспечения. (VS)
18. **Технологии разработки приложений Wpf.** Создание приложений при помощи визуальных компонентов. Создание формы. Компонент Form как основной контейнер для всех визуальных компонентов
19. **События и свойства.** Основные визуальные компоненты программирования: MainWindow, Button, Label, TextBox, ComboBox и т.д.
20. **Элементы управления. Диалоговые окна. Обработчики событий.** Присваивание и вывод результатов при помощи компонента Label. Ввод данных и вывод результатов. Условный оператор. Условный оператор с двумя ветвями. Условный оператор с операторными скобками. Вложенные условные операторы.

Вопросы экзамену за 6 семестр

- 1 События и свойства.** Основные визуальные компоненты программирования: MainWindow, Button, Label, TextBox, ComboBox и т.п.
- 2 Элементы управления. Диалоговые окна. Обработчики событий.** Присваивание и вывод результатов при помощи компонента Label
- 3 Свойства объекта MainWindow. Способы создания групп компонентов.** Создание группы «радиокнопок» (radioButton) и «чек-кнопок» (checkBox). Компонент TrackBar. Компонент ComboBox. Компонент ListBox. Компонент GroupBox. Перезапись строк из разных компонентов. Изменение свойств текста
- 4 Методы формирования меню.** Оформление меню
- 5 Обработка дополнительных форм.** Добавление дополнительных окон и страниц в проект. Обработка диалоговых окон. Окна сообщений (MessageBox). Вызов диалоговых окон

6 Применение вкладок на форме. Компонент tabControl. Разработка программы с помощью вкладок. Отдельная вкладка как индивидуальный контейнер компонентов

7 Компоненты диапазона значений. Компоненты ProgressBar и Slider. Настройка и применение компонентов

8 Работа с компонентом вывода табличных данных. Вывод данных с помощью компонента dataGrid

9 Создание базы данных и привязка для последующего вывода и редактирования БД с помощью компонента dataGridView. Компонент DataSet

10 Рисование простых графических объектов. Изменение масштаба. Вращение в плоскости. Рисование трехмерных фигур

11 Применение в программировании прочих компонентов. Компонент панели (panel). Применение таймера в визуальном программировании (timer). Применение компонента ListView. Ввод «по маске», в том числе паролей, с использованием MaskedTextBox. Использование календаря MonthCalendar. Встраивание компонента WebBrowser.

12 Распараллеливание программ. Способы оптимизации программного кода. Основные принципы оптимизации: естественность, производительность, время. Участки кода, которые не оптимизируются. Настройка окружения. Избавление от ненужного функционала. Мемоизация. Кеширование

13 Понятие, цели и методы рефакторинга (перепроектирования) кода. Причины применения рефакторинга. Признаки плохого кода. Методы и приемы рефакторинга. Проблемы, возникающие при применении рефакторинга. Средства автоматизации рефакторинга

14 Принципы разработки графического интерфейса пользователя. Понятие интерфейса пользователя. GUI. UI-дизайн. Необходимость разработки UI-дизайна. Разработка UI-интерфейсов. Понятие интерфейса пользователя

17 Правила разработки интерфейсов пользователя. Главная задача интерфейса пользователя. Ключевое назначение интерфейсов. Акценты и их расстановка. Взаимодействия пользователя с интерфейсом. Основные принципы и правила разработки пользовательских интерфейсов

18 Основы работы с MS SQL Server (MySql Server). Установка MS SQL Server. Установка SQL Server Management Studio. Установка LocalDB.

19 Работа с базами данных. Понятие и структура базы данных (БД)

Основы интерфейса взаимодействия с базами данных в ADO.NET. Провайдеры данных

20 Начало работы с MS SQL Server (MySql Server). Создание базы данных. Создание таблиц. Первый запрос на T-SQL (Workbench)

21 Доступ к данным. Создание базы данных. Строка подключения. Создание подключения. Пул подключений. Методы и средства миграции и преобразования данных

22 Операторы DDL (Data definition language). Создание и удаление базы данных. Создание и удаление таблиц. Типы данных T-SQL (MySQL). Атрибуты и ограничения столбцов и таблиц. Внешние ключи. Изменение таблицы. Пакеты. Команда GO

23 Создание таблицы, работа с записями. Команды создания и удаления таблиц. Обновление, добавление, изменение и удаление записей в таблицах БД

24 Операторы DML (Data manipulation language). Добавление данных. Команда INSERT. Выборка данных. Команда SELECT

25 Сортировка средствами ORDER BY. Извлечение диапазона строк. Фильтрация средствами WHERE. Операторы фильтрации. Обновление данных. Команда UPDATE. Удаление данных. Команда DELETE

26 Способы создания команд. Выполнение команд и класс SqlCommand. Чтение результатов запроса и SqlDataReader. Типизация результатов SqlDataReader. Получение скалярных значений

27 Параметризация запросов. Выходные параметры

28 Хранимые процедуры и транзакции. Работа с хранимыми процедурами. Выходные параметры хранимых процедур

29 Транзакции. Сохранение и извлечение файлов из базы данных

30 Методы группировки и соединения таблиц. Агрегатные функции. Операторы GROUP BY и HAVING. Расширения SQL Server для группировки

31 Неявное соединение таблиц. Inner Join. Outer Join. Группировка в соединениях. UNION, EXCEPT, INTERSECT

32 Принципы работы с компонентами SqlDataAdapter и DataSet. SqlDataAdapter и DataSet.

Постраничный просмотр в SqlDataAdapter. SqlCommandBuilder и сохранение изменений DataSet в базе данных.

33 Обновление БД из DataSet вручную. Все операции с БД в графическом приложении. DataSet и DataTable. Отношения между таблицами в DataSet. LINQ to DataSet. DataSet и XML

34 Методы преобразования LINQ в SQL. Определение контекста данных и моделей. Операции с данными в LINQ to SQL. Изменение объектов в LINQ to SQL. Добавление в LINQ to SQL. Удаление в LINQ to SQL.

35 Методы ExecuteCommand и ExecuteQuery. Хранимые процедуры

8. ДОПОЛНИТЕЛЬНОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Лекция 2. Технология структурного программирования

Дополнительный материал

Эдсгер Вйбе Дэйкстра ([нидерл. Edsger Wybe Dijkstra](#)^[5] (11 мая 1930, [Роттердам, Нидерланды](#) — 6 августа 2002, [Ньюэнен](#)^[nl], Нидерланды — нидерландский учёный, труды которого оказали влияние на развитие [информатики](#) и [информационных технологий](#); один из разработчиков концепции [структурного программирования](#), исследователь [формальной верификации](#) и [распределённых вычислений](#). [Тьюринговский лауреат](#) (1972).

Биография

Родился 11 мая 1930 года в [Роттердаме](#), в семье учёных (отец — [химик](#), мать — [математик](#)).

По окончании школы поступил на факультет [теоретической физики](#) [Лейденского университета](#).

В 1951 году увлёкся [программированием](#), поступил на трёхнедельные [компьютерные](#) курсы в [Кембридже](#), с 1952 года работал [программистом](#) в [Математическом центре Амстердама](#) под руководством профессора [Адриана ван Вейнгаардена](#), впоследствии — автора одного из способов формального описания грамматики формальных языков — так называемых [двухуровневых грамматик](#) ван Вейнгаардена.

Уже в 1952 году принял решение окончательно специализироваться на программировании, но всё же окончил курс теоретической физики.

Во второй половине 1950-х годов в поисках путей оптимизации разводки [плат](#) разработал алгоритм поиска кратчайшего пути на графе, ставший известным как «[алгоритм Дейкстры](#)».

В 1957 году женился, по собственным воспоминаниям, в графе «профессия» анкеты, которую положено заполнять при бракосочетании, написал «программист» щ— и его заставили переписывать документы, заявив, что такой профессии не существует, в результате пришлось указать «физик-теоретик»^[6].

В 1958—1960 годах принимал участие в разработке языка программирования [Алгол](#), работал в команде по созданию [компилятора](#) языка; соревнуясь с датской командой [Петера Наура](#), поклялся не бриться до завершения проекта и победил, написав компилятор за шесть недель, заодно изобретя новое правило компиляции — «вызов по имени».

В [1960-е годы](#) участвовал в создании [операционной системы ТНЕ](#), построенной в виде множества параллельно исполняющихся взаимодействующих процессов^[7]. Именно в ходе этой работы появились понятия [синхронизации процессов](#), идея [семафора](#), а также была чётко осознана необходимость в структуризации процесса программирования и самих программ.

Длительное время работал в компании [Burroughs](#). В [1970-е годы](#) вместе с [Тони Хоаром](#) и [Никлаусом Виртом](#) разработал основные положения [структурного программирования](#).

В последние годы жизни преподавал в [Техасском университете](#).

Умер 6 августа 2002 года после долгой борьбы с [раком](#).

Научные достижения

Известность Дейкстре принесли его работы в области применения математической [логики](#) при разработке компьютерных программ.

Он активно участвовал в разработке [языка программирования Алгол](#) и написал

первый компилятор Алгол-60.

Будучи одним из авторов концепции [структурного программирования](#), он проповедовал отказ от использования инструкции [goto](#).

Также ему принадлежит идея применения «[семафоров](#)» для синхронизации процессов в многозадачных системах и алгоритм нахождения кратчайшего пути на [ориентированном графе](#) с неотрицательными весами рёбер, известный как [алгоритм Дейкстры](#).

Также предложил [алгоритм сортировочной станции](#) — способ разбора математических выражений, представленных в [инфиксной нотации](#).

В [1972 году](#) стал лауреатом [премии Тьюринга](#).

В 2002 году получил ежегодную премию, вручаемую [Симпозиумом по принципам распределённых вычислений](#) (англ. [Symposium on Principles of Distributed Computing](#)) [Ассоциации вычислительной техники](#) «за публикацию, оказавшую наибольшее влияние на область распределённых вычислений»; в знак признания заслуг учёного с 2003 года эта премия носит название [премии Дейкстры](#).

Библиография[[править](#) | [править код](#)]

Автор нескольких книг и множества статей, самые известные публикации — книги «Дисциплина программирования», «Заметки по структурному программированию», статья «О вреде оператора GOTO» (англ. *GOTO considered harmful*).

Помимо обсуждения специальных вопросов, в своих статьях и книгах Дейкстра последовательно отстаивал необходимость математического подхода к программированию, который предполагает предварительное точное, всестороннее математическое описание задачи и способа её решения, формальное доказательство правильности выбранного алгоритма и последующую реализацию алгоритма в виде максимально простой, структурированной программы, корректность которой должна быть формально доказана.

По мнению Дейкстры, господствующий в компьютерной индустрии подход к программированию как к процессу достижения результата методом проб и ошибок («написать ко — протестировать — найти ошибки — исправить — протестировать — ...») порочен, поскольку стимулирует программистов не думать над задачей, а писать код, что при этом совершенно не гарантирует корректность программ, которая не может быть доказана тестированием в принципе.

Многokrатно предостерегал от попыток превратить разработку программ в некий тривиальный процесс; по его мнению, программирование в сути своей — чрезвычайно сложная научная и инженерная деятельность, и никакие новые методы и инструменты не смогут кардинально изменить это положение — они лишь освобождают программиста от части рутинной работы. Попытки же превратить программирование в простое занятие, доступное каждому, обречены на провал.

В 1975 году на примере положения дел в Германии Дейкстра показал, что развитие программирования как науки, опираясь на один выбранный язык программирования, невозможно.

Результатом такого подхода стал полный разрыв теории и практики программирования. Дейкстра отмечал, что принятие в Германии на правительственном уровне языка [ALGOL 68](#) в качестве фундаментального средства дальнейшего развития имело столь же парализующее действие, как и решение [правительства СССР](#) о переходе советской промышленности к копированию модельного ряда [IBM/360](#) в конце 1960 годов, которое учёный назвал величайшей победой Запада в [холодной войне](#)^[10].

Интересные факты[[править](#) | [править код](#)]

Интересные факты о Дейкстре:^[11]

1. Свои статьи Дейкстра нумеровал EWD0, EWD1, EWD2 и т. д.
2. Факт о Дейкстре: «Известно, что он мало заинтересован в приеме на старшие курсы университета, где он преподает, студентов со знанием Фортрана по той причине, что вместе с этими знаниями могли привиться дурные привычки программирования».
3. Еще одна цитата из Дейкстры^[12]: «Изящество, ясность и тому подобное в значительной степени определяются количественными аспектами. (Этим владел Моцарт: многие его произведения, от которых замирает дыхание, обманчиво просты; кажется, будто они созданы практически из ничего!))»

+++++

Стиль программирования — набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

Императивными называются такие языки программирования, в которых описываются в основном "инструкции" по пошаговому выполнению алгоритмов. Все внимание в них отводится лишь реализации этой последовательности действий, а данным и их структуре отводится второстепенная роль. Примером такого рода языков может служить язык создания драйверов устройств: Assembler и C++. При использовании этих языков мы не знаем, какие данные будет отправлять/получать устройство, но зато мы знаем, какие действия нужно осуществить над данными.

Декларативными называются языки программирования, в которых алгоритм работы с данными "зашит" в языке программирования, а сама программа представляет собой данные, упорядоченные и структурированные таким образом, что их легко обрабатывают "стандартные" алгоритмы вывода. В декларативных языках как бы "нет разницы" между данными и алгоритмом, их обрабатывающим. От "значения" данных, подаваемых на вход программы, зависит дальнейшее направление расчетов. Примерами декларативных языков могут служить языки: LISP и Prolog. И программа, и данные на языке ЛИСП представляют собой "списки" вместе с функциями, выполняемыми над ними. На языке ПРОЛОГ есть условное разделение программы на данные ("факты") и код ("правила", "продукции"). Но это разделение достаточно условно: и факты, и правила имеют одну и ту же форму записи.

Декларативные языки лучше всего использовать в случаях, когда "данные управляют программой": при написании экспертных систем, при конструировании трансляторов с языков программирования, для большинства задач искусственного интеллекта. Именно там их использование приведет к наибольшей эффективности.

Понятия парадигмы программирования и основные их виды

Парадигма программирования — это комплекс концепций, принципов и абстракций, определяющих фундаментальный стиль программирования.

Общие сведения о парадигмах программирования

За те пятьдесят с небольшим лет, которые существует дисциплина: "программирование", - сменилось несколько поколений правил, концепций, моды, взглядов на то, как надо "писать программы". Все эти правила, концепции, взгляды, мода, наконец, позже назвали в информатике "парадигмами программирования". Вот неполный список парадигм, вместе с датами их появления:

- 1957г., процедурная парадигма, язык ФОРТРАН, создатель Бэкус;
- 1967г., функциональная парадигма, язык ЛИСП, создатель Маккарти;
- 1970г., структурированная парадигма, язык ПАСКАЛЬ, создатель Вирт;
- 1974г., логическая парадигма, язык ПРОЛОГ;
- 1983г., объектно-ориентированная парадигма, пример - язык C++, создатель Страуструп.

В процессе своего развития эти парадигмы "мутировали", "отпочковывались", "объединялись" (так, язык Си объединил в себе процедурную и структурированную парадигму). В результате этих "мутаций" по состоянию на 01.06.2008 года оформились следующие парадигмы программирования:

1. Процедурная (или модульная) парадигма;
2. Логическая парадигма;
3. Объектно-ориентированная парадигма.

Эти три совершенно разные по подходу парадигмы составляют основу современного программирования. Хотя в настоящее время наиболее "модной" является объектно-ориентированная парадигма, остальные парадигмы также используются на практике и изучаются в ВУЗах. .

Следует отметить, что язык программирования не обязательно использует только одну парадигму. Языки, поддерживающие несколько парадигм, называются мультипарадигменными. Создатели таких языков придерживаются точки зрения, гласящей, что ни одна парадигма не может быть одинаково эффективной для всех задач, и следует позволять программисту выбирать лучший стиль программирования для решения каждой отдельной задачи.

Императивное программирование

Императивная парадигма программирования описывает процесс вычислений посредством описания управляющей логики программы, т.е. в виде последовательности отдельных команд, которые должен выполнить компьютер. Каждая команда является инструкцией, изменяющей состояние программы. Программа, написанная в императивном стиле, похожа на набор приказов, выражаемых повелительным наклонением в естественных языках.

Машинный код является наиболее низкоуровневым примером реализации этой парадигмы: состояние программы определяется содержимым памяти, а команды — инструкциями машинного кода. Поскольку эта парадигма естественна для человеческого понимания и непосредственно реализована на аппаратном уровне, большинство языков программирования придерживаются именно ее.

Императивное программирование является противоположностью декларативного программирования; второе описывает, что необходимо сделать, а первое — как именно это сделать.

Функциональное программирование

Функциональное программирование — раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании (то есть тех, чей единственный

результат работы заключается в возвращаемом значении, или другими словами, вычисление которых не имеет побочного эффекта). Противопоставляется парадигме императивного программирования, в которой исполнителю программы предписывается последовательность выполняемых действий, в то время, как в функциональном программировании способ решения задачи описывается при помощи зависимости функций друг от друга (в том числе возможны рекурсивные зависимости), но без указания последовательности шагов.

Одной из близких парадигм программирования является логическое программирование, в котором программа представляет собой множество пар (логическое условие, новые факты). В логическом программировании, также как и в функциональном программировании, программист остается в неведении о методах, применяемых при вычислении, и последовательности исполнения элементарных действий. Большая часть ответственности за эффективность вычислений в логическом и функциональном программировании перекладывается на «плечи» транслятора используемого языка программирования.

Функциональное и логическое программирование являются частями т. н. «декларативного программирования», т. е. такого стиля программирования, при использовании которого в программах описывается способ решения поставленной задачи, а не предписываются шаги для получения результата.

Функциональное программирование основано на теориях λ -исчисления (Алонзо Чёрч, 1936) и комбинаторной логики (Моисей Исаевич Шейнфинкель и Хаскелл Карри).

Логическое программирование

Логическая парадигма программирования предусматривает использование математической логики для разработки программ. В более узком смысле, логическое программирование представляет программу в виде набора декларативных утверждений вида чтобы доказать/решить H , следует доказать/решить B_1 и ... и B_n . Этот набор утверждений рассматривается как процедуры по упрощению целей, которые разбивают одну общую задачу на несколько подзадач, так что процесс поиска доказательства/решения для основной цели программы получает вычислительный смысл.

Логическое программирование является частным случаем декларативного программирования, поскольку программист задает только набор формул, а принятие решений об организации вычислений принимается компилятором.

Объектно-ориентированное программирование

Объектно-ориентированная парадигма (ООП) представляет программу как набор объектов и их взаимодействий. Основными понятиями ООП являются следующие:

- объект — элементарная сущность, описываемая определенными свойствами (хранящимися в виде атрибутов объекта) и поведением (реализованным в виде методов);
- класс описывает структуру свойств и поведения одного типа объектов. Каждый объект программы является экземпляром некоторого класса;
- классы могут наследовать атрибуты и методы их родительских классов, в то же время добавляя свои собственные. Иерархия классов позволяет моделировать сущности решаемой задачи на нескольких уровнях детализации и в

дальнейшем использовать класс, отвечающий уровню детализации, необходимому для решения конкретной подзадачи.

- инкапсуляция подразумевает, что некоторые детали реализации класса скрыты от взаимодействующих с ним объектов. У каждого класса есть интерфейс, описывающий взаимодействие объектов этого класса с прочими объектами, и реализация, описывающая то, как это взаимодействие отражается на объекте этого класса.

- Многие современные языки поддерживают ООП, хотя и в разной степени:

- чисто объектно-ориентированные языки, например, Smalltalk и Ruby, разработаны для того, чтобы поддерживать и даже навязывать объектно-ориентированный стиль разработки, и не поддерживают другие стили программирования;

- преимущественно объектно-ориентированные языки, например, Java, C++ и Python, разработаны в основном для поддержки ООП, но позволяют использовать элементы процедурного программирования;

- исторически процедурные языки, например, Perl и Fortran 2002, были доработаны и в них была добавлена поддержка некоторых элементов ООП;

- экзотические языки, например, Modula-2 и Oberon, реализуют некоторые черты и принципы ООП, но в оригинальной форме.

ООП можно противопоставить модульному или процедурному программированию: как объекты, так и модули/подпрограммы являются самодостаточными (в некоторой степени) единицами, но объекты фокусируются на содержащихся в них данных, а модули/подпрограммы — на сгруппированных в них действиях.

Особенности функционального и логического программирования

Преимущества функционального программирования

- Программа представляет собой множество вычисляемых функций
- Отсутствует операция присваивания (осн. Императивный оператор)
- Порядок выполнения программы (вычислений) несущественен
- Быстродействие (по сравнению с аналогичными императивными вычислениями)

- Компактность кода (особенно для списков и вариантных типов)
- Суперпозиции/склеивания функций: из простых – сложные, редукция

- Суперпозиция моделей: склеивание программ, ленивые вычисления
- Численное дифференцирование/интегрирование (в т.ч. модульное)
- Алгоритмы искусственного интеллекта (эвристический поиск и т.п.)
- Распараллеливание вычислений (Haskell)
- Интерактивная отладка, Unit-тестирование, карринг
- Сопоставление с образцом

Преимущества логического программирования

- Задача программиста – описание логической модели предметной области в терминах объектов, их свойств и отношений между ними (без деталей): описание данных и логики их обработки ~ аналогия с ООП
- Удобство описания отношений между объектами (реляционная модель)
- Компактность кода (обработка структурированных данных, лог. правила)
- Возможность перебора и поиска различных решений, заложенная в язык
- Легкость понимания (описание отд. правил), отладки программ (trace)
- Легкость описания сложных структур данных (деревья, списки и т.п.)
- Эффективный метод вычислений – рекурсия
- Отсутствие указателей, операторов присваивания и безусловного перехода
- Множество областей для применения: автоматический перевод, обработка текстов, экспертные системы, САПР, Data-minig системы, автоматическое управление, СУБД, символьные вычисления

+++++

Структурное программирование

Сутью структурного программирования является возможность разбиения программы на составляющие элементы.

Идеи структурного программирования появились в начале 70-годов в компании IBM, в их разработке участвовали известные ученые Э. Дейкстра, Х. Милс, Э. Кнут, С. Хоор.

Распространены две методики (стратегии) разработки программ, относящиеся к структурному программированию: программирование "сверху вниз" и программирование "снизу вверх".

Программирование «сверху вниз», или нисходящее программирование – это методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой. Является противоположной методике программирования «снизу вверх».

При нисходящем проектировании задача анализируется с целью определения возможности разбиения ее на ряд подзадач. Затем каждая из полученных подзадач также анализируется для возможного разбиения на подзадачи. Процесс заканчивается, когда подзадачу невозможно или нецелесообразно далее разбивать на подзадачи.

В данном случае программа конструируется иерархически - сверху вниз: от главной программы к подпрограммам самого нижнего уровня, причем на каждом уровне используются только простые последовательности инструкций, циклы и условные разветвления.

Программирование «снизу вверх», или восходящее программирование – это методика разработки программ, начинающаяся с разработки подпрограмм (процедур,

функций), в то время как проработка общей схемы не закончилась. Является противоположной методике программирования «сверху вниз».

Такая методика является менее предпочтительной по сравнению с нисходящим программированием так как часто приводит к нежелательным результатам, переделкам и увеличению времени разработки.

Достоинства структурного программирования:

- 1) повышается надежность программ (благодаря хорошему структурированию при проектировании, программа легко поддается тестированию и не создает проблем при отладке);
- 2) повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);
- 3) уменьшается время и стоимость программной разработки;
- 4) улучшается читабельность программ.

Резюме

Технология структурного программирования при разработке серьезных программных комплексов, основана на следующих принципах:

- программирование должно осуществляться сверху вниз;
- весь проект должен быть разбит на модули (подпрограммы) с одним входом и одним выходом;
- подпрограмма должна допускать только три основные структуры – последовательное выполнение, ветвление (if, case) и повторение (for, while, repeat).
- недопустим оператор передачи управления в любую точку программы (goto);
- документация должна создаваться одновременно с программированием в виде комментариев к программе.

Структурное программирование эффективно используется для решения различных математических задач, имеющих алгоритмический характер.

Модульное программирование

Модульное программирование – это организация программы как совокупности небольших независимых блоков (модулей), структура и поведение которых подчиняется определенным заранее правилам.

***Модулем** (в модульном программировании) называется множество взаимосвязанных подпрограмм (процедур) вместе с данными, которые эти подпрограммы обрабатывают.*

Модульное программирование предназначено для разработки больших программ.

Разработкой больших программ занимается коллектив программистов. Каждому программисту поручается разработка некоторой самостоятельной части программы. И он в таком случае отвечает за конструирование всех необходимых процедур и данных для этих процедур. Сокрытие данных (запрет доступа к данным из-за пределов модуля) предотвращает их случайное изменение и соответственно нарушение работы программы. Для взаимодействия отдельных частей (модулей) программы коллективу программистов

необходимо продумать только интерфейс (взаимодействие) сконструированных модулей в основной программе.

Напомним понятие и структуру модуля в терминах языка Pascal.

Модуль (unit) – программная единица, текст которой компилируется независимо (автономно).

Модуль содержит 4 раздела: заголовок, интерфейсная часть (раздел объявлений), раздел реализации и раздел инициализации.

```
UNIT <имя модуля>;           {заголовок}
INTERFACE                     {интерфейсная часть}
    Uses <используемые модули>;
    Const <объявления глобальных констант>;
    Type <объявления глобальных типов>;
    Var <описание глобальных переменных>;
    Procedure <заголовки(!) доступных процедур>;
    ...
    Function <заголовки(!) доступных функций>;
    ...
IMPLEMENTATION                {раздел реализации}
    Uses <используемые при реализации модули>;
    Const <объявления скрытых (локальных) констант>;
    Type <объявления скрытых (локальных) типов>;
    Var <описание скрытых (локальных) переменных>;
    Procedure <тела(!) скрытых (локальных) процедур>;
    ...
    Function <тела(!) скрытых (локальных) функций>;
BEGIN
    <основной блок модуля = раздел инициализации>
END.
```

Замечание. Различие всех описаний, содержащихся в разделах интерфейса и реализации, заключается в сфере их использования. В интерфейсном разделе – внешние описания, в разделе реализации – внутренние.

В качестве методики структурного программирования Э. Дейкстра предложил пользоваться лишь конструкциями *цикла* и условного оператора, изгоняя **go to** как концептуально противоречащее этому стилю

Пожалуй, это первое *концептуальное противоречие*, явно отмеченное и учтенное в теории и практике программирования (и даже во всей современной науке). Но,

поскольку не было даже наметок теории неформализуемых понятий, и на работу с ними переносили *опыт* работы с малыми формализациями, структурное противоречие было воспринято следующим образом.

К несчастью, оператор **goto** формально совместим с другими конструкциями традиционных (тогда говорили - универсальных) алгоритмических языков. Но реально он плохо взаимодействует с ними. Значит, он плох сам по себе.

Структурное программирование основано главным образом на теоретическом аппарате теории рекурсивных функций. Программа рассматривается как частично-рекурсивный оператор над библиотечными подпрограммами и исходными операциями.

Структурное *программирование* базируется также на теории доказательств, прежде всего на естественном выводе. ***Структура программы соответствует структуре простейшего математического рассуждения***, не использующего сложных лемм и абстрактных понятий².

Средства структурного программирования в первую очередь включаются во все языки программирования традиционного типа и во многие нетрадиционные языки. Они занимают основное место в учебных курсах программирования и в теоретических работах.

При структурном программировании присваивания и локальные действия становятся органичной частью программы. Достаточно лишь внимательно следить, чтобы каждая *переменная* в модуле использовалась для одной конкретной цели, и не допускать "экономии", при которой ненужная в данном месте *переменная* временно используется под совсем другое *значение*. Такая "экономия" запутывает структуру информационных зависимостей, которая при данном стиле должна быть хорошо согласована со структурой программы.

Структурное *программирование* естественно возникает во многих классах задач, прежде всего в таких, где **задача естественно расщепляется на подзадачи, а информация - на достаточно независимые структуры данных**. Основной его **инвариант: действия и условия локальны**.

Необходимой чертой хорошей реализации структурного стиля программирования является соблюдение согласованности, а в идеале и единства, следующих компонентов программы:

1. **Структура информационного пространства.** Содержательно любую задачу можно описать как переработку объектов, полный набор которых называется **информационным пространством** задачи.

Для структурного стиля программирования требуется следующее.

Задача разбивается на подзадачи, и таким образом выстраивается дерево вложенности подзадач.

- Информационное пространство структурируется в точном соответствии с деревом вложенности: для каждой подзадачи оно состоит из ее **локальных объектов**, определяемых вместе с подзадачей и для нее, и так называемы **глобальных объектов**, определяемых как информационное пространство непосредственно объемлющей подзадачи.

Таким образом, информационное пространство всей задачи (подзадачи самого верхнего уровня) расширяется по мере перехода к подзадачам за счет их локальных объектов. Для различных дочерних подзадач одной подзадачи оно имеет общую часть - информационное пространство родительской подзадачи³.

2. **Структуры управления.** Стиль структурного программирования в его общепринятом варианте предполагает использование строго ограниченного набора *управляющих конструкций: последовательность операторов, условные и выбирающие операторы, все вычислительные ветви которых сходятся в одной точке программы, а также процедуры, вычисления которых всегда заканчиваются возвратом управления в точку вызова.*

К структурным операторам добавляются **либо циклы, либо рекурсии.**

Внимание!

Этой альтернативы Вы не встретите в традиционных изложениях структурного программирования. *Концептуальное противоречие между циклами и рекурсиями* намного мягче, чем между операторами структурного программирования и *структурными переходами*, и оно отмечается лишь в виде изредка встречающихся прагматических указаний (благих пожеланий) не смешивать их произвольно.

3. **Потоки передачи данных.** Разбивая задачу на подзадачи, программист предусматривает их взаимодействие по данным: одни подзадачи передают другим данные для переработки.
4. **Структуры данных.** Данные объединяются в логически связанные фрагменты, соответствующие структурам задачи либо вспомогательных конструкций, вводимых для ее решения.
5. **Призраки.** *Часто даже сама программа не может быть объяснена через понятия, которые используются внутри нее. Еще чаще это происходит для ее связей с внешним миром. Понимание программы возможно лишь после сопоставления реальных внутрипрограммных объектов с идеальными внепрограммными. Эти идеальные внепрограммные объекты (призраки) часто не просто не нужны, но даже вредны для исполнения программы⁴.*

Первым обратил внимание на необходимость введения *призраков* для логического и концептуального анализа программ Г. С. Цейтин в 1971 г. В Америке это "независимо" открыли заново в 1979 г., хотя упомянутая статья Цейтина была опубликована на английском языке в общедоступном издании. Даже название сущностям было дано то же самое... Этому важнейшему и традиционно игнорируемому понятию посвящена отдельная лекция в курсе "Основания программирования"

6. **Подпорки** в программе - *значения, конструкции и сущности*, которые не нужны для понимания задачи и программы, не определяются сущностью задачи и алгоритма, но *вынужденно вставляются для реализации данного алгоритма на конкретной системе.*

Подпорки противоположны *призракам*. На самом деле они являются той формой, в которой материя проникает в программу, и в этом качестве противостоят всей совокупности идеальных сущностей, порождающих структуру программы: как

реальных, так и призрачных, - и порою грубо ее искажают. Но без *подпорок* программа просто не будет работать или будет работать неэффективно.

Для структурного программирования весьма важно требование:

Все структуры подчиняются структуре информационного пространства.

Это общее требование конкретизируется в следующие.

1. Необходимо, чтобы структура управления программы была согласована со структурой ее информационного пространства. Каждой структуре управления соответствуют согласующиеся с ней структуры данных и часть информационного пространства. Это условие позволяет человеку легко отслеживать порядок выполнения конструкций в программе.
2. Подзадачи могут обмениваться данными только посредством обращения к объектам из общей части их информационных пространств (в современных языках чаще всего к глобальным).
3. Информационные потоки должны протекать согласно иерархии структур управления; мы должны четко видеть для каждого блока программы, что он имеет на входе и что дает на выходе. Таким образом, **свойства каждого логически завершенного фрагмента программы должны ясно осознаваться и в идеале четко описываться в самом тексте программы и в сопровождающей ее документации**⁵.
4. Описание переменных, представляющих перерабатываемые объекты, а также других, вспомогательных переменных при структурном программировании строго подчиняется разбиению задачи на подзадачи.
5. Все *призраки* действуют на своем структурном месте и соответствуют идеальным сущностям, которые, согласно парадоксу изобретателя, должны вводиться для эффективного решения задачи.
6. Все *подпорки* строго локализованы в том месте, где их вынуждены ввести. Желательно даже обозначать их по-другому, чем идеальные сущности, например, оставляя мнемонические имена лишь для идеальных сущностей, а *подпорки* именовать джокерами типа **x** или **i**. Необходимо строго следить за тем, чтобы *подпорки* не искажали идеальную структуру программы.

Структурное *программирование* лучше всего описано теоретически, но частные описания не сведены в единую систему. Одни книги трактуют его с точки зрения программиста, другие - с точки зрения теоретика. Так что даже здесь единой системы взглядов еще нет, хотя, видимо, все основания для ее формирования уже имеются.

+++++

Сети данных

Рассмотрим основную структуру данных, которая появляется при структурном программировании. Учет этой структуры позволяет преобразовать благие пожелания о согласованности информационных потоков и хода передач управления в достаточно строгую методику.

Сеть данных может быть формально описана как **ациклический ориентированный граф, в котором все ко-пути (т. е. пути, взятые наоборот) конечны и вершинам которого сопоставлены значения.**

Рассмотрим пример. Известному стандартному приему программирования в языках без кратных присваиваний - обмену двух значений через промежуточное

`z := second;`

`second := first;`

`first := z;`

соответствует следующая *сеть данных*:

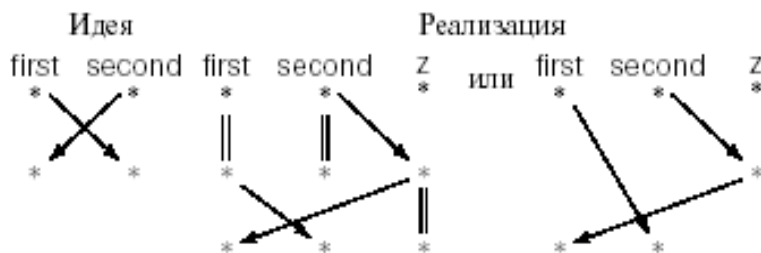


Рисунок 1 -- Обмен значений

Здесь `first`, `second`, `z` можно считать комментариями, а сами данные опущены, поскольку их конкретные значения не важны.

На этом примере видно, что порой для лучшего структурирования сети целесообразно вводить дополнительные вершины, соответствующие сохраняющимся значениям. *Ребро*, ведущее из одной такой вершины в другую, обозначается при помощи стрелочки, похожей на *равенство*. Видно так же, как материя воздействует на идею, заставляя вводить дополнительные *операторы* и дополнительные значения. В данном случае *переменная z* и включающие ее *операторы* являются *подпорками*, и, если их исключить, *сеть данных* становится проще. Но в общераспространенных языках программирования нет кратных присваиваний типа

`first,second:=second,first;`

Даже если бы они были, представьте себе, как неудобно станет читать длинное кратное *присваивание* и понимать, какое же *выражение* какой переменной присваивается!

В случае программы вычисления факториала *сеть* потенциально бесконечна вниз, поскольку аргументом может быть любое число, но по структуре еще проще:



Рисунок 2 – Перекрестные зависимости.

Перекрестных зависимостей между параметрами нет, следовательно, возможны две известные реализации факториала: циклическая и рекурсивная. Покажем их на разных языках, ибо все равно, на каком традиционном языке их писать.

```
function fact(n: integer): integer;
var j, res: integer;
begin
  res:=1;
  for j:=1 to n do res:=res*j;
  result:=res;
end;
```

```
int fact(int n)
{ if (n==0) return(1);
  else return(n*fact(n-1));}
```

Схема построения циклической программы называется **поточковой обработкой**. Значения на следующей итерации *цикла* зависят от значений на предыдущей.

Для чисел Фибоначчи (та же схема ([14.1.2](#))) структура уже несколько сложнее предыдущих, поскольку каждое следующее *число Фибоначчи* зависит от двух предыдущих, но метод потоковой обработки применим и здесь.

```
int fib(int n)
{ int fib1, fib2;
  fib1=1; fib2=1;
  if (n>2){
    for (int i=2; i<n; i++){
      int j; j=fib1+fib2; fib1=fib2; fib2=j;
    }
  };
  return(fib2);
}
```

Итак, в потоке изменяется структура из двух элементов. Ее можно было бы прямо описать как структуру данных, и это следовало бы сделать, будь *программа* хоть чуть-чуть посложнее. Тогда вместо *подпорки j* пришлось бы ввести в качестве *подпорки* новое значение структуры.

В программе имеется еще одна *подпорка* – *параметр цикла i*, который нужен лишь для формальной организации *цикла*.

Рекурсивная реализация чисел Фибоначчи пишется еще проще и служит великолепным примером того, как презренная материя убивает красивую, но неглубокую идею.

```
int fib(int n)
{ if (n<3) return(1);
  else return(fib(n-1)+fib(n-2));
}
```

Если *n* достаточно велико, каждое из предыдущих значений функции Фибоначчи будет вычисляться много раз, причем без всякого толку: результат всегда будет один и тот же! Зато все *подпорки* убраны...

В следующем примере неэффективность *рекурсии* по сравнению с хорошо организованным *циклом* еще более очевидна. Пусть надо найти *путь*, на котором можно собрать максимальное количество золота, через *сеть* значений, подобную показанной на [Рисунок --14.1](#).

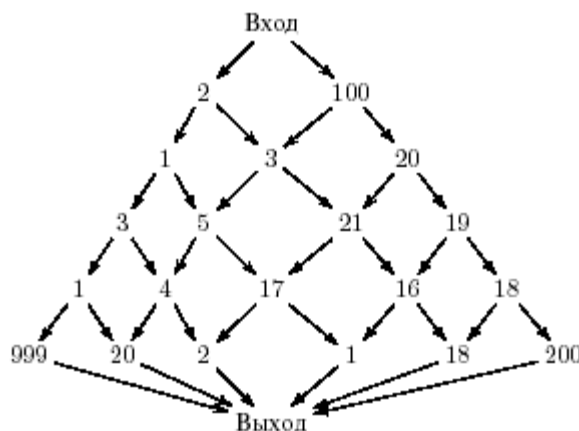


Рисунок 3 --Золотая гора

При циклической организации вычислений нам придется посчитать *значение* в каждой точке горы всего один раз (найти добычу на оптимальном пути в эту точку). Здесь используется то свойство оптимальных путей, которое делает возможным так называемые методы волны или *динамического программирования*: каждый начальный *отрезок* локально оптимального пути локально оптимален. Это свойство является *призраком*, стоящим за эффективной циклической реализацией алгоритма, а многочисленные пути, соответственно, призрачными значениями, которые не нужно вычислять. В рекурсивной реализации мы не учитываем данного *призрака*, и он беспощадно мстит за вопиющее незнание теории.

Теперь рассмотрим случай, когда рекурсивная реализация намного изящнее циклической, легче обобщается и не хуже по эффективности⁷

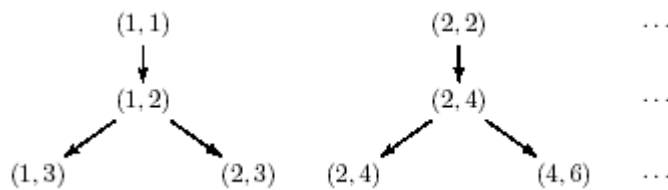


Рисунок 4 --Ал горитм Евклида

В данном случае *путь* для получения результата не разветвляется, и нам остается лишь двигаться по нему в правильном направлении (от цели к исходным данным) и достаточно большими шагами.

```

function Euklides(n,m: integer) integer; {
  предполагаем m<=n}
begin
  if n=m then resut:=n
  else result:=Euklides(n mod m, m);
end;
```

Если пытаться вычислить наибольший общий делитель методом движения от данных к цели, то нам придется построить громадный *массив* значений НОД, лишь ничтожная часть значений в котором будет нужна для построения результата. *Затраты* на *вычисление* каждого отдельного элемента в данном случае малы, а при обратном направлении движения повторный счет не возникает.

Алгоритм Евклида в простейшем случае моделирует ту ситуацию, которая появляется в задачах обработки рекурсивных структур, например списков. То, что в отдельных случаях возникает повторный счет, - небольшое зло, когда эти случаи редки и нерегулярны. Повторный счет возникает и в циклических программах, поскольку найти в большом массиве совпадающие элементы порою труднее, чем заново посчитать нужные нам значения. Именно поэтому *рекурсия* оказалась столь эффективным методом работы со списками и позволила построить адекватный первопорядковый фундамент для современного функционального программирования.

Рассмотрим два крайних случая движения по сети. Когда *сеть* представлена в виде структуры данных, естественно возникает метод ленивого движения по сети, когда после вычисления значений в очередной точке выбирается одна из точек, для которой все предыдущие значения уже вычислены, и вычисляются значения в данной точке. Как видно, в частности на примере золотой горы, этот метод недетерминирован и в значительной степени может быть распараллелен. Но в конкретном алгоритме нам придется выбрать конкретный способ ленивого движения, и он может быть крайне неудачен: например, он будет провоцировать длительное движение в *тупик*, когда у нас есть короткий *путь* к цели. Даже в теоретических исследованиях приходится накладывать условия на метод ленивого движения, чтобы гарантировать достижение результата (см., например, теорему о полноте семантических таблиц в книге [20]).

Другой крайний случай движения по сети, когда *сеть* делится на одинаковые слои. Например, в сети (14.1.4)

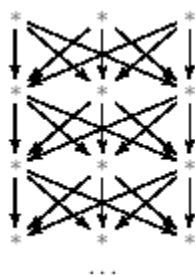


Рисунок 5 – Полностью заменяемый массив

Можно представить слой *статическим массивом* и вроде бы полностью забыть о самой сети. Забытый *призрак* мстит за себя, в частности, при необходимости *распараллеливания вычислений*, и предыдущий случай отнюдь не эквивалентен следующему.

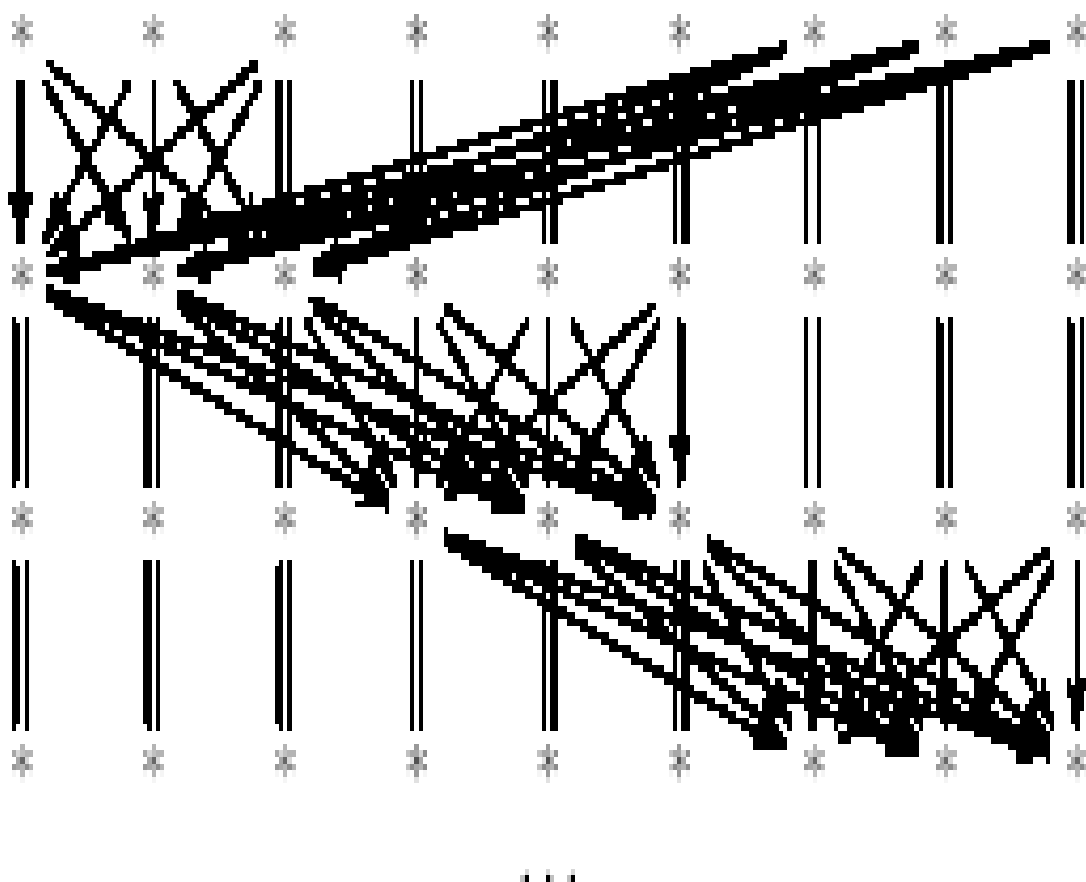


Рисунок 6 -- Постепенно заменяемый массив

Конечно же большая *сеть данных* становится необозримой. Справиться с ней можно, лишь разбив *сеть* на подсети. Таким образом, блоки программы соответствуют относительно автономным подсетям.

Еще Э. Дейкстра в книге [\[11\]](#) предложил в каждом блоке описывать импортированные и экспортируемые им глобальные значения. Но такая "писанина" раздражала хакеров и в итоге так и не вошла в общепризнанные системы программирования. Сейчас индустриальные технологии требуют таких описаний, но из-за отсутствия поддержки на

уровне синтаксического анализа все это остается благими пожеланиями, так что, если хотите, чтобы Ваша *программа* была понятна хотя бы Вам, описывайте все перекрестные информационные связи!

Резюмируя вышеизложенное, можно сделать следующие выводы.

1. *Сеть данных* сама по себе в программу не переходит, в программу переходят лишь некоторые свойства сети в качестве *призраков* и некоторые куски сети в качестве *реальных значений*.
2. Программа определяется не только *сетью данных*, но и конкретной дисциплиной движения по этой сети.
3. В случае, если очередные слои сети, появляющиеся при движении согласно заданной дисциплине, примерно одинаковы и состоят из многих взаимосвязанных значений, у нас возникает циклическая программа.
4. В случае, если вычисление можно свести к вычислениям для независимых элементов сети, чаще всего удобней *рекурсия*.
5. Самый общий способ движения по сети - ленивое движение, когда мы имеем право вычислить следующий объект сети, если вычислены все его предшественники.
6. Структурное программирование нейтрально по отношению к тому, каким именно способом будет исполняться полученная программа: последовательно, детерминированно, недетерминированно, совместно, параллельно либо даже на распределенной системе, - поскольку сеть лишь частично предписывает порядок действий.
7. Конкретный выбор порядка действий в последовательной детерминированной программе является *подпоркой*, о которой постоянно забывают. Поэтому он чаще всего вредит при перестройке программы.

Внимание!

Поскольку структурное программирование отработано и преподается лучше всего, здесь появилась возможность остановиться на гораздо более глубоких и принципиальных вопросах, чем в других стилях. В частности, призраки и подпорки возникают везде, они не являются спецификой структурного программирования, это важнейшие понятия для спецификации, документации и перестройки любой нетривиальной программы.

+++++

Выбор

Рассмотренные до сих пор *сети данных* представляли в первую очередь тот случай, когда в программе нет значительных альтернативных блоков. Условие было лишь средством проверки перехода от одного этапа вычислений к другому. Однако на самом деле, как правило, *программа* содержит выбор. Для представления выбора в языках программирования имеются условные *операторы* и *операторы* выбора. Рассмотрим, что же стоит за выбором.

Пример 14.3.1. Пусть в некоторый момент исполнения программы Вам необходимо временно выбросить больший из двух хранимых в основной памяти обрабатываемых блоков на *диск*. Поскольку разница в длине менее $2^{16}=65536$) незначительна, мы можем записать выбор примерно в следующей форме.

```

if
length(A)-length(B)>65536 -> {
    Save(A); Dispose(A); A_present:=false;},
length(A)-length(B)<65536 -> {
    Save(B); Dispose(B); B_present:=false;}
fi

```

Мы воспользовались данной формой, чтобы ярче подчеркнуть условия, при которых производятся действия.

Предложенная форма записи базируется на концепции *охраняемых команд*, предложенной Э. Дейкстрой. **Охраняемая команда** выполняется **лишь** при условии, когда выполнена охрана. Но если данный текст читает программист, он должен понимать, что «лишь» не всегда означает, что при выполнении условия команда будет выполнена. *Оператор выбора* по Дейкстре состоит из множества *охраняемых команд*. В конкретном синтаксисе мы используем для них форму

Guard -> Command

Относительное расположение *охраняемых команд* в операторе выбора безразлично⁸. Выполняется **одна** из *команд*, охрана которой истинна. Имеющиеся в языках конкретные формы условных предложений и предложений выбора являются *подпорками* для реализации *охраняемых команд*.

Из изложенного следует, что по своей сути выбор так же недетерминирован, как и *исполнение* структурной программы. Если выполнено несколько охран, с точки зрения задачи абсолютно все равно, какое из действий выбирать. Однако имеющиеся средства программирования⁹ заставляют нас однозначно сделать выбор, и конечно же почти всегда мы забываем написать в комментариях, что на самом деле выбор безразличен, а затем при модификациях программы появляются *заплатки* на *подпорках* и т. п.

Когда имеется выбор, мы вынуждены переходить от *сети данных* к более сложной структуре: &- $\bigvee$ -графам. Некоторые вершины могут быть помечены как $\bigvee$ -вершины, это означает, что достаточно получить один из результатов, соответствующий входящим дугам, и инициировать лишь одно из исполнений, соответствующее выходящей дуге. Для структурированности &- $\bigvee$ -графа необходимо, чтобы он был сетью, удовлетворяющей следующему условию: имеется *инъекция* ψ , сопоставляющая каждой $\bigvee$ -вершине ν , из которой выходит несколько дуг, $\bigvee$ -вершину $\psi(\nu)$, из которой выходит лишь одна дуга, такую, что любой *путь*, проходящий через первую вершину, проходит и через вторую. Это неудобоваримое теоретическое условие всего лишь формулирует на точном языке, что $\bigvee$ -вершины должны группироваться в структуры следующего вида, показанного на [Рисунок --14.2](#) (количество вариантов может быть любым).

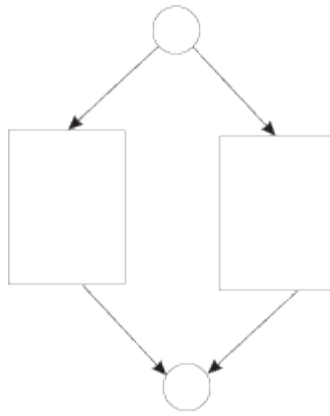


Рисунок 7 --Сеть охраняемых команд

+++++

О дисциплине циклического структурного программирования

Сейчас сосредоточимся на том варианте структурного программирования, который ориентируется на *циклы* и массивы.

Прежде всего, нужно остановиться на совместимости структурного циклического программирования с *рекурсиями*. Опыт показывает, что процедура, в которой есть ее *рекурсивный вызов* внутри *цикла*, практически почти всегда ошибочна, теоретически же она выходит за рамки примитивной *рекурсии* (см. курс логики и теории алгоритмов) и, как следствие, становится практически невычислимой. С тем, чтобы здесь не попасть впросак, соблюдайте простое правило.

Внимание!

Не используйте рекурсивный вызов процедуры внутри цикла! Рекурсия и циклы должны быть "территориально разделены"!

Данное правило пригодно в подавляющем большинстве случаев. Оно является конкретизацией для структурного программирования известного политико-социологического наблюдения:

Самые яростные противоречия возникают либо между двумя близкими сектами, либо при борьбе двух фракций одной и той же секты.

Люди старшего поколения еще помнят, как во время ожесточенной вражды между китайскими и советскими коммунистами китайцы не переставая повторяли, что у них "из десяти пальцев девять - общие".

Тем не менее иногда бывают исключения. Рассмотрим, например, схему поиска вглубь на дереве.

```
int search (ELEMENT x)
ELEMENT y; int result;
if (good(x)){
    return id(x)}
else for(int i=0; i<100; i++)
    {y=get_successor(x,i);
    result=search(y);
    if (result>0) return result;
```

```

    }
    return 0;
}

```

14.4.1.

Здесь *рекурсии* вместе с *циклом* задают *обход дерева* возможностей, и гибельного размножения рекурсивных вызовов не происходит. Причина этого исключения в том, что *цикл* в данной программе - всего лишь *подпорка* для *рекурсии*. В обычном программировании нет функционалов типа *mapcar* языка *LISP*, применяющих свой первый *аргумент* ко всем членам второго¹⁰.

Структурное *программирование* основано на предположении о локальности действий и условий, поэтому для него в особенности органично подходит иерархическое *разбиение* задачи на подзадачи - так называемое ***нисходящее планирование***.

При *нисходящем планировании* начинают с предположения о том, что поставленная задача решена. Затем пытаются реализовать решение общей задачи, выделяя те его блоки, которые оказались неэлементарными, в конструкции следующего уровня. В итоге мы получаем структуру взаимоотношений между конструкциями второго уровня и спецификации на каждую из этих конструкций. Если мы реализуем каждую из конструкций в точном соответствии с полученной спецификацией, то получится решение общей задачи.

В *планировании процессов* деятельности каждая из подзадач является процессом, чаще всего реализуемым отдельным человеком или группой людей, в конструировании технических систем каждая из подзадач дает блок конструкции, в программировании она дает *модуль* (в наиболее часто встречающихся достаточно простых случаях - подпрограмму).

Нисходящее *программирование* - конкретизация *нисходящего проектирования* для нужд программирования. Оно обладает следующими преимуществами.

- Нисходящее программирование породило хорошо согласованное с ним нисходящее построение структур данных, а нисходящее построение структур данных является простейшим частным случаем глубокой идеальной концепции - абстракции данных. Согласно парадоксу изобретателя, даже простейшие частные случаи высокоуровневых концепций при удачном применении улучшают все характеристики умственных конструкций (в частности, программ).
- При таком проектировании на каждом уровне можно ограничиваться одной моделью вычислений, в частности операционной, на которую ориентировано структурное программирование.
- Возможно раннее программирование, когда прототип программы, работающий хотя бы в условиях грубой эмуляции будущих решений низкого уровня, позволяет продемонстрировать, как будет работать полная программа, и улучшить ее пользовательские характеристики.

Ни один конкретный способ анализа либо синтеза не является универсальным. Нисходящее *программирование*, конечно же, тоже таковым не является. Причины тому, в частности, следующие¹¹.

- Невозможность при нисходящем структурном программировании увидеть тождественность процедур, работающих на разных ветвях декомпозиции, а тем более

унифицировать несколько формально различных процедур на разных ветвях в одну общую.

- Недостаточный учет особенностей, которые могут возникнуть после конкретной реализации запланированных блоков программы, что довольно часто вызывает необходимость полной перепланировки системы после реализации ее блоков, а вслед за такой перепланировкой реализованные для других спецификаций блоки часто оказываются отнюдь не лучшими из возможных.
- Недостаточный перенос опыта и наработок из одного проекта в другой.
- В нынешних методиках *нисходящего проектирования* ко всему перечисленному добавляется еще и навязывание последовательного стиля мышления, что мешает воспользоваться преимуществами нетрадиционных машин.

Это указывает на необходимость средств, позволяющих поднимать уровень понятий. Поэтому в разработке структурных программ применяется также подход, получивший название *восходящего программирования*. К примеру, когда строят библиотеку, занимаются обобщением задачи. Части, выделяемые в виде библиотечных средств, выбираются таким образом, чтобы они были применимы в различных контекстах.

При создании глобального контекста могут применяться всевозможные стили. Например, модули могут программироваться в стиле автоматного программирования. Если разделение стилей по модулям, пакетам и другим автономным структурным единицам проведено строго, то эклектического их смешения не происходит.

Реальный проект¹² - это смесь нисходящего и восходящего подходов:

- сначала сверху вниз для выяснения крупных строительных блоков;
- затем попытка движения снизу вверх, чтобы спроецировать понятия, оформившиеся ранее, на абстрактные структуры, допускающие адекватную реализацию;
- далее проверка соответствия, углубление нисходящей декомпозиции и обобщение понятий, выделенных восходящими приемами.

Как в нисходящем, так и в восходящем подходе важно обеспечить согласование потоков управления и потоков данных. Основным инструментом здесь может быть взгляд на программу со стороны *сетей данных*. Поскольку *цикл* появляется как реализация послойного движения по сети, а *массив* – как *представление* слоя сети, то видно, что на самом деле основной информационной структурой *цикла* является слой сети. Например, в *цикле*, реализующем числа Фибоначчи, - это структура из **fib1** и **fib2**. Структура, представляющая очередной слой сети, называется **реальным параметром** (в отличие от формального параметра, диктуемого языком программирования и часто являющегося *подпоркой*) *цикла*. Реальный *параметр* мы будем называть просто параметром.

Далее, поскольку *цикл* возникает при повторении однородных действий, а действия диктуются свойствами реальных объектов программы, в *цикле* должно быть общее свойство, зависящее от параметра и называемое **инвариантом цикла**. Например, в *цикле* сортировки - это соотношение между отсортированными и неотсортированными фрагментами массива.

При формулировке инварианта *цикла* мы практически с неизбежностью наталкиваемся на парадокс изобретателя, который заставляет нас вводить призрачные значения, например *дерево* разбиения массива на отсортированные подмассивы в эффективных алгоритмах сортировки.

Методика циклического и рекурсивного структурного программирования с использованием инвариантов и недетерминированности (но без явного упоминания *сетей данных*) прекрасно изложена в учебных пособиях Алагича, Арбиба и Гриса [11], [9]. Настольной книгой программиста должна служить также книга Кормена и др. (учебник MIT) [13], в которой можно найти множество хороших примеров того, как находить правильные программные решения и со вкусом использовать *подпорки*.

Сделаем еще один шаг. Поскольку *присваивание* с точки зрения решаемой задачи лишь средство задать элементы нового слоя сети, есть смысл считать, что во время всей итерации *цикла параметр* фиксирован, его изменение происходит лишь в промежутке между двумя итерациями. Таким образом, для **согласования потоков данных и потоков управления необходимо сосредоточить все присваивания реальным сущностям в одном месте: либо в конце, либо в начале очередной итерации**. Более того, в принципе (и даже не совсем теоретическом, но плохо поддерживаемом современными системами программирования) присваивания можно было бы вообще изгнать, сделав переход старого значения реального параметра *цикла* к новому неявным, так же, как это сделано для формального, скажем, в языке *Pascal*.

И, наконец, необходимо заметить следующее.

Внимание!

Призраки и инварианты необходимо осознавать до начала написания текста работающей программы. Если Вы оставите это на потом, то наверняка спутаете подпорки, которые займут у Вас основную часть мысленных ресурсов на завершающем этапе реализации, с сущностями. Более того, восстановить обоснование по тексту уже готовой программы, как ни парадоксально, обычно труднее, чем построить его заранее (то, что восстановить его во всяком случае не легче, было обосновано даже теоретически; смотри последнюю главу книги [20]).

+++++

Переходы и выдаваемые значения

В общем употреблении структурное *программирование* вошло после популяризовавшей его работы Э. Дейкстры, в которой, к сожалению, не было даже намек на его ограничения. Ограничения структурного программирования вытекают как из самой его сути, так и из *теоремы Бема-Джакопини*. Применение *структурных переходов*, которые ввел в практику и теорию Д. Кнут (откопавший оригинальную работу Бема-Джакопини и четко выделивший ограничения дейкстровского структурного подхода¹³), избавляет от многих недостатков, присущих методике *Дейкстры*. **Структурные переходы** - переходы лишь вперед и на более высокий уровень структурной иерархии управления, ни в каком случае не выводящие нас за пределы данного модуля.

/ Примеры структурных goto.*/*

...

```
do {y = f( x ,y)};  
  if (y>0) break;  
  x=g(x,y);  
}while (x>0);
```

```

...
{...
{
  if (All_Done) goto Success;
}
...
Success: Hurra;}
14.5.1.

```

Структурные переходы в настоящее время также включаются в общераспространенные языки программирования. Их использование понемногу проникает и в учебные курсы.

Структурные переходы являются паллиативом. Они возникли из-за необходимости выразить мысль о том, что успех либо неудача глобального процесса может выявиться внутри одной из решаемых подзадач, и дальнейшая работа над текущей задачей, и над всей последовательностью вложенных подзадач становится просто бессмысленной. В этом случае нужны даже не переходы, а *операторы* завершения. Но они во многих распространенных языках действуют лишь на один уровень иерархии вверх, а даже теоретически этого недостаточно¹⁴. Стоит заметить, что между идеей и ее корректной реализацией часто проходят долгие годы. Ныне в общераспространенном языке *Java* завершители наконец-то более или менее корректно реализованы.

Есть одно ограничение *структурных переходов*, известное с 80-х гг. XX века (см. [201]), по крайней мере один раз достоверно повредившее создателям отечественной серии машин Эльбрус, в которых на аппаратном уровне поддерживалось структурное и функциональное *программирование*. *Структурные переходы* (в том числе и завершители) некорректны, когда они выводят нас из функции-параметра вызова другой функции. Ирония в том, что абсолютно четкая и полная реализация завершителей еще до осознания необходимости данного средства и тем более задолго до осознания пределов его применимости была проделана именно там, где они в общем случае некорректны (в языке *LISP*). В нем, как мы видели, процедуры являются полноправными значениями, могут быть параметрами и результатами других процедур. Самый очевидный случай некорректности (правда, вылавливаемый системой обнаружения динамических ошибок *Common Lisp*), когда мы внутри созданной процедуры завершаем блок, существовавший в момент создания процедуры, но переставший существовать в тот момент, когда ее вызвали. Наверняка многие наталкивались на непонятное поведение программ с завершителями, но в соответствии с общераспространенным "позитивным" мышлением не обращали внимания на данный феномен.

Есть еще одна, на первый взгляд исключительно привлекательная, возможность. В обычных языках программирования часто раздражает то, что *значение*, выработанное один раз, не удастся сразу же использовать в следующем операторе, и, более того, по какой-то очевидной глупости совершенно безвредное и использованное еще в *Algol 60* понятие условного выражения, подобного

```
if x=0 then sin(y) else tan(y)
```

оказалось выброшено из некоторых распространенных языков (может быть, потому, что в данном случае часть *else* обязательна). В том же языке *LISP*, где была (достаточно уникальный случай в программировании) создана четкая и достаточно *замкнутая*

система концепций (ни одной неувязки, которую можно было диагностировать при тогдашнем состоянии теории и практики, найти в нем не удалось), каждый оператор выдает *значение*, которое может быть использовано объемлющим оператором.

Эту концепцию попытались перенести на язык традиционного типа в языке *Алгол 68*. На первый взгляд получилось очень красиво. Например, оператор

if x>0 then a1 else a2 fi [if y>0 then j else i fi]:=

if z>0 then sin(u) else tan(u) fi

намного компактней и красивей последовательности условных операторов, которую придется написать в *Pascal* или C++. Но концепция вырабатываемого значения оказалась в *концептуальном противоречии* и с оператором присваивания, и с совместными вычислениями. Например, согласно семантике Алгола 68, следующая *запись*

(x:=3, y:=x+y, z:=x+y)

является блоком программы, в котором все три присваивания исполняются совместно. Но очевидно, что в данном случае результат вычислений неоднозначен. Одновременно, поскольку значения не забываются, эта же конструкция является изображением массива из трех элементов либо записи с тремя полями.

Следовательно, если распространить систему выдаваемых значений на все *операторы* структурного программирования, то появляются формально допустимые и соблазнительные для хакерских трюков неустойчивые выражения. *Выражение*, изменяющее свой *контекст* (в частности, *присваивание*), выдавать *значение* не должно, чтобы не было соблазна поместить его в окружение, где оно будет вызывать трудно диагностируемую неоднозначность.

+++++

Общая характеристика структурного программирования

На самом деле изложение структурного стиля не может уместиться в рамки одной лекции. Но данный стиль программирования (вернее, его вариант, основанный на *циклах* и массивах, слегка пополненный рекурсивными процедурами) описывается и навязывается как единственно возможный во всех ныне предлагаемых учебных пособиях по программированию на традиционных языках. В связи с этим мы **имеем право** предположить, что обучающийся знаком с ним (более того, знаком только с ним, и мы надеемся, что он еще не потерял способность воспринимать другие стили). И хотя Вы считаете, что с этим вариантом структурного стиля уже освоились, особенности, опускаемые в традиционных изложениях, могут полностью изменить Ваш взгляд на данный стиль.

Мы рассматриваем структурное *программирование* как равноправный член сообщества альтернативных ему друзей-соперников¹.

* * *

Начнем с того, что обратимся к истории.

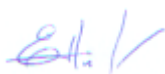
В теории схем программ было замечено, что некоторые случаи блок-схем легче поддаются анализу [16]. Поэтому естественно было выделить такой *класс* блок-схем, что и сделали итальянские ученые С. Бем и К. Джакопини в 1966 г. Они доказали, что любую блок-схему можно привести к структурированному виду, используя несколько дополнительных булевых переменных. Э. Дейкстра подчеркнул, что программы в таком

виде, как правило, являются легче понимаемыми и модифицируемыми, так как каждый блок имеет один вход и один *выход*.

ЛИСТ
изменений рабочей учебной программы по учебной дисциплине
МДК.01.01 Разработка программных модулей
Дополнения и изменения, вносимые в рабочую программу модуля

Основания внесения дополнений и изменений	Раздел РПД, в который вносятся изменения	Содержание вносимых дополнений, изменений
Предложение работодателя		
Предложение составителя программы	Обновление списков использованных источников	
Приобретение, издание литературы, обновление перечня и содержания ЭБС, баз данных		

Составитель: преподаватель



Е.А. Поддубная

Утверждена на заседании предметной (цикловой) комиссии профессиональных дисциплин программирования в компьютерных системах

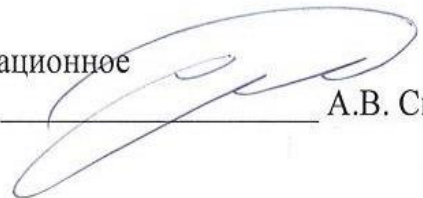
Протокол № 10 от 27 мая 2025 г.

Председатель предметной (цикловой) комиссии профессиональных дисциплин специальности **09.02.07 «Информационные системы и программирование»** и **09.02.03 «Программирование в компьютерных системах»**

 _____ Л.А. Благова
подпись

Заместитель директора по УР филиала _____  Т.А. Резуненко

Заведующая сектором библиотеки филиала _____  Л.Г. Соколова

Инженер-электроник (программно-информационное обеспечение образовательной программы) _____  А.В. Сметанин

Рецензия
на рабочую программу по междисциплинарному комплексу
МДК.01.01 «Разработка программных модулей» для специальности 09.02.07
«Информационные системы и программирование»

Рабочая программа междисциплинарного комплекса разработана на основании Федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.07 «Информационные системы и программирование», учебного плана и примерной программы

Программа состоит из титульного листа, листа согласования, содержания, паспорта программы, структуры и содержания междисциплинарного комплекса, описания применяемых образовательных технологий, условий реализации программы, перечня основной и дополнительной литературы, методических указаний для обучающихся, оценочных средств для контроля и оценки результатов освоения программы и примера дополнительного обеспечения междисциплинарного комплекса.

В паспорте программы междисциплинарного комплекса изложены область применения программы, место междисциплинарного комплекса в структуре основной профессиональной образовательной программы, цели и задачи междисциплинарного комплекса, количество часов, отведённое на освоение междисциплинарного комплекса.

Структура и содержание программы междисциплинарного комплекса включают в себя объём междисциплинарного комплекса, виды учебной работы, тематический план и содержание учебной дисциплины.

В тематическом плане программы междисциплинарного комплекса отражены наименования разделов и тем, содержание учебного материала, перечислены дидактические единицы, максимальная нагрузка обучающегося, темы практических занятий, самостоятельная работа. Содержание программы рассчитано на 343 часа.

Тематика программы охватывает в достаточном объёме вопросы отладки и тестирования ПО различными способами, а также документирование ПО в соответствии с ЕСКД.

Оформление соответствует всем предъявляемым требованиям. Содержание программы междисциплинарного комплекса включает 8 разделов. Темы и виды самостоятельной работы подобраны с целью закрепления получаемых на аудиторных занятиях навыков. Рабочая программа изложена грамотно, язык и стиль соответствуют общепринятым нормам, профессиональная лексика употребляется правильно.

Программа имеет методическую ценность и может быть использована для обучения в среднем профессиональном образовании.

Рецензент

Директор ООО «Современные
информационные технологии»



А.В.Сметанин

20.05.2025

Рецензия
на рабочую программу по междисциплинарному комплексу
МДК.01.01 «Разработка программных модулей» для специальности 09.02.07
«Информационные системы и программирование»

Рабочая программа междисциплинарного комплекса разработана на основании Федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.07 «Информационные системы и программирование», учебного плана и примерной программы

Программа состоит из титульного листа, листа согласования, содержания, паспорта программы, структуры и содержания междисциплинарного комплекса, описания применяемых образовательных технологий, условий реализации программы, перечня основной и дополнительной литературы, методических указаний для обучающихся, оценочных средств для контроля и оценки результатов освоения программы и примера дополнительного обеспечения междисциплинарного комплекса.

В паспорте программы междисциплинарного комплекса указаны область применения программы, место междисциплинарного комплекса в структуре ОПОП по специальности «Информационные системы и программирование», цели и задачи МДК.01.01 «Разработка программных модулей», количество часов, отведённое на освоение междисциплинарного комплекса.

Структура и содержание программы междисциплинарного комплекса включают в себя объём междисциплинарного комплекса, виды учебной работы, тематический план и содержание учебной дисциплины.

В тематическом плане программы МДК указаны наименования разделов и тем, содержание учебного материала, перечислены дидактические единицы, максимальная нагрузка обучающегося, темы практических занятий, самостоятельная работа. Содержание программы рассчитано на 343 часа. Рабочая программа МДК рассматривает темы, указанные во ФГОС и примерной программе. Рабочая программа МДК рассматривает темы, «Отладка и тестирование программного обеспечения» и «Документирование» программы по ЕСКД.

Оформление рабочей программы соответствует всем предъявляемым требованиям. Рабочая программа междисциплинарного комплекса содержит 8 разделов. Тематика самостоятельной работы и ее формы соответствуют поставленной цели: закреплению получаемых на аудиторных занятиях навыков.

Рабочая программа написана грамотно, в соответствии с научным и деловым стилем изложения, а также общепринятыми нормами. Профессиональная лексика употребляется правильно.

Программа может быть использована для обучения в среднем профессиональном образовании по указанной специальности.

Рецензент

Заместитель директора по научной работе филиала ФГБОУ ВО «КубГУ» в г.

Геленджике



Л.В. Галицкая, к.т.н.

20.05.2025