1. История развития персональных ЭВМ.

1972г.-появл. 8-бит микропроц. Intel 8008. 1974г.-Intel 8880. 1979г.-Intel 8086 с 4,77; 8; 10 МГц.

1981г.- IBM MSDOS 1.0. 1981г. принцип открытой архитектуры. 1982г.-Intel 80286 (16МГц, 16МБ ОЗУ, 1Гб) 1985г.-Inyel 80386(40МГц) 1989г.-Intel 80486(66МГц) 1991г.-MSDOS 5.0. 1992г.-Intel 80486X4 (100МГц) 1992г.-Windows 3.1. 1995г.-Windows 95.

1. Теория алгоритмов.

Решение люб. задачи на люб. ЯП основывается на том или ином алгоритме. Начальным этапом разработки люб. пр-ммы явл. сост. алгоритма.

1. Понятие алгоритма и его свойства.

Алгоритм-послед. действий для получения требуем. рез-та из исх. данных за конечное время.

Св-ва: 1) дискретность-описуем. процесс долж.быть. разбит на последов. шагов образующие стр-ру алгоритма. 2) понятность- кажд предпис. должно быть понятно тому, для кого оно создано. 3) определенность-все предпис. алгор. долж.быть однозначны, и не долж.быть ситуаций, когда после вып. определ. шага неясно какой вып. следующим. 4) Массивность-использование алгоритма для групп задач. 5) Конечность-не должен быть бескон. и долж. иметь точку выхода.

1. Основные этапы алгоритмизации.

1) Разработка. Структурн. проектирование - методология разработки сложных алгоритном и их компонентов. Принципы: 1) Разделения-разбиение на модули и допускающее независим. проверку от других частей алгоритма. 2) Пошаговая детализация-разбиение на отдельные подзадачи, к кажд. из кот. составл. алгоритм.

2) Обоснование. Обоснование - док-ва недопущения ошибки и точности рез-та за кон. кол-во шагов.

3) Представление*.* Представление – запись алгоритма в виде схемы. Словесн. опис - текст в кот. идет опис. алгоритмов по пунктам.

4) Анализ и тестирование. Включают док-во правильности алгоритма и его тестирование на различных наборах данных. Ошибки: синтаксические и логические. Тестирование – процесс вып. алгоритма с целью найти ошибки путем реш. тест. задач., рез-т. кот. известен заранее.

1. Правила оформления схемы алгоритма.

ГОСТ 19 701-90. Алг.сост.из:1) Символов 2) Пояснит. текста 3) Соедин. линий. Бывают основные(неизв. тип или вид процесса) и специфические(известен точн. вид или тип процесса). Схема – граф. решение задачи, в кот. исп. фигуры для отобр. операций, данных, действий.

1. ЯП Си. История создания.

1969г.-создана первая ОС на «С»-UNIX. Можно было писать пр-ммы не завис. от архит. памяти и процессора. Изобретения строгой спецификации типов ф-ий (стандарт ANSI). 1983г.-появился 1-й транслятор С – С++.

1. Достоинства и особенности языка Си.

1)Соверменность – управл. конструкции рекомендуемые теорет и практ. программированием. 2) Эффективность – наил. обра. исп. возможности соврем. ЭВМ. Прога компактна и быстра. 3) Переносимость(мобильность)-перенос с 1 эвм на другую. 4) Мощный и гибкий 5) Удобный.

1. Алфавит языка Си.

1)Все латинские буквы и “\_” 2) Цифры 0-9. 3) Спец. символы (.,;?:!/~\+-\*|[]{}<>=&^%#”) 4) Упр. послед-ти исп. при вводе/вывод инфы. Начин. со знака \.

1. Идентификаторы.

Это имя кот. обозн. нек. объект в программе. 1) не долж. совп с ключ словами языка и именами ф-й из библиотек. 2) не рекоменд. начин. ид-р со знака «\_». 3) Правельн. выбор ид-ров облегчает понимание пр-мы.

1. Ключевые(зарезервированные) слова.

Это имена исп. в С с заранее определ. смыслом. Их нельзя примен. в кач-ве ид-ров. Они сообщ компилятору о типе данных, способе организации и послед-ти вып. операторов. (auto, do, sizeoff, break, double, int, switch, chart, count, extern, return, union, short, for, signed, void и т.д.)

1. Типы данных языка Си.

char-символьный; int-целый; float-вещественный; double-двойной в-ный; void-пустой. Модификаторы: signed-знаковый; unsigned-беззнаковый; short-короткий; long-длинный. Тип и мод-р определяют: 1) формат хранения данных в ОЗУ 2) диапозон зн., в пределах кот. может измен. переменная. 3) операции кот. могут вып. над данными соотв. типа. Типы: 1) Скалярные (символы,арифметические(цел, вещ.), указатели, перечисления) 2) Составные(массив, поля битов, стр-ра, объединения).

1. Константы и переменные.

Переменные - это эл-ты данных, зн-я кот. во время вып. пр-ммы можно изменять. Константы – не изменяемые данные.

1. Данные целого типа: chart(1байт, 0..255) int(2байта, 0..65536) long(4байта,0..4,294,967,295)
2. Вещественные типы данных.

Переменные с плав. точкой Float(7 знаков,4байта,1,17E-38..3.37E+38), double(15знаков,8байт,2.23E-308..1.67E+308), long double(19 знаков,10байт,3.37E-4932..1.2E+4932). Сост. из мантиссы m и порядка p. С=m\*2p.

1. Символьный тип данных.

char (1 байт, 0..255), целый, как со знаком так и без. Из ASCII по заданному коду выбир. изображ. символа.

1. «Пустой» тип void.

Отсутствие описания типа. Описание указателя общего типа, когда заранее извест. тип объекта на кот. он будет ссылаться.

1. Совместимость типов.

Типы char и int могут свободно смешиваться в арифм. выр-ях: char преобраз. в int. Преобразования: 1) Если +,\* то низший тип преобрю. в высший (сhar→int; unsigned long→float). 2) Если 1 из операндов unsigned, то др. операнд и рез-т также преобраз. в этот тип. 3) При сужающихся преобраз. в целых типах лишние биты более старш. порядков отбрасыв. а дробные типы округл.

1. Структура простой пр-ммы на ЯП С.

Исходники быв. 2 типов: 1) Заголовочный – h-файлы 2) файлы реализации с, сpp, cc, c++. Заголовчные ф-лы сод-т только описание(прототипы ф-ий, число и типы аргументов ф-ий, константы, стр-ры, служат для передачи инф-ии между модулями).Ф-лы реализ. сод-т тексты ф-ий и определения глоб. переменных, и предст. соб. отдельные модули. Вып. люб. пр-ммы нач с ф-ии main. Исходная пр-мма сост. из: директив, указание компилятору, объявлений и опредлений.

Препроцессор – пр-мма предварит. обраб. текста перед трансляцией, используется для подключения заголовочных файлов. Директивы – команды препроцессора, сод-т символ # в начале строки. Указание компилятору – команды, вып. комплилятором во время процесса компиляции. Объявления – задают имена и атрибуты переменных ф-й и типов, исп. в пр-мме. Определения – объявления определяющие переменные(в дополн. к имени мож. задавать нач. значение) и ф-ии.

1. Директивы препроцессора Си.

Предст. соб. инструкции в виде текст. пр-ммы, позвол. измен. текст пр-ммы, вставить текст из др. файла и т.д.

1. Директива #include - включ. в пр-мму содержимое указанного файла.
2. Директива #define.

Исп. для замены часто использующихся констант, операторов, ключ. слов или выр. некот. идентификаторами. Ид-ры заменяющие текст или числ. конст. наз-ся именовынными константами. Ид-ры замен. фрагмент пр-ммы наз. макроопределения. 1) #define ид-р текст 2) #define ид-р (список параметров) текст. Эта директива замен. все вхождения ид-ра на текст, такой процесс наз. макроподстановкой. #define width 80; #define length\_(width+10).

1. Директива #undef – исп. для отмены действия #define. #undef ид-р; #undef width.
2. Коментарии

Текст. части для аннотирования пр-ммы. Способы задания: /\*..\*/ -коменты допуск. везде, где разрешенны пробелы. // -в 1-й строке. Компилятор игнорирует текст комментария.

1. Стандартная математическая библиотека функций.

<math.h> преобладает тип double, т.е. корректно раб. с типом float, longdouble-приводит к ошибкам. Примеры: abs(intx), fabs(double x), log(double x), log10(double x), exp(double x), sin(double x), floor(double x), hypot(double x, double y), pow(double x, double y).

1. Операции в языке Си.

Это спец. комбинации символов выполняющие действия по преобраз. различных типов величин. Компилятор интерпретирует кажд. из этих комбин. как самост. ед-цу наз. лексемой. Все операции делятся на унарные(действие над 1 операндом) и бинарные(над 2-мя).

1. Арифметические операции. +, -, -(отрицание), \*, /, %(остаток от деления).
2. Операции поразрядной арифметики. &(и), |(или), ^(исключающее или), ~(инверсия), <<(сдвиг влево), >>(сдвиг вправо).
3. Логические операции. &&(и) ||(или) !(отрицание).
4. Операции отношения. ==(сравнение на равенство), >, <, >=, <=, !=.
5. Операции присваивания. =, ++(унарный инкримент), --(унарный декремент).
6. Условная (тернарная) операция. &(адресация) \*(разадресация), «,»(послед. вычисление),

? (условное выражение, усл. тернарн. операция).

1. Операция sizeof.

Определение размера в байтах. a>sizeoff(int) - определяет размер памяти кот-му соотв. ид-р в памяти или тип a присваивает размер типа int.

1. Операции «увеличить на», «домножить на» и т.п.

+= (x+y→x+=y), - = (x=x-y→x-=y), \*=, /=, %=, &&=, ||=.

1. Операция приведения типа.

type cast – исп. когда значение 1 типа преобр. к другому в случае, есле сущ. разумный способ такого преобразования. Пример 1. int→float: double x; int n; x=(double)n; Пример 2. float→int: double x,y; int n,k; x=3,7; y=(-1,5); n=(int)x; //n=3 k=(int)y //k=-1.

1. Операции «&» и «\*».

& - операция взятия адреса явл. унарной, т.е. переменной y присв. адрес переменной x. y=&x.

\* - воспринимает операнд как адрес нек. объекта и исп. его для выборки содержимого. z=\*y.

int \*L, float \*p – L,p – указатели.

1. Выражения. Приоритет операций.

Операнд-константа либо переменная. ()[] . \* - > (выражение, л→п); - ~ ' \* & ++ -- sizeoff (унарный, п→л); \* / % (мультипликативный, л→п); + - (аддитивный, л→п); << >>(сдвиг, л→п); < > >= <= (л→п); == , ! = (отношение, л→п); & ^ (л→п); ^( л→п); | (побит. вкл. или, л→п); && || (л→п); ?(условная, п→л); = \*= /= %=(прост. и сост., п→л); += -= <<= >>= &= != ^= (присваивание) , (послед. преобраз, л→п).

1. Функции ввода-вывода в языке Си.

<stdio.h> printf(“упр. строка”,аргументы1,аргументы2); Упр. строка может содерж. как обычн. символы так и спецификаторы преобразования, управляющие символьные константы. Между спец-ром и символом встречаются: «-» указывает что преобразованный параметр должен быть выравнен влево в своем поле; «.»-отделяет размер поля от след. строки; строка цифр - задает мах размер поля; символ длины цифр-указывает что соотв. аргумент имеет тип long. Символы: d-десят. число, o-восьмеричное, х-шестнадцатеричное, с-символ, s-строка, е-вещ.десят.число, f-вещ.дес.число с плав точкой, g-искл. вывод незначащих нулей, p-знач. аргумента явл-ся указателем, u-беззнаковое число(unsigned).

1. Функция форматного вывода printf().

printf исп. упр. строку чтобы определить ск-ко всего аргументов и каковы их типы. \n-переход на новую строку, \t-горизонтальная табуляция.

1. Функция puts()

Записывает симв. строку в стандартный поток данных(выводит на экран). int puts (const chat \*string). После вып. этой ф-ии курсор переход. в новую строку и ф-ия возвр. код символа.

1. Функция putchar().

int putchar (int ch); записывает символ в стандартный поток данных, возвр. выведенный на экран символ.

1. Функция форматного ввода scanf().

scanf(“упр.строка”,&aрг1,арг2,...); Аргументы–указатели на соотв. значения. Символы: d-десят. число, o-восьмеричное, х-шестнадцатеричное, с-одиночный символ, s-строка, е-вещ.десят.число, f-вещ.дес.число, p-указатель в виде 16-го числа.

1. Функция gets().

char \*gets(char \*buffer); Счит. симв. строку и помещ. ее по адрусу заданному указателем в буфер. Прием строки заканч. на символе \n, он удаляется и заменяется нуль-терминатором \o.

1. Функция getchar().

int getchar(void); Считывает символ из стандартного входного потока и возвр. его.

1. Операторы языка си.

Условные(if-разветвления switch-выбора), организации циклов(while, do..while, for), перехода(break, continue, return, goto).

1. Условные операторы.
2. Оператор if.

3 способа задания: 1) if (усл.) оператор; 2) if (усл.) { оператор }else оператор }3) if м.б. вложенным. Если рез-т вып. усл. истина то вып. оператор 1, если нет то 2.

1. Оператор выбора switch.

Выбор 1 варианта из многих. switch (выражение) {case const1: оператор1, break) case const2: оператор2, break)..defolt (оператор N, break); } Св-ва switch: 1) выр-е должно иметь целочисленный тип 2) defolt м.б. записан в люб. месте. 3) break передает упр. за пределы оператора switch 4) switch отлич. от if тем что может вып. только операции строгого равенства.

1. Операторы цикла.

Цикл - многократно повт. участок выполняемого процесса. Каждый переход по телу цикла наз. операцией. Цикл не сод. внутри себя др. циклов наз. простым, содержащий – сложным. Анализируемая переменная наз. параметром цикла. Цикл. процесс- разветвл. пр-сс с 2-мя ветвями из кот. 1 возвр. на предыд. блоки.Кол-во повт. цикла может указываться а м.б. вычислено-это циклы со счетчиком.

1. Оператор while.

while (усл.) оператор { оператор1…оператор N } Повторн. вып. 1 или неск. операторов в {}, до тех пор пока логич. усл. не примет зн. ложь. While-цикл с предусл., т.к. истинность улс. пров. перед вхождением в цикл, зн. возможна ситуация, когда цикл не вып. ниразу. В if услю пров. 1 раз, в while-много раз. В while надо вкл. конструкции изменяющие величину проверяемого выр-я, чтобы в итоге оно стало ложным, иначе цикл будет бесконечным.

1. Оператор do…while.

do {<оператор>} while (<условие>); Вып. цикла 1 раз, do…while – цикл с постусловием.

1. Оператор for.

for (инициализация; условие; изменение параметров) – цикл с предусловием, может не вып. ниразу. 1) вып.инициализация 2) проверка условия 3) вып. тело цикла 4) вып. изм. парам-в 5) возвр. к пункту 2.

1. Выбор оператора цикла.
2. Операторы переходов break, continue, return, goto.

break-прекращ. вып. цикла и передает выполнение оператору следующему после данного цикла. Continue-пропускает часть цикла стоящую после его записи. Return – возвращ. знач. нек. выр-я, кот. становится значением ф-ии. Goto указывает что вып. пр-ммы необх. продолжить с инструкцией, перед кот записана метка. goto <метка>. В пр-мме должна быть указана строка с меткой, поставлено «:» и записана инструкция к кот. осущ. переход. метка: <идентификатор>.

1. Пустой оператор.

« ; » при вып. ничего не происходит. Исп. в операторах do, while, for, if когда место оператора не требуется, но по синтаксису требуется.

1. Оператор выражение.

Люб. выр. кот. заканч. на « ; » явл. оператором. Вызвать ф-ю не возвращ. значения можно только при помощи оператора выражения. a(x,y);

1. Массивы и указатели в СИ.

Массив – группа эл-в одного типа располож. друг за другом в памяти и имеющих 1 общее имя. Индекс – порядковый номер эл-та в массиве. float array [12]; int buffer [5\*10]; char arr[3]={“A”,”B”,”C”); int arr []={34,19,5,17}; arr[arr[2]]=19;

1. Одномерные массивы.

А че писать то, тут и так все понятно ^-^

1. Многомерные массивы

Это массив, эл-ми кот. явл. другие массивы. Размерность массива-кол-во индексов, исп. для ссылки на конкретн. эл-т. int array [10][30]; int array [2][2][2]={25,54,16,43..}

1. Указатели.

Указатель – вид переменной, кот. хранит адрес эл-в в памяти, где м.б. записано зн. др. переменной. тип \*имя\_указателя; int var, \*point. Операции с указателями: 1) доступа: \*E 2) присваивание: аналогично как и для типов данных. 3) увеличения либо уменьшения указателя: int i, \*p; p+i //указ. на адрес i-го эл-та после данного p-i //перед данным. 4)сложного присваивания: p+=i; p-=i; 5) инкремента и декремента: p++; ++p; p--; --p; 6) индексирования p[i] аналогична \*(E+i), т.е. из памяти извл. и исп. зн. i-го эл-та, адрес кот. присвоен указателю E. 7) вычитание указателей: E1-E2, где E1, E2 – переменные типа указатель на 1 и тот же тип данных, рез-т имеет тип int. 8) отношений: E1==E2; E1>=E2; E1!=E2 и т.д. рез-т имеет тип int.

1. Строки.

Строка – последоват. люб. символов заключенных в “ ”. Строка предст. со. массив с эл-ми типа char в конце кот. помещен «\о». такой массив наз. строкой формата ASCIIZ. char str[3]={“H”,”I”,\o}; char str[3]=”hi”; char str []=”hi”; scanf(“%s”, str);

1. Стандартная библиотека работы со строками.

#include <string.h>; strcopy(char\*dst, char\*src) //копирование строки адресованной src в область памяти на кот. указано dst. size\_t – целочислен. тип исп. для хранен. индексов массивов. strncpy (char\*dst, size\_t n); //копир. только 1-е n-символов в строке src. strdup (char\*s); //выдел. память и копир. в нее содержим. строки s. strlwr(char\*s); //преобраз. все символы строки к нижнему или верхнему регистру. strref(char\*s); //меняет порядок следования символов в строке на противоположный. strcut(char\*s); //присоедин. строку src в dst. strcmp(const char\*s1, const char\*s2); //сравнивает строки заданные указателями s1,s2 в лексикограф. порядке, возвр 0 если обе идентичны. strleng (char\*s); //возвр. длину s в байтах. strstr (const char \*s1, const char\*s2); //находит место 11-го вхождения s2 в s1 и возвр. указатель на соотв. позицию.

1. Поиск и сортировка.

Поиск в массиве из n-эл-в для нах. зн. соответствия с ключом key, возвр. индекс соотв. эл-та: int seqsearch (int list[], int key, int n); { int i; for (i=0, i<n i++) in (list[i]==key); return i; //возвр. индекс найден.эл-та. return -1 //если поиск неудачн. возвр. «-1». Cортировка целочисленного массива: #include <stdio.h> | #include <stdlib.h> | #define N1000 | int cmp(const void\*a, const+void\*b) | { return (\*(int\*)a \*(int\*)b); | } main() | { int n,i,a [N]; scanf («%d; &n); | for (i=0;, i<n; i++) | scanf (“%d,i) | qsort (a,n,sizeoff(in) cmp) | for (i=0; i<n; i++) | printf(“%d”, a[i]) |.

1. Организация последовательного поиска.

При послед. поиске различ. наилучший и наихудший случаи. Наил.-когда ключ нах-ся в 1 эл-те, наих.- когда нах-ся в конце списка или не нах-ся вообще.

1. Организация бинарного поиска.

Наилучший метод поиска если список упорядочен. Например поиск номера в тел. книге. Алгоритм: 1) находим середину списка: mid-(low+high)/2; 2) сравниваем зн. в среднем эл-те с ключом key, если совпад. найдено, то возвр. индекс mid для нах-я ключа. if (A[mid]==key) return(mid); 3) Определение новых границ: if (A[mid]>key) high=mid-1; if (A[mid]<key) low=mid+1; 4) переход к пункту 1, пока не будет найден ключ или пройден весь путь. Наилучший случай – когда совпад. с ключом эл-т нах-ся в середине списка.

1. Основные алгоритмы сортировки массивов.

Метод пузырька, выбора, вставками, быстрая сортировка.

1. Сортировка методом пузырька.

Для сорт. массива А из n-эл-в требуется n-1 проходов. Проход – прохожд. всего амссива 1 раз от 1 до последн. эл-та. В кажд. проходе сравн. соседние эл-ты, и если 1-й из них > второго, то они меняются местами. По окончании кажд. прохода наим. эл-т подним. к вершине текущего подсписка. #include <stdio.h> | void main () | { const int n=5; | int A[n]={50,20,40,75,35}; | int i,j, aux; | for (i=0; i<(n-1); i++) | for (j=0; j<(n-1); j++) | { if (A[j]>A[j+1]) | { aux=A[j]; | A[j]=A[j+1]; | A[j+1]=aux; | } for (i=0; i<n; i++) | printf (“%5d”, A[i]); |.

1. Сортировка методом выбора.

В нулевом проходе выбир. наим. эл-т, кот. затем меняется местами с A0. После этоо неупоряд. эл-ми становятся A1 по An-1. В след. проходе рассм. неупоряд. хвостовая часть списка. оттуда выбир. наим. эл-т и запоминается в А1. В след. проходе происх. поиск наим. эл-та в подсписке А2..Аn-1. Найденное зн-е запомин. в А2. Таким обр. вып-ся n-1-проход. После чего неупоряд. хвост сокращ. до 1 эл-та, кот. и явл. наибольшим. #include <stdio.h> | void main () | { const int n=5; | int A[i]={50,20,40,75,35}; | int i,j,k,m,n; | for (i=0; i<(n-1); i++) | { min=A[i]; | k=i; | for (j=i+1; j<n; j++) | { if (A[j]<min | k=j; | min=A[j]; }} | A[k]=A[i] | A[i]=min | { for (i=0; i<n; i++) | printf (“%d, A[i]; }.

1. Сортировка вставками.

Для сортироки A из n-эл-в потребуется n-1 переход. #include <stdio.h> | main() | { const int=5; | int A[n]={50,20,40,75,35}; | for (i=0; i<(n>1); i++) | { for (j=i+1; j<n; j++) | {for (k=0; k<(); k++) | { if (A[j]<A[] | { Aj=A[j]; | for (m=j; m<k; m--) | A[m]=A[m-1]; | A[k]=Aj; | }}}} for (i=0; i<n; i++) | printf (“%5d” A[i]) }.

1. Быстрая сортировка.

Общая схема: 1) Есть нек. массив A из n эл-в. Из него выбир. нек. опорный эл-т – p. 2) Запускается процедура разделения массива, кот. перемещ. все эл-ты, кот. ≤p влево от него, а > вправо. 3) Так. обр. мы получили новый амссив из 2-х подмассивов, причем левый < правого. 4) Для обоих подмассивов, если в подмассиве > 2х эл-в, рекурсивно запускаем ту же процедуру. 5) Получаем полностью отсортир. посл. Достоинства: 1) Самый быстродействующий 2) Простая реализация 3) Хорошо рботает на почти отсортированных данных.

1. Функции пользователя в Си.

Вып. пр-мы нач. с ф-и main(). В Си запрещено объявлять 1 ф-ю внутри другой. Причины исп. ф-й: 1) Уменьш. дублирования кода 2) Сделать пр-му более структурированной и понятной. Ф-и необх. объявл. в начале пр-мы, такое объявл. наз прототипом ф-ии. Прототип должен предшествовать исполнению ф-ии а полное ее описание м.б. помещено как после так и до тела пр-мы. тип имя ф-ии (тип имя арг.1, арг.2..) Тип перед именем ф-и опр. тип значения, кот. возвр. ф-я. Если тип не указан, то возвр. тип int. Полное опис. ф-й и параметров надо для того чтобы ф-я могла к ним обращаться. int function (int a, int b) //прототип; int function (int a,b,c); Передача знач. из вызванной ф-и в вызвавшею происх. с помощью оператора возврата return\_значение. Вызваная ф-я не мож. менять зн. аргумента вызвавшей ее пр-мы. Но это можно сделать если передавать ф-ям не переменные а их адреса. int swap (int \*a, int \*b); | { int \*tmp; | \*tmp=\*a; | \*a=\*b; | } swap (&b, &c) //передаются адреса b и c, выполнение ф-и приведет к тому, что b и с поменяются местами.

1. Структуры в Си.

Структура – конструкция, кот. позвол. объедин. несколько эл-в с разными типами и именами в 1 составной объект. Отличия от массива: 1) Могут содерж. данные различн. типов 2) В массиве обращ. к данным идет по номеру, в стр-ре по имени.

1. Объявление шаблонов структур.

Общий синтаксис объявления шаблонов стр-р: struct имя шаблона { | тип эл-та: имя 1 | тип эл-та: имя 2 } |. Struct Database { char FAM[20]; | long Tel; | double w; }|. Задание шаблона не влечет резервирование в памяти. struct date { | int day, month, year } | struct FAM[30], inia[15], otch[20]; | struct date Birthday; }.

1. Объявление структур-переменных.

Ничем не отл. от обычной переменной с предопределенным типом. Синтаксис: struct имя\_шаблона | имя\_переменной. struct date date1[15] //объявл. массив из 15 стр-р | struct person bstu [20]; | \*per\_ptr=&bstu[0]; //объявл. массив из 20 стр-р типа person и задет указатель на данный тип. разрешается совмещать описание шаблона и определение стр-го эл-та. struct date { | int date, month, year } | a,b,date[15], \*ptr\_a=&a;

1. Доступ к компонентам структуры.

Осуществл. с пом. оператора «**.**» при работе со структ. или «->» при исп. указателя на стр-ру. Синтаксис: имя перемен. стр-ры**.**имя поля; имя указателя->имя поля. Доступ к эл-ту: date1[15]**.**day=10 | struct (bstu[10].FAM, “Иванов”); Доступ по указателю: strcpy((per\_ptr+10)->FAM, “Иванов”); | (per\_ptr+10)->heigh=180;

1. Анонимное определение структуры и оператор определения типа typedef.

Исп. когда имя стр-ры после ключевого слова struct опускается. В этом случае список опис. перемен. должен быть не пустым. struct { double x; | double y; } | R2point, \*R2point ptr; //опрделены 2 пер.-R2point, R2point ptr. 1-я имеет структ. тип с полями х и у типа double, 2-я указатель на данный структ. тип. Чаще исп. оператор определения типа typedef. typedef struct { | double x; double y; }R2point, \*R2point ptr;