1. **История создания ЭВМ**Электро́нно-вычисли́тельная маши́на - комплекс технических, аппаратных и [программных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) средств, предназначенных для автоматической обработки информации, вычислений, [автоматического управления](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F).  
   История создания ЭВМ.

1642 г. – Паскаль придумал механическую счетную машину(суммирование и вычитание 6-8 разрядных чисел)

1670 г. – Лейбниц придумал первый арифмометр (4 арифметических действия).

1834 г. – Бэббидж предложил идею универсальной вычислительной машины с программным управлением. Её структура схожа с структурой современного ЭВМ. Но так и не было реализовано.

1945 г. – создание первой ЭВМ в США.

1970 г. – первые персональные ЭВМ (ПВМ).

**2) Поколения компьютеров**1-е поколение(1940-1945) на электронных лампах

Быстродействие 10-20 кГц

Лампы имели существенные размеры и выделяли тепло в следствии чего перегорали

Одна и таже группа людей и разрабатывали компьютер и его использовали.

Система выполняла одновременно 1 операцию либо ввод-вывод либо обработку.

Высокая стоимость, малая производительность(30 тонн) = компьютер 701 IBM

2-е поколение (1955 - 1965) компьютер на транзисторах

Быстродействие 100-500 кГц

Меньшие размеры, повысилась надежность, теплоотдача снизилась

Появились ЯП высокого уровня – фортран, кабун

Появляются компиляторы, библиотеки математических и служебных программ, возможность параллельных программ.

Персонал поделился на программистов и операторов(загружают программу, запускают на выполнение)

Появляются системы пакетной обработки, которые автоматизировано запускаются программ из пакетов.

Развиваются способы хранения: магнитные ленты, бобины

3-е поколение (1965-1980) компьютеры на интегральных схемах

Быстродействие 1 МГЦ.

Интегральная схема – электронная схема, вытравленная на кристале кремния.

Компьютеры меньше, дешевле, быстрееПроникают в различной деятельности человека – более специализированные

Появилось мультипрограммирование – способ организации, когда на 1 процессоре выполняются попеременно несколько программа = более эффективное использование ресурсов но остались пакетными

Появилась проблема совместимости ПО и систем.

1 семейство программно совместимых ПК – IBM, PDP

ОС 360-IBM( тысячи ошибок)

4-е поколениекомпьютеры на больших интегральных схемах

Большие интегральные схемы содержат более сотни различных элементов.

Можно создать ЭВМ на 1 кристалле

Быстродействие сотни МГц

Цена снижена, покупают отдельные личности, не являющиеся программистами, что приводит к появлению спец ПО

конец 70- начало 80 – популярность пк apple, позднее пк IBM(на процах intel)

Томпсон создал клон Unix – Minix – Linux

Середина 80 – развитие сетевых компьютеров

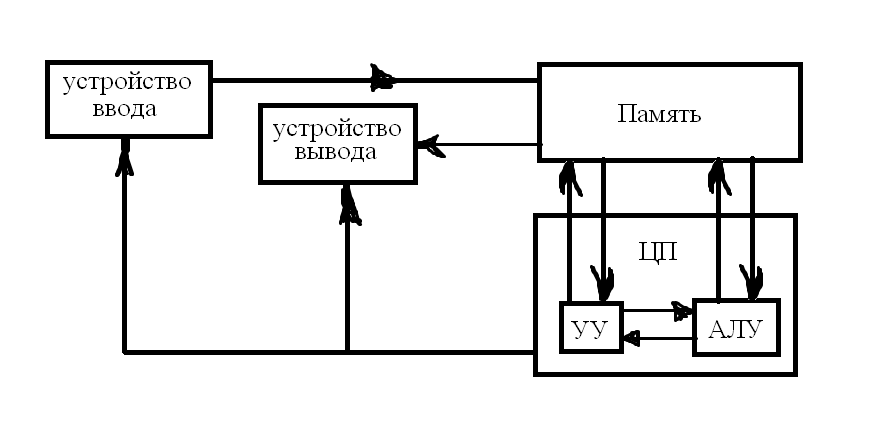
5-е поколение(2005- до н.д.)

Неудачные проекты в ЯПонии

Невидимые компьютеры (встраиваемые)

1.   
    **3) Архитектура Фон – Неймана. Принципы фон Неймана.**Архитектура Фон – Неймана. Принципы фон Неймана.

1945 г. – Джон фон Нейман описал как должен быть устроен компьютер, чтобы он был универсальным.



УУ – устройство управления

АЛУ – арифметическое логическое устройство

Компьютер должен:

* Иметь АЛУ, арифм и логические операции
* Уу организует процесс управления
* Запоминающее устройство хранения программы и данных
* Внешнее устройство ввода вывода информации

Принципы фон Неймана:

1. Использование двоичной системы счисления
2. Программное управление ЭВМ

Работа ЭВМ зависит от программы

1. Память хранит и данные и программы (код в 2-ичной системе счисления)
2. Ячейки памяти имеют адреса которые последовательно пронумерованы(возможность использовать переменные)
3. Возможность условного перехода в процессе выполнения программ(существует возможность перехода к любому участку кода)
4. Программа не является постоянной частью машины, ее можно изменить, аппаратура неизменна.

**4) Языки программирования и их классификация.**  
Компьютеры понимают только очень ограниченный набор инструкций. Чтобы заставить что-то сделать, нужно четко сформулировать задание используя эти инструкции.

Для представления алгоритма в виде понятном компьютеру служат ЯП.

Синтаксис языка – совокупность требований для записи команд.

Семантика языка – смысл каждой команды.

Синтаксическая ошибка – нарушение формы записи программы.

Семантическая ошибка – правильно написанная программа, не отвечающая алгоритмам.

Тестирование – процесс поиска ошибки.

Отладка – процесс устранения ошибок

Классификация ЯП:

По степени абстракции:

* Машинные(машинный язык – очень ограниченный набор инструкций который понимает процессор.)
* Язык мнемонических кодов(Assembler)
* Алгоритмические языки(языки высоких ровней)

По структурному строения программы:

* Процедурно-ориентированный
* Объектно-ориентированный

машинный язык – очень ограниченный набор инструкций, который понимает процессор.

* Каждая команда – последовательность 1 и 0(10110000 01100001 – перемещает число в определенную ячейку памяти)
* Каждый набор бит переводится процессора в набор инструкций разные типы процессоров имеют разный набор инструкций.

Раньше программист должен был полностью владеть архитектурой ЭВМ.

Классифик.:

* Компилируемые
* Интерпретируемые

Транслятор – программа переводчик(например Assembler в машинный код)

**5) Размещение данных и программ в памяти ПЭВМ.**Внутренняя память – устройство, которое хранит информацию необходимую пк в данный момент времени. Во ВП входит ОЗУ, кэш память, ПЗУ.

ОЗУ – энергозависимое быстрое запоминающее устройство сравнительно небольшого объема непосредственно связанного с процессором.

**6) Программные модули. Понятие транслятора, компилятора и интерпретатора.**Исходный модуль - исходный текст программы введенный с помощью клавиатуры в память пк(.cpp в языке С).

Объектный модуль – результат обработки исходного кода компилятором(.obj в С) Не может быть выполнен, так как к нему должны быть присоединены стандартные библиотеки.

Исполняемый модуль – компоновщик, который создает исполняемую программу на основе 1 или нескольких объектных модулей.

Загрузочный модуль – программный модуль, представленный в форме пригодной для загрузки в память и выполнения.

Транслятор – программа переводчик(например Assembler в машинный код)

Компилятор – программа, которая читает код и создает автономную(независимую от другого по) исполняемую программу, которую цп принимает напрямую.

Интерпретатор – программа, которая напрямую выполняет код.(более гибкие чем компиляторы, но менее эффективные, интерпретируется каждый раз при запуске программы.)

**7) Этапы решения задач на ЭВМ.**1)***Постановка задачи***( точная формулировка условий задач с описанием её входных и выходных данных)  
Подзадачи: сбор информации о задаче

2. Формулировка условия задачи

3. Определение конечных целей решения задачи

4. Определение формы выдачи результатов

5. К описанию данных а именно их типов диапазона величин структуры и так далее  
2***) Анализ и исследования задачи под задачу:***

1. Анализ существующих аналогов

2. Анализ технических и программных средств

3. Разработка математической модели

4. Разработка структуры данных.  
3) ***разработка алгоритмов под задачу:***

1. выбор методов проектирования алгоритмов

2. Выбор формы записи алгоритма

3. Выбор тестов и методов тестирования

4. Проектирование алгоритмов  
4***) программирование подзадачи:***

1. Выбор языка программирование

2. Уточнение способов организации данных

3. Запись алгоритмов на выбранном языке  
5)***тестирование и отладка:***

1. Синтаксическая отладка

2. Отладка семантики и логической структуры

3. Тестовые расчёты и анализ результатов расчёта

4. Совершенствования программ  
6)***анализ результатов решения задачи и уточнения в случае*** необходимости математическая модель и повторным выполнением этапов 2-5  
7) ***сопровождение программы подзадачи:***

1. Доработка программ для решения конкретных задач

2. составление документации к решаемым задачам алгоритмом по математическим моделям

Алгоритм. Способы записи алгоритмов

**8) Алгоритм. Способы записи алгоритмов.**Алгоритм - конечная последовательность инструкций, исполнителю в результате выполнения которой обеспечивается получение из входных данных требуемых выходных результатов. Алгоритм должен описываться на формальном языке исполнителя исключающим неоднозначность толкования предписаний. Множество инструкций исполнителя конечно. Запись алгоритма на формальном языке называется программой.

способы описания алгоритмов:

1. словесное описание алгоритмов

2. Псевдокод занимает промежуточное место между словесным и программным описанием

3. Графическое описание алгоритма

4. Алгоритмический язык.  
Графическое описание алгоритма это представление алгоритма в виде схемы состоящей из последовательности блоков геометрические фигуры каждой из которых изображает очередной шаг алгоритма. Внутри фигур кратко описывает действие выполняемое в этом блоке. К правилам изображения фигура сведены в единую систему программной документации ГОСТ 19.701-90.  
схема данных содержит символы данных могут отображать тип носители данных символ процесса которые нужно выполнить над данными символы линий указывающие потоки данных между носителями и процессами специальные символы которые используют для удобства чтения схема

**9) Алгоритм. Свойства алгоритма. Принципы построения алгоритмов при решении сложных задач. Пример алгоритма.**Алгоритм- конечная последовательность инструкций, исполнителю в результате выполнения которой обеспечивается получение из входных данных требуемых выходных результатов. Алгоритм должен описываться на формальном языке исполнителя исключающим неоднозначность толкования предписаний. множество инструкций исполнителя конечно. Запись алгоритма на формальном языке называется программой. Свойства алгоритмов:

1. Дискретность алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов. Для выполнения каждого шага требуется конечный отрезок времени.

2. Определённость каждое правило должно быть чётким и однозначным

3. К результативности алгоритм должен приводить к решению за конечное число шагов.

4. Массовость алгоритм решения задачи разрабатывается в общем виде, то есть он должен быть применим к некоторым у классу задач отличающихся исходными данными.

5. правильность алгоритм (правильный если его выполнение даёт правильные результаты решения поставленной задачи).

Принципы алгоритма при решении сложных задач:

1. Декомпозиция и синтез. Первоначальную разработку алгоритма укрупнённых блоков и их дальнейшая детализация. Синтез - сборка блоков в единую программу.

2. Принцип от главного к второстепенному. Составление алгоритма от главной конструкции

3. Принцип структурирования; использование только типовых алгоритмических структур. используя типовые структуры можно собрать алгоритм любой сложностью. К типам алгоритмических структур относится следование ветвление повторение.

Пример словестного алгоритма:

Решение квадратного уравнения ax^2+bx+c=0

1)вычисления D=b^2-4ac

2)Если D<0, то к шагу 4

3)вычисление x1,2=

4)конец

Пример псевдокода:

Function max(a[1..n])

Max\_i=1

For i = 2 to n do

If a[i]>a[max\_i] then max\_i=i

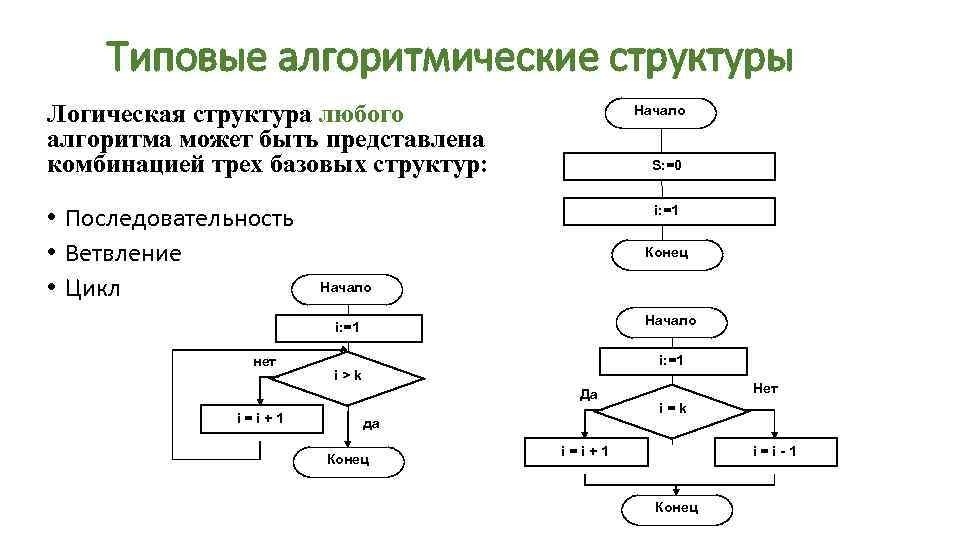
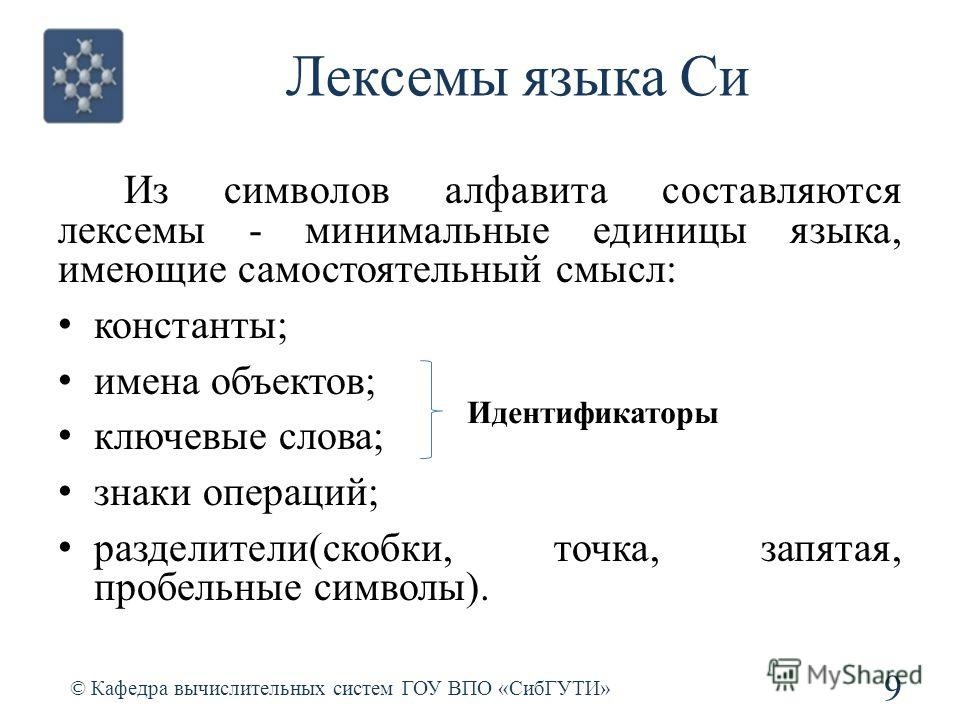
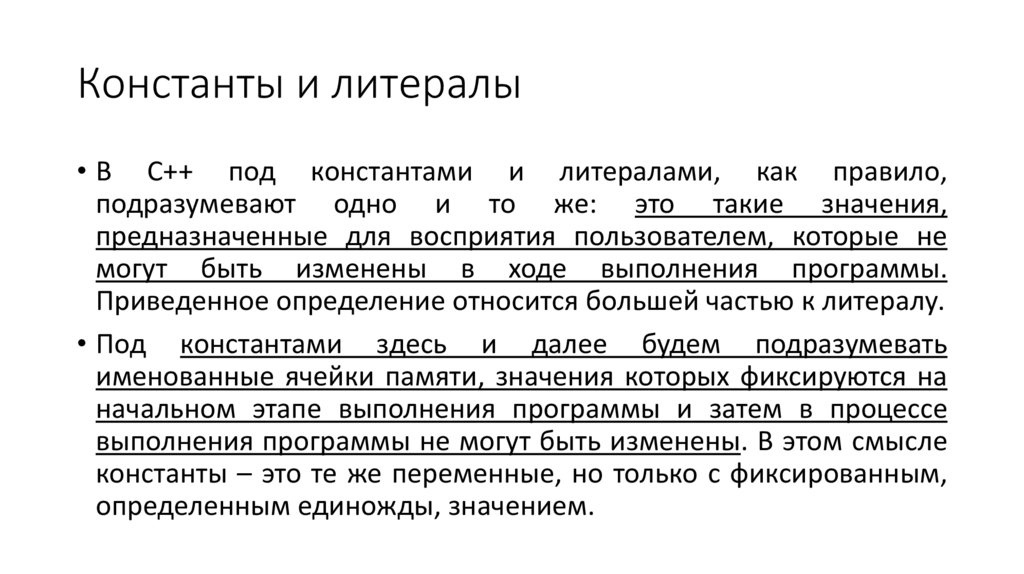
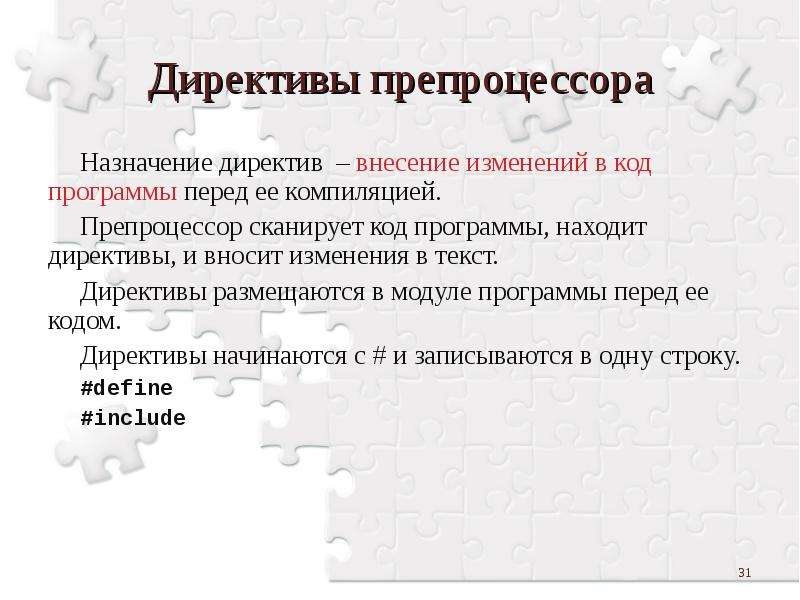
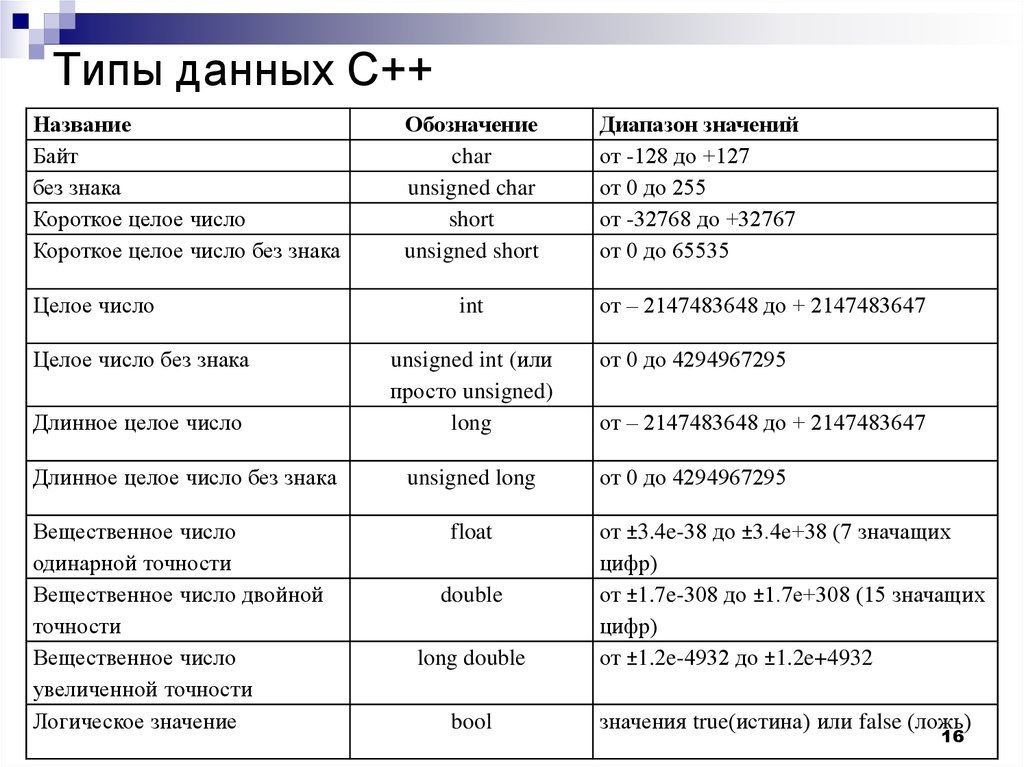
End for

Return max\_i

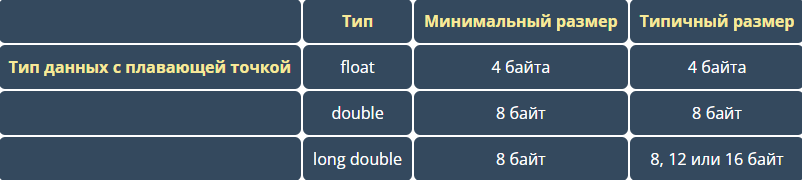
End function

1. **10) Правила применения символов и выполнения блок-схем. Описание символов**Правила применения символов и выполнения блок-схем. Описание символов.

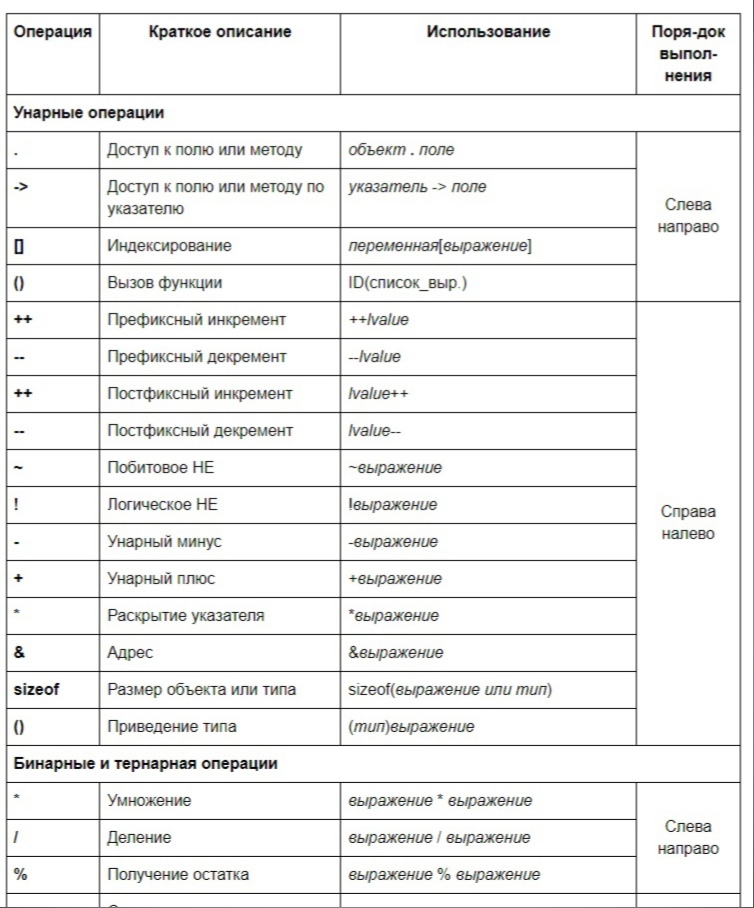
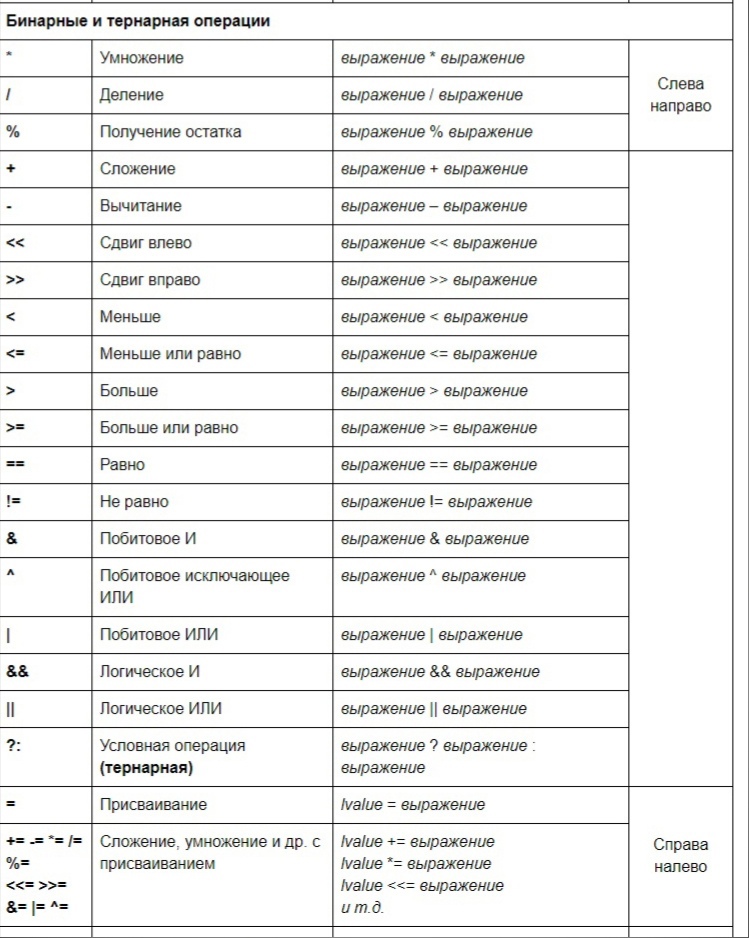
|  |  |
| --- | --- |
| Блок | Пояснение |
|  | Данные ввода-вывода |
|  | Ручной ввод данных с устройства любого типа |
|  | Отображение данных на устройстве |
|  | Процесс операции операции приводящей к изменению значения объекта |
|  | Предопределенный процесс – отображение группы операции которые определены в другом месте |
|  | Функция отображения имеющий один вход и ряд альтернативных выходов из которых 1 активен. |
|  | Граница цикла- отображение начала и конца цикла(в верхнем элементе имя цила, условие, завершения) |
|  | Комментарий |
|  | Терминатор начала/конца |
|  | соединитель |

**11) Типовые алгоритмические структуры.**К типовым алгоритмическим структурам относятся структуры трех типов: - структура следования, в которой блоки действия следуют друг за другом. - логическая структуры, в которой имеется логический блок и несколько параллельно расположенных веток с блоками действия (а точнее структур следования), из которых в зависимости от конкретного значения условия выполняется только одна ветка.  
 **12) Синтаксис языка СИ. Алфавит. Лексемы.** **13)Идентификаторы. Ключевые слова.**Имена, использующиеся для переменных, функций, меток и других определяемых пользователем объектов, называются идентификаторами. Идентификаторы могут состоять как из одного, так и из нескольких символов.  
  
Ключевыми словами называются слова, которые имеют особое значение для компилятора C.  
В языке C используются следующие ключевые слова:  
  
auto do  
break double  
case else  
char extern  
const float  
continue  
default **14) Литералы (константы). Комментарии. Директивы.**Комментарий это примечание написанное для лучшего восприятия программы другими чаще всего используется для описания того что происходит некоторых аспектов кода  
  
Директивы — это специальные команды, которые начинаются с символа # и НЕ заканчиваются точкой с запятой.  
 **15) Структура программы на языке СИ.**Программа на языке Си состоит из одной или более подпрограмм, называемых функциями.  
  
Язык Си является блочно-структурированным. Каждый блок заключается в фигурные скобки {}.  
  
Основным блоком в программе консольного приложения на языке Си является главная функция, имеющая имя main().  
  
Каждое действие в языке Си заканчивается символом «точка с запятой» — ;. В качестве действия может выступать вызов функции или осуществление некоторых операций. **16) Основные типы данных.**К основным типам относят:  
void (пустой тип)  
int (целый тип)  
float (вещественные числа с плавающей точкой)  
double (вещественные числа с плавающей точкой двойной точности)  
char (символьный тип)  
bool - логический **17) Целый тип. Внутреннее представление в языке Си. Переполнение.**Целочисленные типы данных. Целочисленный тип данных — это тип, переменные которого могут содержать только целые числа (без дробной части, например: -2, -1, 0, 1,  
  
Целочи́сленное переполне́ние (англ. integer overflow) — ситуация в компьютерной арифметике, при которой вычисленное в результате операции значение не может быть помещено в n-битный целочисленный тип данных. Различают переполнение через верхнюю границу представления и через нижню

Внутреннее представление величины целого типа — это целое число в двоичном коде

 **18) Типы с плавающей (фиксированной) точкой.**Переменная такого типа может хранить любые действительные дробные значения, например: 4320.0, -3.33 или 0.01226. Почему точка «плавающая»? Дело в том, что точка/запятая перемещается (*«плавает»*) между цифрами, разделяя целую и дробную части значения.  
Есть три типа данных с плавающей точкой: **float**, **double** и **long double**. Язык C++ определяет только их минимальный размер (как и с целочисленными типами). Типы данных с плавающей точкой всегда являются signed (т.е. могут хранить как положительные, так и отрицательные числа).

**19) Объявление объектов.**Декларация (объявление) объектов  
Все объекты (переменные, функции и пр.), с которыми работает программа, в языке Си необходимо декларировать, т.е. объявить компилятору об их присутствии в программе. При этом возможны две формы декларации:  
  
- описание, не приводящее к выделению памяти;  
  
- определение, при котором под объект будет выделен объем оперативной памяти, в соответствии с его типом; в этом случае объект можно сразу инициализировать, т.е. задать его начальное значение.  
  
Кроме констант, которые можно задавать в исходном тексте, все объекты программы должны быть явно декларированы по следующему формату:  
  
<атрибуты> <список ID объектов>;  
  
элементы списка разделяются запятыми, а атрибуты - разделителями. Например: int i,j,k; float a,b;  
  
Объекты программы в общем случае имеют следующие атрибуты:  
  
<класс памяти> - характеристика способа размещения объектов в памяти (статическая, динамическая), определяет область види мости и время жизни переменной (по умолчанию - auto), данные атрибуты будут рассмотрены позже;  
  
<тип> - характеристика механизма интерпретации данных, т.е. это совокупность информации о том, сколько объекту нужно выделить памяти, какой вид имеет представление информации и какие действия над ней допустимы (по умолчанию - int).  
  
Класс памяти и тип – атрибуты необязательные и могут отсутст во вать, тогда их значения установятся по умолчанию.  
  
Примеры декларации простых переменных:  
  
int i,j,k;  
  
char r;  
  
double gfd; **20) Преобразование типов. Приведение типов.**При выполнении операций могут встречаться операнды различных типов. Но для выполнения операции оба операнда должны быть преобразованы к общему типу в соответствии с небольшим набором правил.  
  
Типы операндов преобразуются к общему типу в порядке увеличения их "допустимого диапазона значений". Поэтому неявные преобразования всегда идут от "меньших" объектов к "большим". Схема выполнения преобразований операндов арифметических операций:  
  
short, char → int → unsigned → long → double  
  
float → double  
  
При этом действуют следующие правила:  
  
значения типов char и short всегда преобразуются в int (даже если оба операнда имеют одинаковый тип);  
  
аналогично, значения типа float всегда преобразуются в double;  
  
после этого определяется "старший" из двух типов операндов, и другой операнд преобразуется к этому типу. Порядок "старшинства":  
  
long double  
  
double  
  
unsigned long long  
  
long long  
  
unsigned long  
  
long  
  
unsigned int  
  
int

* В любом выражении преобразование типов может быть осуществлено явно. Для этого достаточно перед любым выражением поставить в скобках идентификатор соответствующего типа.  
    
  Вид записи операции: (тип) выражение;  
    
  Ее результат - значение выражения, преобразованное к заданному типу представления.  
    
  Операция приведения типа вынуждает компилятор выполнить указанное преобразование, но ответственность за последствия возлагаются на программиста. Рекомендуется использовать эту операцию в исключительных случаях.  
    
  Например:  
    
  double x;  
    
  int n=6, k=4;  
    
  1) x=n/k; - дробная часть будет отброшена  
    
  2) x=(double)n/k; - использование операции приведения типа здесь позволяет избежать округления результата деления целочисленных операндов.  
    
  Другой пример:  
    
  int i=100000, j=100000;  
    
  long long k;  
    
  k=(long long)i\*j; - использование операции приведения типа здесь позволяет избежать переполнения типа int.
* **21) Константы в программах (целочисленные, вещественные, символьные, строковые)**Константы - объекты, не подлежащие использованию в левой части оператора присваивания, т.к. константа - является неадресуемой величиной и, хотя она хранится в памяти ЭВМ, обычно нет никакого способа узнать ее адрес. В языке Си константами являются:  
    
  - самоопределенные арифметические, символьные и строковые данные;  
    
  - идентификаторы массивов и функций;  
    
  - элементы перечислений.  
    
  Арифметические константы могут быть целого или вещественного типов.  
  2.4. Целочисленные константы  
  Общий формат: ±n (+ обычно не ставится).  
    
  Десятичные константы - последовательность цифр 0...9, первая из которых не должна быть 0. Например, 22 и 273 - обычные целые константы, если нужно ввести длинную целую константу, то указывается признак L(l) - 273L (273l). Для такой константы будет отведено – 4 байта. Обычная целая константа, которая слишком длинна для типа int , рассматривается как более длинный тип ( long или long long).  
    
  Существует система обозначений для восьмеричных и шестнадцатеричных констант.  
    
  Восьмеричные константы - последовательность цифр от 0 до 7, первая из которых должна быть 0, например: 020 = 16-десятичное.  
    
  Шестнадцатеричные константы - последовательность цифр от 0 до 9 и букв от A до F (a...f), начинающаяся символами 0Х (0х), например: 0X1F (0х1f) = 31-десятичное.  
    
  Восьмеричные и шестнадцатеричные константы могут также заканчиваться буквой L(l) - long, например, 020L или 0X20L.  
    
  Примеры целочисленных констант:  
    
  1992 13 1000L - десятичные;  
    
  0777 00033 01l - восьмеричные;  
    
  0x123 0X00ff 0xb8000l - шестнадцатеричные.  
    
  2.5. Константы вещественного типа  
  Данные константы размещаются в памяти по формату double, а во внешнем представлении могут иметь две формы:  
    
  1) с фиксированной десятичной точкой, формат записи: ±n.m, где n, m - целая и дробная части числа;  
    
  2) с плавающей десятичной точкой (экспоненциальная форма): ±n.mE±p, где n, m - целая и дробная части числа, р - порядок, например, 1,25⋅10-8 записывается как 1.25E-8.  
    
  Примеры констант с фиксированной и плавающей точками:  
    
  1.0 -3.125100е-10 0.12537е+13  
  2.6. Символьные константы  
  Символьная константа - это символ, заключенный в одинарные кавычки: 'A', 'х' (занимает 1 байт).  
    
  В языке Си используются и. специальные (управляющие) символы, не отображаемые на экране; их назначение - влиять на порядок изображения других символов.. Поскольку они не отображаются на экране, для их обозначения в тексте программы используется пара символов, первый из которых всегда - обратная косая черта (обратный слеш) ("\"). Основные их них:  
    
  \n  
    
  - новая строка;  
    
  \t  
    
  - горизонтальная табуляция;  
    
  \0  
    
  - нулевой символ (нуль-терминатор).  
    
  С помощью обратного слеша в символьных и строковых (см. ниже) константах представляются и некоторые обычные символы, чье написание там могло бы привести к двусмысленности:  
    
  \\  
    
  - сам обратный слеш;  
    
  \'  
    
  - апостроф;  
    
  \"  
    
  - кавычки.  
    
  При присваивании символьной переменной эти последовательности должны быть заключены в апострофы. Символьная константа '\0' (не путать с символом - цифрой '0' !) часто записывается вместо целой константы 0, чтобы подчеркнуть символьную природу некоторого выражения (см. тему "Строки").  
    
  Примеры символьных констант: 'А', '9', '$', '\n', '\"'.  
    
  2.7. Строковые константы  
  Строковая константа представляет собой последовательность символов кода ASCII, заключенная в кавычки (”) . Во внутреннем представлении к строковым константам добавляется нулевой символ '\0', еще называемый нуль-терминатор, отмечающий конец строки. Кавычки не являются частью строки, а служат только для ее ограничения. Строка - это массив, состоящий из символов. Внутреннее представление константы "01234\0ABCDEF":  
    
  '0','1','2','3','4','\0','A','B','C','D','E','F','\0'  
    
  Примеры строковых констант:  
    
  "Система", "\n\t Аргумент \n", "Состояние \"WAIT\""  
    
  В конец строковой константы компилятор автоматически помещает нуль-символ (нуль-терминатор). Нуль-символ - это не цифра 0, он на печать не выводится и в таблице кода ASCII имеет код 0.  
    
  Например, строка "" - пустая строка, содержащая лишь нуль-терминатор.
* 2.8. Логические константы  
  Логические выражения (тип bool) могут принимать лишь два значения, которые задаются логическими константами true (истина) и false (ложь).  
    
  2.9. Именованные константы  
    
  Рассмотренные выше константы не имеют имен. Но иногда одну и ту же постоянную (т.е. не меняющуюся в ходе работы программы) величину приходится использовать в тексте программы многократно. Тогда может быть удобно назвать ее именем, чтобы в дальнейшем обращаться к ней по этому имени. Значение же ее будет указываться лишь в одном месте - при ее объявлении, и для его изменения необходимо лишь единственное исправление в тексте программы.  
    
  Объявление такой именованной константы пишется так же, как и объявление переменной, но перед ее типом указывается слово const :  
    
  const double pi=3.14159  
    
  Далее ее можно использовать в выражениях, указывая ее имя, например:  
    
  double x=2\*pi;  
    
  Таким образом, именованная константа выглядит, как переменная, но ее значение нельзя менять в процессе работы программы. Зато ее можно использовать там, где разрешается использовать только константы. В языке С++ в сложных программах, разбитых на модули, употребление именованных констант часто считается предпочтительнее, чем директива [#define](https://vk.com/im?q=%23define).
* **22) Операции и выражения. Унарные, бинарные и тернарные операции. Приоритет в выражениях. Приоритет операций.**Операции языка Си предназначены для управления данными (более 40). Для их использования необходимо знать:  
    
  - синтаксис;  
    
  - приоритеты (15 уровней);  
    
  - порядок выполнения.  
    
  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. Для изменения порядка выполнения операций используются круглые скобки.  
    
  Большинство операций выполняются слева направо, например, a+b+c → (a+b)+c. Исключение: унарные операции, операции присваивания и условная операция (?:) - справа налево.  
    
  Полный список операций в соответствии с их приоритетом приводится ниже, в § 4.7.  
    
    
  
* **23) Операции и выражения. Арифметические операции. Операция присваивания и ее сокращенная запись.**Операции языка Си предназначены для управления данными (более 40). Для их использования необходимо знать:  
    
  - синтаксис;  
    
  - приоритеты (15 уровней);  
    
  - порядок выполнения.  
    
  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. Для изменения порядка выполнения операций используются круглые скобки.  
    
  Большинство операций выполняются слева направо, например, a+b+c → (a+b)+c. Исключение: унарные операции, операции присваивания и условная операция (?:) - справа налево.  
    
  Полный список операций в соответствии с их приоритетом приводится ниже, в § 4.7.  
    
  Рассмотрим кратко основные операции языка Си.  
  приоритет  
  НОВОЕ  
  3.2. Арифметические операции  
  Перечень арифметических операций и их обозначений:  
    
  + - сложение;  
    
  - - вычитание (либо унарная операция - изменение знака);  
    
  \* - умножение;  
    
  / - деление (для int операндов - с отбрасыванием остатка);  
    
  % - остаток от деления целочисленных операндов, со знаком первого операнда (деление по модулю).  
    
  Примеры:  
    
  A=23%5; // A=3  
    
  B=23/5; // B=4  
    
  C=1/3.; // C=0.3333333333333333 (без точки было бы C=0)  
    
  Операндами традиционных арифметических операций (+ - \* /) могут быть константы, переменные, элементы массивов, любые арифметические выражения.  
    
  Порядок выполнения операций:  
    
  выражения в круглых скобках;  
    
  функции (стандартные математические, функции пользователя);  
    
  \* / % выполняются слева направо;  
    
  + ─ слева направо.  
    
  Порядок выполнения операций можно определять круглыми скобками, тогда выражение в скобках выполняется в первую очередь (слева направо).  
    
  Унарные операции + и – обладают самым высоким приоритетом, определены только для целых и вещественных операндов; «+» не выполняет никаких действий (введен лишь для полноты), а результатом «–» является значение операнда с противоположным знаком.  
    
  A=+b;  
    
  C=-d;  
    
  Таким образом, так как операции \*, /, % обладают высшим приоритетом над операциями +, -, при записи сложных выражений нужно использовать общепринятые математические правила:  
    
  x+y\*z-a/b ⇔ x+(y\*z)-(a/b)  
  3.5. Операция присваивания  
  Формат операции присваивания:  
    
  Операнд\_1 = Операнд\_2  
    
  Операндом\_1 может быть только переменная. Этот (левый) операнд операции присваивания получил название L–значение, (L–value, Left–value). Операндом\_2 может быть любое выражение, составленное в соответствии с синтаксисом языка Си. Этот (правый) операнд операции присваивания назвали R–значение, (R–value, Right–value).  
    
  Операция присваивания может быть как отдельным оператором (тогда после нее ставится « ; »), так и частью другого выражения (тогда ее результатом считается присвоенное значение). Если в выражении подряд без скобок идут несколько операций « = », то они выполняются справа налево, например:  
    
  int i, j, k;  
    
  float x, y, z;  
    
  ...  
    
  i = j = k = 0; ↔ k = 0, j = 0, i = 0;  
    
  x = i + (y = 3) – (z = 0); ↔ z = 0, y = 3, x = i + y – z;  
    
  Примеры недопустимых выражений:  
  – присваивание константе: 2 = x + y;  
    
  – присваивание функции: getch() = i;  
    
  – присваивание результату операции: (i + 1) = 2 + y;  
    
  3.6. Сокращенная запись операции присваивания  
  В языке Си используются два вида сокращенной записи операции присваивания:  
    
  1) вместо записи:  
    
  v = v # e;  
    
  где # – любая арифметическая или битовая операция, рекомендуется использовать запись v #= e;  
    
  Например,  
    
  i = i + 2; ↔ i += 2; («+=» – без пробелов);  
    
  2) вместо записи:  
    
  x = x # 1;  
    
  где # означает «+» либо «-» , x – переменная одного из целочисленных типов (или переменная-указатель), рекомендуется использовать запись:  
    
  #[#x](https://vk.com/im?q=%23x); – префиксную, или x##; – постфиксную.  
    
  Если эти операции используются в отдельном виде, то различий между постфиксной и префиксной формами нет. Если же они используются в выражении, то в префиксной форме (#[#x](https://vk.com/im?q=%23x)), сначала
* значение x изменится на 1, а затем будет использовано в выражении; в постфиксной форме (x##) – сначала значение используется в выражении, а затем изменяется на 1. (Операции над указателями будут рассмотрены позже.)  
    
    
    
  Рекомендации использования сокращений обоснованы возможностью оптимизации программы (ускорение в работе программы), т.к. схема выражения вида v #= e соответствует схеме выполнения многих машинных команд типа "регистр-память".

**24) Операции и выражения. Операции сравнения. Логические операции. Операция запятая.**Операции языка Си предназначены для управления данными (более 40). Для их использования необходимо знать:  
  
- синтаксис;  
  
- приоритеты (15 уровней);  
  
- порядок выполнения.  
  
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 сравнения**  
== - равно или эквивалентно;  
  
!= - не равно;  
  
< - меньше;  
  
<= - меньше либо равно;  
  
> - больше;  
  
>= - больше либо равно.  
  
Пары символов соответствующих операций разделять нельзя.  
  
Общий вид операций отношений:  
  
<выражение1> <знак\_операции> <выражение2>  
  
Общие правила:  
  
- операндами могут быть любые базовые (скалярные) типы;  
  
значения операндов перед сравнением преобразуются к одному типу;  
  
- результат операции отношения - целое значение 1, если отношение истинно, или 0 в противном случае. Следовательно, операция отношения может использоваться в любых арифметических выражениях.  
3.4. Логические операции  
Перечень логических операций в порядке убывания относительного приоритета и их обозначения:  
  
! - отрицание (логическое НЕТ);  
  
&& - коньюнкция (логическое И);  
  
|| - дизьюнкция (логическое ИЛИ).  
  
Общий вид операции отрицания:  
  
!<выражение>  
  
Примеры: x=!a ; !(a==b)  
  
Общий вид операций коньюнкции и дизьюнкции  
  
<выражение1> <знак\_операции> <выражение2>  
  
Например:  
  
y>0 && x==7 → истина, если 1-е и 2-е выражения истинны;  
  
e>0 || x==7 → истина, если хотя бы одно выражение истинно.  
  
Ненулевое значение операнда трактуется как "истина", а нулевое - "ложь".  
  
Например:  
  
!0 → 1  
  
!5 → 0  
  
x=10; y=10; !((x==y)>0) → 0  
  
a=(b==c)+(c==d)+(d==b); // В переменную a записывается количество совпадений между значениями b,c,d  
  
Пример правильной записи двойного неравенства:  
  
0<x<100 → 0<x&&x<100  
  
Особенность операций коньюнкции и дизьюнкции – экономное последовательное вычисление выражений-операндов:  
  
<выражение1> <операция><выражение2>,  
  
- если выражение1 операции коньюнкция ложно, то результат операции - ноль и выражение2 может не вычисляться;  
  
- если выражение1 операции дизьюнкция истинно, то результат операции - единица и выражение2 может не вычисляться.  
  
Таким образом, появляется возможность записью логического выражения задать условную последовательность вычисления выражений в направлении слева направо:  
  
scanf("%d",&i) && test1(i) && test2(i) → нулевой результат одной из функций может привести к игнорированию вызова остальных;  
  
+search1(x) || search2(x) || search3(x) → только ненулевой результат одной из функций может привести к игнорированию вызова остальных.  
3.7. Операция «,» (запятая)  
Данная операция используется для организации заданной последовательности вычисления выражений (обычно используется там, где по синтаксису допустима только одна операция, а необходимо разместить две и более, например, в операторе for). Форма записи:  
  
выражение\_1, …, выражение\_N;  
  
выражения 1, 2,…, N вычисляются последовательно друг за другом и результатом операции становится значение последнего выражения N, например:  
  
m = ( i = 1, j = i ++, k = 6, n = i + j + k );  
  
+получим последовательность вычислений: i =1, j =i =1, i =2, k=6, n=2+1+6, и в результате m=n=9.  
 **25) Операции и выражения. Побитовые логические операции. Операции над битами.**Операции и выражения. Побитовые логические операции. Операции над битами.

Операции языка Си предназначены для управления данными . Для их использования

необходимо знать:

- синтаксис;

- приоритеты (15 уровней);

- порядок выполнения.

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.

Побитовые логические операции это набор операции для работы с битами.

~ дополнение ( унарное).(меняет 1 на 0 и наоборот)

& побитовое и.

| Побитовое или.

^ побитовое исключающее или.

>> Сдвиг вправо

<< Сдвиг влево.(новые биты заполняются 0)

Бинарные операции могут использоваться и в сокращенной форме.  
 **26) Стандартные математические функции. Файлы math.h, stdlib.h**Математические вычисления не ограничиваются лишь арифметическими действиями. Кроме них, можно ещё встретить корни, модули, логарифмы, тригонометрические функции и пр. Научимся же использовать подобные функции в своих программах.  
Для использования математических функций нужно подключить заголовочный файл **math.h**. В ней определено много различных функций, но мы пока рассмотрим следующие:Некоторые математические функции

**fabs(x)** модуль числа x  
**sqrt(x)** квадратный корень из числа x  
**sin(x)** синус числа x (х в радианах)  
**cos(x)** косинус числа x (х в радианах)  
**pow(x, y)** вычисление xy  
**exp(x)** вычисление ex  
**log(x)** натуральный логарифм числа x  
**log10(x)** десятичный логарифм числа x

Два важных момента.

* Все функции возвращают значение типа **double**.

Параметры функций – вещественные числа(**double**), но можно передавать и целые числа. При этом произойдёт неявное преобразование типа. Компилятор из целого числа, например 3, сделает вещественное 3.0.  
  
Stlib.h - Этот заголовочный файл определяет несколько функций общего назначения, в том числе - функции динамического управления памятью, генерации случайных чисел, целой арифметики, поиска, сортировки и преобразования типов данных.  
Расшифровывается как стандартная библиотека ( standard library )  
  
  
  
  
  
  
**27. Функции форматного ввода. Управляющая строка. Управляющие символы. Спецификаторы формата.**Ввод и вывод информации осуществляется через функции стандартной библиотеки. Прототипы рассматриваемых функций находятся в файле stdio.h. Эта библиотека содержит функции

* printf() — для вывода информации
* scanf() — для ввода информации.

Функция форматированного ввода данных с клавиатуры scanf() выполняет чтение данных, вводимых с клавиатуры, преобразует их во внутренний формат и передает вызывающей функции.Общая форма записи функции scanf( ): **scanf ("CтрокаФорматов", адрес1, адрес2,...);**Строка форматов аналогична функции printf().  
Для формирования адреса переменной используется символ амперсанд ‘&’:  
адрес = &объект  
  
***Управляющие символы*** не выводятся на экран, а управляют расположением выводимых символов. Отличительной чертой управляющего символа является наличие обратного слэша ‘\’ перед ним.  
  
Основные управляющие символы:

* ‘\n’ — перевод строки;
* ‘\t’ — горизонтальная табуляция;
* ‘\v’ — вертикальная табуляция;
* ‘\b’ — возврат на символ;
* ‘\r’ — возврат на начало строки;
* ‘\a’ — звуковой сигнал.

***Форматы*** нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент ‘%’ перед ним:  
  
%d — целое число типа int со знаком в десятичной системе счисления;  
%f — вещественный формат (числа с плавающей точкой типа float);  
%x — целое число типа int со знаком в шестнадцатеричной системе счисления;

%s — строковый формат  
%c — символьный формат;

**28. Функции форматного вывода. Управляющая строка. Спецификаторы формата.**Функция printf() предназначена для форматированного вывода. Она переводит данные в символьное представление и выводит полученные изображения символов на экран.  
 **printf("СтрокаФорматов", объект1, объект2, ..., объектn);  
  
  
  
*Форматы*** нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент ‘%’ перед ним:  
  
%d — целое число типа int со знаком в десятичной системе счисления;  
%f — вещественный формат (числа с плавающей точкой типа float);  
%x — целое число типа int со знаком в шестнадцатеричной системе счисления;

%s — строковый формат  
%c — символьный формат;

**29) Категории операторов. Синтаксис и семантика операторов. Пустой оператор. Составной оператор.**Оператор – это наименьшая автономная часть языка программирования, команда.  
  
Все операторы языка СИ разделены на категории:

* условные операторы, к которым относятся оператор условия if и оператор выбора switch;
* операторы цикла (for,while,do while);
* операторы перехода (break, continue, return, goto);
* Логические операторы (И, ИЛИ, НЕ);
* другие операторы (оператор “выражение”, пустой оператор);
* Составные операторы.

Синтаксис операторов:  
  
У **if:**  
if(условие){  
 блок операций 1;

} else {

блок операций 2;}

У **switch:**  
switch (целое выражение) {

case константа1: блок операций1;

break;

case константа2: блок операций2;

break;  
  
У **for:**  
for ( выражение 1 ; выражение 2 ; выражение 3 ) тело  
  
Пример:  
int main()

{ int i,b;

for (i=1; i<10; i++)

b=i\*i;

return 0;

}  
  
Выражение 1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение 2 – это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.  
  
У **while:**

while (выражение) тело ;

Выражение в скобках может принимать ненулевое (истинное) или нулевое (ложное) значение. Если оно истинно, то выполняется тело цикла и выражение вычисляется снова.  
  
Пустой оператор - это оператор, состоящий только из точки с запятой. Он может появиться в любом месте программы, где по правилам синтаксиса требуется оператор. Выполнение пустого оператора не меняет состояния программы.

Пример:

for(i = 0; i < 10; line[i++] = 0);

Для таких операторов, как **do**, **for**, **if**, **while**, требуется, чтобы в их теле был хотя бы один оператор. Пустой оператор удовлетворяет требованиям синтаксиса в случаях, когда никаких действий не требуется.  
  
Составной оператор представляет собой несколько операторов и объявлений, заключенных в фигурные скобки:

{ [oбъявление]

:

оператор; [оператор];

:

}

Выполнение составного оператора заключается в последовательном выполнении составляющих его операторов.

**30. Операторы цикла. Цель использования операторов цикла. Итерация. Параметры цикла.  
*Циклом*** называется блок кода, который для решения задачи требуется повторить несколько раз. Цикл выполняется до тех пор, пока блок проверки условия возвращает истинное значение.  
  
В языке Си следующие виды циклов:

* while — цикл с предусловием;
* do…while — цикл с постусловием;
* for — параметрический цикл (цикл с заданным числом повторений).  
    
  Циклы нужны для экономии памяти отводимой под програмные инструкции при необходимости повторять одни и те же действия много раз. Так же это удобно.  
    
  Итерация - организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя

Параметр цикла – переменная , управляющая выполнением цикла. Параметр цикла изменяет свое значение при каждом проходе тела цикла. **31. Условные операторы. Тернанрая операция.**Условные операторы позволяют выполнить проверку какого-либо условия и в зависимости от результата выполнить определенный код в программе.Конструкция **if - else**   
За счёт if else можно проверить одно или несколько условий и в случае их успешной проверки будет выполнен один, иначе другой.  
  
if (Условие)  
{  
  БлокОпераций1;  
}  
else  
{  
  БлокОпераций2;}  
  
  
Тернарная операцияЭто сокращенная форма if else. Записывается следующим образом:  
  
**int x = 90, a = 8;  
int res = x < a ? (x + a) : (x - a);  
printf("%d\n", res);  
  
32. Оператор выбора альтернатив switch. Операторы передачи управления (goto, continue, break, return)**Оператор *switch* (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Общий вид оператора:

***switch*** ( ***выражение*** )

{

***case*** ***константа*1**: *список операторов* 1

***case*** ***константа*2**: *список операторов* 2  
  
Выполнение оператора начинается с вычисления ***выражения***, значение которого должно быть целого или символьного типов. Это значение сравнивается со значениями ***констант***, и используется для выбора ветви, которую нужно выполнить.  
  
В данной конструкции ***константы*** фактически выполняют роль меток. Если значение выражения совпало с одной из перечисленных констант, то управление передается в соответствующую ветвь.

**33. Массивы. Объявления. Инициализация. Доступ к элементам массива. Действия над элементами массива.  
  
  
*Массив*** — это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем. ***Элемент массива (значение элемента массива)*** – значение, хранящееся в определенной ячейке памяти, расположенной в пределах массива, а также адрес этой ячейки памяти.  
  
Каждый элемент массива характеризуется тремя величинами:

* адресом элемента — адресом начальной ячейки памяти, в которой расположен этот элемент;
* индексом элемента (порядковым номером элемента в массиве);

значением элемента.  
  
Для объявления массива в языке Си используется следующий синтаксис:

**тип имя[размерность]={инициализация};**  
  
***Инициализация*** представляет собой набор начальных значений элементов массива, указанных в фигурных скобках, и разделенных запятыми.

**int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // массив a из 10 целых чисел**Если количество инициализирующих значений, указанных в фигурных скобках, меньше, чем количество элементов массива, указанное в квадратных скобках, то все оставшиеся элементы в массиве (для которых не хватило инициализирующих значений) будут равны нулю.

**34. Одномерный, двумерный и многомерный массив.**В программе **одномерный массив** объявляется следующим образом:

***тип*** ***ID\_массива*** [***размер***] = {список начальных значений};

*тип* – тип элементов массива;  
*размер* – количество элементов в массиве.  
  
*Список начальных значений* используется при необходимости задать начальные значения (инициализировать элементы массива), он может отсутствовать.  
  
Примеры объявления массива:

int a[6];

double x[5]={2, 3.5, -4.5e3, 2.34, -.7 };

+Если в группе {…} список значений короче, то оставшимся элементам присваивается 0.  
  
Обращение к конкретному элементу массива в программе осуществляется путем записи имени массива и за ним в квадратных скобках - номера элемента, например:

a[0]=1;

a[i]++;

a[3]=a[i]+a[i+1];

***Индексы массивов в языке Си начинаются с 0*Многомерные массивы** отличаются от одномерных только тем, что каждый элемент характеризуется не одним, а двумя или более индексами. Они используются, например, для хранения таблиц (каждый элемент таблицы имеет 2 индекса - № строки и № столбца).

Декларация многомерного массива имеет следующий формат:

*тип* *ID*[*размер*1] [*размер*2]…[*размерN*] =

{ {*список начальных значений*},

{*список начальных значений*},

…};

Списки начальных значений – атрибут необязательный  
  
Пример:

int a[3][3]={ {1,2,3}, {4,5,6}, {7,8,9} };

**35)Указатели. Инициализация указателей. Примеры работы. Область применения.  
  
Указатели** часто используются при работе с массивами.  
**Указатель** – это переменная, которая может содержать адрес некоторого объекта. Простейшая декларация указателя имеет формат:

***тип* \* *ID\_указателя*;**

### Например: int \*a; double \*f; char \*w; **Способы инициализации указателей**

Присваивание указателю адреса существующего объекта:

с помощью операции получения адреса:  
int а = 5; // целая переменная  
int\* р = &а; //в указатель записывается адрес а  
int\* р (&а); // то же самое другим способом  
  
с помощью значения другого инициализированного указателя:  
int\* r = р;  
  
  
  
  
  
  
  
  
  
Два основных оператора для работы с указателями – это оператор & взятия адреса, и оператор \* разыменования. Рассмотрим простой пример.  
  
#include <conio.h>

#include <stdio.h>

void main() {

    int A = 100;

    int \*p;

    //Получаем адрес переменной A

    p = &A;

    //Выводим адрес переменной A

    printf("%p\n", p);

    //Выводим содержимое переменной A

    printf("%d\n", \*p);

    //Меняем содержимое переменной A

    \*p = 200;

    printf("%d\n", A);

    printf("%d", \*p);

    getch();

}

Рассмотрим код внимательно, ещё раз

|  |  |
| --- | --- |
|  | int A = 100; |

Была объявлена переменная с именем *A*. Она располагается по какому-то адресу в памяти. По этому адресу хранится значение 100.

|  |  |
| --- | --- |
|  | int \*p; |

Создали указатель типа ***int***.

|  |  |
| --- | --- |
|  | p = &A; |

Теперь переменная *p* хранит адрес переменной *A*. Используя оператор \* мы получаем доступ до содержимого переменной *A*.  
Чтобы изменить содержимое, пишем

|  |  |
| --- | --- |
|  | \*p = 200; |

После этого значение *A* также изменено, так как она указывает на ту же область памяти.  
  
**Область применения указателей:**Работа в системе косвенной адресации для экономии памяти  
  
Динамическое управление памятью. Переменные, для которых память выделена таким образом, называются динамическими.

**36. Операторы адреса и разыменования. Операции над указателями. Область применения указателей.  
  
Указатель** – это переменная, которая может содержать адрес некоторого объекта. Простейшая декларация указателя имеет формат:

***тип* \* *ID\_указателя*;**

Например: int \*a; double \*f; char \*w;

Обращение к объектам любого типа как операндам операций в языке Cи может производиться:

– по имени (идентификатору);

* по указателю (операция косвенной адресации):

***ID*\_указателя = &*ID*\_объекта;** -операция разыменования;

**\**ID*\_указателя** - операция косвенной адресации.  
  
Элементы одного массива хранятся в памяти подряд, поэтому адрес каждого последующего элемента больше адреса предыдущего на размер одного элемента, т.е на **sizeof(тип)** байт, где **тип**- тип элемента массива. Поэтому, зная адрес одного элемента, легко вычислить адрес другого.  
  
  
  
  
  
  
Можно воспользоваться этим с помощью ***арифметических операций с указателями,*** т.е. прибавлением к ним (или вычитанием из них) целой величины:

**p+i**

**p-i**

**p+=i**

**p-=i**

**p++**

**p--**

где: p - указатель, i - целочисленное выражение.

Допускается также вычитание указателей:

**p1-p2**

**Область применения указателей:**Работа в системе косвенной адресации для экономии памяти  
  
Динамическое управление памятью. Переменные, для которых память выделена таким образом, называются динамическими.   
  
**37. Указатели и const.  
  
Указатель** – это переменная, которая может содержать адрес некоторого объекта. Простейшая декларация указателя имеет формат:

***тип* \* *ID\_указателя*;**

Например: int \*a; double \*f; char \*w;  
  
На Си для хранения данных могут использоваться константы, которые предваряются ключевым словом const, и указатели также могут указывать на константы, но в этом случае перед определением указателя также ставится слово const:  
  
Пример 1:  
  
int x, – переменная типа int ;  
  
\*y; – указатель на объект типа int;  
  
y = &x; – y – адрес переменной x;  
  
\*y=1; – косвенная адресация указателем поля x, т.е. по указанному адресу записать 1: x = 1.  
  
  
**38. Ссылки.  
  
  
  
  
  
  
  
39. Применение указателей для работы с массивами. Одномерный динамический массив. Выделение/высвобождение памяти.  
  
Указатель** – это переменная, которая может содержать адрес некоторого объекта. Простейшая декларация указателя имеет формат:

***тип* \* *ID\_указателя*;**

Например: int \*a; double \*f; char \*w;  
  
Идентификатор массива указывает адрес памяти, начиная с которого он расположен, т.е. адрес его первого элемента. Работа с массивами тесно связана с применением указателей.

Пусть объявлен массив *a* из 5 целочисленных элементов:

int a[5];

***a***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a[0] | a[1] | a[2] | a[3] | a[4] |

4000 4004 4008 4012 4016

Здесь приведено символическое изображение оперативной памяти, выделенной компилятором для объявленного целочисленного массива а[5]. Адрес массива выбирается компилятором в зависимости от размера доступной памяти, наличия других переменных и массивов и др. Для конкретности, здесь положено значение адреса, равное 4000. В реальной программе вместо 4000 может быть другое значение, но относительное положение элементов массива всегда остается постоянным.  
  
В языке С ***идентифи­каторы массивов считаются константными указателями***(т.е. в данном примере ***а*** "имеет значение" 4000)***.*** Такую константу можно присвоить переменной типа указатель, но нельзя подвергать преобразованиям, например:

**int a[5], \*q;**

+**q = a;** //Правильно - присваивание константы переменной

**a = q;**// Ошибка: в левой части - указатель-константа  
  
Формат операции ***new*** для массивов:

указатель = new тип [количество] ;

Результат операции new, присваиваемый указателю – адрес начала области памяти для размещения данных указанного количества и типа. При нехватке памяти – результат NULL.

Формат операции ***delete***:

delete указатель;

либо: delete [] указатель; (эти записи равносильны)

После выделения памяти с этим массивом можно работать, аналогично работе с обычным массивом, используя вышеописанную операцию косвенной адресации

**40. Двумерный динамический массив. Выделение/высвобождение памяти.**Операция new способна выделить память лишь под одномерный массив. Тогда для двумерного массива удобный способ - это представить двумерный массив как *массив из массивов*, т.е каждую строку матрицы - как одномерный массив. При этом под каждую строку матрицы память будет выделена по отдельности, как под одномерный массив; и на нее потребуется отдельный указатель. Поскольку строк в матрице может быть много, указателей тоже будет много, и уже для их хранения потребуется вспомогательный массив из указателей. А так как количество строк в матрице переменного размера заранее неизвестно, то этот вспомогательный массив тоже будет динамическим, и сначала нужно выделить память под него. Декларировать при этом придется только указатель на вспомогательный массив; он будет, таким образом, указателем на указатель (\*\*).Пример. Создать динамическую матрицу размера n1 х n2. Присвоить каждому ее элементу значение, равное сумме номеров его строки и столбца:

...

**int \*\*m, n1, n2;**

**puts("Vvedite chislo strok i stolbtsov");**

**scanf(“%d%d”, &n1, &n2);**

**m = new int \* [n1];** // Захват памяти для массива указателей

// - на рисунке А (n1=3)

**for ( int i=0; i<n1; i++)** // Захват памяти для каждого из

**m[i] = new int [n2];** // массивов элементов (n1 таких

// массивов)- на рисунке B (n2=4)

. . .

**for ( i=0; i<n1; i++)**

**for ( j=0; j<n2; j++)**

**m[i][j] = i+j;** // \*(\*(m+i)+j) = i+j;

. . .

**for ( i=0; i<n1; i++)** // Освобождение памяти

**delete m[i];**

**delete m;**

+. . .

**41. Многомерные массивы и указатели на указатели.**Многомерный массив — это массив, элементы которого являются массивами. Например, первый элемент трехмерного массива является двумерным массивом.  
  
Пример:  
В этом примере массив с именем prop объявляется с тремя элементами, каждый из которых представляет собой массив 4x6, содержащий значения типа **int**.  
  
**int prop[3][4][6];**

## int i, \*ip, (\*ipp)[6]; Указатели на указатели

Обычный указатель типа int объявляется с использованием одной звёздочки:  
**int \*ptr; // указатель типа int, одна звёздочка**

Указатель на указатель типа int объявляется с использованием двух звёздочек:  
**int \*\*ptrptr; // указатель на указатель типа int, две звездочки**

Указатель на указатель работает подобно обычному указателю: вы можете его разыменовать для получения значения, на которое он указывает. И, поскольку этим значением является другой указатель, для получения исходного значения вам потребуется выполнить разыменование еще раз.  
  
**42. Динамическая память, функции malloc(), calloc(), realloc(), free()**Функции динамического выделения памяти находят в оперативной памяти непрерывный участок требуемой длины и возвращают начальный адрес этого участка.  
  
Функции динамического распределения памяти:

void\* malloc(РазмерМассиваВБайтах);  
void\* calloc(ЧислоЭлементов, РазмерЭлементаВБайтах);

Для использования функций динамического распределения памяти необходимо подключение библиотеки <malloc.h>:

#include <malloc.h>

Память, динамически выделенная с использованием функций calloc(), malloc(), может быть освобождена с использованием функции

free(указатель);

Функция realloc() изменяет величину выделенной памяти, на которую указывает ptr, на новую величину, задаваемую параметром newsize. Величина newsize задается в байтах и может быть больше или меньше оригинала.

**43. Строки. Объявление. Инициализация. Действия над строками и элементами строки.**

Строки в Си рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык C по своему происхождению является языком системного программирования, то**строковый тип данных в C как таковой отсутствует, а в качестве строк в Си используются обычные массивы символов**.Строки реализуются посредством массивов символов. Поэтому объявление ASCII строки имеет следующий синтаксис:

**char имя[длина];**Примеры объявления строк с **инициализацией**:

char str1[20] = "Введите значение: ", str2[20] = "";

Пример:

const char message[] = "Сообщение об ошибке!";  
  
Действия над строками:  
  
Функция вычисления длины строки:  
size\_t strlen(const char \*string);  
  
Пример:

char str[] = "1234";  
int n = strlen(str); //n == 4

Функции копирования строк:

char \* strcpy(char \* restrict dst, const char \* restrict src);  
char \* strncpy(char \* restrict dst, const char \* restrict src, size\_t num);

Функции сравнения строк:

int strcmp(const char \*string1, const char \*string2);  
int strncmp(const char \*string1, const char \*string2,size\_t num);  
  
**44. Строки. Ввод/вывод строк.**Строки в Си рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык C по своему происхождению является языком системного программирования, то**строковый тип данных в C как таковой отсутствует, а в качестве строк в Си используются обычные массивы символов**.Строки реализуются посредством массивов символов. Поэтому объявление ASCII строки имеет следующий синтаксис:

**char имя[длина];**  
**Вывод строки:**  
  
#include <stdio.h>  
  
int main(void) {  
  
char str[10];  
char str1[10] = {'Y','o','n','g','C','o','d','e','r','\0'};  
char str2[10] = "Hello!";  
char str3[] = "Hello!";  
  
for(int i = 0; i < 10; i = i + 1)  
printf("%c\t",str[i]);  
printf("\n");  
  
puts(str1);  
printf("%s\n",str2);  
puts(str3);  
  
return 0;  
}

есть несколько основных способов вывести строку на экран.

* использовать функцию **printf** со спецификатором **%s**
* использовать функцию **puts**
* использовать функцию **fputs**, указав в качестве второго параметра стандартный поток для вывода **stdout**

**Ввод строки:**  
Функция gets приостанавливает работу программы, читает строку символов, введенных с клавиатуры, и помещает в символьный массив, имя которого передаётся функции в качестве параметра.  
Завершением работы функции gets будет являться символ, соответствующий клавише ввод и записываемый в строку как нулевой символ.

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

Строки в Си рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык C по своему происхождению является языком системного программирования, то**строковый тип данных в C как таковой отсутствует, а в качестве строк в Си используются обычные массивы символов**.Строки реализуются посредством массивов символов. Поэтому объявление ASCII строки имеет следующий синтаксис:

**char имя[длина];**

**string.h - библиотека обработки строк**Большинство функций string.h не производят никакого выделения памяти и контроля границ; эта обязанность целиком возлагается на программиста.  
  
**46. Модульное программирование.**

**Мо́дульное программи́рование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.  
Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.  
  
**Мо́дуль** — функционально законченный фрагмент программы. Во многих языках (но далеко не обязательно) оформляется в виде отдельного файла с исходным кодом или поименованной непрерывной её части.  
  
**48. Функции. Объявление и вызов.**  
  
***Функция*** — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.  
Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.  
  
Общий вид вызова функции

Переменная = ИмяФункции(СписокФактическихАргументов);

***Фактический аргумент*** — это величина, которая присваивается формальному аргументу при вызове функции.

Функция должна быть ОБЪЯВЛЕНА до ее первого вызова. Определение включает в себя заголовок функции, поэтому может выполнить функции объявления и может выноситься перед первым вызовом этой функции.void f(void); /\* объявление \*/  
  
**49. Обмен информацией между функциями.**При работе программы функции должны обмениваться информацией. Это можно осуществить с помощью глобальных переменных, через параметры и через возвращаемое функцией значение.  
Использование глобальных переменных  
  
Глобальные переменные видны во всех функциях, где не описаны локальные переменные с теми же именами, поэтому использовать их для передачи данных между функциями очень легко. Тем не менее, это не рекомендуется, поскольку затрудняет отладку программы и препятствует помещению функций в библиотеки общего пользования. Нужно стремиться к тому, чтобы функции были максимально независимы, а их интерфейс полностью определялся прототипом функции.  
  
Если в теле функции имя глобальной переменной совпадает с именем локальной переменной, то все операции выполняются с локальной переменной. Локальная переменная в этом случае скрывает глобальную переменную. Для доступа к глобальной переменной необходимо применить операцию разрешения области видимости. Для этого перед переменной ставится префикс  :: . Пример:  
  
**#include <iostream.h>  
  
int Turn = 5; //объявление глобальной переменной  
  
int main ()  
  
{int Turn = 70; // объявление локальной переменной  
  
cout << Turn <<’\n’;  //вывод локального значения  
  
cout << ::Turn <<’\n’;  //вывод глобального значения  
  
return 0;  
  
}**

**50. Передача массивов в качестве параметров функции.**Когда массив используется в качестве аргумента функции, передается только адрес массива, а не копия всего массива. При вызове функции с именем массива в функцию передается указатель на первый элемент массива. (Надо помнить, что в С имена массивов без индекса - это указатели на первый элемент массива.) Параметр должен иметь тип, совместимый с указателем. Имеется три способа объявления параметра, предназначенного для получения указателя на массив. Во-первых, он может быть объявлен как массив, как показано ниже:  
  
#include <stdio.h>  
void display(int num[10]);  
int main (void) /\* вывод чисел \*/  
{  
int  t [10], i;  
for (i=0; i<10; ++i) t[i]=i;  
display(t);  
return 0;  
}  
  
void display(int num[10])  
{  
int i;  
for (i=0; i<10; i++) printf ("%d", num[i]);  
}  
  
**51. Указатель на функцию.**  
  
C функция тоже имеет адрес и может иметь указатель. Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.  
  
Самым распространенным указателем на функцию является ее имя. С помощью имени функции мы можем вызывать ее и получать результат ее работы.  
Но также указатель на функцию мы можем определять в виде отдельной переменной с помощью следующего синтаксиса:  
  
тип (\*имя\_указателя) (параметры);