НЕКОММЕРЧЕСКОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
УЧЕБНО-НАУЧНО-ПРОИЗВОДСТВЕННЫЙ КОМПЛЕКС
«МЕЖДУНАРОДНЫЙ УНИВЕРСИТЕТ КЫРГЫЗСТАНА»

«СОГЛАСОВАНО»
Проректор по учебно-административной работе НОУ УНПК «МУК»,
к.ю.н., Карабалаева С.Б.
_____
«18» 11 2019г.

«УТВЕРЖДЕНО»
Ректор НОУ УНПК «МУК»,
к.т.н., доцент Савченко Е.Ю.
_____
«18» 11 2019г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

Название дисциплины: Программирование

Направление: 710100 «Информатика и вычислительная техника»

Профиль: Компьютерные информационные системы для бизнеса

Квалификация выпускника: Бакалавр

Форма обучения: Очная

Составитель (и): к.т.н., доц. Мусакулова Ж.А. _____

к.т.н., и.о. доц. Нежинских С.С._____

График проведения модулей
3-семестр

неделя	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
лекц. зан.	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
лаб. зан.	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

«РАССМОТРЕНО»
На заседании кафедры
«КИСиУ»
НОУ УНПК «МУК»
Протокол № 2
от «14» 10 2019 г.
Зав. кафедрой
д.т.н., проф. Миркин Е.Л.
_____

«ОДОБРЕНО»
На заседании Учебно-методического объединения НОУ УНПК «МУК»
Протокол № 2
от «15» 11 2019 г.
Председатель Учебно-методического объединения
Матвеева Т.В. _____

Директор Научной библиотеки
НОУ УНПК «МУК»
Асанова Ж.Ш._____

Бишкек 2019г.

ОГЛАВЛЕНИЕ

АННОТАЦИЯ 2
УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ (МОДУЛЕЙ) 3
1. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 3
1.1. Миссия и стратегия 3
1.2. Цель и задачи дисциплины 3
1.3. Формируемые компетенции, а также перечень планируемых результатов обучения по дисциплине 3
1.4. Место дисциплины (модулей) в структуре ООП ВПО 4
2. СТРУКТУРА ДИСЦИПЛИНЫ 5
3. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ 8
4. КОНСПЕКТ ЛЕКЦИЙ 10
5. ИНФОРМАЦИОННЫЕ И ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ 10
6. ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ТЕКУЩЕГО, РУБЕЖНОГО И ИТОГОВОГО КОНТРОЛЕЙ ПО ИТОГАМ ОСВОЕНИЮ ДИСЦИПЛИНЫ (МОДУЛЕЙ) 13
6.1. Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины 13
6.2. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности 15
6.3. Описание показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания 17
6.4. Типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности. 19
7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ 27
7.1. Список источников и литературы 27
7.2. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимый для освоения дисциплины (модулей) 28
8. ПЕРЕЧЕНЬ УЧЕБНО-МЕТОДИЧЕСКОГО ОБЕСПЕЧЕНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ 28
8.1. Планы практических (семинарских) и лабораторных занятий. Методические указания по организации и проведению 28
8.2. Методические указания для обучающихся по освоению дисциплины (модулей) 33
8.3. Методические рекомендации по подготовке отчетов по лабораторным работам 33
9. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ 35
10. ГЛОССАРИЙ 35
11. ПРИЛОЖЕНИЯ 38

АННОТАЦИЯ

Дисциплина «Программирование» представляет собой специализированный курс, который является одним из важнейших при подготовке специалистов в области информационных технологий. Язык программирования Visual C++, изучаемый в предлагаемом курсе, является инструментом, сочетающим гибкость/мощь «низкоуровневых» возможностей, присущих языкам ассемблера, с удобством и возможностью «абстрактного» программирования, предоставляемыми сегодня объектно-ориентированными языками. Это позволяет с одинаковым успехом создавать, как сколь угодно сложные прикладные программы, так и компоненты системного уровня. Кроме того, в наши дни Visual C++ лидирует среди продуктов для программирования в среде Windows.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ (МОДУЛЕЙ)

1. Пояснительная записка

1.1. Миссия и стратегия

Миссия НОУ УНПК "МУК" – подготовка международно - признанных, свободно мыслящих специалистов, открытых для перемен и способных трансформировать знания в ценности на благо развития общества.

Видение НОУ УНПК «МУК»- создание динамичного и креативного университета с инновационными научно-образовательными программами и с современной инфраструктурой, способствующие достижению академических и профессиональных целей.

Стратегии развития - модернизация образовательной деятельности университета – совершенствование образовательного процесса в соответствии с требованиями Болонского процесса.

1.2. Цель и задачи дисциплины

Цель дисциплины: является предоставление студентам навыков программирования на языке высокого уровня. Знакомство с основными принципами процедурного и объектно-ориентированного программирования. Изучение данной дисциплины позволит студентам адаптироваться в среде программирования Visual C++, и освоить основные приемы и принципы программирования.

Задачи дисциплины:

- формирование у студентов знаний об основах структурного, процедурного и объектно-ориентированного программирования;
- ознакомление с теоретическими и практическими приемами и методами объектно-ориентированного программирования;
- приобретение практических навыков написания программных продуктов на языке C++;
- подготовить специалистов к использованию объектно-ориентированного программирования в научно-исследовательской деятельности.

1.3. Формируемые компетенции, а также перечень планируемых результатов обучения по дисциплине

Дисциплина «Программирование» направлена на формирование следующих компетенций:

- *общенаучными (*ОК-2*):*
  - способен использовать базовые положения математических /естественных/ гуманитарных/ экономических наук при решении профессиональных задач;
- *профессиональными* (ПК-2, ПК-3)*:*
  - способен освоить методики использования программных средств для решения практических задач;
  - способен разрабатывать интерфейсы «человек - электронно-вычислительная машина»;

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1. Знать:

- основные принципы объектно-ориентированного программирования; *(*ОК-2*)*
- методику использования языка C++ для решения практических задач; (ПК-2)
- Виды функций. Понятие классов и наследования. *(*ПК-2)

2. *Уметь:*
   - использовать базовые принципы математики при алгоритмизации поставленных задач (ОК-2)
   - использовать среду Visual Studio (Функции, Классы) для решения практических задач. (ПК-2).
   - разрабатывать готовые приложения в среде Visual Studio C++ (ПК-3).

3. *Владеть:*
   - инструментальным средством разработки приложений Visual Studio C++. (ПК-2).
   - навыком разработки готовых интерфейсов «человек - электронно-вычислительная машина» в среде Visual Studio C++.;

Результаты обучения могут быть представлены в виде таблицы

Коды компетенции	***Содержание компетенций***	***Перечень планируемых результатов обучения по дисциплине***
*(*ОК-2*)*	*общенаучные*	*Знать:* Понятия функция, метод, класс. Уметь: Использовать различные виды функций, классов. *Владеть:* способами создания классов, дружественных и производных классов;
(ПК-2)	*профессиональные*	*Знать:* основные виды функций; *Уметь:* создавать функции, объекты различных классов; *Владеть:* практическими навыками создания иерархии объектов;
(ПК-3)	*профессиональные*	*Знать:* возможности создания классов и объектов; *Уметь:* решать и алгоритмизировать различные задачи, с использованием иерархии объектов *Владеть* - навыками самостоятельной работы при разработки пользовательских приложений

1.4. Место дисциплины (модулей) в структуре ООП ВПО

Дисциплина (модуль) «Программирование» является частью профессионального цикла (блока) дисциплин учебного плана по направлению подготовки 710100 «Информатика и вычислительная техника» подготовки бакалавров (специализации Информатика и вычислительная техника).

Для освоения дисциплины (модулей) необходимы компетенции, сформированные в ходе изучения следующих дисциплин и прохождения практик: основные разделы математики, основы программирования, математическая логика и теория алгоритмов.

В результате освоения дисциплины (модулей) формируются компетенции, необходимые для изучения следующих дисциплин и прохождения практик: WEB-ориентированные информационные системы, нейросетевые технологии, JAVA программирование информационных систем.

2. Структура дисциплины

Общая трудоемкость дисциплины составляет 6 кредитов, 180ч., в том числе аудиторная работа обучающихся с преподавателем 102 ч., самостоятельная работа обучающихся 78 ч;

№ п/п	Раздел, Темы Дисциплины	***Семестр***	***Неделя семестра***	***Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)***				***Формы текущего контроля успеваемости (по неделям семестра) Форма промежуточной аттестации (по семестрам)***
				Лекции 34ч	***Лаб. Зан. 68ч***	***СРС 46ч***	***СРСиП 32ч***	
1	Функции. Собственные функции. Назначение собственных функций.			*2*	*4*	*3*	*2*	
2	Виды собственных функций Формальные и фактические аргументы.			*2*	*4*	*3*	*2*	
3	Функции с переменным числом аргументов. Макроопределения.			*2*	*4*	*3*	*2*	
4	Перегрузка функций.			*2*	*4*	*3*	*2*	*Сдача модуля*
5	Понятие динамической			*2*	*4*	*3*	*2*	

	памяти. Назначение и использование функций new и delete							
6	Создание динамических массивов			*2*	*4*	*3*	*2*	
7	Понятие структуры. Понятие объединения. Переменные структуры.			*2*	*4*	*3*	*2*	
8	Понятие и назначение классов. Компонентные данные класса.			*2*	*4*	*3*	*2*	*Сдача модуля*
9	Способы доступа к компонентам класса. Спецификаторы доступа.			*2*	*4*	*3*	*2*	
10	Конструкторы. Деструкторы			*2*	*4*	*3*	*2*	
11	Способы доступа к компонентам класса. Использование указателей.			*2*	*4*	*3*	*2*	
12	Дружественные функции и их возможности. Определение дружественных функций.			*2*	*4*	*3*	*2*	*Сдача модуля*
13	Организация дружественных классов.			*2*	*4*	*3*	*2*	
14	Механизм наследования. Базовые и производные классы.			*2*	*4*	*3*	*2*	

15	Определение производных классов. Статусы доступа при наследовании.			*2*	*4*	*2*	*2*	
16	Множественное наследование.			*2*	*4*	*1*	*1*	*Сдача модуля*
17	Абстрактные классы.			*2*	*4*	*1*	*1*	

3. Содержание дисциплины

№	*Наименование раздела, темы дисциплины*	*Краткое содержание*
1	Функции. Собственные функции. Назначение собственных функций.	Понятие функции. Собственные и библиотечные функции. Прототип функции. Описание и вызов функции.
2	Виды собственных функций Формальные и фактические аргументы.	Виды собственных функций. Функции с аргументами и без аргументов. Функции возвращающие и не возвращающие значения. Понятие формальных и фактических аргументов функции.
3	Функции с переменным числом аргументов. Макроопределения.	Функции с переменным числом аргументов. Использование макросов из библиотеки stdarg.h. Вызов функции с переменным числом аргументов
4	Перегрузка функций.	Перегрузка функций. Правила перегрузки функций. Использование ссылок и указателей. Передача аргументов функции по значению.
5	Понятие динамической памяти. Назначение и использование функций new и delete	Понятие динамической памяти. Назначение и использование функций new и delete. Массив, как аргумент функции.
6	Создание динамических массивов	Создание динамических массивов. Динамическое распределение памяти. Создание одномерных и двумерных динамических массивов.
7	Понятие структуры. Понятие объединения. Переменные структуры.	Структуры данных. Понятие объединения. Переменные структуры. Использование ключевых слов struct, union.
8	Понятие и назначение классов. Компонентные данные класса.	Понятие и назначение классов. Определение класса. Компонентные данные класса. Создание объектов класса.
9	Способы доступа к компонентам класса. Спецификаторы доступа.	Использование операции «точка» для доступа к компонентам класса. Спецификаторы доступа. Использование ключевых слов public, private, protected.
10	Конструкторы. Деструкторы	Конструкторы. Создание конструкторов. Уничтожение объекта с помощью деструкторов. Правила использования конструкторов и деструкторов.
11	Способы доступа к компонентам класса. Использование указателей.	Способы доступа к компонентам класса. Использование указателей.

12	Дружественные функции и их возможности. Определение дружественных функций.	Дружественные функции и их возможности. Определение дружественных функций. Использование ключевого слова friend.
13	Организация дружественных классов.	Организация дружественных классов. Механизм доступа к компонентам класса из дружественного класса.
14	Механизм наследования. Базовые и производные классы.	Механизм наследования. Базовые и производные классы. Операция уточнения области видимости. Принципы и правила наследования.
15	Определение производных классов. Статусы доступа при наследовании.	Определение производных классов. Статусы доступа при наследовании. Использование спецификаторов доступа при наследовании.
16	Множественное наследование.	Множественное наследование. Правила множественного наследования. Полиморфизм и инкапсуляция.
17	Абстрактные классы.	Виртуальные функции, раннее и позднее связывание. Абстрактные классы.

4. Конспект лекций

См. приложение 1.

5. Информационные и образовательные технологии

Информационные и образовательные технологии

№ п/п	***Наименование раздела***	***Виды учебной работы***	***Формируемые компетенции (указывается код компетенции)***	***Информационные и образовательные технологии***
1	***2***	***3***	***4***	***5***
1	Функции. Собственные функции. Назначение собственных функций.	*Лекция* *Лабораторная работа.* *Самостоятельная работа*	*(ОК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
2	Виды собственных функций Формальные и фактические аргументы.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ОК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
3	Функции с переменным числом аргументов. Макроопределения.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ОК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
4	Перегрузка функций.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ОК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
5	Понятие динамической памяти. Назначение	*Лекция*	*(ПК-2)*	***Лекция-визуализация*** *с применением проектора*

	и использование функций new и delete	*Лабораторная работа* *Самостоятельная работа*	*(ОК-2) (ПК-2)* *(*ПК-3*)*	***Решение*** *(программирование) задач*
6	Создание динамических массивов	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ОК-2*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
7	Понятие структуры. Понятие объединения. Переменные структуры.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
8	Понятие и назначение классов. Компонентные данные класса.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
9	Способы доступа к компонентам класса. Спецификаторы доступа.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
10	Конструкторы. Деструкторы	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*

11	Способы доступа к компонентам класса. Использование указателей.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
12	Дружественные функции и их возможности. Определение дружественных функций.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
13	Организация дружественных классов.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ОК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
14	Механизм наследования. Базовые и производные классы.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
15	Определение производных классов. Статусы доступа при наследовании.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ОК-2) (ПК-2)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*
16	Множественное наследование.	*Лекция* *Лабораторная работа*	*(ПК-2)* *(ПК-2) (ПК-3)*	***Лекция-визуализация*** *с применением проектора*

		Самостоятельная работа	*(*ПК-3*)*	***Решение*** *(программирование) задач*
17	Абстрактные классы.	*Лекция* *Лабораторная работа* *Самостоятельная работа*	*(ПК-2)* *(ПК-2) (ПК-3)* *(*ПК-3*)*	***Лекция-визуализация*** *с применением проектора* ***Решение*** *(программирование) задач*

6. Фонд оценочных средств для текущего, рубежного и итогового контролей по итогам освоению дисциплины (модулей)

6.1. Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины

Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины представляется в виде таблицы:

№ п/п	***Контролируемые разделы дисциплины (модулей)***	***Код контролируемой компетенции (компетенций)***	***Наименование оценочного средства***
1	Функции. Собственные функции. Назначение собственных функций.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
2	Виды собственных функций Формальные и фактические аргументы.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
3	Функции с переменным числом аргументов. Макроопределения.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины) ТЕСТ
4	Перегрузка функций.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание)

			(Вопросы по темам/разделам дисциплины) Контрольная работа
5	Понятие динамической памяти. Назначение и использование функций new и delete	*(ПК-2) (ПК-3)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины) Темы рефератов
6	Создание динамических массивов	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины) Темы рефератов
7	Понятие структуры. Понятие объединения. Переменные структуры.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
8	Понятие и назначение классов. Компонентные данные класса.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины) Контрольная работа
9	Способы доступа к компонентам класса. Спецификаторы доступа.	*(ОК-1)* *(ПК-2) (ПК-3)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
10	Конструкторы. Деструкторы	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
11	Способы доступа к компонентам класса. Использование указателей.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
12	Дружественные функции и их возможности.	*(ПК-2) (ПК-3)* *(ОК-2)*	Контрольная работа

	Определение дружественных функций.		
13	Организация дружественных классов.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
14	Механизм наследования. Базовые и производные классы.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
15	Определение производных классов. Статусы доступа при наследовании.	*(ПК-2) (ПК-3)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины)
16	Множественное наследование.	*(ПК-2) (ПК-3)* *(ПК-2) (ОК-2)*	Задача (практическое задание) (Вопросы по темам/разделам дисциплины) Контрольная работа
	Абстрактные классы.	*(ПК-2) (ПК-3)*	Задача (практическое задание)

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

Методические материалы составляют систему текущего, рубежного и итогового (экзамена) контролей освоения дисциплины (модулей), закрепляют виды и формы текущего, рубежного и итогового контролей знаний, сроки проведения, а также его сроки и формы проведения (устный экзамен, письменный экзамен и т.п.). В системе контроля указывается процедура оценивания результатов обучения, при использовании балльно-рейтинговой системы приводится таблица с баллами и требованиями к пороговым значениям достижений по видам деятельности обучающихся; показывается механизм получения оценки (из чего складывается оценка по дисциплине (модулю).

Текущий контроль *осуществляется в виде опроса, участие в дискуссии на семинаре, выполнение самостоятельной работы и других видов работ, указанных в УМК, а также посещаемости студентов занятий - оценивается до 80 баллов.*

Рубежный контроль *(сдача модулей) проводится преподавателем и представляет собой письменный контроль, либо компьютерное тестирование знаний по теоретическому и практическому материалу. Контрольные вопросы рубежного контроля*

включают полный объём материала части дисциплины (модулей), позволяющий оценить знания, обучающихся по изученному материалу и соответствовать УМК дисциплины, которое оценивается до 20 баллов.

***Итоговый контроль** (экзамен) знаний принимается по экзаменационным билетам, включающий теоретические вопросы и практическое задание, и оценивается до 20 баллов.*

Форма контроля	*Срок отчетности*	*Макс. количество баллов*	
		За одну работу	*Всего*
Текущий контроль: *- Прием лабораторных работ*	*1,2,3,4 неделя*	*8 баллов*	*До 40 баллов*
-опрос	*1,,2,3,4 неделя*	*6 баллов*	*До 30 баллов*
- посещаемость	*1,2,3,4 неделя*	*2 балла*	*10 баллов*
Рубежный контроль: *(сдача модуля)*	*4 неделя*	*100%×0,2=20 баллов*	
Итого за I модуль			*До 100 баллов*

Форма контроля	*Срок отчетности*	*Макс. количество баллов*	
		За одну работу	*Всего*
Текущий контроль: *- Прием лабораторных работ*	*5,6,7,8 неделя*	*10 баллов*	*До 40 баллов*
-опрос	*5,6,7,8 неделя*	*6 баллов*	*До 30 баллов*
- посещаемость	*5,6,7,8 неделя*	*2 балла*	*10 баллов*
Рубежный контроль: *(сдача модуля)*	*8 неделя*	*100%×0,2=20 баллов*	
Итого за II модуль			*До 100 баллов*

Форма контроля	*Срок отчетности*	*Макс. количество баллов*	
		За одну работу	*Всего*
Текущий контроль: *- Прием лабораторных работ*	*9,10,11,12 неделя*	*8 баллов*	*До 40 баллов*

-опрос	*9,10,11,12 неделя*	*6 баллов*	*До 30 баллов*
- посещаемость	*9,10,11,12 неделя*	*2 балла*	*10 баллов*
Рубежный контроль: (сдача модуля)	*12 неделя*	*100%×0,2=20 баллов*	
Итого за I модуль			*До 100 баллов*

Форма контроля	*Срок отчетности*	*Макс. количество баллов*	
		За одну работу	*Всего*
Текущий контроль:			
- Прием лабораторных работ	*13,14,15, 16 неделя*	*10 баллов*	*До 40 баллов*
-опрос	*13,14,15, 16 неделя*	*6 баллов*	*До 30 баллов*
- посещаемость	*13,14,15, 16 неделя*	*2 балла*	*10 баллов*
Рубежный контроль: (сдача модуля)	*16 неделя*	*100%×0,2=20 баллов*	
Итого за II модуль			*До 100 баллов*

Экзаменатор выставляет по результатам балльной системы в семестре экзаменационную оценку без сдачи экзамена, набравшим суммарное количество баллов, достаточное для выставления оценки от 55 и выше баллов – автоматически (при согласии обучающегося).

Полученный совокупный результат (максимум 100 баллов) конвертируется в традиционную шкалу:

Рейтинговая оценка (баллов)	Оценка экзамена
От 0 - до 54	неудовлетворительно
от 55 - до 69 включительно	удовлетворительно
от 70 – до 84 включительно	хорошо
от 85 – до 100	отлично

6.3.Описание показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания

Текущий контроль (0 - 80 баллов)

При оценивании посещаемости, опроса и приема лабораторных работ из расчета на одну неделю учитываются:

- посещаемость (2 балла одно занятие (10 баллов за модуль)

- степень раскрытия содержания материала (2.8 балла одно занятие (14 баллов за модуль);

- изложение материала (грамотность речи, точность использования терминологии и символики, логическая последовательность изложения материала (2.8 балла одно занятие (14 баллов за модуль);

- знание теории изученных вопросов (2.8 балла одно занятие (14 баллов за модуль);

- сформированность и устойчивость используемых при ответе умений и навыков (2.8 балла одно занятие (14 баллов за модуль);

-точность решения задачи (2.8 балла одно занятие (14 баллов за модуль).

Рубежный контроль (0 – 20 баллов)

При оценивании контрольной работы учитывается:

- полнота выполненной работы (задание выполнено не полностью и/или допущены две и более ошибки или три и более неточности) – 8 баллов;

- обоснованность содержания и выводов работы (задание выполнено полностью, но обоснование содержания и выводов недостаточны, но рассуждения верны) – 14 баллов;

- работа выполнена полностью, в рассуждениях и обосновании нет пробелов или ошибок, возможна одна неточность - 17 баллов.

- работа выполнена полностью, в рассуждениях и обосновании нет пробелов или ошибок - 20 баллов.

При оценивании теста учитывается:

- полнота выполненной работы (задание выполнено не полностью и/или допущены две и более ошибки или три и более неточности) – до 20 баллов;

Итоговый контроль (экзаменационная сессия) - ИК = Бср × 0,8+Бэкз × 0,2

При проведении итогового контроля обучающийся должен ответить на 3 вопроса (два вопроса теоретического характера и один вопрос практического характера).

При оценивании ответа на вопрос теоретического характера учитывается:

- теоретическое содержание не освоено, знание материала носит фрагментарный характер, наличие грубых ошибок в ответе (2 балла);

- теоретическое содержание освоено частично, допущено не более двух-трех недочетов (5 баллов);

- теоретическое содержание освоено почти полностью, допущено не более одного-двух недочетов, но обучающийся смог бы их исправить самостоятельно (8 баллов);

- теоретическое содержание освоено полностью, ответ построен по собственному плану (10 баллов).

При оценивании ответа на вопрос практического характера учитывается:

- ответ содержит менее 20% правильного решения (3 балла);

- ответ содержит 21-89 % правильного решения (7 баллов);

- ответ содержит 90% и более правильного решения (10 баллов).

6.4.Типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности.

Раздел УМК включает образцы оценочных средств, примерные перечни вопросов и заданий в соответствии со структурой дисциплины и системой контроля.

Контрольные вопросы

1. Функции. Собственные функции. Назначение собственных функций.
2. Прототип функции. Определение функции. Вызов функции.
3. Виды собственных функций Формальные и фактические аргументы.
4. Функции, возвращающие значение. Функции, не возвращающие значения
5. Функции с переменным числом аргументов. Макроопределения.
6. Функции, как отдельный файл.
7. Область действий и время жизни переменных.
8. Перегрузка функций.
9. Указатели и функции.
10. Понятие динамической памяти. Назначение и использование функций new и delete.
11. Создание динамического одномерного массива.
12. Создание двумерного динамического массива.
13. Понятие структуры.
14. Понятие объединения. Переменные структуры.
15. Понятие и назначение классов. Компонентные данные класса.
16. Способы доступа к компонентам класса. Спецификаторы доступа.
17. Конструкторы. Назначение конструктора.
18. Деструкторы. Назначение деструкторов.
19. Способы доступа к компонентам класса. Использование указателей.
20. Дружественные функции и их возможности. Определение дружественных функций.
21. Организация дружественных классов.
22. Наследование.
23. Механизм наследования. Базовые и производные классы.
24. Определение производных классов. Статусы доступа при наследовании.
25. Множественное наследование.
26. Виртуальные функции.
27. Абстрактные классы.
28. Инкапсуляция.
29. Полиморфизм.

Тематика рефератов

1. Основные понятия языков программирования. Трансляция. Компиляция и интерпретация.
2. Структуры и типы данных языка программирования.
3. Эволюция и классификация языков программирования.
4. История языков программирования.
5. Язык компьютера и человека.
6. Объектно-ориентированное программирование.
7. Непроцедурные системы программирования.
8. Искусственный интеллект и логическое программирование.
9. Языки манипулирования данными в реляционных моделях.
10. История программирования в лицах.

Контрольная работа 1

1. Написать собственную функцию с аргументами не возвращающую значение, для расчета площади и длины окружности.
2. Написать собственную функцию с аргументами не возвращающую значение, которая выводит название дня недели, в зависимости какое число было введено пользователем.
3. Написать собственную функцию с аргументами не возвращающую значение, для проверки является ли число, введенное пользователем четным.
4. Написать собственную функцию, с аргументами возвращающую значение для расчета периметра прямоугольника. Функцию оформить как отдельный файл.
5. Написать собственную функцию, с аргументами не возвращающую значение, для проверки, является ли год, введенный пользователем високосным.
6. Написать собственную функцию, с аргументами возвращающую значение, для пересчета веса из фунтов в килограммы. (1 фунт-это 405,9 грамма). Функцию оформить как отдельный файл.
7. Написать собственную функцию с аргументами не возвращающую значение для расчета объема куба и площади одной его грани.
8. Написать собственную функцию с аргументами возвращающую значение, для нахождения минимального числа из двух чисел. Функцию оформить как отдельный файл.
9. Написать собственную функцию, с аргументами возвращающую значения для вычисления площади поверхности параллелепипеда.
10. Написать собственную функцию, с аргументами не возвращающую значение для нахождения корней квадратного уравнения.
11. Написать собственную функцию, с аргументами не возвращающую значение, которая проверяет, делиться ли на три без остатка число, введенное пользователем.
12. Написать собственную функцию, с аргументами возвращающую значения, для нахождения остатка от деления двух целых чисел. Функцию оформить как отдельный файл.
13. Написать собственную функцию, с аргументами не возвращающую значения, для нахождения гипотенузы и площади прямоугольного треугольника.
14. Написать собственную функцию, с аргументами возвращающую значение, которая возводит в квадрат число, если оно четное, и в куб, если число нечетное. Функцию оформить как отдельный файл.
15. Написать собственную функцию с аргументами не возвращающую значение, которая в зависимости от оценки, выводит название оценки (отлично, хорошо, удов, неуд) или сообщение об ошибке, если такой оценки не существует.
16. Написать собственную функцию с аргументами возвращающую значение, для нахождения объема цилиндра.
17. Написать собственную функцию с аргументами не возвращающую значение, которая выводит название времени года, в зависимости от введенного номера месяца.
18. Написать собственную функцию, с аргументами возвращающую значение, для пересчета расстояния из верст в километры (1 верста - это 1066,8 м).
19. Написать собственную функцию, с аргументами не возвращающую значение, которая проверяет, длится ли на пять без остатка введенное число.
20. Написать собственную функцию, с аргументами возвращающую значение, для нахождения катета прямоугольного треугольника по известному катету и гипотенузе. Написать собственную функцию, с аргументами не возвращающую значение для нахождения объема конуса.

21. Написать собственную функцию, с аргументами возвращающую значение, которая возвращает 1, если число четное, и -1, если число нечетное. Функцию оформить как отдельный файл.
22. Написать собственную функцию, с аргументами возвращающую значение, для нахождения суммы всех четных делителей введенного числа.
23. Написать собственную функцию, с аргументами не возвращающую значение, для нахождения силы тока в электрической цепи.
24. Написать собственную функцию, с аргументами не возвращающую значение, которая в зависимости от введенного номера дня недели выводит «рабочий день» или «выходной»
25. Написать собственную функцию, с аргументами возвращающую значение, для нахождения количества делителей введенного числа. Функцию оформить как отдельный файл.
26. Написать три функции (объемы геометрических тел) для геометрических расчетов и оформить их в отдельном файле.
27. Написать три функции (степень, обмен значений, сумма чисел) для арифметических расчетов и оформить их в отдельном файле.
28. Написать три функции для округления чисел (по правилам, к меньшему целому, к большему целому) и оформить их в отдельном файле.
29. Написать три функции для определения характеристик числа (четное/нечетное, простое/непростое, совершенное/несовершенное) и оформить их в отдельном файле.
30. Написать три функции для работы с числом и его делителями (сумма делителей, сумма четных делителей, сумма нечетных делителей) и оформить их в отдельном файле.
31. Написать три функции для нахождения произведения, суммы, среднегеометрического значения первых k-натуральных чисел.
32. Написать две функции (наибольший общий делитель двух чисел, остаток от деления двух чисел) для арифметических расчетов и оформить их в отдельном файле.

Контрольная работа 2

1. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает собственную функцию для замены всех четных элементов массива на 0.
2. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает собственную функцию для нахождения суммы четных элементов массива.
3. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для замены всех элементов кратных 3-м на 3.
4. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения максимального элемента массива.
5. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения количества элементов кратных 3-м и 5-ти.
6. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения произведения нечетных элементов массива.
7. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения среднего геометрического значения элементов кратных 3-м и 5-ти.

8. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения произведения элементов кратных 3-м.
9. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения среднего арифметического значения элементов кратных 5-ти.
10. Дан массив из 10 элементов. Элементы массива вводятся с клавиатуры. Написать программу, которая включает функцию для нахождения максимального и минимального элемента, затем меняет их местами.
11. Написать программу, которая включает функцию с переменным числом параметров, для нахождения среднеарифметического значения нечетных аргументов функции.
12. Написать программу, которая включает функцию с переменным числом параметров, для нахождения минимального аргумента функции.
13. Написать программу, которая включает функцию с переменным числом параметров, для нахождения среднеарифметического значения четных аргументов функции.
14. Написать программу, которая включает функцию с переменным числом параметров, для нахождения минимального аргумента среди четных аргументов функции.
15. Написать программу, которая включает функцию с переменным числом параметров, для нахождения суммы аргументов функции.
16. Написать программу, которая включает функцию с переменным числом параметров, для нахождения максимального аргумента функции.
17. Написать программу, которая включает функцию с переменным числом параметров для нахождения произведения аргументов функции.
18. Написать три функции (произведение чисел, средне арифметическое значение, сумма чисел) для арифметических расчетов и оформить их в отдельном файле.

Контрольная работа 3

1. Дан динамический массив. Элементы массива вводятся с клавиатуры. Найти количество четных элементов массива.
2. Дан динамический массив. Элементы массива вводятся с клавиатуры. Найти сумму нечетных элементов массива.
3. Дан динамический массив. Элементы массива вводятся с клавиатуры. Найти произведение элементов массива кратных 3-м.
4. Дан динамический массив. Элементы массива вводятся с клавиатуры. Найти сумму элементов массива.
5. Дан динамический массив. Элементы массива вводятся с клавиатуры. Найти произведение элементов массива.
6. Написать программу, которая включает функцию (не возвращающую значение) для создания массива, каждый элемент которого равен максимальному из соответствующих значений двух других массивов.
7. Написать программу, которая включает функцию (возвращающую значение) для создания массива, каждый элемент которого, равен минимальному из соответствующих значений двух других массивов.
8. Написать функцию. В одномерном динамическом массиве найти максимальный по модулю элемент массива и сумму элементов массива, расположенных между первым и вторым положительными элементами.
9. Написать функцию. В одномерном динамическом массиве найти минимальный по модулю элемент массива. Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.
10. Написать функцию. В одномерном динамическом массиве найти сумму модулей элементов массива, расположенных после первого элемента, равного нулю.

Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине — элементы, стоявшие в нечетных позициях.

11. Написать функцию. В одномерном динамическом массиве найти номер минимального по модулю элемента массива и сумму модулей элементов массива, расположенных после первого отрицательного элемента.
12. Написать функцию. В одномерном динамическом массиве найти номер максимального по модулю элемента массива. Сжать массив, удалив из него все элементы, величина которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.
13. Написать функцию. В одномерном динамическом массиве вычислить сумму элементов массива, расположенных после первого положительного элемента. Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a, b], а потом — все остальные.
14. Написать функцию. В одномерном динамическом массиве найти количество элементов массива, лежащих в диапазоне от A до B и сумму элементов массива, расположенных после максимального элемента.
15. Написать функцию. В одномерном динамическом массиве найти количество элементов массива, равных нулю и вычислить сумму элементов массива, расположенных после минимального элемента.
16. Написать функцию. В одномерном динамическом массиве найти количество элементов массива, больших C и вычислить произведение элементов массива, расположенных после максимального по модулю элемента.
17. Написать функцию. В одномерном динамическом массиве найти количество отрицательных элементов массива. Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом — все положительные.
18. Написать функцию. В одномерном динамическом массиве вычислить сумму модулей элементов массива, расположенных после минимального по модулю элемента. Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.
19. Написать функцию. В одномерном динамическом массиве найти количество положительных элементов массива. Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает единицу, а потом — все остальные.
20. Написать функцию. В одномерном динамическом массиве вычислить сумму элементов массива, расположенных после последнего элемента, равного нулю. Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20 %, а потом — все остальные.
21. Написать функцию. В одномерном динамическом массиве найти количество элементов массива, меньших C и вычислить сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.
22. Написать функцию для нахождения суммы чисел в каждой строке двумерного динамического массива.
23. Написать функцию для нахождения произведения чисел в каждом столбце двумерного динамического массива.
24. Написать функцию для нахождения количества положительных чисел в каждом столбце двумерного динамического массива.
25. Написать функцию для нахождения суммы чисел для каждого столбца двумерного динамического массива, удовлетворяющих условию $x_{i,j} > a$. Здесь a- произвольная величина.

26. Написать функцию для нахождения произведения чисел для каждого столбца двумерного динамического массива, удовлетворяющих условию xi,,j < b. Здесь b – произвольная величина.
27. Написать функцию для нахождения значения наибольшего по модулю элемента для каждой строки двумерного динамического массива.
28. Написать функцию для нахождения значения наименьшего элемента для каждого столбца двумерного динамического массива.
29. Написать функцию для нахождения для каждой строки двумерного динамического массива значения наименьшего элемента и его индекс.
30. Написать функцию для нахождения значения наибольшего по модулю элемента и его индекс для каждого столбца двумерного динамического массива.
31. Написать функцию для нахождения суммы отрицательных чисел в каждой строке двумерного динамического массива.
32. Написать функцию для нахождения среднего значения чисел в каждой строке двумерного динамического массива.
33. Написать функцию для нахождения среднего значения чисел в каждом столбце двумерного динамического массива.
34. Написать функцию для нахождения для каждой строки двумерного динамического массива отклонение ее элементов от среднего значения для этой строки.
35. Написать функцию для нахождения для каждого столбца двумерного динамического массива отклонение его элементов от среднего значения для этого столбца.
36. Написать функцию для нахождения для каждого столбца двумерного динамического массива значение разности между наибольшим и наименьшим элементами.
37. Написать функцию для нахождения для каждой строки двумерного динамического массива значение разности между наибольшим и наименьшим элементами.
38. Написать функцию для нахождения суммы элементов для каждого столбца двумерного динамического массива. Поменять местами столбцы с максимальным и минимальным значением суммы.
39. Написать функцию для нахождения суммы элементов для каждой строки двумерного динамического массива. Поменять местами строки с максимальным и минимальным значением суммы.
40. Написать функцию для нахождения произведения элементов для каждого столбца двумерного динамического массива. Поменять местами столбцы с максимальным и минимальным значением произведения.
41. Написать функцию для нахождения произведения элементов для каждой строки двумерного динамического массива. Поменять местами строки с максимальным и минимальным значением произведения.
42. Написать функцию для нахождения суммы элементов для каждой строки двумерного динамического массива. Вывести на экран строки с максимальным и минимальным значениями суммы.
43. Написать функцию для нахождения суммы элементов двумерного динамического массива, которые расположены выше главной диагонали.

Контрольная работа 4

1. Описать структуру «Жесткий диск». Структура должна включать 5 компонент.
2. Описать структуру «Страны мира». Структура должна включать 5 компонент.
3. Описать структуру «Библиотека». Структура должна включать 5 компонент.
4. Описать структуру «Процессор». Структура должна включать 5 компонент.
5. Описать структуру «Компьютерная игра». Структура должна включать 5 компонент.

6. Создать переменную структуру, которая в зависимости от введенной метки находит или площадь окружности, или площадь прямоугольника, или площадь прямоугольного треугольника.
7. Создать переменную структуру, которая в зависимости от введенной метки находит или объем цилиндра, или объем конуса, или объем куба.
8. Создать переменную структуру, которая в зависимости от введенной метки находит или среднее арифметическое значение четных элементов массива, или среднее арифметическое значение нечетных элементов массива.
9. Создать класс «окружность», который включает два метода: для нахождения длины окружности, для нахождения объема шара. Инициализацию данных произвести в конструкторе.
10. Создать класс «параллелепипед», который включает два метода: для нахождения объема параллелепипеда, для нахождения площади поверхности параллелепипеда. Инициализацию данных произвести в конструкторе.
11. Создать класс «объемы», который включает три метода: для нахождения объема цилиндра, для нахождения объема конуса, для нахождения объема шара. Инициализацию данных произвести в конструкторе.
12. Написать программу, которая содержит класс «круг». Класс, включает методы для нахождения длины окружности и площади круга. Включить дружественную функцию для нахождения объема шара. Инициализацию данных произвести в конструкторе.
13. Написать программу, которая содержит класс «степень». Класс, включает методы для возведения одного числа в степень второго ab и наоборот ba. (Без использования математических функций). Включить дружественную функцию для сравнения чисел. Инициализацию данных произвести в конструкторе.
14. Написать программу, которая содержит класс «массив». Класс, включает методы для нахождения суммы и количества элементов массива, которые являются простыми числами. Включить дружественную функцию для нахождения среднеарифметического значения этих чисел. Инициализацию данных произвести в конструкторе.
15. Написать программу, которая содержит класс «массив». Класс, включает методы для нахождения максимального и минимального элемента массива. Включить дружественную функцию для нахождения разности этих чисел. Инициализацию данных произвести в конструкторе.
16. Написать программу, которая содержит класс «массив». Класс, включает методы для нахождения максимального и минимального элемента массива. Включить дружественную функцию, которая меняет местами максимальный и минимальный элемент. Инициализацию данных произвести в конструкторе.
17. Написать программу, которая содержит класс «число». Класс, включает методы для нахождения суммы и количества делителей числа. Включить дружественную функцию, для нахождения среднеарифметического значения делителей числа. Инициализацию данных произвести в конструкторе.
18. Написать программу, которая содержит класс «число» Класс, включает методы для нахождения суммы и количества делителей числа. Включить дружественный класс, который содержит методы для нахождения суммы и количества чисел, на которые не делится число, а также для нахождения разности среднеарифметического значения делителей числа и среднеарифметического значения «не делителей» числа. Инициализацию данных произвести в конструкторе.
19. Дан целочисленный массив из 7 элементов. Ввод и вывод элементов исходного массива описывается в базовом классе. В производном классе описываются функции вычисления суммы и произведения элементов массива.

20. Дана последовательность 5 целых чисел. Определить количество членов этой последовательности, являющихся нечетными числами и являющихся кратными 5. Ввод и вывод исходной последовательности осуществляется в базовом классе, остальные действия в производном.
21. Вычислить сумму вклада, которая будет находиться в банке по истечении 3-х месяцев. Известна начальная сумма вклада, которая вводится в базовом классе (функция доступа). Конечная сумма вклада вычисляется и выводится в производном классе. Проценты начисляемые за срок вклада 3 месяца - 25% годовых.
22. Класс вкладчик имеет следующие функции-методы: ввода начального вклада, вычисления суммы выплаты за определенный срок вклада, функция вывода суммы выплаты. Срок вклада задается с помощью конструктора и может быть 3 или 6 месяцев. Если срок вклада 3 мес - 25% годовых, 6 - 30% годовых. Вычислить какой из двух вкладчиков получит большую сумму денег.
23. Класс Фирма имеет следующие функции-методы: ввода суммы на которую торговый представитель данной фирмы продал товар, функция, определяющая его комиссионные, функция вывода комиссионных. Комиссионные начисляются по следующему правилу: если сумма продаж не превышает 1000 сом, то комиссионные составляют 10% от суммы продаж, если больше 1000 сом, но меньше 3000 сом - 15%, свыше 3000 сом -25%. Решить следующую задачу. У фирмы 2 торговых представителя. Определить комиссионные каждого торгового представителя и общую сумму комиссионных.
24. Создать 2 класса Банк1 и Банк2. В каждом классе есть конструкторы, которые задают срок вклада и сумму начального вклада, функции, вычисляющие выплату в классе Банк1 за 3 месяца -25% годовых, в классе Банк2 -30% годовых. Решить следующую задачу. Вкладчик положил различные суммы в различные банки, определить суммарную выплату, используя дружественную функцию.
25. Имеются ежедневные сведения о курсе доллара по отношению к сому за первую неделю месяца. Определить максимальный курс и средний курс доллара за неделю. Ввод и вывод курсов доллара осуществляется в базовом классе, Вычисление максимального и среднего курса и их вывод в производном.
26. Разработать иерархическую структуру классов для обработки данных. Класс Транспортное средство содержит данные класса: name_transport, конструктор, операцию:PrintInfo, которая распечатывает name_transport. Производный класс : Автомобиль содержит поле auto_name, цена и операцию PrintInfo, которая распечатывает данные базового и производного класса. Напишите главную часть программы, в которой объявите объекты для Автомобиля и для каждого объекта вызовите PrintInfo.
27. Разработать иерархическую структуру классов для обработки данных. Класс Печатное издание содержит данные класса: тип_издания (книга, журнал и т.д.), конструктор, операцию:PrintInfo, которая распечатывает. тип_издания Производный класс :Книга содержит поле name, цена и операцию PrintInfo, которая распечатывает данные базового и производного класса. Напишите главную часть программы, в которой объявите объекты для Книга и для каждого объекта вызовите PrintInfo.
28. Разработать иерархическую структуру классов для обработки данных. Класс Жилище содержит данные класса: вид_жилья (квартира, особняк и т.д.), конструктор, операцию:PrintInfo, которая распечатывает. вид_жилья Производный класс :Квартира содержит поле name(например: пятикомнатная и.т.д.), цена и операцию PrintInfo, которая распечатывает данные базового и производного класса. Напишите главную часть программы, в которой объявите объекты для Квартира и для каждого объекта вызовите PrintInfo.

29. Разработайте Класс Box для коробки. Включите в этот Класс функции инициализации, функции доступа, которые возвращают длины сторон, функцию, которая вычисляют объем. Обхват коробки – это периметр прямоугольника, образованного двумя сторонами. Коробка имеет три возможных значений обхвата. Почтовая длина определяется обхватом плюс расстояние третьей стороны. Упаковка пригодна для пересылки по почте, если какая либо из ее длин меньше 100. Определить дружественную функцию, которая сравнивает объемы двух объектов этого класса и выдает соответствующее сообщение, если они равны и наоборот.
30. Определите класс Параллелипипед прямоугольный.(призма в основании которой - прямоугольник) Данные:a, b, c- ребра параллелипипеда, d – диагональ. (d2=a2+b2+c2) Методы: объем(V=abc) площадь поверхности:(2*(ab+bc+ca). Использовать класс для решения следующей задачи: Требуется построить будку для собаки прямоугольной формы из дерева, стоимость материалов определяется из расчета $10,50 за кв фут (1фут=0,3048 м)
31. Разработайте класс Элипс:Ellipse. Данные a и b – полуоси элипса. (Площадь элипса =πab) Методы: конструктор и Area-площадь. Используйте этот класс для решения следующей задачи. :Необходимо построить овальный плавательный бассейн, полуоси которого задаются 30x40, внутри прямоугольной площади 80x60. Стоимость бассейна $25000. Площадь снаружи бассейна необходимо зацементировать. Стоимость цемента :$50 за кв.фут . Вычислить общую стоимость строительства.
32. Класс Фирма имеет следующие данные сумма продажи, функции-методы: ввода суммы на которую торговый представитель данной фирмы продал товар, функция, определяющая его комиссионные, функция вывода комиссионных. Комиссионные начисляются по следующему правилу: если сумма продаж не превышает 1000 сом, то комиссионные составляют 10% от суммы продаж, если больше 1000 сом, но меньше 3000 сом - 15%, свыше 3000 сом -25%. Решить следующую задачу. У фирмы 2 торговых представителя. Определить комиссионные каждого торгового представителя и общую сумму комиссионных.
33. Создайте класс Шар. Данные: радиус, . Функции: Вычисления объема, Поверхности шара,конструктор по умолчанию, конструктор инициализации. V=4/3 πR3 , P=4πR2 (пл поверхности шара). Использовать этот класс для решения следующей задачи: На покраску поверхности полушарового купола диаметром 5 м идет 6, 5 кг олифы. Сколько олифы нужно для окраски купола заданных размеров?
34. Создать класс Circle данные:радиус, угол сектора, функции: конструктор:,площадь, длина окружности, площадь сектора. В главной части прграммы используйте этот класс для решения задачи: Круглая игровая площадка определяется как объект с радиусом 100футов. Определить стоимость ограждения этой площадки. Стоимость ограждения $2,40 за фут. Площадь поверхности площадки в основном травяная. Один сектор, измеряемый углом в 300 , не является лужайкой. Программа определяет стоимость лужайки для катания: $4,00 за полосу 2x8 (16 кв фута).(Стоимость ограждения= длина окружности*2,40.) Площадь лужайки=: Площадь площадки-площадь сектора. Стоимость лужайки: цена за 1кв фут: площадки для катания *4,00, где цена за 1кв фут: площадки= Площадь лужайки/16. (s сектора -πR2 n/360, где n – дуга в градусах)
35.

7. Учебно-методическое и информационное обеспечение дисциплины

7.1.Список источников и литературы

Основные учебники

1. Шлее, М. Qt 5.3. Профессиональное программирование на C++. В подлиннике / М. Шлее. - СПб.: BHV, 2016. - 928 с.
2. Пахомов, Борис C/C++ и MS Visual C++ 2012 для начинающих / Борис Пахомов. - Москва: СИНТЕГ, 2015. - 518 с.
3. Пахомов, Борис C/C++ и MS Visual C++ 2012 для начинающих / Борис Пахомов. - М.: "БХВ-Петербург", 2013. - 502 с.
4. Полубенцева, М. C/C++. Процедурное программирование / М. Полубенцева. - М.: БХВ-Петербург, 2014. - 448 с.
5. Понамарев, В. Программирование на C++/C# в Visual Studio .NET 2003 / В. Понамарев. - М.: БХВ-Петербург, 2015. - 917 с.
6. Кениг, Э. Эффективное программирование на C++. Практическое программирование на примерах. Т. 2 / Э. Кениг, Б.Э. Му. - М.: Вильямс, 2016. - 368 с.
7. Сидорина, Татьяна Самоучитель Microsoft Visual Studio C++ и MFC / Татьяна Сидорина. - М.: "БХВ-Петербург", 2014. - 848 с.

Дополнительная литература

1. Давыдов, В. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций / В. Давыдов. - М.: БХВ-Петербург, 2014. - 576 с.
2. Довбуш, Галина Visual C++ на примерах / Галина Довбуш , Анатолий Хомоненко. - М.: БХВ-Петербург, 2012. - 528 с.
3. Зиборов, В. MS Visual C++ 2010 в среде .NET / В. Зиборов. - М.: Питер, 2012. - 320 с.
4. Мешков, А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. - М.: БХВ-Петербург, 2013. - 546 с.
5. Панюкова, Т. А. Языки и методы программирования. Создание простых GUI-приложений с помощью Visual C++. Учебное пособие / Т.А. Панюкова, А.В. Панюков. - Москва: Мир, 2015. - 144 с.
6. Роберт, С. Сикорд Безопасное программирование на C и C++ / Роберт С. Сикорд. - Москва: РГГУ, 2014. - 496 с.

7.2.Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимый для освоения дисциплины (модулей)

1. https://www.intuit.ru/search
2. Библиотека все для студента https://www.twirpx.com/
3. Библиотека книг по C++: http://www.ph4s.ru/bookprogramir_1.html
4. Библиотека книг по C++: http://www.libkruz.com/1-41/c.html
5. Языки программирования: http://cat.codenet.ru/Languages

8. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся

8.1. Планы практических (семинарских) и лабораторных занятий. Методические указания по организации и проведению

Лабораторные задания выполняются в среде программирования Visual Studio/

1. Подсчет количества часов, минут и секунд в данном числе суток.
2. Получить элементы таблицы, которые находятся между max и min.
3. Определить слово с максимальным числом букв "а" и количество таких букв "а".
4. Дано слово. Найти сколько раз буква "а" встречается в этом слове. Будет ли это число простым.
5. Дан текст. Установить пробелы вместо символов, номера позиций которых при делении на 4 дают в остатке 3.
6. Заменить данную букву в слове многоточием.
7. Даны слово и буква. Сколько раз эта буква встречается в данном слове.
8. Дано предложение. Определить все слова которые начинаются с заданной буквы.
9. Дан текст. Определить все слова оканчивающиеся на "ая".
10. Дано натуральное число n . Верно ли, что сумма цифр этого числа является нечётной.
11. Дано число n. Верно ли, что это число содержит ровно 3 одинаковых цифры. n<10000
12. Дано натуральное число n. Выбросить из записи числа все чётные цифры. n<10000
13. Перевести число из десятичной в двоичную систему счисления.
14. Перевести число из двоичной в десятичную систему счисления.
15. Дано предложение составить программу располагающую слова в порядке убывания длины слов.
16. Получить n четырёхзначных чисел, в записи которых нет двух одинаковых цифр. n=10
17. Ввод '352', вывод - 'три пять два'.
18. Зашифровать текст, поменяв местами соседние символы.
19. Дано четырехзначное число. Определить входит ли в него цифра 4.
20. Определить является ли заданное шестизначное число счастливым. (Счастливым называю такое шестизначное число, что сумма его первых трех цифр равна сумме его последних трех цифр).
21. Известны год, номер месяца и день рождения человека, а также год, номер месяца и день сегодняшнего дня . Определить возраст человека (Число полных лет).
22. Найти сумму положительных чисел, больших 30 и меньших 100, кратных трем и оканчивающихся на 2,4,8.
23. Найти и вывести все целые числа из промежутка от 1 до 300, у которых ровно пять делителей.
24. Найти и вывести сумму делителей каждого из целых чисел от 50 до 70.
25. Найти и вывести все целые числа из промежутка от 300 до 600, у которых сумма делителей кратна 10.
26. Дано предложение. Определить, сколько в нем одинаковых соседних букв.
27. Дано слово. Добавить к нему в начале и конце столько звездочек, сколько букв в этом слове.
28. Дано предложение. Определить, каких букв в нем больше а или о.
29. Дано предложение. Все пробелы в нем заменить на символ «_».
30. Дано предложение. Найти и вывести любое слова на букву «К».
31. Дано предложение. Найти длину его самого короткого слова.
32. В одномерном массиве имеются только два одинаковых элемента. Найти их.
33. Дан одномерный массив из 20 элементов. Найти пять соседних элементов, сумма значений которых максимальна.
34. Дан одномерный массив из 20 элементов. Переставить первые три и последние три элемента, сохранив порядок их следования.
35. Дан массив целых чисел. Удалить из него все четные элементы, стоящие на нечетных местах.
36. Дан одномерный массив, вставить в массив число 10 после второго элемента.

37. Дан одномерный массив. Определить, есть ли в массиве хотя бы пара одинаковых соседних элементов. В случае положительного ответа определить номера элементов первой из таких пар.
38. Дан массив из 20 элементов. Сформировать два массива размером 10, включив в первый из них элементы заданного массива с четными индексами, а во второй – с нечетными.
39. Используя датчик случайных чисел, заполнить одномерный массив из 20 элементов неповторяющимися числами.
40. С помощью датчика случайных чисел получить 30 целых чисел, лежащих в диапазоне от 0 до 5 включительно, но вывести на экран только нечетные числа.
41. С помощью датчика случайных чисел получить 50 целых чисел, равных 0 или 1, и подсчитать количество единиц и количество нулей.
42. Дано трехзначное число. Найти число, полученное при перестановке первой и второй цифр заданного числа.
43. Дано трехзначное число. Найти число, полученное при перестановке второй и третьей цифр заданного числа.
44. Дано трехзначное число, в котором все цифры различны. Получить шесть чисел, образованных при перестановке цифр заданного числа.
45. Дано трехзначное число. Определить является ли сумма его цифр двузначным числом.
46. Дано трехзначное число. Определить, есть ли среди его цифр одинаковые.
47. Дано четырехзначное число. Определить, равна ли сумма двух первых его цифр сумме двух его последних цифр.
48. Дано натуральное число. Верно ли, что оно заканчивается четной цифрой.
49. Написать функцию. В одномерном динамическом массиве вычислить сумму отрицательных элементов массива и произведение элементов массива, расположенных между максимальным и минимальным элементами.
50. Написать функцию. В одномерном динамическом массиве вычислить сумму положительных элементов массива и произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.
51. Написать функцию. В одномерном динамическом массиве вычислить произведение элементов массива с четными номерами и сумму элементов массива, расположенных между первым и последним нулевыми элементами.
52. Написать функцию. В одномерном динамическом массиве вычислить сумму элементов массива с нечетными номерами и преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом — все отрицательные (элементы, равные нулю, считать положительными).
53. Написать функцию. В одномерном динамическом массиве вычислить сумму элементов массива, расположенных между первым и последним отрицательными элементами и сжать массив, удалив из него все элементы, модуль которых не превышает единицу. Освободившиеся в конце массива элементы заполнить нулями.
54. Написать функцию. В одномерном динамическом массиве найти максимальный элемент массива и сумму элементов массива, расположенных до последнего положительного элемента.
55. Написать функцию. В одномерном динамическом массиве найти минимальный элемент массива и его номер. Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.
56. Написать функцию. В одномерном динамическом массиве найти сумму элементов массива, расположенных между первым и последним положительными

элементами. Преобразовать массив таким образом, чтобы сначала располагались все элементы, равные нулю.

57. Написать функцию. В одномерном динамическом массиве найти номер максимального элемента массива. Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине — элементы, стоявшие в четных позициях.
58. Написать функцию. В одномерном динамическом массиве найти номер минимального элемента массива и произведение элементов массива, расположенных между первым и вторым нулевыми элементами.
59. Написать функцию. В одномерном динамическом массиве найти сумму элементов массива, расположенных между первым и вторым отрицательными элементами. Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает единицу, а потом — все остальные.
60. Написать функцию. В одномерном динамическом массиве вычислить произведение отрицательных элементов массива и сумму положительных элементов массива, расположенных до максимального элемента.
61. Написать функцию. В одномерном динамическом массиве вычислить сумму элементов массива, расположенных до минимального элемента. Изменить порядок следования элементов в массиве на обратный.
62. Написать функцию. В одномерном динамическом массиве вычислить произведение положительных элементов массива. Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.
63. Даны два целочисленных динамических одномерных массива. Размерности массивов равны. В массивах хранятся оценки за две контрольные работы студентов. Каждый элемент массива – это оценка за контрольную работу определенного студента. Студенты разделяются на несколько категорий. Категория «отличники» состоит из студентов, у которых обе контрольные написаны на оценку 5, к категории «хорошо успевающие» относятся студенты, у которых оценка за каждую контрольную - 4 или 5, но студент не отличник. Категорию «успевающие» составляют студенты, у которых хотя бы одна контрольная написана на 3, наконец «неуспевающие» - те студенты, которые имеют 2 хотя бы за одну контрольную. Требуется написать программу определения числа студентов в каждой категории.
64. Число называется совершенным, если оно равно сумме всех своих делителей за исключением его самого. Найти все совершенные числа от 1 до 1000.
65. Мажорирующим элементом в массиве A[1..N] будем называть элемент, встречающийся в массиве более N/2 раз. Легко заметить, что в массиве может быть не более одного мажорирующего элемента. Например, массив
66. 3, 3, 4, 2, 4, 4, 2, 4, 4
67. имеет мажорирующий элемент 4, тогда как в массиве
68. 3, 3, 4, 2, 4, 4, 2, 4
69. мажорирующего элемента нет.
70. Написать функцию для определения, есть ли в массиве мажорирующий элемент, и если есть, то какой.
71. Палиндромом считается строка, которая читается одинаково как слева направо, так и справа налево. Палиндромами являются слова: казак, казак, кок, шалаш… Палиндромами могут быть фразы, обычно считается, что пробелы между словами, знаки препинания и различия между маленькими и большими буквами игнорируются. Палиндромами являются следующие фразы: А роза упала на лапу Азора, Кит на море не романтик…

72. Написать программу, определяющую, является ли заданное предложение палиндромом.
73. Дана действительная квадратная матрица порядка 2n. Получить новую матрицу, переставляя ее блоки размером n x n: а) крест-накрест; б) по часовой стрелке (левый верхний блок становится правым верхним, правый верхний – правым нижним и т.д.).
74. Составить программу транспонирования целочисленной матрицы.
75. Составить программу, которая заполняет квадратную матрицу порядка n натуральными числами 1, 2, 3, … n2, записывая их в нее «по спирали». Например, для n=5 получаем следующую матрицу:
    1 2 3 4 5
    16 17 18 19 6
    15 24 25 20 7
    14 23 22 21 8
    13 12 11 10 9
76. Дана действительная квадратная матрица порядка N (N – нечетное), все элементы которой различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.
77. Разработать проект, который позволяет сортировать заданный линейный массив целых чисел различными методами, например, методом линейной сортировки, пузырька, Шелла и др. Предусмотреть использование не менее трех методов.
78. Элемент матрицы называется 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 матрицы размером (N x M) определить индексы всех ее седловых точек.
79. Составить программу, проверяющую, образуют ли элементы двумерного массива магический квадрат. В магическом квадрате суммы чисел по всем вертикалям, всем горизонталям и двум диагоналям одинаковы.
80. Дана вещественная матрица размером (N x M). Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (или один из них) оказался в левом верхнем углу.
81. В таблице размером (N x N), где N 20, клетки заполнены цифрами случайным образом. Найти маршрут из клетки (1,1) в клетку (N,N), удовлетворяющий следующим условиям: 1) любые две последовательные клетки в маршруте имеют общую сторону; 2) количество клеток маршрута минимально; 3) сумма цифр в клетках маршрута максимальна.
82. Разработать проект, который обеспечивает выполнение операций сложения, вычитания и умножения над матрицами целых чисел. Выбор выполняемой операции осуществляет пользователь.
83. Разработать проект, который позволяет сортировать строковый массив (например, содержащий компьютерные термины) по алфавиту. Обеспечить сортировку внутри группы строк, начинающихся на одну и ту же букву (например, строка, содержащая слово ПРИНТЕР должна предшествовать строке, содержащей слово ПРОГРАММА).
84. Из массива удалить самую длинную цепочку четных элементов. Пример, из массива A[8]: 4 1 4 2 1 2 4 6 должен получиться массив A[5]: 4 1 4 2 1 (самая длинная цепочка четных чисел включает элементы с 6 по 8: 2 4 6).
85. Из массива A удалить те элементы, которые встречаются и в массиве A и в массиве B по крайней мере по 2 раза. Пример, массив A[8]: 3 3 4 5 2 3 5 9, массив B[7]: 1 2 3

4 5 2 5. По 2 раза в обоих массивах встречается только элемент, равный 5. Массив А после удаления примет вид: А[6]: 3 3 4 2 3 9.

86. Из массива А удалить те цепочки четных элементов, в которых есть хотя бы один элемент из массива В. Пример, массив А[9]: 3 2 4 5 2 3 2 6 5, массив В[6]: 1 3 4 7 8 9. Массив А после удаления примет вид: А[7]: 3 5 2 3 2 6 5.

8.2. Методические указания для обучающихся по освоению дисциплины (модулей)

- следить за новинками изданий в области программирования;
- принимать активное участие в работе на занятиях;
- предлагать преподавателю новые формы работы;
- в процессе проведения семинарских занятий накапливать опыт для написания дипломных работ (подготовка рефератов, докладов, выступлений).

8.3. Методические рекомендации по подготовке отчетов по лабораторным работам

Правила оформления курсовой работы

Курсовая работа выполняется в соответствии с индивидуальным заданием.

Курсовая работа должна быть выполнена как работающая программа, позволяющая продемонстрировать выполнение всех функций в любой последовательности.

Пояснительная записка является законченным документом, содержащим основные результаты выполненной работы.

Пояснительная записка должна иметь объем не менее 25 стр. текста (без приложения, шрифт 12) и содержать следующие разделы:

1. Оглавление.
2. Постановка задачи.
3. Назначение и область применения программного продукта.
4. Выбор и обоснование выбора инструментальных средств.
5. Теоретический материал – особенности программирования в данной области. Теоретический материал должен быть переработан применительно к содержанию работы.
6. Структурное описание разработки. Описание форматов данных. Описание используемых структур данных (классов), форматов данных, сопровождаемое необходимыми графическими иллюстрациями (рисунками). Содержательное описание типов данных, структур данных.
7. Описание программы.
8. Описание алгоритмов и методов решения. Функциональное описание. Содержательное описание основных алгоритмов, их особенностей, интерфейсов функций.
9. Описание пользовательского интерфейса.
10. Руководство пользователя.
11. Заключение.
12. Список используемой литературы.
13. Тексты программных модулей (приложение).

Разделы должны иметь порядковый номер: 1, 2 и т.д. Нумерация подраздела включает номер раздела и порядковый номер подраздела, разделенные точкой: 2.1, 2.2 и т.д. Допускаются стандартные сокращения слов.

Заголовки разделов (глав) пишутся прописными буквами и размещаются с выравниванием по центру, шрифт Arial, 14, полужирный, переносы в словах и сокращения не допускаются. Заголовки подразделов записываются с абзаца строчными буквами (кроме первой прописной), шрифт Arial, 12, полужирный, переносы в словах и сокращения не допускаются. Точка в конце заголовка не ставится. Если заголовок состоит из двух предложений, их разделяют точкой. Каждый раздел необходимо начинать с нового листа. Интервал между заголовками и последующим текстом равен 12 пунктам;

Необходимые пояснения к тексту могут оформляться сносками. Сноска обозначается цифрой со скобкой. Если сноска относится к отдельному слову, знак сноски помещается непосредственно у этого слова, если же к предложению в целом, то в конце предложения. Текст сноски располагается в конце страницы.

Иллюстрации могут быть расположены в тексте или в приложениях. Иллюстрации, если их более одной, нумеруются арабскими цифрами в пределах всего документа. Ссылки на иллюстрации дают с сокращенным словом «смотри», например, «см. рис. 12».

Таблицы имеют свой заголовок, который следует выполнять строчными буквами. Сноски к таблицам располагаются непосредственно под таблицей. Ссылки на таблицы в тексте дают аналогично иллюстрациям.

Номера страниц указываются вверху страницы в центре (в колонтитулах), снизу справа указываются фамилия, инициалы студента, номер группы (в колонтитулах), номер на первой странице (титульном листе) не ставится.

На основании проделанной работы и выступления выносится решение об оценке курсовой работы (“отлично”, “хорошо”, “удовлетворительно” или “неудовлетворительно”).

Правила защиты

1. Программа должна быть проверена на разнообразных входных данных.
2. Пояснительная записка и файл программы сдаются за 1-2 дня до защиты.
3. Курсовую работу принимает комиссия. Состав комиссии, утверждается на кафедре.
4. Для защиты курсовой работы необходим доклад на 5 – 7 мин. С использованием презентационного материала (Демонстрация MS Power Point). И демонстрация работы разработанной программы.

При наличии грамматических и синтаксических ошибок в тексте пояснительной записки оценка снижается на 1 балл, более 20 – на 2 балла.

Некоторые практические советы

- Сохраняйте свою работу как можно чаще, секунды потраченные на сохранение сэкономят многие часы.
- Создавайте резервные копии всех рабочих документов: программ, тестовых наборов, текстов пояснительной записки.
- Не храните все копии на одном компьютере или одном носителе.
- Не работайте с безымянными документами.

9. Материально-техническое обеспечение дисциплины

Минимальные требования к материально-техническому обеспечению дисциплины:

- Компьютерный класс
- проектор, экран
- колонки
- программное обеспечение Visual Studio C++

10. Глоссарий

1. **Абстрактный класс** - это класс, содержащий хотя бы один виртуальный метод. Абстрактные классы не бывают изолированными, т.е. всегда абстрактный класс должен быть наследуемым. Поскольку у чисто виртуального метода нет тела, то создать объект абстрактного класса невозможно. Абстрактным классом можно назвать класс, специально определенный для обеспечения наследования характеристик порожденными классами.
2. **Абстракция** - процесс изменения уровня детализации программы. Когда мы абстрагируемся от проблемы, мы предполагаем игнорирование ряда подробностей с тем, чтобы свести задачу к более простой.
3. **Абстракция через параметризацию** - прием программирования, позволяющий, используя параметры, представить фактически неограниченный набор различных вычислений одной программой, которая есть абстракция этих наборов.
4. **Абстракция через спецификацию** - прием программирования, позволяющий абстрагироваться от процесса вычислений описанных в теле процедуры, до уровня знания того, что данная процедура делает. Это достигается путем задания спецификации, описывающей эффект ее работы, после чего смысл обращения к данной процедуре становится ясным через анализ этой спецификации, а не самого тела процедуры. Мы пользуемся абстракцией через спецификацию всякий раз, когда связываем с процедурой некий комментарий, достаточно информативный для того, чтобы иметь возможность работать без анализа тела процедуры. Абстракция через спецификацию позволяет абстрагироваться от процесса вычислений описанных в теле процедуры, до уровня знания того, *что данная процедура делает*. Это достигается путем задания*спецификации*, описывающей эффект ее работы, после чего смысл обращения к данной процедуре становится ясным через анализ этой спецификации, а не самого тела процедуры. Мы пользуемся абстракцией через спецификацию всякий раз, когда связываем с процедурой некий комментарий, достаточно информативный для того, чтобы иметь возможность работать без анализа тела процедуры.
5. **Аспектно-ориентированное сборочное программирование** - разновидность сборочного программирования, основанная на сборке полнофункциональных приложений из многоаспектных компонентов, инкапсулирующих различные варианты реализации.
6. **Декомпозиция программы** - создание модулей, которые в свою очередь представляют собой небольшие программы, взаимодействующие друг с другом по хорошо определенным и простым правилам.
7. **Диаграмма деятельности, Activity diagram** - методология объектно-ориентированного проектирования, предназначенная для детализации особенностей алгоритмической и логической организации системы. При этом каждое действие расчленяется на фундаментальные процессы. На диаграмме деятельности управление осуществляется:
   - либо через потоки управления (явно);
   - либо через определяемые потоки данных (неявно).
8. **Диаграмма классов, Class diagram** - методология объектно-ориентированного проектирования, предназначенная для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования.

9. **Диаграмма компонентов, Component diagram** - метод объектно-ориентированного проектирования, описывающий особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, устанавливая зависимости между компонентами.
10. **Диаграмма кооперации, Collaboration diagrams** - метод объектно-ориентированного проектирования, основанный на графическом представлении всех структурных отношений между объектами, участвующими во взаимодействии. Диаграмма кооперации представляет собой граф, в вершинах которого располагаются объекты, соединенные дугами-связями. При этом дуги могут быть аннотированы сообщениями, которыми обмениваются объекты.
11. **Диаграмма последовательности, Sequence diagram** - методология объектно-ориентированного проектирования, предназначенная для моделирования взаимодействия во времени. Диаграмма последовательности позволяет отслеживать поведение взаимодействующих групп объектов.
12. **Диаграмма развертывания, Диаграмма применения, Диаграмма размещения Deployment diagram** - метод объектно-ориентированного проектирования, отображающий физические взаимосвязи между программными и аппаратными компонентами системы.
13. **Диаграмма состояний, Statechart diagram** - методология объектно-ориентированного проектирования, предназначенная для представления жизненного цикла объектов в реальном или абстрактном мире.
    Диаграмма состояний состоит
    - из множества состояний объектов;
    - из множества событий, сообщающих о перемещении чего-либо в новое состояние;
    - из множества правил переходов, определяющих новое состояние объекта при возникновении тех или иных событий;
    - из множества действий, которые должны быть выполнены объектом, когда он переходит в новое состояние.
14. **Инкапсуляция, Encapsulation** - От лат.In - в + Capsula - ящичек, в объектно-ориентированном программировании - сокрытие внутренней структуры данных и реализации методов объекта от остальной программы. Другим объектам доступен только интерфейс объекта, через который осуществляется все взаимодействие с ним.
15. **Карты класс-ответственность-кооперация, Class-responsibility-collaboration** - Карты класс-ответственность-кооперация - методология объектно-ориентированного проектирования, предназначенная для описания классов и оперирующая понятиями:
    - ответственность - суть - высокоуровневое описание функций, которые выполняет класс;
    - кооперация - суть - ссылка на другие классы, с которыми необходимо кооперироваться для реализации функций.
16. **Класс, Class** - Класс - в программировании - множество объектов, которые обладают одинаковой структурой, поведением и отношением с объектами из других классов.
17. **Компонентное сборочное программирование** - объектно-ориентированное сборочное программирование, основанное на распространении классов в бинарном виде и предоставление доступа к методам класса через строго определенные интерфейсы. Компонентное сборочное программирование поддерживают технологические подходы COM, CORBA, .Net.
18. **Конструкторы** - Эти операции используют в качестве аргументов объекты соответствующего им типа и создают другие объекты такого же типа. Например, операция сложения матриц создает новую матрицу.
19. **Локальность** - означает, что реализация одной абстракции может быть создана и рассмотрена без необходимости анализа реализации какой-либо другой абстракции.

Принцип локальности позволяет составлять программу из абстракций, создаваемых людьми, работающими независимо друг от друга. Один человек может создать абстракцию, которая использует абстракцию, созданную кем-то другим.

20. **Метод объектно-ориентированной декомпозиции** - основной метод объектно-ориентированного программирования, описывающий:
    - статическую структуру системы в терминах объектов и связей между ними;
    - поведение системы в терминах обмена сообщениями между объектами.
21. **Модификаторы** - эти операции модифицируют объекты соответствующего им типа. Например, операция push для стека.
22. **Модульность** - это такая организация объектов, когда они заключают в себе полное определение их характеристик, никакие определения методов и свойств не должны располагаться вне его, это делает возможным свободное копирование и внедрение одного объекта в другие.
23. **Наблюдатели** - эти операции используют в качестве аргумента объекты соответствующего им типа и возвращают элемент другого типа, они используются для получения информации об объекте. Сюда относятся, например, операции типа size.
24. **Наследование, Inheritance** - Наследование - в объектно-ориентированном программировании - свойство объекта, заключающееся в том, что характеристики одного объекта (объекта-предка) могут передаваться другому объекту (объекту-потомку) без их повторного описания. Наследование упрощает описание объектов.
25. **Объект, Object** - Объект - в программировании - программный модуль:
    - объединяющий в себе данные (свойства) и операции над ними (методы);
    - обладающий свойствами наследования, инкапсуляции и полиморфизма.
    Объекты взаимодействуют между собой, посылая друг другу сообщения.
26. **Объектно-ориентированное программирование** - технология программирования, при которой программа рассматривается как набор дискретных объектов, содержащих, в свою очередь, наборы структур данных и процедур, взаимодействующих с другими объектами.
27. **Объектно-ориентированное сборочное программирование** - разновидность сборочного программирования:
    - основанная на методологии объектно-ориентированного программирования; и
    - предполагающая распространение библиотек классов в виде исходного кода (obj) или упаковку классов в динамически компонуемую библиотеку (dll).
28. **Полиморфизм, Polymorphism** - в объектно-ориентированном программировании - способность объекта выбирать правильный метод в зависимости от типа данных, полученных в сообщении.
29. **Примитивные конструкторы** - эти операции создают объекты, соответствующего им типа, не используя никаких объектов в качестве аргументов. Примером такой операции является создание пустого списка.
30. **Процедурная абстракция, процедура** - наиболее известный в программировании тип абстракции. Всякий, кто применял для выполнения функции подпрограмму, реализовывал тем самым процедурную абстракцию. Процедуры объединяют в себе методы абстракции через параметризацию и спецификацию, позволяя абстрагировать отдельную операцию или событие.
31. **Свойство объекта** - в объектно-ориентированном программировании - характеристика объекта. Обычно свойства изменяются с помощью методов.
32. **Программный сниппет** - (англ. snippet — фрагмент, отрывок) в практике программирования — небольшой фрагмент исходного кода или текста, пригодный для повторного использования. Сниппеты не являются заменой процедур, функций или других подобных понятий структурного программирования. Они обычно используются для более лёгкой читаемости кода функций, которые без их использования выглядят слишком перегруженными деталями, или для устранения повторения одного и того же

общего участка кода. Интегрированные среды разработки (IDE) содержат встроенные средства для ввода конструкций языка. Например, в Microsoft Visual Studio, Borland Developer Studio, для этого необходимо ввести ключевое слово и нажать определённую клавишную комбинацию. В IDE Geany существует специальный файл snippets.conf (путь к файлу: /home/user/.config/geany) позволяющий создавать свои сниппеты. Другие программы, такие как Macromedia Dreamweaver и Zend Studio, позволяют использовать сниппеты в Веб-программировании.

33. **Событийно-управляемое программирование** - объектно-ориентированное программирование, при котором задаются реакции программы на различные события.
34. **Спецификация** - описывает соглашение между разработчиками и пользователями. Разработчик берется написать модуль, а пользователь соглашается не полагаться на знания о том, как именно этот модуль реализован, т.е. не предполагать ничего такого, что не было бы указано в спецификации. Такое соглашение позволяет разделить анализ реализации от собственно использования программы. Спецификации дают возможность создавать логические основы, позволяющие успешно "разделять и властвовать".
35. **Технология программирования, Инжиниринг ПО, Software engineering** - дисциплина, изучающая технологические процессы программирования и порядок их прохождения. (см. онлайн-курс "Технология программирования")
36. **Экземпляр объекта, Instance** - в объектно-ориентированном программировании - конкретный объект из набора объектов данного класса. Все экземпляры одного класса имеют одинаковый набор операций.

11. Приложения

Конспект лекций

Функция определения собственных функций

Функция - поименованная часть программы, к которой можно многократно обращаться. Функции бывают *собственные* и *библиотечные*. *Библиотечные* функции, описанные отдельным файлом, который подключается с помощью директивы **include** в начале программы. *Собственные* функции описываются с помощью прототипов функции. Прототип функции задаётся следующим образом:

тип имя функции (список параметров)

Использование прототипов функции является объявлением функции. Прототипы функции записываются перед функцией **main.** При объявлении функции, компилятор важно знать: имя функции, количество и тип параметра, а также тип возвращаемого значения.

Пример: *Нахождение длины окружности по известному радиусу.*

```
#include<iostream.h>
void dlina (float-r); //прототип функции
void main ()
{
float radius;
cout<<”Введите радиус \n”;
cin>>radius;
dlina (radius); //вызов функции
}
void dlina (float r) n-описание функции
{
cout<<”Длина=”;
cout<<3.14*2*r;                тело функции
```

cout<<endl;
}

Данная программа состоит из двух функций: **main(); dlina.** Имя функции, должно быть, объявлено в начале программы. Заголовок функции заканчивающийся **;** называются **прототипом функции** и являются **объявлением функции.**

Описание функции **dlina** состоит из имени функций, за которым в круглых скобках следует аргумент функции с указанием его типа.

Аргументы, стоящие в заголовке функции называются **формальными параметрами функции.** Аргументы, стоящие в операторе вызова функции называются **фактическими параметрами.** За заголовком функции следует тело функции, заключённое в операторные фигурные скобки. Типы аргументов фактических и формальных параметров функции должны совпадать. Конкретное значение, аргумент получает при вызове функции.

В языке С++ функция полностью определяется по имени, по количеству параметр и их типов. При объявлении функции, т.е. при написании прототипа, имена формальных параметров не используются и не рассматриваются компилятором, поэтому их можно отпустить. Функции бывают:

1. функции без аргументов не возвращающие значения;
2. функции без аргументов, возвращающие значения;
3. функции с аргументами, возвращающие значения;
4. функции без аргументов, возвращающие значения.

Ключевое слово **void** в заголовке функции говорит о том, что функция не возвращает никакого значения. Для возвращения значения вызывающую программу использует оператор **return.**

Функции, возвращающие значения имеют следующие свойства:

1. вызов функции, использованный внутри выражения, а не появляются в программе как отдельный оператор;
2. функция вычисляет значения, т.е. результат, который затем может использоваться в выражении;
3. функция возвращает один результат.

Между определением функций, возвращающий и не возвращающий значения существуют два различия:

место типа, возвращающего значений в определение, т.е. результат, может использовать выражение в операторе **return,** не указывается выражение для возвращаемого значения, потому что его нет.

Собственные функции как отдельный файл.

Функцию можно определить в отдельном файле и подключить с помощью директивы препроцессора **include.**

#include “имя файла”

Пример: *Написать функцию для нахождения факториала числа. Файл функции factorial.cpp*

int factorial (int x); ***//прототип функции (объявление)***
int factorial (int x)
{
int f=1;
for (int i=1; i<=x; i++)
*f=f*I;*
return f;

```
}
```
Функцию компилировать нельзя!

```
#include<iostream.h>
#include “factorial.cpp”
void main ()
{
int x=0;
cout<<”Enter x”;
cin>>0;
cout<<”factorial=n<<factorial(a)”;
cout<<endl;
}
```
Т.о. можно выделить следующие свойства функции:

1. функция является программным блоком;
2. функция может быть выполнена в виде подпрограммной, так и в виде самостоятельной функции, пригодного для использования в любой программе;
3. функция может иметь и не иметь аргумента и возвращающего значения;
4. прототип или описание функции предшествует его первому вызову;
5. в теле функции может быть вызвана другая функция;
6. при вызове функции аргумента может быть вызов другой функции или арифметическое выражение.

Область действия и время жизни переменных.

Область жизни – область программы, при выполнении которой переменная существует с памяти компьютера.

Область действия переменной – область программы, где на её идентификатор можно ссылаться.

Названные области не всегда совпадают. Для определения этих атрибутов в С++ имеется **4** спецификатора класса памяти:

auta- автоматический
registr- регистровый
extern- внешний
} ***локальные переменные***

static- статический

auta и **registr** используются для объявления переменных с локальным временем жизни. Такие переменные создаются в блоке в момент их объявления и исчезают при выходе из блока. Спецификатор **auta** устанавливается по умолчанию и в явном виде используется редко. Спецификатор **registr** даёт указание компилятору, сохраняет переменную в высокоскоростном регистре, если её предполагается часто использовать.

Класс памяти **extern** используется для объявления переменных с глобальным временем жизни. Память для них выделяется в момент объявления и сохраняется до окончания программы. Объявление глобальных переменных размещается вне описания какой-либо функции и на них может ссылаться любая функция, объявленная после их объявления.

Спецификатор **extern** для глобальных переменных, также как и **auta** для локальных устанавливается по умолчанию. Аргументы функции в её заголовке являются локальными в теле функции.

Переменные с атрибутом **static** являются локальными, т.е. доступны только в том блоке, где они объявлены, но сохраняются в памяти компьютера до конца исполнения программы.

Перегрузка функций.

Программирование на языке С++ это процесс чувствительный к типам. Любая переменная имеет тип и при действиях с переменными их типы необходимо учитывать. Для функции определены типы возвращаемых значений и типы аргументов. Если при вызове функций используется аргумент непредусмотренного типа, компилятор может попробовать сделать приведение типов по правилам или зафиксировать ошибку.
С++ позволяет определить несколько функций с одним и тем же именем, но с различным набором аргументов, что и называется **перегрузкой функции.**

Правила перегрузки функций.

1. можно перегружать функции, у которых различаются аргументы по типу, по количеству, по порядку следования.
2. нельзя перегружать функцию, различающийся только типом возвращающегося значения, т.к. при выборе функции, компьютер не получает указаний для выбора варианта.
3. внутри тел перегружать функцию могут выполняться отличные друг от друга алгоритмы.

Функции с переменным числом параметра.
При вызове функции с переменным числом параметра в вызове этой функции задаётся любое требуемое число аргументов. В объявлении и определении такой функции переменное число аргументов задаётся **многоточием в конце стека** формальных параметров или стека типов аргумента. Все аргументы, заданные в вызове функции размещаются в стеке. **Стек** - структурированная область памяти.
Количество формальных параметров объявленных функций определяется числом аргумента, которое берутся из аргумента и присваиваются формальным параметром.

Пример 1:
```
#include<iostream.h>
int sum(int, int);
int sum(int, int, int);
void main ()
cout<<sum(5,6)<<endl;
cout<<sum(1,2,3)<<endl;
}
int sum (int x, int y)
{
return x+y;
}
int sum (int x, int y, int z)
{
return x+y+z;
}
```

Пример 2: *Возведение не целого числа в целочисленную степень:*
```
#include<iostream.h>
#include<math.h>
double gosha(int, double);
double gosha(double, int);
void main()
{
```

```
cout<<gosha(5, 3.5)<<endl;
cout<<gosha(2.3, 3)<<endl;
}
double gosha (int x, double y)
{
return pow (y,x);
}
double gosha (double x, int y)
{
return pow (x,y);
}
```

Пример 3: *Написать функцию с аргументом, возвращающий значения, которое возводит в кв. число, если оно чётное и в куб, если оно нечётное. Функцию оформить, как отдельный файл.*

```
#include<iostream.h>
#include<math.h>
# include ”cub.cpp”
void main()
{
int b=0;
cout<<”Enter chislo”<<endl;
cin>>b;
cout<<cub(b)<<endl;
}
```

Функция:

```
int cub(int);
int cub(int cc)
{
int x;
x=0;
if (cc%2==0)
{
x=pow (cc, 2);
}
else if (cc%2!=0)
{
x=pow(cc, 3);
}
return x;
}
```

Для обеспечения удобного способа доступа аргументам функций с переменными числовыми параметрами имеются **3** макроса:

va_start
va_arg
va_end

Находятся в заголовочном файле **stdarg.h**

Эти макросы указывают на то, что функция разработанная программистом имеет некоторое число обязательных аргументов, за которое следует переменное число необязательных аргументов.
Рассмотрим применение этих макросов для обработки параметров функций, вычисляющей среднее значение произвольной последовательности целых чисел. Поскольку функция имеет переменное число параметров, будем считать концом списка значение равное -1. Поскольку в списке должен быть хотя бы один элемент, у функции будет один обязательный параметр.

```
#include<iostream.h>
#include<stdarg.h>
double avg (int,...);
void main ()
{
cout<<avg (2,3,4,-1)<<endl;
cout<<avg(5,2,3,8,7,-1)<<endl;
}
```

Функция:

```
double avg (int x,...)
{
int i, j, s;
i=0; j=1; s=x;
if (x==-1)
return 0;
va_list a;
va_start (a,x);
while((i=va_arg(a,int) //выборка очередного парам-ра и проверка на конец списка
{
s+=i; i++;
va_end (a);
return s/j;
}
```

Макрос va_start предназначен для установки аргумента, а на начало списка необязательных параметров и имеет вид функций с двумя параметрами. Параметр **x** должен быть членом **va_list.**
Макрос va_art также имеет вид функций с двумя параметрами и обеспечивает доступ к текущему параметру. Это макрокоманда извлекает значение типа **int** по адресу заданному указателю **a** и увеличивает значение указателя на длину. Т.о. параметр **a** будет указывать на следующий параметр вызываемой функции.
Макрос va_end используется по окончанию обработки всех параметров функции и устанавливается указатель на 0.

Указатели.
Значение переменной записывается в ячейку памяти. **Указатель** - целочисленная переменная, содержащая адрес другой переменной. Как и всякая переменная, указатель должен быть объявлен:
int x; *xP
Знак ***** показывает, что объявлен указателем, т.к. указатель есть адрес, а адреса распределяет сам компьютер. Указатель не может быть инициализирован. Присваивание указателей адреса переменной называется **операцией адресаций.**

```
int i;
```

i=0;
iP=&I;
Знак ***&*** перед переменной возвращает её адрес.
Знак ***** перед указателем возвращает значение переменной, на который ссылается указатель.
*cout<<i; //**0***
*cout<<*iP; //**0***
cout<<iP; адрес
Указателем на массив является само имя массива, а также указатель на его первый элемент.
int x[5]={1,2,3,4,5};
*int *y;*
y=x; //y=&x[0];
cout<<y;
*cout<<1; //**первые элементы***

Передача аргументов по ссылке.
Все рассмотренные ранние функции после их исполнения не изменяли значения переменных использованных в качестве аргументов. Это объясняется тем, что в функцию передаётся не сама переменная, а её временная копия, которое уничтожается после выхода из функций, такой способ называется **передача переменных по значению.**
Копии переменных требуют дополнительную память, что существенно, если аргументов много. Бывает необходимо по смыслу алгоритма изменять значение переменных фактических аргументов. Обе проблемы можно снять, если передать функции не копии, а сами переменные, такой способ называется **передачей аргументов по ссылке.** На передачу по ссылке указывается знак **&.** Аргументы, изменённые в процессе выполнения функции, сохраняют новые значения для дальнейшего использования.

Пример 1:
#include<iostream.h>
void fun (int &x, int &y);
void main()
{
int x1, y1;
x1=y1=0;
cin>>x1>>y1;
fun(x1,y1);
cout<<x1<<endl;
cout<<y1<<endl;
}
Функция:
void fun(int &x, int &y)
{
*x=x*2;*
*y=y*2;*
}

Пример 2: *Рассмотрим предыдущий пример, но с использованием указателей.*
#include<iostream.h>
*void fun (int *x, int *y);*
void main()
{

```
int x1, y1;
x1=y1=0;
cin>>x1>>y1;
fun(&x1, &,y1);
cout<<x1<<endl;
cout<<y1<<endl;
}
```

Функция:

```
void fun (int *x, int *y)
{
*x=*x*2;
*y=*y*2;
}
```

Т.о. передача аргументов может осуществляться по значению, по ссылке и с помощью указателей.

Массив в качестве аргументов функций.

Пример 1: *Реализация функций для нахождения минимального элемента массива.*

```
#include<iostream.h>
int min (int a[ ]);
void main()
{
        for (int m=a[0], i=1; i<5; i++)
        if(a[i]<m)
m=a[i];
return m;
```

Пример 2: *Реализация функций и формирование массива с из максимальных элементов массивов a и b.*

```
#include<iostream.h>
void max_v (int n, int *x, int *y, int *z);
void main ()
{
int a[5]={3,4,1,2,7,};
int b[5]={8,1,6,9,4,};
int c[5];
max_v (5, a, b, c);
for (int i=0; i<5; i++)
cout<<c[i]<<”\t”;
}
void max_v (int n, int *x, int *y, int *z)
{
for (int i=0; i<n; i++)
        if (x[i]<y[i])
       z[i]=y[i];
else
z[i]=x[i];
}
```

Динамическое распределение памяти. Динамические массивы.
В языке C++ предусмотрена возможность динамически выделять и удалять память для переменных. Эти действия выполняют операторы **new, delete** в сочетании с указателем:
*int *ip; **//объявление указателя***
ip=new int

Оператор **new** возвращает указатель заданного типа на выделенную память достаточную для хранения переменной. После выделения памяти и использования переменной необходимо освободить память следующим образом: **delete ip;**
Для массива выделение и освобождение памяти выглядит следующим образом:
*int *mp;*
mp=new int[n];
……
delete [] mp;
Инициализация массивов при динамическом выделении памяти не производится. Т.о. динамическое распределение памяти даёт возможность не задавать заранее размеры массива.

***Пример 1:** Найти сумму элементов одномерного массива. Размерность массива задаётся в ходе выполнения программ.*

```
#include<iostream.h>
void main()
{
int x, n, *iP;
x=0;
cout<<"Enter size";
cin>>n;
iP=new int[n];
for(int i=0; i<n; i++)
cin>>iP[i];   //cin>>*(iP+i);
for(int i=0; i<n; i++)
x+=iP[i];    //x+=*(iP+i);
cout<<x<<endl;
delete []iP;
}
```

***Пример 2:** Рассмотрим этот же пример с использованием функции.*

```
#include<iostream.h>
int sum(int n, int iP[]);
void main()
{
int n, *iP;
cout<<"Enter size";
cin>>n;
iP=new int[n];
for(int i=0; i<n; i++)
cin>>*(iP+i);
cout<<sum(n, iP);
delete iP [];
int sum(int n, int iP[])
{
```

```
int x=0;
for (int i=0; i<n; i++)
x=x+*(iP+i);
return x;
}
```

Двумерные динамические массивы.

Особенности использования массивов в С++ заключается в том, что по имени массива нельзя определить размерность и размеры по определению многомерный массив не существует. Он рассматривается как одномерный массив, каждый элемент которого представляет собой массив. При необходимости передать в функцию, многомерный массив можно указать значение каждой размерности, что не принимается компилятором.

Указанные ограничения на возможность применения в качестве параметров можно обойти несколькими путями: **1)** подмена многомерного массива одномерным и имитация внутри функции доступа к многомерному массиву; **2)** использование вспомогательных указателей на массивы. Одномерные массивы служат для представления строк матрицы, т.к. вспомогательные массивы являются одномерным, то размер может быть опущен при написании формальных параметров, тем самым появляется возможность обработки в теле функции в двумерных, а в общем случае многомерных с изменяющимися размерами. Конкретные значения размеров должны передаваться функциям ими с помощью дополнительных параметров и с использованием глобальных переменных.

Пример: Рассмотрим пример для нахождения количества чётных элементов, расположенных на главной диагонали матриц. В функцию будем передавать размерность и указатель на массив, а внутри функции будем рассматривать этот указатель как указатель кв. матрицу, заданную размерностью.

```
#include<iostream.h>
int fun (int s1, int s2, int *iP);
void main ()
{
int s1, s2;
int *iP;
s1=s2=0;
cout<<”Enter size”;
cin>>s1>>s2;
iP=new  int[s1*s2];
for(int i=0; i<s1; i++)
for(int j=0; j<s2; j++)
cin>>*(iP+i*s2+j);
cout<<fun (s1, s2, iP);
delete []iP;
}
int fun(int s1, int s2, int *iP)
{
int k=0;
for(int i=0; i<s1; i++)
for(int j=0; j<s2; j++)
if (i==j && *(iP+i*s2+j)%2==0)
k++
return k;
}
```

Объектно-ориентированное программирование. Структуры данных.
Структура - набор данных, где данные могут быть разного типа, т.е. структуры позволяют объединить данные различного типа, например, структура может содержать несколько переменных типа **int** и несколько переменных типа **char.** Переменные, которые содержатся в структуре, называются **элементами** или **полями структуры.** Структуры определяются с помощью ключевого слова **struct**.

Пример:
```
struct student
{
char name [50];
int course;
int age;
};
```

Т.о. определили структуру, в которую входят переменные **course, age** и массив **name.** В описании **student** является **шаблоном структуры struct student- тип данных.** После описания структуры необходимо ставить **;** (точка с запятой)
Чтобы использовать структуры необходимо объявить переменные **struct student**
struct student s1, s2;
Для получения доступа к полям структуры используется операция **.** (точка)
s1.agp
В языке С++ есть возможность объявить переменные структуры при описании структуры
}s1, s2;
Пример 1:
```
#include<iostream.h>
 struct student
{
char name [50];
int course;
int age;
};
void main ()
{
struct student s1, s2;
cin>>s1.name;
cin>>s1.course;
cin>>s1.age;
s2=s1;
cout<<"name-"<<s2.name;
cout<<"age-"<<s2. age;
cout<<"course-"<<s2.course;
```

Структуры могут быть объединены в массивы структур. Объявление в массивы структур делается аналогично массивов переменных. Например: если нужно хранить информацию 10 студентов, то объявление массивов будет выглядеть след. образом:
struct student s [10];
*s [4].age-**возраст пятого студента***
s [0].course
Т.о. структура – производный тип данных, определённый программистом. Имя структуры. Имя структуры – идентификатор типа.

Объединения.
Структура данного объединения подобно структуре, однако, в каждый момент времени может использоваться только один из его компонентов. Шаблон объединения может использоваться в след. виде:
union
{
имя типа комп 1;
имя типа комп 2;

Поля структуры в оперативной памяти одно за другим, который перечислены в описании. В поля объединений размещаются, начиная с одного места памяти и, следовательно, накладываются друг за друга. Доступ к компонентам объединения осуществляется тем же способом, что и компонентам структур.
Пример:
```
#include<iostream.h>
union Gosha
{
int a;
char b;
}x;
void main ()
{
x.a=123;
cout<<x.a;
}
```

Объединения применяются для минимизации, используемого объёма памяти, если в каждый момент времени только один объект из многих является активным.
В языке С++ имеется тип данных подобный записи с вариантами, называемой **переменной структурой,** которое может быть реализованной с использованием комбинации структуры и объединения. Т.о. переменная структура- это такая структура, компоненты которой меняются в зависимости от значения метки активного компонента. В общем случае переменные структуры могут состоять из **3** частей: **1)**набор общих компонентов; **2)**метки активного компонента; **3)**части, с меняющимися компонентами.

Пример: *Реализуем программу, которое в зависимости от введённой метки будет находить определённые геометрические величины.*
```
#include<iostream.h>
define CIRCLE 1 (окружность)
define RECT 2 (прямоугольник)
define TRIANGLE 3 (треугольник)
struct figure
{
int area, perimeter; //общие компоненты
int type; //метка активного компонента
union
{
int radius;
```

```
int a[2];
int b[3];
}x;
}y;
void main()
{
cout<<"Enter metku";
cin>>y.type;
switch (y.type)
{
case CIRCLE:
cout<<"Enter radius";
cin>>y.x.radius;
break;
case RECT:
cout<<"Enter storonu";
cin>>y.x.a[0]>> y.x.a[1];
y.area=y.x.a[0]* y.x.a[1];
cout<<"area"<<y.area;
break;
case TRIANGLE:
cout<<"Enter storonu";
cin>>y.x.b[0]>>y.x.b[1] >>y.x.b[2];
y.perimetr=y.x.b[0]+y.x.b[1]+y.x.b[2];
cout<<"Perimetr="<<y.perimetr;
break;
default: cout<<"Error";
}
}
```

Классы

Класс - производный, структурированный тип, введённый программистом на основе уже существующих типов.

Механизм классов позволяет создавать типы в полном соответствии с принципами абстракции данных, т.е. класс задаёт некоторую структурную совокупность типизированных данных и позволяет определить набор операции этими данными. Т.о. класс – это производный тип данных, определённый программистом.

Имя класса - это спецификатор типа. Класс не содержит данных, он описывает общую структуру. Класс можно определить с помощью след. конструкции:

```
Ключ класса   имя класса
{
Список компонентов;
};
```

где ключ класса это одно из служебных слов: **struct, union, class,** где имя класса это произвольно выбираемый идентификатор, а список компонентов это определения и описания типизированных данных и принадлежащих классу функции. Инициализация элементов класса при его объявлении запрещена, т.к. выделение памяти происходит при объявлении объекта. Принадлежащие классу функции, будем называть **методами класса.** Данные классы будем называть компонентными данными или элементами данных класса.

В соответствии с правилами в С++ все компоненты введённого с помощью слова struct являющиеся общедоступными. Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа. Существуют **3** спецификатора доступа: **public(общедоступный), private(собственный), protected(защищённый).**
Появление любого из спецификаторов доступа в тексте определения, либо до другого спецификатора доступа. Все компоненты класса имеют указательный статус. По умолчанию (если не указано спецификатор доступа) компоненты класса **struct** и **union** являются **общедоступными (public),** а если класс определён с помощью ключевого слова **class,** то компоненты будут **собственными (private).**
Пример: *Реализуем класс треугольник, который будет включать две функции:* ***1)****нахождение периметра треугольника;* ***2)****площадь.*

```
#include<iostream.h>
class treug
{
public:
int a, b, c;
int perimetr ();
double plosh ();
};
void main ()
{
treug  x;
cin>>x.a>> x.b >> x.c;
cout<<x.perimetr ()<<endl;
cout<< x.plosh ()<<endl;
}
int treug ::perimetr ()
{
return a+b+c;
}
double treug ::plosh ()
{
return (a*b)/2;
}
```

Конструкторы и деструкторы.
Для инициализации объектов класса его определения можно включать специальную компонентную функцию, называемую **конструктором.** Формат определения конструктором в теле класса может быть следующим:

```
имя класса (параметр)
{
операторы тела констр;
}
```

Имя этой компонентной функции (конструкции) по правилам языка С++ должно совпадать с именем класса. Такая функция автоматически вызывается при определении или размещении памяти каждого объекта класса. Основное назначение конструктора – это **инициализация объекта.** В соответствии с синтаксисом языка для конструктора не определяется тип возвращающего значения, даже тип **void** недопустим. С помощью параметров к конструктору могут быть переданы любые данные, необходимые для создания инициализации объектов класса.

Динамическое выделение памяти для объектов какого-либо класса создают необходимость в освобождении этой памяти при уничтожении объекта.
Желательно, чтобы освобождение памяти происходило автоматически и не требовало вмешательства программиста. Такую возможность обеспечивает специальный компонент класса – **деструктор (разрушитель объекта).**
Синтаксис деструктора:
~ имя класса () // ~тильда
{
операторы тела деструктора;
}

Название деструктора начинается с символа **~** за которым без пробела идёт имя класса. При использовании деструктора необходимо, что **1)** у деструктора не может быть параметра; **2)** деструктор не имеет типа, возвращающего значения; **3)** вызов деструктора выполняются автоматически.

Пример: *Рассмотрим класс цилиндр, который содержит функцию: для нахождения площади основания и функцию для нахождения объёма.*

```
#include<iostream.h>
#include<math.h>
#define PI 3.1415
class cilindr
{
public:
int r, h;
double area ( int);
double obem ();
cilindr ();
};
void main ()
{
cilindr x;
cout<<x.obem ()<<endl;
}
cilindr ::cilindr ()
{
cin>>r>>h;
double cilindr ::area (int r)
{
return PI*pow(r,2);
}
double cilindr ::obem ()
{
return area(r)*h;
}
```

Использование указателей для доступа компонентом класса.
Другой способ доступа к элементам объекта некоторого класса предусматривает явное использование указателя на объект класса и операции косвенного выбора компонента.

- *> операция косвенного выбора указателя*

указатель _ на _ объект _ класса - > имя _ элемента

Указательный объект класса позволяет вызывать принадлежащий классу функций для обработки данных того объекта, который адресуется указателем.

: *формат вызова функций*

указатель _ на _ объект _ класса _ - > обращение к функциям

Пример: *Рассмотрим класс, описывающий товары на складе магазина. Компонентами класса будут: название товара; закупочная цена (оптовая цена); торговая наценка; функция ввода данных о товаре; функция вывода сведений о товаре с указанием розничной цены.*

```
#include<iostream.h>
struct goods
{
char name [40];
float price;
static int percent;
void Input ()
{
cout<<"Название:";
cin>>name;
cout<<"Закупочная цена";
cin>>price;
}
void Display ()
{
cout<<name;
cout<<"Розничная цена:";
cout<<price*(goods ::percent*0.01);
}
};
int goods ::percent=12; //инициализация статического компонента
void main ()
{
goods wares [5];
int k=5;
cout<<"Enter сведения о товарах";
for (int i=0; i<k; i++)
wares [i].Input();
cout<<"Товары при наценки"<<wares [0].percent<<".\n";
for ( i=0; i<k; i++)
wares [i].Display();
goods ::percent=0;
cout<<"Товары при наценки"<<wares [0].goods ::percent<<"%\n";
goods ::*pGoods=wares; //объявляем goods
for ( i=0; i<k; i++)
pGoods++- >Display ();
}
```

Данной программе торговая наценка определена как статический компонент класса. Статические компоненты класса не дублируются при создании объекта класса, т.е. каждый статический компонент существует в единственном экземпляре. Доступ к статическому компоненту возможен только после его инициализации. Инициализация размещается в

глобальной области программ. Только при инициализации статический компонент класса получает память и становится доступным. Для обращения к статическому компоненту можно использовать две формы записи:
имя класса ***::*** *имя компонента*
*имя объекта****.*** *имя класса* ***::*** *имя компонента*
*имя объекта****.*** *имя компонента*
Для иллюстрации разных способов доступа к компонентам класса определён указатель ***pGoods** на объекты класса **goods.** Он инициализирован значением адреса первого элемента, массива объекта. В цикле указатель с операцией - > используется для вызова компонентной функции **Display.** После каждого вызова указатель изменяется, т.е. настраивается на следующий элемент массива.

Дружественные функции и их возможности.
Одним из важных принципов языка С++ является защита данных от несанкционированного использования. Реализуется этот принцип благодаря режиму доступа, составляющей класса. Элементы класса с режимом доступа **private** могут использ. только функциями этого класса, но возникают ситуации, когда необходимо, чтобы к приватной части класса имело доступ функция, не являющаяся составляющей класса. Для этого необходимо использовать механизм дружественных функций.
Дружественные функции класса называется функция, которая, не являясь его компонентом, имеет доступ к его собственным компонентам.
Функция не может стать другом класса “без его согласия”. Для получения прав “друга”, функция должна быть описана в теле класса со спецификатором **friend** именно при наличии такого описания, класс предоставляет функции право доступа к собственным компонентам.
Дружественная функция:

1. не может быть компонентной функцией того класса по отношению к которому определяется как дружественные;
2. может быть глобальной функцией;
3. может быть компонентной функцией другого, ранее определённого класса;
4. может быть дружественной по отношению к нескольким классам.

Использование механизма дружественных функций позволяет упростить интерфейс между классами.

Пример: *Опишем класс “круг”, который включает в себя функцию для нахождения площади, а также дружественную функцию для нахождения площади кольца.*

```
#include<iostream.h>
#include<math.h>
#define PI 3.1415
class ring
{
private:
double area (int);
friend double area_k (int, int);
};
double ring ::area_(int r)
{
return PI*pow (r,2);
}
double area_k (int x, int y);
{
```

```
ring p;
return p.area (x)-p.area (y);
}
void main ()
{
        double s, rp, rm;
cout<<"Enter radius";
cin>>rp>>rm;
s=area_k (rb, rm);
cout<<s<<endl;
}
```

Дружественные классы.

Класс может быть дружественным другому классу, это означает, что все компонентные функции класса являются дружественными для другого класса. Другой класс должен быть определён в не тела класса, предоставляющего дружбу.

```
class x2
{
friend class x1;
……
};
class x1
{
void f1 ();
void f2 ();
……
};
```

В данном примере функции **f1** и **f2** из класса **x1** являются друзьями **x2.** Все компоненты класса соответственно доступны в дружественном классе.

Пример: *Даны два класса. Существует класс массив и дружественный ему класс чётный массив.*

```
#include<iostream.h>
class massiv
{ public :
friend class ch_massiv;
massiv ();
double sr_arif ();
private :
int x[10];
double summa ();
}obj 1;
class ch_massiv
{public :
double sr_arif ();
private :
double summa ();
int kol_vo ();
}obj 2;
massiv ::massiv ()
{
```

```
cout<<”Enter elementy”;
for (int i=0; i<10; i++)
cin>>x[i];
}
double massiv::summa ()
int s=0;
for (int i=0; i<10; i++)
s+=x[i];
return s;
}
double massiv:: sr_arif ();
{
return summa ()/10;
}
double ch_massiv::summa ()
{
int s=0;
for (int i=0; i<10; i++)
if (obj1. x[i]%2==0)
s+=obj1. x[i];
return s;
}
double ch_massiv::kol_vo()
{
int k=0;
for (int i=0; i<10; i++)
if (obj1. x[i]%2==0)
k++;
return k;
}
double ch_ massiv:: sr_arif ()
{
return summa()/kol_vo();
}
void main ()
{
cout<< “sr_arif=”<<obj1. sr_arif();
cout<< “sr_ar.ch=”<<obj2. sr_arif();
}
```

Механизм наследования класса.

Каждый объект является конкретным представлением класса. Объекты одного класса имеют разные имена, но одинаковые по типам и внутренним именам данные. Объектом данного класса для обработки своих данных доступны одинаковые компонентные функции класса и одинаковые операции, настроенные на работу с объектами класса. Т.о. класс выступает в роли типа, позволяющего вводить нужное количество объекта, имена, которых программист выбирает сам. Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объекта, соответствующая заранее предусмотренной иерархии классов. Иерархия классов позволяет определять новые классы на основе уже имеющихся. Имеющиеся классы обычно называются **базовыми** (иногда называют порождающие, родительские), а новые классы, формируемые на основе

базовых, называют **производными** (или классами потомками, наследниками, или дочерними).

Производные классы получают в наследство данные и методы своих базовых классов и кроме того могут пополняться собственным компонентом. Наследуемые компоненты не перемещаются в производный класс, а остаются в производных классах.

Сообщение, обработку которого не могут выполнить методы производного класса, автоматически передаётся в базовый класс. Если же для обработки сообщения нужны данные, отсутствующие в производном классе, то их пытаются автоматически отыскать в базовом классе.

При наследовании некоторые имена компонентных функций или компонентных данных базового класса могут быть по- новому определены в производном классе. В этом случае, соответствующие компоненты базового класса становятся недоступными из производного класса. Для доступа из производного класса к компонентам базового класса, имена которых повторно определены в производном классе исполняется операция **уточнения области видимости** (**::**).

Любой производный класс может в свою очередь становиться базовым для других классов и т.о. формируется направленный **граф** иерархии классов и объектов.

В иерархии производный объект наследует разрешённые для наследования компоненты всех базовых объектов, другими словами у объекта имеется возможность доступа к данным и методам всех своих базовых классов.

Наследование в иерархии класса может отображаться в виде дерева, допускается множественное наследование, это когда некоторый класс может наследовать компоненты некоторым несвязанных между собой базовых классов.

При наследовании классов важную роль играет статус доступа компонента. Для любого класса все его компоненты лежат в области его действия, тем самым любая принадлежащая к классу функция может использовать любые компонентные данные и вызывать любые принадлежащие к классу функции.

При создании иерархии классов, важно помнить следующее:

1. собственные методы и данные класса доступны внутри того класса, где они определены и также доступны друзьям **(private)**;
2. защищённые компоненты доступны внутри класса, в котором они определены и также доступны во всех производных классах **(protected)**;
3. общедоступные компоненты класса видимы из любой точки программы **(public)**.

Определение производного класса.

В определении и описании производного класса приводится список базовых классов, из которых он непосредственно наследует данные и методы. Между именем нового класса и списком базовых классов помещается **двоеточие :** например, при след. определении:

class S: x, y, z
{
…….
}

class S является наследником классов **x, y, z.** По умолчанию из базовых классов наследуется методы и данные со спецификаторами доступа **public** и **protected.** В новом классе эти унаследованные компоненты получают статус доступа **private,** если новый класс определён с помощью ключевого слова **class** или получает статус доступа **public,** если класс определён с помощью ключевого слова **struct,** т.е.

class B
{protected:

```
int t;
public:
char U;
};
class  E:B
{…};  //t, U- private
struct S:B // t, U- public
{…};
```

Явно изменить статус доступа при наследовании можно, используя спецификаторы доступа, которые указываются в описании производного класса, непосредственно перед нужными именами и базовых классов, т.е.

```
class  M :protected B
class  P: public B
{…};  //t, U- protected, U-public
class  D :private B
{…};  //t, U- private
struct F: private B
{…};  //t, U- private
```

также необходимо помнить, что ни базовый класс, ни производный, не могут быть объявлены с помощью ключевого слова **union,** т.е. объединения не могут использоваться при построении иерархии классов.

Пример: Создадим класс и производные от него классы конус и цилиндр.

```
#include<iostream.h>
#define PI 3.1415
class krug
{
protected:
double ploshad (double);
};
class cilindr :krug
{public:
void obem_cilindr ();
cilindr ();
protected:
int h;
int r;
};
class konus :krug
{public:
double obem_kon();
konus();
protected:
int h;
int r;
};
cilindr::cilindr()
{
cout<<"Enter radius";
cin>>r;
cout<<"Enter h cilindr";
cin>>h;
```

```
}
konus::konus()
{
cout<<"Enter r konusa";
cin>>r;
cout<<"Enter h konusa";
cin>>h;
}
double krug::ploshad (double rr)
{
return PI*rr*rr;
}
void cilindr::obem_cilin()
{
 cout<<"obem cilindr="<<ploshad.(r)*h<<endl;
}
double konus:: obem_kon();
{
return ploshad(r)*h/3;
}
void main ()
{konus obj3;
cilindr obj2;
cout<<"obem konusa=" obj3.obem_kon()<<endl;
obj2.obem_cilin();
```

Множественное наследование.

В С++ производный класс может быть порожден из любого числа непосредственных базовых классов. Наличие у производного класса более чем одного непосредственного базового класса называется **множественным наследованием.**

Синтаксический множественный наследователь отличается от единичного наследования списками баз, состоящим более чем из одного элемента, например:

```
class A
{
};
class B
{
};
class C :public A, public B
{
};
```

При создании объектов представителей производного класса, порядок расположений непосредственных базовых классов списки баз определяет очередность вызова конструкторов. Этот порядок также влияет на очередность вызова деструкторов при уничтожении этих объектов.

При множественном наследовании необходимо учитывать одно ограничение, согласно которому, одно и то же имя класса не может входить более одного раза. Список баз при объявлении производного класса это означает, что в наборе непосредственных базовых классов, которые участвуют в формировании производного класса не должно встречаться повторяющихся элементов. Вместе с тем один и тот же класс может участвовать в формировании производного нескольких непосредственных базовых классов данного производного класса. Т.о. для не прямых базовых классов, участвующих в формировании

производного класса не существует никаких ограничений на количество вхождений объявлений производного класса.

Пример:

```
#include<iostream.h>
#define PI 3.1415
class A
{
protected:
double ra;
double area (double);
};
class B :public A
{
public:
B();
protected:
double rb, hb;
double volume1(double, double);
};
class C :public B
{public:
C();
protected:
double rc, hc;
double volume2(double, double);
};
class D :public C
{
public:
void print ();
};
B::B()
{
rb=1;
hb=1;
}
C::C()
{
rc=3;
hc=3;
}
double A::area(double r)
{
return PI*r*r;
}
double B:: volume1(double rb, double hb)
{
return area(rb)*hb;
}
double C:: volume2(double rc, double hc)
{
```

```
return volume1(rc, hc)/3;
}
void D::print ()
{
cout<<"cilindr=" <<volume1 (rb, hb)<<endl;
cout<<"konus="<< volume2 (rc, hc)<<endl;
}
void main ()
{
D x;
x.print ();
}
```

Виртуальные функции, раннее и позднее связывание.

Во время раннего связывания, вызывающий и вызываемый методы связываются при первом удобном случае, обычно при компиляции.

При позднем связывании вызываемого и вызывающего метода они не могут быть связаны во время компиляции, поэтому реализован спец. механизм, который определяет, как будет происходить связывание вызываемого и вызывающего метода, когда вызов будет сделан фактически.

Очевидно, что скорость и эффективность при раннем связывании выше, чем при использовании позднего связывания, в то же время позднее связывание обеспечивает некоторую универсальность связывания.

Виртуальный метод – метод, который будучи описан в потомках замещает собой соответствующий метод везде, даже в методах, описанных для предка, если он вызывается для потомка, т.е. виртуальные методы сущ. Для того, чтобы наследник ввел себя отлично от предка, сохраняя при этом свойство совместимо с ним.

Адрес виртуального метода известен только в момент выполнения программы, когда происходит вызов виртуального метода, его адрес берётся из таблицы виртуального метода своего класса. Т.о. вызывается то, что нужно.

Преимущество применения виртуального метода заключается в том, что при этом используется, именно механизм позднего связывания который допускает обработку объектов, тип которых неизвестен во время компиляции.

Виртуальные методы описываются с помощью ключевого слова **virtual** в базовом классе. Это означает, что в производном классе этот метод может быть замещен методом более подходящим для этого производного класса. Объявленный виртуальным в базовом классе метод останется виртуальным для всех производных классов. Если в производном классе виртуальный метод не будет переопределен, то при вызове будет найден метод с таким именем вверх по иерархии классов.

Пример:

```
#include<iostream.h>
class vehicle
{
int wheels;
float weight;
public:
virtual void message()
{
cout<<"Транспортное средство\n";
}
};
class car:public vehicle
```

```
{
int passenger_load;
public:
void message ()
{
cout<<"Легковая машина";
}
};
class truck: public vehicle
{
int passenger_load;
float payload;
public:
int passengers();
{
return passenger_load;
}
};
class boat: public vehicle
{
int passenger_load;
public:
void message()
{
cout<<"Лодка\n";
}
};
void main()
{
vehicle *unicycle; //переменная unicycle явл. на объект класса vehicle
unicycle=new vehicle; //создаём об. кл. vehicle на который указывает unicycle
unicycle - > message();
delete unicycle;
unicycle::new car;
unicycle - > message();
delete unicycle;
unicycle=new truck;
unicycle - > message();
delete unicycle;
unicycle=new boat;
unicycle - > message();
delete unicycle;
}
```

Рассмотрим **void main:**

В функции **main** описывается переменная **unicycle** как указатель на объект типа **vehicle.** Применение указателя позволяет использовать один и тот же указатель для всех производных классов. Если бы не было указано, что функция **message** класса **vehicle** является виртуальной, то компилятор статически связал бы любой вызов метода объекта указателя **unicycle** с методом **message** класса **vehicle**, т.к. при описании было указано, что переменная **unicycle** указывает на объект класса **vehicle**, т.е. произвели бы раннее связывание.

Результатом бы такой работы был бы вывод четырех строк «транспортное средство».
Очень часто класс, содержащий виртуальный метод называется полиморфным классом, т.е. функции, описанные в базовом классе, как виртуальные могут быть модифицированы в производных классах, причем связывание произойдет не на этапе компиляции, а в момент обращения к данному методу (позднее связывание).
Таким образом, виртуальные методы описываются с помощью ключевого слова **virtual** в базовый класс. Это означает, что в производном классе этот метод может быть замещен к более подходящим для этого производного класса.
Объявление виртуальным в базовом классе метод останется виртуальным для всех производственных классов. Если в производственном классе виртуальный метод не будет переопределен, то при вызове будет найден метод с таким именем вверх по иерархии классов.

Абстрактные классы.
Очень часто в базовом классе определяется виртуальная функция, которая не выполняет каких-либо значимых действий. Отметим, что чем ниже находится класс в иерархии классов, тем он более конкретен, т.е. базовый класс обычно не определяет законченный тип. Очень часто получается, что базовый класс не имеет какой-либо практической ценности для его непосредственного применения в программе. Однако имеет ценность для формирования производственных классов.

На рассмотренном выше примере можно сказать, что мы в программе создавали объект класса **vehicle** только для предания примеру большей наглядности. Нас больше интересует конкретная реализация возможностей этого класса, которую нам дают производные из него класса.
Класс **vehicle** имеет ценность как базовый класс, а функция **message** класса **vehicle** нужна только для того, чтобы её переопределили в производных классах.
Чисто виртуальным методом называется такой метод, который не определяется в базовом классе, т.е. у него нет тела, а есть только декларация о его существовании.
Абстрактный класс - это класс, содержащий хотя бы один чистый виртуальный метод.

```
class vehicle
{
int wheels;
float weight;
public:
virtual void message ()=0; //чистый виртуальный метод
}
```

Абстрактные классы не бывают изолированными, т.е. всегда абстрактный класс должен быть наследуемым, поскольку у чисто виртуального метода нет тела, то создать объект абстрактного класса **невозможно.**

Некоторые особенности объектно-ориентированного программирования.
В ООП (объектно-ориентированное программирование) три базовых понятия остаются неизменными к ним относятся: инкапсуляция, наследование, полиморфизм.
ООП позволяет разложить проблему на связанные между собой задачи. Каждая проблема становится самостоятельным объектом, содержащим свои собственные данные, которые относятся к этому объекту, в этом случае исходная задача в целом упрощается, и программисты получает возможность оперировать с большими по объёму программами.
В этом определении ООП отражается известный подход к решению сложных задач, когда мы разбиваем задачу на подзадачи и решаем эти подзадачи по отдельности.

С точки зрения программирования подобный подход значительно упрощает разработку, отладку программ.

Инкапсуляция.

Инкапсуляция – механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то, и другое от внешнего вмешательства ими неправильного использования. Когда методы и данные объединяются, т.о. создаётся объект. Применение этого принципа очень часто помогает локализовать возможные ошибки в коде программы, а это намного упрощает процесс поиска и исправления этих ошибок, можно сказать, что инкапсуляция подразумевает под собой вскрытие данных, что позволяет защитить эти данные.

Суть инкапсуляции заключается в том, что переменные состояния объекта скрыты от внешнего мира. Изменения состояния объекта возможно только с помощью его методов. Это существенно ограничивает возможность введения объекта в недопустимое состояние или несанкционированное разрушение этого объекта.

Но у инкапсуляции есть один недостаток. Применение инкапсуляции ведет к снижению эффективности доступа к элементам объекта. Это обусловлено необходимостью вызова методов для изменения внутренних элементов объекта, однако при современном уровне развитии Вычислительной Техники эти потери в эффективности не играют существенную роль.

Рассмотрим также ещё одно понятие ООП – **абстрактные типы данных.**

Абстрактный тип данных – это группа тесно связанных между собой данных и методах, которые могут осуществлять операции над этими данными. Поскольку подразумевается, что эта структура защищена от внешнего влияния, то она считается **инкапсулированной структурой,** данные заключённые в этой структуре тесно связанным и активно взаимодействуют между собой внутри структуры, но имеет слабые связи с внешним миром по средствам ограниченного числа метода.

Т.о. структуры подобного рода имеют достаточно простой и эффективный интерфейс, что позволяет их легко интегрировать в программах.

Полиморфизм.

В переводе с греческого, полиморфоз – многообразный. Если один и тот же объект может по-разному использоваться в зависимости от обстоятельств, то он обладает полиморфизмом.

Полиморфизм – свойство, которое позволяет использовать одно и то же имя для решения нескольких разных технических задач. В общем случае концепцией полиморфизма является идея «один интерфейс множество методов». Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного интерфейса для единого класса действий. Выбор конкретного действия в зависимости от ситуаций возлагается на компилятор.

Применительно к ООП с целью полиморфизма является использование одного имени для задания общих для класса действий. На практике это означает способность объектов выбирать внутреннюю процедуру, исходя из типов данных принятых сообщений.

Наследование.

Это процесс по средствам, которого один объект может наследовать свойство другого объекта и добавлять им черты характерные только для него. Смысл и универсальность наследования заключается в том, что не надо каждый раз заново описывать новый объект, а можно указать родителя и описать отличительные особенности нового класса. В результате, новый объект будет обладать всеми свойствами родительского класса плюс своими собственными отличительными свойствами.

Приложение 3

СОГЛАСОВАНО
Протокол заседания кафедры
№_____ от_____

УТВЕРЖДЕНО
Ректор УНПК «МУК»
_____
(название)
_____
(подпись, ф.и.о.)

ЛИСТ ИЗМЕНЕНИЙ

в учебно-методический комплекс (модуле) дисциплины _____
(название дисциплины)
по направлению подготовки (специальности)_____

на 20__/20__ учебный год

1. В _____ вносятся следующие изменения:
(элемент УМК)
1.1. ……..;
1.2. ……...;
…
1.9. …… .

2. В _____ вносятся следующие изменения:
(элемент УМК)
2.1. ……..;
2.2. ……...;
…
2.9. …… .

3. В _____ вносятся следующие изменения:
(элемент УМК)
3.1. ……..;
3.2. ……...;
…
3.9. …… .

Составитель подпись расшифровка подписи
дата