***1. Понятие информации, виды и способы её представления***

Термин информация происходит от латинского informatio, что означает «сведения, разъяснения, изложение».

Информация - это отображение окружающего нас мира с помощью знаков и сигналов или иначе сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые воспринимают информационные системы (живые организмы, управляющие машины и др.) в процессе жизнедеятельности и работы.

Однако можно лишь утверждать, что понятие ИНФОРМАЦИЯ предполагает наличие материального носителя информации, источника информации, передатчика информации, приемника и канала связи между источником и приемником.

Виды информации.

* общественно-политическая,
* социально-экономическая,
* актуальность,
* ясность,
* краткость,
* убедительность.

Формы информации.

Существует множество способов передачи и обработки информации. Человек может передавать информацию, используя тот или иной язык, жесты, мимику, звуки и воспринимать информацию используя любые органы чувств. Иными словами, информация человеком передается, обрабатывается и принимается в форме знаков или сигналами. Сигнал может быть световым, звуковым (радиоволны), электромагнитным, биохимическим и т.д.

Знаками можно считать алфавит любого языка, знаки языка жестов, любые коды или шифры, нотные значки и т.д.

Процесс обработки информации предусматривает наличие носителя информации и средства передачи информации и обработки информации.

Информацию можно:

* создавать
* принимать
* комбинировать
* хранить
* передавать
* копировать
* обрабатывать
* искать
* воспринимать
* формализовать
* делить на части
* измерять
* использовать
* распространять
* упрощать
* разрушать
* запоминать
* преобразовывать
* собирать и т.д.

Все эти процессы, связанные с определенными операциями над информацией, называются информационными процессами.

***2. Задачи получения, передачи, преобразования и хранения информации***

Обработка информации — получение одних информационных объектов из других путем выполнения некоторых алгоритмов.

Обработка является одной из основных операций, выполняемых над информацией, и главным средством увеличения объема и разнообразия информации.

Средства обработки информации — это всевозможные устройства и системы, созданные человечеством, и в первую очередь компьютер — универсальная машина для обработки информации.

Компьютеры обрабатывают информацию путем выполнения некоторых алгоритмов. Живые организмы и растения обрабатывают информацию с помощью своих органов и систем.

Информационные ресурсы — это идеи человечества и указания по реализации этих идей, накопленные в форме, позволяющей их воспроизводство.

Информационные ресурсы (в отличие от всех видов ресурсов — трудовых, энергетических, минеральных) тем быстрее растут, чем больше их расходуют.

Информационные технологии — это совокупность методов и устройств, используемых людьми для обработки информации.

К способам представления информации можно отнести:

всевозможные символы, языки и коды (русский, английский, азбука Морзе, язык жестов, язык мимики), чертежи, картины, музыкальные произведения, кино- и видео фильмы, компьютерные игры и т.п.

Хранить информацию можно:

\* в форме знаков на бумаге, перфокартах, перфоленте, на камне, дереве, ткани, в рельефе ключа и т.п.;

\* в форме электромагнитных сигналов на дисках, дискетах, кассетах,

\* кинопленке и т.п.;

\* в форме структурных комбинаций в клетках и генах биологических объектов;

Передача информации происходит:

\* в форме сигналов, как в технических устройствах, так и в общении людей, в телекоммуникациях, в жизни общества, в живых существах;

\* механическим путем в общении людей и в технических устройствах.

К техническим средствам хранения, передачи и обработки информации относятся:

- различные диски,

- дискеты,

- аудио и видео кассеты,

- перфокарты,

- перфоленты,

- кинопленка,

- объекты телекоммуникации (радио, телевидение, телефон, телеграф, персональные компьютеры и компьютерные сети и т.п.)

ЭВМ — это универсальная вычислительная машина, которая обрабатывает информацию, используя электрические сигналы.

Если сигнал непрерывен во времени, то его называют АНАЛОГОВЫМ, и машины, использующие такие сигналы, называют АНАЛОГОВЫМИ.

Если сигнал имеет прерывистую (дискретную) форму, он называется ЦИФРОВЫМ (дискретным), и машины соответственно называются ЦИФРОВЫМИ. Современные ПК относятся к ЦВМ.

Всю информацию ПК может воспринимать в форме сигналов, которые имеют только два различных состояния: есть сигнал или нет сигнала.

Этим сигналам поставлен в соответствие разряд кода, который может принимать только два значения: 0 или 1.

Сигнал, который имеет только два различных значения, или соответствующий ему разряд кода, который может принимать только два значения - 0 или 1, называется бит. Бит — это минимальная единица информации (англ. Bit- binary digit — двоичная цифра).

На практике чаще применяется более крупная единица — байт, равная восьми битам. Именно восемь бит требуется для того, чтобы закодировать любой из 256 символов (256 = 2^8).

1 байта достаточно для кодирования одного знака текстовой, числовой, звуковой или графической информации.

Крупные производные единицы информации:

1 Килобайт (Кбайт) = 1024 байт

1 Мегабайт (Мбайт) = 1024 Кбайт

1 Гигабайт (Гбайт) = 1024 Мбайт

1 Терабайт(Тбайт) = 1024Гбайт

1 Петабайт(Пбайт)= 1024 Тбайт

1 Экзабайт (Эбайт)=1024 Пбайт

1 Зеттабайт (Збайт)= 1024 Эбайт

1 Йоттабайт (Йбайт)= 1024 Збайт

***3.Язык как способ представления и передачи информации.***

Знаковая форма восприятия, хранения и передачи информации означает использование какого-либо языка. Языки делятся на разговорные (естественные) и формальные. Естественные языки носят национальный характер. Формальные языки чаще всего относятся к специальной области человеческой деятельности (например, язык математики или язык флажков на флоте).

Информация во внешней среде выражается с помощью некоторых материальных объектов (носителей), ассортимент и способ расположения которых задает информацию. Человек воспринимает сообщение посредством органов чувств. Приемник информации в технике воспринимает сообщения с помощью различной измерительной и регистрирующей аппаратуры. Носителем информации в различных информационных процессах может быть, например, камень, бумага, электрический кабель, магнитный диск.

Отображение множества состояний источника во множество состояний носителя называется способом кодирования. Таким образом, при выбранном способе кодирования какое-либо состояние заменяется своим образом — кодом состояния (или кодом информации, задаваемой этим состоянием). Так, мысли источника-человека могут быть закодированы определенным набором звуков, которые в свою очередь можно закодировать какими-то символами. Конечный набор знаков (символов) любой природы, из которых конструируются сообщения, образует алфавит некоторого языка.

Итак, последовательность символов алфавита, кодирующая состояние

источника и воспринимаемая адресатом как сообщение, как информация,

образует слово на этом языке. На передачу и переработку информации

влияет то, сигналами какой природы отображается одна и та же информация,

то есть каким кодом задана одна и та же информация. Если говорить о

сигналах, дискретных по виду, то их множество конечно, поэтому их принято

кодировать буквами алфавита того или иного естественного языка или

цифрами той или иной системы счисления. Таким образом, дискретная

информация отождествляется с алфавитно-цифровой, а простейшим

алфавитом, достаточным для записи (представления) информации, является

алфавит из двух символов, допустим 0 и 1.

***4.Задачи, требующие автоматизированной обработки информации.***

Это задачи, в которых при обработке информации нельзя справится вручную, но одновременно нельзя обойтись без человека. К таким задачам относятся:

***5.Системы автоматизированной и автоматической обработки информации***

Системы автоматизированной обработки информации — это системы, в которых при обработке информации нельзя справится вручную, но одновременно нельзя обойтись без человека. К таким задачам относятся:

Системы автоматической обработки информации — это системы, в которых при обработке информации требуется непосредственное участие человека. К таким задачам относятся:

***6. Исторические этапы автоматизации обработки информации***

**Первый этап (1946, до конца1950-х).** Были созданы электронные цифровые компьютеры, которые строились на электронных лампах. Основной режим использования этих машин состоял в том, что математик, составивший программу, садился за пульт управления машиной и производил необходимые вычисления. В этот период началась интенсивная разработка средств автоматизации программирования, создание входных языков разных уровней, систем обслуживания программ, упрощающих работу на машине и увеличивающих эффективность её использования.

**Второй этап (до середины 1960-х).** С технической точки зрения период чётко очерчен переходом на полупроводники, с точки зрения ЭВМ второго поколения характеризуются расширенными возможностями по вводу-выводу, увеличенным объёмом запоминающих устройств, развитыми системами программирования. Первые ОС просто автоматизировали работу оператора ЭВМ, связанную с выполнением задания пользователя: ввод в ЭВМ текста программы, вызов нужного транслятора, вызов необходимых библиотечных программ и т.д. Теперь же вместе с программой и данными в ЭВМ вводится ещё и инструкция, где перечисляются этапы обработки и приводится ряд сведений о программе и её авторе. Затем в ЭВМ стали вводить сразу по нескольку заданий пользователей (пакет заданий), ОС стали распределять ресурсы ЭВМ между этими заданиями – появился мультипрограммный режим обработки.

**Третий этап (до начала 1970-х).** Базой ЭВМ третьего поколения являются интегральные схемы, состоящей из десятков электронных элементов, образованных в прямоугольной пластине кремня с длиной стороны не более 1 см, позволило увеличить быстродействие и надёжность ЭВМ на их основе, а также уменьшить габариты, потребляемую мощность и стоимость ЭВМ. Машины третьего поколения имеют развитые операционные системы, обладают возможностями мультипрограммирования, т.е. одновременного выполнения нескольких программ. Многие задачи управления памятью, устройствами и ресурсами стала брать на себя операционная система или же непосредственно сама машина. В период машин третьего поколения произошёл крупный сдвиг в области применения ЭВМ. Если раньше ЭВМ использовалась в основном для научно-технических расчётов, то в 1960-1970-е гг. всё больше места стала занимать обработка символьной информации.

**Четвёртый этап (по настоящее время).** На первый план вышла задача экономии человеческих, а не машинных ресурсов. Существовавшая концепция первых этапов информационной технологии постепенно была заменена на новую: «всё, что могут делать машины должны делать машины; люди выполняют лишь ту часть работы, которую нельзя автоматизировать». С точки зрения структуры машины этого поколения представляют собой многопроцессорные и многомашинные комплексы, работающие на общую память и общее поле внешних устройств. Для этого периода характерно широкое применение систем управления базами данных, компьютерных сетей, систем

Последующие поколения ЭВМ будут представлять, по-видимому, оптоэлектронные ЭВМ с массовым параллелизмом и нейронной структурой – с распределённой сетью большого числа (десятки тысяч) несложных процессоров, моделирующих структуру нейронных биологических систем, произойдёт качественный переход от обработки данных к обработке знаний.

Примечание:  
БИС - большие интегральные схемы;  
АСУ - автоматизированная система управления;  
САПР - система автоматизированного проектирования;  
АРМ - автоматизированное рабочее место;  
БЭСМ и МЭСМ - соответственно большая и малая электронные счетные машины;  
ПЭВМ - персональная электронная вычислительная машина.

С каждым новым поколением ЭВМ увеличивались быстродействия и надежность их работы при уменьшении стоимости и размеров, совершенствовались устройства ввода и вывода информации. В соответствии с трактовкой компьютера - как технической модели информационной функции человека - устройства ввода информации (зрительному, звуковому) и, следовательно, операция по ее вводу в компьютер становится все более удобной для человека.

***7. Последовательность обработки прикладных программ***

***8. Жизненный цикл программного обеспечения***

**Системное программное обеспечение:** Комплекс аппаратно - программных средств, расширяющих функциональные возможности операционной системы.

Все программы можно разбить на два класса по характеру использования:

      утилитарные программы — предназначены для удовлетворения нужд их разработчиков, программы «для себя»;

     программные продукты — предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи.

Программный продукт должен быть соответствующим образом подготовлен к эксплуатации, иметь необходимую техническую документацию, предоставлять сервис и гарантию надежной работы программы, иметь товарный знак изготовителя. Только при таких условиях созданный программный комплекс может быть назван **программным продуктом.**

Программный продукт имеет несколько качественных характеристик:

-         алгоритмическая сложность;

-         полнота функций обработки;

-         объём файлов программ;

-         требования к операционной системе и техническим средствам обработки со стороны программного средства;

-         объём дисковой памяти;

-         размер оперативной памяти.

Показатели качества должны содержать следующие аспекты:

-         насколько хорошо можно использовать программный продукт (просто, надёжно, эффективно);

-         насколько легко эксплуатировать программный продукт;

-         можно ли использовать программный продукт при изменении условия его применения.

В условиях существования рынка программных продуктов важными характеристиками являются стоимость, количество продаж, время нахождения на рынке, известность фирмы-производителя и самой программы, наличие на рынке программных продуктов аналогичного назначения.

Программный продукт любого вида характеризуется жизненным циклом, состоящим из отдельных этапов.

**Маркетинг** предназначен для изучения требований к создаваемому программному продукту (технических, программных, пользовательских). Изучаются также существующие аналоги и продукты-конкуренты. Оцениваются необходимые для разработки материальные, трудовые и финансовые ресурсы, а также устанавливаются примерные сроки разработки.

**Проектирование структуры** — алгоритмизация процесса обработки данных, детализация функций, разработка архитектурного проекта, выбор методов и средств создания программ.

**Программирование, тестирование и отладка** — основной этап работы по разработке программного средства. Часто отдельные работы этого этапа ведутся параллельно, что позволяет сократить общее время разработки.

**Документирование** — обязательный вид работы. Документация должна содержать необходимые сведения по установке, обеспечению надёжной работы продукта, справочное пособие для пользователя, демонстрационные версии, примеры документов, создаваемых при помощи данного программного продукта, обучающие программы.

**Выход программного продукта на рынок** связан с организацией продаж массовому пользователю. Здесь применяются стандартные методы — реклама, увеличение числа каналов реализации, создание дилерской и дистрибьюторской сети, гибкая ценовая политика.

**Эксплуатация и сопровождение** идут, как правило, параллельно. В процессе эксплуатации могут выявляться ошибки, и устранение этих ошибок ведётся в режиме сопровождения, то есть оказание сервисной помощи, обеспечение новыми версиями программ, организация «горячих телефонных линий» для консультаций.

**Снятие программного продукта с продажи** и отказ от его сопровождения происходит, как правило, в случае изменения технической политики фирмы-изготовителя, неэффективности работы программного продукта, наличия в нём неустранимых ошибок, отсутствие спроса.

Длительность жизненного цикла разных программных продуктов неодинакова. Для большинства современных программ его длительность составляет 2-3 года. Хотя часто встречаются на компьютерах и давно снятые с производства программные продукты.

***9. Задачи системного программного обеспечения***

Этапы решения задачи на ЭВМ

**Программирование** (programming) - теоретическая и практическая деятельность, связанная с созданием программ. Решение задач на компьютере включает в себя следующие основные этапы, часть из которых осуществляется без участия компьютера.

**1. Постановка задачи:**

• сбор информации о задаче;

• формулировка условия задачи;

• определение конечных целей решения задачи;

• определение формы выдачи результатов;

• описание данных (их типов, диапазонов величин, структуры и т. п.).

**2. Анализ и исследование задачи, модели:**

• анализ существующих аналогов;

• анализ технических и программных средств;

• разработка математической модели;

• разработка структур данных.

**3. Разработка алгоритма:**

• выбор метода проектирования алгоритма;

• выбор формы записи алгоритма (блок-схемы, псевдокод и др.);

• выбор тестов и метода тестирования;

• проектирование алгоритма.

**4. Программирование:**

• выбор языка программирования;

• уточнение способов организации данных;

• запись алгоритма на выбранном языке

программирования.

**5. Тестирование и отладка:**

• синтаксическая отладка;

• отладка семантики и логической структуры;

• тестовые расчеты и анализ результатов тестирования;

• совершенствование программы.

**6. Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5.**

**7. Сопровождение программы:**

• доработка программы для решения конкретных задач;

• составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

Основная категория специалистов, занятых разработкой программ, — это программисты. Программисты неоднородны по уровню квалификации, а также по характеру своей деятельности.

**Системный программист** - занимается разработкой, эксплуатацией и сопровождением системного программного обеспечения, поддерживающего работоспособность компьютера и создающего среду для выполнения программ.

**Прикладной программист** - осуществляет разработку и отладку программ для решения функциональных задач (т. е. задач по реализации функций управления в рамках информационной системы - управление деятельностью торгового предприятия, управление перевозкой грузов, планирование выпуска продукции).

**Программист-аналитик -** программист, анализирующий и проектирующий комплекс взаимосвязанных программ.

**Постановщик задач -** разработчик формальных постановок задач, требующих реализации на ЭВМ.

**Администратор базы данных -** человек, который обеспечивает организационную поддержку базы данных.

**Администратор сети -** человек, который обеспечивает организационную поддержку работы локальной сети.

Основным потребителем программ является **конечный пользователь,** который, как правило, не является специалистом в области программирования.

Для работы с ЭВМ существует группа специально обученных технических работников - **операторов ЭВМ**. Они не программируют, а используют готовые программы для обеспечения работы на ЭВМ конечных пользователей: набор текстов, печать документов, копирование информации, запись на внешние носители и др.

***10. Назначение и структура операционных систем***

Операционная система является программной надстройкой над архитектурой компьютера, которая обеспечивает удобный пользовательский интерфейс, берет на себя функции автоматического управления рядом его подсистем и предоставляет готовые процедуры управления внутренними и внешними ресурсами. То есть, операционная система является некоей автоматизированной системой управления работой и ресурсами компьютера, повышающая удобство и эффективность его использования.

**Операционная система** (ОС) - неотъемлемая часть большинства современных вычислительных систем, управляющая всеми его аппаратными компонентами. Другими словами, Операционная (супервизорная, мониторная, исполнительная) Система (ОС) - есть организованная совокупность программ (систем), которая действует как интерфейс между аппаратурой ЭВМ и пользователями. Она обеспечивает пользователей набором средств для облегчения проектирования, программирования, отладки и сопровождения программ и в то же время управляет распределением ресурсов для обеспечения эффективной работы.

Структуру ОС составляют следующие модули:

* *базовый модуль* (ядро ОС) - управляет работой программ и файловой системой, обеспечивает доступ к ней и обмен файлами между периферийными устройствами;
* *командный процессор* - расшифровывает и исполняет команды пользователя, поступающие прежде всего через клавиатуру;
* *драйверы периферийных устройств* - программно обеспечивают согласованность работы этих устройств с процессором (каждое периферийное устройство обрабатывает информацию по-разному и в различном темпе);
* *дополнительные сервисные программы* (утилиты) - делают удобным и многосторонним процесс общения пользователя с компьютером.

В общем случае операционная система выполняет следующие четыре функции:

1. предоставляет другим программам определенный вид обслуживания (посредством программ-утилит), например выделение и назначение памяти, синхронизацию процесса вычислений и организацию взаимосвязи между различными процессами в вычислительной системе;
2. обеспечивает защиту (в определенной мере) других программ от последствий различных особых ситуаций, возникающих при машинной реализации данной программы, таких, как прерывания и машинные сбои;
3. реализует с той или иной степенью сложности принцип “виртуальной машины”, что позволяет группе программ использовать общие вычислительные ресурсы, например процессор (процессоры) и основную память;
4. организует и следит за выполнением принципов управления при решении таких задач, как обеспечение защиты данных от несанкционированного доступа и реализация системы приоритетов доступа программ к вычислительным ресурсам.

**Графический интерфейс пользователя**

Графический интерфейс пользователя изначально был несвойственен неигровым программам, однако, будучи призванным облегчить общение пользователя с компьютером и программой, хорошо прижился на IBM PC и стал неотъемлемой частью любой уважающей себя операционной системы.

Оболочка Microsoft Windows не была изначально операционной системой, она возникла в виде стандарт затора графического интерфейса и прижилась исключительно потому, что пользователь хотел видеть программу, с которой ему часто приходится работать красивой, практичной, удобной и легкой в освоении и использовании.

***11.Назначение и последовательность функционирования компиляторов***

Компьютерные программы, или *программное обеспечение,* представляют собой файлы, содержащие инструкции, которые указывают компьютеру, что следует делать.

*Программирование* представляет собой процесс определения последовательности инструкций, которые должен выполнить компьютер для решения определенной задачи. Для указания этих инструкций мы используем *язык программирования* C++.

Система должна выделять место в памяти для написания текста. С помощью текстового редактора мы вносим программные операторы в *исходный файл.* Потом мы запускаем программу, различающую текст, т.е. используем — *компилятор —* для преобразования операторов из формата, который мы можем читать и понимать, в единицы и нули, которые понимает компьютер. Переводом текста в машинные команды занимаются программы-трансляторы. Функциями трансляторов являются либо ликвидация неполадок, либо предупреждение о совершенных ошибках. Основной задачей является оптимизация текста.

Современные трансляторы являются компиляторами, следовательно, мы можем сформулировать последовательность функционирования компилятора:

-сначала компилятор проверяет программу на наличие ошибок и сообщает о них.

-создает объектный файл, который работает с относительными адресами (неизвестна точка старта).

-после загрузки работает с абсолютными адресами (.exe). Для этого требуется загрузчик, который является частью операционной системы и компилятора.

После компиляции (экзекуции) информация приходит к драйверам- вывод на экран, принтер, остальную периферию.

Компиляторы бывают однопроходными, т.е. выполняют compile и linking за один раз (присущи огромные размеры), и с разделенными функциями.

Для маломощных машин и программ, не зависящих от аппаратуры, созданы интерпретаторы. Для работы этих программ требуется небольшой объем памяти, т.к. интерпретатор выполняет инструкции последовательно, одна за другой, переводит текст и находит ошибки поштучно(первые версии Basic).

Современные Java – интерпретаторы, поэтому программы, написанные на этом языке, идут на любой маломощной машине, но работают они довольно медленно.

***12. Требования к языкам программирования и их классификация***

**Машинно – ориентированные языки** – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно –ориентированные языки позволяют использовать все возможности и особенности Машинно – зависимых языков:

- высокое качество создаваемых программ (компактность и скорость выполнения);

- возможность использования конкретных аппаратных ресурсов;

- предсказуемость объектного кода и заказов памяти;

- для составления эффективных программ необходимо знать систему команд и особенности функционирования данной ЭВМ;

- трудоемкость процесса составления программ (особенно на машинных языках и ЯСК), плохо защищенного от появления ошибок;

- низкая скорость программирования;

- невозможность непосредственного использования программ, составленных на этих языках, на ЭВМ других типов.

Машинно-ориентированные языки по степени автоматического программирования подразделяются на классы.

- Машинный язык

Отдельный компьютер имеет свой определенный Машинный язык (далее МЯ), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому МЯ является командным.

- Языки Символического Кодирования

Языки Символического Кодирования (далее ЯСК), так же, как и МЯ, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в ЯСК заменены на символы (идентификаторы), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ. Использование символических адресов – первый шаг к созданию ЯСК.

- Автокоды

Есть также языки, включающие в себя все возможности ЯСК, посредством расширенного введения макрокоманд - они называются Автокоды.

Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в «остов» программы, превращая её в реальную машинную программу.

Развитые автокоды получили название Ассемблеры. Сервисные программы и пр., как правило, составлены на языках типа Ассемблер.

- Макрос

Язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ наиболее сжатую форму - называется Макрос (средство замены).

В основном, Макрос предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется макропроцессором. На макропроцессор поступает макро определяющий и исходный текст. Реакция макропроцессора на вызов-выдачу выходного текста.

Макрос одинаково может работать, как с программами, так и с данными.

**Машинно – независимые языки** – это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ и ВС.

Подобные языки получили название высокоуровневых языков программирования. Программы, составляемые на таких языках, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка (задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на МЯ.

**Проблемно – ориентированные языки**

С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач, ими стали проблемно – ориентированные языки. Эти языки, языки, ориентированные на решение определенных проблем, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме.

Фортран, Алгол – языки, созданные для решения математических задач;

**Универсальные языки**

Универсальные языки были созданы для широкого круга задач: коммерческих, научных, моделирования и т.д.

**Диалоговые языки**

Появление новых технических возможностей поставило задачу перед системными программистами – создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ, их назвали диалоговыми языками.

Задачи: управление и описание алгоритмов решения задач...

Одним из примеров диалоговых языков является Бэйсик.

Бэйсик использует обозначения подобные обычным математическим выражениям. Многие операторы являются упрощенными вариантами операторов языка Фортран. Поэтому этот язык позволяет решать достаточно широкий круг задач.

**Непроцедурные языки**

Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по фиксированным алгоритмам (табличные языки и генераторы отчетов), и языков связи с операционными системами.

Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе

***13.Понятие алгоритма, его основные свойства и способы записи***

Исторически термин «алгоритм» произошел от фамилии узбекского математика IX века Мухаммада ибн Муса ал-Хорезми, который впервые сформулировал правила четырех основных арифметических действий. Поначалу именно эти правила назывались алгоритмами, но затем термин получил дальнейшее развитие в первую очередь в математике – алгоритмом стал называться любой способ вычислений, единый для некоторого класса исходных данных, например, нахождение производной функции. На уровне здравого смысла можно сказать, что алгоритм – это точно определенная (однозначная) последовательность простых (элементарных) действий, обеспечивающих решение любой задачи из некоторого класса.

Свойства, которые характерны для алгоритмов:

-Дискретность алгоритма означает, что алгоритм разделен на отдельные шаги (действия), причем, выполнение очередного шага возможно только после завершения всех операций на предыдущем шаге. При этом набор промежуточных данных конечен, и он получается по определенным правилам из данных предыдущего шага.

-Детерминированность алгоритма состоит в том, что совокупность промежуточных величин да любом шаге однозначно определяется системой величин, имевшихся на предыдущем шаге. Данное свойство означает, что результат выполнения алгоритма не зависит от того, кто (или что) его выполняет (т.е. от исполнителя алгоритма), а определяется только входными данными и шагами (последовательностью действий) самого алгоритма.

-Направленность алгоритма: если способ получения последующих величин из каких-либо исходных не приводит к результату, то должно быть указано, что следует считать результатом алгоритма.

-Массовость алгоритма: начальная система величин может выбираться из некоторого множества.

Исполнитель алгоритма – это субъект или устройство, способные правильно интерпретировать описание алгоритма и выполнить содержащийся в нем перечень действий.

Указания по выполнению действий для каждого исполнителя формулируются посредством некоторого языка, включающего набор служебных слов, обозначающих действия (команды), а также синтаксические правила их объединения.

Таким образом, при записи алгоритмов возможны ситуации, когда язык представления алгоритма является формальным, но в нем используются сложные команды, которые самим исполнителем переводятся на уровень истинно элементарных действий.

***14. Временная и объёмная сложность алгоритма***

Наибольший интерес представляет порядок роста необходимых для решения задачи времени и емкости памяти при увеличении входных данных. Обычно с каждой конкретной задачей связывается некоторое число, называемое ее размером, которое выражает меру количества входных данных.

Время, затрачиваемое алгоритмом, как функция размера задачи, называется временной емкостью этого алгоритма.

Для решения одной и той же задачи как правило можно использовать

различные алгоритмы. В связи с этим, возникает необходимость сравнивать

их между собой, и для этого нужны определенные критерии качества алгоритмов.

Временные характеристики алгоритма определяют длительность решения или временную сложность [4].

Длительность решения часто выражается в единицах времени, но удобнее ее выражать через количество операций, так как количество операций не

зависит от быстродействия конкретной машины.

Временной сложностью алгоритма называется зависимость времени

счета, затрачиваемого на получение результатов от объема исходных данных.

Временная сложность позволяет определить наибольший размер задачи,

которую можно решить с помощью данного алгоритма на ПК. Каждый алгоритм можно характеризовать функцией f(n), выражающей скорость роста объема вычислений при увеличении размерности задачи – n. Если эта зависимость

имеет линейный или полиномиальный характер, то алгоритм считается

″хорошим″, если экспоненциальный – ″плохим″.

Для сложных задач эта характеристика имеет большое значение, т.к. ее

изменение значительно сильнее влияет на время решения, чем изменение быстродействия ПК. Например, при зависимости f(n) = 2 увеличение производительности в 10 раз увеличивает размерность задачи, решаемой за то же время, всего на 15% [4].

Объемные характеристики алгоритма определяют его информационную

сложность. Информационная сложность связана со сложностью описания, накопления и хранения исходных, промежуточных и результирующих данных при

решении определенной задачи.

Объем текста алгоритма (программы) определяется количеством операторов, использованных для записи алгоритма.

Объем внутренней и внешней памяти необходимой для хранения данных и

программ при использовании данного алгоритма определяется на основании

расчетов или опытным путем. При недостатке памяти носителей информации используется сегментация программ.

Сложность структуры алгоритма определяется количеством маршрутов, по

которым может реализовываться процесс вычислений и сложностью каждого

маршрута.

Очевидно, что при выборе алгоритмов нужно учитывать не только их характеристики качества, но и способ реализации алгоритма. Например, многие

итерационные алгоритмы удобны для ПК, но слишком трудоемки для человека.

Тип используемой ПК также может влиять на выбор алгоритма (иногда имеет

место и обратный вариант, когда сначала определяется алгоритм и лишь затем

способ реализации).

***15. Оценки временной сложности алгоритма***

***16. Структура рекурсивных алгоритмов***

Применение стека для хранения записей активации позволяет реализовывать не только последовательный вызов подпрограмм, но и рекурсивный вызов.

Рекурсивным вызовом подпрограммы называется вызов подпрограммы из самой себя. При n вызовах подпрограммы A в стек будет последовательно добавлено n записей активации этой подпрограммы. Последний вызов подпрограммы A будет завершен первым, и его запись активации будет первой удалена из стека.

Примером использования рекурсии может служить программа вычисления факториала n! =(n\*(n-1)!), 0!=1.

int factorial(int n)

{

if (n) return n\* factorial(n-1);

else return 1;

}

***17. Структура программы на языке С++***

Препроцессор- специальная программа, которая распознает текст программы. Операторы языка разделяются друг от друга. Программа должна обрабатывать информацию, значит нужны средства ввода/вывода информации. Для некоторых программ не нужно ввода (генератор простых чисел), но вывод должен быть обязательно.

Первоначальная инструкция для пакета ввода/вывода.

*#include <iostream>;*

Команда #include вставляет в программу заранее подготовленные тексты из включаемых файлов.

*< >* означает поиск в служебных файлах.

*iostream (input/output stream)* – название пакета. В отдельных пакетах собраны готовые части, содержащие служебные тексты для ввода и вывода .

*using namespace std;* -использование шаблонов из стандартных пакетов.

*void main ( )*

( ) обязательны, они означают, что main – функция. В скобках указывается список параметров, но обычно в них ничего не пишут.

void означает, что можно не указывать тип переменных.

Если мы указываем тип функции, определяющей результат *(int main* - целые), то необходимо будет выдать результат. Для этого return 0; -любое целое число.

{ – открывающая скобка.

} – закрывающая скобка.

Потоковый ввод: если предположить, что существует всего лишь 2 устройства(ввод и вывод информации). Консоль — это совокупность монитора и клавиатуры, с ней и работает потоковый ввод. Есть 2 объекта:

* консольный ввод (клавиатура)
* консольный вывод (экран)

Для перехода курсора на следующую строку надо добавить endl.

Знак = означает присваивание, а знак = = означает сравнение

***18. Методы доступа к функциям из библиотек и пакетов***

***19. Описания, выражения и операторы в языке С++***

Переменная определяется как пара «имя»-«значение». Имени соответствует ссылка на участок памяти, выделенный переменной, а значением является содержимое этого участка. Именем служит идентификатор, а значение соответствует типу переменной. Переменные типизируются с помощью определений и описаний.

При определении переменных им можно приписывать начальные значения, которые заносятся в выделяемую для них память в процессе инициализации.

Например, *char a=’\n’;*

Все взаимосвязанные атрибуты объектов (тип, класс памяти, области действия имени) приписываются объекту с помощью определений и описаний. Каждое описание делает указанные в нём идентификаторы известными компилятору. Описания сообщают свойства того объекта, к которому относится имя. Описание ассоциирует тип с именем, а определение задает все другие свойства объекта и выполняет его инициализацию. В большинстве случаев определения и описания совпадают.

Выражение — это последовательность операндов, разделителей и знаков операций, задающая вычисление.

Основным исходным элементом любого выражения является *первичное выражение*. К ним относятся:

-константа

-имя

-выражение

-идентификатор

**Константа:**

-целая константа

-символьная константа

-перечислимая константа

-вещественная константа

-строковая константа.

**Имя:**

-**идентификатор**

может использоваться в качестве имени только в том случае, если он введен с помощью подходящего определения. Самый распространенный - идентификатор как имя переменной.

**Выражение**

Третий вариант первичного выражения содержит рекурсию, так как это произвольное выражение, заключенное в круглые скобки.

**Идентификатор.**

Четвертый вариант первичного выражения включает операцию изменения области действия, смысл которой объяснялся.

Часто оператор-выражение служит для вызова функции, не возвращающей никакого значения. еще чаще оператор-выражение – это не вызов функции, а выражение присваивания.

**Операторы.**

Операторы определяют действия и логику (порядок) выполнения действий в программе. Среди операторов выделяют операторы, выполняемые последовательно, и управляющие операторы.

**Последовательно выполняемые операторы.**

Каждый оператор языка Си++ заканчивается и идентифицируется разделителем «точка с запятой». Любое выражение, после которого поставлен символ***;*** воспринимается компилятором как отдельный оператор.

***20. Арифметические типы данных в языке С++***

-32767<a <32767- *int*- тип целочисленных переменных.

*Long* преобразует вычисляемое вещественное значение к целому «длинному» типа (в результате значение без дробной части, дробная часть отбрасывается при приведении типа).

*Long* (около 9 млрд) *long int a* не пишут, *int-* целое, добавляется автоматически.

Действительные числа размером до 10^32 образуют скалярный тип данных-*Float*. Спецификатор типа *float* указывает, что это вещественная арифметическая переменная с внутренним представлением в формате с плавающей точкой. Для десятичных чисел 3-4 знака после запятой – это инженерная точность, однако иногда этого недостаточно, поэтому, чтобы считать с большей точностью, используют тип

*Double.* Работа с числами этого типа занимает больше места в памяти*,* следовательно, результат дольше обрабатывается. При этом последняя цифра может быть ошибочной, т.к. используется двоичная система отсчета.

Нет смысла сравнивать действительные числа знаком =, лучше < или >.

Если мы хотим получить из 5/2 2.5, необходимо:

1)5\*1.0/2- результат *float*

2)*int a*;

Нужно временное преобразование типа *int* в *float*:

*(float) a* /2

Но если мы рассматриваем дальше *(a+3)*, то это уже *int*

3) *float(a)/*2

*float* –не только объявление типа, но еще и функция перевода переменной в тип *float*

4) Специальная функция отбора (от слова «кастинг»):

*static\_cast < float>a*-функция перевода переменной *a* к типу *float*

*static\_cast < int>a*- функция перевода переменной *a* к типу *int*

*static\_cast < long>a*- функция перевода переменной *a* к типу *long*

При переводе из дробных чисел в целые:

3,7 -3

7,999-7.

***21. Символьные и логические данные в языке С++***

Символьные(литерные) константы — это один или два символа, заключенные в апострофы. Одно символьные константы имеют стандартный тип *char*. Для представления их значений могут вводиться переменные символьного типа, т.е. типа char. Переменная типа char имеет размер, нужный для хранения одного символа на данной машине (обычно это один байт) Примеры констант: *‘z’, ‘\*’, ’\012’, ‘\0’, ‘\n’*- одно символьные константы*. ‘ab’, ‘\n\t’*- двух символьные переменные. Символ обратной косой черты *‘\’* используется для записи кодов, не имеющих графических изображений. Кроме того, обратная косая черта позволяет вводить символьные константы, явно задавая их коды в восьмеричном или шестнадцатеричном виде.

Примеры наиболее часто используемых последовательностей:

*\a*- звуковой сигнал

*\b*- возврат на шаг

*\n*- новая строка

*\t*- горизонтальная табуляция

*\v*- вертикальная табуляция

*\\*- обратная косая черта

*\”*- двойная кавычка

*\000* - восьмеричный код символа

*\xhh* - шестнадцатеричный код символа

*Char t = ‘A’; -* это или символьная константа, или значение символьной переменной t.

*String z; -* строка. Строковая константа определяется как последовательность символов, заключенная в кавычки. Формально предела длины строки нет, т.е. строка может быть очень длинной. Но обычно 65-75 символов,80 символов.

Примеры использования служебных символов:

1)*string z = ”AB”*

*cout<<z;*

на экране появится: **AB\_**

2)*cout<<z<<’\n’;*соответственно на экране

**AB**

3)*string z = “A\nB”*

*cout<<z;*

На экране:

**A**

**B\_**

Логический тип данных *(true/false)*

*bool k;* переменная k имеет логический смысл (значением может быть только истина или ложь)

*bool k = true;*

*cout<< k;*

на экране: **1**

Чтобы избежать этого, используются манипуляторы ввода/вывода:

*cout<<boolalpha<< k;* на экране: **true**

Оператор *boolalpha* можно использовать 1 раз, и с этого момента он будет действовать до конца.

!- логическое отрицание

*cout<< boolalpha<< (!k);*  на экране: **false**

Результатом выражения *k == k* будет истина, а *k == !k*- ложь.

!= - не равно

== - сравнение

Операции, которые порождают логические значения:

>< >= <= != =

Они используются при сравнении величин одинакового типа.

***22. Операции над данными стандартных типов и их старшинство***

Унарные операции:

& - операция получения адреса операнда

\* - операция обращения по адресу, т.е. раскрытия ссылки, иначе операция разыменования (доступа по адресу к значению того объекта, на который указывает операнд (адрес)).

- - унарный минус - изменяет знак арифметического операнда.

+ - унарный плюс (введен для симметрии с унарным минусом)

! – логическое отрицание значения операнда.

++ - увеличение на единицу (инкремент или авто увеличение):

префиксная операция – увеличение значения операнда на 1 до его использования

постфиксная операция – увеличение значения операнда на 1 после его использования.

Операнд не может быть константой.

-- - уменьшение на 1 (декремент)

sizeof – операция вычисления размера (в байтах) для объекта того типа, который имеет операнд.

Бинарные операции:

- Аддитивные

* + — сложение арифметических операндов
* - — вычитание арифметических операндов

- Мультипликативные

* \* — умножение операндов арифметического типа
* / — деление операндов арифметического типа
* % — получение остатка от деления целочисленных операндов (деление по модулю)

- Операции сдвига

* << - сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда
* >> - сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда)

-Операции отношения (сравнения) (> < <= >= != = =-равно)

-Логические бинарные операции (&&-конъюнкция(И) и || дизъюнкция (или))

-Операции присваивания

* = присвоить значение выражения-операнда из правой части операнду левой части p=10.3 – 2\*x
* \*= присвоить левой части произведение значений обоих операндов P\*=2 эквивалентно P = P\*2
* /= P/=2.2-d эквивалентно P=P/ (2.2-d)
* %= N%3 эквивалентно N=N % 3;
* += присвоить операнду левой части сумму значений обоих операндов А+= В эквивалентно А=А+В,
* -= Х -=4.5 – z эквивалентно Х=Х – (4.2 – z)

-Запятая в качестве операции (несколько выражений, разделенных запятыми, вычисляются последовательно слева направо. В качестве результата сохраняются тип и результат самого правого значения).

Приоритеты операций задают последовательность вычислений в сложном выражении. Например, какое значение получит ival?

int ival = 6 + 3 \* 4 / 2 + 2;

Если вычислять операции слева направо, получится 20. Среди других возможных результатов будут 9, 14 и 36. Правильный ответ: 14.  
В С++ умножение и деление имеют более высокий приоритет, чем сложение, поэтому они будут вычислены раньше. Их собственные приоритеты равны, поэтому умножение и деление будут вычисляться слева направо. Таким образом, порядок вычисления данного выражения таков:

1. 3 \* 4 => 12

2. 12 / 2 => 6

3. 6 + 6 => 12

4. 12 + 2 => 14

Следующая конструкция ведет себя не так, как можно было бы ожидать. Приоритет операции присваивания меньше, чем операции сравнения:

while (ch = nextChar() != '\n')

Программист хотел присвоить переменной ch значение, а затем проверить, равно ли оно символу новой строки. Однако на самом деле выражение сначала сравнивает значение, полученное от nextChar(), с '\n', и результат – true или false – присваивает переменной ch.  
Приоритеты операций можно изменить с помощью скобок. Выражения в скобках вычисляются в первую очередь. Например:

4 \* 5 + 7 \* 2 ==> 34

4 \* (5 + 7 \* 2) ==> 76

4 \* ((5 + 7) \* 2) ==> 96

Вот как с помощью скобок исправить поведение предыдущего примера:

while ((ch = nextChar()) != '\n')

Операторы обладают и приоритетом, и ассоциативностью. Оператор присваивания право ассоциативен, поэтому вычисляется справа налево:

ival = jval = kva1 = lval

Сначала kval получает значение lval, затем jval – значение результата этого присваивания, и в конце концов ival получает значение jval.  
Арифметические операции, наоборот, левоассоциативны. Следовательно, в выражении ival + jval + kva1 + lva1 сначала складываются ival и jval, потом к результату прибавляется kval, а затем и lval.

***23. Потоковый ввод и вывод информации в языке С++***

Дело в том, что никакая полезная программа не может быть написана на языке С++ без привлечения библиотек, включаемых в конкретную среду (в компилятор) языка. Самая незаменимая из этих библиотек - библиотека ввода-вывода.

**Потоки ввода-вывода**

В соответствии с названием заголовочного файла iostream.h (stream - поток; "i" - сокращение от input - ввод; "o" - сокращение от output - вывод) описанные в этом файле средства ввода-вывода обеспечивают программиста механизмами для извлечения данных из потоков и для включения (внесения) данных в потоки. Поток определяется как последовательность байтов (символов) и с точки зрения программы не зависит от тех конкретных устройств (файл на диске, принтер, клавиатура, дисплей, стример и т.п.), с которыми ведется обмен данными. При обмене с потоком часто используется вспомогательный участок основной памяти - буфер потока.

В буфер потока помещаются выводимые программой данные перед тем, как они будут переданы к внешнему устройству. При вводе данных они вначале помещаются в буфер и только затем передаются в область памяти выполняемой программы. Использование буфера как промежуточной ступени при обменах с внешними устройствами повышает скорость передачи данных, так как реальные пересылки осуществляются только тогда, когда буфер уже заполнен (при выводе) или пуст (при вводе).

Работу, связанную с заполнением и очисткой буферов ввода-вывода, операционная система очень часто берет на себя и выполняет без явного участия программиста. Поэтому поток в прикладной программе обычно можно рассматривать просто как последовательность байтов. При этом очень важно, что никакой связи значений этих байтов с кодами какого-либо алфавита не предусматривается. Задача программиста при вводе-выводе с помощью потоков - установить соответствие между участвующими в обмене типизированными объектами и последовательностью байтов потока, в которой отсутствуют всякие сведения о типах представляемой (передаваемой) информации.

Используемые в программах потоки логически делятся на три типа:

• входные, из которых читается информация;

• выходные, в которые вводятся данные;

• двунаправленные, допускающие как чтение, так и запись.

Все потоки библиотеки ввода-вывода последовательные, т.е. в каждый момент для потока определены позиции записи и (или) чтения, и эти позиции после обмена перемещаются по потоку на длину переданной порции данных.

В соответствии с особенностями "устройства", к которому "присоединен" поток, потоки принято делить на

• стандартные,

• консольные,

• строковые и

• файловые.

В заключение перечислим отличительные особенности применения механизма потоков. Потоки обеспечивают:

• буферизацию при обменах с внешними устройствами;

• независимость программы от файловой системы конкретной операционной системы;

• контроль типов передаваемых данных;

• возможность удобного обмена для типов, определенных пользователем.

***24.Манипуляторы потокового ввода и вывода***

Манипуляторами называют специальные функции, позволяющие программисту изменять состояния и флаги потока. Особенность манипуляторов и их отличие от обычных функций состоит в том, что их имена (без параметров) и вызовы (с параметрами) можно использовать в качестве правого операнда для операции обмена (<< или >>). В качестве левого операнда в этом выражении, как обычно, используется поток (ссылка на поток), и именно на этот поток оказывает влияние манипулятор.

Манипуляторы библиотеки классов ввода-вывода языка С++ делятся на две группы: манипуляторы с параметрами и манипуляторы без параметров.

Манипуляторы без параметров:

• dec - при вводе и выводе устанавливает флаг десятичной системы счисления;

• hex - при вводе и выводе устанавливает флаг шестнадцатеричной системы счисления;

• oct - при вводе и выводе устанавливает флаг восьмеричной системы счисления;

• endl - действует только при выводе, обеспечивает включение в выходной поток символа новой строки и сбрасывает буфер (выгружает содержимое) этого потока;

Манипуляторы с параметрами определены в файле iomanip.h. Перечислим их:

**setbase(int n) -** устанавливает основание (n) системы счисления. Значениями параметра n могут быть: 0, 8, 10 или 16. При использовании параметра 0 основание счисления при выводе выбирается десятичным.

**setfill(int n) -** значение параметра n в дальнейшем используется в качестве кода символа-заполнителя, который помещается в незанятых позициях поля при вводе значения **setprecision(int n) -** определяет с помощью значения параметра n точность представления вещественных чисел, т.е. максимальное количество цифр дробной части числа при вводе и выводе

**setw(int n)-** значение параметра n задает минимальную ширину поля вывода.

**showpoint-** показывать точку в десятичной записи числа.

**showbase**- показывать систему исчисления.

**skipws**- пропуск пробелов.

С помощью манипуляторов можно управлять представлением информации в выходном потоке.

***25. Ссылочные типы данных и их применение***

Чтобы упростить процесс изменения параметров, С++ вводит такое понятие как *ссылка.* Как вы узнаете из этого урока, ссылка представляет собой псевдоним (или второе имя), который ваши программы могут использовать для обращения к переменной. К концу данного урока вы освоите следующие основные концепции:

• Для объявления и инициализации ссылки внутри программы объявите переменную, размещая амперсанд (&) сразу же после типа переменной, и затем используйте оператор присваивания для назначения псевдонима, например *int& alias\_name = variable',.*

*•* Ваши программы могут передавать ссылки в функцию в качестве параметров, а функция, в свою очередь, может изменять соответствующее значение параметра, не используя указателей.

• Внутри функции вам следует объявить параметр как ссылку, размещая амперсанд (&) после типа параметра, затем можно изменять значение параметра внутри функции без помощи указателей.

## ПРАВИЛА РАБОТЫ СО ССЫЛКАМИ

Ссылка не является переменной. Один раз присвоив значение ссылке, вы уже не можете ее изменить. Кроме того, в отличие от указателей вы не можете выполнить следующие операции над ссылками:

• Вы не можете получить адрес ссылки, используя оператор адреса C++.

• Вы не можете присвоить ссылке указатель.

• Вы не можете сравнить значения ссылок, используя операторы сравнения C++.

• Вы не можете выполнить арифметические операции над ссылкой, например добавить смещение.

• Вы не можете изменить ссылку.

***26. Указатели в языке С++. Доступ к данным по указателю.***

Указатель — это переменная, содержащая адрес другой переменной, т.е. значением переменной типа указатель является целое число, равное адресу того объекта, на который ссылается указатель.

Указатель существенно связан с типом объекта, на который он ссылается. Если в описании перед обозначением объекта поставить символ "\*", то оно будет описывать указатель на объект того же типа и класса памяти, которые соответствуют данному обозначению без звездочки.

Указатели и целые не являются взаимозаменяемыми объектами. Константа - единственное исключение из этого правила: ее можно присвоить указателю и указатель можно сравнить с нулевой константой. Чтобы показать, что это специальное значение для указателя, вместо числа 0, как правило, записывают NULL - константу, определенную, в том числе, в файле stdio.h.

Унарная операция "\*", называемая операцией косвенной адресации, рассматривает свой операнд как адрес объекта и обращается по этому адресу, возвращая его содержимое.

***27. Структурный тип данных***

Структура, в отличие от массива и строки, может содержать элементы различных типов.

Описание структур может выполняться двумя способами. Первый - унаследован от С:

**struct [<имя структуры>] {<описание полей>} [<список переменных>];**

Как следует из формы, при описании структур в этом случае возможны два варианта:

а) описание структуры выполняется отдельно от объявления списка переменных, например:

**struct student { char name[22];**

**char family[22];**

**int old; };** // описание структуры

**struct student stud1, stud2, \*ptrstud;** // объявление переменных

б) описание структуры выполняется совместно с объявлением переменных (в этом случае имя структуре можно не присваивать), например:

**struct { char name[22];**

**char family[22];**

**int old; } stud1, stud2, \*ptrstud;**

Второй способ описания структур появился позднее, когда в язык был введен оператор объявления нового типа **typedef.** С использованием этого оператора создается новый тип данных - структура, например:

**typedef struct { char name[22];**

**char family[22];**

**int old; } student;** // в данном случае студент - это имя нового типа данных. Соответственно возможно объявление переменных данного типа, например:

**student stud1, stud2, \*ptrstud;** // объявляются две переменные типа **student** и один указатель на поле данного типа. В последнем случае память под размещение структуры надо запрашивать специально, например: **ptrstud=new student;**

Обращение к элементам структуры выполняется с указанием имени переменной и имени поля:

**<имя переменной>.<имя поля>**

Например:

**stud1.family** или (в том случае, если переменная задается своим адресом) **(\*ptrstud).family**

Примечание. В том случае, если переменная задается адресом, удобнее использовать сокращенную форму записи обращения к элементу структуры: **ptrstud->family**  вместо **(\*ptrstud).family**

Статические и внешние структуры при объявлении можно сразу инициализировать. При этом значения полей указываются после знака равенства в фигурных скобках через запятую, например:

**static student stud1={“Петр”,”Петров”,18},**

**\*ptrstud={“Иван”,”Иванов”,19};**

Возможно объявление массивов структур и структур, элементами которых являются вложенные структуры.

***28. Массивы в языке С++. Связь массивов и указателей***

Массив- конечная последовательность элементов одного и того же типа. Каждый элемент массива имеет селектор (индекс, номер). С его помощью можно определить любой элемент.

double t[48];

В С++ индексы всегда начинаются с 0, поэтому в данном случае последний индекс- 47.

*double t[48];*

*double \*tt= &t или =&t[0]*

Имя массива система понимает как указатель на этот массив.

*cout << \*tt;* //значение 1-го элемента массива

*cout << \*t << t[0];*

Чтобы получить доступ к 3-му элементу массива, можно:

*t[3]* или *\*(t+3)*

*int a []= {3,4,5,6,7,8,9);* Можно объявить массив, не указывая явно, сколько в нем элементов. При этом система должна знать, сколько места в памяти будет этот массив занимать.

Каждому массиву при обработке нужно задать еще один параметр: сколько элементов в нем в действительности находится.

Исключение: массивы символов. *char t [ ]= { ‘7’, ‘6’, ‘5’ };*

Каждый символ занимает 1 байт. Символьные массивы отличаются от всех остальных тем, что после символов стоит значок- символ, означающий, что перечисление символов закончено.

*char t [ ]= { ‘7’, ‘6’, ‘5’ };* \0 -1 символ

Поэтому в памяти выделяется 4 байта.

Если задать *char [3]= {‘7’, ‘6’, ‘5’};* ,мы допустим ошибку, т.к. 1 место уже занято \0. Поэтому в символьных массивах лучше не указывать [x] Т.е. лучше оставлять[ ].

Последовательность символов – строка.

*{ ‘7’, ‘6’, ‘5’}= “765”*

*char t[ ] = “765”;*

1. Массив представляет собой переменную, которая может хранить одно или несколько значений одного и того же типа.
2. Для объявления массива вам следует указать тип, имя массива, а также количество значений, хранящихся в массиве.
3. Значения внутри массива называются элементами массива.
4. Первый элемент массива хранится как элемент 0 *(array[OJ),* индекс последнего элемента массива на единицу меньше размера массива.
5. Программы часто используют индексные переменные для обращения к элементам массива.
6. Если функция воспринимает массив как параметр, она должна указать тип и имя, но не размер массива.
7. Если программа передает массив в функцию, она, как правило, передает и параметр, который сообщает функции количество элементов, содержащихся в массиве.
8. Так как C++ передает массив в функцию с помощью адреса массива функция может изменять значения, содержащиеся в массиве.

***29. Многомерные массивы***

Многомерный массив представляет собой массив массивов, то есть массив, элементами которого служат массивы. Определение многомерного массива в общем случае должно содержать сведения о типе, размерности и количествах элементов каждой размерности, например описание: int ARRAY[4][3][6]; определяет массив, состоящий из четырех элементов, каждый из которых - двухмерный массив с размерами 3 на 6. В памяти массив ARRAY размещается в порядке возрастания самого правого индекса, то есть самый младший адрес имеет элемент ARRAY[0][0][0], затем идет элемент ARRAY[0][0][1] и т.д.

Компилятор первым делом вычисляет, сколько всего элементов в массиве умножается на double, отводится место в памяти.

В массиве, где n- индексов, при вычислении функции делается n- умножений.

a[3][4]= a[3][4]+1-ужасная эффективность.

Лучше использовать еще одну ячейку памяти. Чтобы ускорить процесс работы системы нужно не каждый раз вызывать переменную из массива, а, вызвав 1 раз, вложить новую переменную и совершить нужные действия.

С учетом порядка расположения в памяти элементов многомерного массива нужно размещать начальные значения его элементов в списке инициализации. Например, конструкция:

int ARRAY [4][3][6] = {0,1,2,3,4,5,6,7}

инициализирует только первые 8 элементов этого массива:

ARRAY[0][0][0]=0, ARRAY[0][0][1]=1, ARRAY[0][0][2]=2, ARRAY[0][0][3]=3,

ARRAY[0][0][4]=4, ARRAY[0][0][5]=5, ARRAY[0][1][0]=6, ARRAY[0][1][1]=7.

Если многомерный массив при определении инициализируется, то его самая левая размерность может в скобках не указываться. Количество элементов компилятор определяет по числу членов в инициализирующем списке. Например, определение:

float matrix [ ][5] = { {1}, {2}, {3} };

формирует массив matrix с размерами 3 на 5, но не определяет явно начальных значений всех его элементов.

***30. Строки и массивы символов***

Строки предназначены для ввода, обработки и вывода символьной информации.

Строковая константа — это последовательность из нуля или более символов, заключенных в кавычки. Кавычки не являются частью строковой константы, а служат только для ее ограничения.

Строки представляются в виде массива элементов типа char. Это означает, что символы строки можно представить расположенными в соседних ячейках памяти - по одному символу в ячейке. Но массив символов - не всегда строка!

последним элементом массива является символ '\0'. Это нуль-символ: в языке C++ он используется для того, чтобы отмечать конец строки. Нуль-символ - не есть цифра 0; он не выводится на печать и в таблице кодов ASCII имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть по крайней мере на одну больше, чем количество символов, которые необходимо размещать в памяти.

Для символьных массивов существует специальный способ инициализации. Вместо фигурных скобок и запятых можно прямо использовать строку символов, заключенных в кавычки. При этом в описании не обязательно задавать размеры массива, поскольку компилятор "сам" определяет его длину, подсчитывая число начальных значений.

#include <iostream.h>

int n=5;

char line[] = "Дом!"; /\* Инициализация символьного

массива внешнего класса памяти. \*/

void main ()

{

cout << "Получили слово: ";

for (int i=0; i<n; i++)

cout << line[i];

1. Символьная строка представляет собой массив символов, завершающийся 0 (символом NULL).
2. Вы создаете символьную строку, объявляя массив типа *char.*
3. Ваша программа ответственна за размещение символа NULL за последним символом строки.
4. Если программа использует строковые константы, заключенные в
5. двойные кавычки, компилятор C++ автоматически добавляет символ NULL.
6. C++ позволяет вам инициализировать строки при объявлении, указывая требуемые символы внутри двойных кавычек.
7. Большинство компиляторов C++ в своих библиотеках этапа выполнения обеспечивают широкий набор функций для манипулирования строками.

***31. Файлы и потоки***

***32. Методы открытия файлов для обмена информацией***

***33. Прямой доступ к отдельным записям в файлах***

***34. Программная реализация алгоритмов линейной структуры***

Базовой структурой **следование** называют последовательность операций, которые выполняются по очереди в порядке их записи. Программную реализацию такой алгоритмической структуры называют **линейной программой**. Линейные программы обычно предназначены для решения простейших задач. Как правило, для вычисления по формуле. В них не предусмотрен выбор из нескольких возможных направлений хода программы или циклическое повторение операций.

Алгоритм программы с линейной структурой может быть представлен следующим образом:



***35. Программная реализация разветвляющихся алгоритмов***

Базовой структурой **следование** называют последовательность операций, которые выполняются по очереди в порядке их записи. Программную реализацию такой алгоритмической структуры называют **линейной программой**. Линейные программы обычно предназначены для решения простейших задач. Как правило, для вычисления по формуле. В них не предусмотрен выбор из нескольких возможных направлений хода программы или циклическое повторение операций.

Алгоритм программы с линейной структурой может быть представлен следующим образом:



***36. Реализация циклических алгоритмов с заранее неизвестным количеством повторений***

Для программирования циклических процессов в С++ используется три оператора: “повторять пока выполняется условие”(**while)**, “повторять до выполнения условия”(**do ... while)** и “повторять от исходного значения до конечного значения с заданным шагом”(**for)**.

Но мы рассмотрим только первые два случая, количество повторений заранее не известно.

Цикл while.

Используется, если выход из цикла должен осуществляться при нарушении некоторого условия, заданного в виде выражения.

**while (<выражение>) <оператор>**

Тело цикла выполняется до тех пор, пока результат выражения отличен от нуля (**true**).

Цикл do ... while.

Используется, если выход из цикла также при нарушении некоторого условия, но проверка должна производиться после выполнения тела цикла.

**do <оператор > while (<выражение>);**

Тело цикла выполняется, пока результат выражения отличен от нуля (**true**), но в отличие от цикла “пока”, тело цикла всегда выполняется хотя бы один раз.

***37. Реализация циклических алгоритмов с заранее известным количеством повторений***

### Оператор for

Оператор for это наиболее общий способ организации цикла. Он имеет следующий формат:

for (выражение\_1; выражение\_2; выражение\_3) тело

Выражение\_1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение\_2 это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение\_3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла. В качестве тела может быть использован любой оператор, в том числе пустой или составной.

Любое из выражений, а также все сразу, могут быть опущены, при этом разделяющие их символы; пропускать нельзя. Схема выполнения оператора for:

1. Вычисляется выражение\_1.

2. Вычисляется выражение\_2.

3. Если значения выражения\_2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение\_3 и осуществляется переход к пункту 2, если выражение\_2 равно нулю (ложь), выполнение оператора for завершается и управление передается на оператор, следующий за оператором for. При отсутствии выражения\_2 оно подразумевается истинным.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

int main ()

{

int i, b;

for (i=1; i<10; i++) b=i\*i;

return 0;

}

В этом примере вычисляются квадраты чисел от 1 до 9.

Другим вариантом использования оператора for является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют условный оператор и оператор break.

Пример:

for ( ; ; )

{ ...

if ( некоторое условие ) break;

... }

Согласно синтаксису языка Си тело цикла должно иметься у любого цикла, но если по смыслу оно не требуется, то в качестве тела можно использовать пустой оператор. Такой цикл может быть использован, например для организации поиска.

for (i=0; t[i]>=0; i++) ;

В данном примере переменная цикла i примет значение номера первого по порядку отрицательного элемента массива t.

***38. Структура функции в языке С++. Прототип, заголовок и вызов функции.***

Прототип — это явное объявление функции, которое предшествует определению функции.

void inputmat(float A, int\* n, int\* m);

Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции.

Если прототип функции не задан, а встретился вызов функции, то строится неявный прототип из анализа формы вызова функции. Тип возвращаемого значения создаваемого прототипа int, а список типов и числа параметров функции формируется на основании типов и числа фактических параметров, используемых при данном вызове.

Таким образом, прототип функции необходимо задавать в следующих случаях:

1. Функция возвращает значение типа, отличного от int.

2. Требуется проинициализировать некоторый указатель на функцию до того, как эта функция будет определена.

Наличие в прототипе полного списка типов аргументов параметров позволяет выполнить проверку соответствия типов фактических параметров при вызове функции типам формальных параметров, и, если необходимо, выполнить соответствующие преобразования.

В прототипе можно указать, что число параметров функции переменно, или, что функция не имеет параметров.

Вызов функции имеет следующий формат:

адресное-выражение ([список-выражений])

Поскольку синтаксически имя функции является адресом начала тела функции, в качестве обращения к функции может быть использовано адресное-выражение, имеющее значение адреса функции.

Список-выражений представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, но наличие круглых скобок обязательно.

Фактический параметр может быть величиной любого основного типа, структурой, объединением, перечислением, или указателем на объект любого типа. Массив и функция не могут быть использованы в качестве фактических параметров, но можно использовать указатели на эти объекты.

Выполнение вызова функции происходит следующим образом:

1. Вычисляются выражения в списке выражений и подвергаются обычным арифметическим преобразованиям. Затем, если известен прототип функции, тип полученного фактического аргумента сравнивается с типом соответствующего формального параметра. Если они не совпадают, то либо производится преобразование типов, либо формируется сообщение об ошибке.

2. Происходит присваивание значений фактических параметров соответствующим формальным параметрам.

3. Управление передается на первый оператор функции.

4. Выполнение оператора return в теле функции возвращает управление и возможно, значение в вызывающую функцию. При отсутствии оператора return управление возвращается после выполнения последнего оператора тела функции, а возвращаемое значение не определено.

Адресное выражение, стоящее перед скобками, определяет адрес вызываемой функции. Это значит, что функция может быть вызвана, через указатель на функцию.

Формальные аргументы при описании функции показывают, что будет

делаться с фактическими аргументами, когда функция будет вызвана.

Для вызова функции достаточно написать ее имя со списком

передаваемых фактических аргументов.

Компилятор использует объявление заголовка функции

для сравнения типов фактических аргументов при вызове функции

с формальными параметрами даже при отсутствии явного описания.

***39. Предварительное описание функции***

При внимательном рассмотрении каждой из предшествующих программ можно отметить, что определения функций всегда появляются в программном коде раньше, чем к ним выполняется обращение. Большинство современных Си-компиляторов требует, чтобы информация о типе возвращаемых значений и типах параметров функций была известна до вызова этих функций в программе. Расположение определения функции текстуально ранее первого ее использования обеспечивает Си-компилятор требуемой информацией. Однако, по мере усложнения программы размещение функций в исходном коде с соблюдением "правильного" порядка может оказаться невозможным. Для таких случаев языком Си предлагается использовать прототипы функций, описывающие возвращаемые значения и типы параметров функций. Например, рассмотрим программу, использующую функции i\_cube и f\_cube. До появления первого вызова этих функций в программу можно включать прототипы следующим образом:

int i\_cube(int); // Возвращает int, параметр int

int f\_cube(float); // Возвращает float, параметр float

Как можно видеть, прототип функции определяет тип возвращаемого значения и типы параметров функции.

***40.Локальные и глобальные величины программы***

### Локальные переменные

Как известно, в языке Си переменные можно объявлять внутри функций. Такие переменные называются локальными переменными, поскольку их имена и значения имеют смысл только внутри функций, в которых они определены.

### Объявление глобальных переменных

Кроме локальных переменных, Си также позволяет использовать в программе глобальные переменные, чьи имена, значения и существование известны по всей программе.

К сожалению, глобальные переменные часто становятся причиной появления большого числа ошибок. Поскольку значение глобальной переменной может быть изменено фактически из любого места программы, это очень усложняет поиск в программе всех мест изменения этих переменных. В этом случае можно сделать некоторое изменение в программе, не имея полного представления о влиянии этого изменения на глобальную переменную. Как правило, функции должны изменять только те переменные, которые передаются ей в качестве параметров. В таком случае для быстрого определения изменяемых параметров функции достаточно ознакомиться с прототипом этой функции. Если обнаруживается, что в разрабатываемой программе используются глобальные переменные, то следует пересмотреть структуру этой программы с целью сведения к минимуму использования глобальных переменных.

### Правильное задание области действия глобальной переменной

Как известно, глобальная переменная — это переменная, которой могут оперировать все функции программы. В действительности, в зависимости от выбора места определения глобальной переменной, можно контролировать набор функций, имеющих доступ к этой переменной. Другими словами, можно управлять областью действия глобальных переменных. При определении в программе глобальной переменной эта переменная может использоваться в функциях, определения которых располагаются после объявления этой переменной и до конца исходного файла. Функции, определения которых располагаются до определения глобальной переменной, не имеют доступа к этой переменной.

***41. Способы обмена информацией между вызывающей и вызываемой функциями***

Механизм параметров является основным способом обмена информацией между вызываемой и вызывающей функциями. Параметры, перечисленные в заголовке описания функции, называются *формальными*, а записанные в операторе вызова функции — *фактическими (или аргументами)*.

При вызове функции в первую очередь вычисляются выражения, стоящие на месте фактических параметров; затем в стеке выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего фактического параметра. При этом проверяется соответствие типов и при необходимости выполняются их преобразования. При несоответствии типов выдается диагностическое сообщение.

Существует два вида передачи величин в функцию: по значению и по адресу.

*При передаче по значению* в стек заносятся копии значений фактических параметров, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и возможности их изменить.

*При передаче по адресу* в стек заносятся копии адресов параметров, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения параметров.

*При передаче по ссылке* в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются. Поэтому использование ссылок вместо указателей улучшает читаемость программы. Использование ссылок вместо передачи по значению более эффективно, поскольку не требует копирования параметров.

Если требуется запретить изменение параметра, используется модификатор const:

int f(const char\*);

char\* t(char\* a, const int\* b);

Параметры, передаваемые в функцию, могут быть любого типа (например, вещественного, структурой, перечислением, объединением, указателем), кроме массива или функции, которые передаются с помощью указателей.

***42.Передача функций в качестве параметров функций***

Функцию можно вызвать через указатель на нее. Для этого объявляется указатель соответствующего типа и ему с помощью операции взятия адреса присваивается адрес функции:

void f(int a ){ /\* ... \*/ }//определение функции

void (\*pf)(int);//указатель на функцию

...

\*pf = &f;// указателю присваивается адрес функции (можно написать pf = f;)

pf(10);// функция f вызывается через указатель pf (можно написать (\*pf)(10) )

Указатели на функции передаются в подпрограмму таким же образом, как и параметры других типов:

void fun(PF pf)// функция fun получает в качестве параметра указатель типа PF

{... pf(10); ...}//вызов функции, переданной через указатель

Тип указателя и тип функции, которая вызывается посредством него, должны совпадать в точности.

***43.Перегрузка функций в языке С++***

Перегрузка функций позволяет вашим программам определять несколько функций с одним и тем же именем и типом возвращаемого значения. Перегрузка функций улучшает удобочитаемость программ. Перегрузка функций C++ позволяет вашим программам определять несколько функций с одним и тем же именем. Перегруженные функции должны возвращать значения одинакового типа, но могут отличаться количеством и типом параметров.

Перегрузка функций позволяет вам указать несколько определений для одной и той же функции. В процессе компиляции C++ определит, какую функцию следует использовать, основываясь на количестве и типе передаваемых параметров.

***44. Динамические переменные. Выделение и освобождение памяти при выполнении программы.***

Динамические переменные – это переменные, которые создаются (и уничтожаются) во время выполнения программы. В зависимости от того, кто отвечает за работу с такими переменными – компилятор или программист – различают языки с динамическим или статическим управлением памятью.

Динамические переменные создаются в куче (heap) - специально отведенной для этой цели области оперативной памяти, которая располагается между стеком и статической областью команд и данных.

|  |
| --- |
| **Область программного кода** |
| **Область статических данных** |
| **Область динамических данных (куча)** |
| **С** (вершина стека) **т** **е** **к** |

Для создания и уничтожения динамических переменных в С++ используются операторы new и delete, соответственно. Значением оператора new является адрес созданной переменной, а в случае неудачи NULL. Созданная переменная не имеет имени и связь с нею можно осуществить только одним способом – присвоив ее адрес указателю соответствующего типа. При уничтожении динамической переменной ее адрес (т.е. значение соответствующего указателя) передается оператору delete.

Область видимости динамической переменной совпадает с областью видимости указателя на нее, а время жизни заключено между действием оператора new и оператора delete.

***45. Построение списковых (последовательных) структур***

Списковые структуры данных (ССД) – это множество физически не связанных элементов, для которых отношение следования определено с помощью специальных адресов связи.

В адресе связи указывается адрес элемента, следующего в логическом порядке хранения за данным элементом.

Элементы ССД могут быть двух типов: простые, логически неделимые (их называют подсписками) или сложные – совокупность простых и сложных меньшего объема.

В простые ССД (или строки, или цепи) входят только простые элементы. В сложные ССД входят и простые, и сложные элементы.

Каждый элемент ССД содержит собственную информацию – значение элемента и ассоциативную информацию – адреса связи с другими элементами структуры, которые объединяются в звенья связи.

По виду взаимосвязи элементов различают однонаправленные, двунаправленные и кольцевые списковые структуры.

В однонаправленных списках реализуется взаимосвязь между элементами типа «следующий». Каждый элемент такого списка содержит указатель с адресом следующего элемента. Последний элемент имеет в указателе вместо адреса связи специальный знак – признак конца списка. Указатель списка содержит адрес его первого элемента.

Для задания однонаправленной списковой структуры требуется следующая ассоциативная информация:

-указатель списка с адресом первого элемента;

-звено связи элементов, в которых для простого элемента содержатся адрес следующего элемента списка и адрес значения элемента, а для сложного элемента – адрес следующего элемента списка и адрес первого элемента подсписка.

Двунаправленные списки ориентированы на обработку как в прямом, так и в обратном направлении. Для этого в звенья связи дополнительно вводится адрес, реализующий связь типа «предыдущий». Для задания двунаправленной списковой структуры необходима следующая ассоциативная информация:

-указатель списка, содержащий адрес первого и последнего элементов;

-звенья связи элементов, для простого элемента это звено содержит адреса предыдущего и последующего элементов, а также адрес значения элемента, для сложного элемента в звене связи содержится адрес последующего и предыдущего элементов списка и адреса первого и последнего элемента подсписка.

Кольцевой называется такая списковая структура, элементы которой могут быть просмотрены в циклической последовательности заданное число раз. Кольцевые структуры могут быть как однонаправленными, так и двунаправленными, могут быть простыми (строчными) и сложными (сложные кольцевые структуры редко используются на практике).

Для задания однонаправленной простой кольцевой структуры необходимо иметь следующую ассоциативную информацию:

-указатель строки, который содержит адрес указателя начала кольца;

-указатель начала кольца, который хранит константу N – число просмотров строки, и адрес первого элемента строки;

-звенья связи элементов, содержащие адрес последующего элемента и адрес значения элемента; звено связи последнего элемента вместо признака конца списка содержит адрес указателя начала кольца. При каждом просмотре кольца значение N уменьшается на единицу и проверяется условие N=0. Если N≠0, просмотр продолжается; при N=0 просмотр заканчивается.

Двунаправленная кольцевая строка отличается от однонаправленной тем, что вместо указателя начала кольца вводятся два указателя со своими константами з указатель начала прямого направления и указатель начала обратного направления со своими константами чисел просмотра N1 и N2. Кроме того, звенья связи содержат адреса предыдущего и последующего элементов.

***46. Последовательные, циклические и рекурсивные алгоритмы***

***47. Базовая структура цифровой машины с хранимой программой***

***48. Архитектура и структура команд четырёхадресной машины***

Команда к процессору ЭВМ первоначально содержала следующую информацию:

1. Код операции, указывающий на операцию, которую должна выполнить ЭВМ (сложение, вычитание, и т.п.)
2. Адрес первой ячейки памяти (первый операнд),
3. Адрес второй ячейки памяти (второй операнд)
4. Адрес ячейки памяти, куда нужно поместить результат
5. Адрес следующей команды алгоритма.

Адресная часть описывает, где используемая информация хранится и куда поместить результат.

Итого пять полей, из них один код операции (КОП) и четыре адреса.

При такой структуре ЭВМ первый операнд приходится сохранять в процессоре, так как нельзя одновременно прочесть из памяти оба операнда. Ячейка, в которую сохраняется значение, называется аккумулятором.

В четырехадресной команде на каждый адрес приходится по 2 бита, то есть можно адресовать только 23=8 ячеек памяти. В современных ЭВМ используется намного большее количество ячеек, поэтому четырехадресные команды не применяются

***49. Архитектура и структура команд трёхадресной машины***

Если команды размещать друг за другом в памяти, то можно не указывать в команде адрес следующей. Таким образом получаются трехадресные команды. При их использовании также вводятся специальные команды перехода, которые используются там, где нарушается естественная последовательность команд. Например, при условии if, по которому нужно перейти на другую команду, пропустив несколько. Команда перехода – одноадресная: она состоит всего из двух полей: кода операции и адреса команды, на которую осуществляется переход.

***50. Архитектура и структура команд двухадресной машины***

Трехадресная команда легко расшифровывалась и была удобна в использовании, но с ростом объемов ОЗУ ее длина становилась непомерно большой. Для технической реализации использование трехадресных машин стало неудобным, поэтому появились двухадресные машины, длина команды в которых сокращалась за счет исключения адреса записи результата. В таких ЭВМ результат операции оставался в специальном регистре (**сумматоре**) и был пригоден для использования в последующих вычислениях. В некоторых машинах результат записывался вместо одного из операндов.

***51. Архитектура и структура команд одноадресной машины***

Часто результат выполнения предыдущей команды используется как операнд для следующей. В таких случаях можно использовать одноадресные команды – они будут содержать в себе только КОП и адрес второго операнда. Если пользоваться только одноадресными командами, их можно поделить на группы:

1. арифметической или логической обработки. Первое значение берется с указанного адреса, второе из аккумулятора. С ними совершается действие, соответствующее коду команды, а затем результат записывается в аккумулятор.
2. пересылки. Дают приказ на обмен информации между аккумулятором и памятью (запись в указанную ячейку данных из аккумулятора или наоборот, запись в аккумулятор данных из ячейки памяти)
3. передачи управления. Обеспечивает переход (возможно по некоторому условию) к команде из указанной ячейки памяти (такая команда уже описана в трехадресной машине)
4. ввод-вывод

***52. Классификация устройств памяти систем обработки информации***

**Память** - устройство для хранения информации в виде данных и программ. Память делится прежде всего на внутреннюю (расположенную на системной плате) и внешнюю (размещенную на разнообразных внешних носителях информации).

**Внутренняя память** в свою очередь подразделяется на:

**- ПЗУ** (постоянное запоминающее устройство) или ROM (read only memory), которое содержит - постоянную информацию, сохраняемую даже при отключенном питании, которая служит для тестирования памяти и оборудования компьютера, начальной загрузки ПК при включении. Запись на специальную кассету ПЗУ происходит на заводе фирмы-изготовителя ПК и несет черты его индивидуальности. **Объем** ПЗУ относительно невелик - от 64 Кб до нескольких Мб.

**- ОЗУ** (оперативное запоминающее устройство, ОП — оперативная память) или RAM (random access memory), служит для оперативного хранения программ и данных, сохраняемых только на период работы ПК. Она энергозависима, при отключении питания информация теряется.

**Объем** ОП колеблется в пределах от 64 Кб до 1024 Мб и выше, как правило, ОП имеет модульную структуру и может расширяться за счет добавления новых микросхем.

**Кэш-память** - имеет малое время доступа, служит для временного хранения промежуточных результатов и содержимого наиболее часто используемых ячеек ОП и регистров МП.

**Объем** кэш-памяти зависит от модели ПК и составляет обычно 512 до 1024 Кб.

В состав памяти ЭВМ входят также ЗУ, **принадлежащие отдельным функциональным блокам** компьютера. Формально эти устройства непосредственно не обслуживают основные потоки данных и команд, проходящие через процессор. Их назначение обычно сводится к буферизации данных, извлекаемых из каких-либо устройств и поступающих в них.

Типичным примером такой памяти является **видеопамять** графического адаптера, которая используется в качестве буферной памяти для снижения нагрузки на основную память и системную шину процессора.

Другими примерами таких устройств могут служить **буферная память** контроллеров жестких дисков, а также память, использовавшаяся в каналах (процессорах) ввода-вывода для организации одновременной работы нескольких внешних устройств.

Емкости и быстродействие этих видов памяти зависят от конкретного функционального назначения обслуживаемых ими устройств. Для видеопамяти, например, объем может достигать величин, сравнимых с оперативными ЗУ, а быстродействие – даже превосходить быстродействие последних.

Следующей ступенью памяти, ставшей фактически стандартом для любых ЭВМ, являются **жесткие диски**. В этих ЗУ хранится практически вся информация, которая используется более или менее активно, начиная от операционной системы и основных прикладных программ и кончая редко используемыми пакетами и справочными данными.

Емкость этой ступени памяти, которая может включать в свой состав до десятков дисков, обеспечивая хранение очень большого количества данных, зависит от области применения ЭВМ. Типовая емкость жесткого диска, составляющая на начало 2000-х годов десятки гигабайт, удваивается примерно каждые полтора года.

**Внешняя память.** Устройства внешней памяти весьма разнообразны. Предлагаемая классификация учитывает тип **носителя**, т.е. материального объекта, способного хранить информацию.

**Диски** относятся к носителям информации с прямым доступом, т.е. ПК может обратиться к дорожке, на которой начинается участок с искомой информацией или куда нужно записать новую информацию, непосредственно.

**Магнитные диски (МД)—** в качестве запоминающей среды используются магнитные материалы со специальными свойствами, позволяющими фиксировать два направления намагниченности. Каждому из этих состояний ставятся в соответствие двоичные цифры — 0 и 1. При записи и чтении МД вращается вокруг своей оси, а механизм управления магнитной головкой подводит ее к выбранной для записи или чтения дорожке.

Данные на дисках хранятся в **файлах** — именованных областях внешней памяти, выделенных для хранения массива данных.

«Винчестеры» изготовлены из сплавов алюминия или из керамики и покрыты ферро лаком, вместе с блоком магнитных головок помещены в герметически закрытый корпус. **Емкость** накопителей за счет чрезвычайно плотной записи достигает нескольких гигабайт, быстродействие также выше, чем у съемных дисков (за счет увеличения скорости вращения, т.к. диск жестко закреплен на оси вращения).

**Дисковые массивы RAID** - применяются в машинах-серверах БД и в суперЭВМ, они представляют собой матрицу с резервируемыми независимыми дисками, несколько НЖМД объединены в один логический диск. **Емкость** таких дисков составляет до 5Т6 (терабайт=1012).

**НОД** (накопители на оптических дисках) делятся на:

**не перезаписываемые** лазерно-оптические диски или компакт-диски (CD-ROM). Поставляются фирмой-изготовителем с уже записанной на них информацией. Запись на них возможна в лабораторных условиях лазерным лучом большой мощности. В оптическом дисководе ПК эта дорожка читается лазерным лучом меньшей мощности. Ввиду чрезвычайно плотной записи CD-ROM имеют емкость до 1,5 Гб.

**перезаписываемые** CD-диски имеют возможность записывать информацию прямо с ПК, но для этого необходимо специальное устройство.

***53. Реализация устройств оперативной и долговременной памяти***

Мы уже говорили о том, что память компьютера по характеру доступа к ней и объемам хранящейся в ней информации разделяется на оперативную и долговременную (постоянную). К оперативной памяти центральный процессор компьютера обращается в любой момент, считывание и запись информации в оперативной памяти происходят быстро, в темпе работы компьютера. В долговременную память компьютер записывает большие объемы информации и обращается к ней эпизодически.

         Различие между оперативной и долговременной памятью состоит во времени доступа к памяти, поэтому часто вместо этих названий используют их физическую реализацию - полупроводниковую и магнитную память, однако уже сейчас есть предпосылки создания устройства памяти большой емкости и в то же время с быстрым доступом, малой ценой и размерами.

         Компьютер оперирует двумя символами: “да” (1) и “нет” (0). Состояния “да” и “нет” физически реализуются в электрическом реле, имеющем два устойчивых состояния. На смену реле пришли в свое время электронная лампа, а затем транзистор. Устройство памяти на лампах или транзисторах реализуется в схеме “триггера”, имеющего два устойчивых состояния, следовательно, способного запоминать значения 0 и 1. Для выполнения этой операции используют различные физические принципы. Триггер (trigger означает спусковой механизм, защелку) представляет собой “электронное реле”, которое, как и электрическое реле, может находиться в одном из двух возможных состояний, выражаемых разным напряжением в выбранной точке схемы. Одно напряжение принимается условно за 0, другое за 1. Триггер сколь угодно долго сохраняет одно из двух устойчивых состояний и скачкообразно переключается из одного состояния в другое под действием внешнего сигнала.

         Для запоминания одного бита информации необходим один триггер. Соединяя последовательно несколько триггеров, можно получить устройство для хранения больших двоичных чисел, причем каждый предыдущий триггер будет служить источником сигнала для последующего. Совокупность триггеров, рассчитанную на хранение двоичного числа определенной длины, называют регистром. Следует оговориться, что такое устройство памяти работает только при включенном электропитании.

         Если доступ к ячейкам памяти (триггерам) организован так, что запись и считывание двоичной информации производится одновременно у всех ячеек, устройство памяти называется памятью с произвольным доступом. Если же регистр выполнен так, что информация в нем передается последовательно от предыдущей ячейки к после дующей, он называется регистром сдвига или устройством с последовательной памятью.

         Оперативная память компьютера может состоять из множества триггерных элементов любой природы. В годы существования компьютеров разработаны и технически реализованы принципиально разные устройства оперативной памяти, правда некоторые из них в настоящее время можно встретить только в музеях. Они реализуются на простейших полупроводниковых структурах, на основе криогенных элементов, электронно-лучевых трубок, цилиндрических магнитных доменов, голографии, с помощью сложных молекулярных и биологических систем.

**Сравнение устройств памяти.**

         Длительное время между устройствами оперативной и постоянной памяти по таким основным параметрам, как время доступа в память и емкость памяти, существовал заметный разрыв (по времени доступа от 5·10-3 до 10-3 с, т.е. почти на три порядка). Так, традиционная оперативная память на регистрах сдвига существенно отличалась по времени доступа от памяти на магнитных дисках или барабанах.

         Еще более заметные успехи произошли в решении проблемы увеличения емкости памяти. Особого внимания заслуживает память на оптических дисках, где емкость может измеряться величинами до 6·103 Мбит, а максимальное время доступа в память составляет 10-5 с

***54. Позиционные системы счисления и выполнение арифметических операций***

**Позиционные системы счисления.**

Основные достоинства любой позиционной системы счисления - простота выполнения арифметических операций и ограниченное количество символов (цифр), необходимых для записи любых чисел. Основанием позиционной системы счисления называется возводимое в степень целое число, которое равно количеству цифр, используемых для изображения чисел в данной системе счисления. Основание показывает также, во сколько раз изменяется количественное значение цифры при перемещении ее на соседнюю позицию. Возможно множество позиционных систем, так как за основание системы счисления можно принять любое число не меньшее 2. Наименование системы счисления соответствует ее основанию (десятичная, двоичная, восьмеричная, шестнадцатеричная и т. д.).  
Система называется *позиционной*, если значение каждой цифры (ее вес) изменяется в зависимости от ее положения (позиции) в последовательности цифр, изображающих число.

**Арифметические операции в позиционных системах счисления.**  
   Арифметические операции в двоичной системе счисления. Рассмотрим более подробно арифметические операции в двоичной системе счисления. Арифметика двоичной системы счисления основывается на использовании таблиц сложения, вычитания и умножения цифр. Арифметические операнды располагаются в верхней строке и в первом столбце таблиц, а результаты на пересечении столбцов и строк:  
  
   Рассмотрим подробно каждую операцию.  
   **Сложение.** Таблица двоичного сложения предельно проста. Только в одном случае, когда производится сложение 1+1, происходит перенос в старший разряд.  
   **Сложение двоичных чисел:**  
  
   **Вычитание.** При выполнении операции вычитания всегда из большего по абсолютной величине числа вычитается меньшее и ставится соответствующий знак. В таблице вычитания 1 с чертой означает заем в старшем разряде.  
  
   **Умножение.** Операция умножения выполняется с использованием таблицы умножения по обычной схеме, применяемой в десятичной системе счисления с последовательным умножением множимого на очередную цифру множителя.  
   **Деление.** Операция деления выполняется по алгоритму, подобному алгоритму выполнения операции деления в десятичной системе счисления.

***55. Представление целых и действительных чисел в позиционных системах счисления***

Запись произвольного числа *x* в *P*-ичной позиционной системе счисления основывается на представлении этого числа в виде многочлена

x = anPn + an-1Pn-1 + ... + a1P1 + a0P0 + a-1P-1 + ... + a-mP-m

**Алгоритм перевода целых чисел:**

1. основание новой системы счисления выразить цифрами исходной системы и все последующие действия производить в исходной системе счисления.
2. последовательно выполнять деление данного числа и получаемых целых частных на основание новой системы счисления до тех пор, пока не получим частное, меньше делителя.
3. полученные остатки, являющиеся цифрами в новой системе счисления, привести в соответствие с алфавитом новой системы счисления.
4. составить число в новой системе счисления, записывая его, начиная с последнего остатка.

***Десятичная система.***Основание O = 10, алфавит: 0,1,2,3,4,5,6,7,8,9.  
    Характеризуется тем, что в ней 10 единиц какого-либо разряда образуют единицу следующего старшего разряда. Другими словами, единицы различных разрядов представляют собой различные степени числа 10.  
    В позиционной системе счисления любое вещественное число в развернутой форме может быть представлено в следующем виде:  
   Ao=±(an-1On-1+an-2On-2+a0O0+a-mO-m)  
   где А - само число,  
   O - основание системы счисления,  
   аi - цифры, принадлежащие алфавиту данной системы счисления,  
   n - число целых разрядов числа,  
   m - число дробных разрядов числа.  
   Свернутой формой записи числа называется запись в виде А= an-1 an-2 a9, a-m  
    Иначе свернутую форму записи называют естественной или цифровой. Десятичное число А10=4718.63 в развернутой форме запишется так:  
    А10 = 4\*103 + 7\*102 + 1\*101 + 8\*100 + 6\*10-1 + 3\*10-2  
***Двоичная система счисления.***  
   Основание O = 2, алфавит: 0,1.  
   A2=±(an-12n-1+an-22n-2+a020+a-m2-m)  
   где аi - возможные цифры (0 и 1).  
   Итак, двоичное число представляет собой цепочку из нулей и единиц. При этом оно имеет достаточно большое число разрядов. Быстрый рост числа разрядов - самый существенный недостаток двоичной системы счисления.  
   Записав двоичное число А2=1001,1 в развернутом виде и произведя вычисления, получим это число, выраженное в десятичной системе счисления:  
   А2 = 1\*23+0\*22+0\*21+1\*20+1\*2-1 = 8+1+0,5 = 9,510.  
    ***Восьмеричная система счисления.***  
   Основание O = 8, алфавит: 0,1,2,3,4,5,6,7.  
   Записав восьмеричное число А8=7764,1 в развернутом виде, получим это число, выраженное в десятичной системе счисления:  
   А8 = 7\*83 +7\*82+6\*81+4\*80+1\*8-1 = 3584+448+48+4+0125 = 4084,12510.  
   ***Шестнадцатеричная система счисления***  
   Основание: O = 16. Алфавит:0,1,2,3,4,5,6,7,8,9,А, В,С,D,Е,F.  
   Записав шестнадцатеричное число А16=3AF в развернутом виде и произведя вычисления, получим это число, выраженное в десятичной системе счисления:  
   3AF16 = 3\*162 +10\*161+15\*160 = 768+160+15 = 94310.

***56.Связь между системами счисления с основаниями вида 2K***

***57.Перевод целых и действительных чисел из одной позиционной системы в другую***

***58. Прямой, обратный и дополнительный коды***

***59. Нормализованное представление чисел, переполнение, модифицированные коды***

***60. Алгоритмы сложения и вычитания в дополнительных и обратных кодах***

***61. Алгоритмы умножения в дополнительных и обратных кодах***

***62.Структуры рекурсивных алгоритмов***

***63.Применение рекурсивных алгоритмов***

***64.Оценка сложности рекурсивных алгоритмов***