МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«БРАТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра управления в технических системах

УТВЕРЖДАЮ:
Проректор по учебной работе
_____ Е.И. Луковникова
«_____» _____ 201__ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ ПРОГРАММИРОВАНИЕ СЕТЕВЫХ ПРИЛОЖЕНИЙ

Б1.В.ДВ.05.02

НАПРАВЛЕНИЕ ПОДГОТОВКИ

11.03.02 Инфокоммуникационные технологии и системы связи

ПРОФИЛЬ ПОДГОТОВКИ

Многоканальные телекоммуникационные системы

Программа академического бакалавриата

Квалификация (степень) выпускника: бакалавр

СОДЕРЖАНИЕ ПРОГРАММЫ **Стр.**

1. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ, СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Вид деятельности выпускника

Дисциплина охватывает круг вопросов, относящихся к проектной и экспериментально-исследовательским видам профессиональной деятельности выпускника в соответствии с компетенциями и видами деятельности, указанными в учебном плане.

Цель дисциплины

Изучение языка программирования Java и современных технологий эффективной разработки и использования локальных, корпоративных и глобальных сетей для решения практических задач.

Задачи дисциплины

Формирование у студентов базовых понятий и навыков создания сетевых программных комплексов в операционной среде Windows.

Код компетенции	Содержание компетенций	Перечень планируемых результатов обучения по дисциплине
1	2	3
ОПК-4	способность иметь навыки самостоятельной работы на компьютере и в компьютерных сетях, осуществлять компьютерное моделирование устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ	**знать:** - основные термины и процессы компьютерного моделирования. **уметь:** - самостоятельной работы на компьютере и в компьютерных сетях. **владеть:** - навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ.
ПК-9	умение проводить расчёты по проекту сетей, сооружений и средств инфокоммукаций в соответствии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ	**знать:** - основные типы сигналов, используемых в телекоммуникационных системах; **уметь:** - проводить расчёты по проекту сетей, сооружений и средств инфокоммуникаций в соотвествии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ; **владеть:** - техникой инженерной и компьютерной графики (ввод, вывод, отображение, преобразование и редактирование графических объектов на компьютере).

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Дисциплина Б1.В.ДВ.05.02 программирование сетевых приложений относится к вариативной части.

Дисциплина программирование сетевых приложений базируется на знаниях, полученных при изучении таких учебных дисциплин, как: «Математический анализ», «Информатика», «Инженерная и компьютерная графика», «Компьютерные технологии».

Основываясь на изучении перечисленных дисциплин, программирование сетевых приложений представляет основу для изучения дисциплин: «Вычислительная техника и

информационные технологии», «Цифровая обработка сигналов», «Основы построения инфокоммуникационных систем и сетей».

Такое системное междисциплинарное изучение направлено на достижение требуемого ФГОС уровня подготовки по квалификации бакалавр.

3. РАСПРЕДЕЛЕНИЕ ОБЪЕМА ДИСЦИПЛИНЫ

3.1. Распределение объема дисциплины по формам обучения

Форма обучения	*Курс*	*Семестр*	*Трудоемкость дисциплины в часах*						*Курсовая работа*	*Вид промежуточной аттестации*
			Всего часов (с экз.)	*Аудиторных часов*	*Лекции*	*Лабораторные работы*	*Практические занятия*	*Самостоятельная работа*		
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Очная	2	3	180	51	17	34	-	75	КР	экзамен
Заочная	-	-	-	-	-	-	-	-	-	-
Заочная (ускоренное обучение)	-	-	-	-	-	-	-	-	-	-
Очно-заочная	-	-	-	-	-	-	-	-	-	-

3.2. Распределение объема дисциплины по видам учебных занятий и трудоемкости

Вид учебных занятий	*Трудоемкость (час.)*	*в т.ч. в интерактивной, активной, инновационной формах, (час.)*	*Распределение по семестрам, час*
			3
1	**2**	**3**	**4**
I. Контактная работа обучающихся с преподавателем (всего)	51	12	51
Лекции (Лк)	17	6	17
Лабораторные работы (ЛР)	34	6	34
Курсовая работа	+	-	+
Индивидуальные (групповые) консультации	+	-	+
II.Самостоятельная работа обучающихся (СР)	75	-	75
Подготовка к лабораторным работам	25	-	25
Подготовка к экзамену в течение семестра	25	-	25
Выполнение курсового работы	25	-	25
III. Промежуточная аттестация экзамен	54	-	54
Общая трудоемкость дисциплины …..... час.	180	-	180
зач. ед.	5	-	5

4. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

4.1. Распределение разделов дисциплины по видам учебных занятий

- для очной формы обучения:

№ раздела и темы	*Наименование раздела и тема дисциплины*	*Трудоемкость, (час.)*	*Виды учебных занятий, включая самостоятельную работу обучающихся и трудоемкость; (час.)*		
			учебные занятия		
			лекции	*лабораторные работы*	*самостоятельная работа обучающихся*
1	2	3	4	5	7
1.	**Языки гипертекстовой разметки документов**	**14**	**2**	**-**	**12**
1.1	Языки гипертекстовой разметки документов (HTML, DHTML, XML, XSL). Клиентские скрипты (JavaScript, VbScript).	3,5	0,5	-	3
1.2	Введение в язык разметки HTML, основные элементы и структура HTML, разработка HTML-документов.	3,5	0,5	-	3
1.3	DHTML- назначение и особенности его использования. Понятие и особенности применения клиентских скриптов.	3,5	0,5	-	3
1.4	Язык описания стилей отображения XML документов XSL и особенности его приме-нения.	3,5	0,5	-	3
2.	**Язык Java. Обзор базовых конструкций и основных элементов языка**	**13**	**2**	**5**	**6**
2.1	Особенности форматов основных выражений. Особенности объявления и использования типов данных. Операторы языка.	5	1	2	2
2.2	Динамическая инициализация, область действия и время жизни переменных. Преобразование и приведение типов, расширение типов.	5	1	2	2
2.3	Особенности объявления структурных типов. Особенности разработки архитектуры приложений в Java.	3	-	1	2
3.	**Объектно-ориентированное программирование на языке Java**	**14**	**2**	**6**	**6**
3.1	Классы и объекты. Динамическая инициализация объектов. Методы класса, конструкторы, параметризация методов. Использование ключевого слова this. Перегрузка и переопределение методов.	6	1	3	2
3.2	Ограничения и управления доступом. Вложенные и внутренние классы. Основы наследования. Создание многоуровневой иерархии.	6	1	3	2
3.3	Переопределение методов и их применение. Интерфейсы. Определение, реализация и применение интерфейсов.	5	-	3	2
4.	**Основы ввода/вывода. Работа со строковыми данными**	**16**	**1**	**6**	**9**
4.1	Чтение консольного ввода. Чтение символов. Чтение строк. Запись консольного вывода. Чтение и запись файлов.	6,25	0,25	3	3
4.2	Классы и интерфейсы ввода/вывода Java. Буферизированные байтовые потоки. Символьные потоки. Классы и интерфейсы потоков ввода/вывода Java.	6,5	0,5	3	3
4.3	Обработка строк. String-конструкторы. Специальные строковые операции. Преобразование и изменение	6,25	0,25	3	3

	строк. Методы для работы со строками.				
5.	**Методы и средства обработки исключений**	**7**	**1**	**-**	**6**
5.1	Необходимость обработки исключительных ситуаций. Основные принципы обработки исключений. Типы исключений.	3,5	0,5	-	3
5.2	Использование операторов try и catch. Оператор throw. Блок finally. Встроенные исключения Java. Создание собственных подклассов исключений.	3,5	0,5	-	3
6.	**Средства для организации работы в сети**	**12**	**1**	**5**	**6**
6.1	Основы работы в сети. Клиент-сервер. Proxy-серверы. Адресация Internet. Сетевые классы и интерфейсы. Класс InetAddress. Обзор сокетов. Зарезервированные сокеты. Сокеты TCP/IP клиентов. Сокеты TCP/IP серверов.	6,5	0,5	3	3
6.2	Дейтаграммы. Использование URL. Основные классы и интерфейсы реализации сетевого взаимодействия. Распределенная обработка данных. Вызов удаленных методов (RMI).	5,5	0,5	2	3
7.	**Многопоточное программирование**	**10**	**1**	**-**	**9**
7.1	Поточная модель Java. Класс Thread и интерфейс Runnable. Главный поток. Создание потока. Реализация интерфейса Runnable.	3,2	0,2	-	3
7.2	Создание множественных потоков. Приоритеты потоков. Синхронизация и передача сообщений. Использование синхронизированных методов.	3,5	0,5	-	3
7.3	Межпоточные связи. Блокировка. Приостановка, возобновление и остановка потоков.	3,3	0,3	-	3
8.	**Апплеты и события**	**19,5**	**3**	**6**	**10,5**
8.1	Основы апплетов. Класс Applet. Архитектура апплета. Инициализация и завершение апплета. Простые методы отображения апплетов. Требование перерисовки.	7,5	1	3	3,5
8.2	Пересылка параметров в апплеты. Основные методы класса Applet. Вывод в консоль.	7,5	1	3	3,5
8.3	Обработка событий. Модель делегирования событий. Источники событий. Классы событий. Обработка событий мыши. Обработка событий клавиатуры.	7,5	1	3	3,5
9.	**Разработка пользовательского интерфейса в Java**	**20,5**	**4**	**6**	**10,5**
9.1	Основы оконной графики. Класс Component. Класс Container. Класс Panel. Класс Window. Класс Frame. Работа с фреймовыми окнами. Создание фрейм-окна в апплете. Обработка событий фрейм-окна. Отображение информации в окне.	7,8	1,3	3	3,5
9.2	Работа с графикой. Работа со шрифтами. Управление текстовым выводом. Использование элементов управления, менеджеров компоновки и меню AWT.	7,9	1,4	3	3,5
9.3	Элементы управления. Основные понятия. Добавление и удаление элементов управления. Реагирование на элементы управления. Понятие менеджера компоновки. Работа с меню и диалоговыми окнами.	7,8	1,3	3	3,5
	ИТОГО:	**126**	**17**	**34**	**75**

4.2. Содержание дисциплины, структурированное по разделам и темам

№ раздела и темы	*Наименование раздела и темы дисциплины*	*Содержание лекционных занятий*	*Вид занятия в интерактивной, активной, инновационной формах, (час.)*
1	**2**	**3**	**4**
1.	**Языки гипертекстовой разметки документов**		
1.1	Языки гипертекстовой разметки документов (HTML, DHTML, XML, XSL). Клиентские скрипты (JavaScript, VbScript).	Документ HTML 4 состоит из трех частей: • строка, содержащая информацию о версии HTML, • объявляющий раздел header/"шапка" (ограниченный элементом HEAD), • тело, содержащее собственно сам документ. Тело может содержаться в элементах BODY или FRAMESET. *Пробельные символы* (пробелы, *символы новой строки*, символы табуляции и комментарии) могут появляться до или после этого раздела.	-
1.2	Введение в язык разметки HTML, основные элементы и структура HTML, разработка HTML-документов.	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> <HTML> <HEAD> <TITLE> Simple page </TITLE> </HEAD> <BODY> <H1>Hello world!</H1> </BODY> </HTML> Документ начинается с элемента *типа* документа, или doctype. Он описывает, какой тип HTML будет использован - чтобы клиентское*приложение* пользователя могло определить, как интерпретировать документ, и решить, следует ли он тем правилам, которым собирался следовать *по* своему заявлению. После этого можно видеть открывающий *тег* элемента html. Это *оболочка* вокруг всего документа. Закрывающий *тег* html является последним объектом в любом документе HTML. Внутри элемента html имеется элемент head. Он содержит информацию о документе (*метаданные*). Внутри head находится элементtitle, который определяет заголовок "Simple page" в панели *меню*. После элемента head следует элемент body, который является оболочкой, содержащей реальное содержимое страницы - в данном случае только элемент заголовка первого уровня (h1), который содержит текст "Hello world!" . Элементы часто содержат другие элементы. *Тело документа* всегда будет содержать множество вложенных друг в друга элементов. *Разделы* страницы создают общую структуру документа, и могут содержать подразделы. Они также могут содержать *заголовки*, параграфы, списки и т.д. Параграфы могут содержать элементы, которые создают ссылки на другие элементы, цитаты, выделения и т.д.	-
1.3	DHTML-назначение и особенности его использования. Понятие и особенности применения клиентских скриптов.	1. Публикация во всемирной паутине (WWW – World Wide Web) реализуется в форме **Web-сайтов**. Web-сайт по своей структуре напоминает журнал, который содержит информацию, посвященную какой-либо теме или проблеме. Как журнал состоит из печатных страниц, так и Web-сайт состоит из компьютерных Web-страниц. Сайт является ***интерактивным*** средством представления информации. Интерактивность сайта обеспечивают различные формы, с помощью которых посетитель сайта может зарегистрироваться на сайте, заполнить анкету и т. д.	-

		Обычно сайт имеет титульную страницу (страница с оглавлением), на которой имеются гиперссылки на его основные разделы (Web-странички). Гиперссылки имеются и на других Web-страницах сайта, что позволяет посетителю свободно перемещаться по сайту. Web-сайты обычно являются ***мультимедийными***, т. к. на них кроме текста есть иллюстрации, анимация, звуковая и видеоинформация. **Web-сайт состоит из Web-страниц, объединенных гиперссылками. Web-страницы могут быть интерактивными и могут содержать мультимедийные и динамические объекты (изменение страницы по алгоритму).** 2. Создание Web-сайтов реализуется с использованием языка гипертекстовой разметки – **HTML (HyperText Markup Language)**. **HTML – набор команд(*тэгов*), вставляемых в текст WWW-страницы и определяющих форматирование абзацев, вид шрифта, ссылки на внешние файлы и другие страницы.** Технология HTML состоит в том, что в обычный текстовый документ вставляются управляющие символы (тэги) и в результате мы получаем Web-страницу. Браузер (например, Internet Explorer) при загрузке Web-страницы представляет её на экране в том виде, который задаётся тэгами. Основными достоинствами HTML-документов являются: o малый информационный объём; o возможность просмотра на ПК, оснащенных различными операционными системами. 3. Для создания Web-страниц используются простейшие текстовые редакторы, которые не включают в создаваемый документ управляющие символы форматирования текста. В качестве такого редактора в Windows можно использовать стандартное приложение**Блокнот**. Создаваемая Web-страничка сохраняется с расширением .htm (.html) в Блокноте, затем запускается в **браузере IE** через меню ***Файл/Открыть***. Если вам необходимо редактировать страничку, то пользуйтесь командой главного меню IE – ***Вид/В виде HTML*** (открывается Блокнот для редактирования). Не забудьте сохранить изменения в Html-коде Web-страницы и только затем возвращайтесь в браузер для её просмотра, использовав команду ***Вид/Обновить*** **(<*F*5>).**	
1.4	Язык описания стилей отображения XML документов XSL и особенности его применения.	Вместе с документами XML тоже можно применять таблицы стилей CSS. Для этого в язык XML введена инструкция по обработке stylesheet, которая используется так, как показано в листинге 8.4. **Листинг 8.4. Документ XML, использующий таблицу стилей CSS** <?xml version="1.0" encoding="windows-1251"?> <?xml:stylesheet type="text/css" href="xmlcontract.ess"?> <contract> <type>TpyflOBoii floroBop</type> <name>Иванов Петр Сидорович</name> <date>01.07.03</date> <period>12</period> <i— и Так далее... —> </contract> Всякий браузер, "понимающий" XML, например Mozilla или Internet Explorer 6.x, покажет приведенный в листинге 8.4 документ в соответствии со стилями, записанными в файле xmlcontractcss. Таблица стилей, включенная в файл xmlcontract.css, оформляется по правилам CSS и может выглядеть так, как показано в листинге 8.5. **Листинг 8.5. Таблица стилей для документа XML** type { margin-top: 8em; margin-bottom: 3ет; font-size: 2em; color: blue } name { text-indent: 2em; margin-top: 3ет; font-size: 1.5em; margin-bottom: 2em; } date, period { margin: 0.5em; } Хотя таблицы стилей CSS и можно использовать в XML, но реализация стилей для документов XML должна быть другой. Как видно из приведенных примеров, синтаксис CSS весьма оригинален и никак не похож на синтаксис XML. Кроме того, стили CSS определяют способы показа документа HTML в окне браузера, его визуализацию, а язык XML выявляет структуру документа, ничего не говоря о его представлении в виде, удобном для чтения. Поэтому в технологии XML для записи стилей был разработан специ-	-

		альный язык XSL — одна из реализаций XML. **Язык описания стилей XSL** Таблицы стилей для документов XML записываются при помощи специально сделанной реализации языка XML, названной XSL (XML Stylesheet Language). Язык XSL, как и язык XPath, представляет документ в виде дерева. Процессор языка XSL преобразует это дерево, руководствуясь таблицей стилей, и форматирует его для вывода в окно браузера, на принтер, экран проектора или на какое-то другое устройство. Таким образом, обработка проходит два этапа: преобразование дерева документа (XML transform) и форматирование (formatting) дерева, полученного после преобразования. Первый этап — этап преобразования — может быть достаточно сложным и кардинально поменять структуру дерева: изменить уровни вложенности, удалить или добавить новые узлы, создать оглавление, предметный указатель, индекс. Результат преобразования может стать новым самостоятельным документом или даже несколькими документами. Таблица стилей, по которой идет преобразование, содержит правила, состоящие из двух частей: образцов (patterns) для отбора узлов, предназначенных для преобразования, и шаблонов (templates) или конструкторов (sequence constructors) для построения преобразованных узлов. Второй этап — этап форматирования — абстрагируется от конечного устройства, хотя может выполняться непосредственно в нем, например, в браузере, пейджере, принтере, проекторе. Форматирование формулируется в терминах классов и объектов. Под объектами форматирования FO (formatting objects) понимаются узлы дерева, а под их классами — некие конечные структуры: страницы, абзацы, таблицы, списки. Таблица стилей определяет правила форматирования (formatting properties). По этим правилам строится дерево, но уже не дерево узлов, а дерево геометрических областей (area tree), на которые разбивается документ, и определяются их характеристики: размеры, цвет, шрифт. Вскоре после выхода рекомендации языка XSL стало ясно, что преобразование документа XML — это самостоятельная и независимая задача, которую можно выполнять не только для приведения их к одному стилю, но и для многих других целей. Например, можно преобразовать документ XML в документ HTML, XHTML или даже в документ PDF. Можно обновить документ, изменив некоторые узлы, или разделить его на несколько документов. Поэтому преобразование документов XML было выделено в отдельную область исследования и описано отдельным языком XSLT (XSL Transformation), первая версия которого изложена в рекомендации "XSL Transformation (XSLT). Version 1.0". После выделения преобразований в отдельный язык XSLT первоначальная рекомендация языка XSL была переработана, сейчас основной акцент в ней сделан на форматировании, поэтому ее часто называют рекомендацией "XSL-FO". Мы рассмотрим форматирование в следующей главе, а в этой займемся преобразованием документов XML с помощью языка XSLT.	
2.	**Язык Java. Обзор базовых конструкций и основных элементов языка**		
2.1	Особенности форматов основных выражений. Особенности объявления и использования типов данных. Операторы языка.	**Характеристики операций** • Ассоциативность (левоассоциативные, правоассоциативные) • Количество аргументов (унарные, бинарные, тернарные) • Приоритет • Префиксная и постфиксная форма (для ++, — —) **Ассоциативность** a=b=c=d=e a+b+c+d+e **Аргументы** a+b !a (a>b)?c:d **Приоритет** a+b*c/d-e a=b==c	Лекция с текущим контролем (1 час)

		Приоритет операций **Результат операций** В языке Си у операций могут быть следующие результаты: • Числовая константа, означающая результат присвоения, математического выражения, адрес ячейки памяти. • Числовая константа, означающая логическое значение (0 - ложь, 1 - истина). • Ссылка на область памяти. В Си не имеется встроенного логического типа со значениями **true** и **false**, но любое *ненулевое*целое значение означает **истину**, а *нулевое* - **ложь.**	
2.2	Динамическая инициализация, область действия и время жизни переменных. Преобразование и приведение типов, расширение типов.	**Область видимости и время жизни переменных** Существует два основных вида области видимости: локальная область видимости и глобальная область видимости. Переменная, объявленная вне всех функций, помещается в глобальную область видимости. Доступ к таким переменным может осуществляться из любого места программы. Такие переменные располагаются в глобальном пуле памяти, поэтому время их жизни совпадает со временем жизни программы. Переменная, объявленная внутри блока (часть кода, заключенная в фигурные скобки), принадлежит локальной области видимости. Такая переменная не видна (поэтому и недоступна) за пределами блока, в котором она объявлена. Самый распространенный случай локального объявления – переменная, объявленная внутри функции. Переменная, объявленная локально, располагается на стеке, и время жизни такой переменной совпадает со временем жизни функции. Так как областью видимости локальной переменной является блок, в котором она объявлена, то существует возможность объявлять переменные с именем, совпадающим с именами переменных, объявленных в других блоках; а также объявленных на более верхних уровнях, вплоть до глобального. Ее область видимости – только цикл for, за пределами этого цикла действует другая переменная с тем же именем, объявленная в начале функции. Кроме того, в теле цикла объявлена переменная k, областью видимости которой является тело цикла. Локальные переменные можно объявлять со спецификатором доступа static. В этом случае компилятор располагает такую переменную в глобальном пуле памяти. Поэтому, время жизни статической переменной совпадает со временем жизни программы. При этом область видимости такой переменной ограничивается пределами блока, в котором она объявлена.	Лекция с текущим контролем (1 час)
2.3	Особенности объявления структурных типов. Особенности разработки архитектуры приложений в Java.	**Структуры языка Си** *Структуры, как агрегатный тип данных* Агрегатными называют типы данных, которые содержат в себе элементы других типов данных. Один из примеров – массивы. Агрегатные типы имеют внутреннюю организацию, которая называется структурой. Во многих задачах бывает удобно объединить в одном объекте данных данные разных типов. Массивы для этого не годятся. Поэтому в языках программирования имеются инструменты, позволяющие объединять в единый объект разнородные данные. В Паскале это записи, в Си – это структуры. Отдельные элементы структуры будем называть полями. Если в массивах доступ к отдельным элементами обеспечивается с помощью числового индекса, то в структурах каждое поле получает своё имя. Поэтому объявление структурного типа выглядит сложнее, чем объявление массива. Для массива достаточно задать тип элементов и их количество, а для структуры нужно определить тип и имя каждого поля. Определяя структурный тип, мы создаём новый тип данных, который удобен для решения нашей задачи. Разработка структурных типов относится к этапу проектирования. Структуры ещё не являются полноценным типом данных, потому что тип данных это множество значений + множество операций с ними. Операции со структурами можно выполнять с помощью функций, но эти функции определяются отдельно от структурного типа. **Объявление структурных типов.** Объявление структурного типа начинается со служебного слова struct.	

		Обычно структурный тип имеет имя, которое иногда называют тегом. Тег это ярлык. Сам структурный тип определяется как совокупность определений отдельных полей, заключенный в фигурные скобки.
		struct complex {
		float real ;
		float imag;
		} ;
		или можно было описать в одной строке – float real, imag;
		Выше объявлен структурный тип с названием struct complex
		Struct complex z;
		Объявление структурного типа это объявление шаблона, по которому будут создаваться объекты этого типа.
		Поля структурного типа могут иметь базовые типы данных, быть массивами или структурными данными. Однако поле структуры не может иметь тот же тип что и сама структура.
		struct my_struct {
		…
		struct my_struct ms;
		…
		} ;
		Это невозможно, потому что когда компилятор создаёт объект структурного типа он должен выделить для него память, а для этого сначала нужно вычислить размер структуры. В общем случае размер структурного типа является результатом суммирования отдельных его полей. Если поле имеет тот же тип что и сама структура, то возникает бесконечная рекурсия, и размер подсчитать нельзя. Однако поле структуры может иметь тип указателя на свою структуру.
		После того, как объявлен структурный тип, он может использоваться не только для определения переменных, но и при описании аргументов функций и возвращаемого значения. Есть возможность совместить определение структурного типа с определением переменных:
		Struct my_struct {
		float real;
		float image;
		} ; z1,z2, zz[10];
		Объявления структурных типов могут быть размещены на глобально м или локальном уровне. На локальном уровне они должны быть помещены в начало блока вместе с остальными определениями. Расположение объявления влияет на область действия объявления. Можно создавать структурный тип без имени. Это имеет смысл в том случае, если нам нужно только в одном месте программы определить переменную этого структурного типа. Анонимным структурным типом больше нигде нельзя будет воспользоваться. Имена структурных типов могут совпадать с именами своих полей и с именами переменных.	

3.	**Объектно-ориентированное программирование на языке Java**		
3.1	Классы и объекты. Динамическая инициализация объектов. Методы класса, конструкторы, параметризация методов. Использование ключевого слова this. Перегрузка и переопределение методов.	Объекты в Java создаются с помощью зарезервированного слова new, после которого идет конструктор - специальная подпрограмма, занимающаяся созданием объекта и инициализацией полей создаваемого объекта. Для него не указывается тип возвращаемого значения, и он не является ни методом объекта (вызывается через имя класса когда объекта еще нет), ни методом класса (в конструкторе доступен объект и его поля через ссылку this). На самом деле конструктор в сочетании с оператором new возвращает ссылку на создаваемый объект и может считаться особым видом методов, соединяющим в себе черты методов класса и методов объекта. Если в объекте при создании не нужна никакая дополнительная инициализация, можно использовать конструктор, который по умолчанию присутствует для каждого класса. Это имя класса, после которого ставятся пустые круглые скобки - без списка параметров. Такой конструктор при разработке класса задавать не надо, он присутствует автоматически. Если требуется инициализация, обычно применяют конструкторы со списком параметров. Порядок вызовов при создании объекта некого класса (будем называть его дочерним классом): 1. Создается объект, в котором все поля данных имеют значения по	Компьютерная презентация (1 час)

		умолчанию (нули на двоичном уровне представления). 2. Вызывается конструктор дочернего класса. 3. Конструктор дочернего класса вызывает конструктор родителя (непосредственного прародителя), а также по цепочке все прародительские конструкторы и инициализации полей, заданных в этих классах, вплоть до класса Object. 4. Проводится инициализация полей родительской части объекта значениями, заданными в декларации родительского класса. 5. Выполняется тело конструктора родительского класса. 6. Проводится инициализация полей дочерней части объекта значениями, заданными в декларации дочернего класса. 7. Выполняется тело конструктора дочернего класса.	
3.2	Ограничения и управления доступом. Вложенные и внутренние классы. Основы наследования. Создание многоуровневой иерархии.	**Система контроля и управления доступом** (СКУД) — совокупность программно-аппаратных технических средств безопасности, имеющих целью ограничение и регистрацию входа-выхода объектов (людей, транспорта) на заданной территории через «точки прохода»: двери, ворота, КПП. Основная задача — управление доступом на заданную территорию (кого пускать, в какое время и на какую территорию), включая также • ограничение доступа на заданную территорию • идентификация лица, имеющего доступ на заданную территорию Дополнительные задачи: • учёт рабочего времени; • расчет заработной платы (при интеграции с системами бухгалтерского учёта); • ведение базы персонала / посетителей; • интеграция с системой безопасности, например: • с системой видеонаблюдения для совмещения архивов событий систем, передачи системе видеонаблюдения извещений о необходимости стартовать запись, повернуть камеру для записи последствий зафиксированного подозрительного события; • с системой охранной сигнализации (СОС), например, для ограничения доступа в помещения, стоящие на охране, или для автоматического снятия и постановки помещений на охрану. • с системой пожарной сигнализации (СПС) для получения информации о состоянии пожарных извещателей, автоматического разблокирования эвакуационных выходов и закрыванияпротивопожарных дверей в случае пожарной тревоги. На особо ответственных объектах сеть устройств СКУД выполняется физически несвязанной с другими информационными сетями.	Компьютерная презентация (1 час)
3.3	Переопределение методов и их применение. Интерфейсы. Определение, реализация и применение интерфейсов.	**Переопределение метода** (англ. *Method overriding*) в объектно-ориентированном программировании — одна из возможностей языка программирования, позволяющая подклассу или дочернему классу обеспечивать специфическую реализацию метода, уже реализованного в одном из суперклассов или родительских классов. Реализация метода в подклассе переопределяет (заменяет) его реализацию в суперклассе, описывая метод с тем же названием, что и у метода суперкласса, а также у нового метода подкласса должны быть те же параметры или сигнатура, тип возвращаемого результата, что и у метода родительского класса[1]. Версия метода, которая будет исполняться, определяется объектом, используемым для его вызова. Если вызов метода происходит от объекта родительского класса, то выполняется версия метода родительского класса, если же объект подкласса вызывает метод, то выполняется версия дочернего класса[2]. Некоторые языки программирования позволяют программисту защищать методы от переопределения.	
4.	**Основы ввода/вывода. Работа со строковыми данными**		
4.1	Чтение консольного ввода. Чтение символов. Чтение строк. Запись консоль-	**Чтение консольного ввода** В Java 1.0 единственным способом выполнения консольного ввода было использование байтового потока, и существует большой объем старого кода, в котором применяется этот подход. Сегодня применение байтового потока для чтения консольного ввода попрежнему технически возможно, но	-

	ного вывода. Чтение и запись файлов.	поступать так не рекомендуется. Предпочтительный метод чтения консольного ввода – это использовать символ – орентированный поток, что значительно упрощает возможности интернационализации и поддержки разрабатываемых программ. В Java консольный ввод выполныется чтением System.in. Чтобы получить символьный поток, присоединенный к консоли, вы должны поместить System.in в оболочку объекта BufferedReader. BufferedReader поддерживает буфферизованный входной поток. Наиболее часто импользуемый его конструктор выглядит так:
		
		BufferedRader(Reader inputReader)
		
		Приведенной выше примере inputReader – это поток, который связывается с создаваемым экземпляром BufferedRader. Rader – абстрактный класс. Одним из его конскретных наследников является InputStreamReader, который преобразует байты в символы. Для получения объектаInputStreamRaeder, который присоединен к System.in можно применить следующую строку кода:
		
		InputStreamReader()
		
		Поскольку System.in ссылается на объект типа InputStream, он должен быть использован как параметр inputStream. Собрав все вместе, получим следующую строку кода, которая создает BufferedReader, соединенный с клавиатурой:
		BufferaedReader br = new BufferedReader(new InputStreamReader(System.in));
		
		После выполнения этого оператора br представляет собой основанный на символах поток, подключенный к консоли через System.in.
		
		Чтение символов
		
		Для чтения символов из BufferedReader применяется read(). Ниже показана версия read(), которая будет использоваться:
		
		Int read() throws Ioexception
		
		Каждый раз, когда вызывается метод read(), он читает символ из входного потока и возвращает его как целое значение. При достижении конца потока везвращается -1. Как видите, метод может возбудить исключение IOException. В следующей программе демонстрируется применение read(), читая символы с консоли до тех пор, пока не пользователь не ведет "q”. Обратите внимание, что любые исключения ввода - вывода, которые могут быть сгенирированы, просто передаются в main(). Такой подход распространен при чтении с консоли, но при жеании вы можете обработать ошибки такого рода самостоятельно.
		//Использование BufferedReader для чтения символов с консоли.
		//Import java.io.*;
		//Class BRRead{
		//Public static void main(Straing args[]) throws IOException
		//{
		//char c;
		//BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		//System.out.println("Вводите символы, ’q’ – для вывода.”);
		//// читать символы
		//do{
		//c = (char) br.read();
		//System.out.println(c);
		//}while(c!= ‘q’);
		//}
		//}
		
		Ниже показан пример запуска этой прогаммы:	

		Вводите симолы, ‘q’ – для вывода. 123abcq 1 2 3 A B C Q Этот вывод может выгдядеть немного не так, как вы ожидали, потому что System.in Является строчно – буферизованными по цмолчанию. Это значит, что никакого ввода действительности программе предается до тех пор, пока будет нажата клавиша (ENTER). Как можно предположить, это делает read() лишь отчасти пременимым для интерактивного консольного.	
4.2	Классы и интерфейсы ввода/вывода Java. Буферизированные байтовые потоки. Символьные потоки. Классы и интерфейсы потоков ввода/вывода Java.	Создание хорошей системы ввода/вывода (I/O) является одной из наиболее сложных задач для разработчиков языка. Доказательством этому служит наличие множества различных подходов. Сложность задачи видится в охвате всех возможностей. Не только различный исходный код и виды ввода/вывода, с которыми вы можете общаться (файлы, консоль, сетевые соединения), но так же вам необходимо общаться с ними большим числом способов (последовательный, в случайном порядке, буферный, бинарный, посимвольный, построчный, пословный и т.п.). Разработчики библиотеки Java атаковали эту проблему путем создания множества классов. Фактически, существует так много классов для системы ввода/вывода в Java, что это может сначала испугать (по иронии, дизайн ввода/вывода Java I/O на самом деле предотвращает взрыв классов). Также произошли значительные изменения в библиотеке ввода/вывода после версии Java 1.0, когда изначально **byte**-ориентированная библиотека была пополнена **char**-ориентированными, основанными на Unicode I/O классами. Как результат, есть некоторое количество классов, которые необходимо изучить прежде, чем вы поймете достаточно хорошо картину ввода/вывода Java и ее правильно использовать. Кроме того, достаточно важно понимать историю эволюции библиотеки ввода/вывода, даже если вашей первой реакцией было: “не надоедайте мне историей, просто покажите мне, как использовать это!” Проблема в том, что без исторической перспективы вы постоянно будете смущаться некоторыми классами, определяя, когда вы должны, а когда не должны использовать их. Эта глава даст вам введение в различные классы ввод/вывода стандартной библиотеки Java и расскажет о том, как их использовать. **Класс File** Прежде чем перейти к классам, которые действительно читают и записывают данные в поток, мы посмотрим на утилиты, обеспечивающиеся библиотекой в помощь вам в обработке директории файлов. Класс **File** имеет обманчивое имя — вы можете подумать, что он ссылается на файл, но это не так. Он может представлять либо *имя* определенного файла, либо *имя* набора файлов в директории. Если это набор файлов, вы можете опросить набор с помощью метода **list( )**, который вернет массив **String**. Есть смысл возвращать массив, а не гибкий класс контейнера, потому что число элементов фиксировано, и если вам нужен список другого директория, вы просто создаете другой объект **File**. Фактически, “FilePath” был бы лучшим именем для класса. Этот раздел покажет пример использования этого класса, включая ассоциированный **FilenameFilter interface.**	-
4.3	Обработка строк. String-конструкторы. Специальные строковые операции. Преобразование и изменение строк. Методы для работы со строками.	**Класс String.** Класс**String** является основным классом, предназначенным для хранения и обработки строк символов. Для создания экземпляров класса **String** может быть использован один из следующих конструкторов: String() String(String str) String(StringBu er strbuf) String(**char**[] arr) String(**char**[] arr,**int** first,**int** count) Первый из них создаёт пустую строку, второй и третий копируют содержимое объектов классов **String** и**StringBu er** в созданный объект. По-	-

		следние два конструктора позволяют создать строку на основе символьного массива или его части. Кроме того, любая объектная ссылка типа**String** может быть проинициализирована посредством присвоения ей строкового литерала: String filename = "data.txt"; Возможно, несколько неожиданной особенностью класса **String** является то, что экземпляры этого класса не могут быть изменены после их создания (immutable). Однако это не создаёт ограничений для их использования, поскольку все методы, которые должны были бы изменять строку, просто создают новую модифицированную строку, оставляя исходную без изменений. Поясним работу этого механизма на примере: String s = "abcd"; s = s.toUpperCase(); Здесь метод **toUpperCase()** создаёт новую строку, содержащую последовательность символов "ABCD", и возвращает ссылку на эту строку, которая присваивается переменной**s**, старое значение переменной теряется. Исходная строка остаётся в неизменном виде и, поскольку на неё больше не осталось объектных ссылок, будет удалена сборщиком мусора.	
5.	**Методы и средства обработки исключений**		
5.1	Необходимость обработки исключительных ситуаций. Основные принципы обработки исключений. Типы исключений.	В языке C# есть операторы, позволяющие обнаруживать и обрабатывать ошибки (исключительные ситуации), возникающие в процессе выполнения программы. Об этом уже упоминалось в разделе «Введение в исключения» (см. с. 46), а сейчас мы рассмотрим механизм обработки исключений более подробно. *Исключительная ситуация,* или *исключение,* — это возникновение аварийного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы, например делением на ноль или переполнением. Обычно эти события приводят к завершению программы с системным сообщением об ошибке. C# дает программисту возможность восстановить работоспособность программы и продолжить ее выполнение. Исключения C# не поддерживают обработку асинхронных событий, таких как ошибки оборудования или прерывания, например нажатие клавиш Ctrl+C. Механизм исключений предназначен только для событий, которые могут произойти в результате работы самой программы и указываются явным образом. Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом другая часть программы может попытаться сделать что-нибудь иное. *Исключения позволяют логически разделить вычислительный процесс на две части — обнаружение аварийной ситуации и ее обработка.* Это важно не только для лучшей структуризации программы. Главное то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может знать, что делать, но не уметь определить место возникновения. Это особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей. Другое достоинство исключений состоит в том, что для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение или параметры, поэтому заголовки функций не разрастаются.	-
5.2	Использование операторов try и catch. Оператор throw. Блок finally. Встроенные исключения Java. Создание собственных подклассов исключений.	Исключение Java представляет собой объект, который описывает исключительную (то есть, ошибочную) ситуацию, возникающую в части программного кода. Когда такая исключительная ситуация возникает, создается объект, представляющий исключение, который возбуждается в методе, вызвавшем ошибку. Этот метод может либо обработать исключение самостоятельно, либо пропустить его. В обоих случаях, в некоторой точке исключение перехватывается и обрабатывается. Исключения могут генерироваться системой времени выполнения Java, либо они могут быть сгенерированы вручную вашим кодом. Исключения, которые возбуждает Java, имеют отношение к фундаментальным ошибкам, которые нарушают правила языка Java либо ограничения системы выполнения Java. Вручную сгенерированные исключения обычно применяются для того, чтобы сообщить о некоторых ошибочных ситуациях тому, кто вызвал данный метод.	-

		Обработка исключений Java управляется пятью ключевыми словами: **try**, **catch**, **throw**, **throws** и **finally**. Если кратко, они работают следующим образом. Операторы программы, которые вы хотите отслеживать на предмет исключений, помещаются в блок try. Если исключение возникает в блоке **try**, оно возбуждается. Ваш код может перехватить исключение (используя **catch**) и обработать его некоторым осмысленным способом. Сгенерированные системой исключения автоматически возбуждаются системой времени выполнения Java. Чтобы вручную возбудить исключение, используется ключевое слово **throw**. Любое исключение, которое возбуждается внутри метода, должно быть специфицировано в его интерфейсе ключевым словом **throws**. Любой код, который в обязательном порядке должен быть выполнен после завершения блока **try**, помещается в блок **finally**. Ниже показана общая форма блока обработки исключений.
		try {
		// блок кода, в котором отслеживаются ошибки
		}
		catch (Тип_исключения_1 переменная_1) {
		// обработчик исключений типа ExceptionTypel
		}
		catch (Тип_исключения_2 переменная_2) {
		// обработчик исключений типа ExceptionType2
		}
		// ...
		finally {
		// блок кода, который должен быть выполнен после завершения блока try
		}
		Здесь Тип_исключения — тип исключения, которое возникает. Остаток настоящей главы посвящен описанию применения этой программной структуры.	

6.	**Средства для организации работы в сети**		
6.1	Основы работы в сети. Клиент-сервер. Proxy-серверы. Адресация Internet. Сетевые классы и интерфейсы. Класс InetAddress. Обзор сокетов. Зарезервированные сокеты. Сокеты TCP/IP клиентов. Сокеты TCP/IP серверов.	Глобальная сеть Интернет, ставшая символом нового этапа развития информационных технологий на рубеже веков, оказывает значительное влияние на работу специалистов в самых разных областях человеческой деятельности, в том числе и на работу специалистов-гуманитариев. По-видимому, роль Интернета в их профессиональной деятельности будет только возрастать. В связи с этим большое значение для специалистов приобретают знание основных возможностей сети и практические навыки работы в Интернете. Основные понятия и определения Идея соединения друг с другом нескольких компьютеров для обмена информацией между ними появилась почти одновременно с созданием первых компьютеров. Такое соединение по аналогии с телефонным стали называть **сетью ЭВМ, компьютерной сетью, информационной сетью, сетью передачи данных или просто сетью.** **ВНИМАНИЕ** Компьютерной сетью называется объединение двух и более вычислительных машин специальными средствами связи, с помощью которых можно осуществлять обмен информацией между любыми включенными в сеть компьютерами. Основные возможности и проблемы работы в сетях Специалисты очень быстро оценили основные достоинства компьютерных сетей. К их числу относятся: • возможность оперативного, практически мгновенного обмена информацией между пользователями, имеющими доступ к компьютерам сети; • возможность совместного использования дорогостоящей и эффективной аппаратуры, включенной в состав сети (например, лазерных принтеров); • возможность совместного использования программного обеспечения и данных, хранящихся в компьютерах сети, что позволяет экономить дисковую память из-за отказа от дублирования файлов на каждом из компьютеров;	-

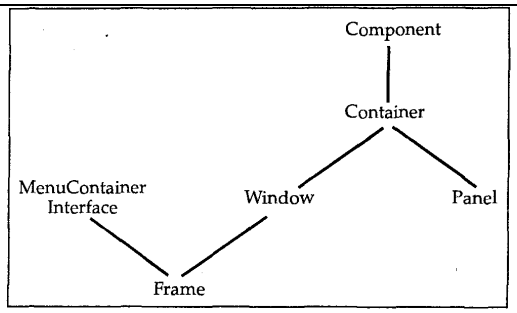
		• доступ к уникальной, то есть имеющейся в единичных экземплярах информации для большого числа людей; • возможность использования для обработки информации более мощных компьютеров; • возможность объединения вычислительных мощностей для решения сложных задач. Однако работа в сети выдвигает и целый ряд проблем: • сохранность ценной информации общего использования; • обеспечение надежности работы сетевой аппаратуры и сетевого программного обеспечения; • ограничение доступа к конфиденциальной информации; • защита от вирусов; • разрешение конфликтов, когда несколько пользователей одновременно пытаются использовать одну и ту же аппаратуру, одни и те же программы или данные и т. д.	

6.2	Дейтаграммы. Использование URL. Основные классы и интерфейсы реализации сетевого взаимодействия. Распределенная обработка данных. Вызов удаленных методов (RMI).	Для отправки и получения дейтаграмм используются классы Datagram Packet и DatagramSocket. Эти объекты создаются и инициализируются по разному в зависимости от того, отправляете вы или принимаете дейтаграммы. Пример 5.11 показывает, как отправлять дейтаграммы. Пример 5.12 показывает, как принимать дейтаграммы и как узнать, кем они были отправлены.
		Чтобы отправить дейтаграмму, сначала создается объект DatagramPac ket, содержащий передаваемые данные, их длину, узел и порт этого уз ла, на которые посылается дейтаграмма. Затем для отправки пакета применяется метод send() объекта DatagramSocket. Объект DatagramSoc ket является универсальным, он создается без аргументов. Он может использоваться многократно для отправки любого пакета по любому адресу и порту.
		Пример 5.11. UDPSend.java
		package com.davidflanagan.examples.net; import java.io.*;
		import java.net.*;
		/**
		*Этот класс отправляет заданный текст или файл в виде дейтаграммы
		*на заданный порт заданного узла.
		**/
		public class UDPSend {
		public static final String usage =
		"Формат: java UDPSend <hostname> <port> <msg>...\n" + " или: java UDPSend <hostname> <port> f <file>";
		public static void main(String args[]) {
			-

7.	**Многопоточное программирование**		
7.1	Поточная модель Java. Класс Thread и интерфейс Runnable. Главный поток. Создание потока. Реализация интерфейса Runnable.	При однопоточном программировании в бесконечном цикле выполняется один поток управления, который опрашивает единую очередь событий и принимает решение, какое действие выполнять следующим. Примером может быть процесс считывания информации из файла. После получения сигнала о готовности файла к считыванию управление передается соответствующему обработчику, и пока из этого обработчика не будет получен ответ, никаких новых действий не предпринимается. В противовес этому при многопоточном программировании один поток может делать паузу, не прерывая выполнение других потоков. Именно такой подход реализован в Java. Как и все в Java, поточная модель реализуется посредством иерархии классов, описывающих потоки. Основу этой иерархии составляют класс Thread и интерфейс Runnable. Для создания потока необходимо либо расширить класс Thread, либо реализовать интерфейс Runnable. При этом класс Thread инкапсулирует поток исполнения.	-
7.2	Создание множественных потоков. Приоритеты потоков. Синхронизация и передача сообще-	Язык Java поддерживает многопоточное программирование. Многопоточная программа содержит две или больше частей, которые могут выполняться одновременно. Каждая такая часть программы называется потоком. С понятием многопоточности тесно связано понятие многозадачности. Многозадачность реализуется, как правило, либо на потоках, либо на процессах. Различие между потоками и процессами достаточно зыбкое. Обычно для процессов выделяется отдельная область памяти, которая доступна	-

	ний. Использование синхронизированных методов.	только для этих процессов. Это повышает безопасность, но снижает скорость выполнения программы. На процессах основана работа операционных систем. При многопоточности обычно память по потокам не разбивается. Хотя такая ситуация может сказаться на стабильности программы, системные ресурсы используются экономнее и программа работает быстрее. Обычно многопоточное программирование применяют для сведения к минимуму времени простоя системы, поскольку сразу несколько задач могут выполняться одновременно.	
7.3	Межпоточные связи. Блокировка. Приостановка, возобновление и остановка потоков.	**Межпроцессное взаимодействие** (англ. ***i**nter-**p**rocess **c**ommunication*, **IPC**) — обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети. Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить: • механизмы обмена сообщениями; • механизмы синхронизации; • механизмы разделения памяти; • механизмы удалённых вызовов (RPC). Для оценки производительности различных механизмов IPC используют следующие параметры: • пропускная способность (количество сообщений в единицу времени, которое ядро ОС или процесс способно обработать); • задержки (время между отправкой сообщения одним потоком и его получением другим потоком). IPC может называться терминами ***межпотоковое взаимодействие*** (англ. ***i**nter-**t**hread **c**ommunication*) и ***межпрограммное взаимодействие*** (англ. ***i**nter-**a**pplication **c**ommunication*). Межпроцессное взаимодействие, наряду с механизмами адресации памяти, является основой для разграничения адресного пространства между процессами.	-
8.	**Апплеты и события**		
8.1	Основы апплетов. Класс Applet. Архитектура апплета. Инициализация и завершение апплета. Простые методы отображения апплетов. Требование перерисовки.	Все апплеты являются подклассами Applet. Таким образом, они должны импортироватьjava.applet, а такжеjava.awt.AWT— сокращение Abstract Window Toolkit (абстрактный оконный интерфейс). Так как все апплеты выполняются в окне, необходимо включить поддержку для этого окна. Апплеты не исполняются Java-интерпретатором времени выполнения, работающим в консольном режиме. Они выполняются или Web-браузером или программой просмотра апплета, называемойappletviewerи поставляемой с пакетом разработки JDK (Java Developer Kit, инструментарий разработчика Java). Выполнение апплета не начинается с метода main(). Некоторые из них даже содержат методmain(), однако выполнение апплета начинается и управляется совершенно иным механизмом, который требует хотя бы краткого объяснения. Вывод в окно апплета не выполняется методомsystem.out.println().Скорее, он обрабатывается различными AWT-методами, такими как drawstring(), который выводит строку в указанную точку экрана. Ввод также обрабатывается иначе, чем в приложении. Как только апплет откомпилирован, он включается в HTML-файл, используя тег<appiet>. Апплет будет выполнятьсяJava-совместимым браузером, когда тот встретит вHTML-файле указанный тег. Для более удобного просмотра и проверки апплета просто включите в начало файла исходного кодаJava-комментарий, который содержит тег<appiet>. Этим способом код документируется вместе с инструкциямиHTML, необходимыми апплету, и вы можете проверить откомпилированный апплет, запустив программу просмотра с вашим файлом исходного кода в качестве параметра. Пример такого комментария: /* <applet code="MyApplet" width=200 height=60> </applet>	-

		*/ Этот комментарий содержит тег <applet>, который выполнит апплет с именемMyAppiet в окне с размерами**200 x 60** пикселов. Так как включение команды<appiet> делает тестирование апплетов проще, все показанные далее апплеты будут содержать соответствующий тег, внедренный в комментарий.	
8.2	Пересылка параметров в апплеты. Основные методы класса Applet. Вывод в консоль.	Иногда желательно передать апплету, который встроен в HTML-документ, определенные параметры, например размеры апплета или какие-нибудь другие данные. Для этого используется тег <param>. В HTML-тексте каждого параметра, который требуется передать апплету, следует использовать тег <param>: <applet code=”NewApplet.class” Width = “400” height = “200”> <param name = “width” value = “400”> <param name = “height” value = “200”> </applet> Каждый параметр состоит из пары “имя/значение”, указываемой в <param> с помощью атрибутов name и value. Атрибут value служит для указания имени параметра, а value задает значение этого параметра.	-
8.3	Обработка событий. Модель делегирования событий. Источники событий. Классы событий. Обработка событий мыши. Обработка событий клавиатуры.	Модель событий, применяемая в Java 1.1, подходит для использования в AWT и в Swing. В этой модели каждое событие - это класс, наследуемый от класса java.util.EventObject. Все AWT события наследуются от класса java.awt.AWTEvent. Для удобства различные типы событий AWT помещены в отдельный пакет java.awt.event. Модель событий базируется на концепции "слушаю события". Объект, ожидающий какое-либо событие, "слушает" его, является event listener. Объект, вырабатывающий событие, (источник события) поддерживает список объектов, ожидающих событие, и оповещает все объекты в списке о его появлении. Источник события имеет методы для добавления объектов, ожидающих события, и методы для удаления таких объектов. Источник события при возникновении события запускает метод – обработчик события и передает в него объект типа EventObject или производного от него. Для того чтобы запустить нужный метод все объекты типа listener должны реализовывать определенный интерфейс. Интерфейс может определять несколько методов. Например, класс MouseEvent представляет несколько событий: нажатие кнопки, отпускание кнопки, и другие. В таблице 2.1 приводятся тип объекта, интерфейс для его обработки и методы, определяемые в каждом интерфейсе. Для каждого из интерфейсов, содержащих более одного метода, определен класс-адаптер, который содержит пустые тела методов. Классы-адаптеры имеют имена, такие же, как и имена интерфейсов с заменой Listener на Adapter: например, MouseAdapter, WindowAdapter. Если Вы реализовали интерфейс или создали класс адаптера, нужно создать объект для того, чтобы он "слушал" событие. Затем зарегистрировать этот объект в источнике события. В AWT источник события всегда компонента: кнопка, список, и т.д. Регистрация делается с помощью методов с именами вида **addXXXListener**. Удаление объекта из списка слушающих выполняется с помощью**removeXXXListener**. Здесь XXX - имя типа события, генерируемого источником.	-
9.	**Разработка пользовательского интерфейса в Java**		
9.1	Основы оконной графики. Класс Component. Класс Container. Класс Panel. Класс Window. Класс Frame. Работа с фреймовыми окнами. Создание фрейм-окна в	AWT определяет окна согласно иерархии классов, которая с каждым уровнем добавляет функциональные возможности и специфику. Два наиболее общих типа окон являются производными от типа Panel, который пользуется апплетами, и от типаFrame, который создает стандартное окно. Многое из функциональных возможностей этих окон получено от их родительских классов. Описание иерархии, имеющей отношение к этим двум классам, фундаментально для их понимания. Иерархия классов, связанных с классамиPanel иFrame, показана на рис. 1.	Компьютерная презентация (1 час)

	апплете. Обработка событий фрейм-окна. Отображение информации в окне.	 Рис. 1. Иерархия классов дляPanelиFrame	
9.2	Работа с графикой. Работа со шрифтами. Управление текстовым выводом. Использование элементов управления, менеджеров компоновки и меню AWT.	Основной класс для работы со шрифтами в GDI+ – это класс **Font**. Объекты этого класса представляют конкретные шрифты, установленные на компьютере. В этом классе предусмотрено множество перегруженных конструкторов, но наиболее часто используются следующие варианты: **//создаем объект Font, указывая имя шрифта и его размер** **Font f = new Font(“Times New Roman”, 12);** **//создаем объект Font, указывая имя, размер и стиль** **Font f2 = new Font(“WingDings”, 50, FontStyle.Bold |** **FontStyle.Underline);** При создании **f2** использовались стили из перечисления **FontStyle**. При этом можно задавать несколько стилей одновременно.	Компьютерная презентация (1 час)
9.3	Элементы управления. Основные понятия. Добавление и удаление элементов управления. Реагирование на элементы управления. Понятие менеджера компоновки. Работа с меню и диаловыми окнами.	*Элементы управления* (controls) — это компоненты, которые предоставляют пользователю различные способы взаимодействия с приложением (например, командная*кнопка* (push button)).*Менеджер компоновки* (layoutmanager) автоматически позиционирует (размещает, располагает) компоненты в контейнере. Вид окна, таким образом, определяется комбинацией элементов управления, содержащихся в окне, и менеджера компоновки, используемого для их размещения. В дополнение к элементам управления, фрейм-окно может также включать *строку меню* (menu bar) стандартного стиля. Каждый вход в строке меню активизирует раскрывающееся меню элементов, которые пользователь может выбирать. Строка меню всегда позиционируется наверху окна. Строки меню, хотя и различаются по виду, обрабатываются похожим способом, что и другие элементы управления. Хотя компоненты окна можно позиционировать вручную, это весьма утомительно. Менеджер компоновки предназначен для автоматизации этой задачи. В первой части данной главы, которая представляет различные элементы управления, будет использован менеджер компоновки, заданный по умолчанию. Он отображает компоненты в контейнере, размещая их по принципу "слева-направо, сверху-вниз". Затем будут рассмотрены все менеджеры компоновки. Там вы и увидите, как лучше управлять позиционированием элементов управления.	-

4.3. Лабораторные работы

№ п/п	*Номер раздела дисциплины*	*Наименование лабораторной работы*	*Объем (час.)*	*Вид занятия в интерактивной, активной, инновационной формах, (час.)*
1	**1.**	Разработка WEB – ресурса с использованием языков гипертекстовой разметки документов HTML и DHTML.	5	–
2	**1, 6.**	Создание и администрирования WEB – сервера.	5	Тренинги в малой группе (2 часа)
3	**1, 2, 3, 4.**	Разработка WEB – приложения с использованием JavaScript.	6	–
4	**2, 3, 4.**	Программирование мульти платформенного Java – приложения в интегрированной среде JBuilder	6	–
5	**2, 3, 4, 8.**	Создание WEB – ресурса с использованием апплетов языка Java.	6	Тренинги в малой группе (2 часа)
6	**2, 3, 4, 9.**	Разработка сетевого программного приложения в клиент-серверной архитектуре.	6	Тренинги в малой группе (2 часа)
		ИТОГО	**34**	**6**

4.4. Практические занятия

Учебным планом не предусмотрено.

4.5. Контрольные мероприятия: курсовая работа

Цель работы:

Закрепление и углубление теоретических знаний студентами по наиболее важным вопросам дисциплины, совершенствование у них навыков объектно-ориентированного проектирования программного обеспечения, умения проектировать и программировать дружественный интерфейс, взаимодействовать с базами данных, проверка способностей студентов квалифицированно применять полученные ими теоретические знания и практические навыки при решении задач.

Структура:

Пояснительная записка должна содержать титульный лист, задание, описание разработанных алгоритмов и приложения, блок схемы и листинг программы.

Основная тематика:

Разработка сетевого приложения с наличием графического интерфейса и базы данных, реализующего минимальный набор типовых функциональных возможностей.

Рекомендуемый объем работы*:* Курсовая работа должна включать титульный лист, задание, описание разработанных алгоритмов и программы, блок схему и листинг программы. Курсовая работа выполняется в виде пояснительной записки объемом 30 – 35 страниц.

Выдача задания, прием и защита КР проводится в соответствии с календарным учебным графиком.

Оценка	Критерии оценки курсовой работы
отлично	Курсовая работа сдана в первую неделю защит. В курсовой работе представлена разработка программы, включающей в себя алгоритмы по обработки линейных двухсвязных списков. Обучающийся владеет материалом и отвечает на все поставленные вопросы.
хорошо	Курсовая работа сдана в срок со второй по четвертую неделю защит или курсовой проект содержит незначительные ошибки. Обучающийся имеет недостаточные знания по данному материалу.
удовлетворительно	Курсовая работа сдана в срок с пятой недели по седьмую неделю защит или содержит значительное количество ошибок, или ошибка подразумевает полную переработку всего курсового проекта.
неудовлетворительно	Курсовая работа не сдана в установленный срок.

5. МАТРИЦА СООТНЕСЕНИЯ РАЗДЕЛОВ УЧЕБНОЙ ДИСЦИПЛИНЫ К ФОРМИРУЕМЫМ В НИХ КОМПЕТЕНЦИЯМ И ОЦЕНКЕ РЕЗУЛЬТАТОВ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Компетенции / *№, наименование разделов дисциплины*	*Кол-во часов*	*Компетенции*		*Σ комп.*	*tср, час*	*Вид учебной работы*	*Оценка результатов*
		ОПК-4	*ПК-16*				
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1. Инструментарий технологии программирования	14	-	+	1	14	Лк, СР	экзамен
2. Основные принципы и подходы к разработке программных алгоритмов	13	-	+	1	13	Лк, ЛР, СР	экзамен
3. Алгоритмы сортировки и поиска информации	14	+	-	1	14	Лк, ЛР, СР	экзамен
4. Основы программирования на языке высокого уровня Си++	16	+	-	1	16	Лк, ЛР, СР	экзамен
5. Типы данных, выражения и операции в языке программирования Си++	7	+	-	1	7	Лк, СР	экзамен
6. Операторы языка программирования Си++ и управление их исполнением	12	+	-	1	12	Лк, ЛР, СР	экзамен
7. Указатели и динамическое распределение памяти	10	+	-	1	10	Лк, СР	экзамен
8. Функции языка программирования Си++	19,5	+	-	1	19,5	Лк, ЛР, СР,	экзамен, КР
9. Статические и динамические структуры данных	20,5	+	-	1	20,5	Лк, ЛР, СР,	экзамен, КР
всего часов	**126**	**99**	**27**	**2**	**63**	-	-

6. ПЕРЕЧЕНЬ УЧЕБНО-МЕТОДИЧЕСКОГО ОБЕСПЕЧЕНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

1. Орлов С. А. Теория и практика языков программирования : учебник для бакалавров и магистров / С. А. Орлов. - Санкт-Петербург : Питер, 2014. - 688 с. - (Учебник для вузов. Стандарт третьего поколения). - ISBN 978-5-496-00032-1

7. ПЕРЕЧЕНЬ ОСНОВНОЙ И ДОПОЛНИТЕЛЬНОЙ ЛИТЕРАТУРЫ, НЕОБХОДИМОЙ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

№ п/п	*Наименование издания*	*Вид занятия*	*Количество экземпляров в библиотеке*, шт.	*Обеспеченность*, (экз./ чел.)
1	**2**	**3**	**4**	**5**
Основная литература				
1.	Подбельский В. В. Язык СИ++ : учебное пособие для вузов / В. В. Подбельский. - 5-е изд. - Москва : Финансы и статистика, 2007. - 559 с. - ISBN 5279022047	Лк	21	1
2.	Павловская Т. А. Паскаль. Программирование на языке высокого уровня: практикум : учеб. пособие для вузов / Т. А. Павловская. - Санкт-Петербург : Питер, 2006. - 317 с. - (Учебное пособие). - ISBN 5947230089	Лк	15	1
Дополнительная литература				
3.	Крупник А. Изучаем С++ : учебное пособие / А. Крупник. - Санкт-Петербург : Питер, 2003. - 250 с. - ISBN 5318000975	Лк	10	0,5
4.	Хусаинов Б. С. Структуры и алгоритмы обработки данных. Примеры на языке Си : учебное пособие для вузов / Б. С. Хусаинов. - Москва : Финансы и статистика, 2004. - 464 с. - ISBN 5279027758	Лк, КР	9	0,6
5.	Незнанов А. А. Программирование и алгоритмизация : учебник / А. А. Незнанов. - Москва : Академия, 2010. - 304 с. - (Высшее профессиональное образование. Информатика и вычислительная техника). - ISBN 978-5-7695-6767-4	Лк	10	0,5
6.	Макконелл Дж. Основы современных алгоритмов : учеб. пособие для вузов / Дж. Макконелл; Пер. с англ. - 2-е изд., доп. - Москва : Техносфера, 2006. - 368 с. - (Мир программирования). - ISBN 5-9483-6005-9	Лк	27	1
7.	Франка, П. С++ : учебный курс / П. Франка. - Санкт-Петербург : Питер, 2004. - 522 с. - (Учебный курс). - ISBN 5314001365	Лк	10	0,5
8.	Дьяконица, С. А. Программирование и основы алгоритмизации : лабораторный практикум / С. А. Дьяконица, Д. С. Семенов. - Братск : БрГУ, 2007. - 173 с.	ЛР	45	1
9.	Дьяконица, С. А. Основы программирования на языке Си/Си ++ : лабораторный практикум / С. А. Дьяконица, Д. С. Семенов. - Братск : БрГУ, 2015	ЛР	46	1
10.	Асламова В.С., Елькина И.М. Основы алгоритмизации и программирования. Часть третья. Структурированные типы данных: Учебно-методическое пособие для студентов факультета технической кибернетики. - Ангарск: АГТА, 2003. - 95 с. http://window.edu.ru/resource/095/62095	КР	ЭР	1
11.	Березин Б.И. Начальный курс С и С++: учебное пособие / Б.И. Березин, С.Б. Березин. – М.: Диалог-МИФИ, 2012. – 280 с. ISBN 5-96404-075-4; [Электронный ресурс] http://biblioclub.ru/index.php?page=book&id=448000	Лк, ЛР	ЭР	1
12.	Технология программирования/ Ю.Ю. Громов, О.Г. Иванова, М.П. Беляев, Ю.В. Минин – Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2013. – 173с. ISBN 978-5-8265-1207-4; [Электронный ресурс] http://biblioclub.ru/index.php?page=book&id=277802	Лк, ЛР	ЭР	1
13.	Лавлинский В.В. Технология программирования на современных языках программирования / В.В. Лавлинский, О.В. Коровина. – Воронеж: Воронежская государственная лесотехническая	Лк, ЛР	ЭР	1

	академия, 2012. – 118с. [Электронный ресурс] http://biblioclub.ru/index.php?page=book&id=142453			
14.	Хиценко В.П. Основы программирования: учебное пособие / В.П. Хиценко; Министерство образования и науки Российской Федерации, Новосибирский государственный технический университет. – Новосибирск: НГТУ, 2015. – 83 с. ISBN 978-5-7782-2704-4; [Электронный ресурс] http://biblioclub.ru/index.php?page=book&id=438365	Лк, ЛР	ЭР	1

8. ПЕРЕЧЕНЬ РЕСУРСОВ ИНФОРМАЦИОННО - ТЕЛЕКОММУНИКАЦИОННОЙ СЕТИ «ИНТЕРНЕТ» НЕОБХОДИМЫХ ДЛЯ ОСВОЕНИЯ ДИСЦИПЛИНЫ

1.Электронный каталог библиотеки БрГУ
http://irbis.brstu.ru/CGI/irbis64r_15/cgiirbis_64.exe?LNG=&C21COM=F&I21DBN=BOOK&P21DBN=BOOK&S21CNR=&Z21ID=.
2. Электронная библиотека БрГУ
http://ecat.brstu.ru/catalog .
3. Электронно-библиотечная система «Университетская библиотека online»
http://biblioclub.ru .
4. Электронно-библиотечная система «Издательство «Лань»
http://e.lanbook.com .
5. Информационная система "Единое окно доступа к образовательным ресурсам"
http://window.edu.ru .
6. Научная электронная библиотека eLIBRARY.RU http://elibrary.ru .
7. Университетская информационная система РОССИЯ (УИС РОССИЯ)
https://uisrussia.msu.ru/ .
8. Национальная электронная библиотека НЭБ
http://xn--90ax2c.xn--p1ai/how-to-search /.

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

9.1. Методические указания для обучающихся по выполнению лабораторных работ

Рекомендуемые источники литературы, необходимые при выполнении лабораторных работ указаны в п.7 (дополнительная [8-9, 11-14]).

Лабораторная работа № 1

Разработка WEB – ресурса с использованием языков гипертекстовой разметки документов HTML и DHTML.

Цель работы:
Выработать практические навыки работы с системой программирования языка Си/Си++, научиться создавать, вводить в компьютер, выполнять и исправлять простейшие программы на языке Си/Си++ в режиме диалога, познакомиться с диагностическими сообщениями компилятора об ошибках при выполнении линейных программ.

Порядок выполнения:
Все программы на языке Си и Си++ строятся по модульному принципу, т.е. состоят из множества модулей. Под модулем здесь имеется в виду библиотеки или ранее написанные части кода программы. Согласно принципам скрытия информации текст модуля обычно раз-

деляют на заголовочный файл с расширением **.h** или **.hpp** и файл реализации с одним из расширений **.c, .cpp, .obj, .lib, .asm.**

После того, как исходный текст программы сформирован, на основе его должен быть выполнен или создан исполняемый файл с расширением **.exe**. Этот процесс осуществляется в несколько этапов, структура такого процесса представлена на рис.1.

Вначале с помощью текстового редактора формируется и сохраняется в файлах с расширением **.cpp** исходный текст программы. Затем осуществляется этап препроцессорной обработки, содержание которого определяется директивами препроцессора, расположенными в начале исходного кода. В частности, по директиве **#include** препроцессор подключает к тексту программы заголовочные файлы стандартных библиотек. Результатом работы препроцессорной обработки является формирование полного текстового кода программы, который в явном виде не доступен программисту и достаточно объемен из-за подключаемых заголовочных файлов. Далее происходит компиляция программы, в ходе которой могут быть обнаружены синтаксические ошибки, которые должны быть устранены. В случае успешной компиляции получается объектный код программы в файле с расширением **.obj**. После компиляции выполняется этап компоновки и редактирования связей, осуществляемый с помощью встроенных системных программ языка. В результате компоновки создается исполняемый двоичный код программы в файле с расширением **.exe**, который запускается на выполнение и который является результатом написания программы.

Особенности выполнения перечисленных действий зависят от конкретного компилятора языка Си/Си++ и той операционной системы, в которой он работает. Технические подробности можно изучить по документации для конкретного программного продукта. Например, при работе с интегрированными средами фирмы Borland необходимая информация может быть получена из руководств пользователя.

В простейшем случае программа на языке Си и Си++ представляет собой набор описаний и определений, и состоит из набора функций. Среди этих функций всегда должна быть функция с именем **main** (для консольных программ) или **WinMain** (для программ в 32 – разрядных операционных систем). Данные функции являются точками входа и выхода, и без них программа не может быть выполнена. Если функция не возвращает никакого значения в результате своего выполнения, то перед именем функции помещается служебное слово **void**, обозначающее тип отсутствия значения. Также каждая функция должна иметь набор параметров, если параметры отсутствуют, то скобки оставляют пустыми или в них указывается **void**.

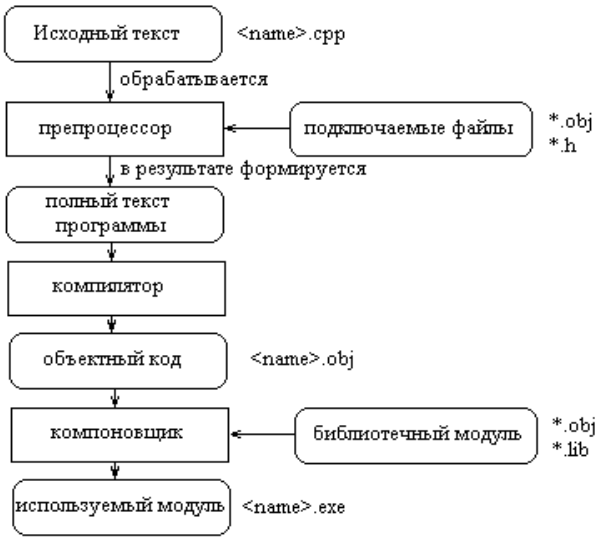


Рис.1 Структура процесса компиляции

За заголовком размещается тело функции. Тело функции - это последовательность определений, описаний и исполняемых операторов, заключенных в фигурные скобки. Ниже приведен листинг простейшей программы на языке Си/Си++, вычисляющая объем цилиндра по радиусу и высоте, значения которых вводятся с клавиатуры.

```
// подключение средств консольного потокового ввода/вывода
#include <iostream.h>
//подключение математических функций
#include <math.h>
/* главная функция программы */
void main ()
  {
// объявление переменных
int r,v,h;
//вывод на экран
cout <<"\nВведите радиус и высоту цилиндра:";
//вывод на экран
cin >> r >> h;
//ввод значений с клавиатуры
v=M_PI*r*r*h;
cout <<"\nобъем цилиндра = " << v;
  }
```

Варианты задания для выполнения лабораторной работы:

1. Заданы координаты трех вершин треугольника (x_1, y_1), (x_2, y_2),(x_3, y_3). Найти его периметр и площадь.
2. Длина отрезка задана в дюймах (1 дюйм = 2,54 см). Перевести значение длины в метрическую систему, то есть выразить ее в метрах, сантиметрах и миллиметрах. Например: 21 дюйм = 0 м 53 см 3.4 мм.
3. В такси одновременно сели три пассажира. Когда вышел первый пассажир, на счетчике было p_1 рублей; когда вышел второй p_2 рублей. Сколько должен был заплатить каждый пассажир, если по окончании поездки счетчик показал p_3 рублей? Плата за посадку составляет p_0 рублей.
4. Дана сторона равностороннего треугольника. Найти радиусы и площади вписанной и описанной окружностей.
5. Коммерсант, имея стартовый капитал *k* рублей, занялся торговлей, которая ежемесячно увеличивает капитал на *p*%. Через сколько лет, он накопит сумму *s*, достаточную для покупки собственного магазина?
6. Треугольник задан величинами своих углов и радиусом описанной окружности. Найти стороны треугольника.
7. Селекционер вывел новый сорт зерновой культуры и снял с опытной делянки *k* кг семян. Посеяв *1* кг семян, можно за сезон собрать *p* кг семян. Через сколько лет селекционер сможет засеять новой культурой поле площадью *s* га, если норма высева *n* кг/га?
8. Найти площадь равнобедренной трапеции с основаниями *a* и *b* и углом α при большем основании *a*.
9. Треугольник задается координатами своих вершин на плоскости: $A\ (x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Найти длину и основание высоты, опущенной из вершины *A* на сторону *BC*.
10. За первый год производительность труда на предприятии возросла на p_1 %, за второй и третий — соответственно на *p2* и *p3* %. Найти среднегодовой прирост производительности (в процентах).
11. Найти сумму членов арифметической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.
12. Заданы координаты точки подвески математического маятника $A(x, y, z)$ и координаты одной из точек его наивысшего подъема $B(x_1, y_1, z_1)$. Найти координаты самой низкой точки траектории и другой наивысшей точки подъема.
13. Найти (в радианах и в градусах) все углы треугольника со сторонами *a*, *b*, *c*.
14. Русские неметрические единицы длины: 1 верста = 500 саженей; 1 сажень = 3 аршина; 1 аршин = 16 вершков; 1 вершок = 44,45 мм. Длина некоторого отрезка составляет *p* метров. Перевести ее в русскую неметрическую систему.

15. Составить программу перевода радианной меры угла в градусы, минуты и секунды.

Обработка результатов измерений:

Заданы вершины треугольника *A(x1,y1), B(x2,y2), C(x3,y3)*. Вычислить длину медианы, проведенной из *A*.

```
//Подключение средств консольного потокового
//ввода/вывода
#include<iostream.h>
//Подключение математических функций
#include<math.h>
//Подключение библиотеки для использования функции
//getch()
#include<conio.h>
//главная функция
main()
{
//Объявление переменных для задания координат
//точек треугольника
double Ax,Ay,Bx,By,Cx,Cy;
//Ввод координат точки A
cout<<"Введите координаты точки A:"<<endl;
cout<<"x=";
cin>>Ax;
cout<<"y=";
cin>>Ay;
//Ввод координат точки B
cout<<"Введите координаты точки B:"<<endl;
cout<<"x=";
cin>>Bx;
cout<<"y=";
cin>>By;
//Ввод координат точки C
cout<<"Введите координаты точки C:"<<endl;
cout<<"x=";
cin>>Cx;
cout<<"y=";
cin>>Cy;
//Объявление переменных, в которых будут
//находится длины сторон
double ab,bc,ca;
//Вычисление стороны AB
ab=sqrt((Ax-Bx)*(Ax-Bx)+(Ay-By)*(Ay-By));
//Вычисление стороны BC
bc=sqrt((Bx-Cx)*(Bx-Cx)+(By-Cy)*(By-Cy));
//Вычисление стороны CA
ca=sqrt((Cx-Ax)*(Cx-Ax)+(Cy-Ay)*(Cy-Ay));
//Объявление переменной для медианы
// и вычисление ее через длины
//сторон треугольника
double M=sqrt(2*ab*ab+2*ca*ca-bc*bc)/2;
//Вывод значения длины медианы
cout<<"Длина медианы, из точки A равна:”
<<M<<endl;
cout<<"Для завершения нажмите любую клавишу";
```

```
getch();
}
```

Примечание: Чтобы сразу после окончания работы программы окно, в котором программа работала, не было автоматически перекрыто другим окном, например окном редактора текста среды разработки, в конце программы желательно использовать функцию getch(), которая приостанавливает выполнение программы до тех пор, пока не будет нажата любая клавиша.

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Назвать этапы работы с программой на языке Си/Си++ в системе программирования.
2. Перечислить основные синтаксические правила и записи программ на языке Си/Си++.
3. Перечислить правила арифметических операций.
4. Установить последовательность действий ввода и вывода консольных текстовых сообщений.

Лабораторная работа № 2

Создание и администрирования WEB – сервера

Цель работы:

Выработать практические навыки работы с системой программирования языка Си++, познакомиться с диагностическими сообщениями компилятора об ошибках при выполнении программ, реализующих алгоритмическую структуру “ветвление” (операторы if, switch).

Порядок выполнения:

Операции отношения и эквивалентности

Операции отношения и эквивалентности используются при сравнении двух операндов. Они возвращают **true** – истина, если указанное соотношение операндов выполняется, и **false** – ложь, если соотношение не выполняется. Определены следующие операции отношения:

Таблица 8

Основные операции отношения

Обозначение	*Операция*	*Пример*
==	Равно	A==B
!=	Не равно	X!=Y
<	Меньше чем	X<Y
>	Больше чем	C>0
<=	Меньше или равно	A<=I
>=	Больше или равно	X>=Y

Операнды должны иметь совместимые типы, за исключением целых и действительных типов, которые могут сравниваться друг с другом.

Логические операции

Логические операции принимают в качестве операндов выражения скалярных типов и возвращают результат булева типа: true или false. В C++ любое выражение, имеющее некоторое значение, может использоваться в логических операциях. Так если значение выражения 0, то оно трактуется как false, любое другое значение трактуется как true.

Таблица 9

Основные логические операции

Обозначение	*Операция*	*Пример*
!	Отрицание	!A
&&	Логическое И	A&&B
||	Логическое ИЛИ	A||B

Унарная операция логического отрицания (!) возвращает true, если операнд возвращает ненулевое значение. Таким образом, выражение !A эквивалентно выражению A==0.

Операция логического И (&&) возвращает true, если оба ее операнда возвращают ненулевые значения. Если хотя бы один операнд возвращает 0 (false), то операция И также возвращает false.

Операция логического ИЛИ (||) возвращает true, если хотя бы один ее операнд возвращает ненулевое значение. Если оба операнда возвращают 0 (false), то операция ИЛИ также возвращает false.

Поразрядные логические операции

Поразрядные логические операции работают с целыми числами и оперируют с их двоичными разрядами операндов.

Таблица 10

Поразрядные логические операции

Обозначение	*Операция*	*Пример*
~	поразрядное отрицание	~X
&	поразрядное И	X&&Y
|	поразрядное ИЛИ	X|Y
^	поразрядное исключающее ИЛИ	X^Y
<<	поразрядный сдвиг влево	X<<2
>>	поразрядный сдвиг вправо	Y>>1

Операция поразрядного отрицания (~) инвертирует каждый бит операнда. Поразрядные операции &, | и ^ работают в соответствии с таблицей 11, где E1 и E2 – сравниваемые биты операндов.

Таблица 11

E1	E2	E1 & E2	E1 ^ E2	E1 | E2
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	0	1

Операция поразрядного сдвига вправо (>>) сдвигает биты левого операнда на число разрядов, указанное правым операндом. При этом правые биты теряются. Если левый операнд представляет собой целое без знака, то левые освободившиеся биты заполняются нулями. В противном случае они заполняются символом знака. Сдвиг целого числа на *n* разрядов вправо эквивалентен целочисленному делению его на 2^n.

Операция поразрядного сдвига влево (<<) сдвигает биты левого операнда на число разрядов, указанное правым операндом. При этом левые биты теряются, а правые заполняются нулями. Сдвиг целого числа на *n* разрядов влево эквивалентен умножению его на 2^n.

Условный оператор выбора

Оператор **if** предназначен для выполнения тех или иных действий в зависимости от истинности или ложности некоторого условия. Условие задается выражением, имеющим результат булева типа.

Оператор имеет две формы:

1) **if(условие)оператор;**

Скобки, обрамляющие условие, обязательны.

Условием может быть выражение, преобразуемое в булев тип. Если условие истинно, то указанный в конструкции оператор выполняется. В противном случае управление сразу передается следующему за конструкцией if оператору.

2) **if (условие) оператор1;**
else оператор2;

Если условие возвращает true, то выполняется первый из указанных операторов, в противном случае выполняется второй оператор. Обратите внимание, что в конце первого оператора перед ключевым словом else ставится точка с запятой.

При вложенных конструкциях if могут возникнуть неоднозначности в понимании того, к какой из вложенных конструкций if относится элемент else. Компилятор всегда считает, что else относится к последней из конструкций if, в которой не было раздела else. Например,

```
if(условие1)
  if(условие2)
    оператор1;
  else оператор2;
```

else будет отнесено компилятором ко второй конструкции if, т.е. оператор2 будет выполняться в случае, если первое условие истинно, а второе ложно.

Если же вы хотите отнести else к первому if, это надо записать в явном виде с помощью фигурных скобок:

```
if(условие1)
 {
   if(условие2) оператор1;
 }
   else оператор2;
```

В конструкциях в качестве оператор, оператор1 и оператор2 понимается использование одного оператора или выражения. Если необходимо выполнение нескольких операторов, то следует использовать составной оператор.

Поскольку в Си++ любое арифметическое значение может преобразовываться к булеву типу, т.е. если значение выражения 0, то оно трактуется как false, а любое другое значение трактуется как true. То в условии можно использовать практически любые арифметические выражения. Например:

```
int a,b,c;
…
if(a–b/c) …;
```

В данном случае условие if(a–b/c) будет false, если результат выражения a-b/c буде нуль, и условие будет true при всех остальных результатах выражения.

Так же, следует предостеречь от довольно распространенной ошибки: случайного применения вместо операции эквивалентности (==) операции присваивания (=). Например, если по ошибке вместо оператора

```
if (A==2) …;
```

используется оператор

```
if (A=2) …;
```

то это не будет расценено как синтаксическая ошибка. Результат операции A=2 будет трактоваться как true независимо от того, чему было равно значение переменной A до выполнения этого ошибочного оператора. К тому же выполнение этого оператора приведет к изменению переменной.

Условный оператор множественного выбора

Оператор **switch** позволяет провести анализ значения некоторого выражения и в зависимости от его значения выполнить те или иные действия. В общем случае формат записи опе-

ратора является следующим:

```
switch (выражение_выбора){
  case значение_1 : оператор_1;
            break; // не обязательно
  …
  case значение_n : оператор_n;
            break; // не обязательно
  default : оператор;     // не обязательно
              }
```

В этой конструкции выражение выбора должно иметь порядковый тип – целый, перечислимый и т.д. Поэтому, например, нельзя использовать выражения, возвращающие действительные числа или строки.

Значения, указываемые в метках **case**, должны быть константными выражениями, соответствующими возможным значениям выражения выбора. После значения ставится двоеточие <:>, а затем пишется оператор (может писаться составной оператор), который должен выполняться, если выражение приняло указанное в метке значение.

Если значение выражения выбора совпало со значением, указанным в одной из меток case, то выполняется оператор, записанный после этой метки, после чего, если не принять соответствующих мер, будут выполняться все последующие операторы остальных меток. Поскольку это обычно нежелательно, то, как правило, после оператора, который должен выполняться, записывают оператор break, который прерывает выполнение структуры switch и управление передается следующему за ней оператору.

Если значение выражения выбора не соответствует ни одному из перечисленных в метках, то выполняется оператор, следующий за меткой default. Впрочем, метка default является не обязательной.

Значения в метках могут содержать константы и константные выражения, которые совместимы по типу с объявленным выражением и которые компилятор может вычислить заранее, до выполнения программы. Недопустимо использование переменных и функций. В метках не допускается повторение одних и тех же значений, поскольку в этом случае выбор был бы неоднозначным.

Приведенный ниже пример анализирует переменную Key типа char, содержащую символ, введенный пользователем.

```
switch (Key)   {
case ‘y’: case ‘Y’:
  cout<< “Вы нажали клавишу Y или y”; break;
case ‘n’: case ‘N’:
  cout<< “Вы нажали клавишу N или n”; break;
default:
  cout<<“Вы не нажали клавишу Y/y и N/n”;
        }
```

При необходимости выполнять одинаковые действия при нескольких значениях выражения выбора, надо размещать подряд несколько меток case**.**

Оператор передачи управления

Оператор **goto** позволяет прервать обычный поток управления и передать управление в произвольную точку кода, помеченную специальной меткой и имеет следующую форму:

goto метка;

Метка в тексте программы обозначается идентификатором с последующим двоеточием. Например

Lbegin:

Метка отмечает точку, в которую передается управление оператором goto. Метка может располагаться в любом месте блока, как после оператора goto передающего на нее управление, так и до этого оператора. При этом нужно иметь в виду, что передача управления извне или внутрь цикла может приводить к непредсказуемым последствиям.

Метки можно использовать внутри всей функции, в которой они указаны, но на них

нельзя ссылаться вне тела функции.

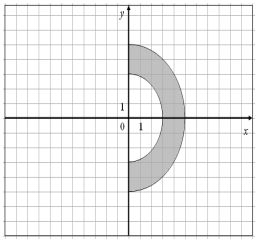
Следует помнить, что чрезмерно широкое применение оператора goto делает структуру программы крайне запутанной и плохо читаемой, что затрудняет ее дальнейшее ее сопровождение.

Варианты задания для выполнения лабораторной работы:

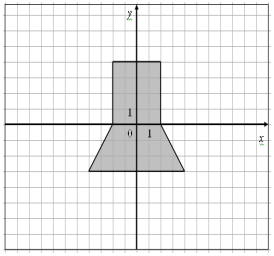
ЗАДАНИЕ I:

Для данных областей составить программу, которая определяет принадлежность точки с координатами (x,y) закрашенной области:

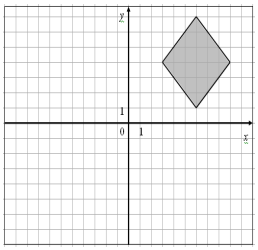
1.



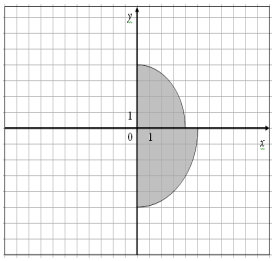
2.



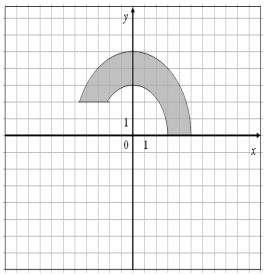
3.



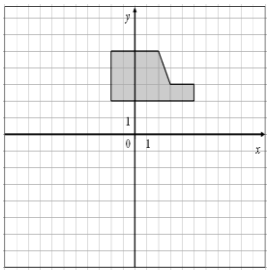
4.



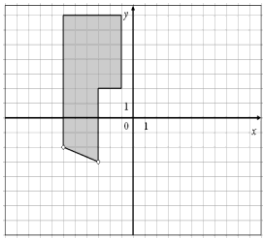
5.



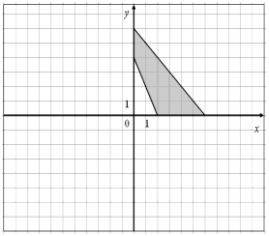
6.



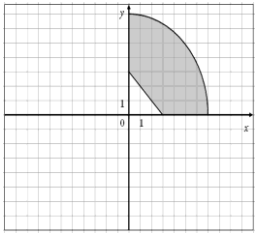
7.



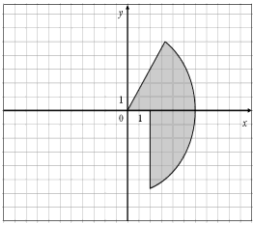
8.



9.

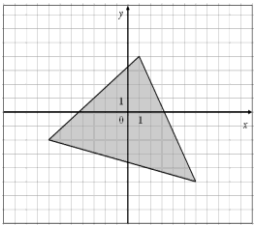


10.

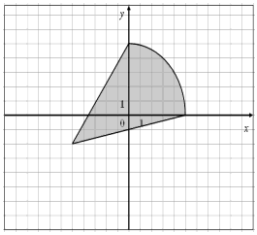


11.

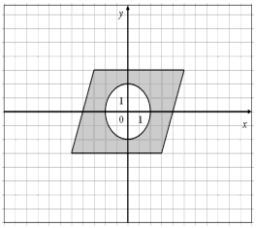
12.



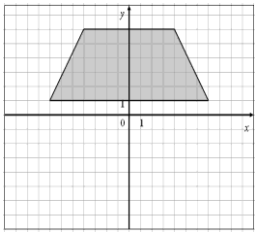
13.



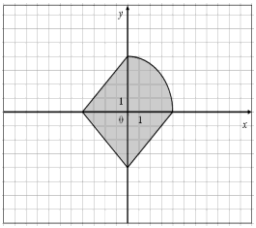
14.



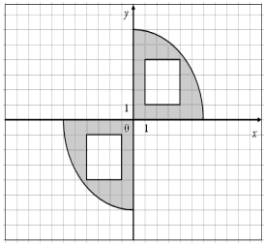
15.



16.



17.



18.

ЗАДАНИЕ 2:

Для всех вариантов заданий, при организации ветвления в программе, следует использовать конструкцию множественного выбора *switch … case.*

1. Написать программу расчета площади фигуры по названию (треугольник, квадрат, прямоугольник, ромб, трапеция и окружность). Для вычисления площади, программа должна производить запрос необходимых параметров исходя из выбранной фигуры.
2. Написать программу расчета периметра фигуры по названию (треугольник, квадрат, прямоугольник, ромб, трапеция и окружность). Для вычисления периметра, программа должна производить запрос необходимых параметров исходя из выбранной фигуры.
3. Написать программу расчета объема фигуры по названию (шар, конус, усеченный конус, куб, параллелепипед и цилиндр). Для вычисления объема, программа должна производить запрос необходимых параметров исходя из выбранной фигуры.
4. Написать программу, которая по введенной длине отрезка и номеру единицы измерения (1 — дециметр, 2 — километр, 3 —метр, 4 — миллиметр, 5 — сантиметр) выводит соответствующее значение длины отрезка в метрах.
5. Написать программу, которая по введенному числу и номеру месяца выводит информацию, является ли этот день праздничным и название праздника.
6. Составить программу случайного выбора места летнего отдыха из семи предлагаемых туристическим агентством курортов, причем с вероятностью 3/10 придется отдыхать на даче.
7. Даны два вещественных положительных числа x и y. Арифметические действия над числами пронумерованы (сложение — 1, вычитание — 2, умножение — 3, деление — 4). Составить программу, которая по введенному номеру выполняет то или иное действие над числами.
8. Написать программу, которая по введенной массе и номеру единицы измерения (миллиграмм — 1, грамм — 2, килаграмм — 3, центнер — 4, тонна —5) выводит соответствующее значение массы в килограммах.
9. Пусть элементами треугольника являются: сторона a (первый элемент), площадь *S* (второй элемент), высота *h* (третий элемент), радиус вписанной окружности *r* (четвертый элемент), радиус описанной окружности *R* (пятый элемент). Составить программу, которая

по введенному номеру и значению соответствующего элемента вычисляет значения всех остальных элементов треугольника. (Углы треугольника считаются известными).

10. Составить программу случайного выбора дежурного из списка, в котором 4 девушки и 4 молодых человека, причем для девушек вероятность выбора в два раза ниже.

11. Пусть элементами параллелограмма являются: сторона a (первый элемент), площадь *S* (второй элемент), высота *h* (третий элемент), диагональ *d* (четвертый элемент). Составить программу, которая по введенному номеру и значению соответствующего элемента вычисляет значения всех остальных элементов треугольника. (Углы параллелограмма считаются известными).

12. Написать программу, которая запрашивает натуральной число (от 0 до 99) в десятичном представлении и выводит его название на естественном языке. Например: 7 семь, 52 пятьдесят два

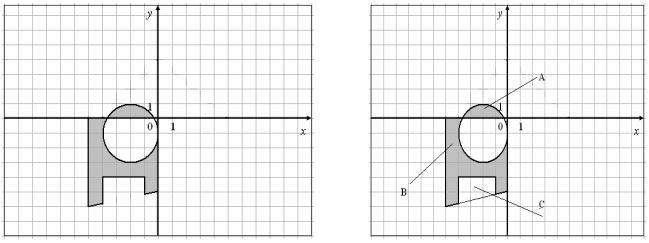
13. Вычислить порядковый номер дня по заданному числу, месяцу и году.

14. Написать программу, которая по введенному числу и месяцу выдает в качестве результата расписание занятий в этот день (год считать текущим).

15. Пусть элементами круга являются радиус (первый элемент), диаметр (второй элемент) и длина окружности (третий элемент). Составить программу, которая по номеру элемента запрашивала бы его соответствующее значение и вычисляла бы площадь круга.

Обработка результатов измерений:

Пример 1. Для данной области (рис. 5.а) составить программу, которая определяет принадлежность точки с координатами (x,y) закрашенной области:



a) b)

Рис. 5

Разобьем закрашенную область на три фигуры (рис. 5.б) и для каждой фигуры приведем ограничения в виде неравенств.

Фигура A ограничена осью абсцисс (y=0) и окружностью с центром в точке (-2;-1) и радиусом R=2. Чтобы точка принадлежала фигуре A нужно выполнение следующих условий: $y > 0$ И $(x+2)^2 + (y+1)^2 < 4$.

Фигура B ограничена осью абсцисс (y=0), осью ординат (x=0), прямой, проходящей через точки (-5;-6) и (0;-5), а также окружностью с центром в точке (-2;-1) и радиусом R=2. Чтобы точка принадлежала фигуре B нужно выполнение следующих условий: $y < 0$ И $x < 0$ И $y > 0.2x - 5$ И $(x+2)^2 + (y+1)^2 > 4$.

Фигура C ограничена прямыми, параллельными осям координат (y=-4), (x=-5), (x=-1), а также прямой, проходящей через точки (-5;-6) и (0;-5). Чтобы точка принадлежала фигуре C нужно выполнение следующих условий: $y < -4$ И $x > -4$ И $x < -1$ И $y > 0.2x - 5$.

Для того, чтобы точка принадлежала закрашенной области, необходимо выполнение одного из двух условий:

1) чтобы точка попала в область фигуры A;
2) чтобы точка попала в область фигуры B и одновременно не попала в область фигуры C.

В общем случае условие принадлежности закрашенной области выглядит следующим:

A ИЛИ (B И (НЕ C))

Листинг программы, выполняющий данное задание:

```
#include<iostream.h>
#include<conio.h>
```

```
main()
{
double x,y;
cout <<"Введите координаты точки:"<<endl;
cout <<"x=";
cin >> x;
cout <<"y=";
cin >> y;
int A,B,C;
//проверка условий попадания в область фигуры A
A=(y>0)&&((x+2)*(x+2)+(y+1)*(y+1)<4);
//проверка условий попадания в область фигуры B
B=(y<0)&&(x<0)&&(y>0.2*x-5)
  &&((x+2)*(x+2)+(y+1)*(y+1)>4);
//проверка условий попадания в область фигуры C
C=(y<-4)&&(x>-4)&&(x<-1);
//проверка условия попадания в закрашенную область
//если A ИЛИ (B И (НЕ C))
if (A||(B&&(!C)))
  cout << "Точка принадлежит заданной области";
 else
  cout << "Точка не принадлежит заданной области";
getch();
}
```

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Что такое логические операции? Основные виды.
2. Формы условного оператора выбора.
3. Оператор передачи управления.
4. Привести примеры, реализующие алгоритмы разветвляющейся структуры.

Лабораторная работа № 3

Разработка WEB – приложения с использованием JavaScript

Цель работы:

Закрепить практические навыки работы с системой программирования языка Си++, познакомиться с диагностическими сообщениями компилятора об ошибках при выполнении программ, содержащих операторы цикла while, do ... while и for.

Порядок выполнения:

Оператор цикла с постусловием

Структура **do…while** используется для организации циклического выполнения оператора или совокупности операторов, называемых телом цикла, до тех пор, пока не окажется нарушенным некоторое условие. Синтаксис данного оператора является следующим:

do оператор while (условие);

Структура работает следующим образом: Выполняется оператор тела цикла. Затем вычисляется условие – выражение, которое должно возвращать результат булева типа. Если выражение возвращает true (не нулевое значение), то повторяется выполнение тела цикла и после этого снова вычисляется условие – выражение. Такое циклическое повторение цикла продолжается до тех пор, пока проверяемое условие – выражение не возвратит false (нуль). После этого цикл завершается и управление передается оператору, следующему за структурой do…while.

Поскольку проверка выражения осуществляется после выполнения тела цикла, то цикл будет заведомо выполнен хотя бы один раз, даже если выражение сразу ложно.

В конструкции в качестве оператор понимается использование одного оператора или выражения. Если необходимо выполнение нескольких операторов, то следует использовать составной оператор.

Ниже приведен пример, в котором осуществляется проверка вводимых значений пользователем с клавиатуры. Ввод значений будет продолжаться до тех пор, пока пользователь не введет нужные значения:

```
#include<iostream.h>
main()
{
double A,B;
do {
cout <<"Введите значения больше нуля:"<<endl;
cout <<"A=";
cin >>A;
cout <<"B=";
cin >>B;
if ((A<=0)||(B<=0))
      cout <<"Неправильные значения"<<endl;
   } while((A<=0)||(B<=0));
}
```

Оператор цикла с предусловием

Оператор **while** используется для организации циклического выполнения тела цикла, пока выполняется некоторое условие. Синтаксис структуры:

while (условие) оператор;

Структура работает следующим образом: Сначала вычисляется условие, которое должно возвращать результат булева типа. Если выражение возвращает true (ненулевое значение), то выполняется оператор тела цикла, после чего опять вычисляется выражение, определяющее условие. Такое циклическое повторение выполнения оператора и проверки условия продолжается до тех пор, пока условие не вернет false (нуль). После этого цикл завершается и управление передается оператору, следующему за структурой while.

Если необходимо выполнение нескольких операторов, то следует использовать составной оператор.

Поскольку проверка выражения осуществляется перед выполнением оператора тела цикла, то, если условие сразу ложно, оператор не будет выполнен ни одного раза.

Ниже приведен пример, в котором осуществляется вывод натуральных чисел, не превышающих заданного значения.

```
#include<iostream.h>
main()
{
int N,k=0;
cout << "N=";
cin >>N;
while (k<=N){ cout<<k<<", ";
          k++;
         }
}
```

Универсальный оператор цикла

Оператор **for** обеспечивает циклическое повторение некоторого оператора (в частности, составного оператора) заданное число раз. Повторяемый оператор называется телом цикла. Повторение цикла обычно определяется некоторой управляющей переменной, которая изменяется при каждом выполнении тела цикла. Повторение завершается, когда управляющая переменная достигает заданного значения.

Синтаксис структуры for:

for (выражение1; выражение2; выражение3) оператор;

где выражение1 задает начальное значение переменной, управляющей циклом, выражение 2 является условием продолжения цикла, а выражение3 изменяет управляющую переменную.

Структура for работает следующим образом: Сначала выполняется выражение1 (оно может состоять и из ряда выражений, разделенных запятой т.е. может использоваться операция последования). Это выражение задает начальные значения переменной (или переменных) цикла. Затем проверяется выражение2 – условие продолжения цикла. Если условие истинно (возвращает true – ненулевое значение), то выполняется тело цикла – оператор, записанный в структуре for. После завершения тела цикла выполняется выражение3, определяющее обычно изменение переменной цикла. Затем опять проверяется условие, записанное как выражение2, и при истинности этого условия выполнение цикла продолжается. Как только в какомнибудь цикле выражение2 вернет false (нулевое значение), цикл прерывается и управление передается оператору, расположенному следом за структурой for.

Приведем примеры использования цикла for.

Найти сумму ряда $\sum_{i=0}^{N-1}\frac{1}{i+1}$.

```
#include<iostream.h>
main()
{
int N;
cout <<"N=";
cin >>N;
double S=0;
for(int i=0;i<N; i++) S+=1/double(i+1);
cout <<"S="<<S;
}
```

Здесь первое выражение в структуре for вводит целую переменную i, являющуюся счетчиком циклов, и инициализирует ее значением 0. Второе выражение проверяет условие завершения цикла. В данном случае цикл должен завершиться, когда переменная i, используемая в теле, примет значение, равное значению переменной N, введенное пользователем с клавиатуры. Третье выражение структуры for увеличивает после каждого выполнения цикла значение i на 1 с помощью операции инкремента.

Теперь рассмотрим этот же приме пример, но с использованием в структуре for операции запятая. Если объявить переменные i и S до начала цикла, собственно цикл можно весь разместить в заголовке структуры for:

```
#include<iostream.h>
```

```
main()
{
int N,i;
cout << "N=";
cin >>N;
double S;
for(i=0,S=0; i<N; S+=1/double(i+1),i++);
cout <<"S="<<S;
}
```

В этом примере первое выражение структуры for включает в себя два оператора, разделенных операцией запятая и задающих начальные значения переменной S, накапливающей сумму, и переменной цикла i. Третье выражение структуры for включает также два оператора - формирование суммы и постфиксный инкремент переменной цикла i. После структуры for стоит точка с запятой, что означает пустое тело цикла.

В приведенных примерах переменная цикла увеличивалась на единицу при каждом цикле. Можно организовывать циклы с изменением переменной на любое значение.

Выражения в структуре for являются необязательными. Иногда может отсутствовать первое выражение, если начальное значение управляющей переменной задано где-то в другом месте программы. Если отсутствует второе выражение, предполагается, что условие продолжения цикла всегда истинно и таким образом создается бесконечно повторяющийся цикл. Выйти из такого цикла можно, проверив в теле цикла какие-то условия и прервав выполнение передачей управления за пределы цикла оператором goto или применить другие способы прерывания. Может отсутствовать в структуре for и третье выражение, если приращение переменной осуществляется операторами в теле структуры или если приращение не требуется.

При пропуске какого-то из выражений, точка с запятой после пропущенного выражения (кроме третьего) должна писаться. Например, в заголовке

```
for (; i<10;)   …;
```

пропущено первое условие и третье.

Если условие продолжения цикла не удовлетворяется с самого начала, то операторы тела структуры for не выполняются ни разу.

Операторы цикла for могут быть вложенными.

Операторы прерывания цикла

В некоторых случаях желательно прервать повторение цикла, проанализировав какие-то условия внутри него. Это может потребоваться в тех случаях, когда проверки условия окончания цикла громоздкие, требуют многоэтапного сравнения и сопоставления каких-то данных и все эти проверки просто невозможно разместить в выражении условия операторов for, do…while или while.

Один из возможных вариантов решения этой задачи – использование оператора break. Оператор break прерывает выполнение тела любого цикла for, do или while и передает управление следующему за циклом выполняемому оператору.

Например, пусть в цикле осуществляется ввод с клавиатуры, который будет продолжаться до тех пор, пока пользователь не введет нужные значения:

```
#include<iostream.h>
main()
{
double A,B;
while(1){
cout <<"Введите значения больше нуля:"<<endl;
cout <<"A=";
cin >>A;
cout <<"B=";
cin >>B;
if ((A<=0)||(B<=0)) cout <<"Неправильные значения"<<endl;
```

```
    else break; // Прерывание цикла
     }
}
```

Еще один способ прерывания цикла – использование оператора goto, передающего управление какому-то оператору, расположенному вне тела цикла.

Описанные способы прерывали выполнение цикла. Имеется еще процедура continue, которая прерывает только выполнение текущей итерации, текущего выполнения тела цикла и передает управление на следующую итерацию.

Чтобы продемонстрировать применение continue, рассмотрим пример нахождения произведения $\prod_{\substack{i=1 \\ i\neq 5}}^{N} \frac{1}{i-5}$, которое можно организовать следующим образом:

```
#include<iostream.h>
main()
{
int N;
cout <<"N=";
cin >> N;
double P=1;
for (int i=1; i<N; i++)
  if (i==5) continue; // Прерывание текущей итерации цикла
     else P*=1/double(i-5);
cout <<"P="<<P;
}
```

В этом варианте при i равном 5 текущая итерация прерывается и произведение не вычисляется, но цикл не прекращается.

Методы численного интегрирования

Если функция $f(x)$ непрерывна на отрезке $[a,b]$ и известна ее первообразная функция $F(x)$, то определенный интеграл этой функции в пределах от *a* до *b* может быть вычислен по формуле Ньютона – Лейбница

$$\int_a^b f(x) = F(b) - F(a)$$

.

Однако во многих случаях первообразная функция $F(x)$ не может быть найдена или является слишком сложной. Вследствие этого вычисления определенного интеграла по формуле Ньютона – Лейбница может быть затруднительным или практически невозможным. Кроме того, на практике подынтегральная функция $f(x)$ часто задается таблично и тогда само понятие первообразной теряет смысл. Поэтому, важное значение имеют приближенные и, в первую очередь, численные методы вычисления определенных интегралов.

Численные методы позволяют избежать громоздких расчетов в тех случаях, когда результат приемлем с определенной степенью точности. Точность вычислений ε (максимально допустимая ошибка в результате всех вычислений, из которых складывается погрешность), как правило, оговаривается заранее при постановке задачи. Точность вычислений будет достигнута, если абсолютная погрешность (абсолютное значение разности между точным и приближенным значениями) не превосходит ε (очень маленькое положительное число).

Сущность большинства численных методов вычисления определенных интегралов состоит в замене подынтегральной функции $f(x)$ аппроксимирующей функцией $\varphi(x)$, для которой легко можно отыскать первообразную среди элементарных функций:

$$\int_a^b f(x) = \int_a^b \varphi(x) + R = S + R$$

,

где *S* - приближенное значение интеграла,

R - погрешность вычисления интеграла.

Следует отметить тот факт, что погрешность уменьшается при увеличении числа *n* точек

разбиения отрезка $[a,b]$ за счет более точной аппроксимации подынтегральной функции, однако при этом будет возрастать погрешность суммирования частичных интегралов, и, начиная с некоторого значения n_0 она станет достаточно большой. Поэтому возникает необходимость разработки оценки погрешности каждого из методов интегрирования.

Формулы приближенного интегрирования (квадратурные формулы) можно найти в справочниках по высшей математике. Приведем основные формулы численного интегрирования для методов левых прямоугольников, трапеций и Симпсона.

Метод левых прямоугольников. Пусть *I* - точное значение интеграла $\int_a^b f(x)$, I_n - приближенное значение интеграла, полученное при разбиении отрезка интегрирования на *n* равных частей, I_{2n} - приближенное значение интеграла при удвоенном числе точек разбиения, ε - заданная точность вычислений. Тогда для метода левых прямоугольников квадратурная формула будет иметь вид

$$I_n = h\sum_{i=0}^{n} f(x_i),$$

где $h = \frac{b-a}{n}$, $x_i = a + h \cdot i$.

Оценка точности вычислений:

$$|I_n - I_{2n}| < \varepsilon.$$

Следовательно, как только выполнится условие $|I_n - I_{2n}| < \varepsilon$, это будет означать, что I_{2n} является приближенным значением для *I* с точностью ε.

Метод трапеций. Для метода трапеций квадратурная формула будет иметь вид

$$I_n = h\left(\sum_{i=1}^{n-1} f(x_i) + \left(f(a) + f(b)\right)/2\right),$$

где $h = \frac{b-a}{n}$, $x_i = a + h \cdot i$.

Оценка точности вычислений:

$$\frac{|I_n - I_{2n}|}{3} < \varepsilon.$$

Следовательно, как только выполнится условие $\frac{|I_n - I_{2n}|}{3} < \varepsilon$, это будет означать, что I_{2n} является приближенным значением для *I* с точностью ε.

Метод Симпсона. Для метода Симпсона квадратурная формула будет иметь вид

$$I_n = \left(f(a) + f(b) + 4\sum_{i=1}^{n} f(x_{2i-1}) + 2\sum_{i=2}^{n} f(x_{2i-2})\right) \cdot \frac{h}{3},$$

где $h = \frac{b-a}{2n}$, $x_i = a + h \cdot i$.

Оценка точности вычислений:

$$\frac{|I_n - I_{2n}|}{15} < \varepsilon.$$

Следовательно, как только выполнится условие $\frac{|I_n - I_{2n}|}{15} < \varepsilon$, это будет означать, что I_{2n} является приближенным значением для *I* с точностью ε.

При программной реализации методов численного интегрирования, как правило, задаются оценкой точности вычислений ε и первоначальным количеством отрезков интегрирования *n*. Определяют приближенные значения интегралов по квадратурным формулам для *n* и *2n* отрезков и вычисляют оценку точности по формулам, исходя из конкретного метода интегрирования. Если вычисленная точность меньше заданной, то считается, что значение интеграла, полученное для *2n* отрезков, является приближенным значением с точностью ε. Ес-

ли вычисленная точность больше заданной, то увеличивают количество отрезков интегрирования *n* в два раза и повторяют процедуру вычислений до тех пор, пока вычисленная точность будет меньше заданной.

Варианты задания для выполнения лабораторной работы:

ЗАДАНИЕ I (для всех вариантов задания, при организации циклического повторения, следует использовать оператор цикла с постусловием do … while):

1 – 14. Дан числовой ряд и некоторое число *ε*. Найти сумму тех членов ряда (*n=1,2,3…*), модуль которых больше или равен заданному числу ε и вывести на экран все элементы числового ряда, которые входят в сумму. Общий член ряда имеет вид:

1. $a_n = \frac{(-1)^{n-1}}{n^n}$;
2. $a_n = \frac{2n-1}{2^n}$;
3. $a_n = \frac{10^n}{n!}$;
4. $a_n = \frac{n!}{n^n}$;
5. $a_n = \frac{3^n \cdot n!}{(2n)!}$;
6. $a_n = \frac{n!}{(2^n)!}$;
7. $a_n = \frac{1}{2^n} + \frac{1}{3^n}$;
8. $a_n = \frac{1}{(3n-2)(3n+1)}$;
9. $a_n = \frac{n!}{(2n-1)!}$;
10. $a_n = \frac{2^n \cdot (n+1)!}{n^n}$;
11. $a_n = \frac{n!}{3n^n}$;
12. $a_n = \frac{2^n}{(n-1)!}$;
13. $a_n = \frac{3^n}{2^n n!}$;
14. $a_n = \frac{1}{n} + \frac{1}{n+1} + \frac{1}{n+2}$;

15 – 28. Дана числовая последовательность и некоторое число *ε*. Найти наименьший номер члена последовательности, для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$. Вывести на экран этот номер и все элементы a_i, где *i=1,2,…,n*.

15. $a_n = arctg(a_{n-1}) + 1, \ a_1 = 0.$
16. $a_n = 2 + \frac{1}{a_{n-1}}, \ a_1 = 2.$
17. $a_n = \frac{1}{2} tg(a_{n-1}), \ a_1 = 0{,}5.$
18. $a_n = \frac{1}{(2n)^2}$.
19. $a_n = \frac{1}{2}\cos(a_{n-1}), \ a_1 = 0{,}5.$
20. $a_n = \frac{2 + a_{n-1}^2}{2a_{n-1}}, \ a_1 = 2.$
21. $a_n = \frac{a_{n-1} + a_{n-2}}{2}, \ a_1 = 1, \ a_2 = 2.$
22. $a_n = \frac{n^{\ln(n)}}{(\ln(n))^n}$.
23. $a_n = e^{-a_n - 1}, \ a_1 = 0.$
24. $a_n = \frac{1}{2a_{n-1}^2}, \ a_1 = 1.$
25. $a_n = \frac{1}{n} + \frac{1}{n+1} + \frac{1}{n+2}$.
26. $a_n = \frac{1}{2a_{n-1}} + \frac{1}{4a_{n-2}}, \ a_1 = 1, \ a_2 = 2.$
27. $a_n = \frac{1}{2a_{n-1}^2} + \frac{1}{3a_{n-2}^3}, \ a_1 = 1, \ a_2 = 0{,}5.$
28. $a_n = \frac{(2n-1)(3n+1)}{(4n+1)(5n-1)}$.

ЗАДАНИЕ II:

Вычислить значения определенных интегралов $\int_a^b F(x)dx$ методом прямоугольников, методом трапеций и методом Симпсона с заданной точностью ε. Вычисления организовать в одном цикле таким образом, что при достижении требуемой точности вычисления интеграла одним методом, продолжать вычисления интеграла другими методами. (Для всех вариантов задания, при организации циклического повторения, следует использовать оператор цикла с предусловием while):

1. $F(x)=3\sin\sqrt{x}+0.35x-3.8$.
2. $F(x)=\ell^x+\sqrt{1+\ell^{2x}}-2$.
3. $F(x)=\sqrt{1-x}-\cos\sqrt{1-x}$.
4. $F(x)=0.4+arctg\sqrt{x}-x$.
5. $F(x)=tg\frac{x}{2}-ctg\frac{x}{2}+x$.
6. $F(x)=0.6\cdot 3^x-2.3x-3$.
7. $F(x)=\cos\frac{2}{x}-2\sin\frac{1}{x}+\frac{1}{x}$.
8. $F(x)=\sqrt{1-0.4x^2}-\arcsin x$.
9. $F(x)=0.1x^2-x\ln x$.
10. $F(x)=tgx-\frac{1}{3}tg^3x+\frac{1}{5}tg^5x$.
11. $F(x)=\arccos x-\sqrt{1-0.3x^3}$.
12. $F(x)=3x-4\ln x-5$.
13. $F(x)=\ell^x-\ell^{-x}-2$.
14. $F(x)=\sin(\ln x)-\cos(\ln x)+2\ln x$.
15. $F(x)=x-2+\sin\frac{1}{x}$.
16. $F(x)=\ell^x+\ln x-10x$.
17. $F(x)=xtgx-\frac{1}{3}$.
18. $F(x)=3\ln^2 x+6\ln x-5$.
19. $F(x)=x^2-\ln(1+x)-3$.
20. $F(x)=2x\sin x-\cos x$.
21. $F(x)=\ln x-x+1.8.$
22. $F(x)=\cos x-\ell^{\frac{x^2}{2}}+x-1$.
23. $F(x)=1-x+\sin x-\ln(1+x)$.
24. $F(x)=3x-14+\ell^x-\ell^{-x}$.
25. $F(x)=x+\cos(2+x^2)$.
26. $F(x)=x^2-\ln(5+x)$.
27. $F(x)=0.5x+\sin(1+x)$.
28. $F(x)=\arcsin(2x+1)-x^2$.
29. $F(x)=x-\sqrt{9+x}+x^2-4$.
30. $F(x)=x-\sin x-0.5$.
31. $F(x)=-x+x^2-1$.
32. $F(x)=\ell^x-x^2+3$.
33. $F(x)=\sin x-x+0.15$.
34. $F(x)=\sin x^2+\cos x^2-10x$.
35. $F(x)=0.25x^3+x-1.25$.
36. $F(x)=x-\frac{1}{3+\sin x}$
37. $F(x)=\sqrt{1-x}-tgx$.
38. $F(x)=\sqrt{x}+x+\sqrt[3]{x}-2.5$.
39. $F(x)=\sqrt{1+x}-\cos\sqrt{1+x}$.
40. $F(x)=0.4+arctg\sqrt{x}-x$.
41. $F(x)=-ctg\frac{\sqrt{1-x+x^2}}{2}+x^2$.
42. $F(x)=tg^2x-\frac{1}{x}$.
43. $F(x)=x^2\ln^2 x-5x^3$.
44. $F(x)=5\ln\sqrt{1+x}-x\sin x-\cos\sqrt{1+x}$.
45. $F(x)=\ell^{-x}\ln x-x\cos x$.
46. $F(x)=1-x^2+\frac{1}{x}\sin x$.
47. $F(x)=\ell^x\sqrt{1-x}+x\sqrt{1+\ell^{2x}}$.
48. $F(x)=x-\sin\frac{1}{x}+\ln\frac{1}{x^2+1}$.
49. $F(x)=\frac{tg\sqrt{1-x}}{\sqrt{1+x}}-\sqrt{9+x}$.
50. $F(x)=x\ell^x-\frac{1}{x}ctgx-7x$.

ЗАДАНИЕ III (для всех вариантов задания, при организации циклического повторения, следует использовать оператор цикла for):

1. Определить существует ли двузначное число, которое равно квадрату числа его единиц, сложенному с кубом числа его десятков.
2. Определить, существует ли такая четверка последовательных натуральных чисел, сумма квадратов которых равна сумме квадратов трех следующих натуральных чисел.
3. Определить, является ли заданное число совершенным, т.е. равным сумме всех своих (положительных) делителей, кроме самого этого числа (например, число 6 совершенно: 6=1+2+3).

4. Найти все двузначные числа, удвоенная сумма цифр которых равна их произведению? (Пример: 36, 44, 63).
5. Найти все двузначные числа, равные утроенному произведению своих цифр. (Пример: 15, 24).
6. Найти все двузначные числа, которые обладают следующим свойством: куб суммы цифр числа равен квадрату самого числа. (Пример: 27).
7. Найти все трехзначные числа, представимые в виде сумм факториалов своих цифр. (Пример: 145).
8. Найти все трехзначные числа, которые можно представить разностью между квадратом числа, образованного первыми двумя цифрами и квадратом третьей цифры. (Пример: 100).
9. Найти все двузначные числа, сумма цифр которых не меняется при умножении числа на 2,3,4,5,6,7,8,9.
10. Можно ли заданное натуральное число *M* представить в виде суммы квадратов двух натуральных чисел? Написать программу решения этой задачи.
11. Привести дробь вида $\frac{P}{Q}$ к несократимому виду.
12. Дано натуральное число *N*. Найти и вывести все числа в интервале от *1* до *N-1*, у которых сумма всех цифр совпадает с суммой цифр данного числа. (Пример: *N*=44 – 17, 26, 35).
13. Дано натуральное число *N*. Найти наибольшее число *M* (*M*>1), на которое сумма цифр в цифровой записи числа *N* делится без остатка. (Пример: *N*=12345, сумма=15, *M*=5).
14. Найти все целые корни уравнения $ax^3 + bx^2 + cx + d = 0$, где *a, b, c* и *d* – заданные целые числа, причем $a \neq 0$ и $d \neq 0$. *Примечание*: целыми корнями могут быть только положительные и отрицательные делители коэффициента *d*.
15. Натуральные числа *a, b, c* называются числами Пифагора, если выполняется условие $a^2 + b^2 = c^2$. Напечатать все числа Пифагора меньше заданного.

Обработка результатов измерений:

Пример 1. Дана числовая последовательность и некоторое число ε. Найти наименьший номер члена последовательности, для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$. Вывести на экран этот номер и все элементы a_i, где *i=1,2,…,n.* $a_n = \frac{a_{n-1} \cdot a_{n-2}}{a_{n-1} + a_{n-2}}$, $a_1 = 1$, $a_2 = 2.$ (при организации циклического повторения, следует использовать оператор цикла с постусловием do … while).

```
#include<iostream.h>
#include<math.h>
main()
{
double e;
cout <<"Введите e=";
cin >>e;
double a_n;
double a_n_1=2;
double a_n_2=1;
cout <<"a1="<<a_n_2<<endl;
cout <<"a2="<<a_n_1<<endl;
int i=2;
do { a_n=(a_n_1-a_n_2)/(a_n_1*a_n_2);
   if(fabs(a_n-a_n_1)<e) { cout <<"a"<<i++<<"="<<a_n<<endl;
```

```
                                a_n_2=a_n_1;
                                a_n_1=a_n;
                               }
          else break;l
    } while (1);
cout<<"Последний номер последовательности"<<i;
}
```

Пример 2. Вычислить значения определенного интеграла $\int_{-0,5}^{2,0} \frac{x^2}{(1+x)^3} dx$ методом Симпсона с заданной точностью ε. (при организации циклического повторения, следует использовать оператор цикла с предусловием while).

```
#include<iostream.h>
#include<math.h>
main()
{
const double a=-0.5;
const double b=2;
double e;
cout <<"Введите заданную точность=";
cin >>e;
int n;
cout <<"Введите начальное количество отрезков интегрирования=";
cin >>n;
double In=0,I2n=0;
double h,x;
int i;
while (!In||!I2n||fabs(I2n-In)/15>e)
  {In=I2n;
   h=(b-a)/(2*n);
   I2n=a*a/((1+a)*(1+a)*(1+a));
   I2n+=b*b/((1+b)*(1+b)*(1+b));
   i=1;
   while(i<=n) { x=a+h*(2*i-1);
                 I2n+=4*x*x/((1+x)*(1+x)*(1+x));
                 i++;
                }
   i=2;
   while(i<=n) { x=a+h*(2*i-2);
                 I2n+=2*x*x/((1+x)*(1+x)*(1+x));
                 i++;
                }
   I2n=I2n*h/3;
   n=n*2;
   }
cout <<"Значение интеграла="<<I2n;
}
```

Пример 3. Дано натуральное число *N*. Выдать все пары “близнецов”, меньших *N*. “Близнецами” называют простые числа, разность между которыми равна 2. (при организации циклического повторения, следует использовать оператор цикла for):

```
#include<iostream.h>
main()
{
int N;
cout <<"Введите N=";
```

```
cin >>N;
int j,k;
for (int i=2; i<N-2; i++)
 {for (j=2;i%j;j++);
  for (k=2;(i+2)%k;k++);
  if((j==i)&&(k==i+2))cout<<"("<<i<<";"<<i+2<<")-близнецы"
                    << endl;
  }
}
```

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Оператор цикла с постусловием.
2. Оператор цикла с предусловием.
3. Универсальный оператор цикла.
4. Операторы прерывания цикла.

Лабораторная работа № 4

Программирование мульти платформенного Java – приложения в интегрированной среде JBuilder

Цель работы:

Закрепить практические навыки работы с системой программирования языка С++ на примере реализации алгоритмов над статическими массивами.

Порядок выполнения:

Одномерные массивы

Массив представляет собой структуру данных, позволяющую хранить под одним именем совокупность данных любого, но только одного какого-то типа. Массив характеризуется своим именем, типом хранимых элементов и размерностью (количеством хранимых элементов).

Объявление переменной как одномерного массива имеет вид:

тип имя_массива [размерность]

В качестве размерности в объявлении массива разрешено использовать только константные выражения, ввиду чего такой массив будет являться статическим. Тип массива может быть любым.

Например,

int A [10];

объявляет массив с именем A, содержащий 10 целых чисел.

Доступ к элементам массива осуществляется выражением

имя_массива [номер_элемента]

где номер_элемента – индекс, являющийся целочисленным значением в диапазоне от 0 до размерность – 1. То есть номер первого элемента равен 0, а номер последнего элемента на 1 меньше размерности массива. Для предыдущего примера, *A*[0] - значение первого элемента, *A*[1] - второго, *A*[9] - последнего.

Работа с массивами, как правило, непосредственно связана с использованием операторов цикла.

Ниже приведен пример заполнения массива числами Фибоначчи, первые 2 из которых равны 1, а каждое последующее равно сумме двух предыдущих.

```
#include<iostream.h>
main()
{
int B[20];
B[0]=1;
B[1]=1;
for (int i=2; i<20; i++)
      B[i]=B[i-1]+B[i-2];
for (i=0; i<20; i++)
      cout <<"B["<<i<<"]="<<B[i]<<endl;
}
```

При объявлении массива значения элементов носят случайный характер, поэтому иногда желательно совместить объявление массива с заданием элементам начальных значений, т.е. инициализацией. Эти значения перечисляются в списке инициализации после знака равенства, разделяются запятыми и заключаются в фигурные скобки. Например:

```
int A[10] ={1,2,3,4,5,6,7,8,9,10};
double S[5] ={0.1,2.0,-3.2,0,6.5};
```

Таким образом, после данных объявлений A[2] будет содержать значение 3, а S[3] значение 0.

Если начальных значений меньше, чем элементов в массиве, оставшиеся элементы автоматически получают нулевые начальные значения. Например, оператор

```
int A[10] ={1,2,3};
```

задает значения первым трем элементам, а остальные будут равны 0. Оператор

```
int A[10] ={0};
```

присваивает нулевые значения всем элементам массива.

В объявлении массива со списком инициализации размерность массива можно не указывать. Тогда количество элементов массива будет равно количеству элементов в списке начальных значений. Например, объявление

```
double S[] ={0.1,2.0,-3.2,0,6.5};
```

создает массив из пяти элементов.

В объявлении массива в качестве размерности лучше всегда использовать именованные константы. Ниже приведен пример объявление массива, заполнение его случайными значениями от -10 до 10 и подсчет суммы его элементов:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
randomize();
double c[15];
for (int i=0; i<15; i++)
      c[i]=10-double(random(201))/10;
for (int i=0; i<15; i++)
      cout <<"c["<<i<<"]="<<c[i]<<endl;
double S=0;
for (int i=0; i<15; i++) S+=c[i];
cout<<"S="<<S;
```

```
}
```

Если в дальнейшем потребуется массив не из 15 элементов, а, например, из 100, нужно будет изменить размерность массива и в объявлении, и во всех операторах, работающих с этим массивом (в данном случае в операторах for. Таких операторов в разных программах может быть очень много и поэтому она плохо масштабируется. Грамотнее будет реализовать этот пример следующим образом:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
const int N=15; // объявление константы
randomize();
double c[N];
for (int i=0; i<N; i++)
      c[i]=10-double(random(201))/10;
for (int i=0; i<N; i++)
      cout <<"c["<<i<<"]="<<c[i]<<endl;
double S=0;
for (int i=0; i<N; i++) S+=c[i];
cout<<"S="<<S;
}
```

В этом случае мы вводим именованную константу N и используем ее во всех операторах, в которых требуется указать размерность массива. Тогда при необходимости изменения размерности массива, достаточно будет изменить его только в объявлении константы N.

```
const int N=100;
```

Программа сразу становится масштабируемой. А объявление N как константы гарантирует, что объявленное значение не будет случайно изменено где-то в программе.

Аналогичный результат можно получить, если заменить объявление константы директивой компилятора #define

```
#define N 10
```

В ряде случаев требуются константные массивы, данные из которых программа может только читать. Такие массивы обязательно должны инициализироваться в момент объявления. Например, требуется неизменяемый массив простых чисел, значения которых не больше 30:

```
const prime[] = {2,3,5,7,11,13,17,19,23,29};
```

Двумерные массивы

По аналогии с одномерными массивами, можно объявлять и многомерные статические массивы, т.е. массивы, элементами которых являются массивы. Например, двумерный массив можно объявить следующим образом:

```
int A2 [10] [3];
```

Такое объявление описывает двумерный массив, который можно представить себе как матрицу, состоящую из 10 строк и 3 столбцов.

Доступ к значениям элементов многомерного массива обеспечивается через номера элементов, заключенные в квадратные скобки. Например, A2[3][2] – значение элемента, лежащего на пересечении четвертой строки и третьего столбца. Также как и при работе с одномерными массивами номера элементов начинаются с 0.

Если многомерный массив инициализируется при его объявлении, список значений по каждой размерности заключается в фигурные скобки. Приведенный ниже оператор объявляет двумерный массив C размерностью 4 на 3 и инициализируется.

```
int C[4][3]= {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

1	2	3
4	5	6
7	8	9
10	11	12

В этом случае, элемент C[1][2] равен 6, а элемент C[2][1] равен 8 и т.д.

Если в списке инициализации в какой-то из размерностей не хватает данных, то все дальнейшие не перечисленные элементы считаются равными нулям. Например

```
double D [5] [4] = {{1.0,2.1,3.2,1.8},
                    {-4.1,5.0,1.1},
                    {-7.1,2.2}};
```

создаст матрицу размерностью 5 на 4, состоящую из элементов вещественных чисел и присвоит им следующие начальные значения:

```
     1.      2.      3.      1.
0        1       2       8
     -       5.      1.      0
4.1      0       1
     -       2.      0       0
7.1      2
     0       0       0       0
     0       0       0       0
```

Работа с двумерными массивами непосредственно связана с использованием одновременно двух операторов цикла, внешнего и внутреннего. Например, необходимо определить в двумерном случайно заполненном массиве, каких чисел больше положительных или отрицательных. Ноль во внимание не принимается:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
const int n=5;
const int m=10;
randomize();
int matr[n][m];
int i,j;
for (i=0; i<n; i++)
  for (j=0; j<m; i++) matr[i][j]=100-random(201);
for (i=0; i<n; i++) {
  for (j=0; j<m; i++) cout <<matr[i][j]<<" ";
  cout <<endl;
                  }
int p=0;
for (i=0; i<n; i++)
  for (j=0; j<m; i++)
     if (matr[i][j]!=0)
       if (matr[i][j]>0) p++;
         else p--;
if (p==0) cout <<"равное количество";
    else if (p>0) cout <<"положительных больше";
        else cout <<"отрицательных больше";
}
```

Сортировка массивов

Задачей сортировки массивов является преобразование исходной последовательности элементов массива в последовательность, содержащую те же элементы, но в порядке возрастания (или убывания) значений.

Сортировка методом выбора. При сортировке массива *a*[0], *a*[1], ..., *a*[*N*-1] методом простого выбора среди всех элементов находят элемент с наибольшим значением *a[i]*, и этот элемент ставят в самый конец массива, т.е. поменять местами значения элементов массива

a[i] и *a[N-1]*. Далее, игнорируя последний элемент массива *a[N-1]*, следует выполнить поиск наибольшего элемента в подмассиве *a[0], a[1], ..., a[N-2]* и поменять его местами с последним элементом подмассива *a[N-2]*. Этот процесс повторяется до тех пор, пока не будут найдены *N-1* наибольших значений и не дойдем до подмассива *a[0]*, содержащего к этому моменту наименьшее значение. Работа алгоритма иллюстрируется следующим примером:

Исходная последовательность *N=7*:
3 1 7 5 6 4 2
max =7 из *a[0]…a[N-1]*
3 1 2 5 6 4 7
max =6 из *a[0]…a[N-2]*
3 1 2 5 4 6 7
max =5 из *a[0]…a[N-3]*
3 1 2 4 5 6 7
max =4 из *a[0]…a[N-4]*
3 1 2 4 5 6 7
max =3 из *a[0]…a[N-5]*
2 1 3 4 5 6 7
max =2 из *a[0]…a[1]*
1 2 3 4 5 6 7

Программная реализация этого метода на языке Си++, будет выглядеть следующей:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
const N=7; //размерность массива
int A[N];
randomize();
int i;
for(i=0; i<N; i++) A[i]=random(10);//заполнение массива
int index, max,j;
for (i=N-1; i>0; i--)
  { max=A[0];
    index=0;
    for(j=1; j<=i; j++)
      if (A[j]>max) { index=j;
                      max=A[j];
                    }
    A[index]=A[i];
    A[i]=max;
  }
}
```

Если в условии if(**A[j]>max**), поменять знак операции сравнения на меньше (<), то будут выбираться наименьшие значения и сортировка массива будет осуществляться по убыванию.

Сортировка методом обмена. При сортировке массива *a[0], a[1], ..., a[N-1]* методом обмена (“методом пузырька”), начиная с первого элемента сравнивают два соседних элемента (*a[i]* и *a[i+1]*) и меняют их местами, если *a[i]>a[i+1]*, т.е. если они нарушают порядок. Такое сравнение и обмен выполняют до тех пор, пока не будет достигнут конец массива и последним элементом *a[N-1]* окажется элемент массива с наибольшим значением. Далее, игнорируя последний элемент массива *a[N-1]*, следует повторить такую же схему сравнения и обмена в подмассиве *a[0], a[1], ..., a[N-2]*. И так далее. На последнем шаге будут сравниваться только текущие значения *a[0]* и *a[1]*. Работа алгоритма иллюстрируется следующим примером:

Исходная последовательность	**6**	**3**	**7**	**1**	**5**
N=5:					
Шаг 1:	**6**	**3**	**7**	**1**	**5**
	3	**6**	**7**	**1**	**5**
	3	**6**	**7**	**1**	**5**
	3	**6**	**1**	**7**	**5**
	3	**6**	**1**	**5**	**7**
Шаг 2:	**3**	**6**	**1**	**5**	7
	3	**6**	**1**	**5**	7
	3	**1**	**6**	**5**	7
	3	**1**	**5**	**6**	7
Шаг 3:	**3**	**1**	**5**	6	7
	1	**3**	**5**	6	7
	1	**3**	**5**	6	7
Шаг 4:	**1**	**3**	5	6	7
	1	**3**	5	6	7
Итоговая последовательность:	1	3	5	6	7

Программная реализация этого метода на языке Си, будет выглядеть следующей:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
const N=7;
int A[N];
randomize();
int i;
for (i=0; i<N; i++) A[i]=random(100);
int swap, j;
for (i=N-1; i>1; i--)
   for(j=0; j<i; j++)
     if (A[j]>A[j+1]) {swap=A[j+1];
                       A[j+1]=A[j];
                       A[j]=swap;
                      }
}
```

Если в условии if (**A[j]>A[j+1]**), поменять знак операции сравнения на меньше (<), то последними элементами в подмассивах будут оказываться наименьшие значения и сортировка массива будет осуществляться по убыванию.

Сортировка челночным методом. При сортировке массива *a[0], a[1], ..., a[N-1]* челночным методом, начиная с первого элемента сравнивают два соседних элемента (*a[i]* и *a[i+1]*), если порядок нарушен (*a[i]<=a[i+1]*), то продвигаются на один элемент вперед (*i=i+1*), если порядок нарушен *(a[i]>a[i+1]),* то производится перестановка и сдвигаются на один элемент назад *(i=i-1)*. Таким образом сравнивая и переставляя элементы доходят до конца массива *(i=N-1)*, при этом массив становится упорядоченным. Работа алгоритма иллюстрируется следующим примером:

Исходная последовательность
N=5
i=0 (порядок нарушен)
i=0 (порядок не нарушен)
i=1 (порядок не нарушен)
i=2 (порядок нарушен)
i=1 (порядок не нарушен)

i=2 (порядок не нарушен)
i=3 (порядок нарушен)
i=2 (порядок нарушен)
i=1 (порядок нарушен)
i=0 (порядок не нарушен)
i=1 (порядок не нарушен)
i=2 (порядок не нарушен)
i=3 (порядок не нарушен)
i=4: Итоговая последовательность

Программная реализация этого метода на языке Си++, будет выглядеть следующей:

```
#include<iostream.h>
#include<stdlib.h>
main()
{
const N=7;
int A[N];
randomize();
int i;
for (i=0; i<N; i++) A[i]=random(100);
int swap;
i=0;
while (i<N-1)
  if (A[i]>A[i+1]) {swap=A[i+1];
                    A[i+1]=A[i];
                    A[i]=swap;
                    i--;
                    if (i<0) i=0;
                    }
    else i++;
}
```

Если в условии if (**A[i]>A[i+1]**), поменять знак операции сравнения на меньше (<), то сортировка массива будет осуществляться по убыванию.

Сортировка методом вставок. Данный метод сортировки массива основан на вставке элементов из неупорядоченной части массива в упорядоченную. Считается, что первоначально массив *a[0], a[1], ..., a[N-1]* является не отсортированным, поэтому упорядоченная часть будет состоять из одного первого элемента *a[0]*, а неупорядоченная из всех элементов массива *a[1], ..., a[N-1]*. На каждом шаге метода из неупорядоченной части извлекается первый элемент и помещается в упорядоченную, так чтобы не нарушался порядок. При этом количество элементов в упорядоченной части увеличивается на 1, а в неупорядоченной уменьшается на 1. Процесс вставки элементов происходит до тех пор, пока весь массив не окажется отсортированным.

Исходная последовательность *N=7*:	**3**	**1**	**7**	**5**	**6**	**4**	**2**
	3	1	7	5	6	4	2
	1	3	7	5	6	4	2
	1	3	7	5	6	4	2
	1	3	5	7	6	4	2
	1	3	5	6	7	4	2
	1	3	4	5	6	7	2
Итоговая последовательность:	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Программная реализация этого метода на языке Си, будет выглядеть следующей:

```
#include<iostream.h>
```

```
#include<stdlib.h>
main()
{
const N=7;
int A[N];
randomize();
int i,j,element;
for (i=0; i<N; i++) A[i]=random(100);
for (i=1; i<N; i++)
     { element=A[i];
       j=i;
       while (A[j-1]>element)
          {A[j]=A[j-1];
           j--;
           if (j<1) break;
           }
       A[j]=element;
     }
for (i=0; i<N; i++) cout << A[i]<< " ";
}
```

Если в условии while (**A[j-1]>element**), поменять знак операции сравнения на меньше (<), то сортировка массива будет осуществляться по убыванию.

Варианты задания для выполнения лабораторной работы:

ЗАДАНИЕ I:

Для всех вариантов заполнение массива организовать в ходе выполнения программы вводом значений с клавиатуры. Также в вариантах, где задан упорядоченный по возрастанию массив, осуществить проверку правильности ввода значений.

1. Дан массив, заполненный целочисленными значениями. Найти количество его локальных минимумов и максимумов, а также определить из них максимальный и минимальный.

2. Дан массив, заполненный целочисленными значениями. Определить количество его промежутков монотонности (то есть участков, на которых его элементы возрастают или убывают).

3. Дан целочисленный массив размера *N*. Если он является перестановкой, то есть содержит все числа от 1 до *N*, то вывести 0, в противном случае вывести номер первого недопустимого элемента.

4. Дан целочисленный массив. Назовем *серией* группу подряд идущих одинаковых элементов, а длиной серии — количество этих элементов (длина серии может быть равна 1). Вывести массив, содержащий длины всех серий исходного массива.

5. Дано множество *A* из *N* точек с координатами *(x,y)*. Среди всех точек этого множества найти точки наиболее близкие и наиболее удаленные от начала координат, лежащие в первой, второй, третьей и четвертой четверти.

6. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти пары различных точек этого множества с минимальным и максимальным расстоянием между ними и сами эти расстояния.

7. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти две точки из данного множества. Для одной точки сумма расстояний до остальных точек должна быть минимальна, а для другой максимальна.

8. Даны множества *A* и *B*, состоящие соответственно из *N1* и *N2* точек с координатами *(x,y)*. Найти минимальное и максимальное расстояние между точками этих множеств и сами точки, расположенные на этом расстоянии.

9. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти наименьший периметр треугольника, вершины которого принадлежат различным точкам множества *A*, и сами эти точки (точки выводятся в том же порядке, в котором они перечислены при задании множе-

ства *A*).

10. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти три точки из этого множества, которые составляли бы вершины треугольника, и сумма расстояний от сторон до других точек была бы минимальна.

11. Дано множество *A* из *N* точек с координатами *(x,y)*. Определить количество прямоугольных треугольников, которые создаются этими точками.

12. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти точку из этого множества, которая являлась бы центром окружности с минимальным радиусом. Все точки множества должны находится внутри этой окружности.

13. Дано множество *A* из *N* точек с координатами *(x,y)*. Найти наибольшую длину ломанной линии, которую можно составить из точек этого множества.

14. Секретный замок для сейфа состоит из 10 расположенных в ряд ячеек, в которые надо вставить игральные кубики. Но дверь открывается только в том случае, когда в любых трех соседних ячейках сумма точек на передних гранях кубиков равна 10. (Игральный кубик имеет на каждой грани от 1 до 6 точек.) Напишите программу, которая разгадывает код замка при условии, что два кубика уже вставлены в ячейки.

15. В целочисленном массиве найти повторяющиеся числа. Вывести эти числа, а также количество их повторений.

ЗАДАНИЕ II:

Для всех вариантов заполнение массива организовать с помощью датчика случайных чисел.

1 – 4. Дано множество *C*, состоящее из массивов вещественных чисел *A* и *B* размерностью *N*. Причем значения элементов массива *B* определяются следующим образом: *B[0]=0, B[i]=A[i]-A[i-1] i=1..N-1*. Считать, что значение множества *C[k]≥C[m]*, если либо *A[k]≥A[m]* или *A[k]<A[m]* и *A[k]+B[k]≥ A[m]+B[m]*. Отсортировать множество *C* по возрастанию в соответствии с указанным условием. Сортировку произвести:

1. методом обмена;
2. методом вставкой;
3. методом выбора;
4. методом Шелла.

5 – 8. Дано множество *C* комплексных чисел, состоящее из двух массивов вещественных чисел *A* и *B* размерностью *N* (*c=a+jb*). Отсортировать данное множество комплексных чисел по убыванию значений их модулей. *Примечание*: модуль комплексного числа равен $|c| = \sqrt{a^2 + b^2}$. Сортировку произвести:

5. методом обмена;
6. методом вставкой;
7. методом выбора;
8. методом Шелла.

9 – 12. Целочисленный массив *X* из *n* элементов разбит на *m* фрагментов. В целочисленном массиве *K* из *m* элементов хранятся длины соответствующих фрагментов (все *K[i]* различны, их сумма равна *n*). Упорядочить массив *K* по возрастанию, переставив соотвествующие фрагменты в массиве *X*.

Сортировку произвести:

9. методом обмена;
10. методом вставкой;
11. методом выбора;
12. методом Шелла.

13 – 16. Дана некоторая функция *f(x,y)* и множество A из *N* точек с целочисленными координатами *(x,y)*, состоящее из двух массивов. Известно, что значение функции *f(x,y)* в точке (x_0,y_0) равно площади фигуры, образованной прямыми $y=x_0$ и $x=y_0$ с осями координат. Расположить точки данного множества в соответствии с убыванием значений заданной функции.

Сортировку произвести:

13. методом обмена;
14. методом вставкой;
15. методом выбора;
16. методом Шелла.

Обработка результатов измерений:

Пример 1. Среди заданных *N* точек на плоскости найти две, что прямые, проходящие через них перпендикулярно соединяющему эти точки отрезку, образуют полосу, в которой содержится максимальное количество точек из множества *N*.

Рассмотрим для решения этой задачи необходимые сведения и формулы.

Уравнение прямой, проходящей через точки *P*, *Q*, имеет вид:

$Ax + By + C = 0$,

где $A = Y_P - Y_Q$, $B = Y_Q - Y_P$,

$C = (X_P - X_Q) \cdot Y_P + (Y_Q - Y_P) \cdot X_P$;

Уравнение прямой, проходящей через точку *P* перпендикулярно отрезку *PQ*, имеет вид:

$-Bx + Ay + D = 0$,

где $D = B \cdot X_P - A \cdot Y_P$.

Если в уравнении прямой подставить координаты точек, лежащих по одну сторону от этой прямой, получим значения одного знака, иначе – разных знаков.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
main()
{
clrscr();
const int N=20;
randomize();
double x[N];
double y[N];
int i;
for (i=0; i<N; i++)
   {
    x[i]=double(random(401)-200)/10;
    y[i]=double(random(401)-200)/10;
    }
for (i=0; i<N; i++)
    cout <<i+1<<"-точка ("<<x[i]
        <<";"<<y[i]<<");"<<endl;
double A,B,C;
double AP,BP,CP;
double AQ,BQ,CQ;
double Z1,Z2,Z3;
int max=0,tek,P,Q,N1,N2;
//выбираем очередную пару точек
for(P=0; P<N-1; P++)
  for(Q=P+1; Q<N; Q++)
    {
//находим коэффициенты уравнения прямой PQ
    A=y[P]-y[Q];
    B=x[Q]-x[P];
    C=(x[P]-x[Q])*y[P]+(y[Q]-y[P])*x[P];
```

```
//прямые, проходящие через P, Q перпендикулярно PQ
    AP=-B;
    BP=A;
    CP=B*x[P]-A*y[P];
    AQ=-B;
    BQ=A;
    CQ=B*x[Q]-A*y[Q];
//определим, сколько точек попадает в полосу
    tek=0;
    Z1=AP*x[Q]+BP*y[Q]+CP;
    Z3=AQ*x[P]+BQ*y[P]+CQ;
//просматриваем все точки, кроме P и Q
    for(i=0; i<N; i++)
     if (i!=Q && i!=P)
       {
//точки i и Q должны быть по одну сторону от
//прямой, проходящей через P
        Z2=AP*x[i]+BP*y[i]+CP;
        if (Z1*Z2>0)
          {
//точки i и P должны быть по одну сторону от
//прямой, проходящей через Q
           Z2=AQ*x[i]+BQ*y[i]+CQ;
           if (Z3*Z2>0) tek++;
           }
         }
    if (tek>max)
     {
      max=tek;
      N1=P;
      N2=Q;
     }
    }
cout<<"Исходные точки:"<<endl
   <<"("<<x[N1]<<";"<<y[N1]<<")"<<endl
   <<"("<<x[N2]<<";"<<y[N2]<<")"<<endl;
getch();
}
```

Пример 2. Дано множество *A* из *N* пар точек с целочисленными координатами. Расположить точки данного множества по возрастанию значений пересечения прямой, проходящей через эту точку, с осью абсцисс. Пары точек, через которые проходит прямые параллельные прямые исключить из сортировки. Сортировку произвести методом обмена.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
main()
{
const N=10;
int X1[N],Y1[N];
int X2[N],Y2[N];
int i;
randomize();
//заполнение массивов случайным образом
for (i=0;i<N;i++)
  {
```

```
   X1[i]=random(41)-20;
   Y1[i]=random(41)-20;
   X2[i]=random(41)-20;
   Y2[i]=random(41)-20;
  }
//вывод первоначальных данных
for (i=0;i<N;i++)
   cout<<i+1<<" пара точек:"<<endl
       <<"("<<X1[i]<<";"<<Y1[i]<<") "
       <<"("<<X2[i]<<";"<<Y2[i]<<") "<<endl;
int R=N;
i=0;
int swapX1,swapX2,swapY1,swapY2;
//исключение пар точек, прямые через которые
//параллельны оси абсцисс, поместив их в конец
//массива
while (i<R && R>1)
 if (Y1[i]==Y2[i]) {
                swapX1=X1[R-1];
                swapY1=Y1[R-1];
                swapX2=X2[R-1];
                swapY2=Y2[R-1];
                X1[R-1]=X1[i];
                Y1[R-1]=Y1[i];
                X2[R-1]=X2[i];
                Y2[R-1]=Y2[i];
                X1[i]=swapX1;
                Y1[i]=swapY1;
                X2[i]=swapX2;
                Y2[i]=swapY2;
                R--;
                }
    else i++;
//сортировка R пар точек, прямые через которые
//пересекаю ось абсцисс
int j;
double A1,A2;
for (i=R-1; i>1; i--)
   for(j=0; j<i; j++)
     {
     A1=(X1[j]-X2[j])*Y1[j]-(Y1[j]-Y2[j])*X1[j];
     A1=A1/(Y1[j]-Y2[j]);
     A2=(X1[j+1]-X2[j+1])*Y1[j+1]-(Y1[j+1]-
       Y2[j+1])*X1[j+1];
     A2=A2/(Y1[j+1]-Y2[j+1]);
     if (A1>A2) { swapX1=X1[j+1];
                swapY1=Y1[j+1];
                swapX2=X2[j+1];
                swapY2=Y2[j+1];
                X1[j+1]=X1[j];
                Y1[j+1]=Y1[j];
                X2[j+1]=X2[j];
                Y2[j+1]=Y2[j];
                X1[j]=swapX1;
                Y1[j]=swapY1;
```

```
                X2[j]=swapX2;
                Y2[j]=swapY2;
              }
      }
  //Вывод исключенных пар точек
  if (R!=N)
    {
     cout<<"Исключенные точки:"<<endl;
     for (i=R;i<N;i++)
       cout <<"("<<X1[i]<<";"<<Y1[i]<<") "
           <<"("<<X2[i]<<";"<<Y2[i]<<") "<<endl;
     }
  //Вывод отсортированных пар точек
  cout<<"Отсортированные пары точек:"<<endl;
     for (i=0;i<R;i++)
       cout <<"("<<X1[i]<<";"<<Y1[i]<<") "
           <<"("<<X2[i]<<";"<<Y2[i]<<") "<<endl;
  getch();
  }
```

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Охарактеризовать одномерные массивы.
2. Охарактеризовать двумерные массивы.
3. Задачи и методы сортировки массивов.

Лабораторная работа № 5

Создание WEB – ресурса с использованием апплетов языка Java

Цель работы:

Закрепить практические навыки работы с системой программирования языка С++ на примере реализации алгоритмов над динамическими многомерными массивами.

Порядок выполнения:

Указатели

Указатель – это переменная, значение которой равно значению адреса памяти, по которому лежит значение некоторой другой переменной. В этом смысле имя этой другой переменной отсылает к ее значению прямо, а указатель – косвенно. Ссылка на значение посред-

ством указателя называется косвенной адресацией.

Указатели в языке Си делятся на два вида: указатели на объекты и указатели на функции. Свойства и правила их использования различны.

Указатели, подобно любым другим переменным, перед своим использованием должны быть объявлены. Объявление указателя на объект имеет вид:

тип *имя_указателя;

где тип – один из предопределенных или определенных пользователем типов, а имя_указателя – указатель.

Например,

int *APtr;

объявляет переменную APtr типа int * (т.е указатель на целое число) и читается следующем образом: “ APtr является указателем на объект целочисленного типа”. Каждая переменная, объявляемая как указатель, должна иметь перед собой символ (*), который обозначает операцию косвенной адресации.

При объявлении указателя возможна его инициализация. Имеется две формы инициализации указателя:

1) тип *имя_указателя=инициализирующее_значение

2) тип *имя_указателя (инициализирующее_значение)

В качестве инициализирующего_значения может использоваться:

- явно заданный участок памяти

double *cc=0x1047;

- указатель, уже имеющий значение

float *ca;

…

float *cb=ca;

- выражение, позволяющее получить адрес объекта с помощью операции ссылки (&)

int a;

…

int *ce=&a;

Указатели должны инициализироваться либо при своем объявлении, либо с помощью оператора присваивания. Использовать указатель без присвоения ему какого-нибудь участка памяти нельзя. Указатель может получить в качестве начального значения 0 или NULL. Указатель с начальным значением 0 или NULL ни на что не указывает. NULL – это символическая константа, определенная специально для цели показать, что данный указатель ни на что не указывает. Пример объявления указателя с его инициализацией:

int *countPtr = NULL;

Для присваивания указателю адреса некоторой переменной используется операция адресации & (операция ссылки), которая возвращает адрес своего операнда. Например, если имеются объявления

int y = 5;

int *yPtr, x;

то оператор

yPtr = &y;

присваивает адрес переменной y указателю yPtr.

Присвоив указателю адрес конкретного участка памяти, можно с помощью операции разыменования не только получать, но и изменять содержимое этого участка памяти. Для этого используется операция разыменования * (операция косвенной адресации). Она возвращает значение объекта, на который указывает ее операнд (т.е. указатель).

Если присвоить указателю адрес конкретного объекта или значение уже инициализированного указателя, то это превратит запись *имя_указателя в синоним уже имеющегося имени объекта. Запись,

double z;

double *A=&z;

double *C, *D;

C=&z;

D=A;

превратит *A, *C, *D в синонимы переменной z. Изменяя значение, лежащее по адресу на которое указывают указатели, автоматически изменяется значение переменной, так как указатели A, C, D и переменная z связаны между собой одним участком памяти. Продолжая начатый пример

z=6;
double A1, C1, D1;
A1=*A;
C1=*C;
D1=*D;

присвоит переменным A1, C1, D1 значение 6, т.е. значение переменной z, на которую указывают указатели A, C, D.

Операции new и delete

Чтобы связать неинициализированный участок памяти, еще не занятым никаким объектом программы, используется оператор new:

указатель=new тип (инициализирующее_значение);

Операция возвращает указатель на динамически размещенный в памяти объект, т.е. предоставляет указателю память под объект заданного типа. Пример

double *A=new double;
int *B;
B=new int;

объявляет указатели и выделяет им свободное место, не занятое никаким объектом.

Инициализирующее_значение задает начальные значения создаваемого объекта. Запись

double *C=new double (-1.8);

предоставляет место указателю C под объект вещественного типа и инициализирует его, т.е. записывает начальное значение -1.8.

Если операция new возвратила нулевое значение адреса – NULL, то это значит, что операционная система не может выделить память под данный объект, поэтому после использования оператора new желательно всегда проверять адрес выделенного участка памяти

double *A;
A=new double;
if (A==NULL) …

Динамически распределенную память следует освобождать, когда отпадает необходимость в размещенных в ней объектах. Освобождение памяти осуществляется с помощью операции delete, которая имеет следующий синтаксис

delete имя_указателя;

Пример

int *B=new int;
…
delete B;

объявляет указатель, выделяя и освобождая место под объект в адресном пространстве.

Освобождение памяти уничтожает сам объект, а не указатель. В дальнейшем, указателю можно заново выделить участок памяти под новый объект и освободить эту память, что позволяет более гибко использовать оперативное адресное пространство компьютера. Поэтому, объекты созданные с помощью операции new называются динамическими.

long *P;
…
P=new long;
…
delete P;
…
P=new long;
…
delete P;

Операция delete освобождает память, но сама не задает указателю значение NULL, по-

этому во избежание использования в дальнейшем неинициализированного указателя желательно это делать программно.

```
delete P;
P=NULL;
```

Операции над указателями

Указатели могут применяться как операнды в арифметических выражениях, выражениях присваивания и выражениях сравнения. Однако, не все операции, обычно используемые в этих выражениях, разрешены применительно к переменным указателям.

С указателями может выполняться ограниченное количество арифметических операций. Указатель можно увеличивать (++), уменьшать (--), складывать с указателем целые числа (+ или +=), вычитать из него целые числа (- или -=) или вычитать один указатель из другого.

Сложение указателей с целыми числами отличается от обычной арифметики. Прибавить к указателю 1 означает сдвинуть его на число байтов, содержащихся в переменной, на которую он указывал. Обычно подобные операции применяются к указателям на массивы. Например, запись

```
int *Pt;
…
Pt+=2;
```

увеличит значение Pt (т.е. адрес в памяти, на который указывает Pt), не на два, а на четыре байта, так как один объект целочисленного типа int в памяти занимает два байта.

Аналогичные правила действуют и при вычитании из указателя целого значения.

Однако при использовании арифметических операций над указателями нельзя полагать, чтоб две переменные – указатели одинакового типа будут находится в памяти вплотную друг к другу, если только они не соседствуют в массиве.

Сравнение указателей операциями >, <, >=, <= также имеют смысл только для указателей на один и тот же массив. Однако, операции отношения == и != имеют смысл для любых указателей. При этом указатели равны, если они указывают на один и тот же адрес в памяти.

Указатель можно присваивать другому указателю, если оба указателя имеют одинаковый тип. В противном случае нужно использовать операцию приведения типа, чтобы преобразовать значение указателя в правой части присваивания к типу указателя в левой части присваивания. Исключением из этого правила является указатель на void (т.е. void*), который является общим указателем, способным представлять указатели любого типа. Указателю на void можно присваивать все типы указателей без приведения типа. Однако указатель на void не может быть присвоен непосредственно указателю другого типа – указатель на void сначала должен быть приведен к типу соответствующего указателя.

Связь массивов и указателей

Массивы и указатели Си++ тесно связаны и могут быть использованы почти эквивалентно. При определении массива, имя массива является указателем константой, значением которой служит адрес первого элемента массива (с индексом 0), а запись

имя_массива[индекс]

является выражением с двумя операндами. Первый из них, т.е. имя_массива – константный указатель – адрес начала массива в основной памяти, а индекс – это выражение целого типа, определяющее смещение от начала массива.

Используя операцию обращения по адресу * (операция разыменования), действие бинарной операции можно представить следующим образом:

***(имя_массива + индекс)**

т.е. операндами для операции [] служат имя_массива и индекс. Из этого становится очевидным, почему индекс первого элемента массива равен нулю. Таким образом, запись *имя_массива – обращение к первому элементу массива, *(имя_массива+1) – обращение ко второму, и т.д.

Поскольку сложение *(имя_массива+индекс) коммутативно, то возможна такая эквивалентная запись *(индекс+имя_массива) и, следовательно, индекс [имя_массива] именует тот же элемент массива, что имя_массива [индекс].

В некоторых конструкциях можно использовать выражение имя_массива [индекс] с отрицательным значением индекса. В этом случае имя_массива должен указывать не на начало

массива, т.е. не на его нулевой элемент.

Ссылки

Ссылки – это специальный тип указателя, который позволяет работать с указателем как с объектом. Объявление ссылки делается с помощью операции ссылки, обозначаемой амперсантом (&) – тем же символом, который используется для адресации. Например, если имеется объявление

double*P = new double;

то можно создать ссылку на этот объект оператором:

double& Ref = *P;

Объявленная таким образом переменная Ref является ссылкой на объект вещественного типа. Она может рассматриваться как псевдоним объекта. Эта переменная является именно указателем, а не самим объектом. Но работа с ней производится как с объектом, в отличие от указателя.

*P=*P+0.5;

Ref=Ref+0.5;

Чаще всего ссылки используются при передаче в функции параметров по ссылке.

Одномерные динамические массивы

В соответствии с синтаксисом операция new может иметь следующий формат:

имя_указателя=new тип_массива [размерность];

Такая операция позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать. В результате выполнения этой операции возвращается адрес первого элемента массива.

При выделении динамической памяти размер массива должен быть определен явно, т.к. операция new получает информацию о выделяемом участке из размерности и типе_массива. Оператор

double *A=new double [30];

создает динамический массив из 30 элементов вещественного типа.

В отличие от объявления статических массивов, при выделении динамической памяти под указатель в качестве размерности могут использоваться переменные целочисленного типа, что позволяет задавать количество элементов массива программно.

```
float *e;
int N;
cout << “N=”;
cin >>N;
e=new float [N];
```

Доступ к значениям элементов динамического массива может быть осуществлен таким же образом, как и при работе со статическими массивами. Продолжая пример выше, можно записать

```
for(int i=0; i<N; i++)
e[i]=double(random(41)-20)/10;
```

Изменять значение указателя на динамический массив нужно с осторожностью, так как указатель, значение которого определяется при выделении памяти, используется затем для освобождения этой памяти.

Освободить память, выделенную под динамический массив можно используя операцию delete, которая в этом случае имеет следующий синтаксис

delete [] имя_указателя;

Таким образом, оператор

delete []e;

освободит целиком всю память, выделенную для определенного выше массива, если указатель e адресует его начало.

Массивы указателей или многомерные динамические массивы

Аналогично одномерным массивам, можно с помощью указателя и операции new выделить место под двумерный динамический массив. Для этого используются массивы указателей, объявление которых имеет следующий вид:

тип_массива ** имя_указателя;

Данная запись может быть интерпретирована как объявление указателя на объект типа тип_массива *.

Выделение участка памяти указателю на объект типа тип_массива * с помощью операции new позволяет получить массив нужного размера, элементами которого являются также указатели, только на объекты типа тип_массива.

Например

```
double **B;
B=new double* [20];
```

выделяет место под массив указателей из 20 элементов.

Так как, каждый элемент массива указателей тоже является указателем, то ему соответственно также можно выделить некоторый участок памяти в виде массива, только уже вещественного типа

```
B[3]=new double [10];
```

Доступ к элементам созданного массива осуществляется через индекс элемента массива указателей и индекс массива, на который указывает элемент – указатель, а именно

```
B[3][5]=10;
```

Данная запись обращения к элементам аналогична доступу к элементам в двумерном статическом массиве.

Приведенный ниже пример создает двумерный динамический массив из *n* – строк и *m* – столбцов, введенных пользователем

```
int n,m;
double **matr;
cout << “n=”;
cin>>n;
cout << “m=”;
cin>>m;
//выделение места под массив указателей из
//n элементов, т.е. выделение места под n строк
matr=new double*[n];
//выделение места каждому указателю из массива //указателей по m вещественных элементов,
//т.е. выделение места под m столбцов
for (int i=0; i<n; i++)
    matr[i]=new double[m];
```

Количество элементов, выделяемых всем указателям из массива указателей, не обязательно должно быть одинаковым. Пример

```
int n;
double **matr;
cout << “n=”;
cin>>n;
matr=new double*[n];
for (int i=0; i<n; i++)
    matr[i]=new double[i+1];
```

реализует двумерный динамический массив в виде “лесенки со ступеньками”, т.е. первый указатель matr[0] будет содержать массив только из одного элемента, указатель matr[1] будет содержать массив из двух элементов, а последний указатель matr[n-1] будет содержать *n* элементов.

Освобождение памяти, выделенное под двумерный динамический массив, осуществляется в обратном порядке по отношению к выделению памяти, т.е. вначале освобождается память, выделенная под каждый элемент – указатель, а затем освобождается память, выделенная под массив указателей. Следующая запись демонстрирует освобождение памяти, выделенное в предыдущем примере

```
for (int i=0; i<n; i++)
    delete [] matr[i];
```

```
delete []matr;
```

Варианты задания для выполнения лабораторной работы:

ЗАДАНИЕ I:

1. Заполнить массив следующим образом:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \dots & n \\ 2 & 1 & 2 & 3 & 4 & \dots & n-1 \\ 3 & 2 & 1 & 2 & 3 & \dots & n-2 \\ 4 & 3 & 2 & 1 & 2 & \dots & n-3 \\ 5 & 4 & 3 & 2 & 1 & \dots & n-4 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ n & n-1 & n-2 & n-3 & n-4 & \dots & 1 \end{pmatrix},$$

n – нечетное.

2. Заполнить массив следующим образом:

$$\begin{pmatrix} n+1 & 1 & 1 & \dots & 1 & 1 \\ 2 & n+2 & 2 & \dots & 2 & 2 \\ 3 & 2 & n+3 & \dots & 3 & 3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ n-1 & n-2 & n-1 & \dots & 2n-1 & n-1 \\ n & n-1 & n-2 & \dots & 2 & 2n \end{pmatrix},$$

n – четное.

3. Строки матрицы *A(m, n)* заполнить не полностью: в массиве *L(m)* указать количество элементов в каждой строке. Переслать элементы матрицы построчно в начало одномерного массива *T(m· n)*, подсчитать их количество.

4. Сформировать квадратную матрицу порядка $n \times n$ следующим образом (n – четное):

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ n & n-1 & n-2 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ n & n-1 & n-2 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n & n-1 & n-2 & \dots & 1 \end{pmatrix}$$

5. Сформировать квадратную матрицу порядка $n \times n$ следующим образом:

$$\begin{pmatrix} n & 0 & 0 & \dots & 0 & 0 & 1 \\ n-1 & 1 & 0 & \dots & 0 & 0 & 0 \\ n-2 & n-1 & 0 & \dots & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 2 & 3 & 4 & \dots & n-1 & n & 0 \\ 1 & 2 & 3 & \dots & n-2 & n-1 & n \end{pmatrix}$$

6. Матрицу *M(m, n)* заполнить натуральными числами от 1 до $m \cdot n$ по спирали, начинающейся в левом верхнем углу и закрученной почасовой стрелке.

7. Сформировать квадратную матрицу порядка $n \times n$ следующим образом (n – четное):

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 2 & 0 \\ 0 & 0 & 0 & \dots & 3 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & n-1 & 0 & \dots & 0 & 0 & 0 \\ n & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix}$$

8. Построить квадратную матрицу следующим образом:

$$\begin{pmatrix} \overbrace{1 & 1 & \dots & 1}^{n} & \overbrace{2 & 2 & \dots & 2}^{m} \\ 1 & 1 & \dots & 1 & 2 & 2 & \dots & 2 \\ \vdots & \vdots & \ddots & \dots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 & 2 & 2 & 2 & 2 \\ 3 & 3 & \dots & 3 & 4 & 4 & \dots & 4 \\ 3 & 3 & \dots & 3 & 4 & 4 & \dots & 4 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 3 & 3 & \dots & 3 & 4 & 4 & \dots & 4 \end{pmatrix}$$

9. Ввод элементов матрицы *A(m, n)* осуществляется в произвольном порядке тройками чисел < *i, j, Aij*>. Признаком конца ввода служат три нуля: <*0, 0, 0*>. Проверить корректность такого ввода: все ли элементы введены, нет ли попытки повторного ввода или указания несуществующих координат *i* и *j*. Указание. Разрешается выделение дополнительного (рабочего) массива такой же размерности, что и исходный массив, для хранения признаков «заполненности» элементов матрицы

10. Заполнить массив следующим образом:

$$\begin{pmatrix} n & 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ n-1 & n-1 & 0 & 0 & \dots & 0 & 2 & 1 \\ n-2 & n-2 & n-2 & 0 & \dots & 3 & 2 & 1 \\ n-3 & n-3 & n-3 & n-3 & \dots & 3 & 2 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ n-2 & n-2 & n-2 & n & \dots & 3 & 2 & 1 \\ n-1 & n-1 & n & n & \dots & n & 2 & 1 \\ n & n & n & n & \dots & n & n & 1 \end{pmatrix},$$

n – четное.

ЗАДАНИЕ II:

1. Дана целочисленная матрица размера *M x N*. Строки матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках. Найти количество похожих строк в матрице.

2. Дана целочисленная матрица размера *M x N*. Найти количество ее столбцов, все элементы которых различны.

3. Дана целочисленная матрица размера *M x N*. Вывести номер ее первой строки, содержащей максимальное количество одинаковых элементов.

4. Дана квадратная матрица порядка *M*. Найти набольшее и наименьшее значение сумм элементов матрицы, параллельных главной и побочной диагоналей.

5. Дана квадратная матрица порядка *M*. Найти наибольшее значение из минимальных элементов каждой диагонали, параллельной главной.

6. Дана квадратная матрица порядка *M*. Повернуть ее на 90, 180 и 270 градусов в положительном направлении.

7. Дана матрица размера *M x N*. Вывести количество столбцов, элементы которых монотонно возрастают.

8. Дана матрица размера *M x N*. Найти минимальный среди элементов тех строк, которые упорядочены либо по возрастанию, либо по убыванию.

9. Дана целочисленная матрица размера *M x N*. Найти элемент, являющийся максималь-

ным в своей строке и минимальным в своем столбце.

10. Дана матрица размера *M x N*. Элемент называется локальным минимумом, если он меньше всех окружающих его элементов. Заменить все локальные минимумы данной матрицы на 0.

11. Дана матрица размера *M x N*. Поменять местами ее столбцы так, чтобы их минимальные элементы образовывали возрастающую последовательность.

12. Дана матрица *A* размером *M x N*.. Определить *k* — количество особых элементов массива *A,* считая его элемент особым, если он больше суммы остальных элементов его столбца.

13. Элемент матрицы назовем седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот, является наибольшим в своей строке и наименьшим в своем столбце. Для заданной целой матрицы размером *M x N* напечатать индексы всех ее седловых точек.

14. Задана матрица размером *M x N*. Найти максимальный по модулю элемент матрицы. Переставить строки и столбцы матрицы таким образом, чтобы максимальный по модулю элемент был расположен на пересечении *i* – й строки и *k* – то столбца.

15. Дана целая квадратная матрица *n* - го порядка. Определить, является ли она магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы. Получить транспонированную матрицу. Сформировать одномерный массив из ее диагональных элементов. Найти след матрицы, суммируя элементы одномерного массива.

Обработка результатов измерений:

Пример 1. Даны три одномерных массива *A, B, C*, состоящие из *N* вещественных значений. Сформировать массив *K* такой же длины, элементы которого вычисляются по формуле $K[i] = \frac{A[i] + B[i]}{1 + C[i]}$. Если $1 + C[i] = 0$, то $K[i]$ принять равным нулю. Заполнение массивов *A, B, C* осуществить с помощью датчика случайных чисел.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
main()
{
int i,N;
double *A,*B,*C;
cout<<"Введите N=";
cin>>N;
A=new double [N];
B=new double [N];
C=new double [N];
for(i=0; i<N; i++)
  A[i]=double(random(101)-50)/10;
for(i=0; i<N; i++)
  B[i]=double(random(101)-50)/10;
for(i=0; i<N; i++)
  C[i]=double(random(101)-50)/10;
cout<<"Массив A:"<<endl;
for(i=0; i<N; i++)
  cout<<"A["<<i<<"]="
     <<A[i]<<endl;
cout<<"Массив B:"<<endl;
for(i=0; i<N; i++)
  cout<<"B["<<i<<"]="
     <<B[i]<<endl;
cout<<"Массив C:"<<endl;
```

```
for(i=0; i<N; i++)
  cout<<"C["<<i<<"]="
     <<C[i]<<endl;
double *K=new double[N];
for(i=0; i<N; i++)
   if (1+C[i]) K[i]=(A[i]+B[i])/(1+C[i]);
     else K[i]=0;
cout<<"Массив K:"<<endl;
for(i=0; i<N; i++)
  cout<<"K["<<i<<"]="
     <<K[i]<<endl;
delete []A;
delete []B;
delete []C;
delete []K;
getch();
}
```

Пример 2. Сформировать квадратную матрицу порядка N по правилу $A[i][j] = \sin\left(\frac{i^2 - j^2}{N}\right)$ и подсчитать количество положительных элементов в ней.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
main()
{
int N;
cout<<"Введите размерность матрицы"<<endl;
cout<<"N=";
cin>>N;
double** A=new double* [N];
int i,j;
for (i=0; i<N; i++)
  A[i]=new double [N];
for (i=0;i<N; i++)
 for (j=0; j<N; j++)
    A[i][j]=sin(double(i*i-j*j)/N);
for (i=0;i<N; i++){
 for (j=0; j<N; j++)
    cout <<A[i][j]<<" ";
 cout<<endl;
              }
int kol=0;
for (i=0;i<N; i++)
 for (j=0; j<N; j++)
    if(A[i][j]>0) kol++;
cout<<"Положительных значений = "<<kol;
getch();
}
```

Пример 3. Дан двумерный массив натуральных чисел размером ($N \times M$). Не создавая дополнительных массивов, определить в каждой строке двумерного массива, какой из элементов повторяется наибольшее число раз, и найти его порядковый номер первого появления в строке. Заполнение двумерного массива осуществить с помощью ввода с клавиатуры.

```
#include<iostream.h>
```

```
#include<stdlib.h>
#include<conio.h>
main()
{
int N,M;
cout<<"N=";
cin>>N;
cout<<"M=";
cin>>M;
int **A=new int* [N];
int i,j;
for(int i=0; i<N; i++)
  A[i]=new int [M];
for(i=0;i<N;i++)
  {cout<<"Введите "<<i+1<<" строку:"<<endl;
   for(j=0;j<M;j++)
     {cout<<"A["<<i<<"]["<<j<<"]=";
      cin>>A[i][j];
      }
   }
clrscr();
for(i=0;i<N;i++)
  {cout<<endl;
   for(j=0;j<M;j++)
     cout<<A[i][j]<<" ";
   }
cout<<endl;
int index, count,count_max,m;
for(i=0;i<N;i++)
    {count_max=0;
     index=0;
     for (j=0; j<M; j++)
      {count=0;
       for (m=j+1; m<M; m++)
         if (A[i][j]==A[i][m]) count++;
       if(count>count_max) {count_max=count;
                           index=j;
                           }
       }
      cout<<"В "<<i+1<<"строке "
          <<index<<" элемент повторяется "
          <<count_max+1<<" раз"<<endl;
     }
getch();
}
```

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Что такое указатели и для чего они применяются?
2. Виды указателей.
3. Операции над указателями.
4. Что такое ссылки и для чего они применяются?
5. Одномерные динамические массивы.
6. Многомерные динамические массивы.

Лабораторная работа № 6

Разработка сетевого программного приложения в клиент-серверной архитектуре

Цель работы:

Закрепить практические навыки работы с системой программирования языка С++ на примере реализации алгоритмов над массивами символов.

Порядок выполнения:

Массивы символов

В отличие от других языков программирования, стандарт языка Си не содержит специального типа строк. Строки представляются одномерными массивами символов, т.е. массивами элементов типа char. Но если длина массивов чисел имеет определенный значимый характер, то длина строк постоянно меняется и не всегда, получается, иметь информацию об ее длине. Поэтому, для определения длины строки был введен специальный нулевой символ ‘\0’, который обозначает конец строки в массиве в не зависимости от начальной выделенной для него памяти. В этом случае, число хранимых символов в строке всегда на 1 больше числа значащих символов. Если нулевой символ отсутствует, то обработка строки будет продолжаться до тех пор, пока в памяти не встретится случайный нулевой символ.

Таким образом, строка в языке Си рассматривается, как массив символов, оканчивающийся нулевым символом (`\0`), в не зависимости от длины.

Строка может быть объявлена либо как просто массив символов

```
char St[] = “строка”;
```

либо как переменная указатель типа char*

```
char *Sp = “строка”;
```

Каждое из этих приведенных эквивалентных объявлений присваивает строковой переменной начальное значение “строка”, а длина при этом определяется автоматически компилятором и равна 7 элементам, содержащих соответственно символы: ‘с’, ‘т’, ‘р’, ‘о’, ‘к’, ‘а’ и '\0'. Можно объявлять строковые переменные заданной длины. Например, операторы

```
char Str[100];
char *Stp=new char [100];
```

объявляют переменные, которые могут содержать строки до 99 значащих символов плюс заключительный нулевой символ.

Доступ к отдельным символам строки осуществляется по индексам, начинающимся с нуля. Например, St[0] и Sp[0] – первые символы объявленных выше строк, St[5] и Sp[5] – последние, а St[6] и Sp[6] содержат нулевые символы.

Так как, строка представляется массивом, то доступ к ней осуществляется через указатель или указатель - константу, который является адресом ее первого символа, поэтому при работе со строками необходимо учитывать общие правила работы с указателями и с масси-

вами, так запись

Str=“переменная”;

или

Str=St;

вызовет ошибку компилятора, потому что, во первых – нельзя присваивать значение указателю – константе, а во вторых – St, Str, Sp и Stp содержат лишь адрес на участок памяти по которому располагается набор символов, а не сам этот набор. Присваивание литералы – константы возможно лишь только при объявлении строки.

Так же, следует помнить, что резервирование памяти полностью лежит на программисте, а при работе с указателем на строку этот указатель должен указывать на выделенный ранее участок памяти.

Ввод – вывод массивов символов

Имеется несколько возможностей для ввода и вывода строк, рассмотрим только основные из них.

С помощью форматируемого ввода-вывода. Форматируемый ввод и вывод можно осуществить с помощью функций scanf и printf из стандартной библиотеки stdio.h, которые были рассмотрены в лабораторной работе №1. Эти функции имеют существенный недостаток, так как останавливают ввод и вывод строки, не только если встретится нулевой символ, но и если встретится пробел. Поэтому являются не удобными при использовании.

С помощью неформатируемого ввода-вывода. Такой ввод и вывод осуществляется функциями gets и puts без указания формата из библиотеки stdio.h и имеют следующие прототипы:

```
char *gets(char *string);
int *puts(char *string);
```

Функция gets осуществляет ввод набора символов с клавиатуры в строку string, до тех пор, пока не будет нажата клавиша Enter. Возвращает адрес начала введенной строки или значение NULL, если произошла ошибка при вводе. Следует помнить, что функция gets не осуществляет выделение памяти под введенный набор символов и не отслеживает размер зарезервированной под него памяти.

Функция puts выводит строку string на монитор и переводит курсор на новую строку. Возвращает последний выводимый символ, которым обычно является символ новой строки '\n'.

Следующий пример иллюстрирует работу данных функций

```
#include <stdio.h>
#include <conio.h>
main()
{
char A[100];
char *B=new char [100];
char C[]="Ввод строки:";
char D[]="Вывод строки:";
puts(C);
gets(A);
puts(C);
gets(B);
puts(D);
puts(A);
puts(D);
puts(B);
getch();
}
```

С помощью потокового ввода-вывода. Наиболее распространенным способом ввода и вывода массивов символов является потоковый ввод-вывод, осуществляемый стандартной библиотекой классов, подключаемой к программе с помощью заголовочного файла iostream.h. Некоторые элементы потокового ввода – вывода были рассмотрены в лаборатор-

ной работе №1.

Если необходимо прочитать из входного потока (с клавиатуры) строку символов, содержащую пробелы, то можно воспользоваться перегруженными компонентными функциями get и getline объекта cin, которые имеют следующие прототипы

istream& get(char *string, int max_len, char='\n');

istream& getline(char *string, int max_len, char='\n');

Каждая из этих функций выполняет чтение последовательности символов с клавиатуры и перенос их в символьный массив string, задаваемый первым параметром. Второй параметр max_len определяет максимально допустимое количество вводимых символов. Третий параметр определяет ограничивающий символ при появлении, которого следует завершить ввод. Второй и третий параметры могут быть опущены, в этом случае размер вводимых символов не ограничен, а конец ввода будет осуществляться при нажатии клавиши Enter. Если из входного потока извлечены ровно max_len – 1 символов, а ограничивающий символ не встретился, то нулевой символ помещается после введенной последовательности автоматически, при этом массив, в который выполняется чтение, должен иметь длину не менее max_len символов. Различие между get и getline заключается в том, что в последней функции в конец строки помещается ограничивающий символ, который прервал ввод.

Потоковый вывод массива символов можно осуществить стандартным методом, а именно с помощью стандартного потока вывода на экран

cout << string;

Ниже представлен пример потокового ввода – вывода строк, подобный примеру, который был представлен выше с помощью функций gets и puts

```
#include <iostream.h>
#include <conio.h>
main()
{
char A[100];
char *B=new char [100];
char C[]="Ввод строки:";
char D[]="Вывод строки:";
cout << C;
cin.getline(A,100);
cout << C;
cin.getline(B,100);
cout << D << endl;
cout << A << endl;
cout << D << endl;
cout << B << endl;
getch();}
```

Также библиотека классов iostream.h содержит множество других удобных и интересных способов ввода и вывода, рассмотрении которых лежит вне данного учебного пособия.

Варианты задания для выполнения лабораторной работы:

ЗАДАНИЕ I:

1. Дана строка, содержащая текст. Найти длину самого короткого и самого длинного слова.
2. Дана строка, содержащая текст, заканчивающийся точкой. Вывести на экран составляющие ее слова из трех букв.
3. Дана строка, содержащая текст. Определить, сколько в нем слов, заканчивающихся гласными буквами.
4. Дана строка, содержащая текст. Указать те слова, которые содержат хотя бы одну букву из звонких согласных.
5. Дана строка, содержащая текст. Найти в ней те слова, которые начинаются и оканчиваются одной и той же буквой.

6. Дана строка, содержащая текст. Вывести, в скольких словах этого текста встречаются гласные буквы. Ответ должен приводиться в грамматически правильной форме, например «а — 5 слов», «о — 3 слово» и т. д.
7. Дана строка, содержащая текст. Найти слова в тексте, которые можно составить из первого и последнего слова (буквы можно использовать не более одного раза).
8. Дана строка, содержащая текст в виде чисел, записанных в римской нотации, и разделенных знаками препинания. Найти самое наибольшее число.
9. Дана строка, содержащих текст из двух предложений. Найти слова, которые встречаются в каждом из двух заданных предложений.
10. Дана строка, содержащая текст. Вывести слова в алфавитном порядке.
11. Дана строка, содержащая текст. Найти те буквы, которые встречаются в каждом слове и только один раз.
12. Дана строка, содержащая текст. Часто встречающаяся ошибка начинающих наборщиков — дважды записанное подряд слово. Обнаружить в тексте такие ошибки.
13. Дана строка, содержащая текст. Найти слова в тексте, которые являются анаграммами с первым или последним словом. Анаграммы – пара слов, при прочтении каждого из которых в обратном направлении образуется другое слово пары, например, (ПОЛК, КЛОП); (БАР, РАБ).
14. Дана строка, содержащая текст. В данном тексте найти самое наибольшее слово из тех, длина которых больше двух. (Пример: Варвар> Академия).
15. Один из способов идентификации автора литературного произведения — подсчет частоты вхождения отдельных слов. В заданном тексте найти наиболее часто встречающееся слово.

ЗАДАНИЕ II:

1. Дана строка, содержащая текст. Преобразовать строку так, чтобы все буквы в словах были отсортированы по алфавиту.
2. Дана строка, содержащая текст. Преобразовать строку так, чтобы все слова в ней стали идентификаторами, слова состоящие только из цифр и слова, которые начинаются с цифры - удалить.
3. Дана строка, содержащая текст. Преобразовать строку таким образом, чтобы в ее начале были записаны слова, содержащие только цифры, потом слова, содержащие только буквы, а затем слова, которые содержат и буквы и цифры.
4. Дана строка, содержащая текст. Преобразовать строку таким образом, чтобы все слова в ней были напечатаны наоборот.
5. Дана строка, содержащая текст. Отредактировать заданное предложение, удаляя из него те слова, которые встречаются в предложении несколько раз.
6. Дана строка, содержащая текст. Отредактировать заданное предложение, удаляя из него все слова с нечетными номерами и переворачивая слова с четными номерами.
7. Дана строка, содержащая текст. Удалить из нее каждое слово нечетной длины.
8. Дана строка, содержащая текст. В каждом слове переставить его первую букву на место последней. При этом вторую, третью,..., последнюю буквы сдвинуть влево на одну позицию, то есть осуществить циклический сдвиг влево.
9. Перенос. Примем следующие правила переноса русских слов: в каждой из разделяемых частей должно быть более одной буквы, из которых хотя бы одна — гласная;
   - нельзя разделять согласную и следующую за ней гласную;
   - буквы И, Ь, Ъ считать согласными, но перенос последних допустим.

Преобразовать текст таким образом, что в каждом из вводимых слов были поставлены все возможные знаки переноса, например: СЕ-ЛЬ-С-КО-ХО-ЗЯЙ-С-Т-ВЕ-Н-НАЯ. Строчные и прописные буквы считать неразличимыми.

10. Дана строка, содержащая текст. Переставить слова в тексте так, чтобы каждое следующее слово начиналось с той буквы, на которую закончилось предыдущее. Первое слово оставить на месте.
11. Дана строка, содержащая текст. Расставить слова в соответствии с алфавитом.
12. Дана строка, содержащая текст. Изменить порядок букв в словах на противополож-

ный.

13. Дана строка, содержащая текст. Преобразовать строку, содержащую эти же слова, но расположенные в обратном порядке.
14. Дана строка, содержащая текст. Преобразовать каждое слово в строке, удалив из него все последующие вхождения первых двух букв этого слова.
15. Дана строка, содержащая текст и число k (0 < k < 10). Зашифровать строку, выполнив циклическую замену каждой буквы на букву того же регистра, расположенную в алфавите на k-й позиции после шифруемой буквы (например, для k = 2 "А" перейдет в "В", "а" — в "в", "Б" — в "Г", "я" — в "б" и т.д.). Букву "ё" в алфавите не учитывать, знаки препинания и пробелы не изменять.

Обработка результатов измерений:

Пример 1. Дана строка, содержащая текст. Определить слова, в которых количество согласных меньше, чем в предыдущем слове, а количество гласных больше, чем в последующем.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
main()
{
char *str=new char [100];
char *work=new char [100];
char *next,*current,*previous;
int gl_next,gl_current,sg_current,sg_previous;
int i,j;
int log;
char gl[]={128,133,136,142,147,155,157,158,159,
          160,165,168,174,227,235,237,238,239,0};
char *s;
cin.get(str,100);
s=str;
do
 {
 while (s&&(*s==' '||*s==','||*s=='.'))
    s++;
 next=NULL;
 current=NULL;
 previous=NULL;
 strcpy(work,s);
 previous=strtok(work,". ,");
 if (previous)current=strtok(NULL,". ,");
 if (current)next=strtok(NULL,". ,");
 if (previous&&current&&next)
   {
    sg_previous=0;
    sg_current=0;
    gl_current=0;
    gl_next=0;
    i=0;
    while (previous[i])
      {log=(unsigned char) previous[i]>=128
           &&
          (unsigned char)previous[i]<=175;
       log=log||((unsigned char)previous[i]>=224
```

```
                &&
               (unsigned char)previous[i]<=239);
       if(log)
         {
          for(j=0;j<strlen(gl);j++)
              if (previous[i]==gl[j]) break;
          if(j==strlen(gl)) sg_previous++;
         }
       i++;
       }
      i=0;
      while (current[i])
       {log=(unsigned char)current[i]>=128
            &&
           (unsigned char)current[i]<=175;
       log=log||((unsigned char)current[i]>=224
                &&
               (unsigned char)current[i]<=239);
       if(log)
         {
          for(j=0;j<strlen(gl);j++)
              if (current[i]==gl[j])
                {
                 gl_current++;
                 break;
                }
          if(j==strlen(gl)) sg_current++;
         }
        i++;
       }
       i=0;
      while (next[i])
      {log=(unsigned char)next[i]>=128
           &&
          (unsigned char)next[i]<=175;
       log=log||((unsigned char)next[i]>=224
                &&
               (unsigned char)next[i]<=239);
       if(log)
          for(j=0;j<strlen(gl);j++)
              if (next[i]==gl[j])
                {
                 gl_next++;
                 break;
                }
        i++;
       }
       log=sg_previous>sg_current&&gl_current>gl_next;
if (log) cout<<"Найденное слово: "<<current<<endl;
    }
 s=strpbrk(s,". ,");
 }
while(next);
getch();
}
```

Форма отчётности:

Отчет должен содержать задание и результаты выполненной простейшей программы на языке Си/Си++ в режиме диалога, а так же диагностические сообщения компилятора об ошибках при выполнении линейных программ.

Задания для самостоятельной работы:

1. Проработать рекомендуемые источники, основную и дополнительную литературу по изучаемому вопросу с целью углубления, систематизации и расширения полученных знаний.
2. Письменно ответить на контрольные вопросы для самопроверки.

Рекомендации по выполнению заданий и подготовке к лабораторной работе:

Проработка основной и дополнительной литературы, терминов, сведений, требующихся для запоминания и являющихся основополагающими в данной теме. Проработка материалов по изучаемому вопросу, с использованием рекомендуемых ресурсов информационно-телекоммуникационной сети «Интернет».

Контрольные вопросы для самопроверки:

1. Массивы символов.
2. Описать процесс ввод – вывод массивов символов.

9.2 Методические указания по выполнению курсовой работы (КР)

Рекомендуемые источники литературы, необходимые при курсовой работы указаны в п.7 (дополнительная [4,10]).

Порядок выполнения курсовой работы.

Для выполнения курсовой работы обучающимся выдаётся индивидуальное задание по которому обучающийся должен: выбрать форму представления исходных данных и результатов, разработать и обосновать алгоритмы для решения задачи, разработать пользовательские классы (компонентные данные и компонентные функции, реализующие алгоритмы), разработать пользовательский интерфейс для ввода и получения информации, провести отладку и тестирование программного приложения, оформить документацию.

Готовая курсовая работа сдается преподавателю на проверку за 2 недели до начала экзаменационной сессии. Результатом проверки могут быть:

- «допущен к защите»;
- «допущен к защите после доработки по замечаниям»;
- «не допущен к защите».

Если после проверки курсовой работы рекомендован преподавателем к защите, то следует подготовиться к его защите.

В случае выявления при проверке ошибок и неточностей, обучающиеся допускается к защите курсовой работы только после их устранения.

В последнем случае требуется переделать курсовую работу в соответствии с предъявляемыми требованиями. Если курсовой работы не рекомендована преподавателем к защите, то после переработки работа вновь сдается на проверку.

Без защиты курсовой работы обучающиеся не допускается к сдаче экзамена по дисциплине.

Защита курсовой работы производится в часы, определенные в соответствии с расписанием занятий.

На защите курсовой работы обучающиеся в краткой форме излагает основные результаты, полученные в ходе его выполнения и практическую значимость выполненной работы, отвечает на возникшие в ходе защиты вопросы.

Рекомендации по выполнению курсовой работы.

Цель работы: Закрепление и углубление теоретических знаний студентами по наиболее важным вопросам дисциплины, совершенствование у них навыков объектно-ориентированного проектирования программного обеспечения, умения проектировать и программировать дружественный интерфейс, взаимодействовать с базами данных, проверка способностей студентов квалифицированно применять полученные ими теоретические знания и практические навыки при решении задач.

Пояснительная записка должна содержать титульный лист, задание, описание разработанных алгоритмов и приложения, блок схемы и листинг программы.

Исходные данные: Курсовая работа по дисциплине выполняется индивидуально каждым студентом в соответствии с выданным преподавателем заданием. Курсовая работа выполняется в среде Borland JBuilder X Enterprise. При выполнении работы обучающийся должен: выбрать форму представления исходных данных и результатов, разработать и обосновать алгоритмы для решения задачи, разработать пользовательские классы (компонентные данные и компонентные функции, реализующие алгоритмы), разработать пользовательский интерфейс для ввода и получения информации, провести отладку и тестирование программного приложения, оформить документацию.

В заключении необходимо провести анализ выполненной работы. Сделать выводы по работе, где конкретизируются полученные результаты в ходе разработки программы.

10. ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, ИСПОЛЬЗУЕМЫХ ПРИ ОСУЩЕСТВЛЕНИИ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

1. Microsoft Imagine Premium
2. ОС Windows 7 Professional
3. Microsoft Office 2007 Russian Academic OPEN No Level
4. Антивирусное программное обеспечение Kaspersky Security.

11. ОПИСАНИЕ МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЙ БАЗЫ, НЕОБХОДИМОЙ ДЛЯ ОСУЩЕСТВЛЕНИЯ ОБРАЗОВАТЕЛЬНОГО ПРОЦЕССА ПО ДИСЦИПЛИНЕ

Вид занятия	*Наименование аудитории*	*Перечень основного оборудования*	*№ ЛР, Лк, ПЗ*
1	2	3	4
ЛР	специализированная аудитория 1353	ПЭВМ на базе процессора AMD Athlon 64 * 2 Dual / Intel Core 2.10 ГГц / 1 Гб ОЗУ / HDD 250 / Монитор ЖК 17 “Samsung” – 5 шт.; ПЭВМ на базе процессора Celeron 2.26 / HDD 80 / 256 Mb ОЗУ – 2 шт.; ПЭВМ на базе процессора Celeron / 700 / HDD 40 / 128 Mb ОЗУ – 4 шт.; МФУ Canon Laser Base MF – 3228, A4 – 1 шт.	№№ 1÷6
КР	Лекционный кабинет/ дисплейный класс	Оборудование Интерактивная доска SMART Board 680I, проектор Casio XJ-UT310WN; 17-ПК: CPU 5000/RAM 2Gb/HDD; Монитор TFT 19 LG1953S-SF; Принтер: HP LaserJet P2015n; Сканер: Canon LiDE 220	-
СР	ЧЗЗ	Оборудование 15 ПК- CPU 5000/RAM 2Gb/HDD (Монитор TFT 19 LG 1953S-SF);принтер HP LaserJet P3005	-

Приложение 1

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

1. Описание фонда оценочных средств (паспорт)

№ компетенции	Элемент компетенции	Раздел	Тема	ФОС
ПК-9	умение проводить расчёты по проекту сетей, сооружений и средств инфокоммукаций в соответствии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ	**1.** Языки гипертекстовой разметки документов	1.1 Языки гипертекстовой разметки документов (HTML, DHTML, XML, XSL). Клиентские скрипты (JavaScript, VbScript). 1.2 Введение в язык разметки HTML, основные элементы и структура HTML, разработка HTML-документов. 1.3 DHTML- назначение и особенности его использования. Понятие и особенности применения клиентских скриптов. 1.4 Язык описания стилей отображения XML документов XSL и особенности его применения.	Экзаменационные вопросы 1-6
		2. Язык Java. Обзор базовых конструкций и основных элементов языка	2.1 Особенности форматов основных выражений. Особенности объявления и использования типов данных. Операторы языка. 2.2 Динамическая инициализация, область действия и время жизни переменных. Преобразование и приведение типов, расширение типов. 2.3 Особенности объявления структурных типов. Особенности разработки архитектуры приложений в Java.	Экзаменационные вопросы 7-13
ОПК-4	способность иметь навыки самостоятельной работы на компьютере и в компьютерных сетях, осуществлять компьютерное моделирование устройств, систем и процессов с использованием универсальных	**3.** Объектно-ориентированное программирование на языке Java	3.1Классы и объекты. Динамическая инициализация объектов. Методы класса, конструкторы, параметризация методов. Использование ключевого слова this. Перегрузка и переопределение методов. 3.2Ограничения и управления доступом. Вложенные и внутренние классы. Основы наследования. Создание многоуровневой иерархии. 3.3Переопределение методов и их применение. Интерфейсы. Определение, реализация и применение интерфейсов.	Экзаменационные вопросы 14-24
		4. Основы ввода/вывода. Работа со строковыми данными	4.1 Чтение консольного ввода. Чтение символов. Чтение строк. Запись консольного вывода. Чтение и запись файлов. 4.2 Классы и интерфейсы ввода/вывода Java. Буферизированные байтовые потоки. Символьные потоки. Классы и интерфейсы потоков ввода/вывода Java. 4.3 Обработка строк. String-конструкторы. Специальные строковые операции. Преобразо-	Экзаменационные вопросы 25-37

	пакетов прикладных компьютерных программ		вание и изменение строк. Методы для работы со строками.	
		5. Методы и средства обработки исключений	5.1 Необходимость обработки исключительных ситуаций. Основные принципы обработки исключений. Типы исключений. 5.2 Использование операторов try и catch. Оператор throw. Блок finally. Встроенные исключения Java. Создание собственных подклассов исключений.	Экзаменационные вопросы 38-44
		6. Средства для организации работы в сети	6.1 Основы работы в сети. Клиент-сервер. Proxy-серверы. Адресация Internet. Сетевые классы и интерфейсы. Класс InetAddress. Обзор сокетов. Зарезервированные сокеты. Сокеты TCP/IP клиентов. Сокеты TCP/IP серверов. 6.2 Дейтаграммы. Использование URL. Основные классы и интерфейсы реализации сетевого взаимодействия. Распределенная обработка данных. Вызов удаленных методов (RMI).	Экзаменационные вопросы 45-54
		7. Многопоточное программирование	7.1 Поточная модель Java. Класс Thread и интерфейс Runnable. Главный поток. Создание потока. Реализация интерфейса Runnable. 7.2 Создание множественных потоков. Приоритеты потоков. Синхронизация и передача сообщений. Использование синхронизированных методов. 7.3 Межпоточные связи. Блокировка. Приостановка, возобновление и остановка потоков.	Экзаменационные вопросы 55-63
		8. Апплеты и события	8.1 Основы апплетов. Класс Applet. Архитектура апплета. Инициализация и завершение апплета. Простые методы отображения апплетов. Требование перерисовки. 8.2 Пересылка параметров в апплеты. Основные методы класса Applet. Вывод в консоль. 8.3 Обработка событий. Модель делегирования событий. Источники событий. Классы событий. Обработка событий мыши. Обработка событий клавиатуры.	Экзаменационные вопросы 64-73
		9. Разработка пользовательского интерфейса в Java	9.1 Основы оконной графики. Класс Component. Класс Container. Класс Panel. Класс Window. Класс Frame. Работа с фреймовыми окнами. Создание фрейм-окна в апплете. Обработка событий фрейм-окна. Отображение информации в окне. 9.2 Работа с графикой. Работа со шрифтами. Управление текстовым выводом. Использование элементов управления, менеджеров компоновки и меню AWT. 9.3 Элементы управления. Основные понятия. Добавление и удаление элементов управления. Реагирование на элементы управления. Понятие менеджера компоновки. Работа с меню и диалоговыми окнами.	Экзаменационные вопросы 74-81

2. Экзаменационные вопросы

№ п/п	Компетенция		ЭКЗАМЕНАЦИОННЫЕ ВОПРОСЫ	№ и наименование раздела
	Код	Определение		
1	2	3	4	5
1	ПК-9	умение проводить расчёты по проекту сетей, сооружений и средств инфокоммуникаций в соответствии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ	1. Языки гипертекстовой разметки документов (HTML, DHTML, XML, XSL). 2. Клиентские скрипты (JavaScript, VbScript). 3. Введение в язык разметки HTML, основные элементы и структура HTML, разработка HTML-документов. 4. DHTML- назначение и особенности его использования. 5. Понятие и особенности применения клиентских скриптов. 6. Язык описания стилей отображения XML документов XSL и особенности его применения.	**1.** Языки гипертекстовой разметки документов
			1. Особенности форматов основных выражений. **2.** Особенности объявления и использования типов данных. **3.** Операторы языка. **4.** Динамическая инициализация, область действия и время жизни переменных. **5.** Преобразование и приведение типов, расширение типов. **6.** Особенности объявления структурных типов. **7.** Особенности разработки архитектуры приложений в Java.	**2.** Язык Java. Обзор базовых конструкций и основных элементов языка
2	ОПК-4	способность иметь навыки самостоятельной работы на компьютере и в компьютерных сетях, осуществлять компьютерное моделирование устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ	**8.** Классы и объекты. **9.** Динамическая инициализация объектов. **10.** Методы класса, конструкторы, параметризация методов. **11.** Использование ключевого слова this. **12.** Перегрузка и переопределение методов. **13.** Ограничения и управления доступом. **14.** Вложенные и внутренние классы. Основы наследования. **15.** Создание многоуровневой иерархии. **16.** Переопределение методов и их применение. **17.** Интерфейсы. **18.** Определение, реализация и применение интерфейсов.	**3.** Объектно-ориентированное программирование на языке Java
			19. Чтение консольного ввода. **20.** Чтение символов. **21.** Чтение строк. **22.** Запись консольного вывода. **23.** Чтение и запись файлов. **24.** Классы и интерфейсы ввода/вывода Java. **25.** Буферизированные байтовые потоки. **26.** Символьные потоки.	**4.** Основы ввода/вывода. Работа со строковыми данными

			27. Классы и интерфейсы потоков ввода/вывода Java. **28.** Обработка строк. String-конструкторы. **29.** Специальные строковые операции. **30.** Преобразование и изменение строк. **31.** Методы для работы со строками.	
			32. Необходимость обработки исключительных ситуаций. **33.** Основные принципы обработки исключений. **34.** Типы исключений. **35.** Использование операторов try и catch. **36.** Оператор throw. Блок finally. **37.** Встроенные исключения Java. **38.** Создание собственных подклассов исключений.	**5.** Методы и средства обработки исключений
			39. Основы работы в сети. Клиент-сервер. Proxy-серверы. **40.** Адресация Internet. **41.** Сетевые классы и интерфейсы. Класс InetAddress. **42.** Обзор сокетов. Зарезервированные сокеты. **43.** Сокеты TCP/IP клиентов. **44.** Сокеты TCP/IP серверов. **45.** Дейтаграммы. Использование URL. **46.** Основные классы и интерфейсы реализации сетевого взаимодействия. **47.** Распределенная обработка данных. **48.** Вызов удаленных методов (RMI).	**6.** Средства для организации работы в сети
			49. Поточная модель Java. **50.** Класс Thread и интерфейс Runnable. **51.** Главный поток. Создание потока. **52.** Реализация интерфейса Runnable. **53.** Создание множественных потоков. **54.** Приоритеты потоков. Синхронизация и передача сообщений. **55.** Использование синхронизированных методов. **56.** Межпоточные связи. **57.** Блокировка. Приостановка, возобновление и остановка потоков.	**7.** Многопоточное программирование
			58. Основы апплетов. Класс Applet. **59.** Архитектура апплета. Инициализация и завершение апплета. **60.** Простые методы отображения апплетов. **61.** Требование перерисовки. **62.** Пересылка параметров в апплеты. **63.** Основные методы класса Applet. Вывод в консоль. **64.** Обработка событий. Модель делегирования событий. **65.** Источники событий. Классы событий. **66.** Обработка событий мыши. **67.** Обработка событий клавиатуры.	**8.** Апплеты и события

			68. Основы оконной графики. Класс Component. Класс Container. Класс Panel. Класс Window. Класс Frame. **69.** Работа с фреймовыми окнами. Создание фрейм-окна в апплете. **70.** Обработка событий фрейм-окна. Отображение информации в окне. **71.** Работа с графикой. Работа со шрифтами. **72.** Управление текстовым выводом. Использование элементов управления, менеджеров компоновки и меню AWT. **73.** Элементы управления. Основные понятия. **74.** Добавление и удаление элементов управления. Реагирование на элементы управления. **75.** Понятие менеджера компоновки. Работа с меню и диалоговыми окнами.	**9.** Разработка пользовательского интерфейса в Java

3. Описание показателей и критериев оценивания компетенций

Показатели	Оценка	Критерии
знать: (ОПК-4) - основные термины и процессы компьютерного моделирования. (ПК-9) - основные типы сигналов, используемых в телекоммуникационных системах. **уметь:** (ОПК-4) - самостоятельно работать на компьютере и в компьютерных сетях. (ПК-9) - проводить расчёты по проекту сетей, сооружений и средств инфокоммуникаций в соответствии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ. **владеть:** (ОПК-4) - навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ. (ПК-9) - техникой инженерной и компьютерной графики (ввод, вывод, отображение, преобразование и редактирование графических объектов на компьютере).	**отлично**	Оценка «отлично» выставляется в случае, если обучающийся демонстрирует: - знания: основных терминов и процессов компьютерного моделирования. - умения: самостоятельно работать на компьютере и в компьютерных сетях. - владение: навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ.
	хорошо	Оценка «хорошо» выставляется в случае, если обучающийся демонстрирует: - недостаточно полное знание: основных терминов и процессов компьютерного моделирования. - недостаточно полное умение: самостоятельно работать на компьютере и в компьютерных сетях. - недостаточно полное владение: методами самоорганизации; навыками применения навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ.
	удовлетворительно	Оценка «удовлетворительно» выставляется в случае, если обучающийся демонстрирует: - частичное знание: основных терминов и процессов компьютерного моделирования. - частичное умение: самостоятельно работать на компьютере и в компьютерных сетях. - частичное владение: методами самоорганизации; навыками применения навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ.
	неудовлетворительно	Оценка «неудовлетворительно» выставляется в случае, если обучающийся демонстрирует: - существенные пробелы в знание: основных терминов и процессов компьютерного моделирования. - принципиальные ошибки в умение: самостоятельно работать на компьютере и в компьютерных сетях. - невозможность владения: методами самоорганизации; навыками применения навыками компьютерного моделирования устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ

4. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и опыта деятельности

Дисциплина программирование сетевых приложений направлена на формирование у обучающихся знаний и навыков по использованию современных технологий и методов разработки программных систем для решения практических задач с использованием современных инструментальных средств, необходимых в дальнейшем, при проектировании и эксплу-

атации инфокоммуникационных систем.

Изучение дисциплины технологии и языки программирования предусматривает:

- лекции,
- лабораторные работы,
- курсовую работу,
- самостоятельную работу,
- экзамен.

В ходе освоения *раздела 1* «Языки гипертекстовой разметки документов» студенты должны уяснить: языки гипертекстовой разметки документов (HTML, DHTML, XML, XSL); клиентские скрипты (JavaScript, VbScript); введение в язык разметки HTML, основные элементы и структура HTML, разработка HTML-документов; DHTML- назначение и особенности его использования; понятие и особенности применения клиентских скриптов; язык описания стилей отображения XML документов XSL и особенности его применения.

В ходе освоения *раздела 2* «Язык Java. Обзор базовых конструкций и основных элементов языка» студенты должны уяснить: особенности форматов основных выражений; особенности объявления и использования типов данных; операторы языка; динамическая инициализация, область действия и время жизни переменных; преобразование и приведение типов, расширение типов; особенности объявления структурных типов; особенности разработки архитектуры приложений в Java.

В ходе освоения *раздела 3* «Объектно-ориентированное программирование на языке Java» студенты должны уяснить: классы и объекты; динамическая инициализация объектов; методы класса, конструкторы, параметризация методов; использование ключевого слова this; перегрузка и переопределение методов; ограничения и управления доступом; вложенные и внутренние классы; основы наследования; создание многоуровневой иерархии; переопределение методов и их применение; интерфейсы; определение, реализация и применение интерфейсов.

В ходе освоения *раздела 4* «Основы ввода/вывода. Работа со строковыми данными» студенты должны уяснить: чтение консольного ввода. Чтение символов. Чтение строк. Запись консольного вывода. Чтение и запись файлов. Классы и интерфейсы ввода/вывода Java. Буферизированные байтовые потоки. Символьные потоки. Классы и интерфейсы потоков ввода/вывода Java. Обработка строк. String-конструкторы. Специальные строковые операции. Преобразование и изменение строк. Методы для работы со строками.

В ходе освоения *раздела 5* «Методы и средства обработки исключений» студенты должны уяснить: Необходимость обработки исключительных ситуаций. Основные принципы обработки исключений. Типы исключений. Использование операторов try и catch. Оператор throw. Блок finally. Встроенные исключения Java. Создание собственных подклассов исключений.

В ходе освоения *раздела 6* «Средства для организации работы в сети» студенты должны уяснить: Основы работы в сети. Клиент-сервер. Proxy-серверы. Адресация Internet. Сетевые классы и интерфейсы. Класс InetAddress. Обзор сокетов. Зарезервированные сокеты. Сокеты TCP/IP клиентов. Сокеты TCP/IP серверов. Дейтаграммы. Использование URL. Основные классы и интерфейсы реализации сетевого взаимодействия. Распределенная обработка данных. Вызов удаленных методов (RMI).

В ходе освоения *раздела 7* «Многопоточное программирование» студенты должны уяснить: Поточная модель Java. Класс Thread и интерфейс Runnable. Главный поток. Создание потока. Реализация интерфейса Runnable. Создание множественных потоков. Приоритеты потоков. Синхронизация и передача сообщений. Использование синхронизированных методов. Межпоточные связи. Блокировка. Приостановка, возобновление и остановка потоков.

В ходе освоения *раздела 8* «Апплеты и события» студенты должны уяснить: Основы апплетов. Класс Applet. Архитектура апплета. Инициализация и завершение апплета. Простые методы отображения апплетов. Требование перерисовки. Пересылка параметров в апплеты. Основные методы класса Applet. Вывод в консоль. Обработка событий. Модель делегирования событий. Источники событий. Классы событий. Обработка событий мыши. Обработка событий клавиатуры.

В ходе освоения *раздела 9* «Разработка пользовательского интерфейса в Java» студенты должны уяснить: Основы оконной графики. Класс Component. Класс Container. Класс Panel. Класс Window. Класс Frame. Работа с фреймовыми окнами. Создание фреймокна в апплете. Обработка событий фрейм-окна. Отображение информации в окне. Работа с графикой. Работа со шрифтами. Управление текстовым выводом. Использование элементов управления, менеджеров компоновки и меню AWT. Элементы управления. Основные понятия. Добавление и удаление элементов управления. Реагирование на элементы управления. Понятие менеджера компоновки. Работа с меню и диалоговыми окнами.

При подготовке к *экзамену* рекомендуется особое внимание уделить следующим вопросам: Языки гипертекстовой разметки документов. Язык Java. Обзор базовых конструкций и основных элементов языка. Объектно-ориентированное программирование на языке Java. Основы ввода/вывода. Работа со строковыми данными. Методы и средства обработки исключений. Средства для организации работы в сети. Многопоточное программирование. Апплеты и события. Разработка пользовательского интерфейса в Java.

В процессе проведения *лабораторных работ* происходит формирование умений и навыков разработки WEB – ресурса с использованием языков гипертекстовой разметки документов HTML и DHTML; создания и администрирования WEB – сервера; разработки WEB – приложения с использованием JavaScript; программирования мульти платформенного Java – приложения в интегрированной среде JBuilder; создание WEB – ресурса с использованием апплетов языка Java; разработки сетевого программного приложения в клиент-серверной архитектуре.

Работа с *литературой* является важнейшим элементом в получении знаний по дисциплине. Прежде всего, необходимо воспользоваться списком рекомендуемой по данной дисциплине литературой. Дополнительные сведения по изучаемым темам можно найти в периодической печати и Интернете.

Предусмотрено проведение аудиторных занятий (в виде презентаций, проблемной лекции, лекции с запланированными ошибками) в сочетании с внеаудиторной работой.

Приложение 2

АННОТАЦИЯ
рабочей программы дисциплины
Программирование сетевых приложений

1. Цель и задачи дисциплины

Целью изучения дисциплины является изучение языка программирования Java и современных технологий эффективной разработки и использования локальных, корпоративных и глобальных сетей для решения практических задач.

Задачей изучения дисциплины является:

- формирование у студентов базовых понятий и навыков создания сетевых программных комплексов в операционной среде Windows.

2. Структура дисциплины

2.1 Распределение трудоемкости по отдельным видам учебных занятий, включая самостоятельную работу: лекции – 17 часов, лабораторные работы – 34 часа, самостоятельная работа – 75 часов.

Общая трудоемкость дисциплины составляет 180 часов, 5 зачетных единицы.

2.2 Основные разделы дисциплины:

1- Языки гипертекстовой разметки документов.
2- Язык Java. Обзор базовых конструкций и основных элементов языка.
3- Объектно-ориентированное программирование на языке Java
4- Основы ввода/вывода. Работа со строковыми данными.
5- Методы и средства обработки исключений.
6- Средства для организации работы в сети.
7- Многопоточное программирование.
8- Апплеты и события.
9- Разработка пользовательского интерфейса в Java.

3. Планируемые результаты обучения (перечень компетенций)

Процесс изучения дисциплины направлен на формирование следующих компетенций:

ОПК-4 - способностью иметь навыки самостоятельной работы на компьютере и в компьютерных сетях, осуществлять компьютерное моделирование устройств, систем и процессов с использованием универсальных пакетов прикладных компьютерных программ.

ПК-9 - умение проводить расчёты по проекту сетей, сооружений и средств инфокоммукаций в соответствии с техническим заданием с использованием как стандартных методов, приёмов и средств автоматизации проектирования, так и самостоятельно создаваемых оригинальных программ.

4. Вид промежуточной аттестации: экзамен, КР.

Приложение 3

Протокол о дополнениях и изменениях в рабочей программе
на 20___-20___ учебный год

1. В рабочую программу по дисциплине вносятся следующие дополнения:

_____

_____

2. В рабочую программу по дисциплине вносятся следующие изменения:

_____

_____

_____

Протокол заседания кафедры №____ от «___» _____ 20 __ г.,
(разработчик)

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

Программа составлена в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению подготовки 11.03.02 Инфокоммуникационные технологии и системы связи от «06» марта 2015г. № 174.

для набора 2015 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «13» июля 2015 г. №475

для набора 2016 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «06» июня 2016 г. №429

для набора 2017 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «06» марта 2017 г. №125

для набора 2018 года и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «12» марта 2018 г. №130

Программу составил:

Федяев П.А., к.т.н., доцент каф. УТС _____

Рабочая программа рассмотрена и утверждена на заседании кафедры УТС
от «28» декабря 2018 г., протокол № 6

Заведующий кафедрой УТС _____ Игнатьев И.В.

СОГЛАСОВАНО:

Заведующий выпускающей кафедрой УТС _____ Игнатьев И.В.

Директор библиотеки _____ Сотник Т.Ф.

Рабочая программа одобрена методической комиссией ФЭиА
от «28» декабря 2018 г., протокол № 5

Председатель методической комиссии факультета _____ Ульянов А.Д.

СОГЛАСОВАНО:

Начальник
учебно-методического управления _____ Нежевец Г.П.

Регистрационный №_____