**1.Массивы и указатели**

Массив – совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива осущ с помощью индекса. В си массивы сост из соприкасающихся участков памяти. Наименьший адрес соотв первому элементу, наибольший – последнему. Массивы могут иметь одну или несколько размерностей. Напр, объявл int digits[10]={0,1,2,3,4,5,6,7,8,9}; создаёт 10-элементый массив целых чисел с именем digits и последовательно присваивает значения от 0 до 9каждому элементу массива. Результат действия этого объявления аналогичен результату работы след операторов: int i,digits[10];for(i=0;i<10;i++) digits[i]=I; Массивы тесно связаны с указателями.

Про указатели в 4.

**2.Одномерные массивы**

Массив – совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива осущ с помощью индекса. В си массивы сост из соприкасающихся участков памяти. Наименьший адрес соотв первому элементу, наибольший – последнему. Массивы могут иметь одну или несколько размерностей. Напр, объявл int digits[10]={0,1,2,3,4,5,6,7,8,9}; создаёт 10-элементый массив целых чисел с именем digits и последовательно присваивает значения от 0 до 9каждому элементу массива. Результат действия этого объявления аналогичен результату работы след операторов: int i,digits[10];for(i=0;i<10;i++) digits[i]=I;

**#include<stdio.h> #include <conio.h>**

**void main( )**

**{ int m[10],i,s=0; for(i=0; i<10; i++)**

**{ scanf("%d",&m[i]); s += m[i]; }**

**printf("\nсумма = %d", s); }**

**3.Многомерные массивы**

– это массив элементами которого являются массивы. Размерность массива – это количество индексов, исп для доступа к конкретному элементу массива. Пример объявл двухмерного массива: int mas[5][15]; можно инициализировать и многомерные массивы. Инициал выполн построчно, то есть в порядке возрастания самого правого индекса. Именно в таком порядке элементы многомерных массивов распола в памяти комп, напр: int mas[2][3]={2, 14, 36, 23, 1, 9}; Проинициал массив будет выглядеть так: 2 14 36 / 23 19 Отличие в работе с многомер массивами от работы с одномер, состоит в том, что для доступа к элементу многомер массива необход указать все его индексы:

n= mas[i][j]; // работа с элементом располож на i-ой

mas[i][j]=n; // строке и в j-ом столбце матрицы mas

Пример:Разработать программу умножения матриц чисел.

#include<stdio.h> void main() { int i,j,k,n,t; int a[3][4],b[4][2],c[3][2]; printf("\nВведите матрицу a");

for(i=0;i<3;i++) for(j=0;j<4;j++) { printf("\na[%d][%d]=",i,j);

scanf("%3d",&a[i][j]); } printf("\n Введите матрицу b");

for(i=0;i<4;i++) for(j=0;j<2;j++) { printf("\nb[%d][%d]=",i,j);

scanf("%3d",&b[i][j]); } i=0; while(i<3) { j=0; while(j<2)

{ c[i][j]=0; k=0; while(k<4) { c[i][j]+=a[i][k]\*b[k][j];

k++; } j++; } i++; } puts("\nРезультирующая матрица c");

for(i=0;i<3;i++) { printf("\n"); for(j=0;j<2;j++)

printf("%3d",c[i][j]); } }

**4.Указатели**

Указатель - это переменная (группа ячеек), содерж адрес некот

перем. Если перем объявл как указатель, то она содержит адрес памяти, по которому может находится скалярная величина любого типа. При

объявл переменной типа указатель, необходимо опред тип объекта

данных, адрес которых будет содер переменная, и имя указателя с

предшествующей звездочкой (или группой звездочек). Формат объявления указателя: [ модификатор ] спецификатор-типа \* идентификатор.

примеры описания указателей:

int \*p; p - указатель на элементы типа int

float \*k,\*k1; k и k1 – два указателя на элементы типа float

char \*s; // s – указатель на элементы символьного типа

Адрес - это целое число без знака, поэтому при выводе на печать значения указателя (p,k,k1,s) будем использовать формат %u. Исключением является безтиповый указатель: void \*pp;

Пример: перезапись значения переменной a в переменную b.

#include <stdio.h>

void main(void)

{ int \*p;

float a=10.25,b;

p= &a; b= \*p;

printf(“a= %f b= %f”,a,b); }

**5.Строки**

-массив значений типа char,завершающийся нулевым байтом. Нулевой символ следует сразу за последним значащим символом.

Строки обычно запоминаются одним из 3 способов: -как литеральные строки, введённые в текс программы;

-как переменные, имеющие фиксированный размер в памяти;

-как указатели, которые адресуют массивы символов, располагающиеся в динамической памяти.

Литеральные строки вводятся непосредственно в текс программы. Они должны быть заключены в кавычки. Например: #define Title “My programm”

Строковая переменная имеет фиксированный объём памяти. Поскольку строки явл массивами, при их объявлении исп квадратные скобки, внутри которых находится целое значение, опред длину строки: char srtoka[128]; Объявл таким образом переменная может хранить от 0 до 127 символов + завершающий нуль.

Строковые указатели явл не строками, а указателями, которые опред местонахожд в памяти первого символа строки. Объявляются как char\* или, чтобы операторы не могли изменить адресуемые ими данные, как const char\*. В строковых указателях нет ничего особенного – они просто указывают на массив значений типа char и ведут себя аналогично другим указателям.

Пример: #include <stdio.h> #include <stdlib.h> main() { int kk,k,n;

char \*s; puts("введите длину строки ");

scanf("%d",&n); s=(char\*)malloc(n);

if(!s) return 0; // ошибка выделения памяти fflush(stdin); // чистка входного потока puts("введите строку: ");

gets(s); puts("исходная строка: ");

puts(s); k=0; while(\*(s+k)) k++; // переход в конец строки. k – позиция ‘\0’

while(\*(s+k)!=' ') k--; // позиция пробела перед последним словом kk=k--; // запоминаем в kk позицию пробела и сдвиг на // последний символ предпоследнего слова в строке

while(\*(s+k)!=' ') k--; // позиция пробела перед предпоследним словом

while(\*(s+ ++k)=\*(s+ ++kk)); // удаление предпоследнего слова (записываем на его место последнее слово)

puts("преобразованная строка: ");

puts(s); }

**6.Стандартная библиотека работы со строками#include <string.h>**

strcpy() – копирование строки.Функция возвращает указатель на начало скопированной строки.

Stpcpy-то же, что и предыдущая функция, но возвращает указатель на конец результирующей строки.

strncpy() – копируются только первые n символов строки. strdup() – выделяет память и копирует в нее содержимое строки s. strlwr(), strupr() – преобразует все латинские символы строки s соответственно к нижнему и к верхнему регистру. strrev() – меняет порядок следования символов строки на противоположный.strcat() – присоед строку 1 в конец 2 строки. strset() – заполняет все позиции строки s символом ch.strnset() – то же для первых n символов строки s. strcmp() – сравнивает строки, заданные константными указателями s1 и s2, в лексикографическом порядке с учетом различия прописных и строчных букв. stricmp() – то же, но без учета разницы между прописными и строчными латинскими буквами.strchr() – возвращ указатель на первое вхождение символа в строку . strrchr() – то же, но возвращ указатель на последний совпавший символ в строке.strlen() – возвращ длину строки в байтах без учета нуль-терм.

strstr() – находит место первого вхождения строки в строку strspn() – возвращ длину сегмента строки, состоящего только из символов, входящих в строку .strtok() – выделяет лексему в строке s1. Лексема-фрагмент строки s1, ограниченный любыми из символов, встречающихся в строке s2. #include <stdio.h> #include <String.h>

void main(void) { char \*String1 = "Stroka";

printf("String1 = %s\n", String1);int dlin= strlen(String1); printf(" Длина= %d в байтах\n\n", dlin); }

**7.Поиск и сортировка**

Сортировка применяется во многих программах оперирующих инф. Цель сортировки – упорядочить инфу и облегчить поиск требуемых данных. Существует много методов сортировки данных. Причём для разных типов данных иногда целесообразно применять разные методы

сортировки. Практически каждый алгоритм сортировки можно разбить на три части: - сравнение двух элементов, определяющее упорядоченность этой пары; - перестановку, меняющую местами неупорядоченную пару элементов; - сортирующий алгоритм, определяющий выбор элементов для сравнения и отслеживающий общую упорядоченность массива данных.

Сортировка данных в оперативной памяти называется внутренней, а сортировка данных в файлах называется внешней .**Поиск**-процесс, результатом которого явл указание точного места расположения искомого элемента.

void srt2(int \*ms, int k) { register int i,j,n; for(i=1;i<k;++i) // индекс элемента для упорядочивания { j=i-1; // индекс предыдущего элемента n=ms[i]; // значение предыдущего элемента while (j>=0 && n<ms[j]) ms[j--+1]=ms[j]; // сдвиг всех элементов направо ms[j+1]=n; // запись в освободившийся или в тот же элемент } }

**8.Организация последовательного поиска**

Последовательный поиск предусматривает последовательный просмотр всех элементов списка В в порядке их расположения, пока не найдется элемент равный V. Если достоверно неизвестно, что такой элемент имеется в списке, то необходимо следить за тем, чтобы поиск не вышел за пределы списка, что достигается использованием стоппера.

main()

{int k[101],v,i;

for (i=0;i<100;i++)

scanf("%d",&k[i]);

scanf("%d",&v);

k[100]=v;

i=0;while(k[i]!=v) i++;

if (i<100) printf ("%d %d",v,i);else printf ("%d не найден",v);}

**9.Организация бинарного поиска**

Бинарный поиск состоит в том, что ключ V сравнивается со средним элементом списка. Если эти значения окажутся равными, то искомый элемент найден, в противном случае поиск продолжается в одной из половин списка.

main() { int k[101],v,I,j,m;

for (i=0;i<100;i++) scanf("%d",&k[i]);

scanf("%d",&v); i=0; j=100; m=50;

while (k[m]!=v) { if (k[m] < v) i+=m;

else j=m-i;m=(i+j)/2; }

printf("%d %d",v,m);}

**10.Основные алгоритмы сортировки массива**

Сортировка применяется во многих программах оперирующих

информацией. Цель сортировки – упорядочить информацию и облегчить поиск требуемых данных. Существует много методов сортировки данных. Причём для разных типов данных иногда целесообразно применять разные методы сортировки. Практически каждый алгоритм сортировки можно разбить на три части: - сравнение двух элементов, определяющее упорядоченность этой пары;

- перестановку, меняющую местами неупорядоченную пару элементов;

- сортирующий алгоритм, определяющий выбор элементов для сравнения и отслеживающий общую упорядоченность массива данных.

void sort(int \*ms, int k)

{ int i,j,m; for(i=0;i<k-1;++i) // выбор верхней границы массива

for(j=k-1;j>i;--j) // просмотр массива ”снизу” ”вверх” {if(ms[j-1]>ms[j]) // условие замены выполнено

{ m=ms[j-1]; // замена j-1 и j элементов

ms[j-1]=ms[j]; ms[j]=m; } } }

**11. Пузырьковая сортировка.**

Название метода отражает его суть. ”Легкие”элементы массива

”всплывают” вверх(в начало), а ”тяжелые” опускаются вниз(в конец). Пузыр сортир проходит по массиву снизу вверх. Каждый элемент массива сравнивается с элементом, который находится непосредственно над ним. И если при их сравнении оказывается, что они удовлетвор условию их перестановки,то она выполняется. Процесс сравнений и перестановок продолжается до тех пор, пока ”легкий” элемент не “всплывет” вверх. В конце каждого прохода по массиву, верхняя граница сдвигается на один элемент вниз (вправо). Сортировка продолжается анализируя все меньшие массивы void sort(int \*ms, int k)

{ int i,j,m; for(i=0;i<k-1;++i) // выбор верхней границы массива

for(j=k-1;j>i;--j) // просмотр массива ”снизу” ”вверх” {if (ms[j-1]>ms[j]) // условие замены выполнено

{ m=ms[j-1]; // замена j-1 и j элементов

ms[j-1]=ms[j]; ms[j]=m; } } }

**12. Сортировка выбором**

Выбирается очередной исходный элемент массива. Последовательно проходя весь массив, сравниваем этот элемент со всеми, находящимися после него, и, при нахождении элемента удовлетворяющего условию замены запоминаем его индекс.После просмотра всего массива, переставляем выбранный элемент с исходным. После перебора всех элементов,получим отсортированный массив.

void sort(int \*ms, int k)

{ int i,i1,j,m; for(i=0; i<k-1; i++) // выбор исходного элемента к сравнению

{ i1=i; // for(j=i+1; j>i; j--) // просмотр массива ”снизу” ”вверх”

if(ms[k]>ms[j]) i1=j; // фиксируем координату элемента в массиве

m=ms[i]; // замена i1 и i элементов

ms[i]=ms[i1]; ms[i1]=m; } }

**13. Сортировка вставкой**

В методе вставки на первом шаге выполняется сортировка первых двух элементов массива. Далее алгоритм ставит третий элемент в порядковую позицию, соответствующую его положению относительно первых двух элементов. Затем в этот список вставляется четвертый элемент и т.д. Процесс продолжается до тех пор, пока все элементы не будут отсортированы.

void srt2(int \*ms, int k)

{ register int i,j,n; for(i=1;i<k;++i) // индекс элемента для упорядочивания

{ j=i-1; // индекс предыдущего элемента

n=ms[i]; // значение предыдущего элемента while (j>=0 && n<ms[j])

ms[j--+1]=ms[j]; // сдвиг всех элементов направо ms[j+1]=n; // запись в освобод или в тот же элемент } }

**14. Быстрая сортировка**

Быстрая сортировка использует стратегию «[разделяй и властвуй](http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D1%8F%D0%B9_%D0%B8_%D0%B2%D0%BB%D0%B0%D1%81%D1%82%D0%B2%D1%83%D0%B9_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))». Шаги алгоритма таковы: Выбираем в массиве некоторый элемент, который будем называть *опорным элементом*. Операция *разделения* массива: реорганизуем массив таким образом, чтобы все элементы со значением меньшим или равным опорному элементу, оказались слева от него, а все элементы, превышающие по значению опорный — справа от него. [Рекурсивно](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) упорядочиваем подмассивы, лежащие слева и справа от опорного элемента. Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.int n, a[n]; //n - количество элементовvoid qs(int\* s\_arr, int first, int last) { int i = first, j = last, x = s\_arr[(first + last) / 2]; do { while (s\_arr[i] < x) i++;while (s\_arr[j] > x) j--; if(i <= j) {if (i < j) swap(s\_arr[i], s\_arr[j]); i++;

j--; } } while (i <= j); if (i < last)

qs(s\_arr, i, last); if (first < j)

qs(s\_arr, first,j);}

**15.**[**Функции пользователя**](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066350)

Функции - это базовые блоки С, в которых выполняются все операции. Стандартный вид функций следующий:  
спецификатор\_типа имя\_функции (список параметров)  
{тело функции}Спецификатор\_типа определяет тип возвращаемого функцией значения с помощью оператора return. Это может быть любой допустимый тип. Если тип не указан, предполагается, что функция возвращает целочисленные значения. Список параметров - это разделенный запятыми список переменных, получающий значение аргументов при вызове функции. Функция может быть без параметров и в таком случае список параметров содержит ключевое слово void.

#include <stdio.h> int max(int ms[10]) // прототип функции fun { int k=ms[0];

for(int i=1; i<10; i++) if(k<ms[i]) k=ms[i];

return k; } main() { int mas[3][4], i, j;

for(i=0; i<3; i++) { for(j=0; j<4; j++) scanf( ”%d”&mas[i][j]); printf(“\nмакс значение в строке %d =%d”, i,fun(mas[i])); } }

**16. Структура**

Структура - это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместн хранения информации. Объявл структуры приводит к образов шаблона, исп для создания объектов структуры.Переменные, образующие структуру, наз членами структуры. Обычно все члены структуры связаны друг с другом. Напр, информация об имени и адресе, наход в списке рассылки, обычно представл в виде структуры. Следующий фрагмент кода объявляет шаблон структуры, опред имя и адрес. Ключевое слово struct сообщает компилятору об объявлении структуры.  
struct addr {  
char name[30];char street [40]; char city[20];char state[3];};  
Объявление завершается точкой с запятой, поскольку объявление структуры - это оператор. Имя структуры addr идентифицирует структуру данных и является спецификатором типа. Имя структуры часто используют как ярлык.

**17. Объявление шаблонов структур**

Общий синтаксис объявления шаблона структуры: struct имя\_шаблона { тип1 имя\_переменной1; тип1 имя\_переменной1; //другие члены данных; }; struct DateBase { char fam[20]; char name[15]; long TelNumber; char \*Adress; double w; }; Имена шаблонов должны быть уникальными в пределах их области опред для того, чтобы компилятор мог различать различные типы шаблонов. Задание шаблона осущ с помощью ключевого слова struct, за которым следует имя шаблона структуры и список элементов, заключ в фигурные скобки. Имена элементов в одном шаблоне также должны быть уникальными. Однако в разных шаблонах можно исп одинак имена элементов. Задание только шаблона не влечет резервиров памяти компил-ом. Шаблон представляет компилятору необходимую инфу об элементах структурной переменной для резервирования места в оперативной памяти и организации доступа к ней при опред структурной переменой и исп отдельных элементов структурной переменной. Объявление заверш точкой с запятой, поскольку объявление структуры - это оператор.

**18. Определение структур-переменных.**

Определение структуры-переменной ничем не отличается от объявления обычной переменной с предопред типом. Общий синтаксис: struct имя\_шаблона имя\_переменной; Пример: struct person stud[10];

Компилятор выделит под каждую переменную количество байтов памяти, необходимое для хранения всех ее элементов. Разрешается совмещать описание шаблона и определение структурной переменной. Пример:

struct book { char title[20];

char autor[30]; double cast;

} book1, book2, \*ptr\_bk=&book1;

Пример: struct date {int day, month, year;} date1[5]; // объявление массив из 15 структур

**19. Доступ к компонентам структуры.**

Доступ к полям осущ с помощью оператора «.» при непосредственной работе со структурой или «->» - при исп указателей на структуру. Эти операторы наз селекторами членов класса. Общий синтаксис для доступа к компонентам структуры следующий: имя\_переменной\_структуры.член\_данных; имя\_указателя->имя\_поля;

(\*имя\_указателя).имя\_поля;

Пример: Прямой доступ к элементам

1) date1[5].day=10;

2) date1[5].year=1991;

3) strcpy(book1.title, “Война и мир”);

/\* исп прямое обращение к элементу, присваиваем значение выбранной переменной. Текст помещается в переменную, используя функцию копирования – strcpy(); \*/

4) stud[3].birthday.month=1;

5) stud[3].birthday.year=1980;

Доступ по указателю 1) (date1+5)->day=10; 2) (stud+3)->birthday.month=1;

// Используя доступ по указателю на структуру, присваиваем значение соответствующей переменной. Указатель можно использовать и так:

3) (\*(date1+5)).day=10;

4) (\*(stud+3)).birthday.month=1;

**20.** [**Анонимное опред структуры и оператор опред типа typedef**](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066355)

Язык Си предоставляет возможность опред по усмотрению программиста новые имена для любых типов данных. Для этого используется оператор опредтипа typedef, имеющ общий вид записи:Typedef\_тип\_новое\_имя\_типа\_,…,\_новое\_имя\_типа;Например описание:typedef int COUNT;делает имя COUNT синонимом типа int. Теперь можно исп COUNT во всех описаниях, операциях преобразов типов и в других случаях точно так же, как и ключевое слово int: COUNT x,y; /\* x,y переменные целого типа \*/COUNT mas[10]; /\* массив целых чисел.Анонимные структурные типы похожи на обычные структурные типы в C, но имеют ряд существенных отличий: для анонимной структуры нет явного назв типа. члены в анонимной структуре упорядочены, что позвол получ доступ к ним через индекс.они не обязаны иметь имени, они могут иметь только тип. анонимная структура может иметь несколько членов с одним и тем же именем, в этом случае при обращении по этому имени будет возвращён поток. анонимные структуры с одинаковой структурой (члены одинаковых классов в одинаковом порядке) совместимы, и значения могут быть присвоены.

#include<conio.h>#include <stdio.h>//Определяем новую структуруstruct point\_t {int x; int y;};//Опред новый типtypedef struct point\_t Point;void main({//Обращ через имя структуры struct point\_t p = {10, 20};

//Обращение через новый тип

Point px = {10,20};getch();}

**21. Статическая память в Си**

Статическое размещение более эффективно чем динамическое, так как выделение памяти происходит до выполнения программы, однако оно гораздо менее гибко, потому что мы должны заранее знать тип и размер размещаемого объекта.

-статические объекты обозначаются именованными переменными, и действия над этими объектами производятся напрямую, с исп их имен.

-выделение и освобождение памяти под стат объекты произв компилятором автоматически. Программисту не нужно самому заботиться об этом.

**22.** **Динамическая память**

Динамич распред памяти — способ выделения [операт памяти](http://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C" \o "Оперативная память) [компа](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) для объектов в проге, при котором выделение памяти под объект осущ во время выполнения [прог](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0" \o "Компьютерная программа)и.

Ни один объект не может наход в динам памяти без явного указания программиста. Динам объекты не имеют собственных имен, и действия над ними производятся косвенно, с помощью указателей. Выделение и освобожд памяти под динам объекты целиком и полностью возлагается на программиста.  Для работы с динам памятью сущ функции  malloc,calloc, realloc и free.

Функция malloc() возвращает указатель на первый байт области памяти размером size, которая была выделена из динам распред области памяти

Функция calloc() возвращ указатель на выдел память.

Функция realloc() изменяет величину выделенной памяти

Free (); -освобождение выделенной памяти.struct addr {

char name[40];

char street[40];

};struct addr \*get\_struct(void)

{struct addr \*p; if((p = malloc(sizeof(struct addr)))==NULL){ printf ("Ошибка при распределении памяти\n");exit(1);}eturn p;}

**23. Рекурсия**

Функция явл рекурсивной, если некоторая инструкция этой функции содержит вызов самой этой функции. Компилятор допускает любое число рекурс вызовов. Рекурсивный вызов не создает новую копию функции. При каждом вызове значения из предыд вызовов не теряются, но в каждый момент времени доступны только значения текущего вызова. Хотя компилятор языка СИ не ограничивает число рекурсивных вызовов функций, это число ограничивается ресурсом памяти компьютера. Большое число рекурсивных вызовов функции может привести к переполнению стека, что в свою очередь приведет к ошибочному окончанию работы программы. При разработке рекурс функции следует, исп операторы if и return, предусмотреть возможность завершения ее работы. В противном случае возможно "зацикливание" программы.#include <stdio.h>

int nod(int, int ); void main()

{ int a,b; fflush(stdin);

scanf("%d%d",&a,&b);

printf("НОД чисел %d и %d = %d",a,b,nod(a,b)); }

nod(int a,int b) // рекурсивная функция, вычисляющая { int c; // НОД чисел a и b

if (b>a) c=nod(b,a); else if (b<=0) c=a;

else c=nod(b,a%b); return(c); }

**24.Консольный ввод/вывод**

консольными функциями ввода/вывода наз те, которые выполняют ввод с клавиатуры и вывод на экран. В действительности же эти функции работают со стандартным потоком ввода и *стандартным потоком вывода*[[2]](http://lord-n.narod.ru/download/books/walla/programming/Spr_po_C/08/08.htm#22). Более того, *стандартный ввод*[[3]](http://lord-n.narod.ru/download/books/walla/programming/Spr_po_C/08/08.htm#33) и стандартный вывод[[4]](http://lord-n.narod.ru/download/books/walla/programming/Spr_po_C/08/08.htm#44) могут быть перенаправлены на другие устройства. консольными функциями ввода/вывода явл getchar(), которая читает символ с клавиатуры, и putchar(), которая отображает символ на экране. Первая из этих функций ожидает, пока не будет нажата клавиша, а затем возвращ значение этой клавиши. Вторая же функция, putchar(), отображает символ на экране в текущей позиции курсора. Вот прототипы функций getchar() и putchar():

int getchar(void);int putchar(int *c*);

#include <stdio.h>#include<conio.h>

main(){char c; c='A'; hile(c<='Z'){putchar(c);

c=c+1;}getch();return 0; }

**25.Работа с файлами, функции**

Файл - это упорядоч последовательность однотипных компонентов, располож на внешнем носителе.

Формат объявления указателя на файл след:***FILE\**** *ID\_указателя\_на\_файл;*

Открытие файла: FILE\* ***fopen***(char \* ID\_файла, char \*режим);

Режимы: *w* – файл открыв для записи (*write*); если файла с заданным именем нет, то он будет создан; если же такой файл уже сущ, то перед открытием прежняя информация уничтож;

*r* – файл открыв для чтения (*read*); если такого файла нет, то возникает ошибка;

*a* – файл открывается для добавления (*append*) новой информации в конец;

*r*+ (*w*+) – файл открыв для редакт данных, т.е. возможны и запись, и чтение информации;*a*+ – то же, что и для *a*, только запись можно выполнять в любое место файла (доступно и чтение файла);*t* – файл открывается в текстовом режиме; *b* – файл открывается в двоичном режиме;#include < stdio.h > int main (void){ FILE \*mf; printf (“Открытие файла: ”); mf=fopen (“myfile/test.txt” , ”w” ); if (mf == NULL) printf (“ошибка\n”);

else printf (“выполнено\n”); fprintf (mf,“Тест записи в файл”); printf “Запись в файл выполнена\n”); fclose (mf); printf (“Файл закрыт\n”); return 0; }

**26. Низкоуровневый ввод/вывод**

Самый низкий уровень ввода/вывода не предусматр никакой буферизации, а явл прямым обращением к ОС. Ввод и вывод осущ двумя функциями read() и write().Первым аргументом данной функции явл дескриптор, а вторым буфер в вашей программе откуда или куда будут поступать данные, третий аргумент – число подлежащий пересылке байт.Read(fd,buffer,n);

write(fd,buffer,n);При каждом обращении возвращается счётчик байтов, фактически переданных или принятых. N\_read=read(…);

При чтении число байт может быть меньше. Если в результате чтения вернулся 0, то это конец файла, если -1 – ошибка доступа. При записи возвращ количество дописанных байт. Если число записанных меньше чем желаемое, это ошибка. Количество байт подлежащих чтению или записи может быть любым.

#include<stdio.h> #define BUFSIZE 3

main() {char buf[BUFSIZE]; int n;

while((n=read(0,buf,BUFSIZE))>0)

write(1,buf,n); }

**27. Структуры данных и абстрактные типы данных**

Структуры данных- сложные данные, построенные из более простых, определенных с помощью языковых конструкций. Абстрактные типы данных -типы данных, рассматрив независимо от способов их представления или реализации средствами языка программирования. Как и обычные типы данных, АТД опред множеством значений и совокупностью операций, которые могут выполняться над значениями данного типа.Реализация АТД включает: - выбор необходимых структур данных, построенных на основе стандартных типов данных, исп языка программирования. - разработка процедур и функций для выполнения необходимых операций над объектами АТД. // объявить три указателя на элементы списка// указатель head ведет на первый элемент спискаstruct node\* head = NULL;struct node\* second = NULL;

struct node\* third = NULL;

// выделить память под элементы

head = malloc(sizeof(struct node));

second = malloc(sizeof(struct node));

third = malloc(sizeof(struct node));

// инициализировать элементы списка и связать их между собой head->data = 1;

head->next = second;second->data = 2;

second->next = third;third->data = 3;

third->next = NULL;

**28.Списки**

– это набор динамических элементов (чаще всего структурных

переменных) связанных между собой каким либо способом. Списки бывают линейными и кольцевыми, односвязными и двусвязными. Описание структуры и указателя может иметь вид: typedef struct element // структура элемента хранения

{ float val; // элемент списка

struct snd \*n ; // указатель на элемент хранения } SPIS; SPIS \*p; // указатель текущего элемента SPIS \*dl; // указатель на начало списка // объявить три указателя на элементы списка// указатель head ведет на первый элемент спискаstruct node\* head = NULL;struct node\* second = NULL;struct node\* third = NULL;// выделить память под элементы

head = malloc(sizeof(struct node));

second = malloc(sizeof(struct node));

third = malloc(sizeof(struct node));

// инициализировать элементы списка и связать их между собой head->data = 1;

head->next = second;second->data = 2;

second->next = third;third->data = 3;

third->next = NULL;

**29.Очереди и стек**

Частным случаем линейного односвязного списка является очередь ил стек.Очередь – список, в котором новый элемент добавл в конец, а удаляется только первый элемент.

Очередь функционирует по принципу FIFO. Заказы выполняются последовательно. Если при поступлении заказа, устройство свободно, заказ выполняется немедленно, иначе заказ помещается в конце очереди на обслуживание. На выполнение каждый раз поступает заказ хранящийся первый в очереди. После выполнения заказа след становится первым. Над элементами очереди выполн операции двух видов: добавление элемента в очередь, вывод элемента очереди с его удалением. Существуют очереди с приоритетами. От обычных очередей они отличаются тем, что элементы очередей имеют дополнительный параметр – приоритет, который определяет порядок удаления элементов из очереди. Поэтому из очереди удаляется не первый элемент, а тот у которого на данный момент наивысший приоритет.

Стек(Магазин) – представляет собой список упорядоченный по времени поступления элементов таким образом, что из вне доступен только последний из записанных. Стек функционирует по принципу LIFO. Поскольку из вне доступна только вершина стека, то основными операциями над стеком являются:Выборка с одновременным удалением вершины стека

Добавление нового элемента в вершину(Вершина стека – голова)

int stack[MAX];int tos=0; /\* вершина стека \*//\* Затолкать элемент в стек. \*/ void push(int i){ if(tos >= MAX) {printf("Стек полон\n"); return;} stack[tos] = i; tos++;}/\* Получить верхний элемент стека. \*/ int pop(void) { tos--; if(tos < 0) {printf("Стек пуст\n"); return 0; } return stack[tos]; }

**30. Деревья**

- представл собой совокупность элементов(узлов) между которыми установлены иерархические связи типа предок – потомок. Причём каждый из элементов может иметь несколько истинных потомков и только одного предка. Кроме одного элемента, называемого корнем дерева у которого нет предков. Узлы могут не иметь потомков, но предок есть всегда.Дерево без узлов – называется нулевым. Лист дерева – узел не имеющий потомков. Высота дерева –высота его корня. Глубина узла – длина пути от корня до узла. Существует неупорядоченное дерево, в котором порядок следования потомков узлов не учитывается. Упорядоченное дерево – дерево с определённым порядком следования потомков. Обычно потомки узла упорядочиваются слева направо.

Запишем рекурсивную процедуру, вывод на экран узлы дерева в порядке префиксного обхода.void prefix(t \*curr){if (!curr)return;  
printf("%d ", curr->data);  
prefix(curr->left);  
prefix(curr->right);}

**31.Графический режим работы**

Инициализация графики:  
В состав графического пакета входят: 1)заголовочный файл graphics.h, 2)библиотечный файл graphics.lib, 3)драйверы графических устройств (\*.bgi), • шрифты (\*.chr). Управление экраном в графическом режиме производится с помощью набора функций, прототипы которых находятся в заголовочном файле graphics.h. Для работы в графическом режиме файл graphics.h должен быть подключен с помощью директивы #include препроцессора языка Cи ко всем модулям, испграфические подпрограммы #include <graphics.h> Прежде чем исп графические функции, необходимо переключить видеоадаптер в графический режим (по умолчанию он находится в текстовом режиме). Для инициализации графики предназначена функция initgraph().

#include <Windows.h>#include <stdlib.h>

#include <math.h>#define P 3.14 int main(void){ float x; HDC hDC = GetDC(GetConsoleWindow()); HPEN Pen = CreatePen( PS\_SOLID, 2, RGB(100, 100, 150)); SelectObject( hDC, Pen ); MoveToEx( hDC, 0, 85, NULL ); LineTo( hDC, 200, 85 ); MoveToEx( hDC, 100, 0, NULL ); LineTo( hDC, 100, 170 ); for (x = -8.0f; x <= 8.0f; x += 0.01f ) // O(100,85) - center { MoveToEx( hDC, 10\*x+100, -10\*cos(x\*sin(P/4))+85, NULL );// LineTo( hDC, 10\*x+100, -10\*cos(x\*sin(P/4))+85 ); } getch(); return 0; }

**32. Agile**

Гибкая методология разработки (англ. Agile software development, agile-методы) — серия подходов к разработке программного обеспеч, ориентирна исп итеративной разработки, динамическое формирование требований. Большинство гибких методологий нацелены на минимизацию рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели. Каждая включает все задачи, необходимые для выдачи мини-прироста по функциональности: планирование, анализ требований, проектирование, программирование, тестирование и документирование. Основной метрикой agile-методов является рабочий продукт. agile-методы уменьшают объём письменной документации по сравнению с другими методами. Это привело к критике этих методов как недисциплинированных.

**33.Экстремальное программирование**

Экстремальное программирование – одна из гибких методологий разработки программного обеспечения.Приёмы экстремального программирования(ХР): 1)Короткий цикл обратной связи 2)Разработка через тестирование 3)Игра в клонирование 4)Заказчик всегда рядом 5)Парное программирование 6)Непрерывный а не пакетный процесс7)Непрерывная интеграция 8)Рефакторинг 9)Частые небольшие релизы 10)Простота

11)Метафора системы 12)Коллективное владение кодом или выборными шаблонами проектирования 13)Стандарт кодирования 14)Сорокачасовая рабочая неделя. Рассмотрим парное програ-ие:

Преимущества: Программисты в паре чаще «делают то, что нужно» и реже устраивают длинные перерывы. Партнёры в паре менее склонны к неудачным решениям и производят более качественный код. каждый несёт ответственность за весь код. Каждый, даже начинающий программист, знает что-то, чего не знают другие**.** Меньше прерываний.Экономическая обоснованность.Непрерывность проверки кода Команда разработчиков сплачивается, что позволяет закончить работу раньше и в лучшем качестве. Программисты постоянно обмениваются знаниями.

**34.ООП**

Объектно-ориентированное программирование (ООП) — подход к программированию, при котором основными концепциями явл понятия объектов и классов. ***Класс***— это определяемый разработчиком тип данных. ***Объект***— сущность обладающая определённым поведением и способом представления, т. е. относящееся к классу. Ключевые черты ООП : *Инкапсуляция* — это определение классов — пользовательских типов данных, объединяющих своё содержимое в единый тип и реализующих некоторые операции или методы над ним. Классы обычно являются основой модульности, инкапсуляции и абстракции данных в языках ООП. *Наследование* — способ определения нового типа, когда новый тип наследует элементы (свойства и методы) существующего, модифицируя или расширяя их. Это способствует выражению специализации и генерализации. *Полиморфизм*, позволяет единообразно ссылаться на объекты различных классов (обычно внутри некоторой иерархии). Это делает классы ещё удобнее и облегчает расширение и поддержку программ, основанных на них.

**35. Класс**

— это определяемый разработчиком тип данных.

Тип данных характеризуется: Способом хранения и представления этих данных.Назначением этих данных (профилем их исп).Набором действий, которые над этими данными могут производится.Класс позволяет программам группировать данные и функции которые выполняют операции над этими данными. class employee

{    char name[64] ;    long employee\_id;   
   float salary;    void show\_employee(void)

   {     cout << "Имя: " << name << endl;   
      cout << "Номер служащего: " << employee\_id << endl;      cout << "Оклад: " << salary << endl;    }; };

**36. Конструктор**

представляет собой метод класса, который облегчает программам инициализацию полей при создании объекта класса.

Конструктор имеет такое же имя, как и сам класс. Конструктор не имеет возвращаемого значения (по сути, результатом его работы является ссылка на созданный объект).

#include <iostream.h>#include <string.h>

class employee { public:  employee(char \*, long, float);  void show\_employee(void);   int change\_salary(float) ;  long get\_id(void); private:    char name [64] ;    long employee\_id;    float salary; }; employee::employee(char \*name, long employee\_id, float salary) {  strcpy(employee::name, name) ; employee::employee\_id = employee\_id; if (salary < 50000.0)   employee::salary = salary;    else // Недопустимый оклад   
   employee::salary = 0.0; }

void employee::show\_employee(void)

{    cout << "Служащий: " << name << endl;   cout << "Номер служащего: " << employee\_id << endl;  cout << "Оклад: " << salary << endl; }void main(void) {   
   employee worker("Happy Jamsa", 101, 10101.0);    worker.show\_employee(); }

деструктор представляет собой функцию, которая помогает корректно уничтожать объект. Деструктор обычно используется, если при уничтожении объекта нужно освободить динам память, занимаемую объектом. Деструктор имеет такое же имя, как и класс, но с символом (~) .

**37.** **Наследование и полиморфизм** **наследование**

— способ определения нового типа, когда новый тип наследует элементы (свойства и методы) существующего, модифицируя или расширяя их.

// Класс товаров в некотором магазине

class ShopItem { public:сhar model[100]; // модель char brand[100]; // производ

double price; // цена}; // Мобильные телефоны — наследник класса товаров

class MobilePhone : public ShopItem {public:char phoneColor[100]; // цвет

int batteryLife; // время работы от батареи в часах };

// Смартфоны — наследник класса мобильных телефонов class SmartPhone : public MobilePhone {public: char os[100]; // операционная система };

***Полиморфизм*** — это явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, в каком контексте он вызывается

**38.Перегрузка функций**

Перегрузка функций позволяет программам определять несколько функций с одним и тем же именем и типом возвращаемого значения.

#include <iostream.h>

int add\_values(int a,int b)

{    return(a + b); }

int add\_values (int a, int b, int c)

{   return(a + b + c); }

void main(void)

{    cout << "200 + 801 = " << add\_values(200, 801) << endl;   
   cout << "100 + 201 + 700 = " << add\_values(100, 201, 700) << endl; }

**39.Исключения в Си++**

Исключение -это некое действие, которое вступает в силу, при наступлении нестандартной ситуации.  
Пример #include <iostream>;

using namespace std;

 int main() {     setlocale(LC\_ALL, "rus");

 int num1;     int num2;   int var = 2;

 while(var--)    {        cout << "Введите значение num1: ";        cin >> num1;

cout << "Введите значение num2: ";

cin >> num2;  cout << "num1 + num2 = " << num1 + num2 << endl;    cout << "num1 / num2 = ";  try //код, который может привести к ошибке, располагается тут

  {    if (num2 == 0)    {        throw 123; //генерировать целое число 123

}     cout << num1 / num2 << endl;      }

catch(int i)//сюда передастся число 123

  { cout << "Ошибка №" << i << " - на 0 делить нельзя!!!!" << endl; } cout <<  "num1 - num2 = " << num1 - num2 << endl;   cout    }return 0;} исключение сработает так: программа получает конкретное указание от программиста — если значение определённой переменной в определённом участке кода (в try-блоке) будет равно 0, то в этом случае пусть генерируется исключение throw. Это исключение автоматически передастся catch-блоку в виде параметра и выполнится код этого блока.

1.[Массивы и указатели в СИ](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066336), пример

2.[Одномерные массивы, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066337)

3.[Многомерные массивы, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066338)

4.[Указатели, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066339)

5.[Строки, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066340)

6.[Стандартная библиотека работы](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066341)

7.[Поиск и сортировка, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066342)

8.[Организация последовательного](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066343)

9.[Организация бинарного поиска](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066344)

10.[Основные алгоритмы сортировки](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066345)

11.[Сортировка методом пузырька,](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066346)

12.[Сортировка методом выбора, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066347)

13.[Сортировка вставками, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066348)

14.[Быстрая сортировка, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066349)

15.[Функции пользователя в Си, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066350)

16.[Структуры в Си, пример](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066351)

17.[Объявление шаблонов структур](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066352)

18.[Объявление структур-переменных](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066353)

19.[Доступ к компонентам структуры](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066354)

20.[Анонимное определение структуры](file:///E:\ЗАГРУЗКИ\shpory_oaip.docx#_Toc244066355)

21.Статическая память в Си, пример

22.Динамическая память в Си, пример

23.Рекурсия, пример

24.Консольный ввод/вывод, пример

25.Работа с файлами, функции, пример

26.Низкоуровневый ввод/вывод, пример

27.Структуры данных и абстрактные типы данных, пример

28.Списки, пример

29.Очередь и стек, пример

30.Деревья, пример

31.Графический режим работы в Си,

32.Agile, пример

33.Экстремальное программирование

34.ООП, пример

35.Класс, пример

36.Конструктор, деструктор, пример

37.Наследование и полиморфизм

38.Перегрузка функция в С++, пример

39.Исключения в С++, пример