1. История создания ЭВМ

**1642** г. - Б. Паскаль - механическая счетная машина - сложение и вычитание 6-8 разрядных чисел.

**1670** г. - Г. В. Лейбниц - первый арифмометр, выполняющий все четыре арифметических действия.

**1834** г. - Ч. Бэббедж - предложил идею универсальной вычислительной машины с программным управлением. Её структура совпадала по существу со структурой современных ЭВМ. Программы для аналитической машины писала Ада Ловлейс (первый известный программист). На самом деле машина не была реализована в то время из-за технических и финансовых сложностей. Мир отставал от хода мыслей Бэббиджа.

**1945** г. была создана первая ЭВМ в США(около 3000 умножений в сек).

**В начале 70-х годов** - первые персональные ЭВМ (ПЭВМ).

**Компьютер** - это техническое средство преобразования информации, в основу работы которого заложены те же принципы обработки электрических сигналов, что и в любом электронном устройстве:

* входная информация преобразуется в электрический сигнал; сигналы обрабатываются в блоке обработки;
* с помощью преобразователя выходных сигналов обработанные сигналы преобразуются в неэлектрические

сигналы (изображение на экране).

2.**Поколения компьютеров**

**Первое** поколение. Компьютеры на электронных лампах (194x-1955)

Быстродействие: Несколько десятков тысяч операций в секунду.

Особенности:

* Поскольку лампы имеют существенные размеры и их тысячи, то машины имели огромные размеры.
* Поскольку ламп много и они имеют свойство перегорать, то часто компьютер простаивал из-за поиска и замены вышедшей из строя лампы.
* Лампы выделяют большое количество тепла, следовательно, вычислительные машину требуют специальные мощные охладительные системы.

*Примеры* компьютеров:

**Колоссус** - секретная разработка британского правительства (в разработке принимал участие Алан Тьюринг). Это первый в мире электронный компьютер. Не оказал влияние на развитие комп. техники из-за своей секретности.

**Эниак**. Вес машины 30 тонн. Минусы: использование десятичной системы счисления; множество переключателей и кабелей.

**Эдсак**. Достижение: первая машина с программой в памяти.

**Whirlwind I**. Слова малой длины, работа в реальном времени.

**Компьютер 701** (и последующие модели) фирмы IBM. Первый компьютер, лидирующий на рынке в течение 10 лет.

**Второе** поколение. Компьютеры на транзисторах (1955-1965)

Быстродействие: Cотни тысяч операций в секунду.

Использование транзисторов позволило уменьшить размеры вычислительной техники, повысить надежность, увеличить скорость работы (до 1 млн. операций в секунду) и почти свести на нет теплоотдачу. Развиваются способы хранения информации: широко используется магнитная лента, позже появляются диски. В этот период была замечена первая компьютерная игра.

DEC выпускает первый миникомпьютер (размером со шкаф). Зафиксировано появление дисплея.

Фирма IBM также активно трудится, производя уже транзисторные версии своих компьютеров.

**Компьютер 6600** фирмы CDC, имел преимущество над другими компьютерами того времени - это его быстродействие, которое достигалось за счет параллельного выполнения команд.

**Третье** поколение. Компьютеры на интегральных схемах (1965-1980)

Быстродействие: Миллионы операций в секунду.

Интегральная схема представляет собой электронную схему, вытравленную на кремниевом кристалле. На такой схеме умещаются тысячи транзисторов. Следовательно, компьютеры этого поколения были вынуждены стать ещё мельче,

быстрее и дешевле. Последнее свойство позволяло компьютерам проникать в различные сферы деятельности человека. Из-за этого они становились более специализированными.

Появилась проблема совместимости выпускаемых моделей (программного обеспечения под них). Впервые большое внимание совместимости уделила кампания IBM.

Было реализовано мультипрограммирование (когда в памяти находится несколько выполняемых программ, что дает эффект экономии ресурсов процессора).

Дальнейшее развитие миникомпьютеров (PDP-11).

**Четвертое** поколение. Компьютеры на больших (и сверхбольших) интегральных схемах (1980-...)

Быстродействие: Cотни миллионов операций в секунду.

Появилась возможность размещать на одном кристалле не одну интегральную схему, а тысячи. Компьютеры

продолжали дешеветь и теперь их покупали даже отдельные личности, что ознаменовало так называемую эру персональных компьютеров. Но отдельная личность чаще всего не была профессиональным программистом. Следовательно, потребовалось развитие программного обеспечения, чтобы личность могла использовать компьютер в соответствие со своей фантазией.

В конце 70-х - начале 80-х популярностью пользовался компьютера Apple, разработанный Стивом Джобсом и Стивом Возняком. Позднее в массовое производство был запущен персональный компьютер IBM на процессоре Intel. Позднее появились суперскалярные процессоры, способные выполнять множество команд одновременно, а также 64-битные компьютеры.

**Пятое** поколение?

Сюда относят неудавшийся проект Японии. К пятому поколению вычислительных машин относят так называемые

невидимые компьютеры или карманные компьютеры. Также существует мнение, что к пятому поколению следует относить компьютеры с двухъядерными процессорами. С этой точки зрения пятое поколение началось примерно с 2005 года.

**3.Архитектура фон-Неймана. Принципы**

АФ-Н — широко известный принцип совместного хранения команд и данных в памяти компьютера. Вычислительные системы такого рода часто обозначают термином «машина фон Неймана». В общем случае, когда говорят об АФ-Н, подразумевают принцип хранения данных и инструкций в одной памяти.

Компьютер, согласно принципам фон Неймана, должен иметь следующие устройства:

* арифметико-логическое устройство, выполняющее арифметические и логические операции;
* устройство управления, которое организует процесс выполнения программ;
* запоминающее устройство, или память для хранения программы и данных;
* внешнее устройство для ввода вывода информации.

Принципы фон Неймана:

* Использование двоичной системы счисления в вычислительных машинах.( **Принцип двоичного кодирования**)

Преимущество перед десятичной системой счисления(СС) заключается в том, что устройство можно делать достаточно простыми, арифметические и логические операции в двоичной СС также выполняются достаточно просто.

* Программное управление ЭВМ. (**Принцип программного управления**)

Работа ЭВМ контролируется программой, состоящей из набора команд. Команды выполняются последовательно друг за другом. Созданием машины с хранимой в памяти программой было положено начало программированию.

* Память компьютера исп-ся не только для хранения данных, но и программ.(**Принцип однородности памяти).**

При этом и команды, и программы, и данные кодируются в двоичной системе, т. е. их способ записи одинаков. Поэтому в определенных ситуациях над командами можно выполнять те же действия, что и над данными. Принцип однородности памяти.

* Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.( **Принцип адресности**)

В любой момент можно обратиться к любой ячейке памяти по её адресу. Этот принцип открыл возможность использовать переменные в программировании.

* Возможность условного перехода в процессе выполнения программ.

Не смотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода. Главное следствие: программа уже не была постоянной частью машины (ф.е. калькулятора). Программу стало возможно легко изменить. Но аппаратура осталась неизменной и очень простой.

**4.Языки программирования**

Для представления алгоритма в виде, понятном компьютеру, служат языки прогр-я - это искусственные языки.

Совокупность требований для записи команд образуют синтаксис языка, а смысл каждой команды - семантику языка.

Нарушение формы записи программы приводят к синтаксической ошибке, а правильно написанная программа, но не отвечающая алгоритму, приводит к семантической (логической) ошибке.

Процесс поиска ошибок в программе называют тестированием программы, процесс устранения ошибок – отладкой.

**Классификация** языков программирования:

* **По степени абстракции от архитектуры компьютера:**

1. языки программирования низкого уровня - машинный язык (язык машинных кодов). Программист должен полностью владеть архитектурой ЭВМ; (Автокод). Все языки низкого уровня ориентированы на определенный тип компьютера, т. е. являются машинно–зависимыми. Машинно–ориентированные языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ.
2. языки программирования среднего уровня - язык мнемонических кодов (язык ассемблера - символьная форма машинного языка с возможностями языков высокого уровня). Используя такие языки, необходимо владеть архитектурой ЭВМ, a также иметь специальную программу-переводчика инструкции пользовательской программы на язык машинных кодов, называемую **транслятор**;
3. языки высокого уровня, или алгоритмические языки (Паскаль, Алгол, Фортран, Си). Алгоритмические языки имеют свои алфавит и синтаксис, а также трансляторы со встроенной средой программирования. Инструкции программы на алгоритмическом языке записываются в виде операторов. Эти языки машинно–независимы. Однако программы на языках высокого уровня занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

* **По структурному строению программ**:

1. процедурно-ориентированные языки (Pascal, C, и др.) - используется метод разбивки всей задачи на более простые подзадачи (процедуры или функции) и их независимая реализация. Достоинство - создание достаточно сложных программ.
2. объектно-ориентированные языки (C++, Java, и др.) - из области решаемой задачи выделяются классы, объединяющие связанные каким-либо признаком данные + их обработка. Далее создаются объекты данных классов, которые, взаимодействуя друг с другом, осуществляют решение поставленной задачи. Первый объектно-ориентированный язык программирования **Simula** был создан в 1960-х годах

**5.Размещение данных и программ в памяти ПЭВМ.**

Внутренняя память - это устройство, которое хранит информацию, необходимую компьютеру в данный момент работы.

Состав ВП: оперативная память, кэш-память и постоянная (специальная) память.

Св-ва ВП:

* адресуемость
* дискретность

Оперативная память (ОЗУ, англ. RAM - Random Access Memory ) - это энергозависимое быстрое запоминающее устройство сравнительно небольшого объёма, непосредственно связанное с процессором и хранением выполняемых программ. При выключении компа ОЗУ очищается.

**Размещение данных и программ в памяти ПЭВМ**

Данные и программы во время работы ПЭВМ размещаются в оперативной памяти, которая представляет собой последовательность пронумерованных ячеек. По указанному номеру процессор находит нужную ячейку, поэтому номер ячейки называется ее адресом(адресуемость). Минимальная адресованная ячейка состоит из 8 двоичных позиций, т. е. в каждую позицию могут быть записаны либо 0, либо 1. Объем информации, который помещается в одну двоичную позицию, называется битом(дискретность). **8 бит = 1 байт.**

6.Программные модули. Транслятор, компилятор, интерпретатор.

**Программа** – последовательность команд, которые помещаются в памяти и выполняются процессором в указанном порядке.

Программист пишет программу на языке высокого уровня. Исходный текст программы, введенный с помощью клавиатуры в память компьютера, - исходный модуль (source code, в языке Си имеет расширение \*.cpp).

Для перевода программы с языка прогр-я на язык машинных кодов используют специальные программы – трансляторы. Существует **три вида** транслятора:

* интерпретаторы (это транслятор, который производит пооператорную обработку и выполнение исходного кода программы; при использовании интерпретатора отсутствует разделение на две стадии - перевод и

выполнение).

* компиляторы (преобразует всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и потом исполняется; перевод проги с высокого уровня на более низкий).
* ассемблеры (переводят программу, записанную на языке ассемблера, в программу на машинном языке).

Исполняемый (абсолютный, загрузочный) модуль создает вторая специальная программа - “компоновщик”. Ее еще называют редактором связей (Linker). Она и создает модуль, пригодный для выполнения на основе одного или нескольких объектных модулей.

Загрузочный модуль (Load module, расширение \*.exe) - это программный модуль, представляемый в форме, пригодный для загрузки его в память и выполнения.

7.Этапы решения задач на ЭВМ:

|  |  |
| --- | --- |
| **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. |

8.Алгоритм. Способы записи алгоритма

Алгоритм – система последовательных шагов, выполнение которых приводит к решению задачи.

Теория алгоритмов (основатели: Алан Тьюринг и Эмиль Пост) – наука, изучающая общие свойства и закономерности алгоритмов и модели их представления.

Алгоритмизация – процесс подготовки решения задачи на ЭВМ.

Запись алгоритма на формальном языке – **программа**.

**Классы алгоритмов**:

1. Вычислительные – исходные данные простые и их немного; процесс вычислений долгий и сложный. *Матем. и физич. задачи.*

2. Информационные – не очень сложные вычисления, большой объем данных. Требуют организации данных. *Эконом. задачи.*

3.Управляющие – исходные данные поступают от процесса или объекта, которым управляют. Результат – воздействие на процесс. Автоматизированное управление технолог. процессами (участие человека на определенных этапах; автоматическое – без участия).

4.Реального времени – результат должен быть получен к определенному времени.

|  |  |
| --- | --- |
| **Способы записи алгоритма:**  • словесно-формульный;  • структурный или блок-схемный;  • с помощью граф-схем;  • с помощью сетей Петри. | **Формы записи алгоритмов:**   * На естественном языке (словесная); * На языке формул; * На спец.языке (f.e.на языке прогр-я); * В граф.форме (блок-схемы) |

Графическое описание алгоритма

Представление алгоритма в виде схемы, состоящей из последовательности блоков (гeометрических фигур), каждый из которых отображает содержание очередного шага алгоритма, называется графическим описанием алгоритма.

Внутри фигур кратко записывают действие, выполняемое в этом блоке. Такую схему называют блок-схемой алгоритма.

Правила изображения фигур сведены в единой документации (ГОСТ 19.701-90). По данному ГОСТу - это схема данных.

Схема данных содержит:

* символы данных;
* символы процесса, который нужно выполнить над данными;
* символы линий или стрелок;
* специальные символы, которые используются для удобства чтения схемы.

Кроме словесного (на естественном языке) и графического описания алгоритма также иногда используют:

* псевдокод
* алгоритмический язык - запись на конкретном формальном языке (программа)

Псевдокод - представляет собой систему обозначений правил, предназначенную для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественными и формальными языками.

9.Алгоритм. Свойства. Принципы построения. Пример

Алгоритм – система последовательных шагов, выполнение которых приводит к решению задачи.

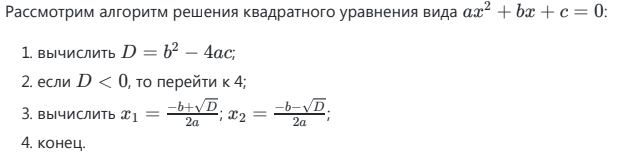
(вопрос 8)

Св-ва из оит мне лень (детерминированность, дискретность, конечность, массовость, направленность алгоритма, элементарность шагов программирования)

Принципы:

* Принцип декомпозиции и синтеза. Первое предполагает первоначальную разработку алгоритма в виде укрепленных блоков и их постепенную детализацию. Затем проводится сборка блоков единую программу.
* Принцип "от главного к второстепенному" предполагает составление алгоритма начиная с главной конструкции.
* Принцип структурирования, т. е. использования только типовых алгоритмических структур (следования, логическая и циклическая).

**Пример**:



10.Правила применения символов и выполнения блок-схем. Описание символов.

**Правила применения символов:**

• Символ предназначен для графической идентификации функции, которую он отображает.

• Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.

• Большинство символов задумано так, чтобы дать возможность включения текста внутри символа.

• Минимальное количество текста, необходимого для понимания функции данного символа, следует помещать внутри данного символа, символа комментария либо на отдельном листе и давать перекрестную ссылку на символ. Текст для чтения должен записываться слева направо и сверху вниз независимо от направления потока.

• В схемах может использоваться идентификатор символов. Идентификатор символа должен располагаться слева над символом.

• В схемах может использоваться описание символов.

**Правила выполнения схем:**

* Операции обработки данных и носители информации изображаются на схеме соответствующими блоками. Большая часть блоков по построению условно вписана в прямоугольник со сторонами а и b. Минимальное значение а = 10 мм, увеличение а производится на число, кратное 5 мм. Размер b=1,5a. Для отдельных блоков допускается соотношение 1:2 между а и b.
* Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводиться параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может входить несколько линий. Из блока (кроме логического) может выходить только одна линия. Логический блок может иметь в качестве продолжения один из двух блоков, и из него выходят две линии. Если на схеме имеет место слияние линий, то место пересечения выделяется точкой.
* Схему алгоритма следует выполнять как единое целое, однако в случае необходимости допускается обрывать линии, соединяющие блоки. Если при обрыве линии продолжение схемы находится на этом же листе, то на одном и другом конце линии изображается специальный символ соединитель — окружность диаметром 0,5 а. Внутри парных окружностей указывается один и тот же идентификатор.

Если схема занимает более одного листа, то в случае разрыва линии вместо окружности используется межстраничный соединитель. Внутри каждого, соединителя указывается адрес — откуда и куда направлена соединительная линия. Адрес записывается в две строки: в первой указывается номер листа, во второй — порядковый номер блока.

Блок-схема должна содержать все разветвления, циклы и обращения к подпрограммам, содержащиеся в программе.

**Описание символов**

|  |  |
| --- | --- |
| Символы данных:  • Данные  • Запоминаемые данные  • Оперативное запоминающее устройство  • Запоминающее устройство с прямым или последовательным доступом  • Документ  • Ручной ввод  • Карта  • Бумажная лента  • Дисплей  Символы процесса:  Указывают на процесс, который следует выполнить над данными:  • Процесс  • Предопределенный процесс  • Ручная операция  • Подготовка  • Решение  • Параллельные действия  • Граница цикла | Символы линий:  Указывают потоки данных между процессами и (или) носителями данных;  • Линия  • Передача управления  • Канал связи  • Пунктирная линия  Специальные символы:  Используются для облегчения написания и чтения схемы.  • Соединитель  • Терминатор  • Комментарий  • Пропуск  (+рисунки) |

11. Типовые алгоритмические структуры.

* **следование**
* **ветвление**
* **повторение**

Используя типовые алгоритмические структуры можно собрать алгоритм любой сложности.

**Ветвление**

Команда ветвления - составная команда алгоритма, в которой в зависимости от условия Р выполняется или одно S1, или другое S2 действие. Из команд следования и команд ветвления составляются алгоритмы ветвления. Примером разветвляющегося алгоритма будет нахождение большего из двух чисел, введенных клавиатуры. Команда ветвления может быть полной и неполной формы. Неполная форма команды ветвления используется тогда, когда необходимо выполнять действие S только в случае соблюдения условия P. Если условие Р не соблюдается, то команда ветвления завершает свою работу без выполнения действия. Примером команды ветвления неполной формы будет уменьшение в два раза только четного числа.

**Повторение**

Команда повторения - составная команда алгоритма, в которой в зависимости от условия Р возможно многократное

выполнение действия S. Из команд следования и команд повторения составляются циклические алгоритмы.

Пример циклического алгоритма может быть следующий. Пока с клавиатуры вводятся положительные числа, алгоритм

выполняет нахождение их суммы. С предусловием.

В команде повторения с постусловием вначале выполняется действие S и лишь затем проверяется условие Р. Разновидностью команды повторения с предусловием является команда повторения с параметром. Она используется тогда, когда известно количество повторений действия. Примером циклического алгоритма с параметром будет нахождение суммы первых 20 натуральных чисел.

12.Синтаксис языка СИ. Алфавит. Лексемы

* В языках программирования вместо номеров ячейкам памяти принято давать имена (идентификаторы), а содержимое ячеек называть переменными, или константами, в зависимости оттого, изменяется значение в процессе или нет.
* В языке Си фундаментальным понятием является инструкция (оператор, операция и функция), которая представляет собой описание определенного набора действий. Таким образом, программа на языке Си состоит и последовательности инструкций.

В любом языке присутствуют 4 основных элемента: символы, слова, словосочетания и предложения. Подобные элементы содержит и язык СИ.

• Слова наз. “лексемами” - элементарными конструкциями.

• Словосочетания -это выражения.

• Предложения - это операторы.

Лексемы образуются из символов. Выражения - из лексем и символов. Операторы - из лексем, символов и выражений.

Алфавит языка - это его символы.

Каждый элемент языка определяется **синтаксисом и семантикой**. Синтаксические определения устанавливают правила построения элементов языка, а семантика определяет их смысл и правила использования.

**Алфавит языка Си включает:**

- прописные и строчные буквы латинского алфавита, а также знак подчеркивания (код ASCII 95);

- арабские цифры от 0 до 9;

- специальные символы:

|  |  |  |
| --- | --- | --- |
| +(плюс)  –(минус)  \*(звездочка)  /(дробная черта)  =(равно)  >(больше)  < (меньше) | ;(точка с запятой)  & (амперсант)  [ ](квадратные скобки)  { }(фигурные скобки)  ()(круглые скобки)  \_(знак подчеркивания)  .(точка) | ,(запятая)  :(двоеточие)  #(решетка)  %(процент)  ~(поразрядное отрицание)  ?(знак вопроса)  !(восклицательный знак)  \(обратный слеш). |

- пробельные (разделительные) символы:

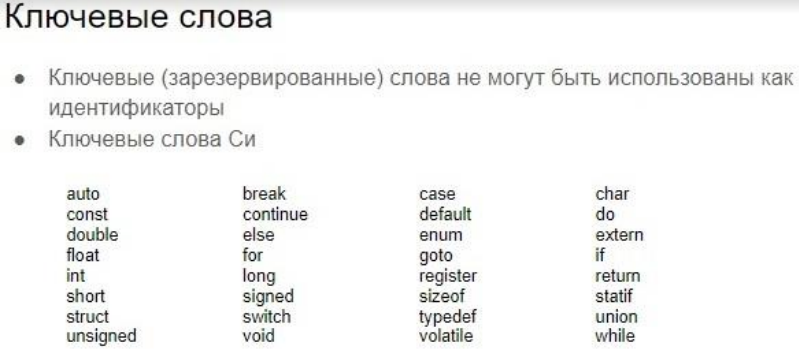
пробел, символы табуляции, перевода строки, возврата каретки, новая страница и новая строка.

Из символов алфавита формируются **лексемы** языка – минимальные значимые единицы текста в программе:

* идентификаторы;
* ключевые (зарезервированные) слова;
* знаки операций;
* константы;
* разделители (скобки, точка, запятая, пробельные символы).

13. Идентификаторы. Ключевые слова.

**Идентификатор(ID)** — это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы имеют значение, например, sigma и SIGMA – различные объекты. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Пробелы внутри имен не допускаются. Длина идентификатора по стандарту не ограничена, но некоторые компиляторы и компоновщики налагают на нее ограничения. Идентификатор должен нести смысл, поясняя назначение объекта в программе. Если ИД состоит из нескольких слов, то принято разделять их знаком подчеркивания или писать каждое слово с большой буквы (birth\_date/birthDate). Идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка.

— это идентификаторы, которые имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Например: for, if, do, int, double, while, auto. ****

14.Литералы (константы). Комментарии. Директивы.

Константы — это данные, не изменяющие своего значения в ходе выполнения программы. В языке СИ можно выделить четыре вида констант: целые, с плавающей запятой, символьные и строковые.

Литерал является неадресуемой величиной, нет никакого способа узнать его адрес.

Комментарии.

Не является лексемой. Внутри можно использовать абсолютно любые доступные на ПК символы, т.к. компилятор игнорирует комментарии.

В Си комментарии ограничиваются парами символов /\* и \*/, а в Си++ был введен вариант комментария //, заканчивающийся переходом на новую строку.

Директивы.

Перед компиляцией программа на языке Си обрабатывается спец.программой, называемой препроцессор. Препроцессор работает под управлением директив.

* Препроцессор директривы начинается с символа #, за которым следует наименование директивы.
* Препроцессор решает ряд задач по предварительной обработке программы, основной из которых является обработка заголовочных файлов (обычных текстов) с декларацией стандартных библиотечных Функций.

Формат: **#include<ID\_файла.h>**

* Если идентификатор файла заключен в <>, то поиск происходит в стандартной директории, если в «», то в текущей директории.

К часто используемым библиотекам относят:

* Stdio.h- содержит стандартные функции файлового ввода-вывода;
* Conio.h-функции для работы с консолью;
* Math.h- математические функции.

Также препроцессор обрабатывает макроопределения.

Макроподстановка define имеет вид: **#dеfine <ID> <cтрока>,** #define PI 3.14

В ходе препроцессорной обработки программы в тексте ПИ везде заменится на 3.14.

15. Структура программы на языке СИ.

Программа на языке Cи состоит из функций, описаний и директив препроцессора.

Директива: #include(включения файла в прогу) #define (определения; заменяет имя константы на то, что следует за ней).

|  |  |
| --- | --- |
| Часто используемые библиотеки:  <stdio.h>- стандартные функции ввода/вывода.  <conio.h>-функции для работы с консолью.  <graphics.h>-содержит прототипы графических функций.  <math.h>-прототипы математических функций. | Программа может состоять из нескольких модулей.  Начало проги - { конец - }  объявления переменных и массивов: int x=1; char str[N];  операторы программы:  puts("Введите имя"); gets(str); printf(«,,», пременные) |

Одна из функций должна иметь имя main(). Выполнение программы начинается с первого оператора этой функции.

Простейшее определение функции: **тип\_возвращаемого\_значения имя (параметры) {операторы, составляющие тело функции}**

Как правило, функция используется для вычисления какого-либо значения, поэтому перед именем функции указывается его тип. Если функция не должна возвращать значение, указывается тип void.

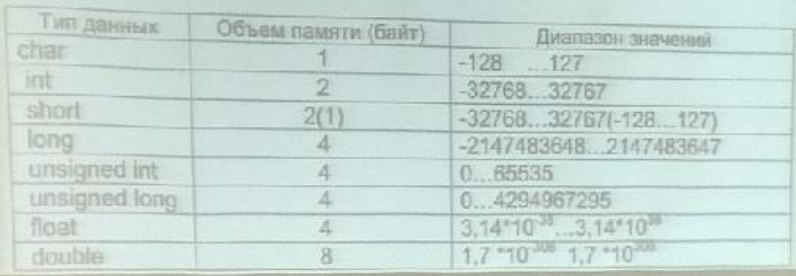
Функции не могут быть вложенными. Тело функции является блоком и заключается в фигурные скобки.

16. Основные типы данных.

* Данные в языке Си делятся на две категории: простые (скалярные, базовые) и сложные (составные) типы данных.
* Основные типы базовых данных: стандартный целый (int), вещественный с одинарной точностью (float), символьный (char).

|  |  |
| --- | --- |
| Данные целого типа:   * Short * Long * Unsigned | Сложные типы:   * Массивы * Структуры * Объединения перечислений |

Данные целого и вещественного типа находятся в определенных цифровых пределах, т.к. занимают разный размер ОЗУ.



17. Целый тип. Внутреннее представление.

Размер типа int не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора отводится 2 байта, для 32-разрядного — 4 байта.

Спецификатор short перед именем типа указывает компилятору, что под число требуется отвести 2 байта независимо от разрядности процессора. Спецификатор long означает, что целая величина будет занимать 4 байта. Таким образом, на 16-разрядном компьютере эквиваленты int и short int, а на 32-разрядном — int и long int.

Внутреннее представление величины целого типа — целое число в двоичном коде. При использовании спецификатора signed старший бит числа интерпретируется как знаковый (0 — положительное число, 1 — отрицательное). Спецификатор unsigned позволяет представлять только положительные числа, поскольку старший разряд рассматривается как часть кода числа. Таким образом, диапазон значений типа int зависит от спецификаторов. По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор signed можно опускать.

18. Типы с плавающей (фиксированной) точкой. Внутреннее представление.

В СИ существуют два типа данных с плавающей точкой: float и double. Типы данных с плавающей точкой предназначены для хранения чисел с плавающей точкой. Типы данных float и double могут хранить как положительные, так и отрицательные числа с плавающей точкой. У типа данных float размер занимаемой памяти в два раза меньше, чем у типа данных double, а значит и диапазон принимаемых значений тоже меньше. Если тип данных float объявить с приставкой long, то диапазон принимаемых значений станет равен диапазону принимаемых значений типа данных double. В основном, типы данных с плавающей точкой нужны для решения задач с высокой точностью вычислений.

Внутреннее представление вещественного числа состоит из 2 частей: мантиссы и порядка. Величины типа float занимают 4 байта, из которых один разряд отводится под знак мантиссы, 8 разрядов под порядок и 24 – под мантиссу.

Величины типы double занимают 8 байтов, под порядок и мантиссу отводятся 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка его диапазон.

19.Объявление объектов.

Объекты программы должны быть явно декларированы по формату: <атрибуты><список ИД объектов>

Две формы:

* Описание(без выделения памяти);
* Определение (будет выделен объем памяти в соответствии с его типом).

Объекту можно сразу задать начальное значение. Элементы списка разделяются запятыми, а атрибуты разделителями.

Объекты программы в общем случае имеют след.атрибуты:

**<класс памяти>** - хар-ка способа размещения объекта в памяти (статическая, динамическая), определяет область видимости и время хранения переменной (по умолчанию auto).

**<тип**> - хар-ка механизма интерпретации данных (по умолчанию int)

Класс памяти и тип – атрибуты необязательные. При их отсутствии их значения устанавливаются по умолчанию.

Пример декларации:

Int I,j,k; Char r; Double sd;

20. Преобразование типов. Приведение типов.

**Данные целого типа**

**Int** – целое число, соответствующее естественному размеру целых чисел в ЭВМ. С этим типом можно использовать квалификаторы short и long. Они определяют размер памяти, выделенной под переменную.

Примеры: short int x; long int x; unsigned int x=8;

Атрибуты signed и unsigned показывают, как интерпретируется старший бит числа: как знак или как часть числа.

**Данные символьного типа**

Символьная переменная занимает 1 байт (8 бит), значит представляется кодом от 0 до 255 (по кодовым таблицам). Данные типа char рассматриваются как целые, поэтому есть возможность использовать signed char (-127… 127) и unsigned char (0 … 255). Пример: **Char r; Char let = ‘s’;**

**Данные вещественного типа** в памяти занимают флоат – 4 байта, дабл – 8 байт, лонг дабл – 10 байт.

Для размещения данных типа флоат 8 бит – для порядка, 24 бита – для мантиссы (точность).

|  |  |  |
| --- | --- | --- |
| Тип | Точность (мантисса) | Порядок |
| флоат | 7 цифр после запятой | +/-38 |
| Дабл | 15 | +/- 308 |
| Лонг дабл | 19 | +/- 4932 |

Преобразование типов:

Любые операнды типа char, unsigned char или short преобразуются к типу int по правилам:

1)char расширяется нулем или знаком в зависимости от умолчания для char;

2)unsigned char расширяется нулем;

3)signed char расширяется знаком;

4)short, unsigned short и enum при преобразовании не изменяются.

Затем любые два операнда становятся либо int, либо float, double или long double.

При вычислении выражений операнды преобразуются к типу того операнда, который имеет наибольший размер.

21.Константы в программах (целочисленные, вещественные, символьные, строковые)

Константы - объекты, не подлежащие использованию в левой части оператора присваивания. Неадресуемая величина. Хранится в памяти, но адреса нет. Константы представляют фиксированную величину, которая не может быть изменена в программе.

* Целочисленные константы
* Константы вещественного типа
* Символьные константы
* Строковые константы

Целочисленные константы:

Общий вид: +/-n (плюс обычно не ставится)

* Десятичные – последовательность цифр 0…9, первая их которых не 0. Если необходимо указать длинную целую константу, то указывается признак L(l) – 273L (273l). Для такой константы будет отведено 4 байта. Обычная целая константа, которая слишком длинна для инт, рассматривается как лонг.
* Восьмеричные – последовательность цифр 0…7, первая из которых должна быть 0.
* Шеснадцатиричные – последовательность цифр 0…9 и букв A…F (a…f), начинающихся символами ОХ (Ох).

8 и 16-ые могут так же оканчиваться L(l).

Константы вещественного типа:

Размещаются в памяти по формату дабл, а во внешнем представлении могут иметь две формы:

* С фикс. десятичной точкой; +/- n.m, где это целая и дробная части.
* С плав. дес. точкой; +/-n.mE+/-p, где р – порядок.

Символьные константы:

- это символ, заключенный в одинарные кавычки. Занимает 1 байт.

Так же используются специальные последовательности символов:

\n – новая строка;

\t – горизонтальная табуляция

\o – нулевой символ.

Текстовые символы вводятся непосредственно с клавиатуры, а спец. и управляющие символы представляются в исходном тексте парами текстовых символов (\\,\»,\’)

Строковые константы:

- последовательность символов кода ASCII, заключенная в кавычки. К строковым константам прибавляется нуль-терминатор (\0), описывающий конец строки.

22.Операции и выражения. Унарные, бинарные и тернарные операции. Приоритет в выражениях. Приоритет операций.

**Операция** — арифметическое или логическое действие.

Операции языка Си предназначены для управления данными. Для их использования необходимо знать:

* Синтаксис
* Приоритеты
* Порядок выполнения

Выражения состоят из операторов, операций, скобок и используются для вычисления некоторого значения определенного типа. Каждый операнд, в свою очередь, может быть выражением или одним из его частных случаев.

Операции, применяемые к одному операнду, - унарные, к двум операндам – бинарные, с тремя операндами - тернарные. Операции выполняются в соответствии с приоритетами. Высший приоритет у унарных операций, затем выполняются бинарные, потом логические и тернарные. Для изменения порядка выполнения операций используются круглые скобки.

Большинство операций выполняются слева направо. Исключение: унарные операции, операции присваивания и тернарная операция ( ? : ) - справа налево.

Приоритет Операции:

1 ++,--,~,!

2 \*,/,%

3 +,-

4 >>,>>>,<<

5 >,>=,<,<=

6 ==,!=

7 &

8 ^

9 |

10 &&

11 ||

23.Операции и выражения. Арифметические операции. Операция присваивания и ее сокращенная запись.

**Операция** — арифметическое или логическое действие. Выражения состоят из операторов, операций, скобок и используются для вычисления некоторого значения определенного типа.

Про операции из вопр.20.

|  |  |
| --- | --- |
| Арифметические операции:  Бинарные:  + сложение  - вычитание (либо унарная операция – изменение знака)  / - деление (для инт – с отбрасыванием остатка)  \* - умножение  % - деление по модулю | Порядок выполнения операций:   1. Выражения в круглых скобках 2. Функции 3. Слева направо умножение и деление 4. Слева направо + и –   Унарные операции + и – обладают самым большим авторитетом. Определены только для целых и вещ-ых операндов. |

Операция присваивания: Формат: **<ID>=<выражение>;**

Примеры НЕдопустимых выражений:

1. Присваивание константе: 2=х+у;
2. Присваивание функции: getch()=i;
3. Присваивание результату операции: (х+1)=2+у;

Сокращенная запись ОП: **Два вида**:

* Вместо записи v=v@e, где @ - арифметическая операция либо операция над битовым представлением операндов, рекомендуется использовать запись: v@=e. **Например, i=i+2 -> i+=2**
* Вместо записи x=x#1, где # - символ + или -, обозначающий операцию инкремента или декремента, х –целочисленная переменная, переменная – указатель, рекомендуется использовать запись ##x(префиксная) и x##(постфиксная).

24. Операции и выражения. Операции сравнения. Логические операции. Операция запятая.

Операция — арифметическое или логическое действие. Выражения состоят из операторов, операций, скобок и используются для вычисления некоторого значения определенного типа.

Про операции из вопр.20.

Операции сравнения:

== равно или эквивалентно

!= не равно

> < <= >=

Пары символов разделять нельзя. Вид: **<выражение1> <знак\_операции> <выражение2>**

Правила:

1. Операндами могут быть любые базовые типы.
2. Значения операндов перед сравнением преобразуются к одному типу.

|  |  |
| --- | --- |
| Логические операции:  ! – отрицание (логическое НЕТ)  && - конъюнкция (логическое И) нельзя разделять  || - дизъюнкция (логическое ИЛИ)  Общий вид операции отрицания:  **!<выражение>**  Общий вид операции конъюнкции и дизъюнкции: <**выражение1> <знак\_операции> <выражение2>**  Y>0&&x=7 – истина, если оба истинны.  y>0||x=7 – истина, если хотя бы одно истинно.  Ненулевое значение операнда трактуется как “истина”, а нулевое - “ложь”:  !0 🡪 1 !5🡪 0  Особенность операции конъюнкции и дизъюнкции - экономное последовательное вычисление выражений-операндов: **<выражение1> <операция> <выражение2>**   * если выражение1 операции конъюнкция ложно, то результат операции - ноль и выражение2 не вычисляется. * если выражение1 операции дизъюнкции истинно, то результат операции единица и выражение2 не вычисляется. | Операция , (запятая)  выражение1, ..., выражениеN  выражение1, ..., выражениеN вычисляется гарантированно последовательно и результатом операции становится значение выражения N.  Пример:  m = (i = 1, j = i++, k = 6, n = i + j + k)  получим последовательность вычислений i = 1, j = i = 1, i = 2, k = 6, n = 2 + 1 + 6 и в результате m = n = 9. Данный пример ничем не отличается от такого участка кода:  i = 1;  j = i;  i++;  k = 6;  n = i + j + k;  m = n; |

25.Операции и выражения. Побитовые логические операции. Операции над битами.

Операция — арифметическое или логическое действие. Выражения состоят из операторов, операций, скобок и используются для вычисления некоторого значения определенного типа. Про операции из вопр.20.

Перечень операций над битами и их обозначения:

~ дополнение (унарная операция), инвертирование (одноместная операция)

& побитовое И - конъюнкция

│ побитовое включающие ИЛИ - дизъюнкция

^ побитовое исключающие ИЛИ - сложение по модулю 2

>> сдвиг вправо << сдвиг влево нельзя разделять

Общий вид операции инвертирования: **~ <выражение>**

Остальные операции над битами имеют вид**: <выражение1> <знак\_операции> <выражение2>**

Операция ( | ) используется для включения битов w = x | y , устанавливает в единицу те биты в x , которые = 1 в y.

Необходимо отличать побитовые операции & и | от логических операций && и || , которые подразумевают вычисление значение истинности слева направо.

Операции сдвига << и >> осуществляют соответственно сдвиг вправо влево своего левого операнда, на число битовых позиций, задаваемых правым операндом. Таким образом, x << 2 сдвигает x влево на две позиции, заполняя освобождающиеся биты нулями, что эквивалентно умножению на 4.

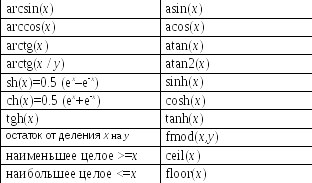
Дополнение: Унарная операция (~) дает дополнение к целому. Это означает, что каждый бит со значением 1 получает значение 0 и наоборот.

26. . Файлы math.h, stdlib.h

math.h — заголовочный файл стандартной библиотеки языка программирования Си, разработанный для выполнения простых математических операций. Все эти функции принимают double, если не определено иначе. Для работы с типами float и long double используются функции с постфиксами **f и l** соответственно. Все функции, принимающие или возвращающие угол, работают с радианами. Например sqrt, fabs, exp, pow, log, tan, sin.

stdlib.h — заголовочный файл стандартной библиотеки языка Си, который содержит в себе функции, занимающиеся выделением памяти, контроль процесса выполнения программы, преобразования типов и другие.

Заголовочный файлы stdlib.h определяет макрос NULL, являющийся константным нуль-указателем, который гарантированно указывает на некорректный адрес памяти. stdlib.h определяет тип данных, называемый sizeof.

27.Функции форматного ввода. Управляющая строка. Спецификаторы формата.

Ввод осуществляется с помощью функции **scanf(“управляющая строка”, список адресов**).

Список адресов представляет собой перечень имён со знаком **&(амперсант).** Управляющая строка для ввода содержит спецификации преобразования и управляющие символы.

Ввод происходит только до первого пробела, для ввод фраз используют gets(<ИД строковой переменной>);

Модификаторы:

L – используется для вывода long double,

l – используется для вывода long int,

h – используется для вывода short int,

u – используется для вывода unsigned.

Примеры: %ld %lu %Lf

Строка и формат как в форматном выводе.

28. Функции форматного вывода. Управляющая строка. Управляющие символы. Спецификаторы формата.

Для вывода используется функция **printf (“управляющая строка”, список объектов вывода);**

Список вывода состоит из имён переменных, констант и выражений. Управляющая строка содержит символы, задающие поясняющий текст, спецификации преобразования, каждая из которых вызывает вывод на экран значения очередного аргумента из списка вывода (начинается с %).

|  |  |
| --- | --- |
| Основные модификации формата:  %d Целое десятичное число 392  %o Беззнаковое целое в восьмеричном виде 657  %x Беззнаковое целое в 16-ом виде 7fa  %f или %F вещ.число с плавающей точкой  %с один символ  %% Знак %  %р указатель И др.  Printf (“Только %d%% предприятий не работало.\n”,5);  Результат: только 5% предприятий не работало. | Управляющие последовательности:  \n – новая строка  \t – горизонт.табуляция  \r – возврат каретки  \v – верт.табуляция  \f – перевод формата  \\ - обратная косая  \’ - апостроф  \0 – нулевой символ |

Функция **Puts**() выводит строку символов, автоматически добавляя к ней символ перехода на начало новой строки.

Функция **Putchar**() выводит один символ без перехода на новую строку.

29. Категории операторов. Синтаксис и семантика операторов. Пустой оператор. Составной оператор.

Категории:

* условные операторы (иф, свитч)
* операторы цикла (фор, ваил, ду ваил)
* операторы перехода (гото, брэйк, континью, ретурн)
* другие (оператор «выражение», пустой оператор)

Любое выражение, заканчивающееся ; является оператором. Пустой оператор состоит только из ; и используется там, где по правилам кода требуется оператор.

Составной оператор – последовательность операторов и объявлений, заключенных в фигурные скобки.

30.Операторы цикла. Цель использования. Итерация. Параметры цикла.

Цикл – многократно повторяемый участок программы, пока истинно некоторое условие. Цикл состоит из блока проверки условия цикла и тела цикла.

Итерация – один проход цикла. Проверка условия цикла осуществляется на каждой итерации до или после кода цикла.

Разновидности операторов цикла:

- while — цикл с предусловием;

- do…while — цикл с постусловием;

- for — параметрический цикл (цикл с заданным числом повторений).

Параметр цикла – переменная, управляющая выполнением цикла.

Цикл с предсловием (**while (выражение) код цикла;)** сначала проверяет условие, затем выполняет тело цикла. Есть вероятность, что цикл не будет выполнен ни разу.

Цикл с постусловием (**do оператор while выражение;**)сначала выполняет тело цикла, зачем проверяет условие. Следовательно, независимо от условия, всегда будет выполняться одна итерация.

Цикл с параметром (**for (инициализация; выражение; модификации) оператор;)** удобно использовать в том случае, когда нам заранее известно количество повторений цикла. Запись через ;

Цель: упростить и ускорить прогу.

31.Условные операторы. Тернарная операция.

Оператор ветвления применяется в случаях, когда выполнение или невыполнение некоторого набора команд должно зависеть от некоторого условия.

Условные операторы:

* простой: if(выражение) оператор1;
* полный: if(выражение) оператор1;

else оператор2;

Тернарная операция: **выр1 ? выр2 : выр3;**

Пример: вычислить максимум из а и b и присвоить z.

If (a>b) z=a; Else z=b;

Тернарная: **z=(a>b)? a : b ;**

Оператор выбора альтернатив switch

Switch () {

Case const1; оператор1; break;

Case const2; оператор2; break;

Case constN; операторN; break;

Default: оператор(N+1); break; }

Значения в () должны быть целого типа, допустим символьный.

Оператор брэйк позволяет организовать выход из оператора свитч на первый выполняемый оператор после данной конструкции.

32. Оператор выбора альтернатив switch. Операторы передачи управления (goto, continue, break, return)

Свитч из 27

|  |  |
| --- | --- |
| Все операторы языка СИ разделены на **категории**:  1)условные операторы, к которым относятся оператор условия if и оператор выбора switch;  2)операторы цикла (for,while,do while);  3)операторы перехода (break, continue, return, goto);  4)Логические операторы (И, ИЛИ, НЕ);  5)другие операторы (оператор “выражение”, пустой оператор);  6)Составные операторы. | Операторы перехода:  • оператор безусловного перехода goto;  • оператор выхода из цикла break;  • оператор перехода к следующей итерации цикла continue;  • оператор возврата из функции return.  Оператор goto |

Оператор безусловного перехода goto имеет формат: **goto <метка>;**

Оператор goto передает управление на помеченный оператор. Метка — это обычный идентификатор, областью видимости которого является функция, в теле которой он задан.

Использование оправдано в двух случаях:

• принудительный выход вниз по тексту программы из нескольких вложенных циклов;

• переход из нескольких мест функции в одно.

В остальных случаях для записи алгоритма существуют более подходящие средства, а исп-е goto приводит к усложнению структуры программы и затруднению отладки программы.

**Оператор break**

Оператор break используется внутри операторов цикла или switch для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break.

**Оператор continue**

Оператор перехода к следующей итерации цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации. НЕ используется в свитч.

**Оператор return**

Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова. Вид оператора: return [ выражение ];

33. Массивы. Объявления. Инициализация. Доступ к элементам массива. Действия над элементами массива.

Массив – сложный тип переменных. Он представляет собой упорядоченную конечную совокупность элементов одного типа. Описание массива в программе отличается наличием после имени квадратных скобок[], в которых задается количество элементов массива (размерность). Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы, что и для простых переменных. Инициализирующие значения для массивов записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются: int b[5] = {3, 2, 1}; / / b[0]=3, b[l]=2, b[2]=l, b[3]=0, b[4]=0

Размерность массива вместе с типом его элементов определяет объем памяти, поэтому размерность может быть задана только целой положительной константой или константным выражением. Если при описании массива не указана размерность, должен присутствовать инициализатор, в этом случае компилятор выделит память по количеству инициализирующих значений.

**Доступ к эл-там:**

Для доступа к конкретному элементу массива используются индексные выражения: **<имя массива> [<целочисл. выражение>];**  Индекс массива может быть не только константой, но и выражением, которое имеет целочисленный тип, например, a[i + 1]. Стоит помнить, что индексы элементов массива в языке C начинаются с 0, и номер последнего элемента на 1 меньше количества элементов массива. Имя массива является адресом его начала. Оно имеет тип константный указатель на <тип элементов массива>. Конструкция a[i] эквивалентна \*(a + i).

**Действия над элементами массива**: Индексированные элементы массива называются индексированными переменными и могут быть использованы так же, как и простые переменные. Например, они могут находиться в выражениях в качестве операндов, использоваться в операторах, им можно присваивать любые значения, соответствующие их типу. Копированием массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Перестановка значений элементов массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массив.

34. Одномерный, двумерный и многомерный массив.

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива.

Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. Нумерация ячеек массива всегда начинается с 0. Индекс ячейки – это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой).

Многомерный массив – массив, в котором положение элемента определяется несколькими индексами. Соответственно, двумерный массив - массив, у которого в описании указаны два индекса. Первый индекс – строки, второй – столбцы. Двумерный массив целочисленных элементов можно инициализировать как: int s[3][5]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}. По аналогии с первым случаем это соответствует набору следующих операторов присвоения: int s[0][0]=1; int s[0][1]=2; и т.д. Вообще многомерные массивы, в том числе и двумерные, можно инициализировать как массив массивов. Например, две следующие инициализации равны между собой: int s[3][5]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}; и int s[3][5]={{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15}}; Во втором случае авторы ввели дополнительные фигурные скобки для записи элементов каждой строки двумерного массива. Если при этом количество элементов в строке не совпадает с числом столбцов в массиве, то соответствующие пропущенные элементы строки считаются неопределенными, так как они не определены.

35. Указатели. Инициализация указателей. Примеры работы. Область применения.

Указатель - переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Чтобы узнать адрес конкретной переменной в С++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных для того, чтобы его присвоить указателю. Указатели ускоряют процесс обработки данных. В основном указатели используются для организации динамического распределения памяти. Любой указатель необходимо объявить перед использованием. **Инициализация указателей /\*тип данных\*/ \* /\*имя указателя\*/;** Принцип объявления указателей такой же, как и принцип объявления переменных. Отличие заключается в том, что перед именем ставится \*. При объявлении указателей компилятор выделяет несколько байт памяти. Чтобы получить значение, на которое ссылается указатель, нужно воспользоваться операцией разыменования указателя \*. Необходимо поставить \* перед именем и получим доступ к значению указателя.

|  |  |
| --- | --- |
| Область применения:  -работа с массивами, особенно типа char;  -передачи данных в функцию по адресу;  -передачи сложных динамических структур данных. | Пример:  Int i, j=8, k=5, \*y;  y=&I;  \*y=2; // i=2  y=&j;  \*y+=i; // j+=i -> j=j+i -> j=j+2=10  Y=&k;  k+=\*y; //k+=k ->k=k+k=10  (\*y)++; //k++ ->k=k+1=10+1=11 |

36. Операции над указателями.

-Доступ по указателю \*E (разадресация или разыменовывание)

-Операция присваивания

-Операция уменьшения/увеличения указателя

int i , \*p;

p+i // указатель на адрес i-го эл-та после данного

p-i // указатель на адрес i-го эл-та перед данным

-Оп-я сложного присваивания p+=i; p-=i;

-Оп-я инкремента/декремента p++;==p;p--;--p.

-Оп-я индексирования p[i]. Эта оп-я полностью аналогична \*(E+i), т.е. из памяти извлекается и используется значение i-го эл-та массива, адрес которого присвоен указателю E.

- Вычитание указателей E1-E2. E1-E2-указатели на один и тот же набор данных одного и того же типа. Результат имеет тип int и равен кол-ву эл-тов, которые можно расположить в ячейке памяти с E2 по E1.

-Операции отношений. E1==E2; E1>=E2; E1>E2; E1<=E2; E1<E2; E1!=E2.

Результат всех операций имеет тип int. Любой указатель может быть проверен на равенство и неравенство специальным значением NULL. Функции выделения памяти возвращают NULL при появлении каких-либо ошибок.

37. Ссылки.

Ссылка представляет собой синоним имени, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда разыменовывается. Формат объявления ссылки: тип & имя; где тип — это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним имя является именем переменной ссылочного типа. Запомните следующие правила. Переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она является параметром функции, описана как extern или ссылается на поле данных класса. После инициализации ссылке не может быть присвоена другая переменная.

Тип ссылки должен совпадать с типом величины, на которую она ссылается. Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки. Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений. Ссылки позволяют использовать в функциях переменные, передаваемые по адресу, без операции разадресации, что улучшает читаемость программы. Ссылка, в отличие от указателя, не занимает дополнительного пространства в памяти и является просто другим именем величины. Операция над ссылкой приводит к изменению величины, на которую она ссылается.

38. Применение указателей для работы с массивами. Одномерный динамический массив. Выделение/высвобождение памяти.

В языке СИ между указателями и массивами существует тесная связь.

-Идентификатор массива – адрес памяти, начиная с которого он расположен, т.е. адрес его первого эл-та.

-Пусть объявлен массив а из 5 целочисленных эл-тов и указатель q на целочисленные переменные: int a[5], \*q;

Id массива а является константным указателем на его начало. Если выполнена операция q=a; - присваивание константы переменой т.е. q=4000 (q=&a[0]), то с учетом адресной арифметики выр-я a[i] и \*(q+i) приводят к одинаковым результатам – обращению к i-му элементу массива.

Для любых указателей можно использовать две эквивалентные формы выражений для доступа к элементам массив: q[i] и \*(q+i). Первая форма удобнее для читаемости текста, вторая – эффективнее по быстродействию программы.

Например, для получения значения 4-го эл-та массива можно написать a[3] или \*(a+3).

Обращение к первому эл-ту массива: \*a <-> a[0]

Указатели, как переменные другого типа, могут объединяться в массивы. int \*a[10], y;

Теперь каждому из эл-тов массива можно присвоить адрес целочисленной переменной y, например: a[1]=&y;

Чтобы теперь найти значение переменной y через данный элемент массива а, необходимо записать \*a[1].

Динамическим называется массив, размер которого может меняться во время исполнения программы.

Выделение памяти под одномерный динамический массив int \*mas = new int \*[x];

Освобождение памяти из-под одномерного динамического массива: delete [ ] mas;

39. Двумерный динамический массив. Выделение/высвобождение памяти.

Динамический массив — это массив, у которого количество ячеек можно задавать и переменной, и числовой константой. Это большой плюс перед использованием статического массива.

Двумерный массив – массив, у которого в описании указаны два индекса. Первый индекс – строки, второй– столбцы.

При формировании двумерного динамического массива сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под одномерные массивы.

int \*\*mas = new int \*[x];

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

{ mas[i] = new int[y]; }

Удаление из динамической памяти двумерного массива осуществляется в порядке, обратном его созданию, то есть сначала высвобождается память, выделенная под одномерные массивы с данными, а затем память, выделенная под одномерные массив указателей:

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

delete[ ] mas[i];

delete[ ] mas;

**40. Многомерные массивы и указатели на указатели.**

В языке Си можно описать переменную типа «указатель на указатель». Это ячейка оперативной памяти, в которой будет храниться адрес указателя на какую-либо переменную Признак такого типа данных — повторение символа «\*»

перед идентификатором переменной. Количество символов «\*» определяет уровень вложенности указателей друг в друга. При объявлении указателей на Указатели возможна их одновременная инициализация. Например:

int a=5;

int \*p1=&a:

int \*\*pp1=&p1;

int \*\*\*ppp1=&pp1;

Теперь присвоим целочисленной переменной a новое значение, например 10.

Одинаковое присваивание произведут следующие операции: a=10; \*p1=10; \*\*pp1=10; \*\*\*ppp1=10;

Для доступа к области ОП, отведенной под переменную а, можно использовать и индексы. Справедливы след. аналоги, если мы работаем с многомерными массивами:

\*p1 <->p1[0]   
\*\*pp1<->pp1 [0][0]   
\*\*\*ppp1 <-> ppp1 [0][0][0]

Идентификатор двухмерного массива – это указатель на массив указателей (типы int, \*\*m;) поэтому выр-е a[i][j] эквивалентно выр-ю \*(\*(m+i)+j)

41. Динамическая память, функции malloc(), calloc(), realloc(), free()

Динамически распределяемую память следует использовать в случае если мы заранее (на момент написания программы) не знаем сколько памяти нам понадобится и при работе с большими объемами данных . Для работы с динамической памятью в языке СИ используются следующие функции:

Для выделения памяти в си используется функция malloc (memory allocation) из библиотеки stdlib.h. void \* malloc(size\_t size); Функция выделяет size байтов памяти и возвращает указатель на неё. Если память выделить не удалось, то функция возвращает NULL. Так как malloc возвращает указатель типа void, то его необходимо явно приводить к нужному нам типу. Например, создадим указатель, после этого выделим память размером в 100 байт. void main() { int \*p = NULL; p = (int\*) malloc(100); free(p); } После того, как мы поработали с памятью, необходимо освободить память функцией free (передать указатель, значение которого получено из функции malloc). Функция free освобождает память, но при этом не изменяет значение указателя. Функция calloc выделяет n объектов размером m и заполняет их нулями. Обычно она используется для выделения памяти под массивы. void\* calloc(size\_t num, size\_t size); функция realloc (re-allocation) позволяет изменить размер ранее выделенной памяти и получает в качестве аргументов старый указатель и новый размер памяти в байтах: void\* realloc(void\* ptr, size\_t size);

42. Строки. Объявление. Инициализация. Действия над строками и элементами строки.

Строка в СИ – это массив элементов типа char, в конце которого помещен '\0' (нуль-терминатор). Символьные строки состоят из набора символьных констант, заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки \0 и отводить дополнительный байт под него. Строка при объявлении мб инициализирована начальным значением, например, так: char string[10] = "abcdefghf"; Если подсчитать кол-во символов, их окажется 9, а размер строки 10 символов, последнее место отводится под \0, причём компилятор сам добавит его в конец строки. При объявлении строки не обязательно указывать её размер, но при этом обязательно нужно её инициализировать начальным значением. Тогда размер строки определится автоматически и в конец строки добавится нуль-терминатор. Строка может содержать символы, цифры и специальные знаки. В СИ строки заключаются в двойные кавычки. Имя строки является константным указателем на первый символ.

Действия над строками:

• инициализация;

• ввод/вывод (gets, puts);

• копирование строк (strcpy, strncpy);

• конкатенация или объединение (strcat, strncat);

• определение длины строки (strlen);

• сравнение строк (strcmp, strncmp);

• поиск символов в строке (strchr, strcspn, strspn, strprbk).

43. Строки. Ввод/вывод строк.

Строка символов – одномерный массив типа char, заканчивающийся нулевым байтом.

Нулевой бат – байт, каждый бит которого равен нулю, при этом для нулевого байта определена символьная константа ‘\0’ (признак окончания строки, нуль-терминатор)

-scanf()-вводит значения для строковых переменных спецификатором ввода %s до появления первого символа «пробел» (символ «&» перед ID строковых данных указывать не надо);

-gets () – ввод строки с пробелами внутри этой строки завершается нажатием клавиши ENTER.

-Обе функции автоматически ставят в конец строки нулевой байт.

-Вывод строк производится функциями printf() или puts() до первого нулевого байта (‘\0’);

-printf() не переводит курсор после вывода на начало новой строки;

-puts() автоматически переводит курсор после вывода строковой информации в начало новой строки.

**44. Строки. Библиотека string.h. Прототипы функций для работы со строкой.**

Строка символов – одномерный массив типа char, заканчивающийся нулевым байтом.  
Нулевой бат – байт, каждый бит которого равен нулю, при этом для нулевого байта определена символьная константа ‘\0’ (признак окончания строки, нуль-терминатор)

Описание прототипов стандартных библиотечных функций находятся в файле string.h  
Функция int srtlen(char \*S) возвращает длину строки (количество символом в строке), при этом завершающий

нулевой байт не учитывается.

Пример:

char \*S1 = "Брест!\0",

S2[] = "БрГТУ - Ура!";

printf("%d, %d.", strlen(S1), strlen(S2));

Результат выполнения данного участка программы: 6, 10.

Функция strcpy(char \*S1, char \*S2) копирует содержимое строки S2 в строку S1.

Функция strcat(char \*S1, char \*S2) присоединяет строку S2 к строке S1 и помещает её в массив, где находилась строка S1, при этом строка S2 не изменяется. Нулевой байт, который завершал S1, заменяется первым символом строки S2.

Функция int strcmp(char \*S1, char \*S2) cравнивает строки S1 и S2 и возвращает значение <0, если S1 < S2; >0, если S1 > S2; =0, если строки равны, т. е. содержат одно и то же число одинаковых символов.

Функция преобразования строки S в число:

целое: int atoi(char \*S);

длинное целое: long atol(char \*S);

действительное: double atof(char \*S);

При ошибке данные функции возвращают значение 0.

Функции преобразования числа V в строку S:

Целое itoa(int V, char \*S, int kod);

длинное целое: ltoa(long V, char \*S, int kod); 2 <= kod <= 36,

для отрицательных чисел: kod = 10.

Пример функции del\_c() , в которой удаляется символ "c" из строки S каждый раз, когда он встречается:

void del\_c(char s[], int c) {

int i, j;

for (i = j = 0; s[i] != "\0"; i++)

if (s[i] != c)

s[j++] = s[i];

s[j] = "\0"; }

**45. Модульное программирование**С увеличением объема программы становится невозможно удерживать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является её разбиение на части.  
В языках Си и С++ задача может быть разбита на простые функции. После чего программу можно рассматривать в более укрупненном виде, а именно на уровне взаимодействия функций, т. е. использование функций ведёт к повышению степени абстракции программы и упрощению её структуры. Разбиение программы на функции также позволяет избежать избыточности кода, потому что функции записывают один раз, а вызвать её на выполнении можно многократно из разных точек программы. Процесс откладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки.   
Следующим шагом повышения уровня абстракции программы является группировка функций и связанных с ними данных в отдельные файлы – **модули**. В результате компиляции объектные модули объединяются в отдельную программу с помощью компоновщика.  
Разделение на модули уменьшает время компиляции, улучшает процесс отладки, скрывая несущественные недостатки за интерфейсом модуля и позволяет отлаживать программу по частям.  
Модуль содержит данные и функции их обработки. Другим модулям не желательно иметь средства обработки этих данных. Они должны пользоваться для этого функциями первого модуля. Чтобы использовать модуль, нужно знать только его интерфейс.  
Разделение программ на модули является сложной задачей, которая должна решаться на этапе проектирования задачи. Модульность поддерживается с помощью директив **include и define.** **Модульное программирование** – технология программирования, когда алгоритмы всей решаемой задачи разбиваются на отдельные, логически завершенные части.

**46. Подпрограммы. Виды подпрограмм**  
Когда размер программы не сложен и не велик, то пишется как одно целое. С ростом размера программы нужно разбивать ее на подзадачи и оформлять в виде подпрограмм, а основная программа служит для вызова этих подпрограмм. **Подпрограмма** – часть всей программы, оформленная особым образом (логически завершенная часть программы). Активное использование подпрограмм – основа модульного программирования.  
**Модульное программирование** – технология программирования, когда алгоритмы всей решаемой задачи разбиваются на отдельные, логически завершенные части.  
Подпрограмма может быть оптимизирована, но не может быть запущена на исполнение.  
Преимущества использования подпрограмм:  
1) возможно создание достаточно больших программ  
2) повторное использование ранее использованного кода  
3) разработку отдельных подпрограмм можно поручить разным людям  
4) сокращается срок разработки программ  
5)повышается надежность программы  
6) уменьшается размер программы  
Недостатки:  
1) уменьшается скорость работы программы  
2) необходима более высокая квалификация программиста  
**Виды подпрограмм:**-Процедуры - подпрограммы, которая не возвращает через свое имя результат работы, поэтому вызывается как отдельный оператор.  
Общается с внешним миром через список параметров (входные/выходные).  
-Функции - Подпрограммы, через имя возвращающие результат работы. Вызываются в выражении, а не как отдельный оператор.   
Через список параметров может получить входные данные и возвращает выходные данные.  
В виде функции – результат отдельный (однострочный, объект и т.д.).  
В виде процедуры – несколько объектов.  
Через список параметров можно вернуть одиночные и составные объекты.  
Возвращать через параметры и имя – дурной тон, плохо. (че)  
**47. Функции. Объявление и вызов**В Си/Си++ имеются только функции(нет процедур). Если необходима именно процедура – функция с типом void.  
Чтобы использовать в программе функцию, нужно:  
1) задать прототип функции (объявление)  
2) вызвать функцию в необходимом месте, в функции main  
3) задать определение функции (реализация)  
Правило:  
Объявление функции должно находиться в тексте раньше ее вызова (проверка правильности компилятором)  
Прототип функции задает ей имя, тип возвращаемого и список передаваемых параметров.  
**[<класс>] тип имя ([список\_параметров]);** **int sum (int a, int b); // объявление функции**  
Определение функции содержит тело функции, представляющей собой последовательность операторов и описаний в фигурных скобках.  
**[класс] тип имя ([список параметров]) {тело функции}** **int sum (int a, int b) {return (a+b);} //определение функции**Составные части объявления  
С помощью модификатора **класс** можно задать область видимости функции, используя ключевые слова extern и static. Все необязательное в [].  
**extern** – глобальная видимость только в пределах модуля, в котором определена функция. При использовании extern имя должно быть уникальным.  
Тип возвращаемого значения может быть любым, кроме массива и функции.  
Если надо вернуть массив и функцию, то исп. Указатель.  
Использование **static** внутри функции является самым простым. Это означает, что после инициализации переменной она остается в памяти до конца программы.  
Если функция не должна возвращать значения, то исп void, иначе возвращается целочисленное значение.  
Список параметров определяет величины, которые требуется передать функции при ее вызове. Элементы списка отделяются запятыми.  
Для каждого параметра указывается его тип и имя.  
В определении, объявлении и при вызове одной и той же функции тип и имя функции должны совпадать.  
Вызов может находиться в любом месте программы, где по синтаксису допустимо его использование.  
Если функция возвращает не void, то может входить в состав выражения и быть в правой части оператора присваивания.  
Объявления обязаны быть до main.  
Величины и параметры внутри функции – локальные.  
При вызове функции выделяется память под локальные переменные.

|  |  |  |
| --- | --- | --- |
| Пример исп-я static: #include <iostream.h> void f(int a) { int m=0; cout<<»nmp\n»; while (a--){ static int n=0; int p=0; cout<<n++<<’’<<m++<<’’<<p++<<’\n’;  } } int main() { f(3); f(2); return 0; } | nmp 000 110 220 nmp 300 410 | **Операция typedef** Любому типу данных можно задать новое имя с помощь операции **typedef**. **typedef <тип> <новое имя>;** Введенный таким образом новый тип используется аналогично стандартным типам, например, введя пользовательские типы typedef unsigned int UINT; typedef char M\_s[100]; декларации идентификаторов вводимых тип имеют вид: UINT i,j -> две переменные типа unsigned int; M\_s str[10]->массив из 10 строк по 100 символов  Идентификатор функции является константным указателем на начало функции в ОЗУ и не может быть значением переменной, но имеется возможность декларировать указатели как функции, с которыми можно обращаться как с переменными. Т.е. можно создать массив, эл-ми которого будут указатели на функции. |

48. Обмен информацией между функциями

При работе программы функции должны обмениваться информацией. Это можно осуществить с помощью:

1) глобальных переменных;

2) параметра;

3) возвращаемое функцией значение.

Использование глобальных переменных

Глобальные переменные видны во всех функциях, где не описаны локальные переменные с теми же именами. Поэтому использовать их для передачи данных между функциями очень легко, но это не рекомендуется поскольку

затрудняет откладку, и препятствует помещению функции в библиотеки общего пользования.

Если в теле функции имя глобальной переменной совпадает с локальной, то все операции выполняются с локальной. В этом случае для доступа к глобальной необходимо применить операцию расширения области видимости (::).

Использование параметров функции

Параметры, перечисленные в заголовке описания функции, называются **формальные**, а записанные в операторе вызова функции - **фактическими**.

Существует 2-ва способа передачи параметров функции:

1) По значению

2) По адресу

При передачи по значению на месте формальном параметров записываются значения фактических параметров. Вычисление функции в стек записывает копии и оператор функции работает с этим копиями. Доступа к исходным параметрам функции нет, и нет возможности их поменять.

При передаче по адресу в стек заносятся копии адресов параметров через указатель, ссылку и операцию взятия адреса. Следовательно, функция осуществляет доступ память по адресу, а значит, может поменять эти параметры.

Использование возвращаемого значения

Механизм возврата из функции вызывающую его функции реализуется оператором return [выражение];

Функция может содержать несколько операций return. Это определяется потребностями алгоритма. После любого из этих операторов прекращается выполнение функции. Выражение, указанное после return, не явно преобразуется к типу возвращаемого к функции значения и передается в точку вызова функции. Вместо выражения можно записать переменную или константу.

49. Передача одномерных и многомерных массивов в качестве параметров функции

Передача массивов в качестве параметров.

При передаче массивов функции передается указатель на его первый элемент. Иными словами, массив всегда передаётся по адресу, при этом информация о количестве элементов массива теряется, и следует передавать размерность через отдельный параметр. В случае строки его длину можно определить через функцию `sizeof()`.

Два способа использования одномерных массивов в качестве параметров функции.

I способ. Синтаксис:

тип\_функции имя\_функции ( тип\_массива имя\_массива [ n ] ){ тело функции}

n – количество элементов в массиве

Вызов функции: имя\_функции (имя\_массива);

Используется для массивов одинаковой длины.

Пример: Написать функцию нахождения минимума в одномерном массиве.

int mini (int a[n])

{ int min = 999;

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

if (a[i] < min) min = a[i];

return min;

}

II способ. Синтаксис:

тип\_функции имя\_функции ( тип\_массива имя\_массива [ ], int n ) { тело функции}

n – количество элементов в массиве

Вызов функции: имя\_функции (имя\_массива, кол-во элементов мас-ва);

В определении функции используется два формальных параметра, значит и при вызове тоже используется два фактических параметра (имя массива и количество элементов). Второй способ используется, когда функция с таким описанием формальных параметров, используется для массивов разной длины.

50. Указатель на функцию

Формат объявления указателя на функцию:  **тип (\*переменная-указатель)(список параметров);**

Например, float (\*p\_f)(char,float);

Чтобы установить переменную-указатель на конкретную функцию, достаточно ей присвоить ее идентификатор: **переменная-указатель=ИД функции**

Например, имеется функция с прототипом: float f1 (char, float); тогда оператор p\_f=f1; установит указатель на данную функцию.

Вызов функции после установки на нее указателя выглядит так: **(\*пер.-указатель)(список аргументов); или**

**пер.-указатель(список аргументов)**

После таких действий (кроме стандартного обращения к функции ИДфункции(список аргументов);) появится еще два способа вызова функции:

(\*пер.-указатель)(список аргументов);

пер.-указатель(список аргументов);

Примеры:

f1(‘z’, 1.5); // обращение к функции по ИД

(\*p\_f)(‘z’,1.5); // по указателю

P\_f(‘z’, 1.5); //по ИД указателя

51. Параметры функции со значениями по умолчанию. Функции с переменным числом параметров

Язык программирования Си допускает использование функций, которые имеют нефиксированное количество параметров. Более того может быть неизвестным не только количество, но и типы параметров. То есть точное определение параметров становится известным только во время вызова функции. Для определения параметров неопределенной длины в таких функциях используется многоточие:

тип имя\_функции(обязательные параметры, ...)

При этом надо учитывать, что функция должна иметь как минимум один обязательный параметр.

52. Параметры командной строки функции main

Функция мэйн может быть определена с параметром. Во внешнем окружении все данные – строки символов.

Два параметра:

* число передач
* сами строки

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

Параметры со значением по умолчанию.

Int f(int a, int b=0);

Void f(int, int 100.char\* =0);

F(100); - по умолчанию b=100

F(a,1); - b присвоено 1

Fl(a); - первому формальному параметру присвоено значение переменной а, остальным по умолчанию

Fl(a,10); - первому формальному параметру присвоено значение переменной а, второму 10, третьему по умолчанию.

Fl(a,10,”Vasia”); - всем формальным параметрам присвоены значения

Fl(a, .”Vasia”); - неверно

|  |  |
| --- | --- |
|  |  |