**1. Алгоритм. Классы алгоритмов. Способы записи алгоритмов. Свойства алгоритма.**

Алгоритм — это точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому конечному результату.

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, объем

данных большой. Требуют хорошей 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.

Свойства алгоритма:

• результативностью (означает возможность получения

результата после выполнения конечного количества операций)

• определенностью (состоит в совпадении получаемых

результатов независимо от пользователя и применяемых технических средств)

• массовостью (заключается в возможности применения

алгоритма к целому классу однотипных задач, различающихся конкретными значениями исходных данных)

• дискретностью (означает разбиение алгоритма на конечную

последовательность действий или шагов при его выполнении)

• конечностью (означает то, что алгоритм должен выполняться

за конечное время)

**3. Базовые конструкции структурированного программирования. Цель использования базовых конструкций.**

Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

• последовательное исполнение — однократное выполнение операций в

том порядке, в котором они записаны в тексте программы;

• ветвление — однократное выполнение одной из двух или более

операций, в зависимости от выполнения некоторого заданного условия;

• цикл — многократное исполнение одной и той же операции до тех пор,

пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Цель использования базовых конструкций:

1. Обеспечить дисциплину программирования

2. Улучшить читабельность программы

3. Повысить эффективность программы

4. Повысить надежность программы

**2. Архитектура Фон – Неймана.**

Архитектура фон Неймана — широко известный принцип совместного хранения команд и данных в памяти компьютера. Вычислительные системы такого рода часто обозначают термином «машина фон Неймана», однако соответствие этих понятий не всегда однозначно. В общем случае, когда говорят об архитектуре фон Неймана, подразумевают принцип хранения данных и инструкций в одной памяти.

Принципы фон Неймана:

* Принцип однородности памяти

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования; то есть одно и тоже значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных. Такой прием носит название модификации команд и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции — перевода текста программы с языка высокого уровня на язык конкретной вычислительной машины.

* Принцип адресности

Структурно основная память состоит из пронумерованных ячеек, причем процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.

* Принцип программного управления

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

* Принцип двоичного кодирования

Согласно этому принципу, вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1. Каждый тип информации представляется двоичной последовательностью и имеет свой формат. Последовательность битов в формате, имеющая определенный смысл, называется полем. В числовой информации обычно выделяют поле знака и поле значащих разрядов. В формате команды можно выделить два поля: поле кода операции и поле адресов.

**4. Базовые средства языка СИ. Состав языка. Основные элементы алгоритмического языка.**

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

* Слова наз. “лексемами” - элементарными конструкциями.
* Словосочетания наз. выражения.
* Предложения - это операторы.

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

**Алфавит языка** - это его символы.

**Лексема** - это минимальная единица языка, имеющая самостоятельный смысл.

**Выражение** задаёт правило вычисление некот. значения.

**Оператор** задаёт законченное описание некот. действия.

Для описания сложного действия требуется последовательность операторов. Операторы могут быть объединены в составной оператор, или блок (блоком в языке СИ считается последовательность операторов, заключенная в фигурные скобки { }). В этом случае они рассматриваются как один оператор.

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

**5. Ввод\вывод строки.**

Строка в СИ – это массив элементов типа char, в конце которого помещен символ '\0' (нуль-терминатор).

Для ввода и вывода строк в библиотеке stdio.h содержатся специализированные функции gets и puts.

Функция gets() имеет прототип:

*char \*gets(char \*str);*

где str - это массив символов. Функция gets() возвращает указатель на str.

Функция puts() выводит передаваемый ей аргумент на экран, завершая вывод переходом на новую строку. Она имеет следующий прототип:

*int puts(char \*str);*

Здесь str - это выводимая строка. Функция возвращает нецелое число в случае удачи и EOF - в случае неудачи.

**6. Двумерный динамический массив. Выделение/высвобождение памяти.**

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;

**7. Двумерный массив. Инициализация многомерного массива.** Двумерный массив – это массив, в котором каждый элемент является также массивом.

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.

**8. 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:

*void \*malloc(size\_t size);*

В качестве входного параметра функция принимает размер памяти, которую требуется выделить. Возвращаемым значением является указатель на выделенный в куче участок памяти.

Для выделения памяти под 1 000 000 int`ов необходимо выполнить следующий код:

*int \* p = malloc(1000000\*sizeof(int));*

Если ОС не смогла выделить память (например, памяти не хватило), то malloc возвращает 0.

После окончания работы с выделенной динамически памятью нужно освободить ее. Для этой цели используется функция free, которая возвращает память под управление ОС.

*void free(void \*ptr);*

В качестве входного параметра в free нужно передать указатель, значение которого полученно из функции malloc. Вызов free на указателях полученных не из malloc (например, free(p+10)) приведет к неопределенному поведению. Это связанно с тем, что при выделении памяти при помощи malloc в ячейки перед той, на которую указывает возвращаемый функцией указатель операционная система записывает служебную информацию. При вызове free(p+10) информация находящаяся перед ячейкой (p+10) будет трактоваться как служебная.

*void \*calloc(size\_t nmemb, size\_t size);*

Функция работает аналогично malloc, но отличается синтаксисом (вместо размера выделяемой памяти нужно задать количество элементов и размер одного элемента) и тем, что выделенная память будет обнулена. Например, после выполнения

*int \* q = (int \*) calloc(1000000, sizeof(int))*

q будет указывать на начало массива из миллиона int`ов инициализированных нулями.

*void \*realloc(void \*ptr, size\_t size);*

Функция изменяет размер выделенной памяти (на которую указывает ptr, полученный из вызова malloc, calloc или realloc). Если размер указанный в параметре size больше, чем тот, который был выделен под указатель ptr, то проверяется, есть ли возможность выделить недостающие ячейки памяти подряд с уже выделенными. Если места недостаточно, то выделяется новый участок памяти размером size и данные по указателю ptr копируются в начало нового участка.

**9. Доступ к элементам массива. Действия над элементами массива.**

Для доступа к элементам массива существует два различных способа.

**Первый способ** связан с использованием обычных индексных выражений в квадратных скобках, например, array[16]=3 или array[i+2]=7. При таком способе доступа записываются два выражения, причем второе выражение заключается в квадратные скобки. Одно из этих выражений должно быть указателем, а второе - выражением целого типа. Последовательность записи этих выражений может быть любой, но в квадратных скобках записывается выражение следующее вторым. Поэтому записи array[16] и 16[array] будут эквивалентными и обозначают элемент массива с номером шестнадцать. Указатель, используемый в индексном выражении не обязательно должен быть константой, указывающей на какой-либо массив, это может быть и переменная. В частности после выполнения присваивания ptr=array доступ к шестнадцатому элементу массива можно получить с помощью указателя ptr в форме ptr[16] или 16[ptr].

**Второй способ** доступа к элементам массива связан с использованием адресных выражений и операции разадресации в форме \*(array+16)=3 или \*(array+i+2)=7. При таком способе доступа адресное выражение равное адресу шестнадцатого элемента массива тоже может быть записано разными способами \*(array+16) или \*(16+array).

При реализации на компьютере первый способ приводится ко второму, т.е. индексное выражение преобразуется к адресному. Для приведенных примеров array[16] и 16[array] преобразуются в \*(array+16).

Для доступа к элементам многомерного массива можно использовать любой из вышеперечисленных способов. Например, для двумерного массива: arr2[1][2] или эквивалентные \*(\*(arr2+1)+2) и (\*(arr2+1))[2].

**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 массива.

**10. Идентификаторы. Ключевые слова. Константы.**

**Идентификатор** — это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются, например, sysop, SySoP и SYSOP — три различных имени. 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 создается на этапе объявления переменной, функции, типа и т. п., после этого его можно использовать в последующих операторах программы. Идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка.

**Ключевые слова** — это зарезервированные 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.

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

**12. Массивы. Объявления. Инициализация.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнять однообразные действия, им дают одно имя, а различают по порядковому номеру. Это позволяет компактно записывать множество операций с помощью циклов. **Конечная именованная последовательность однотипных величин называется массивом**.

Описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива (размерность).

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), 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, необходимый для размещения массива, которое выполняется на этапе компиляции, 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.

**13. Массивы. Объявления. Работа с указателями.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнять однообразные действия, им дают одно имя, а различают по порядковому номеру. Это позволяет компактно записывать множество операций с помощью циклов. Конечная именованная последовательность однотипных величин называется массивом. Описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива (размерность).

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), 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]=1, b[3]=0, b[4]=0

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, 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. В языке СИ между указателями и массивами существует тесная связь. Например, когда объявляется массив в виде int array[25], то этим определяется не только выделение памяти для двадцати пяти элементов массива, но и для указателя с именем array, значение которого равно адресу первого по счету (нулевого) элемента массива, т.е. сам массив остается безымянным, а доступ к элементам массива осуществляется через указатель с именем array. С точки зрения синтаксиса языка указатель arrey является константой, значение которой можно использовать в выражениях, но изменить это значение нельзя. Для доступа к элементам массива можно использовать указатели: \*(a+3) будет ссылаться на 3 элемент массива a.

**14. Одномерный массив. N-мерный массив.**

**Одномерные массивы в СИ.**

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).

**N-мерный массив**

Можно объявлять не только двумерные массивы, но и массивы с большим количеством измерений. Например, объявление int A[m][n][k] создает трехмерный массив из m\*n\*k элементов. Для обращения к каждому элементу такого массива необходимо указать три индекса: A[x][y][z], при этом 0<=x, x<m, 0<=y, y<n, 0<=z, z<n. Количество измерений в массиве может быть практически бесконечным (т.е. достаточным для решения любых практических задач).

**11. Классификация языков программирования.**

Существуют различные классификации языков программирования.

По наиболее распространенной классификации все языки программирования, в соответствии с тем, в каких терминах необходимо описать задачу, делят на языки **низкого** и **высокого** уровня.

Если язык близок к естественному языку программирования, то он называется языком высокого уровня, если ближе к машинным командам, – языком низкого уровня.

В группу языков **низкого** уровня входят машинные языки и языки символического кодирования: Автокод, Ассемблер. Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена. 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 (внутреннего языка, структуры памяти и т.д.).

К языкам программирования **высокого** уровня относят Фортран (переводчик формул – был разработан в середине 50–х годов программистами фирмы IBM и в основном используется для программ, выполняющих естественно – научные и математические расчеты), Алгол, Кобол (коммерческий язык – используется, в первую очередь, для программирования экономических задач), Паскаль, Бейсик (был разработан профессорами Дармутского колледжа Джоном Кемени и Томасом Курцом.), Си (Деннис Ритч – 1972 году), Пролог (в основе языка лежит аппарат математической логики) и т.д.

Эти языки машинно–независимы, т.к. они ориентированы не на систему команд той или иной ЭВМ, а на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

Программу, написанную на языке программирования высокого уровня, ЭВМ не понимает, поскольку ей доступен только машинный язык. Поэтому для перевода программы с языка программирования на язык машинных кодов используют специальные программы – трансляторы.

Существует три вида транслятора: интерпретаторы (это транслятор, который производит пооператорную обработку и выполнение исходного кода программы), компиляторы (преобразует всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и лишь потом исполняется) и ассемблеры (переводят программу, записанную на языке ассемблера, в программу на машинном языке).

Языки программирования также можно разделять на поколения:

– **языки первого поколения**: машинно–ориентированные с ручным управлением памяти на компьютерах первого поколения.

– **языки второго поколения**: с мнемоническим представлением команд, так называемые автокоды.

– **языки третьего поколения**: общего назначения, используемые для создания прикладных программ любого типа. Например, Бейсик, Кобол, Си и Паскаль.

– **языки четвертого поколения**: усовершенствованные, разработанные для создания специальных прикладных программ, для управления базами данных.

– **языки программирования пятого поколения**: языки декларативные, объектно–ориентированные и визуальные. Например, Пролог, ЛИСП (используется для построения программ с использованием методов искусственного интеллекта), Си++, Visual Basic, Delphi.

Языки программирования также можно классифицировать на процедурные и непроцедурные.

В **процедурных** языках программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий. Среди процедурных языков выделяют в свою очередь структурные и операционные языки. В **структурных** языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы и т.д. (Паскаль, Си, Ада, ПЛ/1). В **операционных** языках для этого используются несколько операций.( Фортран, Бейсик, Фокал).

**Непроцедурное** (**декларативное**) программирование появилось в начале 70-х годов 20 века, К непроцедурному программированию относятся функциональные и логические языки.

В **функциональных** языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь делятся на еще более простые задачи и т.д. Один из основных элементов функциональных языков – рекурсия. Оператора присваивания и циклов в классических функциональных языках нет.

В **логических** языках программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого системе можно задавать вопросы. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается Пролог.

Можно выделить еще один класс языков программирования – **объектно–ориентированные** языки высокого уровня. На таких языках не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно–ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме.

Первый объектно-ориентированный язык программирования Simula был создан в 1960-х годах Нигаардом и Далом.

**16. Операторы. Оператор перехода**

В CИ есть четыре оператора, изменяющих естественный порядок выполнения вычислений:

* 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;

В теле той же функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

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

Использование оператора безусловного перехода оправдано в двух случаях:

* принудительный выход вниз по тексту программы из нескольких

вложенных циклов или переключателей;

* переход из нескольких мест функции в одно (например, если перед

выходом из функции всегда необходимо выполнять какие-либо действия).

В остальных случаях для записи любого алгоритма существуют более подходящие средства, а использование goto приводит только к усложнению структуры программы и затруднению отладки (Даже в приведенных случаях допустимо применять goto только в случае, если в этих фрагментах кода не создаются локальные объекты. В противном случае возможно применение деструктора при пропущеном конструкторе, что приводит к серьезным ошибкам в программе). Применение goto нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и один выход.

В любом случае не следует передавать управление внутрь операторов if, switch и циклов. Нельзя переходить внутрь блоков, содержащих инициализацию переменных, на операторы, расположенные после нее, поскольку в этом случае инициализация не будет выполнена.

**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.

**Оператор return**

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

return [ выражение ];

**17. Операторы. Оператор условия.**

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

Существует две основные формы условной инструкции, встречающиеся в программирование: условный оператор (оператор if) и оператор многозначного выбора (переключатель, case, switch).

**Оператор if:**

*if(выражение){*

*блок операций 1;*

*} else {*

*блок операций 2;*

*}*

Если выражение истинно, то выполняется блок операций 1, иначе выполняется блок операций 2. Блок операций может состоять из одной операции. В этом случае наличие фигурных скобок, ограничивающих блок, необязательно. Ветвь else может отсутствовать, тогда при неверном выражении будет выполняться оператор, следующий за if. Оператор if может быть вложенным. При использовании вложенной формы оператора if опция else связывается с последним оператором if. Если требуется связать опцию else с предыдущим оператором if, внутренний условный оператор заключается в фигурные скобки.

**Оператор switch:**

switch (целое выражение) {

case константа1: блок операций1; break;

case константа2: блок операций2; break;

. . .

case константаn: блок операцийn; break;

default: блок операций по умолчанию; break;

}

Оператор ветвления switch выполняется следующим образом:

* вычисляется целочисленное выражение в скобках оператора

switch;

* полученное значение сравнивается с метками (константами) в опциях case, сравнение производится до тех пор, пока не

будет найдена метка, соответствующая вычисленному значению целочисленного выражения;

* выполняется оператор соответствующей метки case;
* если соответствующая метка не найдена, то выполнится оператор в опции default.

Альтернатива default может отсутствовать, тогда не будет произведено никаких действий.

Опция break; осуществляет выход из оператора switch и переход к следующему за ним оператору. При отсутствии опции break будут выполняться все операторы, начиная с помеченного данной меткой и кончая оператором в опции default.

Константы в опциях case должны быть целого типа (могут быть символами).

**15. Операторы цикла. Цель использования операторов цикла. Итерация. Параметры цикла.**

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела цикла, то есть тех операторов, которые выполняются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

Один проход цикла называется итерацией. Проверка условия выполняется на каждой итерации либо до тела цикла (тогда говорят о цикле с предусловием), либо после тела цикла (цикл с постусловием). Разница между ними состоит в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому возможно, что он не выполнится ни разу.

Переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения, называются параметрами цикла. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации, называются счетчиками цикла.

**18. Операторы. Операторы безусловного перехода.**

Оператор безусловного перехода goto имеет формат:

goto метка;

В теле той же функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

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

Использование оператора безусловного перехода оправдано в двух случаях:

принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;

переход из нескольких мест функции в одно (например, если перед выходом из функции всегда необходимо выполнять какие-либо действия).

В остальных случаях для записи любого алгоритма существуют более подходящие средства, а использование goto приводит только к усложнению структуры программы и затруднению отладки (Даже в приведенных случаях допустимо применять goto только в случае, если в этих фрагментах кода не создаются локальные объекты. В противном случае возможно применение деструктора при пропущенгюм конструкторе, что приводит к серьезным ошибкам в программе). Применение goto нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и один выход.

В любом случае не следует передавать управление внутрь операторов if, switch и циклов. Нельзя переходить внутрь блоков, содержащих инициализацию переменных, на операторы, расположенные после нее, поскольку в этом случае инициализация не будет выполнена.

**19. Операторы. Синтаксис. Семантика.**

Оператор задаёт законченное описание некот. действия. Операторы могут быть объединены в составной оператор. **Составной оператор** - это последовательность операторов, заключ. в фиг скобки.

Операторы бывают **исполняемые** (задают действия над данными) и **неисполняемые**.

Существуют 2 понятия: синтаксис и семантика.

**Синтаксис** - это правила построения элементов языка.

**Семантика** определяет смысл и правила использования элементов языка.

**20. Операции сдвига. Операции отношения.**

**Операции сдвига ( << и >> ):**

Применяются к целочисленным операндам. Они сдвигают двоичное представление первого операнда влево или вправо на количество двоичных разрядов, заданное вторым операндом. При сдвиге влево ( << ) освободившиеся разряды обнуляются. При сдвиге вправо (>>) освободившиеся биты заполняются нулями, если первый операнд беззнакового типа, и знаковым разрядом в противном случае. Операции сдвига не учитывают переполнение и потерю значимости.

**Операции отношения (<, <=, >, >=, ==, !=):**

Сравнивают первый операнд со вторым. Операнды могут быть арифметического типа или указателями. Результатом операции является значение true или false (любое значение, не равное нулю, интерпретируется как true). Операции сравнения на равенство и неравенство имеют меньший приоритет, чем остальные операции сравнения.

**21. Операции увеличения и уменьшения на 1. Операция определения размера. Операции отрицания. Операция деления и остаток от деления.**

**Операции увеличения и уменьшения на 1 (++ и —)**

Эти операции, называемые также инкрементом и декрементом, имеют две формы записи — префиксную, когда операция записывается перед операндом, и постфиксную. В префиксной форме сначала изменяется операнд, а затем его значение становится результирующим значением выражения, а в постфиксной форме значением выражения является исходное значение операнда, после чего он изменяется.

**Операция определения размера sizeof**

Она предназначена для вычисления размера объекта или типа в байтах, и имеет две формы:

sizeof выражение

sizeof ( тип )

**Операции отрицания (-, ! и ~)**

Арифметическое отрицание (унарный минус -) изменяет знак операнда целого или вещественного типа на противоположный. Логическое отрицание (!) дает в результате значение 0, если операнд есть истина (не нуль), и значение 1, если операнд равен нулю. Операнд должен быть целого или вещественного типа, а может иметь также тип указатель. Поразрядное отрицание (~), часто называемое побитовым, инвертирует каждый разряд в двоичном представлении целочисленного операнда.

**Деление (/) и остаток от деления (%)**

Операция деления применима к операндам арифметического типа. Если оба операнда целочисленные, результат операции округляется до целого числа, в противном случае тип результата определяется правилами преобразования. Операция остатка от деления применяется только к целочисленным операндам. Знак результата зависит от реализации.

**23. Операции. Приоритеты операций. Операция определения размера.**

Операции, применяемые к одному операнду, - унарные, к двум операндам – бинарные, есть операция с тремя операндами - тернарная. Операции выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки.

Большинство операций выполняются слева направо. Исключение:

унарные операции, операции присваивания и условная операция (?:) - справа налево.

**Операция определения размера sizeof**

Она предназначена для вычисления размера объекта или типа в байтах, и имеет две формы:

sizeof выражение

sizeof ( тип )

**22. Операции. Поразрядные операции. Логические операции.**

К **поразрядным операциям** относятся: операция поразрядного логического "И" (&), операция поразрядного логического "ИЛИ" (|), операция поразрядного "исключающего ИЛИ" (^).

Операнды поразрядных операций могут быть любого целого типа. При необходимости над операндами выполняются преобразования по умолчанию, тип результата - это тип операндов после преобразования.

Операция поразрядного логического И (&) сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если оба сравниваемых бита единицы, то соответствующий бит результата устанавливается в 1, в противном случае в 0.

Операция поразрядного логического ИЛИ (|) сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если любой (или оба) из сравниваемых битов равен 1, то соответствующий бит результата устанавливается в 1, в противном случае результирующий бит равен 0.

Операция поразрядного исключающего ИЛИ (^) сравнивает каждый бит первого операнда с соответствующими битами второго операнда. Если один из сравниваемых битов равен 0, а второй бит равен 1, то соответствующий бит результата устанавливается в 1, в противном случае, т.е. когда оба бита равны 1 или 0, бит результата устанавливается в 0.

**Логические операции**

Операнды могут иметь арифметический тип или быть указателями, причем операнды могут быть различных типов. Преобразование типов не приводится, каждый операнд сравнивается с точки зрения его эквивалентности нулю. Если значение первого операнда достаточно, чтобы определить результат операции, то второй операнд не выполняется.

В С++ существует три логические операции:

* Логическая операция И(&&)-истина, если истинны оба операнда
* Логическая операция ИЛИ(||)-истина, если один из операндов истина;
* Логическая операция НЕ(!) или логическое отрицание – истина, если операнд ложь и наоборот.

**24. Операции. Унарные. Бинарные и тернарные операции.**

Операции по количеству операндов бывают унарные(один операнд), бинарные(два операнда), тернарная(три операнда).

**Унарные операции:**

:: доступ к области видимости

. выбор

-> выбор

[ ] индексация

( ) вызов функции

<тип>( ) конструирование

++ постфиксный инкремент

— постфиксный декремент

sizeof размер объекта или типа

— префиксный декремент

++ префиксный инкремент

~ поразрядное отрицание

! логическое отрицание

- арифметическое отрицание (унарный минус)

+ унарный плюс

& взятие адреса

\* разадресация

new выделение памяти

delete освобождение памяти

(<тип>) преобразование типа

**Бинарные и тернарная операции:**

\* умножение

/ деление

% остаток от деления

+ сложение

- вычитание

<< сдвиг влево

>> сдвиг вправо

< меньше

<= меньше или равно

> больше

>= больше или равно

== равно

!= не равно

& поразрядная конъюнкция (И)

^ поразрядное исключающее ИЛИ

| поразрядная дизъюнкция (ИЛИ)

**Логические операции:**

&& логическое И

|| логическое ИЛИ

? : условная операция (тернарная)

= присваивание

\*= умножение с присваиванием

/= деление с присваиванием

%= остаток отделения с присваиванием

+= сложение с присваиванием

-= вычитание с присваиванием

<<= сдвиг влево с присваиванием

>>= сдвиг вправо с присваиванием

&= поразрядное И с присваиванием

|= поразрядное ИЛИ с присваиванием

^= поразрядное исключающее ИЛИ с присваиванием

**26. Переменные и выражения.**

**Переменная** — это именованная область памяти, в которой хранятся данные определенного типа. Во время выполнения программы значение переменной может быть изменено. Перед использованием любую переменную необходимо объявить и определить.

Объявление информирует о типе переменной и классе памяти, а определение содержит указание компилятору выделить память в соответствии с типом переменной. При определении переменной может быть присвоено начальное значение — это называется инициализацией.

Для определения переменных применяется оператор описания следующего формата:

[класс памяти] [const] тип имя [инициализатор];

В одном операторе можно описать несколько переменных одного типа, разделяя их запятыми.

Модификатор const показывает, что значение переменной нельзя изменять во время выполнения программы. Такую переменную называют константой. Она должна быть инициализирована при объявлении. Инициализатор можно записывать в двух формах — со знаком равенства или в круглых скобках. Если тип инициализирующего значения не совпадает с типом переменной, выполняются преобразования типа.

Определение переменной, кроме ее типа задает область действия и класс памяти переменной.

Если переменная определена внутри блока, она называется локальной, действует от точки описания до конца блока, включая все вложенности.

Если переменная определена вне любого блока, она называется глобальной и областью ее действия считается файл, в котором она определена.

Если при определении начальное значение переменным явным образом не задается, компилятор присваивает глобальным и статическим переменным нулевое значение.

**Выражения** состоят из операндов, знаков операций, скобок и используются для вычисления некоторого значения определенного типа. Каждый операнд является, в свою очередь, выражением, константой или переменной. Например: (а + 0.12)/6

Операции выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки. Если в одном выражении записано несколько операций одинакового приоритета, то унарные операции, условная операция и операции присваивания выполняются справа налево, остальные — слева направо. Порядок вычисления подвыражений внутри выражений не определен: например, нельзя считать, что в выражении (sin(x + 2) + cos(y + 1)) обращение к синусу будет выполнено раньше, чем к косинусу, и что х + 2 будет вычислено раньше, чем y+1. Результат вычисления выражения характеризуется значением и типом. Например, если а и b — переменные целого типа и описаны так:

*int а = 2, b = 5*;

то выражение а + b имеет значение 7 и тип int, а выражение а = b имеет значение, равное помещенному в переменную а (в данному случае 5) и тип, совпадающий с типом этой переменной. Таким образом, в C++ допустимы выражения вида а = b = с: сначала вычисляется выражение b = с, а затем его результат становится правым операндом для операции присваивания переменной а.

В выражение могут входить операнды различных типов. Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип. Если операнды разного типа, перед вычислениями выполняются преобразования, обеспечивающим преобразование более коротких типов в более длинные для сохранения значимости и точности.

Преобразования бывают двух типов:

изменяющие внутреннее представление величин (с потерей точности или без потери точности);

изменяющие только интерпретацию внутреннего представления.

К первому типу относится, например, преобразование целого числа в вещественное (без потери точности) и наоборот (возможно, с потерей точности), ко второму — преобразование знакового целого в беззнаковое.

В любом случае величины типов char, signed char, unsigned char, short int и unsigned short int преобразуются в тип int, если он может представить все значения, или в unsigned int в противном случае.

После этого операнды преобразуются к типу наиболее длинного из них, и он используется как тип результата. Программист может задать преобразования типа и явным образом.

**27. Поразрядные операции. Логические операции. Операции присваивания.**

К **поразрядным операциям** относятся: операция поразрядного логического "И" (&), операция поразрядного логического "ИЛИ" (|), операция поразрядного "исключающего ИЛИ" (^).

Операнды поразрядных операций могут быть любого целого типа. При необходимости над операндами выполняются преобразования по умолчанию, тип результата - это тип операндов после преобразования.

Операция поразрядного логического И (&) сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если оба сравниваемых бита единицы, то соответствующий бит результата устанавливается в 1, в противном случае в 0.

Операция поразрядного логического ИЛИ (|) сравнивает каждый бит первого операнда с соответствующим битом второго операнда. Если любой (или оба) из сравниваемых битов равен 1, то соответствующий бит результата устанавливается в 1, в противном случае результирующий бит равен 0.

Операция поразрядного исключающего ИЛИ (^) сравнивает каждый бит первого операнда с соответствующими битами второго операнда. Если один из сравниваемых битов равен 0, а второй бит равен 1, то соответствующий бит результата устанавливается в 1, в противном случае, т.е. когда оба бита равны 1 или 0, бит результата устанавливается в 0.

**Логические операции**

Операнды могут иметь арифметический тип или быть указателями, причем операнды могут быть различных типов. Преобразование типов не приводится, каждый операнд сравнивается с точки зрения его эквивалентности нулю. Если значение первого операнда достаточно, чтобы определить результат операции, то второй операнд не выполняется.

В С++ существует три логические операции:

* Логическая операция И(&&)-истина, если истинны оба операнда
* Логическая операция ИЛИ(||)-истина, если один из операндов истина;
* Логическая операция НЕ(!) или логическое отрицание – истина, если

операнд ложь и наоборот.

**Операции присваивания** (=, +=, -=, \*= и т. д.)

Операции присваивания могут использоваться в программе как законченные операторы.

Формат операции простого присваивания (=):

операнд\_1 = операнд\_2

Первый операнд должен быть L-значением, второй — выражением. Сначала вычисляется выражение, стоящее в правой части операции, а потом его результат записывается в область памяти, указанную в левой части (мнемоническое правило: «присваивание — это передача данных «налево»»). То, что ранее хранилось в этой области памяти, естественно, теряется.

В сложных операциях присваивания ( +=, \*=, /= и т п.) при вычислении выражения, стоящего в правой части, используется и L-значение из левой части. Например, при сложении с присваиванием ко второму операнду прибавляется первый, и результат записывается в первый операнд, то есть выражение а += b является более компактной записью выражения а = а + b.

**29. Правило общих арифметических преобразований. Приоритеты операций.**

При выполнении операций производится автоматическое преобразование типов, чтобы привести операнды выражений к общему типу или чтобы расширить короткие величины до размера целых величин, используемых в машинных командах. Выполнение преобразования зависит от специфики операций и от типа операнда или операндов.

Рассмотрим общие арифметические преобразования.

1. Операнды типа float преобразуются к типу double.

2. Если один операнд long double, то второй преобразуется к этому же типу.

3. Если один операнд double, то второй также преобразуется к типу double.

4. Любые операнды типа char и short преобразуются к типу int.

5. Любые операнды unsigned char или unsigned short преобразуются к типу

unsigned int.

6. Если один операнд типа unsigned long, то второй преобразуется к типу

unsigned long.

7. Если один операнд типа long, то второй преобразуется к типу long.

8. Если один операнд типа unsigned int, то второй операнд преобразуется к

этому же типу.

Таким образом, можно отметить, что при вычислении выражений операнды преобразуются к типу того операнда, который имеет наибольший размер.

**Приоритеты:** Операции, применяемые к одному операнду, - унарные, к двум операндам – бинарные, есть операция с тремя операндами - тернарная. 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 условная операция (?:) - справа налево.

**25. Основные типы данных. Спецификаторы типов.**

Иерархию типов данных можно представить следующей схемой.

Простые (скалярные) типы: целые, вещественные, символьные, указатели,

перечислимый тип.

Составные (структурированные) типы: массив, структура, объединение.

Основные (стандартные) типы данных часто называют арифметическими, поскольку их можно использовать в арифметических операциях. Для описания основных типов определены следующие ключевые слова: int (целый); char (символьный); wchar\_t (расширенный символьный); bool (логический); float (вещественный); double (вещественный с двойной точностью).

Первые четыре типа называют целочисленными (целыми), последние два — типами с плавающей точкой. Код, который формирует компилятор для обработки целых величин, отличается от кода для величин с плавающей точкой.

Существует четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов: short (короткий); long (длинный); signed (знаковый); unsigned (беззнаковый).

**28. Правила применения символов и выполнения схем. Описание символов (символы данных, символы процесса, символы линий, специальные символы).**

**Правила применения символов**

• Символ предназначен для графической идентификации функции, которую

он отображает, независимо от текста внутри этого символа.

• 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. Формы символов, установленные настоящим стандартом, должны служить руководством для фактически используемых символов.

• 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 - любая другая

информация. Описание символа должно быть расположено справа над символом.

• В схемах работы системы символы, отображающие носители данных, во

многих случаях представляют способы ввода - вывода. Для использования в качестве ссылки на документацию текст на схеме для символов, отображающих способы вывода, должен размещаться справа над символом, а текст для символов, отображающих способы ввода, - справа под символом.

**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 между а и b, равное 1:2. В пределах одной схемы рекомендуется изображать блоки одинаковых размеров. Все блоки нумеруются.

Линии, соединяющие блоки и указывающие последовательность связей между ними, должны проводится параллельно линиям рамки. Стрелка в конце линии может не ставиться, если линия направлена слева направо или сверху вниз. В блок может входить несколько линий, то есть блок может являться преемником любого числа блоков. 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. В случае, когда одна линия подходит к другой и слияние их явно выражено, точку можно не ставить.

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. В качестве идентификатора, как правило, используется порядковый номер блока, к которому направлена соединительная линия.

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 последовательным доступом

• Запоминающее устройство с прямым доступом(символы данных могут

также указывать вид носителя данных)

• Документ

• Ручной ввод

• Карта

• Бумажная лента

• Дисплей

Символы данных могут как предшествовать, так и следовать за символами процесса. Схема данных начинается и заканчивается символами данных (за исключением специальных символов).

Символы процесса:

Cимволов процесса указывают на процесс, который следует выполнить над данными (символы процесса могут также указывать функции, выполняемые вычислительной машиной);

• 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:

Специальные символы, используются для облегчения написания и чтения схемы.

• Соединитель

• Терминатор

• Комментарий

• Пропуск

Специальные символы, используются для облегчения написания и чтения схемы.

**30. Преобразование типов. Преобразование целых типов со знаком. Преобразование целого типа без знака. Преобразование плавающих типов к целым.**

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

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

unsigned char расширяется нулем;

signed char расширяется знаком;

short, unsigned short и enum при преобразовании не изменяются.

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

Преобразование других типов данных осуществляется следующим образом:

Если один из операндов имеет тип long double, то другой преобразуется к типу long double.

Если один из операндов имеет тип double, то другой преобразуется к типу double.

Если один из операндов имеет тип float, то другой преобразуется к типу float.

Иначе, если один из операндов имеет тип unsigned long, то другой преобразуется к типу unsigned long.

Иначе, если один из операндов имеет тип long, то другой преобразуется к типу long.

Иначе, если один из операндов имеет тип unsigned, то другой преобразуется к типу unsigned.

Иначе оба операнда должны иметь тип int.

Тип результата тот же, что и тип участвующих в выражении операндов.

**31. Приведение типов.**

Приведение типа (type conversion) — преобразование значения переменной одного типа в значение другого типа. Выделяют явное и неявное приведения типов.

При явном приведении указывается тип переменной, к которому необходимо преобразовать исходную переменную.

При неявном приведении преобразование происходит автоматически, по правилам, заложенным в данном языке программирования.

Также в языке могут быть заданы специальные функции для приведения

Для **явного** приведения типов некоторой переменной перед ней следует указать в круглых скобках имя нового типа, например:

*int X; int Y = 200; char C = 30; X = (int)C \* 10 + Y;//переменная С приведена к типу int.*

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение С \* 10 + У переменной Х было бы присвоено значение 640, а не корректное 3200. В результате приведения типа переменная С распознается компилятором как 16-ти разрядная, и описанной выше ошибки не возникает.

**Неясное** приведение возможно в том случае, когда значение преобразуется к более широкому типу, например, целое число к вещественному.

**32. Принципы работы компьютера.**

Компьютер **состоит** из центрального процессора, памяти и устройств ввода-вывода. Компьютер работает под управлением некоторой программы, которая предварительно загружается в память с помощью загрузчика, входящего в состав ОС. Программа обращается к содержимому других ячеек памяти и устройствам ввода-вывода для получения данных и сохранения результатов работы программы.

**Память** компьютера разбита на ячейки-байты (один байт содержит 8 бит), можно обращаться к ячейке по её адресу – номеру байта в памяти. В компьютерах с 32-битным процессором максимальный объем адресуемой памяти равен 4 Гб, а с 64-битным – 192 Гб (зависит от ОС). Данные в памяти хранятся в нескольких последовательных байтах, например, для хранения целых чисел нужно 4 байта, а вещественных – 8 байт.

**Центральный процессор** извлекает очередную команду программы и все необходимые данные для ее исполнения из памяти, выполняет её и записывает при необходимости результаты обратно в память.

Данные для программы могут размещаться в регистровой, статической, стековой и динамической памяти. Отличия между этими видами памяти состоит в способе адресации и времени жизни данных.

**Регистры.**

В центральном процессоре есть небольшое количество регистров (обычно от 4 до 32), которые предназначены для ускорения выполнения команд за счет уменьшения обращений к памяти за данными;

для уменьшения сложности и размеров команд за счет того, что только один из операндов может быть адресом ячейки в памяти.

Форматы машинных команд:

В Си с помощью описателя register можно указать, что некоторую часто используемую переменную следует разместить в регистре. Но данный описатель способа хранения можно считать устаревшим, так как, во-первых, оптимизирующие компиляторы эффективно используют регистры и могут размещать переменную в регистре только на время ее частого использования, а во-вторых, в современных процессорах реализовано многоуровневое кэширование, существенно уменьшающее время обращения к памяти.

**Стек.**

Стек (stack – стопка) – это область памяти, в которую добавление и удаление данных происходит с одной стороны. Регистр SP указывает на вершину стека – первую занятую ячейку. При входе в некоторый блок программы данные добавляются в стек, при выходе из блока – удаляются. Данные, размещаемые в стеке, должны иметь фиксированный размер, так как смещение относительного адреса является константой и вычисляется во время компиляции.

Также стек используется для сохранения значений регистров, например, при вызове подпрограммы адрес команды из регистра PC сохраняется в стеке и в PC записывается адрес вызванной подпрограммы, а возврат из подпрограммы заключается в загрузке вершины стека в регистр PC.

**Куча.**

Куча позволяет во время исполнения программы создавать и уничтожать данные произвольного размера в произвольном порядке. Для повторного использования освобожденной памяти поддерживается список свободных областей памяти, в котором при выделении памяти происходит поиск подходящей по размерам области, а если такой области не будет найдено, запрашивается новая область памяти у операционной системы. Количество действий для этого достаточно велико, поэтому выделение и освобождение памяти из кучи являются медленными операциями.

**33. Приоритет в выражениях. Приоритет операций.**

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

Операции, применяемые к одному операнду, - унарные, к двум операндам – бинарные, есть операция с тремя операндами - тернарная. 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.

**34. Рекомендации при программировании циклических структур.**

Часто встречающиеся ошибки при программировании циклов — использование в теле цикла неинициализированных переменных и неверная запись условия выхода из цикла.

Чтобы избежать ошибок, рекомендуется:

1. проверить, всем ли переменным, встречающимся в правой части

операторов присваивания в теле цикла, присвоены до этого начальные значения (а также возможно ли выполнение других операторов);

2. проверить, изменяется ли в цикле хотя бы одна переменная, входящая в

условие выхода из цикла;

3. предусмотреть аварийный выход из цикла по достижению некоторого

количества итераций;

4. и, конечно, не забывать о том, что если в теле цикла требуется

выполнить более одного оператора, нужно заключать их в фигурные скобки.

Операторы цикла взаимозаменяемы, но можно привести некоторые рекомендации по выбору наилучшего в каждом конкретном случае.

Оператор do while обычно используют, когда цикл требуется обязательно выполнить хотя бы раз (например, если в цикле производится ввод данных).

Оператором while удобнее пользоваться в случаях, когда число итераций заранее не известно, очевидных параметров цикла нет или модификацию параметров удобнее записывать не в конце тела цикла.

Оператор for предпочтительнее в большинстве остальных случаев (однозначно — для организации циклов со счетчиками).

**35. Средства программирования. Этапы получения \*.exe модуля.**

Для того, чтобы выполнить программу требуется перевести её язык на язык, понятный процессору.

1) Включаемые файлы.

2) Исходный код.(\*сpp)

3) Полный текст модуля.

4) Компилятор.

5) Объектный код модуля.(\*.obj)

6) Подключ. библиотеки.(\*.lib)

7) Компоновщик.

8) Исполняемый модуль.(\*.exe)

С помощью текстового редактора разрабатываем программу и сохраняем её в файл.

Сначала программа передаётся препроцессору, который выполняет директивы, содержащиеся в тексте программы. Например: включение в текст программы заголовочных файлов.

В включаемых файлах содержится описание, содержащегося в коде.

Получившийся полный текст программы поступает на вход компилятора, который выделяет лексемы, а затем на основе грамотности языка распознаёт выражение и оператор, построенные из лексем. При этом компиляторы выявляют синтаксические ошибки и в случае их отсутствия строят объектный код. Компоновщик формирует исполняемый модуль файлов, подключая к объектному модулю другие модули, в том числе содержащие функции библиотек, обращение к которым содержится в любой программе. Если программа состоит из несколько исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение \*.exe.

**36. 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.

**37. Строка. Объявление. Инициализация. Действия над строками и элементами строки.**

Строка в СИ – это массив элементов типа char, в конце которого помещен символ '\0' (нуль-терминатор).

Символьные строки состоят из набора символьных констант заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-терминатора, и отводить дополнительный байт под него. Строка при объявлении может быть инициализирована начальным значением, например, так:

char string[10] = "abcdefghf";

Если подсчитать кол-во символов в двойных кавычках после символа равно их окажется 9, а размер строки 10 символов, последнее место отводится под нуль–терминатор, 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).

**39. Строки. Функции puts(), fputs(). Функции gets(), fgets().**

Строка в СИ – это массив элементов типа char, в конце которого помещен символ '\0' (нуль-терминатор).

Для ввода и вывода строк в библиотеке stdio.h содержатся специализированные функции gets и puts.

Функция gets() имеет прототип:

*char \*gets(char \*str);*

где str - это массив символов. Функция gets() возвращает указатель на str.

Функция puts() выводит передаваемый ей аргумент на экран, завершая вывод переходом на новую строку. Она имеет следующий прототип:

*int puts(const char \*str);*

Здесь str - это выводимая строка. Функция возвращает нецелое число в случае удачи и EOF - в случае неудачи.

Прототипы функций fgets(),fputs():

*int fputs(const char \*str, FILE \*указатель файла);*

*char \*fgets(char \*str, int длина, FILE \*увазатель файла);*

Функция fputs() помещает (записывает) в определенный поток строку, на которую указывает str. При наличии ошибки эта функция возвращает EOF.

Функция fgets() извлекает (читает) из определенного потока строку. Делается это до тех пор, пока не будет прочитан символ новой строки или количество прочитанных (извлеченных) символов не станет равным длина - 1. При успешном завершении работы функция возвращает str. В случае ошибки возвращается пустой указатель (null).

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

char \*strcpy (char \*dst, const char \*src); dst — указатель (pointer) на буфер назначения. src — указатель (pointer) на исходную строку. Копировать одну строку в другую. Аналогичная функция strncpy() ограничивает количество копируемых символов. Источник (source) и приемник (destination) являются указателями типа char \* или символьными массивами.

char \*strcat(char \*dest, const char \*scr); - Объединяет исходную строку scr и результирующую строку dest, присоединяя первую к последней. Возвращает dest.

char \*strncat(char \*dest, const char \*scr, int maxlen); - Объединяет maxlen символов исходной строки scr и результирующую строку dest, присоединяя часть первой к последней. Возвращает dest.

char \*strchr(const char \*s, int c); - Ищет в строке s первое вхождение символа c, начиная с начала строки. В случае успеха возвращает указатель на найденный символ, иначе возвращает нуль.

char \*strrchr(const char \*s, int c); - Аналогично предыдущему, только поиск осуществляется с конца строки.

int strcmp(const char \*s1, const char \*s2); - Сравнивает две строки. Возвращает отрицательное значение, если s1<s2; нуль, если s1==s2; положительное значение, если s1>s2. Параметры - указатели на сравниваемые строки.

int stricmp(const char \*s1, const char \*s2); - Аналогично предыдущему, только сравнение осуществляется без учета регистра символов.

int strncmp(const char \*s1, const char \*s2, int maxlen); - Аналогично предыдущему, только сравниваются первые maxlen символов.

int strnicmp(const char \*s1, const char \*s2, int maxlen); - Аналогично предыдущему, только сравниваются первые maxlen символов без учета регистра.

int strcspn(const char \*s1, const char \*s2); - Возвращает длину максимальной начальной подстроки строки s1, не содержащей символов из второй строки s2.

int strlen(const char \*s); - Возвращает длину строки s - количество символов, предшествующих нулевому символу. wcslen использовать для wchar\_t (юникода).

char \*strlwr(char \*s); - Преобразует все прописные (большие) буквы в строчные (малые) в строке s.

char \*strupr(char \*s); - Преобразует все строчные (малые) буквы в прописные (большие) в строке s.

char \*strnset(char \*s, int c, int n); - Заполняет строку s символами c. Параметр n задает количество размещаемых символов в строке.

char \*strpbrk(const char \*s1, const char \*s2); - Ищет в строке s1 первое вхождение любого символа из строки s2. Возвращает указатель на первый найденный символ или нуль - если символ не найден.

char \*strrev(char \*s); - Изменяет порядок следования символов в строке на обратный (кроме завершающего нулевого символа). Функция возвращает строку s.

char \*strset(char \*s, int c); - Заменяет все символы строки s заданным символом c.

int strspn(const char \*s1, const char \*s2); - Вычисляет длину максимальной начальной подстроки строки s1, содержащей только символы из строки s2.

char \*strstr(const char \*s1, const char \*s2); - Ищет в строке s1 строку s2. Возвращает адрес первого символа вхождения строки s2. Если строка отсутствует - возвращает нуль.

char \*strtok(char \*s1, const char \*s2); - Делит исходную строку s1 на лексемы (подстроки), разделенные одним или несколькими символами из строки s2.

**41. Структурное программирование. Основные принципы структурного программирования. Стадии разработки проекта. Критерии качества программ.**

**Структурное программирование** — метод разработки по , в соответствии с которым программа представлена в виде структуры блоков расположенных в определенном порядке.

**Базовые конструкции структурного программирования в C++:** следование, ветвление и цикл.

**Идея структурного программирования**

Как уже было отмечено выше, любую программу можно составить только из структур трех типов: следования, ветвления и цикла (это базовые конструкции).

1. Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных).

2. Ветвление задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

3. Цикл задает многократное выполнение оператора.

**Принципы структурного программирования:**

Принцип абстракции.

Этот принцип позволяет разработчику рассматривать программу в нужный момент без лишней детализации. Детализация увеличивается при переходе от верхнего уровня абстракции к нижнему.

Принцип формальности.

Он предполагает строгий методический подход к программированию, придает творческому процессу определенную строгость и дисциплину

Принцип модульности.

В соответствии с этим принципом программа разделяется на отдельные законченные фрагменты, модули, которые просты по управлению и допускают независимую отладку и тестирование. В результате отдельные ветви программы могут создаваться разными группами программистов.

Принцип иерархического упорядочения.

Взаимосвязь между частями программы должна носить иерархический, подчиненный характер.

**Стадии разработки проекта:**

1. Постановка задачи.

2. Анализ и исследование задачи, модели.

3. Разработка алгоритма.

4. Программирование.

5. Тестирование и отладка.

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

7. Сопровождение программы(внедрение и поддержка).

**Показатели качества должны содержать следующие аспекты:**

Программа является правильной, если она работает в соответствии с техническим заданием (ТЗ - документ, которым завершается постановка задачи).

Программа является точной, если выдаваемые ею числовые данные имеют допустимые отклонения от аналогичных результатов, полученных с помощью идеальных математических зависимостей.

Программа является совместимой, если она работает должным образом не только автономно, но и как часть программной системы.

Программа является надежной, если она при всех входных данных обеспечивает полную повторяемость результатов.

Программа является универсальной, если она правильно работает при любых допустимых вариантах исходных данных. В ходе разработки программ предусматриваются специальные средства защиты от ввода неправильных данных, обеспечивающие целостность системы.

Программа является защищенной, если она сохраняет работоспособность при возникновении сбоев (режим реального времени, программа большого времени выполнения).

Программа является полезной, если задача, которую она решает, представляет практическую ценность.

Программа является эффективной, если объем требуемых для ее работы ресурсов ЭВМ не превышает допустимого предела.

Программа является проверяемой, если ее качества могут быть продемонстрированы на практике (проверка правильности и универсальности). Существуют формальные математические методы проверки и неформальные (прогоны программы с остановками в контрольных точках, обсуждение результатов заинтересованными пользователями).

Программа является адаптируемой, если она допускает быструю модификацию с целью приспособления к изменяющимся условиям функционирования.

**40. Структура программы на языке СИ.**

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

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

Простейшее определение функции имеет следующий формат:

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, следовательно, заключается в фигурные скобки.

Программа может состоять из нескольких модулей (исходных файлов).

**42. Схема обработки многомерного динамического массива.**

Каждый элемент массива может быть, в свою очередь, массивом. Именно так конструируются динамические многомерные массивы. Рассмотрим алгоритм создания и обработки двумерного массива.

1. Определяем указатель на массив указателей, задающий адреса начала

строк матрицы: тип \*\*uk.

2. Вводим размеры матрицы x, y.

3. Создаём динамический массив указателей на указатели начала строк:

uk = new тип \*[x];

4. В цикле выделяем память под x массивов – строк по y элементов в

каждом:

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

uk[i] = new тип \*[y];

5. Обработка массива (работа с индексированными элементами uk[i][j]).

6. В цикле высвобождаем память, занятую под n массивов – строк:

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

delete[ ]uk[i];

7. Высвобождаем память, занятую под массив указателей: delete[ ]uk.

**43. 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, например, операции с деньгами.

Внутреннее представление вещественного числа состоит из 2 частей: мантиссы и порядка. В IBM-совместимых ПК 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.

**44. Указатели и массивы. Одномерный динамический массив. Выделение/высвобождение памяти.**

В языке СИ между указателями и массивами существует тесная связь.

Например, когда объявляется массив в виде int array[25], то этим определяется не только выделение памяти для двадцати пяти элементов массива, но и для указателя с именем array, значение которого равно адресу первого по счету (нулевого) элемента массива, т.е. сам массив остается безымянным, а доступ к элементам массива осуществляется через указатель с именем array. С точки зрения синтаксиса языка указатель arrey является константой, значение которой можно использовать в выражениях, но изменить это значение нельзя.

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

Выделение памяти под одномерный динамичесский массив

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

Освобождение памяти из-под одномерного динамического массива:

delete [ ] mas;

**45. Указатели. Инициализация указателей.**

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

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

Любой указатель необходимо объявить перед использованием, как и любую переменную.

**Инициализация указателей**

объявление указателя

/\*тип данных\*/ \* /\*имя указателя\*/;

Принцип объявления указателей такой же, как и принцип объявления переменных. Отличие заключается только в том, что перед именем ставится символ звёздочки \*. Визуально указатели отличаются от переменных только одним символом. При объявлении указателей компилятор выделяет несколько байт памяти, в зависимости от типа данных отводимых для хранения некоторой информации в памяти. Чтобы получить значение, записанное в некоторой области, на которое ссылается указатель нужно воспользоваться операцией разыменования указателя \*. Необходимо поставить звёздочку перед именем и получим доступ к значению указателя.

**48. Файл math.h, stdlib.h**

Файл math.h, stdlib.h

math.h — заголовочный файл стандартной библиотеки языка программирования С, 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.

**46. Указатели. Операции над указателями.**

Над указателями можно выполнять унарные операции: инкремент и декремент.

При выполнении операций ++ и - значение указателя увеличивается или уменьшается на длину типа, на который ссылается используемый указатель.

Пример:

int \*ptr, a[10];

ptr=&a[5];

ptr++; /\* равно адресу элемента a[6] \*/

ptr--; /\* равно адресу элемента a[5] \*/

В бинарных операциях сложения и вычитания могут участвовать указатель и величина типа int. При этом результатом операции будет указатель на исходный тип, а его значение будет на указанное число элементов больше или меньше исходного.

Пример:

int \*ptr1, \*ptr2, a[10];

int i=2;

ptr1=a(i+4); /\* равно адресу элемента a[6] \*/

ptr2=ptr1-i; /\* равно адресу элемента a[4] \*/

В операции вычитания могут участвовать два указателя на один и тот же тип. Результат такой операции имеет тип int и равен числу элементов исходного типа между уменьшаемым и вычитаемым, причем если первый адрес младше, то результат имеет отрицательное значение.

Пример:

int \*ptr1, \*ptr2, a[10];

int i;

ptr1=a+4;

ptr2=a+9;

i=ptr1-ptr2; /\* равно 5 \*/

i=ptr2-ptr1; /\* равно -5 \*/

Значения двух указателей на одинаковые типы можно сравнивать в операциях ==, !=, <, <=",">, >= при этом значения указателей рассматриваются просто как целые числа, а результат сравнения равен 0 (ложь) или 1 (истина).

Пример:

int \*ptr1, \*ptr2, a[10];

ptr1=a+5;

ptr2=a+7;

if (prt1>ptr2) a[3]=4;

В данном примере значение ptr1 меньше значения ptr2 и поэтому оператор a[3]=4 не будет выполнен

**47. Указатели. Типы указателей.**

Когда компилятор обрабатывает оператор определения переменной, например, int i=10;, он выделяет память в соответствии с типом (int) и инициализирует ее указанным значением (10). Все обращения в программе к переменной по ее имени (i) заменяются компилятором на адрес области памяти, в которой хранится значение переменной. Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями.

Итак, указатели предназначены для хранения адресов областей памяти. В C++ различают три вида указателей:

1. указатели на объект,

2. указатели на функцию

3. указатели на void,

Они отличаются свойствами и набором допустимых операций. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом.

**Указатель на функцию** содержит адрес в сегменте кода, по которому располагается исполняемый код функции, то есть адрес, по которому передается управление при вызове функции. Указатели на функции используются для косвенного вызова функции (не через ее имя, а через обращение к переменной, хранящей ее адрес), а также для передачи имени функции в другую функцию в качестве параметра. Указатель функции имеет тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа»:

тип (\*имя) ( список\_типов\_аргументов );

Например, объявление:

int (\*fun) (double, double);

задает указатель с именем fun на функцию, возвращающую значение типа int и имеющую два аргумента типа double.

**Указатель на объект** содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного). Простейшее объявление указателя на объект (в дальнейшем называемого просто указателем) имеет вид:

тип \*имя;

где тип может быть любым, кроме ссылки и битового поля, причем тип может быть к этому моменту только объявлен, но еще не определен (следовательно, в структуре, например, может присутствовать указатель на структуру того же типа).

Звездочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить ее перед именем каждого из них. Например, в операторе

int \*a, b, \*c;

описываются два указателя на целое с именами а и с, а также целая переменная b.

Размер указателя зависит от модели памяти. Можно определить указатель на указатель и т. д.

**Указатель на void** применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов). Указателю на void можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

**50. Цели структурного программирования. Принципы структурного программирования. Требования к структурным программам.**

Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций:

• последовательное исполнение — однократное выполнение операций в

том порядке, в котором они записаны в тексте программы;

• ветвление — однократное выполнение одной из двух или более

операций, в зависимости от выполнения некоторого заданного условия;

• цикл — многократное исполнение одной и той же операции до тех пор,

пока выполняется некоторое заданное условие (условие продолжения цикла).

В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.

Цель структурного программирования - повышение надежности программ, обеспечение сопровождения и модификации, облегчение и ускорение разработки. уменьшить время и стоимость программной разработки. Достижимо при повышении производительности труда программиста.

К основным принципам структурного программирования относятся:

1) структура создаваемой системы должна соответствовать структуре реальной решаемой задачи.

2) структура обрабатываемых в программе системных данных должна соответствовать структурам реальных данных.

3) модули, образующие программную систему должны составляться по правилам модульного програмирования.

а) текстовый модуль есть последовательность управляющих

программных структур (присваивание, условие – if, цикл и выбор

по ключу)

б) в модуле должны отсутствовать метки и оператор GOTO

в) в модуле должны активно использоваться комментарии на

русском языке.

г) операторы записываемые внутри управляющей структуры должны записываться с отступом в 2-3 позиции.

**49. Функции форматного ввода/вывода. Спецификаторы. Управляющие символы. Управляющая строка. Спецификаторы формата.**

Функция printf() позволяет выводить информацию на экран при программировании в консольном режиме. Данная функция определена в библиотеке stdio.h и имеет следующий синтаксис:

int printf( const char \*format [, argument]... );

Здесь первый аргумент \*format определяет строку, которая выводится на экран и может содержать специальные управляющие символы для вывода переменных. Затем, следует список необязательных аргументов, которые поясняются ниже. Функуция возвращает либо число отображенных символов, либо отрицательное число в случае своей некорректной работы.

В самой простой реализации функция printf() просто выводит заданную строку на экран монитора.

Однако с ее помощью можно выводить переменные разного типа: начиная с числовых и заканчивая строковыми. Для выполнения этой операции используются специальные управляющие символы, которые называются спецификаторами и которые начинаются с символа %.

Следует отметить, что спецификатор «%d» выводит только целочисленные типы переменных, например int. Для вывода других типов следует использовать другие спецификаторы. Ниже перечислены основные виды спецификаторов:

%с символ

%d целое десятичное число

%i целое десятичное число

%e десятичное число в виде x.xx e+xx

%E десятичное число в виде x.xx E+xx

%f десятичное число с плавающей запятой xx.xxxx

%F десятичное число с плавающей запятой xx.xxxx

%g ,%f или %e, что короче

%G,%F или %E, что короче

%o восьмеричное число

%s строка символов

%u беззнаковое десятичное число

%x шестнадцатеричное число

%X шестнадцатеричное число

%% символ %

%p указатель

%n указатель

В спецификаторе формата, после символа % может быть указана точность (число цифр после запятой). Точность задаётся следующим образом: %.n<код формата>. Где n - число цифр после запятой, а <код формата> - один из кодов приведённых выше.

С помощью функции printf() можно выводить сразу несколько переменных.

Кроме спецификаторов в функции printf() используются управляющие символы:

\n Новая строка, перевод строки

\r Возврат каретки

\t Горизонтальная табуляция

\v Вертикальная табуляция

\" Двойная кавычка

\' Апостроф

\\ Обратная косая черта

\0 Нулевой символ, нулевой байт

\a Сигнал такие как перевод строки и др.

Для ввода информации с клавиатуры удобно использовать функцию scanf() библиотеки stdio.h, которая имеет следующий синтаксис:

int scanf( const char \*format [,argument]... );

Здесь, как и для функции printf(), переменная \*format определяет форматную строку для определения типа вводимых данных и может содержать те же спецификаторы что и функция printf(). Затем, следует список необязательных аргументов.

Основным отличием применения функции scanf() от функции printf() является знак & перед именем переменной, в которую записываются результаты ввода.

Функция scanf() может работать сразу с несколькими переменными.

Функция scanf() возвращает число успешно считанных элементов. Если операции считывания не происходило, что бывает в том случае, когда вместо ожидаемого цифрового значения вводится какая-либо буква, то возвращаемое значение равно 0.

**51. Целый тип. Внутреннее представление.**

Размер типа int не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора под величины этого типа 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. При использовании различных спецификаторов можно получить диапазон значений, лежащий в пределах от -2^31 до 2^31-1.

По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор signed можно опускать.

**52. Цикл с параметром (For). Инициализации. Модификации. Выражения.**

Цикл с параметром имеет следующий формат:

for (инициализация; выражение; модификации) оператор;

**Инициализация** используется для объявления и присвоения начальных значений величинам, используемым в цикле. В этой части можно записать несколько операторов, разделенных запятой (операцией «последовательное выполнение»).

Областью действия переменных, объявленных в части инициализации цикла, является цикл. Инициализация выполняется один раз в начале исполнения цикла.

**Выражение** определяет условие выполнения цикла: если его результат, приведенный к типу bool, равен true, цикл выполняется. Цикл с параметром реализован как цикл с предусловием.

**Модификации** выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла. В части модификаций можно записать несколько операторов через запятую. Простой или составной оператор представляет собой тело цикла. Любая из частей оператора for может быть опущена (но точки с запятой надо оставить на своих местах!).

**53. Цикл с постусловием.**

Цикл с постусловием (do while).

Цикл с постусловием имеет вид:

do оператор while выражение;

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение. Если оно истинно, тело цикла выполняется еще раз. Цикл завершается, когда выражение станет равным false или в теле цикла будет выполнен какой-либо оператор передачи управления. Тип выражения должен быть арифметическим или приводимым к нему.

**54. Этапы решения задач на ЭВМ.**

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.

1. Алгоритм. Классы алгоритмов. Способы записи алгоритмов. Свойства алгоритма.

2. Архитектура Фон – Неймана

3. Базовые конструкции структурированного программирования. Цель использования базовых конструкций.

4. Базовые средства языка СИ. Состав языка. Основные элементы алгоритмического языка.

5. Ввод\вывод строки.

6. Двумерный динамический массив. Выделение/высвобождение памяти.

7. Двумерный массив. Инициализация многомерного массива.

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

9. Доступ к элементам массива. Действия над элементами массива.

10. Идентификаторы. Ключевые слова. Константы

11. Классификация языков программирования.

12. Массивы. Объявления. Инициализация.

13. Массивы. Объявления. Работа с указателями.

14. Одномерный массив. N-мерный массив.

15. Операторы цикла. Цель использования операторов цикла. Итерация. Параметры цикла.

16. Операторы. Оператор перехода

17. Операторы. Оператор условия.

18. Операторы. Операторы безусловного перехода.

19. Операторы. Синтаксис. Семантика

20. Операции сдвига. Операции отношения.

21. Операции увеличения и уменьшения на 1. Операция определения размера. Операции отрицания. Операция деления и остаток от деления.

22. Операции. Поразрядные операции. Логические операции.

23. Операции. Приоритеты операций. Операция определения размера.

24. Операции. Унарные. Бинарные и тернарные операции.

25. Основные типы данных. Спецификаторы типов.

26. Переменные и выражения.

27. Поразрядные операции. Логические операции. Операции присваивания.

28. Правила применения символов и выполнения схем. Описание символов (символы данных, символы процесса, символы линий, специальные символы)

29. Правило общих арифметических преобразований. Приоритеты операций.

30. Преобразование типов. Преобразование целых типов со знаком. Преобразование целого типа без знака. Преобразование плавающих типов к целым.

31. Приведение типов.

32. Принципы работы компьютера.

33. Приоритет в выражениях. Приоритет операций.

34. Рекомендации при программировании циклических структур.

35. Средства программирования. Этапы получения \*.exe модуля.

36. Ссылки.

37. Строка. Объявление. Инициализация. Действия над строками и элементами строки.

38. Строки. Библиотека string.h. Прототипы функций для работы со строкой.

39. Строки. Функции puts(),fputs(). Функции gets(),fgets().

40. Структура программы на языке СИ.

41. Структурное программирование. Основные принципы структурного программирования. Стадии разработки проекта. Критерии качества программ.

42. Схема обработки многомерного динамического массива.

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

44. Указатели и массивы. Одномерный динамический массив. Выделение/высвобождение памяти.

45. Указатели. Инициализация указателей.

46. Указатели. Операции над указателями.

47. Указатели. Типы указателей.

48. Файл math.h, stdlib.h

49. Функции форматного ввода/вывода. Спецификаторы. Управляющие символы. Управляющая строка. Спецификаторы формата.

50. Цели структурного программирования. Принципы структурного программирования. Требования к структурным программам.

51. Целый тип. Внутреннее представление.

52. Цикл с параметром (For). Инициализации. Модификации. Выражения.

53. Цикл с постусловием

54. Этапы решения задач на ЭВМ.