

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению практических работ
по дисциплине «Информационные технологии и программирование» для студентов
направления подготовки
10.03.01 «Информационная безопасность»
Направленность (профиль)
«Организация и технологии защиты информации (по отрасли или в сфере
профессиональной деятельности)»

Ставрополь, 2026

ВВЕДЕНИЕ

Целью изучения дисциплины является формирование компетенций у студентов направления подготовки 10.03.01 «Информационная безопасность» и формирование теоретических знаний и практических навыков по построению информационных систем и программированию.

Задачей освоения дисциплины является обучение студентов принципам разработки и использования программного обеспечения для построения ИС и программирования в рамках прикладных задач.

Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-1 Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	ИД-2 УК-1 осуществляет поиск, отбор и систематизацию информации для определения альтернативных вариантов стратегических решений в проблемной ситуации	уверенно осуществляет поиск, отбор и систематизацию информации для определения альтернативных вариантов стратегических решений в проблемной ситуации в полном объеме
ОПК-7 Способен использовать языки программирования и технологии разработки программных средств для решения задач профессиональной деятельности	ИД-1 ОПК-7 Применяет основные методы при разработке алгоритмов (рекурсия, отход назад, метод ветвей и границ, анализ арифметических выражений); основные структуры и типы данных; терминологию дисциплины; библиотеки стандартных программ	в совершенстве применяет основные методы при разработке алгоритмов (рекурсия, отход назад, метод ветвей и границ, анализ арифметических выражений); основные структуры и типы данных; терминологию дисциплины; библиотеки стандартных программ
	ИД-2 ОПК-7 Разбивает решение сложной задачи на последовательность более простых задач; использовать базовые алгоритмы на динамических структурах данных	в совершенстве разбивает решение сложной задачи на последовательность более простых задач; использовать базовые алгоритмы на динамических структурах данных
	ИД-3 ОПК-7 создаёт программы на языках общего назначения, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования и способов организации программ	уверенно создает программы на языках общего назначения, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования и способов организации программ

1 СЕМЕСТР

(Формируемые компетенции: УК-1, ОПК-7)

Тема 1. Вводная лекция

Тема 2. Представление знаний о предметной области.

Практическое занятие 1-5. Тема: Понятие ЭС. Главное достоинство и назначение ЭС **Формируемые компетенции: УК-1, ОПК-7**

Понятие ЭС. Главное достоинство и назначение ЭС

В начале 80-х годов в рамках искусственного интеллекта (ИИ) сформировалось самостоятельное направление - "инженерия знаний", в задачу которого входят разработка, исследование и использование экспертных систем.

ИИ – отрасль информатики. Спец-ты в области ИИ стремились разработать программы для ЭВМ, которые могли бы в некотором смысле «думать», т. е. решать задачи таким способом, который мы бы сочли разумным, если бы его применил человек. ИИ – программная система, имитирующая мышление человека.

Огромный интерес к экспертным системам (ЭС) вызван следующим и основными причинами:

- они ориентированы на решение широкого круга задач в неформализованных областях, т. е. на приложения, которые до недавнего времени считались малодоступными для вычислительной техники.
- экспертные системы позволяют специалистам, не имеющим навыков программирования, создавать практически значимые приложения, что резко расширяет сферу использования вычислительной техники.
- экспертные системы при решении практических задач позволяют получать результаты, сравнимые, а иногда и превосходящие те, которые может получить эксперт- человек.
- современные ЭС легко объединяются с традиционными программными системами (системами управления базами данных, пакетами прикладных программ и т. д.) в интегрированные приложения.

Экспертная система — это компьютерная система, которая воплощает в себе опыт эксперта, который основывается на его знаниях в определенной отрасли. Экспертная система (ЕС) на основе обработки этих знаний может давать интеллектуальные советы, принимать решение на уровне эксперта - профессионала, а также по желанию пользователя объяснять ход решения в случае нахождения того или другого решения.

Основные характеристики экспертной системы таковы:

1. Экспертная система, как правило, ограничена определенной предметной областью.
2. ЭС должна уметь принимать решение при неполных или неточных данных.
3. ЭС должна уметь объяснять свои действия при решении задачи.
4. Система должна иметь свойство расширения и наращивания функций.
5. ЭС должна уметь имитировать деятельность

высококвалифицированного специалиста (эксперта).

6. ЭС при решении задач использует, как правило, не точные алгоритмы, а так называемые эвристики, то есть методы, которые опираются на опыт и знание эксперта. **Главное достоинство ЭС** - возможность накапливать знания, сохранять их длительное время, обновлять и тем самым обеспечивать относительную независимость конкретной организации от наличия в ней квалифицированных специалистов. Накопление знаний позволяет повышать квалификацию специалистов, работающих на предприятии, используя наилучшие, проверенные решения.

Основу ЭС составляет **база знаний** (БЗ) о предметной области, которая накапливается в процессе построения и эксплуатации ЭС. Накопление и организация знаний - важнейшее свойство всех ЭС.

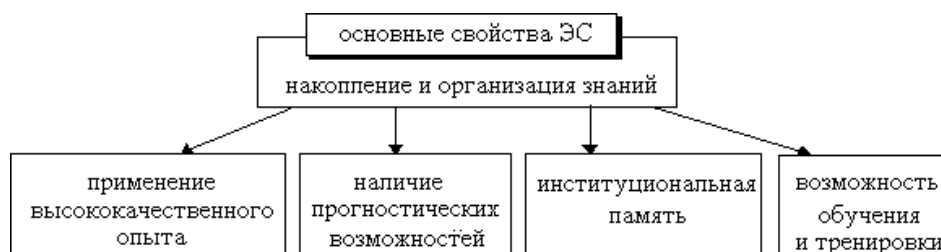


рис.1.

Знания являются явными и доступными, что отличает ЭС от традиционных программ, и определяет их основные свойства, такие, как:

1) **Применение** для решения проблем *высококачественного опыта*, который представляет уровень мышления наиболее квалифицированных экспертов в данной области, что ведёт к решениям творческим, точным и эффективным.

2) Наличие **прогностических возможностей**, при которых ЭС выдаёт ответы не только для конкретной ситуации, но и показывает, как изменяются эти ответы в новых ситуациях, с возможностью подробного объяснения каким образом новая ситуация привела к изменениям.

3) Обеспечение такого нового качества, как **институциональная память**, за счёт входящей в состав ЭС базы знаний, которая разработана в ходе взаимодействий со специалистами организации, и представляет собой текущую политику этой группы людей. Этот набор знаний становится сводом квалифицированных мнений и постоянно обновляемым справочником наилучших стратегий и методов, используемых персоналом. Ведущие специалисты уходят, но их опыт остаётся.

4) **Возможность** использования ЭС для **обучения и тренировки** руководящих работников, обеспечивая новых служащих обширным багажом опыта и стратегий, по которым можно изучать рекомендуемую политику и методы.

5)

ЭС создаются для решения разного рода проблем, но основные типы их деятельности можно сгруппировать в категории, приведенные в табл.1

Табл.1 – Категории применения ЭС

Категория	Решаемая проблема
Интерпретация	Описание ситуации по информации, поступающей от датчиков
Прогноз	Определение вероятных последствий заданных ситуаций
Диагностика	Выявление причин неправильного функционирования системы по результатам наблюдений
Проектирование	Построение конфигурации объектов при заданных ограничениях
Планирование	Определение последовательности действий
Наблюдение	Сравнение результатов наблюдений с ожидаемыми результатами
Отладка	Составление рецептов исправления неправильного функционирования системы
Ремонт	Выполнение последовательности предписанных исправлений
Обучение	Диагностика, отладка и исправление поведения обучаемого
Управление	Управление поведением системы как целого

Список используемой литературы:

Основная: 1-3

Дополнительная 1-2

Тема 3. Модели представления знаний о предметной области

Тема 4. Основные понятия теории нейронных сетей

Практическое занятие 6-9

Тема: Базовые функции ЕС.

Формируемые компетенции: УК-1, ОПК-7

Выделяют 4 основные функции экспертных систем:

1. Приобретение знаний;
2. Представление знаний;
3. Управление процессом поиска решения;
4. Разъяснение принятого решения

Приобретение знаний это передача потенциального опыта решения проблемы от некоторого источника знаний и преобразование его в вид, который позволяет использовать эти знания в программе.

Передача знаний выполняется в процессе достаточно длительных и пространных собеседований между специалистом по проектированию экспертной системы (будем в дальнейшем называть его инженером по знаниям) и экспертом в определенной предметной области, способным достаточно четко сформулировать имеющийся у него опыт.

Представление знаний — еще одна функция экспертной системы. Теория представления знаний — это отдельная область исследований, тесно связанная с философией формализма и когнитивной психологией. Предмет исследования в этой области — методы ассоциативного хранения информации, подобные тем, которые существуют в мозгу человека. В области экспертных систем представление знаний - средство отыскания методов формального описания больших массивов полезной информации с целью их последующей обработки с помощью символических вычислений. Формальное описание означает упорядочение в рамках какого-либо языка, обладающего достаточно четко формализованным синтаксисом построения выражений и такого же уровня семантикой, увязывающей смысл выражения с его формой.

Управление процессом поиска решения. При проектировании экспертной системы серьезное внимание должно быть уделено и тому, как осуществляется доступ к знаниям и как они используются при поиске решения. Знание о том, какие знания нужны в той или иной конкретной ситуации, и умение ими распорядиться — важная часть процесса функционирования экспертной системы. Такие знания получили наименование метазнаний — т. е. знаний о знаниях.

Разъяснение принятого решения. Вопрос о том, как помочь пользователю понять структуру и функции некоторого сложного компонента программы, связан со сравнительно новой областью взаимодействия человека и машины, которая появилась на область исследователей, занимающихся экспертными системами, состоит в разработке методов представления информации о поведении программы в процессе формирования цепочки логических заключений при поиске решения. Способность системы объяснить методику принятия решения

иногда называют прозрачностью системы. Под этим понимается, насколько просто персоналу выяснить, что делает программа и почему.

Отсутствие достаточной прозрачности поведения системы не позволит эксперту повлиять на ее производительность или дать совет, как можно ее повысить. Прослеживание и оценка поведения системы — задача довольно сложная и для ее решения необходимы совместные усилия эксперта и специалиста по информатике.

Список используемой литературы:

Основная: 1-3

Дополнительная 1-2

Тема 5. Введение в нейронные сети
Тема 6. Построение экспертных системы

Практическое занятие 10-12

Тема: История разработки экспертных систем Формируемые компетенции: УК-1, ОПК-7

История разработки экспертных систем является очень обширной.

До середины 60-х годов основные исследования в искусственном интеллекте сосредоточились на создании интеллектуальных систем, которые мало полагались на знания в проблемной области и в основном опирались на мощные методы формирования рассуждений. Методы рассуждений, используемые в универсальных решателях задач, были очень мощными, но эти машины так и не смогли выйти за пределы начального уровня компетентности.

В начале 70-х годов стало очевидно, что ключом к созданию машинных решателей задач, способных функционировать на уровне эксперта-человека являются знания в проблемной области. С тех пор создано множество успешно действующих экспертных систем.

Наиболее известные ЭС, разработанные в этот период, стали в своих областях уже классическими. По происхождению, предметным областям и по преобладающим применяемым идеям, методам и инструментальным программным средствам их можно разделить на несколько семейств.

1. **META-DENDRAL.** Система DENDRAL позволяет определить наиболее вероятную структуру химического соединения по экспериментальным данным. M-D автоматизирует процесс приобретения знаний для DENDRAL. Она генерирует правила построения фрагментов химических структур.

2. **MYCIN-EMYCIN-TEIREIAS-PUFF-NEOMYCIN.** Это семейство

медицинских ЭС и сервисных программных средств для их построения.

3. **PROSPECTOR-KAS.** **PROSPECTOR-** предназначена для поиска (предсказания) месторождений на основе геологических анализов. **KAS-** система приобретения знаний для **PROSPECTOR.**

4. **CASNET-EXPERT.** Система CASNET- медицинская ЭС для диагностики выдачи рекомендаций по лечению глазных заболеваний. На ее основе разработан язык инженерии знаний **EXPERT**, с помощью которой создан ряд других медицинских диагностических систем.

5. **HEARSAY-HEARSAY-2-HEARSAY-3-AGE.** Первые две системы этого ряда являются развитием интеллектуальной системы распознавания слитной человеческой речи, слова которой берутся из заданного словаря. Эти Системы отличаются оригинальной структурой, основанной на использовании доски объявлений- глобальной базы данных, содержащей текущие результаты Работы системы. В дальнейшем на основе этих систем были созданы инструментальные системы **HEARSAY-3** и **AGE** (Attempt to Ge)

6. Системы **AM** (Artificial Mathematician- искусственный математик) и **EURISCO** были разработаны в Стэнфордском университете доктором Д. Ленатом для исследовательских и учебных целей. Ленат считает, что эффективность любой

ЭС определяется закладываемыми в нее знаниями. По его мнению, чтобы система была способна к обучению, в нее должно быть введено около миллиона сведений общего характера. Это примерно соответствует объему информации, каким располагает четырехлетний ребенок со средними способностями. Ленат также считает, что путь создания узкоспециализированных ЭС с уменьшенным объемом знаний ведет к тупику.

К концу 70-х годов сложились три понятия, которые стали основными для большинства экспертных систем.

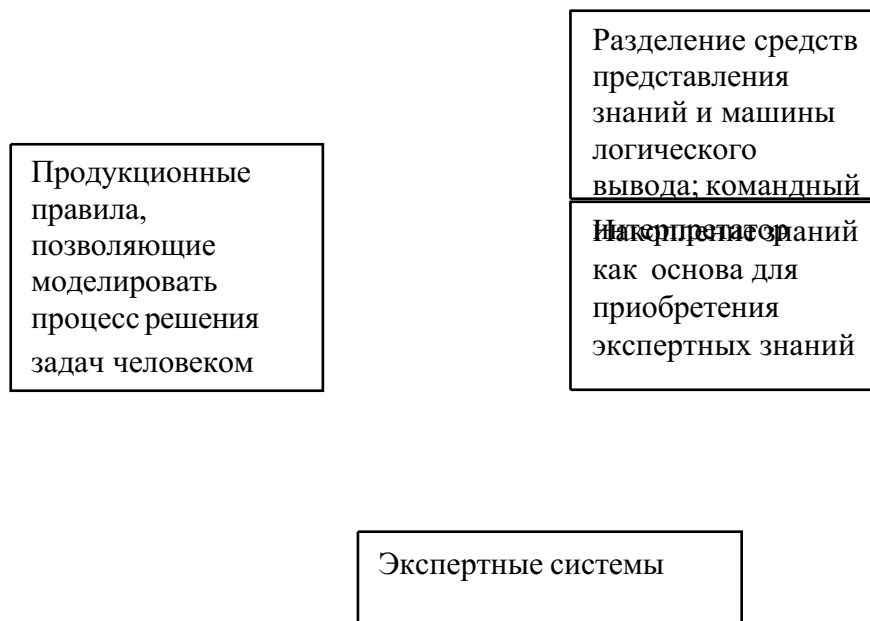


Рисунок 2 – Результаты обобщения понятий, на базе которых создаются экспертные системы

В 1980-х начали появляться новые компании, которые вывели экспертные - системы за пределы лабораторий и приступили к созданию коммерческих программных продуктов. Было создано новое мощное программное обеспечение, написанное на языке LISP. По мере наращивания мощности персональных компьютеров были введены инструментальные средства экспертных систем CLIPS.

*Список используемой литературы: Основная: 1-3
Дополнительная 1-2*

Тема 7. Экспертные системы

Тема 8. Экспертные системы в безопасности

Практическое занятие 13-16

Тема: Состав и взаимодействие участников построения и эксплуатации экспертных систем

Формируемые компетенции: УК-1, ОПК-7

Экспертная система воплощает в себе неписанные знания, которые должны быть получены от эксперта с помощью обширных интервью, проводимых инженером по знаниям в течение длительного периода времени. Такой процесс создания экспертной системы называется **инженерией знаний** и осуществляется инженером по знаниям.

Инженерией знаний называют получение знаний от эксперта-человека или из других источников и последующее представление знаний в экспертной системе. Основные этапы разработки экспертной системы представлены на рисунке 1.

Вначале инженер по знаниям устанавливает диалог с экспертом-человеком, чтобы выявить эксперта. Этот этап аналогичен этапу работы, выполняемому системным проектировщиком при обычном программировании в ходе обсуждения требований к системе с клиентом, для которого создается программа. Затем инженер по знаниям представляет знания в явном виде для внесения в базу знаний. После этого эксперт проводит оценку экспертной системы и передает критические замечания инженеру по знаниям. Такой процесс повторяется снова и снова, до тех пор, пока эксперт не оценит результаты работы системы как

удовлетворительные.

Выражение система, основанная на знаниях, представляет собой лучший термин для обозначения технологии, основанной на знаниях, поскольку он может применяться для обозначения и экспертных систем и систем, основанных на знаниях.



Рисунок 1. - Процесс разработки экспертной системы

Эксперт — это человек, способный ясно выражать свои мысли и пользующийся репутацией специалиста, умеющего находить правильные решения проблем в конкретной предметной области. Эксперт использует свои приёмы и ухищрения, чтобы сделать поиск решения более эффективным, и ЭС моделирует все его стратегии.

Инженер знаний — человек, как правило, имеющий познания в информатике и искусственном интеллекте и знающий, как надо строить ЭС. Инженер знаний опрашивает экспертов, организует знания, решает, каким образом они должны быть представлены в ЭС, и может помочь программисту в написании программ.

Средство построения ЭС — это программное средство, используемое инженером знаний или программистом для построения ЭС. Этот инструмент отличается от обычных языков программирования тем, что обеспечивает удобные способы представления сложных высокоуровневых понятий.

Пользователь — это человек, который использует уже построенную ЭС. Так, пользователем может быть юрист, использующий её для квалификации конкретного случая; студент, которому ЭС помогает изучать информатику и т. д.

Список используемой литературы:

Основная: 1-3

Дополнительная 1-2

Тема: Проблемы, возникающие при создании ЭС. Формируемые компетенции: УК-1, ОПК-7

При попытке применить технологию экспертных систем компании сталкиваются с несколькими типами трудностей. Скудные ресурсы делают трудновыполнимой комплектование квалифицированной бригады разработчиков. Особенности технологии ИИ налагают жесткие ограничения на процесс построения ЭС. Наконец, разработка ЭС требует продолжительного времени. Понимание этих трудностей может помочь компаниям приспособить к ним свои планы.

Первая трудность возникает в связи с постановкой задач. Большинство заказчиков, планируя разработку ЭС, вследствие недостаточной компетентности в вопросах применения методов ИИ, склонна значительно преувеличивать ожидаемые возможности системы. Заказчик желает увидеть в ней самостоятельно мыслящего эксперта в исследуемой области, способного решать широкий круг задач. Отсюда и типичные первоначальные постановки задачи по созданию ЭС: -Разработать ЭС по обработке изображения»; -Создать медицинские ЭС по лечению заболеваний опорно- двигательного аппарата у детей». Однако, мощность эвристических методов решения задач при увеличении общности их постановки резко уменьшается. Поэтому наиболее целесообразно (особенно при попытке создания ЭС в области, для которой у разработчиков еще нет опыта создания подобных систем) ограничиться для начала не слишком сложной обозримой задачей в рассматриваемой области, для решения которой нет простого алгоритмического способа (то есть неочевидно, как написать программу для решения этой задачи, не используя методы обработки знаний). Кроме того, важно, чтобы уже существовала сложившаяся методика решения этой задачи вручную или какими-либо расчетными методами. Для успешной разработки ЭС необходимы не только четкая и конкретная постановка задач, но и разработка подробного (хотя бы словесного) описания ручного (или расчетного) метода ее решения. Если это сделать затруднительно, дальнейшая работа по построению ЭС теряет смысл.

Вторая и основная трудность - проблема приобретения (усвоения) знаний. Эта проблема возникает при -передаче» знаний, которыми обладают эксперты-люди, ЭС. Разумеется, для того чтобы им обучить компьютерную систему, прежде всего требуется сформулировать, систематизировать и формализовать эти знания на бумаге.

Для разработки ЭС необходимо участие в ней особого рода специалистов, обладающих указанной совокупностью знаний и выполняющих функции посредников между экспертами в предметной области и компьютерными (экспертными) системами.

Они получили название инженеры знаний (в оригинале - knowledge engineers), а сам процесс разработки ЭС и других интеллектуальных программ, основанных на представлении и обработке знаний - инженерией знаний (knowledge engineering).

Третья серьезная трудность - в очень большой трудоемкости создания ЭС : требуется разработать средства управления базой знаний, логического вывода, диалогового взаимодействия с пользователем и т.д. Объем программирования столь велик, а программы столь сложны и нетрадиционны, что имеет смысл, как это принято сейчас при разработке больших программ, на первом этапе создать демонстрационный прототип системы - предварительный вариант, в котором в упрощенном виде реализованы лишь ее основные планируемые возможности и которая будет служить для заказчика подтверждением того, что разработка ЭС для решения данной задачи принципиально возможна, а для разработчиков-основой для последующего улучшения и развития системы.

Список используемой литературы: Основная: 1-3

Дополнительная 1-2

2 СЕМЕСТР

(Формируемые компетенции: УК-1, ОПК-7)

Тема 1. Современные технологии программирования

Практическое занятие №1 Введение в языки программирования. Общие принципы построения и использования языков программирования. Язык программирования C++

Цель работы: изучение основных типов данных, способов описания переменных различных типов, операторов присваивания и организации ввода – вывода.

Краткие теоретические сведения:

Алгоритм – это последовательность действий, выполняемых по строго определенным правилам, однозначно определяющая процесс решения задачи и заведомо приводящая к её решению за некоторое количество шагов. Алгоритмизация – разработка формального метода решения практической задачи с возможностью реализации в виде программы для ЭВМ.

Изображение алгоритма в виде схемы выполняется в соответствии с ГОСТ 19.701–90 Единой системы программной документации «Схемы алгоритмов, программ, данных и систем».

Структура программ для Microsoft Visual Studio.

```
// struct_program.cpp: определяет точку входа для
консольного приложения.
#include "stdafx.h"

//здесь подключаем все
необходимые препроцессорные директивы
int main() { // начало главной функции с именем main

//здесь будет находиться ваш программный код

}
```

В строке 1 говорится о точке входа для консольного приложения, это значит, что данную программу можно запустить через командную строку Windows указав имя программы, к примеру, такое struct_program.cpp. Строка 1 является однострочным комментарием, так как начинается с символов //. В строке 2 подключен заголовочный файл "stdafx.h". Данный файл похож на контейнер, так как в нем подключены основные препроцессорные директивы (те, что подключил компилятор, при создании консольного приложения), и вспомогательные (подключенные программистом).

include — директива препроцессора, т. е. сообщение препроцессору. Строки, начинающиеся с символа # обрабатываются препроцессором до компиляции программы.

Заголовочные файлы:

Библиотека `cmath` определяет набор функций для выполнения общих математических операций и преобразований. Математические функции:

1) тригонометрические функции:

cos-вычисление косинуса угла, переведенного в радианы; sin-вычисление синуса угла, переведенного в радианы; tan-вычисление тангенса угла, переведенного в радианы;

acos-вычисление арккосинуса, результат будет в радианах; asin-вычисление арксинуса, результат будет в радианах;

atan-вычисление арктангенса, возвращаемый результат будет в радианах; atan2-вычисление арктангенса и квадранта по координатам x и y, возвращаемый результат будет в радианах;

2) гиперболические функции:

cosh-вычисление гиперболического косинуса; sinh-вычисление гиперболического синуса; tanh-вычисление гиперболического тангенса;

3) экспоненциальные и логарифмические функции: exp-вычисление экспоненты;

frexp-получить мантиссу и показатель степени двойки;

ldexp-генерация числа по значению мантиссы и показателю степени; log-натуральный логарифм;

log10-десятичный логарифм;

modf-разделение вещественного значения на дробную и целую части;

4) функции степени:

pow-возведение числа в степень. Пример использования функции; sqrt-корень квадратный;

5) округление, модуль и другие функции ceil-округление до наименьшего целого значения; fabs-вычислить модуль значения;

floor-округление до наибольшего целого значения; fmod-остаток от деления числителя на знаменатель.

Пример использования функций:

```
#include <iostream> // для оператора cout #include <cmath>      // для функции pow
```

```
int main()
{
    std::cout << "5.0 ^ 4 = "      << pow(5.0, 4) << std::endl;
    std::cout << "2.77 ^ 9 = "      << pow(2.77, 9) << std::endl;
    std::cout << "12.01 ^ 11.54 = " << pow(12.01, 11.54) << std::endl; std::cout <<
    "sqrt(" << param << ") = "
    << sqrt(param)                  // вычисляем корень квадратный
    << std::endl;
    double param = 60.0;           // угол 60 градусов std::cout << "Косинус "
    << param
    << " градусов = " << cos(param * PI / 180) // вычисляем косинус угла,
    //переведённого в радианы
    << std::endl; double param = 0.5;
    std::cout << "Арксинус " << param
```

```

<< " = " << (asin (param) * 180.0 / PI) // вычисляем арксинус
<< " градусов " << std::endl; double val = 5.5, result;
result = log (val); // вычисляем натуральный логарифм std::cout <<
"ln(" << val << ") = "
<< result << std::endl; return 0;}

```

Файлы кода C++ (с расширением .cpp) не являются единственными файлами в проектах и программах. Есть еще один тип файлов, который называется **заголовочный файл** (файл заголовка, подключаемый файл или header file). Они имеют расширение .h, но иногда их можно увидеть и с расширением .hpp или вообще без расширения. Целью заголовочных файлов является удобное хранение предварительных объявлений для использования другими файлами. Всё содержимое из заголовочного файла копируется в файл *.cpp, т.е. всё содержимое становится доступным для использования.

Заголовочные файлы:

- cstdio (stdio.h)-заголовочный файл для выполнения операций ввода/вывода;
- cstring (string.h)-заголовочный файл для работы со строками;
- iostream-заголовочный файл с классами, функциями и переменными для организации ввода-вывода. Для удобства в библиотеке определены три стандартных объекта-потока:

cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;

cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;

cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (>>).

Основные типы данных представлены в таблице 1.

Таблица 1 — Типы данных C++

Тип	байт	Диапазон принимаемых значений
целочисленный (логический) тип данных		
bool	1	0 / 255
целочисленный (символьный) тип данных		
char	1	0 / 255
целочисленные типы данных		
short int	2	-32 768 / 32 767
Unsigned short int	2	0 / 65 535
int	4	-2 147 483 648 / 2 147 483 647

unsigned int	4	0 / 4 294 967 295
long int	4	-2 147 483 648 / 2 147 483 647
Unsigned long int	4	0 / 4 294 967 295
типы данных с плавающей точкой		
float	4	-2 147 483 648.0 / 2 147 483 647.0
long float	8	-9 223 372 036 854 775 808 .0/ 9 223 372 036 854 775 807.0
double	8	-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0

Способы ввода данных в языке возможно двумя способами: форматированные ввод-вывод или потоковый.

При форматированном способе используются операторы ввода scanf вывода printf. Синтаксис операторов имеет вид:

scanf(<строка описания форматов> [, <список ввода>]); printf(<строка описания форматов> [, <список вывода>]);

Строка описания форматов состоит из обычных символов, специальных управляющих последовательностей символов и спецификаций формата. Обычные символы и управляющие последовательности просто копируются в стандартный выходной поток в порядке их появления. Спецификации формата начинаются с символа % и заканчиваются символом, определяющим тип выводимого значения. Кроме того, спецификации формата могут содержать символы и цифры для управления видом выводимого значения (подробно см. ниже). Список вывода состоит из переменных и/или констант, значения которых должны быть выведены. Количество спецификаций формата должно быть равно количеству выводимых значений, которые указываются в списке вывода.

К управляющим последовательностям относятся последовательности символов, представленных в таблице 2.

Таблица 2 — Управляющие символы

Последовательность	Действие
\a	Звуковой сигнал
\b	Удаление предыдущего символа
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\'	Апостроф
\"	Кавычки

\\	Обратный слеш
\ooo	ASCII символ в восьмеричной нотации
\xooo	ASCII символ в шестнадцатеричной нотации

Иногда при работе операторов используются спецификаторы форматов (таблица 3-5).

Таблица 3 — Спецификаторы формата для оператора printf

Символ	Назначение
%c	символ
%d	целое десятичное число
%i	целое десятичное число
%e	десятичное число в виде x.xx e+xx
%E	десятичное число в виде x.xx E+xx
%f	десятичное число с плавающей запятой xx.xxxx
%F	десятичное число с плавающей запятой xx.xxxx

%g	%f или %e, что короче
%G	%F или %E, что короче
%o	восьмеричное число
%s	строка символов
%u	беззнаковое десятичное число
%x	шестнадцатеричное число
%X	шестнадцатеричное число
%%	символ %
%p	указатель
%n	указатель

Кроме того, к командам формата могут быть применены модификаторы l и h (таблица 4).

Таблица 4

Обозначение	Назначение
-------------	------------

%ld	печать long int
%hu	печать short unsigned
%Lf	печать long double

Таблица 5 — Спецификаторы формата для оператора scan f

Символы	Назначение
%c	чтение символа
%d	чтение десятичного целого
%i	чтение десятичного целого
%e	чтение числа типа float (плавающая запятая)
%h	чтение short int
%o	чтение восьмеричного числа
%s	чтение строки
%x	чтение шестнадцатеричного числа
%p	чтение указателя
%n	чтение указателя в увеличенном формате

Примеры использования спецификаторов и модификаторов в операторах ввода-вывода:

```
int m, n, x; double y; char c = '&';
char str[] = "String";
scanf("%d%d", &m, &n); // Ввод десятичных целых чисел в переменные m и
n
```

```

printf("m = %5d\nn = %5d\n", m, n);    // Вывод переменных m и n в
десятичном целом формате, используются как минимум 5 знаков
scanf("%d", &x);    // Ввод десятичного целого числа в переменную x
printf("%010x\n", x);    // Вывод переменной x в шестнадцатеричной
системе, используются 10 знаков,
// впереди добавляются нули и символы 0x
scanf("%lf", &y);    // Ввод вещественного числа в переменную y
printf("y = %7.2lf\n", y);    // Вывод вещественной переменной, используются
как минимум 7 знаков, из них 2 – после точки
printf("c = %c\n", c); // Вывод одного символа printf("%.4s\n", str);
}

```

Составим программу на языке C++ для умножения двух чисел.

//Подключение стандартных библиотек:

```
#include<iostream>
```

```
#include <math.h> // для работы с мат.функциями acos, log10, log, atan
```

```
using namespace std; // пространство имен, где определяются
```

идентификаторы

```

void main()                //заголовок главной функции программы
{
    setlocale(0, "Russian"); //установка вывода сообщений на русском языке float
x,y,z;                    //описание переменных (имя и тип)
system("cls");            //функция очистки экрана
cout<<"Введите числа x,y: "<<endl; //вывод сообщения на экран
cin>>x>>y;                //считать 2 вещ.числа, записать их по адресу
перем. x,y
z=x*y;
cout<<"Произведение="<<z;    //вывести результат на экран
system("pause");
}                            //конец главной функции

```

Пример выполнения задания.

Задание: вычислить $\frac{\sin^3 c \cdot \cos^2 a}{5 \sin^d b}$ при $\frac{2}{15}$

$A = 9,5$; $B = 1,365$; $C = 6,5$; $D = 5$. Использовать два варианта ввода исходных данных и вывода результатов: возможности библиотеки функций языка C и библиотеки классов языка C++.

Схема программы для задания представлена на рисунке 3:

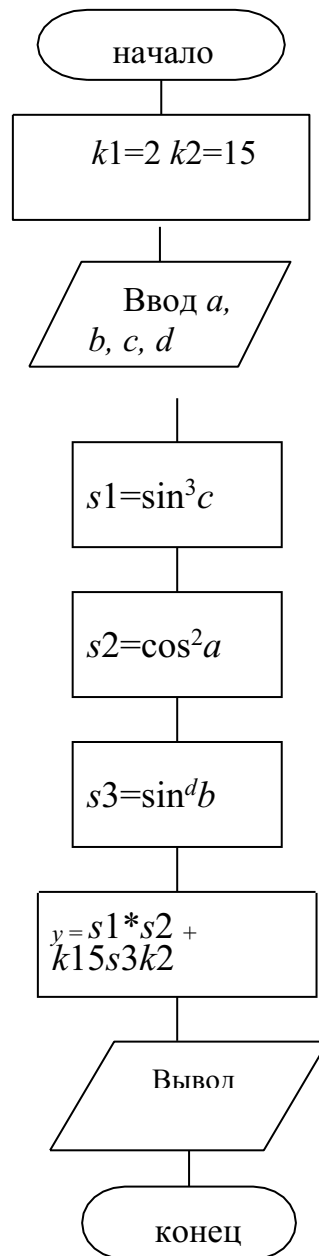


Рисунок 3

Текст программы с использованием библиотеки функций языка C (форматированный ввод-вывод) для ввода исходных данных и вывода результатов:

```

//Лабораторная работа 1
//с использованием библиотеки классов языка C++
// форматированный ввод-вывод # include <iostream>
# include <math.h> using namespace std; void main()
{
    setlocale(0, "Russian"); //Вывод сообщений на русском языке const double
k1=2.0;
    const double k2= 15.0; double a, b, c, d, s1, s2, s3, y;
    system("cls"); //Очистка экрана printf("Введите значения
переменных a, b, c, d:\n"); scanf("%lf%lf%lf%lf", &a, &b, &c, &d); s1=pow(sin(c),
3);
    s2=pow(cos(a), 2 );
  
```

```

s3=pow(sin(b), d ); y=(s1*s2)/(5*s3) + k1/k2;
printf("Искомое значение y=%lf", y); system("pause");    //Пауза
}
Текст программы с использованием библиотеки классов языка
C++ (поточный ввод-вывод) для ввода исходных данных и вывода результатов:
//Лабораторная работа 1
//с использованием библиотеки классов языка C++
// поточный ввод-вывод # include <iostream>
# include <math.h> using namespace std; void main()
{
setlocale(0, "Russian"); //Вывод сообщений на русском языке const double k1=
2.0;
const double k2= 15.0; double a, b, c, d, s1, s2, s3, y;
system("cls"); //Очистка экрана cout<<"Введите
значения переменных a, b, c, d:"<<endl; cin>>a>>b>>c>>d;
s1=pow(sin(c), 3);
s2=pow(cos(a), 2 );
s3=pow(sin(b), d ); y=(s1*s2)/(5*s3) + k1/k2; cout<<"Искомое значение y="<<y;
system("pause"); //Пауза
}

```

Контрольные вопросы

1. Структура программы на языке C++
2. Директивы препроцессора, заголовочные файлы, прототипы библиотечных функций, их вызовы.
3. Этапы обработки текста программы. Включение текстов из заголовочных файлов.
4. Главная функция программы. Структура функции, ее заголовок.
5. Определение переменных в программе. Типы переменных.
6. Функции форматного ввода-вывода в стиле C.
7. Ввод-вывод данных потоком в стиле C++.
8. Операции, выражения. Оператор присваивания.

Тема 2. Приёмы обеспечения технологичности программных продуктов

Тема 3. Полустатические и динамические структуры данных

Практическое занятие №2 Условные операторы if и switch.

Цель работы – изучение условного оператора и приобретение навыков программирования разветвляющихся алгоритмов.

Краткие теоретические сведения.

Синтаксис записи условного оператора if else:

— сокращенная запись: если условие истинно, т.е. выполняется, то выполняется и тело оператора выбора, иначе выполняется оператор, стоящий

следом за if, т.е. оператор n. Графически запись представлена на рисунке 4.

—\полная запись: если проверяемое условие истинно, то выполняется тело оператора выбора 1, иначе, т. е. проверяемое условие ложно, выполняется тело оператора выбора 2.

Пример выполнения задания.

Задание: по заданным координатам x и y определить, где находится точка (рисунок 6): внутри заштрихованной области; вне заштрихованной области; на границе этой области.

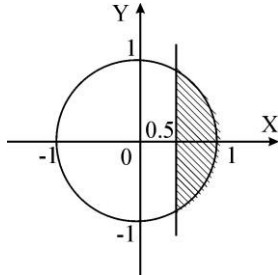


Рисунок 6 — Постановка задачи Метод решения задачи:

Для решения задачи будем использовать уравнение окружности

$x^2 + y^2 = R^2$. Так как $R=1$, то уравнение принимает вид $x^2 + y^2 = 1$.

1. Определяем условие, при котором точка будет находиться внутри заштрихованной области: $(x > 0.5)$ и $(x^2 + y^2 < 1)$.

2. Определяем условие, при котором точка будет находиться вне заштрихованной области: $(x < 0.5)$ или $(x^2 + y^2 > 1)$.

Схема программы представлена на рисунке 7.

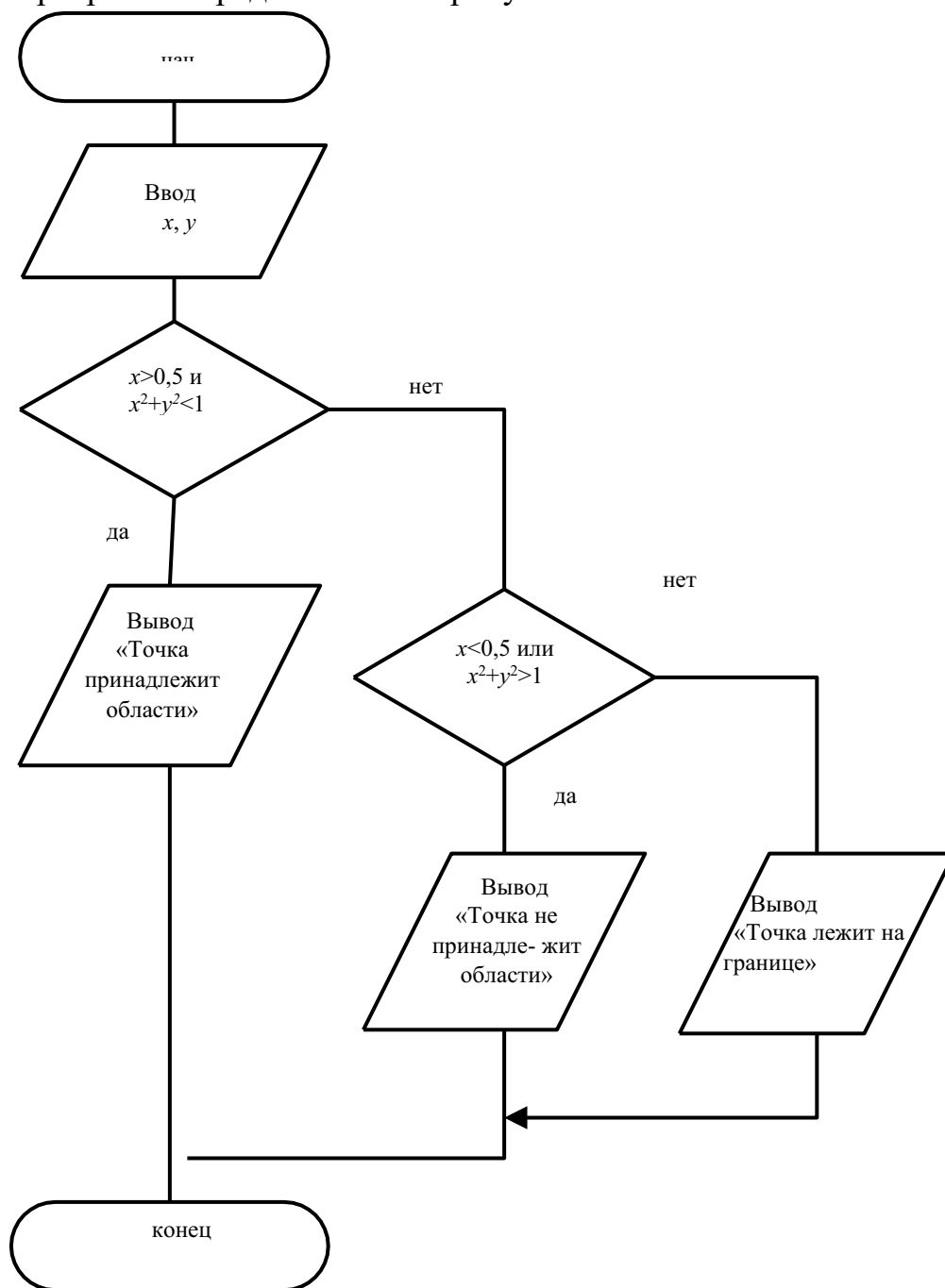


Рисунок 7

Текст программы:

```

#include <iostream> void main ()
{
    double x,y;
    cout<<"Введите координаты точки: "; cin>>x>>y;
    if ((x>0.5)&&(x*x+y*y<1)) cout<<"Точка принадлежит области"; else if
((x<0.5) || (x*x+y*y>1))
        cout<<"Точка не принадлежит области"; else cout<<"Точка лежит на границе
области";
}
  
```

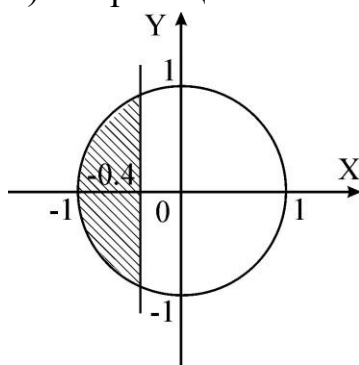
Контрольные вопросы

1. В каких случаях используются условные операторы? Как изображаются условные операторы на схеме программы?
2. Условный оператор в языке C++. Форма записи. Правила выполнения.
3. Истинность и ложность выражений. Значение NULL.
4. Операции конъюнкции, дизъюнкции, отрицания. Знаки операций, их назначение. Какие знаки используются в операциях сравнения?
5. Использование составного оператора в языке C++. Отличие блока от составного оператора.
6. Вложенные операторы if-else.

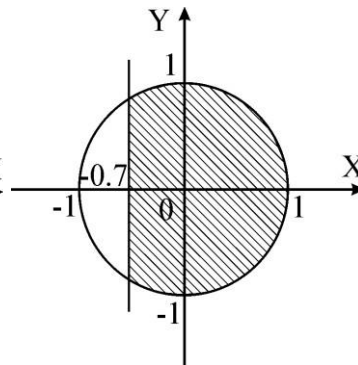
Варианты заданий

По заданным координатам точки определить, где находится точка:

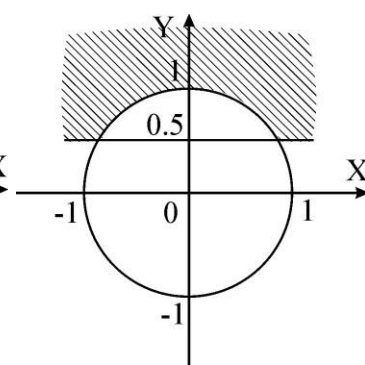
- 1) внутри заштрихованной области;
- 2) вне заштрихованной области;
- 3) на границе этой области.



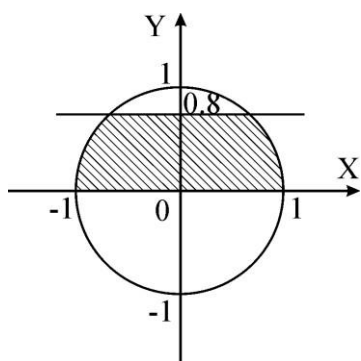
1.



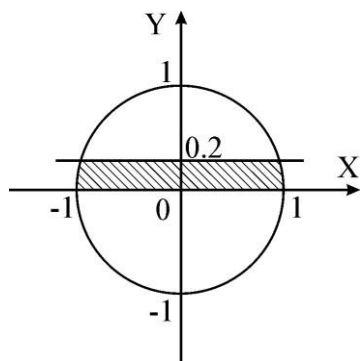
2.



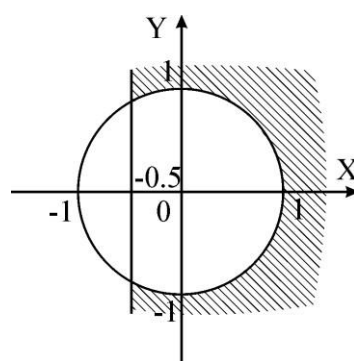
3.



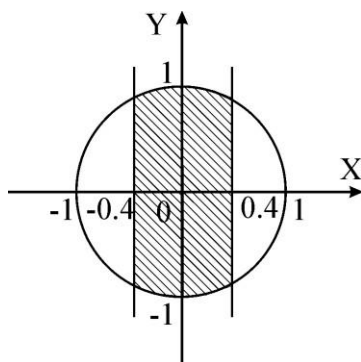
4.



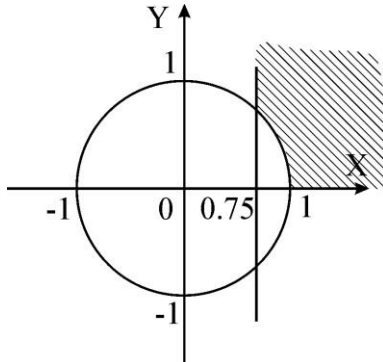
5.



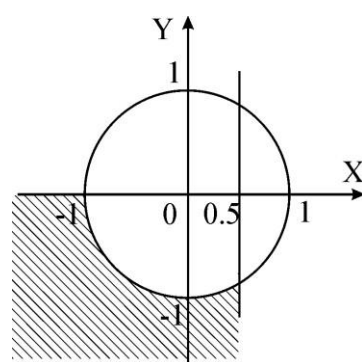
6.



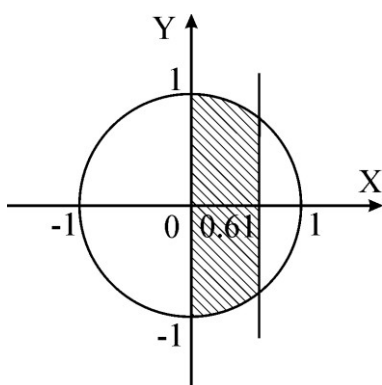
7.



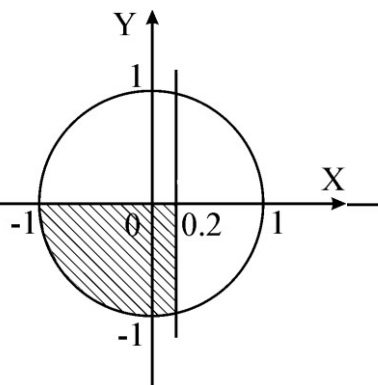
8.



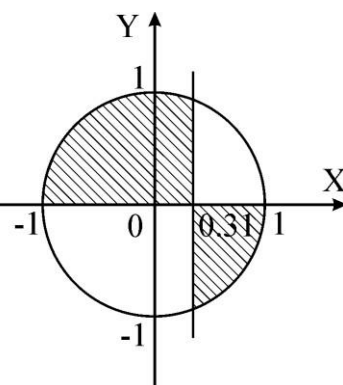
9.



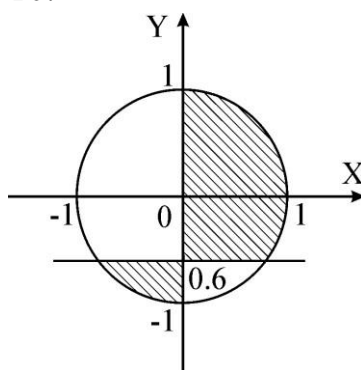
10.



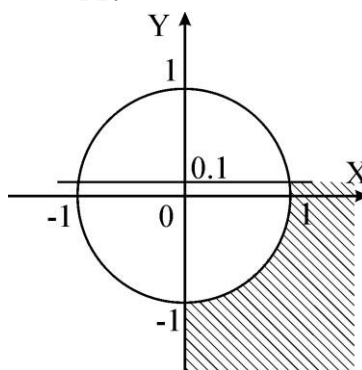
11.



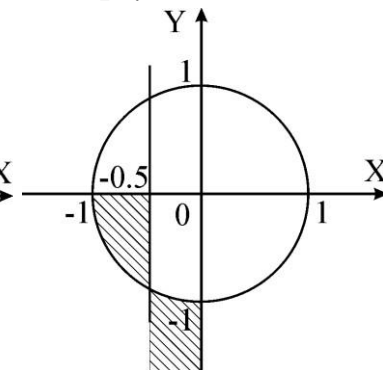
12.



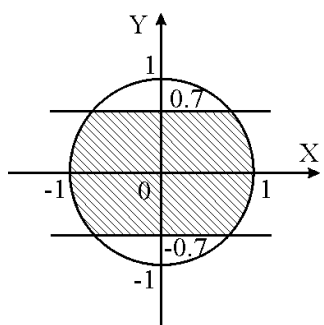
13.



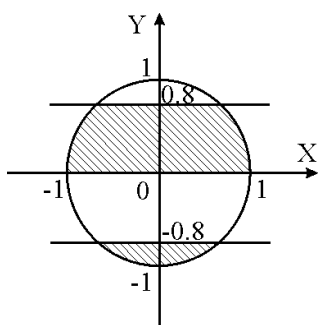
14.



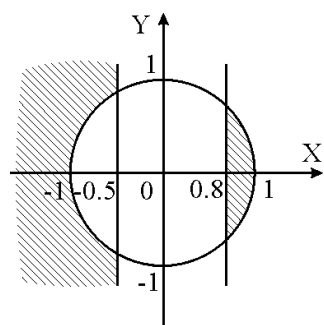
15.



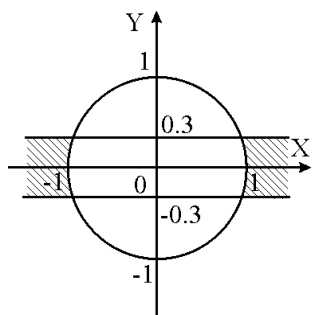
16.



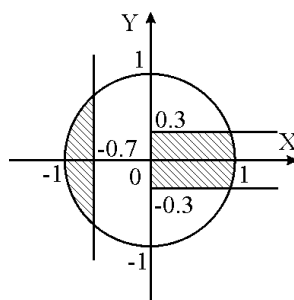
17.



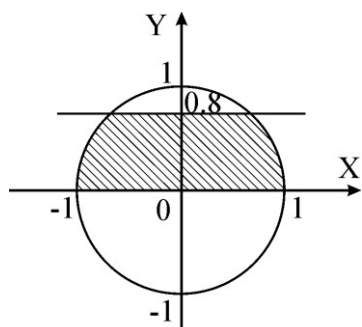
18.



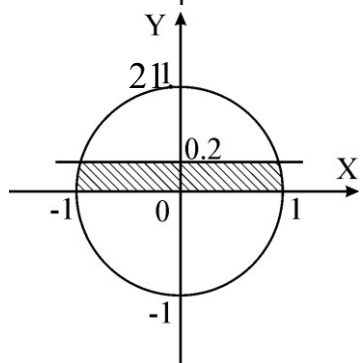
19.



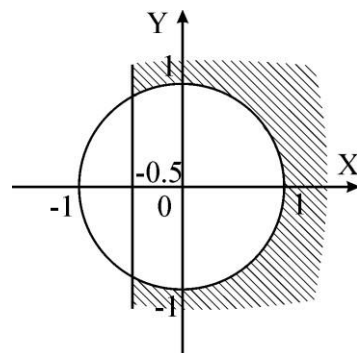
20.



22.



24.



23.

Тема 4. Оценка качества программных систем

Тема 5. Основы анализа алгоритмов

Практическое занятие №3 Программирование с использованием одномерных массивов

Цель работы: освоение средств языка C++ для описания итерационных циклов и закрепление навыков использования их при программировании.

Краткие теоретические сведения.

Иногда необходимо повторять одно и то же действие несколько раз подряд. Для этого используют циклы. Итерационные циклы — это циклы с заранее неизвестным числом шагов. Различают циклы с пред и пост условием. Графически цикл с предусловием представлен на рисунке 8.

Цикл с предусловием будет выполняться, пока условие, указанное в круглых скобках, будет истинным. Синтаксис цикла с предусловием:

Цикл с постусловием `do while` отличается от цикла `while` тем, что в `do while` сначала выполняется тело цикла, а затем проверяется условие продолжения цикла. Из-за такой особенности `do while` называют циклом с постусловием. Таким образом, если условие `do while` заведомо ложное, то хотя бы один раз блок операторов в теле цикла `do while` выполнится. В итоге `do while` отличается от цикла `while` структурой. Если в `while` сначала выполняется проверка условия продолжения цикла, и если условие истинно, то только тогда выполняется тело цикла. Цикл `do while` работает с точностью до наоборот, сначала выполняется тело цикла, а потом проверяется условие, вот почему тело цикла `do while`, хотя бы раз, выполнится. Графическое изображение цикла представлено на рисунке 9.

Пример выполнения задания.

Задание: составить схему программы и программу на языке C++ с использованием оператора цикла с предусловием и оператора цикла с постусловием

$$- \frac{x^7}{7!} + \dots$$

Рисунок 10

Текст программы с использованием оператора **цикла с предусловием**:

```
//Программа вычисления sin x # include <iostream>
# include <math.h> void main ()
{
double x, eps;
cout<<"Введите значения аргумента и точности\n"; cin>>x>>eps;
double F=x, a=x; int n=2;
while (fabs(a)>=eps)
{
a*=-x*x/(n*(n+1)); F+=a;n+=2;
}
cout<<"Приближенное значение sin x="<<F<<endl; cout<<"Точное значение
sin x="<<sin(x);
}
```

Схема программы с использованием цикла с постусловием представлена на рисунке 11.

Текст программы с использованием оператора **цикла с постусловием**:

```
//Программа вычисления sin x # include <iostream.h>
# include <math.h> void main ()
{
double x, eps;
cout<<"Введите значения аргумента и точности\n"; cin>>x>>eps;
double F=x, a=x; int n=2;
do
{
a*=-x*x/(n*(n+1)); F+=a;
n+=2;
}
while (fabs(a)>=eps)
cout<<"Приближенное значение sin x="<<F<<endl; cout<<"Точное значение
sin x="<<sin(x);
}
```

Контрольные вопросы

1. Какие операторы языка C++ используются для организации итерационных циклов?
2. Синтаксис оператора цикла с предусловием.
3. Как выполняется оператор цикла с предусловием?
4. Синтаксис оператора цикла с постусловием.

5. Как выполняется оператор цикла с постусловием?
6. Чем отличаются операторы цикла с предусловием и с постусловием?
7. В каких случаях в операторе цикла используется составной оператор или Блок

Тема 6. Алгоритмы сортировки

Тема 7. Функциональное тестирование программного обеспечения

Практическое занятие №4 Программирование с использованием многомерных массивов

Цель работы: освоение средств языка C++ для описания циклов с параметром и закрепление навыков использования их при программировании, изучение способов описания, ввода-вывода и обработки одномерных массивов.

Краткие теоретические сведения.

Цикл с параметром используется в том случае если известно число шагов обработки, т.е. если известно число итераций. Итерацией цикла называется один проход цикла.

Синтаксис оператора цикла имеет вид:

```
for (действие до начала цикла; условие продолжения цикла;  
действие в конце каждой итерации цикла) {инструкция цикла;  
инструкция цикла 2; инструкция цикла N;  
}
```

Существует частный случай этой записи:

```
for (счетчик = значение; счетчик <значение; шаг цикла)  
{  
тело цикла;  
}
```

Описание синтаксиса:

- 1) сначала присваивается первоначальное значение счетчику;
- 2) затем задается конечное значение счетчика цикла. Счетчик цикла — это переменная, в которой хранится количество проходов данного цикла. После того, как значение счетчика достигнет указанного предела, цикл завершится;
- 3) задаем шаг цикла. Шаг цикла — это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. На рисунке 12 показана структура целочисленного одномерного массива *a*. Размер этого массива — 16 ячеек.

5	-12	-12	9	10	0	-9	-12	-1	23	65	64	11	43	39	-15
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]

Рисунок 12 — Одномерный массив

Максимальный индекс одномерного массива a равен 15, но размер массива 16 ячеек, потому что нумерация ячеек массива всегда начинается с 0. Индекс ячейки – это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой). Синтаксис объявления одномерного массива в C++:

```
тип данных имя одномерного массива [размерность одномерного массива];
//пример объявления одномерного массива, изображенного на рисунке 12:
int a[16];
```

где, `int` — целочисленный тип данных; `a` — имя одномерного массива; 16 — размер одномерного массива, 16 ячеек.

Всегда сразу после имени массива идут квадратные скобочки, в которых задаётся размер одномерного массива, этим массив и отличается от всех остальных переменных.

Рассмотрим на примерах способы ввода-вывода одномерного массива:

1) размер массива можно не указывать только при его инициализации, при обычном объявлении массива обязательно нужно указывать размер массива

```
#include "stdafx.h" #include <iostream> using namespace std;

int main(int argc, char* argv[])
{
    cout << "obrabotka massiva" << endl;
    // объявление и инициализация одномерного массива
    int array1[16] = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -15 };
    cout << "indeks" << "\t\t" << "element massiva" << endl; // печать заголовков for
    (int counter = 0; counter < 16; counter++) //начало цикла
    {
        //вывод на экран индекса ячейки массива, а затем содержимого этой ячейки
        cout << "array1[" << counter << "]" << "\t\t" << array1[counter] << endl;
    }
    system("pause"); return 0;
}
```

Результат работы представлен на рисунке 13.

```

obrabotka massiva
indeks      element massiva
array1[0]   5
array1[1]   -12
array1[2]   -12
array1[3]    9
array1[4]   10
array1[5]    0
array1[6]   -9
array1[7]  -12
array1[8]   -1
array1[9]   23
array1[10]  65
array1[11]  64
array1[12]  11
array1[13]  43
array1[14]  39
array1[15] -15
Для продолжения нажмите любую клавишу . . .

```

Рисунок 13

2) программа должна последовательно считывать десять введенных чисел с клавиатуры. Все введенные числа просуммировать, результат вывести на экран.

```
#include "stdafx.h"
```

```
#include <iostream> using namespace std;
```

```

int main(int argc, char* argv[])
{
    int array1[10]; // объявляем целочисленный массив
    cout << "Enter elements of the array: " << endl;
    int sum = 0;

    for ( int counter = 0; counter < 10; counter++ ) // цикл для считывания чисел
    {
        cin >> array1[counter]; // считываем вводимые с клавиатуры числа
        cout << "array1 = {";

        for ( int counter = 0; counter < 10; counter++ ) // цикл для вывода элементов массива
        {
            cout << array1[counter] << " "; // выводим элементы массива на стандартное устройство вывода
        }

        for ( int counter = 0; counter < 10; counter++ ) // цикл для суммирования чисел массива
        {
            sum += array1[counter]; // суммируем элементы массива
        }

        cout << "sum = " << sum << endl;
        system("pause"); return 0;
    }
}

```

Результат представлен на рисунке 14.

```
Enter elementi massiva:
0
1
2
3
4
5
6
7
8
9
array1 = {0 1 2 3 4 5 6 7 8 9 }
sum = 45
Для продолжения нажмите любую клавишу...
```

Рисунок 14.

Пример выполнения задания.

Задание: произвести следующую обработку 15 целых чисел: подсчитать сумму чисел, входящих в диапазон $[-5..5]$ и количество нечетных чисел.

Схема программы к данному заданию с использованием одного цикла представлена на рисунке 15.

Рисунок 15 Текст программы без использования массива:

```
#include <iostream>

void main()
{
    int x,sum=0,i,kol=0; printf("Enter numbers\n"); for (i=1;i<=15;i++)
    {
        scanf("%d",&x);
        if ((x>=-5)&&(x<=5)) sum+=x; if (x%2!=0) kol++;
    }
    printf("Summa v diapazone [-5,5]=%d\n", sum); printf("Kolichestvo
nechetnih=%d", kol);
}
```

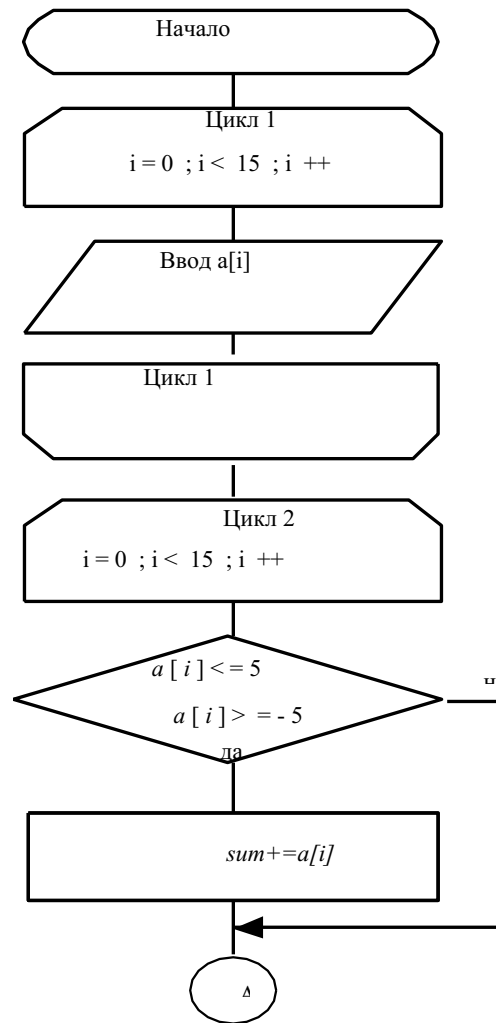


Рисунок 16

Пример программы с использованием одномерного массива

```

#include<iostream> void main()
{
  int a[15],sum=0,i,kol=0; printf("Enter numbers\n");
  for (i=0;i<15;i++) scanf("%d",&a[i]); for (i=0;i<15;i++)
  {
    if ((a[i]>=-5)&&(a[i]<=5)) sum+=a[i]; if (a[i]%2!=0) kol++;
  }
  printf("Summa v diapazone [-5,5]=%d\n", sum); printf("Kolichestvo
nechetnih=%d", kol);
}
  
```

Контрольные вопросы

1. Массивы в языке C++: понятие массива в языке C++, описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.

2. Оператор цикла for в языке C++. Форма записи. Правила выполнения.

Варианты заданий

1. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел, количество нулевых и подсчитать сумму положительных чисел.

2. Произвести следующую обработку 15 целых чисел: найти количество четных чисел, а нечетные числа, входящие в диапазон [1..11] возвести в квадрат.

3. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10] возвести в квадрат.

4. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел.

5. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти сумму чисел, входящих в диапазон [-15..15].

6. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел и подсчитать разность положительных чисел.

7. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое положительных чисел и подсчитать количество чисел, входящих в диапазон [-15..5].

8. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел и подсчитать сумму положительных чисел, делящихся без остатка на 3.

9. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 10.

10. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 3.

11. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое отрицательных чисел и подсчитать количество чисел, входящих в диапазон [0..5].

12. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое нечетных чисел и подсчитать сумму чисел, входящих в диапазон [-15..5].

13. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти синус чисел, входящих в диапазон [-15..15].

14. Произвести следующую обработку 10 целых чисел: подсчитать сумму положительных чисел и определить номера отрицательных чисел.

15. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел и номера нулевых чисел.

16. Произвести следующую обработку 12 целых чисел: подсчитать

количество чисел, делящихся без остатка на 5, и сумму чисел, входящих в диапазон [-5..5].

17. Произвести следующую обработку 10 вещественных чисел: подсчитать количество чисел, отличающихся от числа 3 не более чем на 0.5, и сумму отрицательных чисел.

18. Произвести следующую обработку 15 целых чисел: подсчитать количество нулевых чисел и вычислить квадраты чисел, входящих в диапазон [-5..5].

19. Произвести следующую обработку 12 целых чисел: подсчитать количество нечетных чисел и сумму отрицательных чисел.

20. Произвести следующую обработку 15 вещественных чисел: подсчитать количество чисел, отличающихся от заданного не более чем на 0.5, и сумму положительных чисел.

21. . Произвести следующую обработку 10 вещественных чисел: найти количество отрицательных чисел, находящихся в диапазоне от -5 до 5 и подсчитать сумму положительных чисел.

22. Произвести следующую обработку 15 целых чисел: найти количество чисел, входящих в диапазон [1..11] и каждое число возвести в квадрат.

23. Произвести следующую обработку 15 вещественных чисел: найти количество чисел, равных 0, а числа, входящие в диапазон [-1..1] возвести в куб.

24. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел, входящих в диапазон [-1..0].

25. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, меньших 15, и найти произведение чисел, входящих в диапазон [10..15].

Тема 8. Алгоритмы на графах

Тема 9. Программирование на С#

Практическое занятие №5 Функции

Цель работы: изучение способов описания, ввода-вывода и обработки двумерных массивов, использование указателей при работе с массивами.

Краткие теоретические сведения.

Двумерный массив — это обычная таблица, со строками и столбцами. Фактически двумерный массив — это одномерный массив одномерных массивов. Структура двумерного массива, с именем a , размером m на n показана ниже (см. рисунок 17).

$a[0][0]$	$a[0][1]$	$a[0][2]$	$a[0][3]$	...	$a[0][n]$
$a[1][0]$	$a[1][1]$	$a[1][2]$	$a[1][3]$	...	$a[1][n]$
$a[2][0]$	$a[2][1]$	$a[2][2]$	$a[2][3]$	...	$a[2][n]$
...	...	...	...	...	...
$a[m][0]$	$a[m][1]$	$a[m][2]$	$a[m][3]$	...	$a[m][n]$

Рисунок 17 — Двумерный массивы в С++

где, m — количество строк двумерного массива;
 n — количество столбцов двумерного массива;

$m \times n$ — количество элементов массива.

Наиболее распространенный синтаксис объявления двумерного массива:

`/*тип данных*/ /*имя массива*/[/*количество строк*/][/*количество столбцов*/];`

Примеры инициализации двумерного массива:

— способ 1:

```
int arr[5][3];
```

— способ 2:

```
int arr[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0}, {3, -3, 30}, {1, 1, 1} };
```

В последнем случае представление массива и обращение к элементу массива имеет вид, показанный на рисунке 18,19.

4 a[0][0]	7 [0][1]	8 [0][2]
9 a[1][0]	66 [1][1]	-1 [1][2]
5 a[2][0]	-5 [2][1]	0 [2][2]
3 a[3][0]	-3 [3][1]	30 [3][2]
1 a[4][0]	1 [4][1]	1 [4][2]

Рисунок 18 — Двумерный массив в C++



Рисунок 19 — Обращение к элементу массива

Ввод массива с клавиатуры и его вывод на экран выполняется следующим образом:

```
int m,n,i,j;
cout <<"Введите размеры матрицы:"<<endl; cin>>m;
cin>>n; int elem;
int ar [m][n];
for (i=0; i<m; i++)
{for (j=0; j<n; j++){ cout<<"Введите элемент:"<<endl; cin>>elem;
ar [i][j]=elem;
}
}
for (i=0; i<m; i++)
{for (j=0; j<n; j++){ cout<<ar[i][j]<<endl;
}
}
system("pause"); return 0;
}
```

При работе с массивами можно использовать указатели. Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в C++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Ведь программист заранее не может знать, какого размера нужен массив тому или иному пользователю, в таком случае используется динамическое выделение памяти под массив. Любой указатель необходимо объявить перед использованием, как и любую переменную:

```
/*тип данных*/ * /*имя указателя*/;
```

Работу с указателями можно представить следующим образом:

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

```

using namespace std;

int main(int argc, char* argv[])

{

    int var = 123; // инициализация переменной var числом 123
    int *ptrvar = &var; // указатель на переменную var (присвоили адрес
переменной указателю) cout << "&var      = " << &var << endl; // адрес
переменной var содержащийся в памяти,
извлечённый операцией взятия адреса

    cout << "ptrvar = " << ptrvar << endl; // адрес переменной var, является
значением указателя ptrvar cout << "var      = " << var << endl; // значение в
переменной var

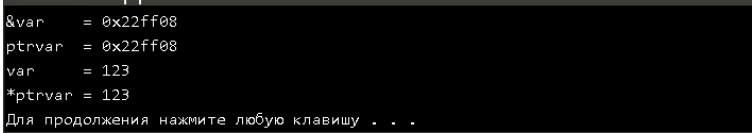
    cout << "*ptrvar = " << *ptrvar << endl; // вывод значения содержащегося в
переменной var через указатель, операцией разыменования указателя

    system("pause"); return 0;

}

```

Результат работы представлен ниже на рисунке 20.



```

&var      = 0x22ff08
ptrvar     = 0x22ff08
var        = 123
*ptrvar    = 123
Для продолжения нажмите любую клавишу . . .

```

Пример выполнения задания.

Задание. Найти максимальную сумму элементов строк матрицы 3×5 .

Написать программы без использования указателей и с использованием указателей.

Схема программы без использования указателей представлена на рисунке 20:

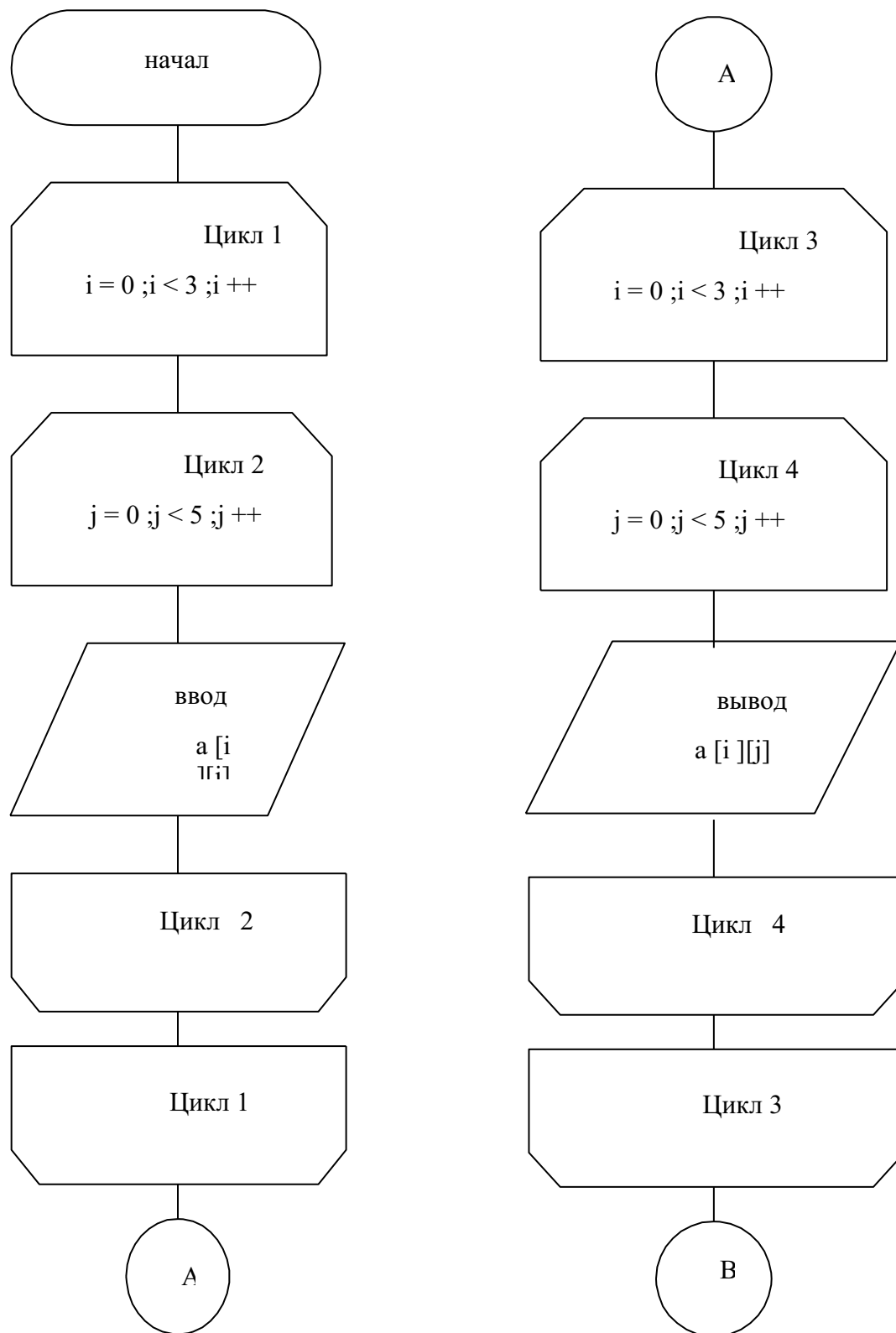
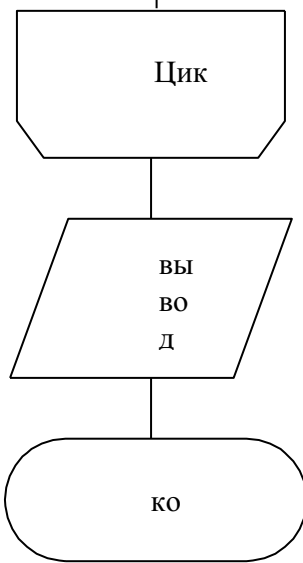
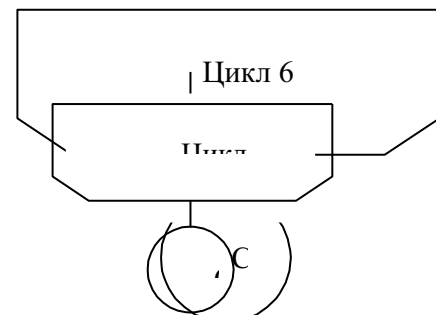
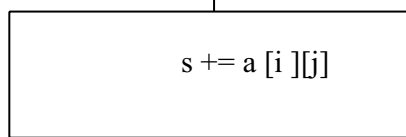
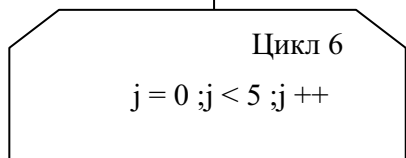
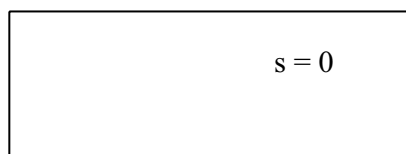
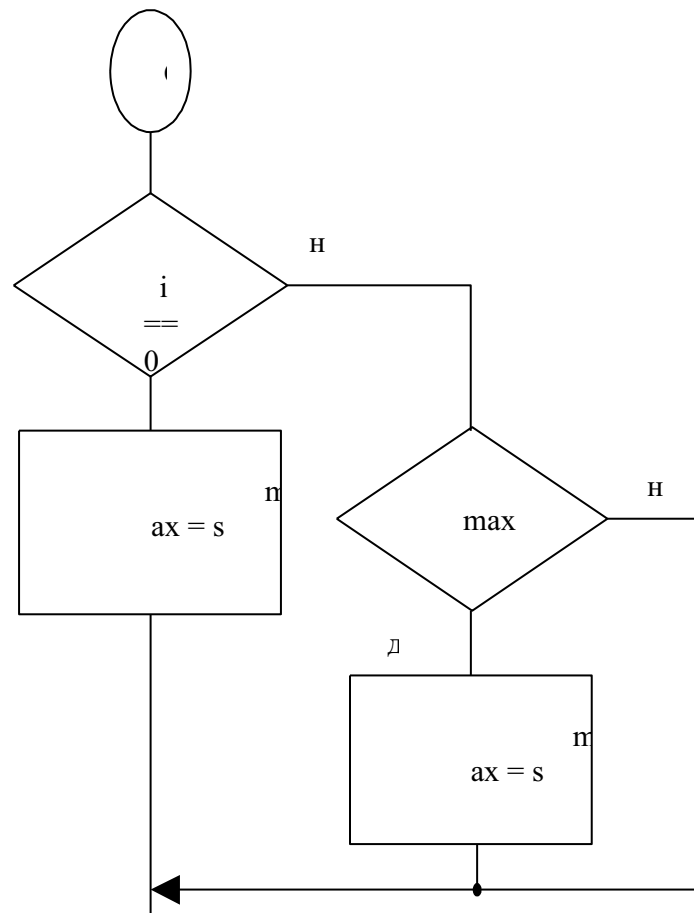
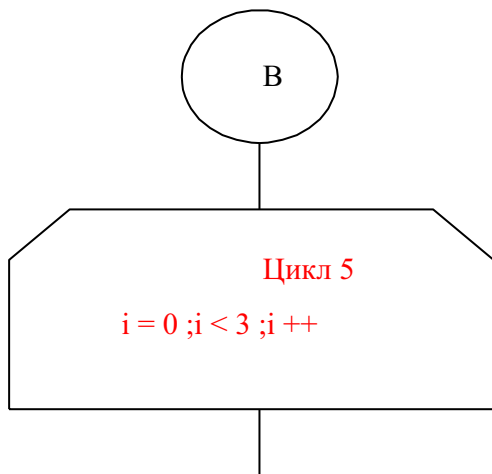


Рисунок 20



Пример программы с использованием указателей:

```

#include <stdio.h> void main()
{

```

```

int a[3][5], *P, i, j, s, max;
printf ("Введите 3 строки по 5 чисел"); for (i=0;i<3;i++)
for (j=0;j<5;j++) scanf("%d",&a[i][j]); printf ("Матрица a :\n"); for (i=0; i<3; i++)
{for (j=0; j<5; j++) printf ("%5d", a[i][j]); printf ("\n");
} P=&a[0][0];
for(i=0;i<3;i++)
{s=0;
for (j=0;j<5;j++)
{s+=*P; P++;
}
if (i==0) max=s;
else if (max<s) max=s;
}
printf("Максимальная сумма строки = %d",max);
}

```

Контрольные вопросы

1. Правила организации вложенных циклов.
2. Особенности организации двумерных массивов: понятие массива в языке C++, описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.
3. Указатели в языке C++: понятие указателя, описание указателя в программе.
4. Операции над указателями.
5. Связь массивов и указателей.

Варианты заданий

1. Вычислить сумму положительных элементов каждого столбца матрицы $A(m \times n)$.
2. Из матрицы $X(m \times n)$ построить матрицу Y , поменяв местами строки и столбцы.
3. Найти наименьший элемент матрицы $X(m \times n)$ и записать нули в ту строку и столбец, где он находится.
4. Переписать первые элементы каждой строки матрицы $A(m \times n)$, большие C , в массив B . Если в строке нет элемента, большего C , то записать ноль в массив B .
5. Дана действительная матрица размера $m \times n$. Найти сумму наибольших значений элементов ее строк.
6. В данной действительной матрице размера $m \times n$ поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением. Предполагается, что такой элемент единственный.

7. В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

8. Дана действительная матрица размера $m \times n$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.

9. Дана целочисленная матрица размера $m \times n$. Найти матрицу, получающуюся перестановкой столбцов (первого с последним, второго с предпоследним и т.д.).

10. Дана целочисленная матрица размера $m \times n$. Найти матрицу, получающуюся перестановкой строк (первой с последней и т.д.).

11. Дана действительная матрица $[a_{ij}]$, где $i, j = 1..n$. Получить действительную матрицу $[b_{ij}]$, где $i, j = 1..n$, элемент b_{ij} которой равен сумме элементов данной матрицы, расположенных в области, определяемой индексами i, j (область заштрихована на рисунке 22):



Рисунок 22

12. Дана действительная квадратная матрица порядка n . Преобразовать матрицу по правилу: строку с номером n сделать столбцом с номером n , а столбец с номером n сделать строкой с номером n .

13. Просуммировать элементы матрицы X ($4,5$), сумма индексов которых равна заданной константе K .

14. Дана матрица M (4×5). Вычислить вектор D , компоненты которого равны сумме элементов строк матрицы.

15. Дана матрица M (6×6). Вычислить сумму элементов главной диагонали.

16. Дана матрица N (6×5). Найти столбец с минимальной суммой элементов.

17. Дана матрица M (4×5) и константа C . Вычислить матрицу D , равную произведению элементов матрицы M на константу.

18. Дана матрица M (4×6). Вычислить вектор D , компоненты которого равны сумме элементов столбцов матрицы.

19. Дана действительная квадратная матрица порядка n , 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.

20. Дана действительная квадратная матрица порядка n . Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы (рисунок 23):

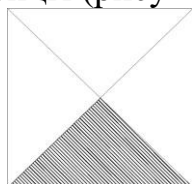


Рисунок 23

21. Дана матрица M (2×5), определить максимальный и минимальный элементы. Поменять местами максимальный и минимальный элементы.

22. В матрице $A(n \times n)$ вычислить сумму элементов матрицы $(n-2 \times n-2)$ и определить максимальный элемент в ней.

23. Дана матрица M (6×6). Вычислить произведение элементов главной диагонали с константой C .

24. Дана матрица N (6×5). Найти строку с минимальной суммой элементов, а элемент с номером n_{ij} возвести в квадрат.

25. Дана матрица вещественных чисел A ($m \times n$). В строке m определить максимальный элемент, а в столбце n количество элементов, меньших порога k .

Тема 10. Программирование на C#

Тема 11. Программирование на C#

Практическое занятие №6 Структуры

Цель работы: изучение правил описания, ввода-вывода и основных функций обработки символьных данных.

Краткие теоретические сведения.

Строки в C++ позволяют работать с символьными данными. С помощью строк возможно осуществить чтение с клавиатуры текста, его обработка и вывод.

В C++ существует 2 типа строк:

1. Массив переменных типа `char`, заканчивающийся нуль-терминатором `\0`. Символьные строки состоят из набора символьных констант заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-терминатора, и отводить дополнительный байт под него.

`char string[10];` // `string` – имя строковой переменной, 10 – размер массива, (в строке может поместиться 9 символов, последнее место отводится под нуль-терминатор)

2. Специальный класс `string`

Для его работы необходимо в начале программы подключить заголовочный файл `string`: `#include <string>`

Для создания строки необходимо в начале программы написать

`using namespace std;`

Теперь чтоб создать строку достаточно написать: `string s;`

Для записи в строку можно использовать оператор `=`

`s="Hello";`

Доступ к `i`-му элементу строки `s` типа `string` осуществляется стандартным образом `s[i]`. Над строками типа `string` определены следующие операции:

- присваивания, например `s1=s2`;
- объединения строк (`s1+=s2` или `s1=s1+s2`) — добавляет к строке `s1` строку `s2`, результат храниться в строке `s1`, пример объединения строк:
- сравнения строк: `s1=s2`, `s1!=s2`, `s1<s2`, `s1>s2`, `s1<=s2`, `s1>=s2` — результатом будет логическое значение.

Существует множество функций для работы со строками (таблица 7).

Таблица 7 – Функции для работы со строками

Функция	Объяснение
<code>s.append(str)</code>	добавляет в конец строки строку <code>str</code>
<code>s.assign(str)</code>	присваивает строке <code>s</code> значение строки <code>str</code>
<code>s.clear()</code>	отчищает строку, т.е. удаляет все элементы в ней
<code>s.compare(str)</code>	сравнивает строку <code>s</code> со строкой <code>str</code> и возвращает 0 в случае совпадение

s.copy(C, I, N)	копирует из строки s в строку C(может быть как строка типа string, так и строка типа char) I символов, начиная с N-го символа
bool b=s.empty()	если строка пуста, возвращает true, иначе false
s.erase(I,N)	удаляет N элементов с I-го символа

s.find(str,I)	ищет строку str начиная с I- го символа
s.insert(pos, s1)	вставляет строку s1 в строку s , начиная с позиции pos
int len=s.length()	записывает в len длину строки
s.push_back(symbol)	добавляет в конец строки символ
s.replace(index, n,str)	берет n первых символов из str и заменяет символы строки s на них, начиная с позиции index
str=s.substr(n,m)	возвращает m символов начиная с позиции n
s.swap(str)	меняет содержимое s и str местами.
s.size()	возвращает число элементов в строке.

Функции для работы со строками, прототипы которых описаны в заголовочном файле string.h:

1. Сравнение строк. Для сравнения строк используются функции strcmp и strncmp. Функция

```
int strcmp ( const char *str1, const char *str2);
```

лексикографически сравнивает строки str1, str2 и возвращает -1, 0 или 1, если строка str1 соответственно меньше, равна или больше строки str2.

Функция

```
int strncmp ( const char *str1, const char *str2, size_t n);
```

лексикографически сравнивает не более чем n первых символов из строк str1 и str2. Функция возвращает -1, 0 или 1, если первые n символов из строки str1 соответственно меньше, равны или больше первых n символов из строки str2.

Пример:

```
// пример сравнения строк
#include
int main() {
char str1[] = "aa bb"; char str2[] = "aa aa"; char str3[] = "aa bb cc";
printf("%d\n", strcmp(str1, str3)); // печатает: -1 printf("%d\n", strcmp(str1, str1));
// печатает: -0 printf("%d\n", strcmp(str1, str2)); // печатает: 1 printf("%d\n",
strncmp(str1, str3, 5)); // печатает: 0 return 1;
}
```

2. Копирование строк. Для копирования строк используются функции strcpy и strncpy. Функция

```
char *strcpy ( char *str1, const char *str2 );
```

копирует строку str2 в строку str1. Строка str2 копируется полностью, включая завершающий нулевой байт. Функция возвращает указатель на str1. Если

строки перекрываются, то результат непредсказуем.

Функция

```
char *strncpy ( char *str1, const char *str2, size_t n );
```

копирует n символов из строки str2 в строку str1. Если строка str2 содержит меньше чем n символов, то последний нулевой байт копируется столько раз, сколько нужно для расширения строки str2 до n символов. Функция возвращает указатель на строку str1.

Пример:

```
char str1[80];
```

```
char str2 = "Copy string."; strcpy ( str1, str2 );
```

```
printf ( str1 ); // печатает: Copy string.
```

4. Соединение строк. Для соединения строк в одну строку используются функции strcat и strncat. Функция

```
char* strcat ( char *str1, const char *str2 );
```

присоединяет строку str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1.

Функция

```
char* strncat ( char *str1, const char *str2, size_t n );
```

присоединяет n символов из строки str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1. если длина строки str2 меньше n, то присоединяются только символы, входящие в строку str2. После соединения строк к строке str1 всегда добавляется нулевой байт. Функция возвращает указатель на строку str1.

Пример:

```
#include <stdio.h>
```

```
#include <string.h> int main ( )
```

```
{
```

```
char str1[80] = "String "; char str2 = "catenation "; char str3 = "Yes No"; strcat ( str1, str2 );
```

```
printf ("%s\n", str1 ); // печатает: String catenation strncat ( str1, str3, 3 );
```

```
printf ("%s\n", str1 ); // печатает: String catenation Yes return 1; }
```

5. Поиск символа в строке. Для поиска символа в строке используются функции strchr, strrchr, strspn, strcspn и strpbrk. Функция

```
char* strchr ( const char *str, int c );
```

ищет первое вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на первый найденный символ, а в случае неудачи – NULL.

Функция

```
char* strrchr ( const char *str, int c );
```

ищет последнее вхождение символа, заданного параметром *c*, в строку *str*. В случае успеха функция возвращает указатель на последний найденный символ, а в случае неудачи – NULL.

Пример:

```
#include <stdio.h> #include <string.h> int main ( )
{
    char str[80] = "Char search";
    printf ("%s\n", strchr ( str, 'r' ));           // печатает: r search printf
    ("%s\n", strrchr ( str, 'r' )); // печатает: rch return 1;
}
```

Функция

size_t strspn (const char *str1, const char *str2);

возвращает индекс первого символа из строки str1, который не входит в строку str2.

Функция

size_t strcspn (const char *str1, const char *str2);

возвращает индекс первого символа из строки str1, который входит в строку str2.

Пример:

```
int main ( )
{
    char str[80] = "123 abc";
    printf ("n = %d\n", strspn ( str, "321" )); // печатает: n = 3 printf ("n = %d\n",
    strcspn ( str, "cba" )); // печатает: n = 4 return 1;
}
```

Функция

char* strpbrk (const char *str1, const char *str2);

находит первый символ в строке str1, который равен одному из символов в строке str2. В случае успеха функция возвращает указатель на этот символ, а в случае неудачи – NULL.

Пример.

```
char str[80] = "123 abc";
printf ("%s\n", strpbrk ( str, "bca" )); // печатает: abc
```

6. Сравнение строк. Для сравнения строк используются функция strstr.

Функция

char* strstr (const char *str1, const char *str2);

находит первое вхождение строки str2 (без конечного нулевого байта) в строку str1. В случае успеха функция возвращает указатель на найденную подстроку, а в случае неудачи – NULL. Если указатель str1 указывает на строку нулевой длины, то функция возвращает указатель str1.

Пример:

```
char str[80] = "123 abc 456;  
printf ("%s\n", strstr ( str, "abc")); // печать: abc 456
```

7.Разбор строки на лексемы. Для разбора строки на лексемы используется функция strtok. Функция

```
char* strtok ( char *str1, const char *str2 );
```

возвращает указатель на следующую лексему (слово) в строке str1, в которой разделителями лексем являются символы из строки str2. В случае если лексемы закончились, то функция возвращает NULL. При первом вызове функции strtok параметр str1 должен указывать на строку, которая разбирается на лексемы, а при последующих вызовах этот параметр должен быть установлен в NULL. После нахождения лексемы функция strtok записывает после этой лексемы на место разделителя нулевой байт.

Пример.

```
#include <stdio.h> #include <string.h> int main( )  
{  
char str[ ] = "12 34 ab cd"; char *p;  
p = strtok ( str, " " ); while (p)  
{  
printf ( "%s\n", p ); // печатает в столбик значения: 12 34 ab cd p = strtok (  
NULL, " " );  
}  
return 1;  
}
```

8.Определение длины строки. Для определения длины строки используется функция strlen. Функция

```
size_t strlen ( const char *str);
```

возвращает длину строки, не учитывая последний нулевой байт. Например,
char str[] = "123";

```
printf ("len = %d\n", strlen ( str )); // печатает: len = 3
```

Пример выполнения задания.

Задание. Найти слова во введенной с клавиатуры строке, вывести их на экран и подсчитать их количество.

Пример программы:

```
#include<stdio.h> #include<string.h>  
vid main()  
{  
char s[100],d[100]; int i=0,j=0,bw,ew,len; gets(s); len=strlen(s); while (i<len)  
{  
while((s[i]==' ')&&(i<len)) i++; bw=i;  
while((s[i]!=' ')&&(i<len)) i++; ew=i;  
strncpy(d,&s[bw],ew-bw+1); d[ew-bw+1]=0;
```

```

if (bw<len)
{ j++;
printf("%s\n",d);}
}
printf("Vsego slov %d\n", j);
}

```

Контрольные вопросы

1. Строки в языке C++: понятие строки, описание строк в программе, обращение к элементам строки.
2. Три способа ввода строк в C++.
3. Три способа вывода строк в C++.
4. Способы инициализации строк (задание значений в программе).
5. Стандартные функции для обработки строк.

Варианты заданий

Дана последовательность символов $S_1, \dots, S_N$. Группы символов, разделенные пробелом (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами.

1. Определить число символов в самом длинном слове строки. Слова отделяются знаком “/”.
2. В произвольном тексте выделить и отпечатать слова, начинающиеся с буквы А.
3. В произвольном тексте вставить между первым и вторым словом новое слово.
4. В произвольном тексте найти и отпечатать слова, содержащие букву Е.
5. Отпечатать второе и третье слова произвольного текста.

6. В произвольном тексте вставить между вторым и третьим словом новое слово.

7. В произвольном тексте найти и отпечатать все слова длиной 5 символов.

8. В произвольном тексте найти самое короткое слово.

9. В последовательности из 10 пятибуквенных слов найти и поменять местами пару слов, у которых первые три буквы одного совпадают с последними тремя буквами другого.

10. Упорядочить в алфавитном порядке последовательность из 10 пятибуквенных слов.

11. В строке из 50 символов отдельные слова разделены пробелом. Упорядочить строку так, чтобы каждое следующее слово было не короче предыдущего.

12. Расположить слова строки в порядке, обратном исходному.

13. Подсчитать количество букв 'а' в последнем слове строки.

14. Найти количество слов, у которых первый и последний символы совпадают между собой.

15. Исключить из строки слова, расположенные между скобками (,). Сами скобки должны быть исключены.

16. В произвольном тексте найти и отпечатать слова, содержащие букву А.

17. Отпечатать первое и второе слова произвольного текста.

18. В произвольном тексте вставить после первого слова новое слово.

19. В произвольном тексте найти и отпечатать все слова длиной 4 символа.

20. В произвольном тексте найти самое длинное слово.

21. Выполнить сравнение двух строк s и d. Результат вывести в виде сообщения «идентичны» или «не идентичны».

22. Добавить в конец строки новое слово, длиною 5 символов, иначе выдать сообщение об ошибке.

23. Добавить в начало строки новое слово, начинающееся с буквы a, иначе, если слово начинается с другой буквы вывести сообщение о невозможности добавления.

24. Посчитать какое количество раз встречается буква n (задается при каждом выполнении алгоритма).

25. Проанализировать массив символов, состоящий из n символов. Если массив состоит из n-5 символов, добавить в конец набор символов gttt.

Тема 12. Программирование на C#

Практическое занятие №7 Указатели

Цель работы: изучение правил составления и использования функций в программах на C++.

Краткие теоретические сведения.

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определённую цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь неэкономичной траты времени была предложена концепция подпрограммы.

Подпрограмма — именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке C++ подпрограммы реализованы в виде функций.

Функция — это поименованный набор описаний и операторов, выполняющих определённую задачу. Функция может принимать параметры и возвращать значение. Информация, передаваемая в функцию для обработки, называется параметром, а результат вычисления функции её значением. Обращение к функции называют вызовом.

Описание функции состоит из заголовка и тела функции:

```
тип имя_функции(список_переменных)
{
    тело_функции
}
```

Заголовок функции содержит:

- тип возвращаемого функцией значения, он может быть любым; если функция не возвращает значения, указывают тип void;

- имя_функции;

- список_переменных — перечень передаваемых в функцию величин (аргументов), которые отделяются друг от друга запятыми; для каждой переменной из списка указывается тип и имя; если функция не имеет аргументов, то в скобках указывают либо тип void, либо ничего.

Тело функции представляет собой последовательность описаний и операторов, заключённых в фигурные скобки.

В общем виде структура программы на C++ может иметь вид:

директивы компилятора

```
тип имя_1(список_переменных)
{
    тело_функции_1;
}
тип имя_2(список_переменных)
{
    тело_функции_2;
}
...
```

```

тип имя_n(список_переменных)
{
тело_функции_n;
}
int main (список_переменных)
{
//Тело функции может содержать операторы вызова
//функций имя_1, имя_2, ..., имя_n тело_основной_функции;
}
Однако допустима и другая форма записи программного кода :
директивы компилятора
тип имя_1(список_переменных); тип имя_2(список_переменных);
...
тип имя_n(список_переменных); int main (список_переменных)
{
//Тело функции может содержать операторы вызова
// функций имя_1, имя_2, ..., имя_n тело_основной_функции;
}
тип имя_1(список_переменных)
{
тело_функции_1;
}
тип имя_2(список_переменных)
{
тело_функции_2;
}
...
тип имя_n(список_переменных)
{
тело_функции_n;
}

```

Здесь функции описаны после функции main(), однако до неё перечислены заголовки всех функций. Такого рода опережающие заголовки называют прототипами функций. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов, и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе.

Вызвать функцию можно в любом месте программы. Для вызова функции необходимо указать её имя и в круглых скобках, через запятую перечислить имена или значения аргументов, если таковые имеются:

```
имя_функции(список_переменных);
```

Рассмотрим пример. Создадим функцию f(), которая не имеет входных значений и не формирует результат. При вызове этой функции на экран выводится строка символов "С Новым Годом, ".

```
#include <iostream> using namespace std;
void f ( ) //Описание функции.
```

```

{
cout << "С Новым Годом, ";
}
int main ( )
{
f ( ); //Вызов функции. cout <<"Студент!" << endl; f ( ); //Вызов функции.
cout <<"Преподаватель!" << endl; f ( ); //Вызов функции.
cout <<"Народ!" << endl;
}

```

Результатом работы программы будут три строки:

С Новым Годом Студент!

С Новым Годом Преподаватель!

С Новым Годом Народ!

Далее приведён пример программы, которая вычисляет значение выражения $\sin^2(\alpha) + \cos^2(\alpha)$ при заданном значении α . Здесь функция `radian` выполняет перевод градусной меры угла в радианную¹.

```

#include <iostream> #include <math.h> #define PI 3.14159 using namespace std;
double radian ( int deg, int min, int sec )
{
return ( deg * PI/180+min* PI /180/60+ sec * PI /180/60/60);
}
int main ( )
{
int DEG, MIN, SEC; double RAD;
//Ввод данных.
cout<<" Input :"<<endl; //Величина угла: cout<<" DEG ="; cin >>DEG;
//градусы, cout<<" MIN ="; cin >>MIN; //минуты, cout<<" SEC ="; cin >>SEC;
//секунды.
//Величина угла в радианах.
RAD=radian (DEG, MIN, SEC); //Вызов функции. cout << " Value in radian
A="<<RAD << endl;
//Вычисление значения выражения и его вывод. cout << " sin (A) ^2+ cos (A)
^2= ";
cout << pow( sin (RAD), 2 )+pow( cos (RAD), 2 ) << endl; return 0;
}

```

Переменные, описанные внутри функции, а также переменные из списка аргументов, являются локальными. Например, если программа содержит пять разных функций, в каждой из которых описана переменная `N`, то для C++ это пять различных переменных. Область действия локальной переменной не выходит за рамки функции. Значения локальных переменных между вызовами одной и той же функции не сохраняются.

Переменные, определённые до объявления всех функций и доступные всем функциям, называют глобальными. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем. Глобальные

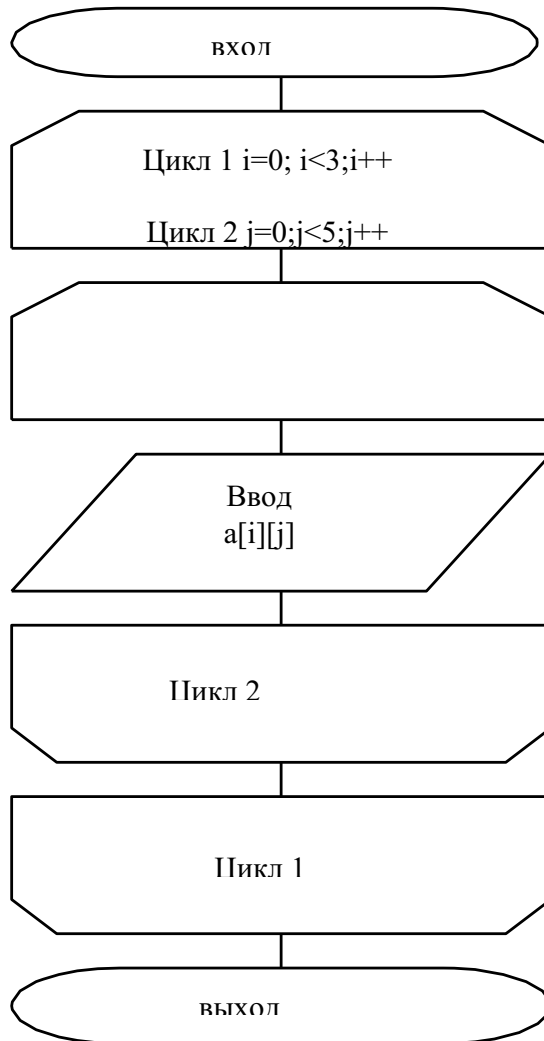
переменные применяют для передачи данных между функциями, но это затрудняет отладку программы. Для обмена данными между функциями используют параметры функций и значения, возвращаемые функциями.

Пример выполнения задания.

Задание. Найти максимальную сумму элементов строк матрицы 3×5 с использованием функций.

Схемы программ функций ввода-вывода и обработки данных представлены на рисунках 25 и 26. Схема главной функции main представлена на рисунке 27:

Inparr (a[m][n])



outarr (a[m][n])

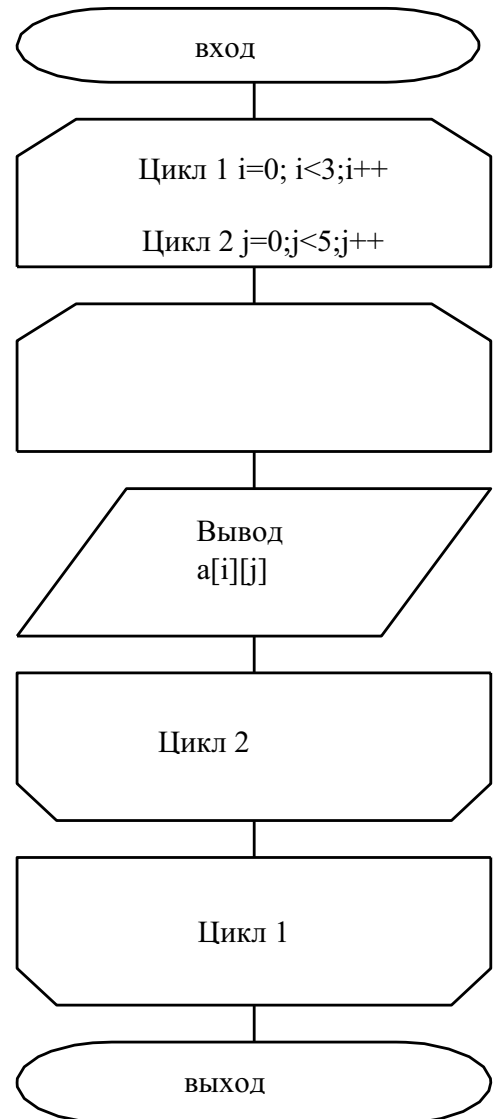


Рисунок 25 – Функции ввода и вывода массива данных

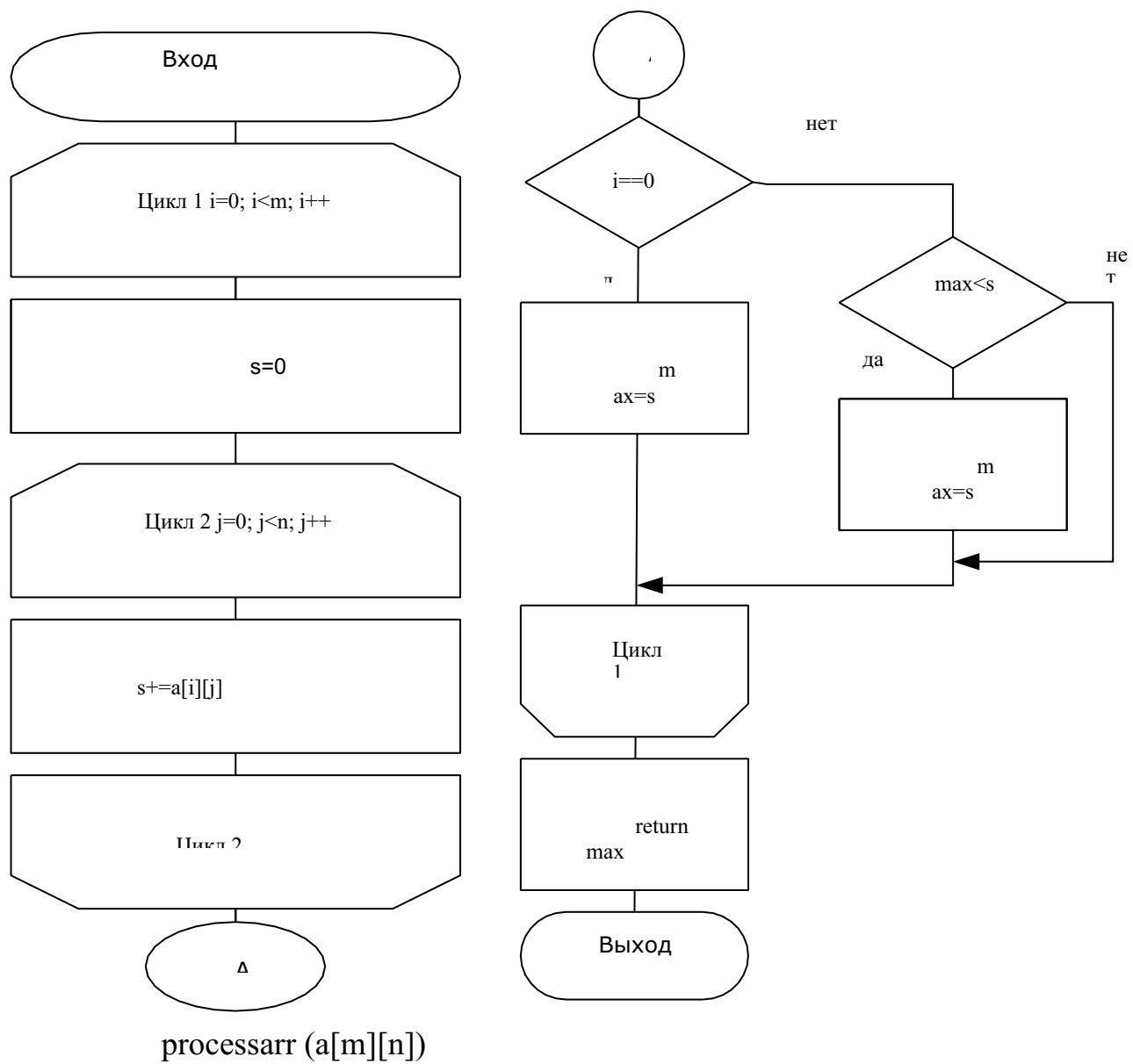


Рисунок 26 – Функция обработки массива данных

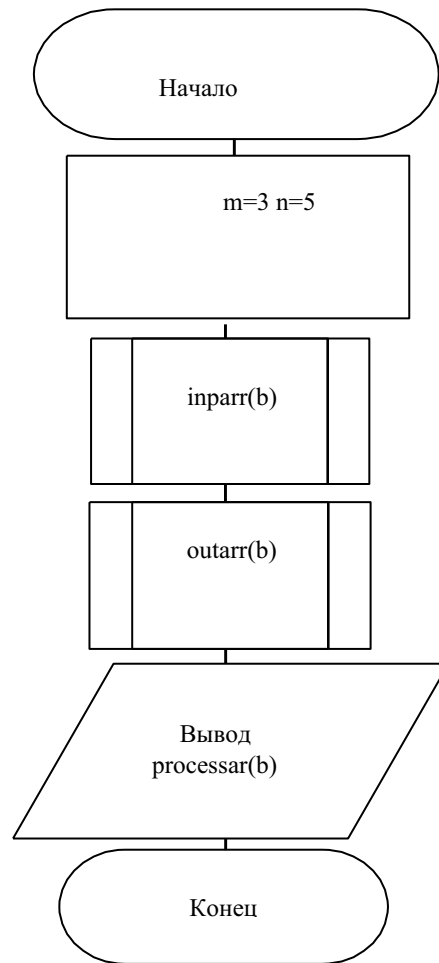


Рисунок 27 – Главная функция программы main

Пример программы:

```

#include <stdio.h> const int m=3, n=5;
//функция ввода
void inparr(int a[m][n])
{
  int i,j;
  for (i=0;i<m;i++) for (j=0;j<n;j++)
    scanf("%d",&a[i][j]);
}
//функция вывода
void outarr (int a[m][n])
{
  int i,j; printf("Matrica:\n"); for (i=0; i<m; i++)
  {
    for (j=0; j<n; j++) printf("%5d", a[i][j]); printf("\n");
  }
}
int processarr(int a[m][n])
{

```

```
int i,j,s,max; for(i=0;i<m;i++)
{
s=0;
for (j=0;j<n;j++) s+=a[i][j];
if (i==0) max=s;
else if (max<s) max=s;
}
return max;
}
void main()
{
int b[m][n]; inparr(b); outarr(b);
printf("Maximalnaya summa stroki = %d", processarr(b));
}
```

Контрольные вопросы

1. Структура функции в языке C++, ее заголовок.
2. Вызов функции.
3. Способы передачи параметров.
4. Оператор return (2 формы записи).
5. Описание функции (прототип).

Практическое занятие 8. Запись и чтение данных файла

Практическое занятие 9. Операторы цикла for и while

Практическое занятие 10. Введение в ООП

Практическое занятие 11. Классы. Связные списки

Практическое занятие 12. Перегрузка методов и конструкторов

Практическое занятие 13. Перегрузка операторов

Практическое занятие 14. Наследование

Практическое занятие 15. Множественное наследование

Практическое занятие 16. Виртуальные функции

Практическое занятие 17. Шаблоны и исключения

Варианты заданий

См. задание к лабораторной работе № 5.

Практическое занятие №8 Реализация классов

Цели:

Изучение структуры класса, механизм создания и использования, описание членов- данных класса и методов доступа к ним.

1. Краткие теоретические сведения Компоненты класса

Класс - это определяемый пользователем тип. Описание класса очень похоже на описание структуры в Си. В этом смысле класс является расширением понятия структуры. В простейшем случае класс можно определить с помощью конструкции:

```
тип_класса имя_класса {список_членов_класса};
```

, где

тип_класса – одно из служебных слов **class**, **struct**, **union**;

имя_класса – идентификатор;

список_членов_класса – определения и

описания типизированных данных и принадлежащих классу функций.

Функции – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Рассмотрим реализацию понятия даты с использованием struct для того, чтобы определить представление даты и множества функций для работы с переменными этого типа:

```
struct date {  
    int month, day, year; // дата: месяц, день, год  
    void set(int, int, int);  
    void get(int*, int*, int*); void next();  
    // ...  
};
```

Функции, описанные таким образом, называются функциями-членами и могут

вызываться только для специальной переменной соответствующего типа с использованием стандартного синтаксиса для доступа к членам структуры.

Например:

```
date today; // сегодня  
void f()  
{  
    today.set(18,1,1985); today.next();  
}
```

Поскольку разные структуры могут иметь функции члены с одинаковыми именами, при определении функции члена необходимо указывать имя структуры:

```
void date::next()
{
    if ( ++day > 28 ) {
        // делает сложную часть работы
        ...
    }
}
```

В функции-члене имена членов структуры могут использоваться без явной ссылки

на объект. В этом случае имя относится к члену того объекта, для которого функция была вызвана.

Описание date в предыдущем подразделе дает множество функций для работы с date, но не указывает, что эти функции должны быть единственными для доступа к объектам типа date. Это ограничение можно наложить, используя вместо struct class:

```
class date { int month, day, year; public:
void set(int, int, int); void next();
};
```

Методы доступа к полям (геттеры и сеттеры)

```
class TPen
{    private:
    string FColor;

                                public:
                                string getColor ();
                                void setColor ( string newColor );

};
```

Получить значение:

```
string TPen::getColor () {
```

```
void TPen::setColor ( string newColor )
```

```
return FColor;    } Записать значение:
```

```
{ if ( newColor.length() != 6 )
    FColor = "000000";
```

```
else
```

```
FColor = newColor;
```

```
}
```

Для описания объекта класса (экземпляра класса) используется конструкция ***имя_класса имя_объекта;***

Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа: **public, private, protected**.

Общедоступные (public) компоненты класса доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его. Доступ извне осуществляется через имя объекта:

***имя_объекта.имя_члена_класса ссылка_на_объект.имя_члена_класса
указатель_на_объект->имя_члена_класса***

Собственные (private) компоненты класса локализованы в классе и не доступны извне. Они могут использоваться функциями – членами данного класса и функциями – “друзьями” того класса, в котором они описаны.

Защищенные (protected) компоненты доступны внутри класса и в производных классах.

В том, что доступ к структуре данных ограничен явно описанным списком функций, есть несколько преимуществ. Любая ошибка, которая приводит к тому, что дата принимает недопустимое значение (например, Декабрь 36, 1985), должна быть вызвана кодом функции-члена, поэтому первая стадия отладки, локализация, выполняется еще до того, как программа будет запущена.

Защита закрытых данных связана с ограничением использования имен членов класса. В функции-члене на члены объекта, для которого она была вызвана, можно ссылаться непосредственно. Например:

```
class x {  
    int m; public:  
    int readm() { return m; }  
};  
x aa; x bb;  
void f(){  
  
                                int a = aa.readm(); int b =  
                                bb.readm();  
                                // ...  
}
```

В первом вызове члена `readm()` `m` относится к `aa.m`, а во втором - к `bb.m`.

Указатель на объект, для которого вызвана функция-член, является скрытым параметром функции. В каждой функции класса `x` указатель `this` неявно описан как

```
x* this;
```

и инициализирован так, что он указывает на объект, для которого была вызвана функция член. `this` не может быть описан явно, так как это ключевое слово.

2. Задание

2.1 Общее задание (50%)

1. Создайте консольный проект. В теле функции `main` будет выполняться демонстрация работы объекта.

2. Добавьте в проект новый класс и назовите этот класс `Worker`. В класс добавьте два общедоступных поля: имя и возраст и одно скрытое поле: вес:

```
class Worker {  
    public:
```

```
    int age; char* name;
```

```
    private:
```

```
    float weight;
```

```
};
```

В теле функции `main` создайте объект класса `Worker`: `Worker *wrk1 = new Worker ();`

```
wrk1->age = 34;
```

```
wrk1->name = "Иванов";
```

Добавьте оператор(функцию) вывода на экран созданного объекта.

3. Запустите программу на выполнение.

4. Попробуйте записать значение в поле `weight`. Почему данные не записались?

5. Для записи и чтения данных из скрытых полей используют методы. Добавим во внутрь класса `Worker` новый метод (действие), который будет отвечать за еду, если человек чего-то там съест, то его вес должен будет увеличиться на количество съеденного.

в структуру класса: `public:`

```
...
```

```
void eat (float how_much);
```

после описания класса, но до функции `main`: `void Worker::eat (float how_much){`

```
    weight = weight + how_much;
```

```
}
```

6. Если поле вес скрытое, то мы в него не только писать не можем, но и читать тоже не можем. Для чтения данных из скрытого поля необходимо использовать еще один метод:

в структуру класса: public:

...

```
float get_weight();
```

```
после описания класса, но до функции main: float Worker::get_weight() {  
return weight;  
}
```

7. Почему в последних двух функциях после слова public идут различные слова? Что они обозначают и на что влияют?

8. Теперь эти два метода надо использовать в нашей программе. Заставьте рабочего съесть 2, а затем 3 кг пищи. Проверьте его вес.

```
wrk1->eat(2);
```

```
wrk1->eat(3); float ves;
```

```
ves = wrk1->get_weight();
```

 Отобразите результат на экран.

9. Запустите программу на выполнение. Проверьте работоспособность. Добавьте комментарии.

10. Усовершенствуйте метод eat таким образом, что если рабочий за раз съедает более чем 10 кг, то его возраст увеличивается на год, а вес увеличивается только на половину от съеденного.

11. Попросите рабочего съесть 15 кг и посмотрите на результат работы программы.

12. Измените программу так, чтобы имя рабочего и его первоначальный возраст вводились с клавиатуры и вносились в соответствующие переменные.

13. Запустите программу. Проверьте ее работоспособность.

14. Добавьте рабочему еще одно скрытое поле, которое будет отвечать за настроение и будет иметь первоначальное значение равное 10.

15. Добавьте три метода: гулять (метод должен увеличивать настроение на 1), танцевать (метод должен увеличивать настроение на 2) и работать (метод должен уменьшать настроение на 2).

16. Дополните основную программу так, чтобы рабочий после еды два раза погулял и три раза потанцевал.

17. Добавьте в класс функцию, которая будет возвращать текущее настроение пользователя.

18. Добавьте в основную программу метод работать 9 раз (можно в цикле) и выведите настроение пользователя на экран.

19. Настроение получилось отрицательным? – ужасно. Измените метод

работать таким образом, чтобы настроение никогда не было меньше нуля (т.е. если настроение было 1 и человек поработал, то оно должно стать не меньше 0).

20. Проверьте заново работоспособность программы.

2.2 Индивидуальное задание (50%)

- Реализовать пользовательский класс в соответствии с вариантом задания.
- При реализации классов поля должны быть скрытыми.
- Определить метод установки свойств (при недопустимых аргументах функции возвращать «false» и выдавать текст ошибки на экран).
- Определить метод чтения свойств.
- Написать демонстрационную программу, в которой показать использование объектов созданного класса.

3. Список рекомендуемой литературы

1. Объектно-Ориентированное программирование: учебник / Г. С. Иванова, Т. Н. Ничушкина ; под общ. ред. Г. С. Ивановой. — М. : Изд-во МГТУ им. Н. Э. Баумана, 2014.
— 455.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9.
Paperback 984 pages. September 2014

4. Контрольные вопросы

1. В определении класса члены класса с ключевым словом `private` доступны:
 - а) любой функции программы;
 - б) в случае, если известен пароль;
 - в) методам этого класса;
 - г) только открытым членам класса.
2. Напишите определение класса `studentgroup`, включающего одно закрытое поле типа `int` с именем `number` и одним открытым методом с прототипом `void add()`.
3. Истинно ли следующее утверждение: поля класса должны быть закрытыми?
4. Для чего при работе с объектами применяется операция «точка»?
5. Для чего при работе с объектами применяется операция «стрелка»?
6. Методу класса всегда доступны данные:
 - а) объекта, членом которого он является;
 - б) класса, членом которого он является
 - в) любого объекта класса, членом которого он является;
7. Что является единственным формальным различием между структурами и классами в C++?
8. Пусть определены три объекта класса. Сколько копий полей класса содержится в памяти? Сколько копий методов класса?

9. Для чего необходимо переопределять операции new и delete?
10. Что такое указатель this?

Практическое занятие №9 Конструкторы и деструкторы

Цели:

Изучение возможности инициализации объектов класса с помощью конструкторов и уничтожение их с помощью деструкторов

1. Краткие теоретические сведения

При создании объектов одной из наиболее широко используемых операций является инициализация элементов данных объекта. Единственным способом, с помощью которого вы можете обратиться к частным элементам данных, является использование функций класса. Чтобы упростить процесс инициализации элементов данных класса, C++ использует специальную функцию, называемую конструктором, которая запускается для каждого создаваемого вами объекта. Подобным образом C++ обеспечивает функцию, называемую деструктором, которая запускается при уничтожении объекта. Основными концепциями конструктора и деструктора являются:

- Конструктор представляет собой метод класса, который облегчает инициализацию элементов данных класса.
- Конструктор имеет такое же имя, как и класс.
- Конструктор не имеет возвращаемого значения.
- Каждый раз, когда программа создает переменную класса, C++ вызывает конструктор класса, если конструктор существует.
- Многие объекты могут распределять память для хранения информации; когда вы уничтожаете такой объект, C++ будет вызывать специальный деструктор, который может освобождать эту память, очищая ее после объекта.
- Деструктор имеет такое же имя, как и класс, за исключением того, что вы должны предварять его имя символом тильды (~).
- Деструктор не имеет возвращаемого значения.

Представьте конструктор как функцию, которая помогает строить (конструировать) объект. Подобно этому, деструктор представляет собой функцию, которая помогает уничтожать объект. Деструктор обычно используется, если при уничтожении объекта нужно освободить память, которую занимал объект.

Создание простого конструктора

Конструктор представляет собой метод класса, который имеет такое же имя, как и класс. Например, если вы используете класс с именем `employee`, конструктор также будет иметь имя `employee`. Конструктор не возвращает никакого значения несмотря на то, что он не объявляется как `void`.

```
class employee
{
```

```

public:
employee(char *, long, float); //Конструктор void show_employee(void);
int change_salary(float); long get_id(void);
private:
char name [64]; long employee_id; float salary;
};

employee::employee(char *name, long employee_id, float salary)
{
strcpy(employee::name, name) ; employee::employee_id = employee_id; if (salary
< 50000.0)
employee::salary = salary; else // Недопустимый оклад employee::salary = 0.0;
}

void main(void)
{
employee worker("Happy Jamsa", 101, 10101.0); worker.show_employee();
}

```

Обратите внимание, что за объявлением объекта worker следуют круглые скобки и начальные значения, как и при вызове функции. Когда вы используете конструктор с параметрами, передавайте ему параметры при объявлении объекта:

```
employee worker("Happy Jamsa", 101, 10101.0);
```

Конструкторы и параметры по умолчанию

C++ позволяет указывать значения по умолчанию для параметров функции. Если пользователь не указывает каких-либо параметров, функция будет использовать значения по умолчанию. Конструктор не является исключением. Например, следующий конструктор employee использует по умолчанию значение оклада равным 10000.0, если программа не указывает оклад при создании объекта. Однако программа должна указать имя служащего и его номер:

```

employee::employee(char *name, long employee_id, float salary = 10000.00)
{
strcpy(employee::name, name); employee::employee_id = employee_id; if (salary
< 50000.0)
employee::salary = salary; else // Недопустимый оклад
employee::salary = 0.0;
}

```

Перегрузка конструкторов

C++ позволяет вашим программам перегружать определения функций, указывая альтернативные функции для других типов параметров. C++ позволяет вам также перегружать конструкторы. Следующая программа перегружает конструктор employee. Первый конструктор требует, чтобы программа указывала имя служащего, номер служащего и оклад. Второй конструктор запрашивает пользователя ввести требуемый оклад, если программа не указывает его:

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

class employee
{
public:
employee(char *, long, float); employee(char *, long);
void show_employee(void); int change_salary(float) ; long get_id(void);
private:
char name [64]; long employee_id; float salary;
};

employee::employee(char *name, long employee_id, float salary)
{
strcpy(employee::name, name); employee::employee_id = employee_id;
if (salary < 50000.0) employee::salary = salary; else // Недопустимый оклад
employee::salary = 0.0;
}

employee::employee(char *name, long employee_id)
{
strcpy(employee::name, name); employee::employee_id = employee_id; do
{
cout << "Введите оклад для " << name << " меньше $50000: "; cin >>
employee::salary;
}
while (salary >= 50000.0);
}

void employee::show_employee(void)
```

```

{
cout << "Служащий: " << name << endl;
cout << "Номер служащего: " << employee_id << endl; cout << "Оклад: " <<
salary << endl;
}

void main(void)
{
employee worker("Happy Jamsa", 101, 10101.0); employee manager("Jane Doe",
102); worker.show_employee(); manager.show_employee();
}

```

Если вы откомпилируете и запустите эту программу, на вашем экране появится запрос ввести оклад для Jane Doe. Когда вы введете оклад, программа отобразит информацию об обоих служащих.

Представление о деструкторе

Деструктор автоматически запускается каждый раз, когда программа уничтожает объект.

При завершении программ C++ уничтожает объекты. Если определен деструктор, C++ будет автоматически вызывать деструктор для каждого объекта, когда программа завершается (т.е. когда объекты уничтожаются). Подобно конструктору, деструктор имеет такое же имя, как и класс объекта. Однако в случае деструктора его имя предваряется символом тильды (~), как показано ниже:

```

~class_name (void) //>указывает деструктор
{
// Операторы деструктора
}

```

В отличие от конструктора деструктору параметры не передаются. Деструктор для класса employee:

```

void employee::~-employee(void)
{
cout << "Уничтожение объекта для " << name << endl;
}

```

В данном случае деструктор просто выводит на ваш экран сообщение о том, что C++ уничтожает объект. Программа автоматически вызывает деструктор, без какого-либо явного вызова функции деструктора.

Что нужно знать?

Конструкторы и деструкторы представляют собой специальные функции класса, которые программа автоматически вызывает при создании или уничтожении объекта.

Большинство программ используют конструктор для инициализации элементов данных класса.

Основные концепции:

- Конструктор представляет собой специальную функцию, которую ваша программа автоматически вызывает каждый раз при создании объекта. Конструктор имеет такое же имя, как и класс объекта.
- Конструктор не имеет возвращаемого значения, но вы не указываете ему тип `void`. Вместо этого вы просто не указываете возвращаемое значение вообще.
- Когда ваша программа создает объект, она может передать параметры конструктору во время объявления объекта.
- C++ позволяет вам перегружать конструкторы и разрешает использовать значения по умолчанию для параметров.
- Деструктор представляет собой специальную функцию, которую ваша программа вызывает автоматически каждый раз при уничтожении объекта. Деструктор имеет такое же имя, как и класс объекта, но его имя предваряется символом тильды (~)

2. Практическое задание (100%)

Пользовательский класс **MyString** должен содержать необходимые элементы- данные, которые создаются в динамической области памяти.

- Конструкторы (без параметров, с параметрами, копирования) для создания строк: **MyString (...);**
- Деструктор: **~MyString();**
- Метод ввода исходной строки: **set();**
- Метод изменения исходной строки согласно варианту (исходная и измененная строка должны сохраняться в файле): **update() ;**
- Метод вывода на экран: **print(...);**
- Каждый вызов методов (в том числе конструкторов и деструктора) сопровождается выдачей соответствующего сообщения;

Код методов – вне пространства определения класса.

Написать демонстрационную программу, в которой показать использование объектов созданного класса

Варианты:

1. Длина L нечетная, то удаляется символ, стоящий посередине строки;
2. Длина L четная, то удаляются 2 первых и 2 последних символа;
3. Длина L кратна 2-м, то удаляются все цифры, которые делятся на 2;
4. Длина L кратна 3-м, то удаляются все цифры, делящиеся на 3;
5. Длина $L > 10$, то удаляются все цифры;
6. Длина $L > 15$, то удаляются все `a..z`;

7. Длина L=10, то удаляются все A..Z;
8. Длина L кратна 4-м, то первая часть строки меняется местами со второй;
9. Длина L кратна 5-и, то подсчитывается количество скобок всех видов;
10. Длина L >5-и, то выделяется подстрока до первого пробела;
11. Длина L >6-и, то выделяется подстрока { } скобках;
12. Длина L >10-и, то удаляется подстрока в [] скобках;
13. Длина L >12-и, то удаляется подстрока до первой (скобки;
14. Длина L кратна 4-м, то выделяется подстрока после последнего пробела;
15. Длина L >5, то удаляются все точки.
16. Длина L четная, то выделяется подстрока до первого пробела
17. Длина L четная, то удаляется подстрока до первого пробела
18. Длина L четная, то выделяется подстрока со второго пробела
19. Длина L нечетная, то выделяется подстрока после первого пробела
20. Длина L нечетная, то удаляется подстрока со второго пробела
21. Длина L кратна 3, то удаляется каждый 3-й символ
22. Длина L четная, то удаляется каждый 2-й символ
23. Длина L нечетная, то
24. Длина L четная, то выделяется подстрока до последнего пробела
25. Длина L нечетная, то выделяется подстрока от последней цифры

3. Контрольные вопросы

1. Что такое конструктор
2. Как задается имя конструктора?
3. Может ли класс иметь более одного конструктора?
4. Что такое деструктор?
5. В чем состоит преимущество определения конструктора со списком инициализации элементов?
6. Какие виды конструкторов создаются по умолчанию?
7. В каком порядке инициализируются поля в классе?
8. Какая ошибка в следующей реализации конструктора?

...

```
typedef unsigned int dlna;
const dlna n=30;
```

```
class Mouse{
    dlna rost;
protected :
    char * name;
public :
    Mouse (const char * a="None"):rost(1)
        {name=new char[n]; strcpy(name,a);}
    ~Mouse () {rost=0;}
    Mouse (Mouse & A):rost(A.rost), name(A.name) {}
```

...

```
};
```

```

...
typedef unsigned int dlina;
const dlina n=30;

class Mouse{
    dlina rost;
protected :
    char * name;
public :
    Mouse (const char * a="None"):rost(1)
        {name=new char[n]; strcpy(name,a);}
    ~Mouse () {rost=0;}
    Mouse (Mouse & A):rost(A.rost), name(A.name) {}
...
};

```

9. Может ли деструктор иметь аргументы?

10.Какая ошибка в следующей реализации деструктора?

Практическое занятие №10 Перегрузка операций

Цели:

Изучить механизм перегрузки операций.

1. Краткие теоретические сведения

Перегрузка операций – это одна из самых мощных и полезных возможностей ООП. Её применение позволяет существенно упростить написание программ, сделать их понятнее за счет применения интуитивно понятных обозначений, таких как $+$, $*$, $[]$ и т.д.

Перегрузка операций, по сути, дает возможность переопределить стандартный язык программирования. Используя классы для создания новых типов переменных и перегрузку для определения операций над этими типами, можно, по сути, создать новый, собственный язык программирования, заточенный под конкретную задачу, и, таким образом, перейти от общего к предметно-ориентированному программированию.

Благодаря перегрузке операций любой класс можно сделать таким, что он будет вести себя подобно встроенному типу данных. В классе можно перегрузить любые из следующих операций:

Нельзя перегружать операции

Функции-операции, реализующие перегрузку операций, имеют вид
operator операция ([операнды]) ;

Если функция является элементом класса, то первый операнд соответствующей операции будет самым объектом, для которого вызвана функция-операция. В случае одноместной операции список параметров будет пуст. Для двухместных операций функция будет иметь один параметр, соответствующий второму операнду. Если функция- операция не является элементом класса, она будет иметь один параметр в случае одноместной операции и два — в случае двухместной.

Для перегрузки операций существуют такие правила:

- Приоритет и правила ассоциации для перегруженных операций остаются теми же самыми, что и для операций над встроенными типами.
- Нельзя изменить поведение операции по отношению к встроенному типу.
- Функция-операция должна быть либо элементом класса, либо иметь один или несколько параметров типа класса.
- Функция-операция не может иметь аргументов по умолчанию.

В качестве примера приведем перегрузку операции сложения и умножения (как скалярного произведения) для класса, описывающего вектор в пространстве с координатами x , y , z .

```

class my_vector { // Объявляем класс - вектор длины 3
public: // с элементами-данными x, y и z - координатами вектора long double x;
long double y; long double z;
//Переопределяем оператор умножения long double operator*(my_vector v2);
//Переопределяем оператор сложения my_vector operator+(my_vector v2);
};

// переопределяем оператор умножения * как
// скалярное произведение векторов
long double my_vector::operator*(my_vector v2)
{ long double help;
help=x*v2.x + y*v2.y + z*v2.z; return help;
};

//переопределяем оператор сложения + как сумму векторов my_vector
my_vector::operator+(my_vector v2)
{ my_vector help; help.x = x + v2.x; help.y = y + v2.y; help.z = z + v2.z; return
help;
};

```

После этого в программе для объектов класса `my_vector` будут доступны операции сложения и умножения:

```

my_vector v1, v2, v3; long double a;
v1 = v2 + v3; a = v1 * v2;

```

При перегрузке операций необходимо помнить следующее:

- С++ не умеет образовывать из простых операций более сложные. Например, в классе со сложением строк мы определили присваивание и сложение; но это не значит, что тем самым будет автоматически определено присвоение суммы (`+=`). Такую операцию нужно реализовывать отдельно.
- Невозможно изменить синтаксис перегруженных операций. Одноместные операции должны быть одноместными, а двухместные — двухместными.
- Нельзя изобретать новые обозначения операций. Возможные операции ограничиваются тем списком, что приведен в начале этого раздела.
- Желательно сохранять смысл перегружаемой операции. Например, конкатенация — естественная семантика сложения для строк.

Операции не обязательно объявлять членами класса. Скажем, предыдущий пример с перегрузкой операций сложения и умножения для класса `my_vector` можно реализовать и так:

```

class my_vector { // Объявляем класс - вектор длины 3

```

```

public: // с элементами-данными x, y и z - координатами вектора long double
x;
long double y; long double z;
};
//переопределяем оператор умножения * как
//скалярное произведение векторов
long double operator*( my_vector v1, my_vector v2)
{ long double help;
help=v1.x*v2.x + v1.y*v2.y +v1. z*v2.z; return help;
};

//переопределяем оператор сложения + как сумму векторов my_vector
operator+( my_vector v1, my_vector v2)
{ my_vector help; help.x = v1.x + v2.x; help.y = v1.y + v2.y; help.z = v1.z + v2.z;
return help;
};

```

Несколько слов о перегрузке унарных операций.

Имеется особенность синтаксиса при перегрузке операций с префиксной и постфиксной формой ++ (инкремент) и -- (декремент).

В случае перегрузки префиксной формы используют следующий синтаксис переопределения:

```
void operator++( );
```

Если требуется переопределить постфиксную форму, то прототип перегружаемой операции будет таким:

```
void operator++( int );
```

Различие состоит лишь в том, что у постфиксной формы в скобках стоит *int*. Здесь *int* не играет роль аргумента и не означает целое число. Это просто сигнал для компилятора, чтобы использовалась постфиксная версия оператора.

Имеется также особенность перегрузки операции [] – индексация массива. Так как данная операция часто используется не только для доступа на чтение, но и для доступа на запись (то есть что-то типа $x[i] = y$), то целесообразно сделать так, чтобы перегруженная функция `operator [ ]` возвращала свое значение по ссылке. Это будет выглядеть примерно следующим образом:

```
int& operator[](int i);
```

Практическое задание (100%)

Все классы следует наделить конструкторами, деструктором. Необходимо явно реализовать конструктор копирования и перегрузить оператор присваивания. Необходимо подготовить демонстрацию по работе перегруженных для класса операторов.

Варианты:

1. Создать класс ПРЯМОУГОЛЬНИК со сторонами параллельными осям координат (прямоугольная система координат ОХУ). Реализовать метод вывода на экран информации о прямоугольнике. Перегрузить бинарный оператор несимметрической разности двух прямоугольников (-); унарный оператор (-): симметричное отображение прямоугольника относительно оси координат ОХ и ОУ. Следует учесть, что результатом выполнения оператора может быть пустой прямоугольник.

2. Создать класс, описывающий тип ВРЕМЯ. Класс должен включать в себя атрибуты, описывающие часы, минуты, секунды и миллисекунды и иметь метод для вывода времени на экран. Для данного класса перегрузить следующие бинарные операторы: суммы(+), разности (-).

3. Создать класс МНОГОЧЛЕН степени n от одной переменной x , задаваемый массивом своих коэффициентов (массив должен храниться в динамической памяти и задаваться внутри конструктора, используя датчик случайных чисел). Класс должен включать конструктор, которому в качестве параметра передается степень многочлена; деструктор; конструктор копирования, метод, который печатает уравнение на экран. Для данного класса перегрузить следующие бинарные операторы: суммы двух многочленов (+), разности двух многочленов (-), операцию присваивания (=).

4. Для пространства $\mathbf{R}^3$ (выбрана правая система декартовых прямоугольных координат $\{0, \mathbf{i}, \mathbf{j}, \mathbf{k}\}$) создать класс ВЕКТОР, предусмотрев для него несколько видов конструкторов, метод для вывода на экран его координат. Для данного класса перегрузить следующие бинарные операторы: суммы(+), разности (-), “векторное произведение” (*).

5. Создать класс ОТРЕЗОК ЧИСЛОВОЙ ПРЯМОЙ (сегмент). Предусмотреть несколько конструкторов для создания объектов класса, реализовать метод вывода отрезка на экран. Перегрузить следующие бинарные операторы: дополнение одного сегмента другим (+), пересечение двух сегментов (*), несимметрическая разность сегментов (-) $\{[1,5]-[3,6]=[1,3]\}$. Следует учесть, что результатом выполнения оператора может быть пустой сегмент, в этом случае следует вернуть отрезок нулевой длины $[0,0]$.

6. Создать класс КВАДРАТНАЯ МАТРИЦА 3X3. Элементы матрицы следует хранить в динамической памяти и задавать внутри конструктора, используя датчик случайных чисел. Класс должен включать конструктор; деструктор; конструктор копирования; метод для вывода матрицы на экран. Перегрузить следующие бинарные операторы: сумма матриц (+), произведение матриц (*).

7. Создать класс ПРЯМОУГОЛЬНИК со сторонами параллельными осям координат (прямоугольная система координат ОХУ). Реализовать метод вывода на

экран информации о прямоугольнике. Перегрузить следующие бинарные операторы: пересечение двух прямоугольников (*), объединение двух прямоугольников (+). Следует учесть, что результатом выполнения оператора может быть пустой прямоугольник.

8. Создать класс ОКРУЖНОСТЬ в прямоугольной системе координат ОХУ. Реализовать метод вывода на экран информации об окружности. Перегрузить бинарный оператор несимметрической разности двух окружностей (-); унарный оператор (-):

9. симметричное отображение окружности относительно оси координат ОХ и ОУ. Следует учесть, что результатом выполнения оператора может быть окружность нулевого радиуса.

10.*Создать класс СТРОКА. Объект класса должен характеризоваться следующими свойствами: длина строки, динамически выделяемый массив символов, заканчивающийся символом '\n', для хранения элементов строки. Предусмотреть несколько конструкторов для создания объектов класса, в том числе и конструктор с параметром, который задает длину будущей строки. Для данного класса реализовать метод вывода строки на экран, перегрузить следующие бинарные операторы: сцепление строк (+), удаление подстроки из строки (-) {“qwerty”-“we”=qrt”, {“qwerty”-“tu”=“qwerty”, “qwerty”-“qwerty”=”” }. Учесть тот случай, когда результатом операции может быть пустая строка (строка нулевой длины).

11.Создать класс СЕКТОР ЕДИНИЧНОГО КРУГА в прямоугольной системе координат ОХУ с центром в точке $O(0,0)$, который определяется двумя различными точкам на окружности единичного радиуса. Реализовать несколько конструкторов, метод вывода на экран информации о секторе. Перегрузить бинарные операторы: пересечение двух секторов (*), объединение двух секторов (+), несимметрическая разность двух секторов (-). Следует учесть все специальные случаи.

12.*Создать класс МНОГОЧЛЕН степени n от одной переменной x , задаваемый массивом своих коэффициентов (массив должен храниться в динамической памяти и задаваться внутри конструктора, используя датчик случайных чисел). Класс должен включать конструктор, которому в качестве параметра передается степень многочлена; деструктор; конструктор копирования, метод, который печатает уравнение на экран. Для данного класса перегрузить бинарный оператор произведения двух многочленов (*), унарный оператор (-).

13.*Создать класс ОКРУЖНОСТЬ в прямоугольной системе координат ОХУ. Реализовать метод вывода на экран информации об окружности. Перегрузить следующие бинарные операторы: пересечение двух окружностей (*), объединение двух окружностей (+). Следует учесть, что результатом выполнения оператора может быть окружность нулевого радиуса.

14. Создать класс КОЛЬЦО (геометрическая фигура на плоскости) в прямоугольной системе координат OXY с центром в точке O(0,0), которая определяется двумя радиусами (R1, R2, R2>R1>0). Реализовать метод вывода на экран информации о кольце. Перегрузить бинарные операторы: пересечение двух колец (*), объединение двух колец (+), несимметрическая разность двух колец (-). Следует учесть, что результатом выполнения оператора может быть получено кольцо с равными радиусами (R1=R2), в этом случае следует сообщить об ошибке.

15. Создать класс ПРЯМОУГОЛЬНИК СО СТОРОНОЙ НА ОСИ OX, для прямоугольной системы координат OXY (одна сторона которого расположена на оси OX). Реализовать несколько конструкторов, метод вывода на экран информации о прямоугольнике. Перегрузить бинарные операторы: пересечение двух прямоугольников (*), объединение двух прямоугольников (+), несимметрическая разность двух прямоугольников (-). Следует учесть все специальные случаи.

16. Создать класс КВАДРАТНАЯ МАТРИЦА 3X3. Элементы матрицы следует хранить в динамической памяти и задавать внутри конструктора, используя датчик случайных чисел. Класс должен включать конструктор; деструктор; конструктор копирования; метод для вывода матрицы на экран. Перегрузить бинарный оператор разности двух матриц, операцию присваивания, унарный оператор (-), который возвращает транспонированную матрицу.

2. Список рекомендуемой литературы

1. Объектно-Ориентированное программирование: учебник / Г. С. Иванова, Т. Н. Ничушкина ; под общ. ред. Г. С. Ивановой. — М. : Изд-во МГТУ им. Н. Э. Баумана, 2014.
— 455.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Для чего в C++ применяется перегрузка операций
2. Истинно ли следующее утверждение: операция >= может быть перегружена?
3. Сколько аргументов требуется для определения перегруженной унарной операции?
4. Сколько аргументов требуется для определения перегруженной бинарной операции?
5. Чем отличается действие операции ++ в префиксной форме от её действия в постфиксной форме?

6. Истинно ли следующее утверждение: перегруженная операция всегда требует на один аргумент меньше, чем количество операндов?

7. Когда перегружается операция арифметического присваивания, то результат

- a. Передается объекту справа от операции
- b. Передается объекту слева от операции
- c. Передается объекту, вызвавшему операцию
- d. Должен быть возвращен

8. Истинно ли следующее утверждение: компилятор не выдаст сообщение об ошибке, если вы перегрузите операцию * для выполнения деления?

9. Существуют ли операции, которые нельзя перегружать?

Практическое занятие №11 Наследование

Цели:

Изучить возможности наследования классов на языке C++.

1. Краткие теоретические сведения

Каждый объект одного и того же класса имеет собственную копию данных класса. Но существуют задачи, когда данные должны быть компонентами класса, и иметь их нужно только в единственном числе. Такие компоненты должны быть определены в классе как *статические (static)*. Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре.

Наследование является наиболее значимой возможностью ООП. Наследованием называется процесс создания новых классов, называемых наследниками, дочерними или производными классами из уже существующих – базовых или родительских классов. Производный класс получает все возможности базового класса, но имеет также и свои собственные.

Выигрыш от применения наследования состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс, его можно больше не модифицировать, а механизм наследования позволит приспособить его для новых задач путем порождения от него новых производных классов. Использование уже написанного и отлаженного кода увеличивает надёжность программ.

Наследование также является упрощением распространения библиотек классов. Программист может использовать классы, созданные кем-то другим, без модификации кода, просто создавая производные классы и добавляя к ним новые возможности.



Рисунок 1 - Сущность наследования

БАЗОВЫЙ И ПРОИЗВОДНЫЙ КЛАССЫ

Класс в C++ может *наследовать* элементы-данные и элементы-функции от одного или нескольких *базовых классов*. Сам класс называется в этом случае *производным* по отношению к базовым классам или *классом-потомком*. В свою очередь, производный класс может являться базовым по отношению к другим классам.

Принцип наследования, или *порождения* новых классов, позволяет абстрагировать (инкапсулировать) некоторые общие свойства и поведение в одном базовом классе, которые будут наследоваться всеми его потомками.

Наследование позволяет также модифицировать поведение базового класса. Производный класс может переопределять некоторые функции-элементы базового класса, оставляя основные свойства класса в неприкосновенности.

Синтаксис производного класса следующий:

```
class имя класса: ключ доступа имя_базового класса [, ...]
{
    тело_объявления_класса
};
```

Ключ_доступа — это одно из ключевых слов *private*, *protected* или *public*. Для производного класса доступны разделы *protected* и *public* базового класса; раздел *private* строго недоступен вне области действия базового класса, раздел *protected* недоступен для функций, не принадлежащих к базовому либо дочернему классу.

Таблица 1 - Наследование и доступ

Спецификатор доступа	Доступ из самого класса	Доступ из производных классов	Доступ из внешних классов и функций
public	Есть	Есть	Есть
protected	Есть	Есть	Нет
private	Есть	Нет	Нет

Для доступа к элементам базового класса через производный можно сформулировать такое правило: права доступа, определяемые для них базовым классом, остаются неизменными, если они такие же или строже, чем специфицировано *ключом доступа*. В противном случае права доступа определяются ключом в определении производного класса. То есть ключ доступа, указанный при наследовании «сдвигает» права доступа в сторону «более строгих».

Например, при наследовании с ключом *public* права доступа к элементам базового класса остаются неизменными; при закрытом наследовании (ключ *private*) все элементы базового класса будут недоступны за пределами производного класса. При наследовании с ключом *protected*, элементы, которые были *public* становятся *protected*, остальные остаются без изменения.

При закрытом наследовании можно сделать некоторые открытые функции базового класса открытыми в производном, если переобъявить их имена в производном классе. Пример:

```
class First { public:
void FFunc(void) ;
//...
};
```

```
class Second: private First { public:
First::FFunc; // First::FFunc() открыта в классе Second.
//.. .
};
```

Как правило, в практических задачах применяется почти исключительно открытое наследование.

Язык C++ допускает простое и сложное наследование. При простом наследовании у класса может быть только один родитель, при сложном – два и более.

ПРОСТОЕ НАСЛЕДОВАНИЕ

Пример простого наследования:

```
class Time // Базовый класс - время.
{
int hr, min; public:
Time(int h=12, int m=0): hr(h), min(m) {} // Конструктор
void SetTime(int h, int m) {hr = h; min = m; } //Метод установки времени void
Show( ) { printf("%02d:%02d", hr, min) }; // Метод, чтобы показать время
// на консоли
};
```

```
class Alarm: public Time// Класс сообщений таймера – дочерний от Time.
{
char *msg; // Указатель на строку-сообщение public:
Alarm(char*); // Конструктор нового класса
~Alarm() { delete [] msg; } // Деструктор
void SetMsg(char*); // Метод установки сообщений void Show(); //
Переопределяет метод Show ( ).
};
```

```
Alarm::Alarm(char *str) // Конструктор класса-потомка
{
msg = new char[strlen (str) + 1]; strcpy(msg, str);
}
```

```
void Alarm::SetMsg(char *str) // Описание функции класса-потомка SetMsg(
)
```

```
{
delete [] msg;
msg = new char[strlen (str) + 1]; strcpy(msg, str);
}
```

```
void Alarm::Show( ) // Переопределение функции Show( )
{
Time::Show(); // Вызов базовой Show() printf(": %s\n", msg);
}
```

Теперь можно создавать переменные производного класса, например, таким образом:

```
int main ( ) {
Alarm a = "Test Alarm!!!"; // Время по умолчанию 12:00. a.Show ( );
a.SetTime(7, 40); // Функция базового класса a.Show ( );
a.SetMsg("It's time! " ); // Функция производного класса. a.Show( );
return 0;
}
```

Несколько слов о наследовании конструкторов. При создании объекта производного класса компилятор перед вызовом конструктора производного класса автоматически вызывает сначала конструктор родительского класса, но только тот, который может быть вызван без параметров (то есть либо конструктор по умолчанию, либо конструктор без параметров, либо конструктор с параметрами по умолчанию – как в примере выше у класса *Time*). Но конструктор родительского класса с параметрами не наследуется производным в том смысле, что он не будет вызываться автоматически при создании объекта производного класса с такими параметрами.

Вместе с тем, нужный конструктор базового класса можно вызвать явно через список инициализации. Например, дочерний класс из предыдущего примера можно было задать и так:

```
class Alarm: public Time // Класс сообщений таймера.
{
char *msg; public:
Alarm(char*);
Alarm(char*, int, int); // Новый конструктор.
~Alarm( ) { delete[] msg; } void SetMsg(char*);
void Show(); // Переопределяет Time:: Show ( ).
};
```

```
Alarm::Alarm(char *str, int h, int m): Time(h, m) - // явный вызов конструктора
//родительского класса
{
msg = new char[strlen(str) + 1]; strcpy(msg, str);
```

```
}
```

Деструкторы базовых классов всегда вызываются компилятором автоматически.

СЛОЖНОЕ НАСЛЕДОВАНИЕ

Язык C++ допускает не только простое, но и *сложное наследование*, т. е. наследование от двух и более непосредственных базовых классов. Это позволяет создавать классы, комбинирующие в себе свойства нескольких независимых классов- предков. Синтаксис производного класса Alarm при сложном наследовании от классов Time и Message:

```
class Alarm: public Time, public Message
{
    .....
};
```

В определенных ситуациях могут появиться проблемы, связанные со сложным наследованием. Например, в обоих базовых классах могут существовать методы с одинаковыми именами, а в производном классе такой метод не переопределяется. Как в этом случае объект производного класса определит, какой из методов базовых классов выбрать? Одного имени метода недостаточно, поскольку компилятор не сможет вычислить, какой из двух методов имеется в виду. Проблема решается путем использования оператора разрешения области

```
class A
{
    public:
        void show ( ) { cout << "Класс A\n"; }
};
class B
{
    public:
        void show ( ) { cout << "Класс B\n"; }
};
class C : public A, public B
{
};
////////////////////////////////////
int main ( )
{
    C objC;           // объект класса C
    // objC.show ( ); // так делать нельзя - программа не скомпилируется
    objC.A::show ( ); // так можно
    objC.A::show ( ); // так можно
    return 0;
}
```

действия, определяющего класс, в котором находится метод. Пример:

Другой вид неопределенности появляется, если создается производный класс от двух базовых классов, которые, в свою очередь, являются производными одного класса. Это создает дерево наследования в форме ромба. Компилятор не сможет решить, какой из методов использовать, и сообщит об ошибке.

Существует множество вариаций этой проблемы. Поэтому технология программирования предписывает избегать множественного наследования.

2. Практическое задание

Реализовать иерархию классов для простого (задание 1) и сложного (задание 2) наследования:

2.1. Индивидуальное задание. Простое наследование (50%)

1. Создать класс квадрат, члены класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – диагональ, периметр, площадь. Создать производный класс – правильная квадратная призма с высотой H , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с этими классами: дано N квадратов и M призм, найти квадрат с максимальной площадью и призму с максимальной диагональю.

2. Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равносторонний треугольник, перегрузить в классе проверку, является ли треугольник равносторонним и метод вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано K треугольников и L равносторонних треугольников, найти среднюю площадь для K треугольников и наибольший равносторонний треугольник.

3. Создать класс окружность, член класса – радиус R . Предусмотреть в классе методы вычисления и вывода сведений о фигуре – площади, длины окружности. Создать производный класс – круглый прямой цилиндр с высотой h , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N окружностей и M цилиндров, найти окружность максимальной площади и средний объем цилиндров.

4. Создать класс квадрат, члены класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – диагоналей, периметр, площадь. Создать производный класс – правильная пирамида с апофемой h , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N квадратов и M пирамид, найти квадрат с минимальной площадью и количество пирамид с высотой более числа a (a вводить).

5. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы проверки существования четырехугольника вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – параллелограмм, предусмотреть в классе проверку, является ли фигура параллелограммом. Написать программу, демонстрирующую работу с классом: дано N четырехугольников и M параллелограммов, найти среднюю площадь N четырехугольников и параллелограммы наименьшей и наибольшей площади.

6. Создать класс треугольник, члены класса – координаты 3-х точек. Предусмотреть в классе методы проверки существования треугольника,

вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равносторонний треугольник, предусмотреть в классе проверку, является ли треугольник равносторонним. Написать программу, демонстрирующую работу с классом: дано N треугольников и M равносторонних треугольников, вывести номера одинаковых треугольников и равносторонний треугольник с наименьшей медианой.

7. Создать класс прямоугольник, члены класса – длины сторон a и b . Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – параллелепипед с высотой c , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N прямоугольников и M параллелепипедов, найти количество прямоугольников, у которых площадь больше средней площади прямоугольников и количество кубов (все ребра равны).

8. Создать класс окружность, член класса – радиус R . Предусмотреть в классе методы вычисления и вывода сведений о фигуре – площади, длины окружности. Создать производный класс – конус с высотой h , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N окружностей и M конусов, найти количество окружностей, у которых площадь меньше средней площади всех окружностей, и наибольший по объему конус.

9. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – равнобокая трапеция, предусмотреть в классе проверку, является ли фигура равнобокой трапецией. Написать программу, демонстрирующую работу с классом: дано N четырехугольников и M трапеций, найти максимальную площадь четырехугольников и количество четырехугольников, имеющих максимальную площадь, и трапецию с наименьшей диагональю.

10. Создать класс равносторонний треугольник, член класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – периметр, площадь. Создать производный класс – правильная треугольная призма с высотой H , добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N треугольников и M призм. Найти количество треугольников, у которых площадь меньше средней площади треугольников, и призму с наибольшим объемом.

11. Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – прямоугольный треугольник, предусмотреть в классе проверку, является ли треугольник прямоугольным. Написать программу, демонстрирующую работу с классом: дано N треугольников и M прямоугольных треугольников, найти треугольник с максимальной площадью и прямоугольный треугольник с наименьшей гипотенузой.

12. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – квадрат, предусмотреть в классе проверку, является ли фигура квадратом. Написать программу, демонстрирующую работу с классом: дано N четырехугольников и M квадратов, найти четырехугольники с минимальной и максимальной площадью и номера одинаковых квадратов.

13. Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равнобедренный треугольник, предусмотреть в классе проверку, является ли треугольник равнобедренным. Написать программу, демонстрирующую работу с классом: дано N треугольников и M равнобедренных треугольников, найти среднюю площадь для N треугольников и равнобедренный треугольник с наименьшей площадью.

14. Создать класс квадрат, член класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – периметр, площадь, диагональ. Создать производный класс – куб, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N1 квадратов и N2 кубов. Найти среднюю площадь квадратов и количество кубов с наибольшей площадью. 15. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – ромб, предусмотреть в классе проверку, является ли фигура ромбом. Написать программу, демонстрирующую работу с этими классами: дано N четырехугольников и M ромбов, найти четырехугольник с минимальным периметром и среднюю площадь ромбов.

2.2 Индивидуальное задание. Множественное наследование (50%)

1. Используя родительский класс «ТРАНСПОРТ» породить производный класс

«АВТОМОБИЛЬ». Используя классы, «ВОДИТЕЛЬ» и «АВТОМОБИЛЬ», описать класс

«ВОДИТЕЛЬ АВТОМОБИЛЯ». Расширить класс «ВОДИТЕЛЬ АВТОМОБИЛЯ» создав два производных класса «ВОДИТЕЛЬ СЛУЖЕБНОГО АВТОМОБИЛЯ» и «ВОДИТЕЛЬ ТАКСИ». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о водителе, автомобиле)

2. Используя родительский класс, «СЛУЖАЩИЙ» породить производный класс

«ДИРЕКТОР». Используя классы «ФИРМА» и «ДИРЕКТОР», описать класс «РУКОВОДИТЕЛЬ ФИРМЫ». Расширить класс «РУКОВОДИТЕЛЬ ФИРМЫ» создав два производных класса «РУКОВОДИТЕЛЬ ГОС.УЧ.» и «РУКОВОДИТЕЛЬ ООО». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о фирме и руководителе)

3. Используя родительский класс «НЕДВИЖИМОСТЬ» породить производный класс «ЗДАНИЕ». Используя классы «ЗДАНИЕ» и «ВЛАДЕЛЕЦ», описать класс

«ВЛАДЕЛЕЦ ЗДАНИЯ». Расширить класс «ВЛАДЕЛЕЦ ЗДАНИЯ» создав два производных класса «ВЛАДЕЛЕЦ ЧАСТНОГО ДОМА» и «ВЛАДЕЛЕЦ ОТЕЛЯ». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о владельце и здании)

4. Используя родительский класс «ТРАНСПОРТ» породить производный класс

«САМОЛЕТ». Используя классы, «ПИЛОТ» и «САМОЛЕТ», описать класс «ПИЛОТ САМОЛЕТА». Расширить класс «ПИЛОТ САМОЛЕТА» создав два производных класса

«ПИЛОТ ГРАЖДАНСКОГО САМОЛЕТА» и «ПИЛОТ ВОЕННОГО САМОЛЕТА».

Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о пилоте, самолете)

5. Используя родительский класс, «РАБОТНИК» породить производный класс

«РЕЖИССЕР». Используя классы «ФИЛЬМ» и «РЕЖИССЕР», описать класс

«РЕЖИССЕР ФИЛЬМА». Расширить класс «РЕЖИССЕР ФИЛЬМА» создав два производных класса «РЕЖИССЕР ХУДОЖЕСТВЕННОГО ФИЛЬМА» и «РЕЖИССЕР ДОКУМЕНТАЛЬНОГО ФИЛЬМА». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о фирме и руководителе)

3. Список рекомендуемой литературы

1. Павловская Т. А.С/С++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. В чем состоит назначение наследования?
2. Когда программисту-разработчику целесообразно прибегнуть к наследованию?
3. Напишите первую строку описания класса Child, который является public-производным от класса Parent.
4. Верно ли утверждение: создание производного класса требует коренных изменений в базовом классе?
5. Члены базового класса, для доступа к ним методов производного класса должны быть объявлены как public или ____.

6. Пусть базовый класс содержит метод `basefunc( )`, а производный класс не имеет такого метода. Может ли объект производного класса иметь доступ к методу `basefunc( )`.

7. Истинно ли следующее утверждение: если конструктор производного класса не определен, то объекты этого класса будут использовать конструкторы базового класса?

8. Как используется оператор разрешения области действия для разрешения неопределенностей?

9. Истинно ли следующее утверждение: иногда полезно создать класс, объектов которого никогда не будет создано?

10. Пусть класс `Derv` является дочерним от класса `Base`. Пусть определена переменная типа `Derv`, расположенная в функции `main( )`. Через эту переменную можно получить доступ к

- а) членам класса `Derv`, объявленным как `public`;
- б) членам класса `Derv`, объявленным как `protected`; в) членам класса `Derv`, объявленным как `private`; г) членам класса `Base`, объявленным как `public`;
- д) членам класса `Base`, объявленным как `protected`; е) членам класса `Base`, объявленным как `private`;

11. Пусть существует класс `Derv`, производный от класса `Base`. Напишите объявление конструктора производного класса, принимающего один аргумент и передающего его в конструктор базового класса.

12. Истинно ли следующее утверждение: невозможно сделать объект одного класса членом другого класса?

Практическое занятие №12 Программирование динамических и виртуальных методов

Цели:

Изучить возможности виртуальных функций при наследовании классов на языке C++, абстрактные классы.

1. Краткие теоретические сведения ВИРТУАЛЬНЫЕ ФУНКЦИИ

Виртуальные функции являются важной частью реализации механизма

полиморфизма (то есть выполнения разных действий с объектами разного типа). Использование виртуальных функций позволяет выполнить нужные действия в том случае, если доступ к объекту осуществляется не по значению, а по указателю.

Пусть имеется класс «Фигура», и имеются его классы-наследники «Круг», «Квадрат» и «Треугольник». Пусть в каждом из этих классов есть функция `draw()`, которая прорисовывает фигуры на экране.

Теперь пусть имеется указатель `X` на объект класса «Фигура». По правилам языка этот указатель может хранить адрес как самого класса «Фигура», так и адрес любого из его потомков, т.е. и «Круга» и «Квадрата» и «Треугольника». Теперь необходимо реализовать функцию `draw()` так, чтобы при обращении к ней через указатель `X->draw()` вызывалась бы именно нужная функция `draw()`, то есть именно того класса, адрес которого и хранится в указателе `X`. Понятно, что у круга, квадрата и треугольника это будут разные функции. Чтобы обеспечить такую возможность для функции `draw()` её необходимо объявить виртуальной в базовом классе.

Ниже приводятся примеры, демонстрирующие возможности виртуальной функции. Первый пример показывает, что бывает, когда базовый и производный классы содержат функции с одним и тем же именем, и к ним обращаются с помощью указателей, но без использования виртуальных функций.

```
class Base           //Базовый класс
{
public:
void show()          //Обычная функция
{ cout << "Base\n"; }
};

class Derv1 : public Base //Производный класс 1
{
public:
void show()
{ cout << "Derv1\n"; }
};
```

```

class Derv2 : public Base //Производный класс 2
{
public:
void show()
{ cout << "Derv2\n"; }

};

int main()
{
Derv1 dv1; //Объект производного класса 1 Derv2 dv2; //Объект производного
класса 2 Base* ptr; //Указатель на базовый класс

ptr = &dv1;      //Адрес dv1 занести в указатель ptr->show();
//Выполнить show()
ptr = &dv2;      //Адрес dv2 занести в указатель ptr->show();
//Выполнить show()
return 0;
}

```

Поскольку указатели на объекты дочерних классов совместимы по типу с указателями на объекты базового класса присвоение $ptr = \&dv1;$ и $ptr = \&dv2;$ вполне корректно. В результате выполнения приведенной программы будет дважды вызвана функция $show()$ именно базового класса, так как компилятор в данном случае не смотрит на содержимое указателя ptr , а выбирает тот метод, который удовлетворяет типу указателя (см. рисунок).

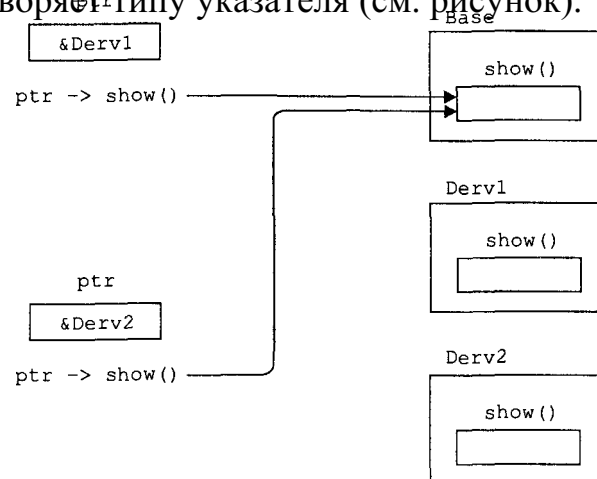


Рисунок 1 - Доступ через указатель без использования виртуальных функций
Теперь изменим обсуждаемую программу таким образом, чтобы функция $show()$ базового класса была бы объявлена как виртуальная функция (то есть добавим перед её объявлением в базовом классе ключевое слово `virtual`).

```

class Base      //Базовый класс
{
public:
virtual void show() //Виртуальная функция

```

```

{ cout << "Base\n"; }
};

class Derv1 : public Base //Производный класс 1
{
public:
void show()
{ cout << "Derv1\n"; }
};
class Derv2 : public Base //Производный класс 2
{
public:
void show()
{ cout << "Derv2\n"; }

};

int main()
{
Derv1 dv1; //Объект производного класса 1 Derv2 dv2; //Объект производного
класса 2 Base* ptr; //Указатель на базовый класс

ptr = &dv1; //Адрес dv1 занести в указатель ptr->show();
//Выполнить show()
ptr = &dv2; //Адрес dv2 занести в указатель ptr->show();
//Выполнить show()
return 0;
}

```

На выходе, в отличие от первого примера, будем иметь
Derv1 Derv2
то есть при вызове функции *show()* будут выполнены методы именно соответствующих производных классов, а не базового. То есть один и тот же вызов *ptr-*

>show(); запускает на выполнение разные функции в зависимости от содержимого указателя *ptr*. Компилятор выбирает функцию, удовлетворяющую тому, что занесено в указатель, а не типу указателя, как в первом примере (см. рисунок).

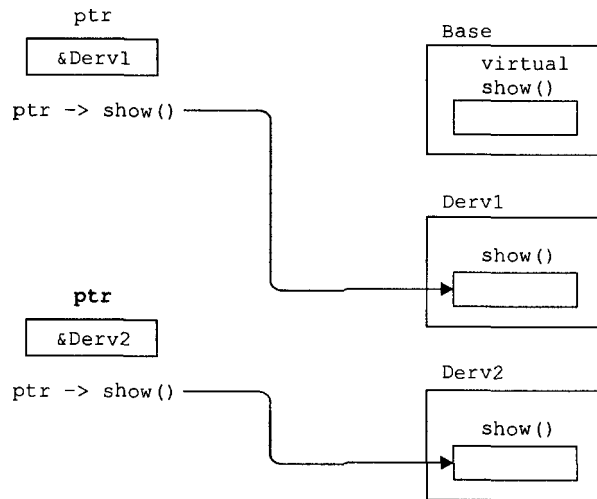


Рисунок 2 - Доступ через указатель к виртуальным функциям

АБСТРАКТНЫЕ КЛАССЫ И ЧИСТЫЕ ВИРТУАЛЬНЫЕ ФУНКЦИИ

Базовый класс, объекты которого никогда не будут созданы, называется абстрактным классом. Такой класс существует с единственной целью – быть родительским по отношению к производным классам, объекты которых уже будут реализованы. Если класс должен быть абстрактным, то от создания объектов этого класса его можно защитить программно. Для этого достаточно ввести в класс хотя бы одну чистую виртуальную функцию. Чистая виртуальная функция – это функция, после объявления которой добавлено выражение `= 0`. В следующем примере демонстрируется объявление и работа с чистыми виртуальными функциями.

```

class Base          //базовый класс
{
public:
virtual void show() = 0; //чистая виртуальная функция
};

class Derv1 : public Base  //порожденный класс 1
{
public:
void show()
{ cout << "Derv1\n"; }
};

class Derv2 : public Base  //порожденный класс 2
{
public:
void show()
{ cout << "Derv2\n"; }
};

int main()
  
```

```

{
// Base bad;           //невозможно создать объект
//из абстрактного класса Base* arr[2]; //массив указателей на
//базовый класс
Derv1 dv1;           //Объект производного класса 1 Derv2 dv2;           //Объект
производного класса 2

arr[0] = &dv1;           //Занести адрес dv1 в массив arr[1] = &dv2;           //Занести
адрес dv2 в массив

arr[0]->show();           //Выполнить функцию show() arr[1]->show();           //над
обоими объектами return 0;
}

```

Знак равенства при объявлении чистой виртуальной функции *virtual void show() = 0;* это не операция присваивания, это просто способ сообщить компилятору, что функция будет чистой виртуальной. Если теперь попытаться создать объект этого класса, компилятор выдаст ошибку.

На практике механизм виртуальных функций часто применяется для корректного вызова деструкторов. Технология программирования прямо предписывает, что деструкторы базовых классов должны быть виртуальными. Если деструктор базового класса не является виртуальным, то, будучи обычным методом, он будет вызываться независимо от того, данные какого типа в нем хранятся. Это приведет к тому, что может быть удалена только та часть объекта, которая относится к базовому классу. Конечно, если ни один из деструкторов (ни у базового, ни у производного класса) ничего особенного не делает, тогда их виртуальность перестает быть такой уж необходимой.

Практическое задание (100%)

1. Определить иерархию классов (в соответствии с вариантом).
2. Определить в классе статическую компоненту - указатель на начало связанного списка объектов и статическую функцию для просмотра списка.
3. Реализовать классы.
4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.
5. Сделать соответствующие методы не виртуальными и посмотреть, что будет.
6. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе.

Методические указания

1. Для определения иерархии классов связать отношением наследования классы заданного варианта. Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.
2. Определить в классах все необходимые конструкторы и деструктор.

3. Поля класса специфицировать как **protected**.
4. Пример определения статических компонентов:

```
static person* begin; // указатель на начало списка
```


static void print(); // просмотр списка
5. Статическое поле инициализировать вне определения класса, в глобальной области.
6. Для добавления объекта в список предусмотреть метод класса, т.е. объект сам добавляет себя в список. Например, `a.Add()` – объект **a** добавляет себя в список.
Включение объекта в список можно выполнять при создании объекта, т.е. поместить операторы включения в конструктор. В случае иерархии классов, включение объекта в список должен выполнять **только** конструктор базового класса. Вы должны продемонстрировать оба этих способа.
7. Список просматривать путем вызова виртуального метода **Show** каждого объекта.
8. Статический метод просмотра списка вызывать не через объект, а через класс.

Варианты заданий

- 1) студент, преподаватель, персона, завкафедрой;
- 2) служащий, персона, рабочий, инженер;
- 3) рабочий, кадры, инженер, администрация;
- 4) деталь, механизм, изделие, узел;
- 5) организация, страховая компания, судостроительная компания, завод;
- 6) журнал, книга, печатное издание, учебник;
- 7) тест, экзамен, выпускной экзамен, испытание;
- 8) место, область, город, мегаполис;
- 9) игрушка, продукт, товар, молочный продукт;
- 10) квитанция, накладная, документ, чек;
- 11) автомобиль, поезд, транспортное средство, экспресс;
- 12) двигатель, двигатель внутреннего сгорания, дизель, турбореактивный двигатель;
- 13) республика, монархия, королевство, государство;
- 14) млекопитающие, парнокопытные, птицы, животное;
- 15) корабль, пароход, парусник, корвет.

3. Список рекомендуемой литературы

1. Павловская Т. А. C/C++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Какие возможности перед программистом открывают виртуальные функции?

2. Истинно ли утверждение о том, что указатель на базовый класс может ссылаться на объекты порожденного класса?

3. Пусть указатель `p` ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется не виртуальный метод `ding()`. Тогда выражение `p->ding()` вызовет метод `ding()` из класса.

4. Напишите описание для виртуальной функции `dang()`, возвращающей результат `void` и имеющей аргумент типа `int`.

5. Пусть указатель `p` ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется виртуальный метод `ding()`. Тогда выражение `p->ding()` вызовет метод `ding()` из класса.

6. Напишите описание для чистой виртуальной функции `aragorn()`, не возвращающей значений и не имеющей аргументов.

7. Чистая виртуальная функция, это виртуальная функция, которая: а) делает свой класс абстрактным;

б) не возвращает результата;

в) используется в базовом классе; г) не имеет аргументов.

8. Напишите определение массива `paqr`, содержащего 10 указателей на объекты класса `dong`.

9. Абстрактный класс используется тогда, когда

а) не планируется создавать порожденные классы;

б) есть несколько связей между двумя порожденными классами в) необходимо запретить создавать объекты класса

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для обучающихся по организации и проведению самостоятельной работы по дисциплине

«Информационные технологии и программирование»

для студентов направления подготовки 10.03.01 «Информационная безопасность»

Направленность (профиль):

«Организация и технологии защиты информации (по отрасли или в сфере
профессиональной деятельности)»

ВВЕДЕНИЕ

Целью изучения дисциплины является формирование компетенций у студентов направления подготовки 10.03.01 «Информационная безопасность» и формирование теоретических знаний и практических навыков по построению информационных систем и программированию.

Задачей освоения дисциплины является обучение студентов принципам разработки и использования программного обеспечения для построения ИС и программирования в рамках прикладных задач

1. Общая характеристика самостоятельной работы студента при изучении дисциплины

Самостоятельная работа включает те разделы курса, которые не получили достаточного освещения на лекциях по причине ограниченности лекционного времени и большого объема изучаемого материала.

Методическое обеспечение самостоятельной работы по дисциплине состоит из:

- 1) Определения учебных вопросов, которые студенты должны изучить самостоятельно;
- 2) Подбора необходимой учебной литературы, обязательной для проработки и изучения;
- 3) Поиска дополнительной научной литературы, к которой студенты могут обращаться по желанию, если у них возникает интерес в данной теме;
- 4) Определения контрольных вопросов, позволяющих студентам самостоятельно проверить качество полученных знаний;
- 5) Организации консультаций преподавателя со студентами для разъяснения вопросов, вызвавших у обучающихся затруднения при самостоятельном освоении учебного материала.

Самостоятельная работа студента – это особым образом организованная деятельность, включающая в свою структуру такие компоненты, как:

- уяснение цели и поставленной учебной задачи;
- четкое и системное планирование самостоятельной работы;
- поиск необходимой учебной и научной информации;
- освоение собственной информации и ее логическая переработка;
- использование методов исследовательской, научно-исследовательской работы для решения поставленных задач;
- выработка собственной позиции по поводу полученной задачи;
- представление, обоснование и защита полученного решения;
- проведение самоанализа и самоконтроля

К основным видам самостоятельной работы студентов относится:

- работа с понятийным аппаратом;
- конспектирование первоисточников;
- проектирование фрагментов исследовательской деятельности;
- подготовка доклада по теме.

Контроль знаний студентов включает формы текущего и итогового контроля. Текущий контроль осуществляется в процессе изучения курса и включает в себя оценку работы студентов на практических занятиях (круглый стол, собеседование, групповые научные проекты, тестирование), а также подготовку домашнего задания.

1. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
2 семестр					
УК-1 ОПК-7	Подготовка к лабораторной работе	Изучение лекций	26.00	2.00	27.00
УК-1 ОПК-7	Самостоятельное изучение литературы	Лабораторные работы	26.00	2.00	27.00
Итого за 2 семестр			52.00	4.00	54.00
3 семестр					
УК-1 ОПК-7	Подготовка к лабораторной работе	Изучение лекций	25.00	2.00	14.00
УК-1 ОПК-7	Самостоятельное изучение литературы	Лабораторные работы	25.00	2.00	14.00
Итого за 3 семестр			50.00	4.00	28.00
Итого			102.00	8.00	72.00

2. Порядок выполнения самостоятельной работы

2.1 Методические рекомендации по изучению теоретического материала

Комплексное изучение предлагаемой студентам учебной дисциплины «Информационные технологии и программирование» предполагает овладение материалами лекций, учебников, программы, творческую работу студентов в ходе проведения практических занятий, а также систематическое выполнение заданий для самостоятельной работы.

В ходе лекций раскрываются основные вопросы в рамках рассматриваемых тем, делаются акценты на наиболее сложные и интересные положения изучаемого материала, которые должны быть приняты студентами во внимание. Материалы лекций являются основой для подготовки студентов к практическим занятиям.

Работа с конспектом лекций

Просмотрите конспект сразу после занятий. Отметьте материал конспекта лекций, который вызывает затруднения для понимания. Попробуйте найти ответы на затруднительные вопросы, используя предлагаемую литературу. Если самостоятельно не удалось разобраться в материале, сформулируйте вопросы и обратитесь на текущей консультации или на ближайшей лекции за помощью к преподавателю.

Каждую неделю отводите время для повторения пройденного материала, проверяя свои знания, умения и навыки по контрольным вопросам и тестам.

Выполнение практических занятий по дисциплине «Информационные технологии и программирование»

На первом занятии получите у преподавателя задания по курсу, планы подготовки к практическим занятиям. Обзаведитесь всем необходимым методическим обеспечением.

Перед практическими занятиями изучите теорию вопроса, предполагаемого к

исследованию, ознакомьтесь с опытом других исследователей в этом направлении.

Целью практических занятий является:

- закрепление методов приложения теории к решению практических задач анализа и синтеза психологического знания;
- проверка уровня понимания студентами вопросов, рассмотренных на лекциях и по учебной литературе, степени и качества усвоения материала;
- обучение навыкам освоения методики эксперимента и работы с нормативно- справочной литературой;
- восполнение пробелов в пройденной теоретической части курса и оказание помощи в его усвоении.

2.2 Методические рекомендации по написанию и оформлению доклада

1. Общие положения

1.1. Доклад, как вид самостоятельной работы в учебном процессе, способствует формированию навыков исследовательской работы, расширяет познавательные интересы, учит критически мыслить.

1.2. При написании доклада по заданной теме студент составляет план, подбирает основные источники.

1.3. В процессе работы с источниками систематизирует полученные сведения, делает выводы и обобщения.

1.4. К докладу по крупной теме могут привлекать несколько студентов, между которыми распределяются вопросы выступления.

2. Выбор темы доклада

2.1. Тематика доклада обычно определяется преподавателем, но в определении темы инициативу может проявить и студент.

2.2. Прежде чем выбрать тему доклада, автору необходимо выявить свой интерес, определить, над какой проблемой он хотел бы поработать, более глубоко ее изучить.

3. Этапы работы над докладом

3.1. Формулирование темы, причем она должна быть не только актуальной по своему значению, но и оригинальной, интересной по содержанию.

3.2. Подбор и изучение основных источников по теме (как правильно, при разработке доклада используется не менее 8-10 различных источников).

3.3. Составление списка использованных источников.

3.4. Обработка и систематизация информации

3.5. Разработка плана доклада.

3.6. Написание доклада.

3.7. Публичное выступление с результатами исследования.

4. Структура доклада:

- титульный лист
- оглавление (в нем последовательно излагаются названия пунктов доклада, указываются страницы, с которых начинается каждый пункт);
- введение (формулирует суть исследуемой проблемы, обосновывается

выбор темы, определяются ее значимость и актуальность, указываются цель и задачи доклада, дается характеристика используемой литературы);

- основная часть (каждый раздел ее, доказательно раскрывая отдельную проблему или одну из ее сторон, логически является продолжением предыдущего; в основной части могут быть представлены таблицы, графики, схемы);

- заключение (подводятся итоги или дается обобщенный вывод по теме доклада, предлагаются рекомендации);

- список использованных источников

5. Структура и содержание доклада

5.1. Введение - это вступительная часть научно-исследовательской работы. Автор должен приложить все усилия, чтобы в этом небольшом по объему разделе показать актуальность темы, раскрыть практическую значимость ее, определить цели и задачи эксперимента или его фрагмента.

5.2. Основная часть. В ней раскрывается содержание доклада. Как правило, основная часть состоит из теоретического и практического разделов. В теоретическом разделе раскрываются история и теория исследуемой проблемы, дается критический анализ литературы и показываются позиции автора. В практическом разделе излагаются методы, ход, и результаты самостоятельно проведенного эксперимента или фрагмента.

В основной части могут быть также представлены схемы, диаграммы, таблицы, рисунки и т.д.

5.3. В заключении содержатся итоги работы, выводы, к которым пришел автор, и рекомендации. Заключение должно быть кратким, обязательным и соответствовать поставленным задачам.

5.4. Список использованных источников представляет собой перечень использованных книг, статей, фамилии авторов приводятся в алфавитном порядке, при этом все источники даются под общей нумерацией литературы. В исходных данных источника указываются фамилия и инициалы автора, название работы, место и год издания.

5.5. Приложение к докладу оформляются на отдельных листах, причем каждое должно иметь свой тематический заголовок и номер, который пишется в правом верхнем углу, например: «Приложение 1»

6. Требования к оформлению доклада

6.1. Объем доклада может колебаться в пределах 5-15 печатных страниц; все приложения к работе не входят в ее объем.

6.2. Доклад должен быть выполнен грамотно, с соблюдением культуры изложения.

6.3. Обязательно должны иметься ссылки на используемую литературу.

6.4. Должна быть соблюдена последовательность написания библиографического аппарата.

7. Критерии оценки доклада

- актуальность темы исследования;
- соответствие содержания теме;
- глубина проработки материала; правильность и полнота использования источников;

- соответствие оформления доклада стандартам.

По усмотрению преподавателя доклады могут быть представлены на семинарах, научно- практических конференциях, а также использоваться как

зачетные работы по пройденным темам.

2.3 Методические рекомендации по конспектированию первоисточников

Изучение и анализ литературных источников является обязательным видом самостоятельной работы студентов. Изучение литературы по избранной теме имеет своей задачей проследить характер постановки и решения определенной проблемы различными авторами, аргументацию их выводов и обобщений, провести анализ и систематизировать полученный материал на основе собственного осмысления с целью выяснения современного состояния вопроса.

Проработка отобранного материала обязательно должна идти с одновременным ведением записей прочитанного и своих замечаний. Запись может иметь как форму конспекта, так и выписок, а также картотеку положений, тезисов, идей, методик, что в дальнейшем облегчит классификацию и систематизацию полученного материала. Такого рода записи являются лучшим способом накопления и первичной обработки материал, одной из обязательных форм организации умственного труда.

Планом удобно пользоваться при подготовке к устному выступлению по выбранной теме. Каждый пункт плана должен раскрывать одну из сторон избранной темы, а весь план должен охватывать ее целиком.

Тезисы предполагают сжатое изложение основных положений текста в форме утверждения или отрицания. Они являются более совершенной формой записей и представляют основу для дискуссии. К тому же их легко запомнить.

Конспектирование – процесс мысленной переработки и письменной фиксации информации, в виде краткого изложения основного содержания, смысла какого-либо текста.

Результат конспектирования – запись, позволяющая конспектирующему немедленно или через некоторый срок с нужной полнотой восстановить полученную информацию. Конспект в переводе с латыни означает «обзор». По существу его и составлять надо как обзор, содержащий основные мысли текста без подробностей и второстепенных деталей. Конспект носит индивидуализированный характер: он рассчитан на самого автора и поэтому может оказаться малопонятным для других.

Для того чтобы осуществлять этот вид работы, в каждом конкретном случае необходимо грамотно решить следующие задачи:

1. Сориентироваться в общей композиции текста (уметь определить вступление, основную часть, заключение).
2. Увидеть логико-смысловую канву сообщения, понять систему изложения автором информации в целом, а также ход развития каждой отдельной мысли.
3. Выявить «ключевые» мысли, т. е. основные смысловые вехи, на которые «нанизано» все содержание текста.
4. Определить детализирующую информацию.
5. Лаконично сформулировать основную информацию, не перенося на письмо все целиком и дословно.

Во всяком научном тексте содержится информация 2-х видов: основная и вспомогательная. Основной является информация, имеющая наиболее

существенное значение для раскрытия содержания темы или вопроса. К ней относятся: определения научных понятий, формулировки законов, теоретических принципов и т. д. Назначение вспомогательной информации – помочь читателю лучше усвоить предлагаемый материал. К этому типу информации относятся разного рода комментарии. Основную – записываем как можно полнее, вспомогательную, как правило, опускаем. Содержание конспектирования составляет переработка основной информации в целях ее обобщения и сокращения. Обобщить – значит представить ее в более общей, схематической форме, в виде тезисов, выводов, отдельных заголовков, изложения основных результатов и т. п. Читая, мы интуитивно используем некоторые слова и фразы в качестве опорных. Такие опорные слова и фразы называются ключевыми. Ключевые слова и фразы несут основную смысловую и эмоциональную нагрузку содержания текста.

Выбор ключевых слов – это первый этап смыслового свертывания, смыслового сжатия материала.

Важными требованиями к конспекту являются наглядность и обозримость записей и такое их расположение, которое давало бы возможность уяснить логические связи и иерархию понятий.

По форме конспекты подразделяются на формализованные и графические.

1. Формализованные (все записи вносятся в заранее подготовленные таблицы).

Это удобно, во-первых, при конспектировании материалов, когда перечень характеристик описываемых предметов или явлений более или менее постоянен, во-вторых, при подготовке единого конспекта по нескольким источникам. Особенно если есть необходимость сравнения отдельных данных. Разновидностью формализованного конспекта является запись, составленная в форме ответов на заранее подготовленные вопросы, обеспечивающие исчерпывающие характеристики однотипных предметов или явлений.

2. Графические (элементы конспектируемой работы располагаются в таком виде, при котором видна иерархия понятий и взаимосвязь между ними).

По каждой работе может быть не один, а несколько графических конспектов, отображающих книгу в целом и отдельные ее части. Ведение графического конспекта – наиболее совершенный способ изображения внутренней структуры книги, а сам этот процесс помогает усвоению ее содержания.

Можно выделить следующие основные **типы конспектов: плановый, текстуальный, сводный, тематический.**

Плановый – легко получить с помощью предварительно сделанного плана произведения, каждому вопросу плана отвечает определенная часть конспекта:

а) вопросно-ответный (на пункты плана, выраженные в вопросительной форме, конспект дает точные ответы);

б) схематичный плановый конспект (отражает логическую структуру и взаимосвязь отдельных положений).

Текстуальный – это конспект, созданный в основном из цитат.

Сводный конспект – сочетает выписки, цитаты, иногда тезисы; часть его текста может быть снабжена планом.

Тематический – дает более или менее исчерпывающий ответ (в зависимости

из числа привлеченных источников и другого материала, например, своих же записей) на поставленный вопрос – тему: обзорный; хронологический.

Роль конспекта – чисто учебная: он помогает зафиксировать основные понятия и положения первичного текста и в нужный момент их воспроизвести, например, при написании реферата или подготовке к экзамену.

Способы конспектирования.

- **Тезисы** – это кратко сформулированные основные мысли, положения изучаемого материала. Тезисы лаконично выражают суть читаемого, дают возможность раскрыть содержание. Приступая к освоению записи в виде тезисов, полезно в самом тексте отмечать места, наиболее четко формулирующие основную мысль, которую автор доказывает (если, конечно, это не библиотечная книга). Часто такой отбор облегчается шрифтовым выделением, сделанным в самом тексте.

- **Линейно-последовательная запись текста.** При конспектировании линейно – последовательным способом целесообразно использование плакатно-оформительских средств, которые включают в себя следующие:

- сдвиг текста конспекта по горизонтали, по вертикали;
- выделение жирным (или другим) шрифтом особо значимых слов;
- использование различных цветов;
- подчеркивание;
- заключение в рамку главной информации.

- **Способ «вопросов – ответов».** Он заключается в том, что, поделив страницу тетради пополам вертикальной чертой, конспектирующий в левой части страницы самостоятельно формулирует вопросы или проблемы, затронутые в данном тексте, а в правой части дает ответы на них.

Одна из модификаций способа «вопросов – ответов» - таблица, где место вопроса занимает формулировка проблемы, поднятой автором (лектором), а место ответа – решение данной проблемы. Иногда в таблице могут появиться и дополнительные графы: например, «мое мнение» и т. п.

- **Схема с фрагментами** – способ конспектирования, позволяющий ярче выявить структуру текста, - при этом фрагменты текста (опорные слова, словосочетания, пояснения всякого рода) в сочетании с графикой помогают созданию рационально - лаконичного конспекта.

- **Простая схема** – способ конспектирования, близкий к схеме с фрагментами, объяснений к которой конспектирующий не пишет, но должен уметь давать их устно. Этот способ требует высокой квалификации конспектирующего. В противном случае такой конспект нельзя будет использовать.

- **Параллельный способ** конспектирования. Конспект оформляется на двух листах параллельно или один лист делится вертикальной чертой пополам и записи делаются в правой и в левой части листа.

Однако лучше использовать разные способы конспектирования для записи одного и того же материала.

Комбинированный конспект – вершина овладения рациональным конспектированием. При этом умело используются все перечисленные способы, сочетая их в одном конспекте (один из видов конспекта свободно перетекает в другой в зависимости от конспектируемого текста, от желания и умения

конспектирующего). Именно при комбинированном конспекте более всего проявляется уровень подготовки и индивидуальность студента.

Принципы составления конспекта прочитанного.

1. Записать все выходные данные источника: автор, название, год и место издания. Если текст взят из периодического издания (газеты или журнала), то записать его название, год, месяц, номер, число, место издания.

2. Выделить поля слева или справа, можно с обеих сторон. Слева на полях отмечаются страницы оригинала, структурные разделы статьи или книги (названия параграфов, подзаголовки и т. п.), формулируются основные проблемы. Справа – способы фиксации прочитанной информации.

Один из видов чтения – углубленное – предполагает глубокое усвоение прочитанного и часто сохранение информации в целях последующего обращения к ней. Эффективность такого чтения повышается, если прочитанное зафиксировано не только в памяти, но и на бумаге. Психологи утверждают, что записанное лучше и полнее усваивается, прочнее откладывается в памяти. Установлено, что если прочитать 1000 слов и затем записать 50, подытоживающих прочитанное, то коэффициент усвоения будет выше, чем, если прочитать 10000 слов, не записав ни одного. Кроме того, при записи прочитанного формируется навык свертывания информации. И, наконец, чередование чтения и записывания уменьшает усталость, повышает работоспособность и производительность умственного труда.

1. Резюмирование.

Резюме – краткий итог прочитанного, содержащий его оценку. Резюме характеризует основные выводы книги, главные итоги.

Выбор языковых средств для построения резюме-выводов подчинен основной задаче свертывания информации: минимум языковых средств – максимум информации. Это обычно одно – три четких, кратких, выразительных предложения, раскрывающих, по мнению автора, самую суть описываемого объекта.

2. Аннотация.

Аннотация – краткая обобщенная характеристика печатной работы (книги, статьи), включающая иногда и его оценку. Это наикратчайшее изложение содержания первичного документа, дающее общее представление о теме.

Основное ее назначение – дать некоторое представление о книге (статье, научной работе) с тем, чтобы рекомендовать ее определенному кругу читателей или воспользоваться своими записями при выполнении работы исследовательского, реферативного характера. Поэтому аннотации не требуются изложения содержания произведения, в ней лишь перечисляются вопросы, которые освещены в первоисточнике (содержание этих вопросов не раскрывается). Аннотация отвечает на вопрос: «О чем говорится в первичном тексте?», дает представление только о главной теме и перечне вопросов, затрагиваемых в тексте первоисточника.

Текст аннотации не стандартизирован. В научной литературе можно встретить различные требования к составлению аннотаций. Например, текст справочной аннотации может включать следующие сведения:

- тип и название аннотируемого документа (монография, диссертация,

сборник, статья и т. п.)

- задачи, поставленные автором аннотируемого документа
- метод, которым пользовался автор (эксперимент, сравнительный анализ, компиляция других источников)
- принадлежность автора к определенной научной школе или направлению
- структуру аннотируемого документа
- предмет и тему произведения, основные положения и выводы автора
- характеристику вспомогательных и иллюстративных материалов, дополнений, приложений, справочного аппарата, включая указатели и библиографию.