

Алгоритмизация и программирование

Ерохина Елена Альфредовна

Контрольные точки

1 модуль:

1. 10 лекций (проверочные работы или тесты)
2. 2 лабораторные работы
3. 4 семинара (проверочные работы или тесты)

2 модуль:

1. 12 лекций (проверочные работы)
2. 3 лабораторные работы
3. 7 семинаров (проверочные работы или тесты)
4. Контрольная работа (на семинаре)

Где найти материалы курса

Материалы курса размещаются в Smart Lms.

Учебный офис автоматически регистрирует студентов на курс.

При возникновении вопросов по регистрации обращайтесь в учебный офис к менеджеру образовательной программы.

Оценка за текущий контроль в 1 и 2 модуле учитывает результаты студента следующим образом.

Модуль 1. $O_{текущая\ 1} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски}$.

Модуль 2. $O_{текущая\ 2} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски} + O_{контр.\ работы}$.

Все оценки рассматриваются без округления.

Результирующая оценка учитывает оценки модулей. Промежуточная оценка за 1 и 2 модуль вычисляется по формуле

$O_{промежуточная\ 1\ и\ 2} = 0,3 * O_{текущая\ 1} + 0,5 * O_{текущая\ 2} + 0,2 * O_{экзамен\ 2\ модуль}$,

где $O_{текущая\ 1}$, $O_{текущая\ 2}$ – оценки текущего контроля 1, 2 модуля, без округления.

Округление производится один раз, после вычисления промежуточной оценки, по правилам арифметики.

Экзаменационная оценка не является блокирующей.
Промежуточная оценка за 1 и 2 модуль не может превышать 10 баллов, в случае превышения ставится промежуточная оценка 10 баллов.

Результирующая оценка за дисциплину вычисляется по формуле

$$O_{\text{результативная}} = 0,4 \cdot O_{\text{промежуточная 1 и 2}} + 0,6 \cdot O_{\text{промежуточная 3 и 4}}$$

Округление производится по правилам арифметики.

В диплом выставляется результирующая оценка.

Для вычисления текущей оценки по дисциплине используется следующая таблица (для групп БИВ).

	Работа на семинарском занятии	Работа на лекции	Выполнение лабораторного практикума	Контрольная работа
1 модуль	1	2	7 (3+4)	-
2 модуль	1	1	5(2+2+1)	3
3 модуль	1	1	5(1,5+1,5+2)	3
4 модуль	1	1	5(2,3+1,7+1)	3

В скобках указано распределение баллов по лабораторным работам.

Для вычисления текущей оценки по дисциплине используется следующая таблица (для групп БИТ).

	Работа на семинарском занятии	Работа на лекции	Выполнение лабораторного практикума	Контрольная работа
1 модуль	1	2	7 (3+4)	-
2 модуль	1	1	5(2+2+1)	3

В скобках указано распределение баллов по лабораторным работам.

- Ни один из элементов текущего контроля не является блокирующим.
- На некоторых семинарах и лекциях проводится тест или проверочная работа. Каждый вид работы оценивается от 1 до 4 баллов. В итоговую оценку эти баллы входят с коэффициентом, получаемым делением числа занятий, на которых проводилось оценивание, на общее количество занятий.
- При пропуске лекции или семинарского занятия по любой причине студент не может решить дополнительное задание для компенсации баллов, которые он мог бы получить на этом занятии.
- Кроме того, преподаватель может оценивать дополнительными баллами ответ студента у доски (максимум 2 балла) и активное участие в решении задач семинаров (например, выявление и исправление неточностей и ошибок в алгоритмах и при кодировании программ, внесение усовершенствований в алгоритм и т.п.) (максимум по 0.2 балла за каждый ответ).

- Для каждой лабораторной работы устанавливается срок защиты отчета (в 1 модуле на 2 и 4 занятиях, считая пары отдельно для каждой подгруппы). При своевременной защите работа оценивается полученным баллом, при опоздании на 1 неделю балл снижается на 40%, при опоздании на 2 недели балл снижается на 60% от полученной оценки. При опоздании более чем на 2 недели работа не оценивается.
- В случае пропуска занятий по уважительной причине (обязательно предоставление справки) срок сдачи лабораторной работы может быть перенесен на соответствующее количество рабочих дней.
- В случае пропуска занятий по уважительной причине (обязательно предоставление справки) предоставляется дополнительное время для написания контрольной работы (единственная дата переписывания заранее сообщается через старосту).
- Переписывание контрольной работы с целью повышения полученной оценки не допускается.

Требования к оформлению отчета

Выполнение каждой работы лабораторного практикума завершается написанием отчета, включающего следующие разделы:

- титульный лист
- содержание;
- задание;
- постановка задачи - 0,5;
- метод решения задачи - 1;
- внешняя спецификация - 0,5;
- описание алгоритма на псевдокоде - 1,5;
- листинг программы - 0,5 +1 программа работает;
- распечатка тестов к программе и результатов – 1;
- вопросы по отчету – 2;
- дополнительное задание – 2.

Через тире указан вес каждого раздела в оценке за выполнение лабораторной работы.

Требования к оформлению отчета

- Для оформления отчета используется формат бумаги А4.
- Печать отчета предусматривает распечатку титульного листа и страницы с заданием. Остальная часть отчета предъявляется для защиты в электронном виде.
- Каждый раздел отчета начинается с нового листа.
- Правки в итоговый отчете нужно внести после исправления всех замечаний.
- Итоговый отчет отправляется на почту преподавателю (или учебному ассистенту), принимавшему отчет.
- Итоговая оценка выставляется на титульном листе отчёта. Титульный лист сдается преподавателю.
- Студенту необходимо сделать фото титульного листа с оценкой и сохранять его до конца модуля.

**Федеральное государственное автономное образовательное учреждение
высшего образования
"Национальный исследовательский университет
"Высшая школа экономики"**

Московский институт электроники и математики им. А.Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии (или департамент электронной инженерии)

Курс: Алгоритмизация и программирование

Раздел	Макс оценка	Итог. оценка
Постановка	0,5	
Метод	1	
Спецификация	0,5	
Алгоритм	1,5	
Работа программы	1	
Листинг	0,5	
Тесты	1	
Вопросы	2	
Доп. задание	2	

**ОТЧЕТ
по лабораторной работе №_____**

Студент: _____ ФИО _____

Группа: _____ номер группы _____

Вариант: № _____ (номера заданий)

Руководитель: _____

Оценка: _____

Дата сдачи _____

МОСКВА 2022

Пример оформления отчета по лабораторной работе 1

Задание.

1. Даны n , x , h , a . Вычислить массив $R[1:n]$ в соответствии с формулами: $R[i]=2,5\sin(ax+i^2h)$, $i=\overline{1, n}$
2. Из вычисленного массива R удалить все отрицательные элементы, расположенные между первым минимальным и последним положительным элементами.
3. В полученном массиве $R[1:k]$, где k – число элементов, оставшихся после удаления, подсчитать среднее арифметическое элементов, расположенных до первого отрицательного элемента.

Постановка задачи

Дано:

1. n-цел., x, h, a-вещ.
2. Нет входных данных
3. Нет входных данных

Результат:

1. R[1:n]-вещ.
2. R[1:k]-вещ. или сообщение <<Нет положительного элемента>> или сообщение <<Первый минимальный и последний положительный расположены рядом или совпадают>>
3. sr-вещ. или сообщение <<Нет среднего значения>>

При: $n \in \mathbb{N}, n \leq l_{max}$.

Связь:

1. См. формулу в условии

2. $\exists n1: \forall i = \overline{1, n}$

$$R[n1] \leq R[i]$$

$$\min = R[n1]$$

$$\exists t: t = \overline{1, n1 - 1}; R[t] = \min$$

$$\exists np: np = \overline{1, n}: R[np] > 0, \exists q: q = \overline{np + 1, n}: R[q] > 0$$

$$c = \min(n1, np); b = \max(n1, np)$$

$$\forall i = \overline{c + 1, b - 1}: R[i] \geq 0 \quad \exists t \in [c + 1, l]: R[t] = R[i]$$

$$\forall i = \overline{b, n} \exists p \in [l + 1, k]: R[p] = R[i]$$

3. $\exists n1: n1 = \overline{1, k}: R[n1] < 0, \exists t: t = \overline{1, n1 - 1}: R[t] < 0$

$$sr = \sum_{i=1}^{n1-1} R[i]/(n1 - 1)$$

Метод решения задачи

1. $\begin{cases} \text{для } i = \overline{1, n} \\ r[i] = 2,5\sin(ax + i^2h) \end{cases}$

2. np=0

n1=1

k=n

$\begin{cases} \text{для } i = \overline{1, n} \end{cases}$

$\begin{cases} np = i, \text{ если } r[i] > 0; \\ n1 = i, \text{ если } r[i] < r[n1] \end{cases}$

c=n1; b=np, если n1<np

c=np; b=n1, если np≤n1

k=c

$\begin{cases} \text{для } i = \overline{c + 1, b - 1} \end{cases}$

$\begin{cases} k = k + 1; r[k] = r[i], \text{ если } r[i] \geq 0 \end{cases}$

$\begin{cases} \text{для } i = \overline{b, n} \end{cases}$

$\begin{cases} k = k + 1; r[k] = r[i] \end{cases}$

3. n1=0

$\begin{cases} \text{для } i = \overline{k, 1} \end{cases}$

$\begin{cases} n1 = i, \text{ если } r[i] < 0 \end{cases}$

sr=0

$\begin{cases} \text{для } i = \overline{1, n1 - 1} \end{cases}$

$\begin{cases} sr = sr + r[i] \end{cases}$

$\begin{cases} sr = sr / (n1 - 1) \end{cases}$

Внешняя спецификация

Лабораторная работа №1

Задание 1

Введите длину массива R от 1 до <<lmax>>:

{< n >}^{*} до n>0 и n≤lmax

Введите x, h, a:

<x> <h> <a>

Массив R из <<n>> элементов

<<R[1]>> <<R[2]>> ... <<R[n]>>

Задание 2

При $pr=0$

{ Нет положительного элемента

Иначе

{ при $|pr - n1| < 2$

{ Первый минимальный и последний положительный расположены рядом или совпадают

{ иначе

{ Массив r состоит из « k » элементов
« $r[1]$ » « $r[2]$ » … « $r[k]$ »

Задание 3

при $n1-1 \leq 0$

{ Нет среднего значения

иначе

{ $sr = << sr >>$

Описание алгоритма на псевдокоде

Алг «Лабораторная работа №1»

нач

{**задание 1**}

{ввод исходных данных для задания 1}

вывод(« Лабораторная работа №1.Задание 1»)

вывод(«Ведите длину массива R от 1 до », lmax)

цикл

ввод(n)

до n>0 и n≤lmax

кц

вывод(«Ведите x, h, a:»)

ввод(x, h, a)

цикл от i:=1 до n

r[i]:=2,5sin(ax+hi²)

кц

вывод(«Массив R из », n, « элементов: ») вывод(r[1:n])

{Задание 2}

вывод(«Задание 2»)

k:=n {число элементов, оставшихся после удаления}

n1:=1 { номер первого минимального элемента}

np:=0 { номер последнего положительного элемента}

цикл от i:=1 до n

если r[i]>0 то np:=i

всё

если r[i]<r[n1] то n1:=i

всё

кц

{анализ существования результата и вывод результата задания 2}

если np=0 то

вывод(«Нет положительного элемента»)

иначе

если |np-n1|<2 то

вывод(«Первый минимальный и последний положительный
расположены рядом или совпадают»)

иначе

{с и b – начало и конец зоны удаления}

если n1<np то

 c:=n1; b:=np

иначе

 c:=np; b:=n1

всё

k:=c

цикл от i:=c+1 до b-1

если r[i]≥0 то

 k:=k+1

 r[k]:=r[i]

всё

КЦ

цикл от i:=b до n

 k:=k+1

 r[k]:=r[i]

КЦ

вывод («Массив r из », k, « элементов»)

вывод (r[1:k])

всё

всё

{Задание3}

вывод(«Задание 3»)

n1:=0 {номер первого отрицательного элемента}

цикл от i:=k до 1 шаг -1

если r[i]<0 то

n1:=i

всё

кц

если n1-1≤0 то

вывод(«Нет среднего значения»)

иначе

sr:=0

цикл от i:=1 до n1-1

sr:=sr+r[i]

кц

sr:=sr/(n1-1)

вывод(sr)

всё

кон

Листинг программы

```
#include <stdio.h>
#include <math.h>
#define lmax 200
int main()
{
    int n, c, b, n1, np, i, k;
    float x, h, a, sr, r[lmax];
    printf(«Лабораторная работа №1\n»);
    printf(«Задание №1\n»);
```

```
//Задание 1
do
{printf(«Введите длину массива R от 1 до %d:»,
, lmax);
    scanf ("%d", &n);
}
while (n <= 0 || n > lmax);
printf(«Введите x, h, a:»);
scanf ("%f", &x);
scanf ("%f", &h);
scanf ("%f", &a);
for( i = 1; i<=n; i++)
    r[i] = 2.5 * sin(a * x + h * i * i);
printf(«Массив R из %d элементов:\n»);
for( i = 1; i<=n; i++)
    printf("%8.3f ", r[i]);
printf ("\n");
```

```
//задание 2
k = n;
printf(«Задание №2\n»);
for(n1 = 1, np = 0, i = 1; i<=n; i++)
{
    if (r[i] > 0 )
        np = i;
    if (r[i] < r[n1])
        n1 = i;
}
if (np == 0 )
    printf(" Нет положительного элемента \n");
else
    if (abs(np - n1) < 2 )
        printf(" Первый минимальный и последний
положительный расположены рядом или совпадают
\n");
```

```
else
{
    if  (n1 < np    )
    {
        c = n1;
        b = np;
    }
else
{
    c = np;
    b = n1;
} ;
k = c;
for (i = c + 1; i<= b - 1;i++)
    if (r[i] >= 0 )
    {
        k = k + 1;
        r[k] = r[i];
    }
}
```

```
for (i = b; i<= n; i++)
{
    k = k + 1;
    r[k] = r[i];
}
printf(«Массив R из %d
элементов:\n», k);
for( i = 1;i<=k; i++)
    printf("%8.3f ",r[i]);
printf("\n");
};
```

```
//Задание 3
    printf(" Задание №3\n");
for(n1 = 0, i = k; i>=1; i--)
    if (r[i] < 0 )
        n1 = i;
if (n1 - 1 <= 0 )
    printf(" Нет среднего значения ");
else
{ sr = 0;
    for( i = 1;i<=n1-1; i++)
        sr = sr + r[i];
    sr = sr / (n1 - 1);
    printf("sr=%8.3f\n", sr);
}
}
```

Распечатка тестов к программе и результатов

№	Исходные данные	Результаты
1	$n=10$ $x=2; h=4; a=5$	$R=\{2.477; 1.906; 2.254; -2.463; -0.111; -0.155; -2.437; 2.150; 2.092; 2.499\}$ $R=\{2.477; 1.906; 2.254; -2.463; 2.150; 2.092; 2.499\}$ $Sr=2.212$
2	$n=1$ $x=5; h=8; a=7$	$R=\{-2.079\}$ Нет положительного элемента Нет среднего значения
3	$n=5$ $x=5; h=2; a=4$	$R=\{-0,022; 0.677; 0.741; 2.467; 1.935\}$ $R=\{-0,022; 0.677; 0.741; 2.467; 1.935\}$ Нет среднего значения

Для выбора набора заданий используйте формулы:

Пример

№ варианта	X – номер варианта	X=13
Задание 1	$(\text{Остаток от деления } x \text{ на } 7) + 1$	7
Задание 2	$(\text{Остаток от деления } x \text{ на } 9) + 1$	5
Задание 3	$(\text{Остаток от деления } x \text{ на } 10) + 1$	4

7, 9, 10 – количество вариантов заданий в л.р. 1.

Номера вариантов указаны в журнале на страницах групп. Также список номеров вариантов есть в Smart Ims.

Методические указания по лабораторному практикуму размещены в Smart Ims.

Язык программирования — формальная знаковая система,

предназначенная для записи компьютерных программ.

Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под ее управлением.



Язык высокого уровня - язык программирования, средства которого обеспечивают описание задачи в наглядном, легко воспринимаемом виде, удобном для программиста. Он не зависит от внутренних машинных кодов ЭВМ любого типа, поэтому программы, написанные на языках высокого уровня, требуют перевода в машинные коды программами компилятора либо интерпретатора.

Язык низкого уровня - язык программирования, предназначенный для определенного типа ЭВМ и отражающий его внутренний машинный код.

Программирование

Процедурное

Операционное (Assembler, Fortran, Basic, C)

Структурное (Pascal, Modula, C)

Непроцедурное

Объектное (Smalltalk, C++, Delphi)

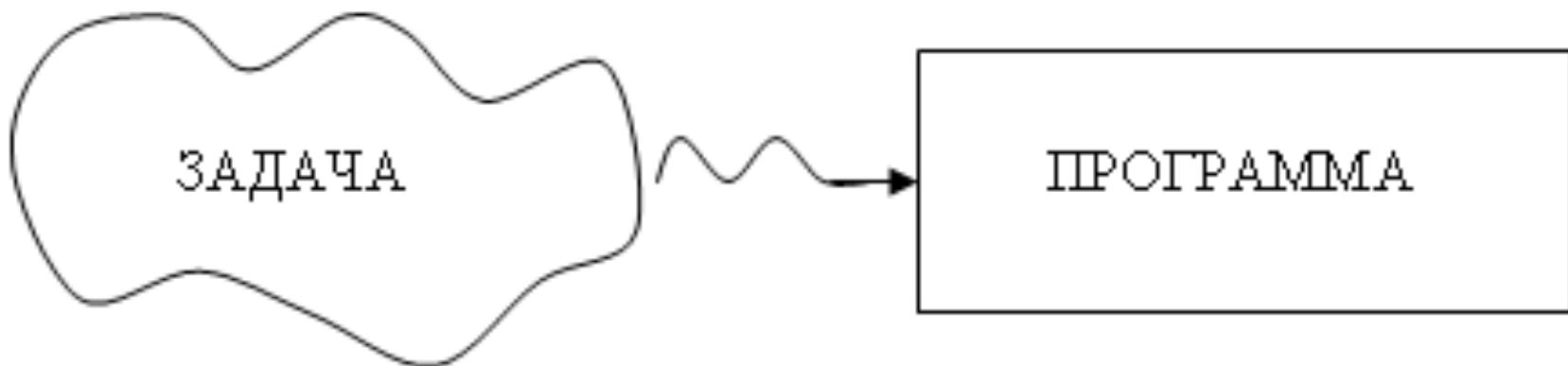
Декларативное

Логическое (Prolog)

Функциональное (Lisp)

Три подхода к составлению алгоритмов и программ

1. «Традиционный»





BS - программа

Недостатки метода

- программы трудно читать;
- трудно искать и исправлять ошибки в программах;
- почти невозможно вносить изменения в программу;
- этому «искусству» трудно научиться и невозможно обучить.

2. Структурный



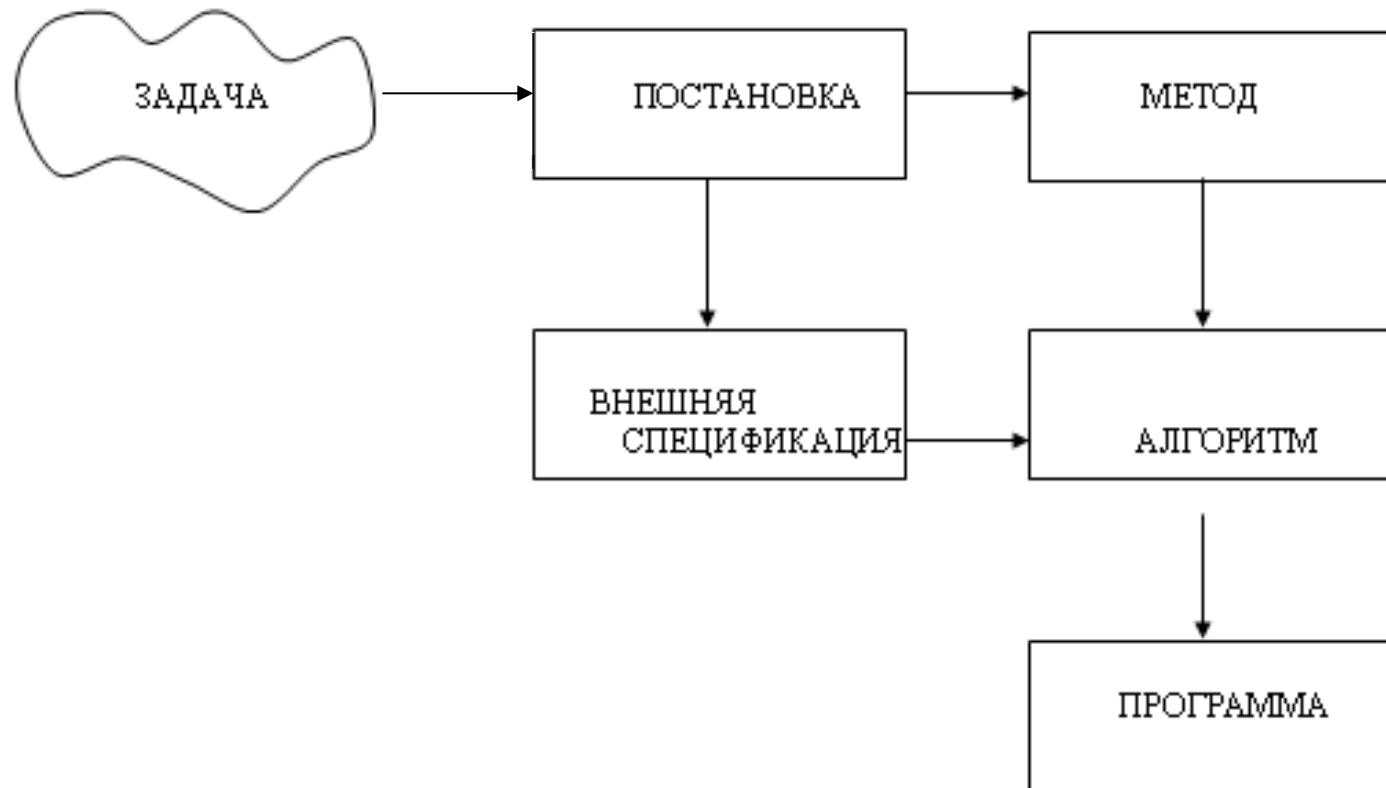
Преимущества

- Есть алгоритм, по которому можно разобраться в работе программы;
- удобно изменять алгоритм и программу;
- если алгоритм структурный, то и программа структурирована.

Недостатки

- Переход от задачи к алгоритму не формализован, т.е. неясно, как он происходит;
- если алгоритм не структурный, то и структура программы может быть нечеткой.

3. Систематический



Преимущества метода:

- надежность программ;
- эффективность (программы легко оптимизировать);
- экономичность (сокращается время отладки);
- эргономичность (алгоритмы удобны для чтения и понимания);
- модифицируемость (программы легко изменять);
- полная документированность программ;
- этот метод эффективен при обучении программированию.

Определение структурного программирования

Структурное программирование – это
программирование с использованием
только четырех базовых конструкций
(структур).

Основные алгоритмические структуры (конструкции)

- простая последовательность действий

- условная конструкция (выбор)

ifelse

switch ...case

- циклы:

- от ... до (со счетчиком)

for ...

- с предусловием

while

- с постусловием

do

- подпрограммы:

функции

Основные определения

- **Постановка задачи** – математически точное описание поставленной задачи.
- **Метод решения задачи** – обобщенный способ решения задач данного класса.
- **Внешняя спецификация** – описание входных и выходных данных задачи, а также всех вариантов хода диалога пользователя и программы.
- **Алгоритм** – набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий.
- **Программа** – алгоритм, записанный на заданном языке программирования.

Общий вид постановки задачи

Задача: <условие задачи>

Дано: <исходные данные>

Результат: <результаты решения задачи>

При: <ограничения на допустимость
исходных данных>

Связь: < как связаны «дано» и «результат»>

Постановка задачи

Задача: решить линейное
уравнение $ax + b = 0$

Вариант 1.

Дано: a, b .

Результат: x .

При: $a \neq 0$.

Связь: $ax + b = 0$

Вариант 2.

Дано: a, b .

Результат: x ,

или сообщение «нет корней»,

или сообщение «корень –
любое число».

При: -

Связь: $ax + b = 0$

Метод решения задачи

- При $a \neq 0$ $x = -b / a$.
- При $a=0$ и $b=0$ сообщение «корень – любое число».
- При $a=0$ и $b \neq 0$ сообщение «нет корней».

Внешняя спецификация

Решение линейного уравнения
Введите коэффициенты a , b
 $< a >$ $< b >$

При $a \neq 0$

{ Корень уравнения $x = << x >>$

При $a=0$ и $b=0$

{ Корень – любое число

При $a=0$ и $b \neq 0$

{ Нет корней

Обозначения

- <> - ввод данных пользователем;
- <<>> - вывод результата;
- { - показывает связь условия и соответствующих действий;
- { }* - повторение действий.

Алгоритм (на псевдокоде)

Алг «решение линейного уравнения»

Нач

вывод(«Решение линейного уравнения »)

вывод(«введите коэффициенты a,b»)

ввод(a,b)

если $a \neq 0$ то

$x := -b/a$

вывод(«Корень уравнения $x =$ »,x)

иначе

если $b=0$ то

вывод(«корень – любое число»)

иначе

вывод(«нет корней»)

всё

всё

Кон

a

вещ

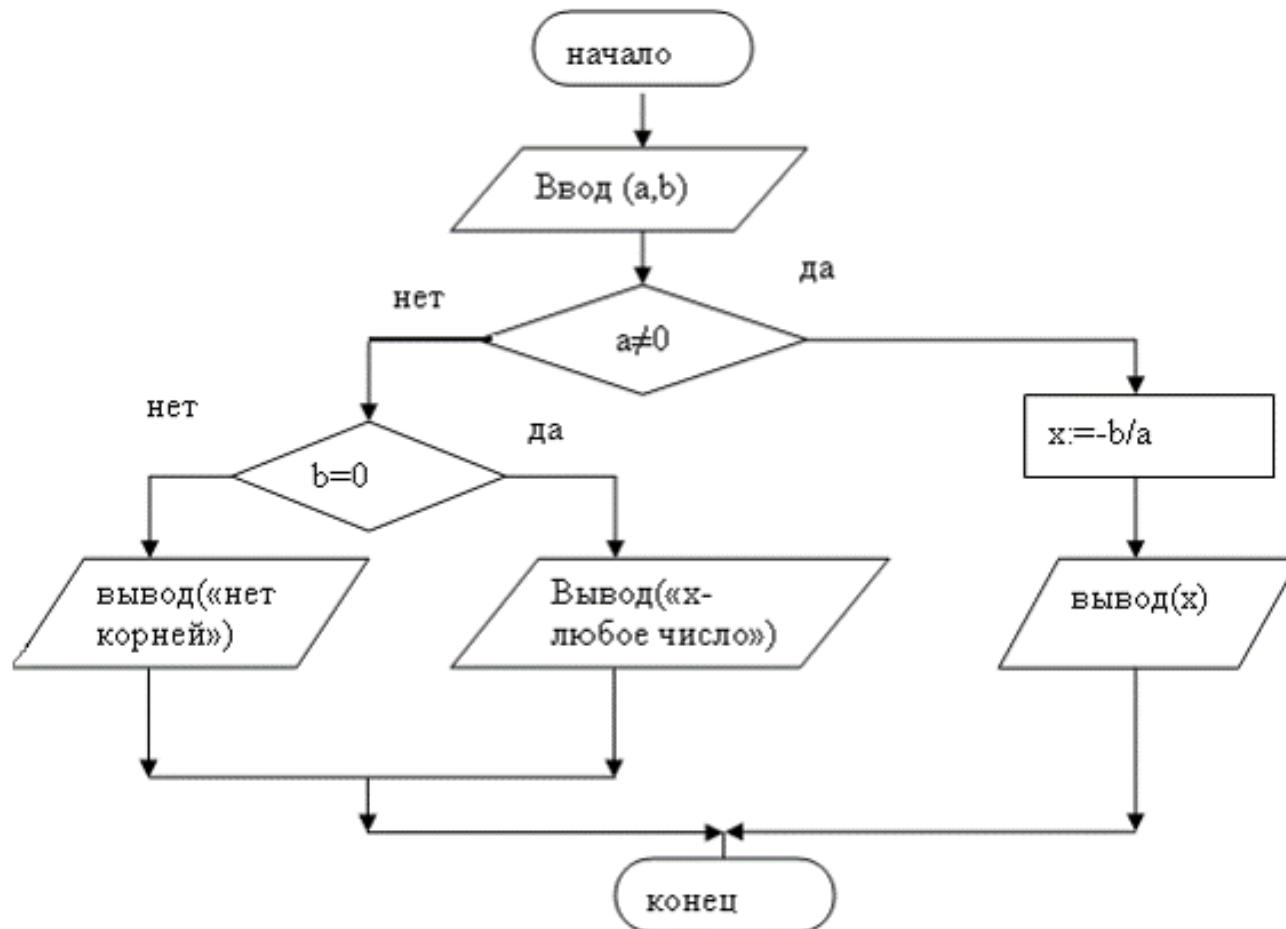
b

вещ

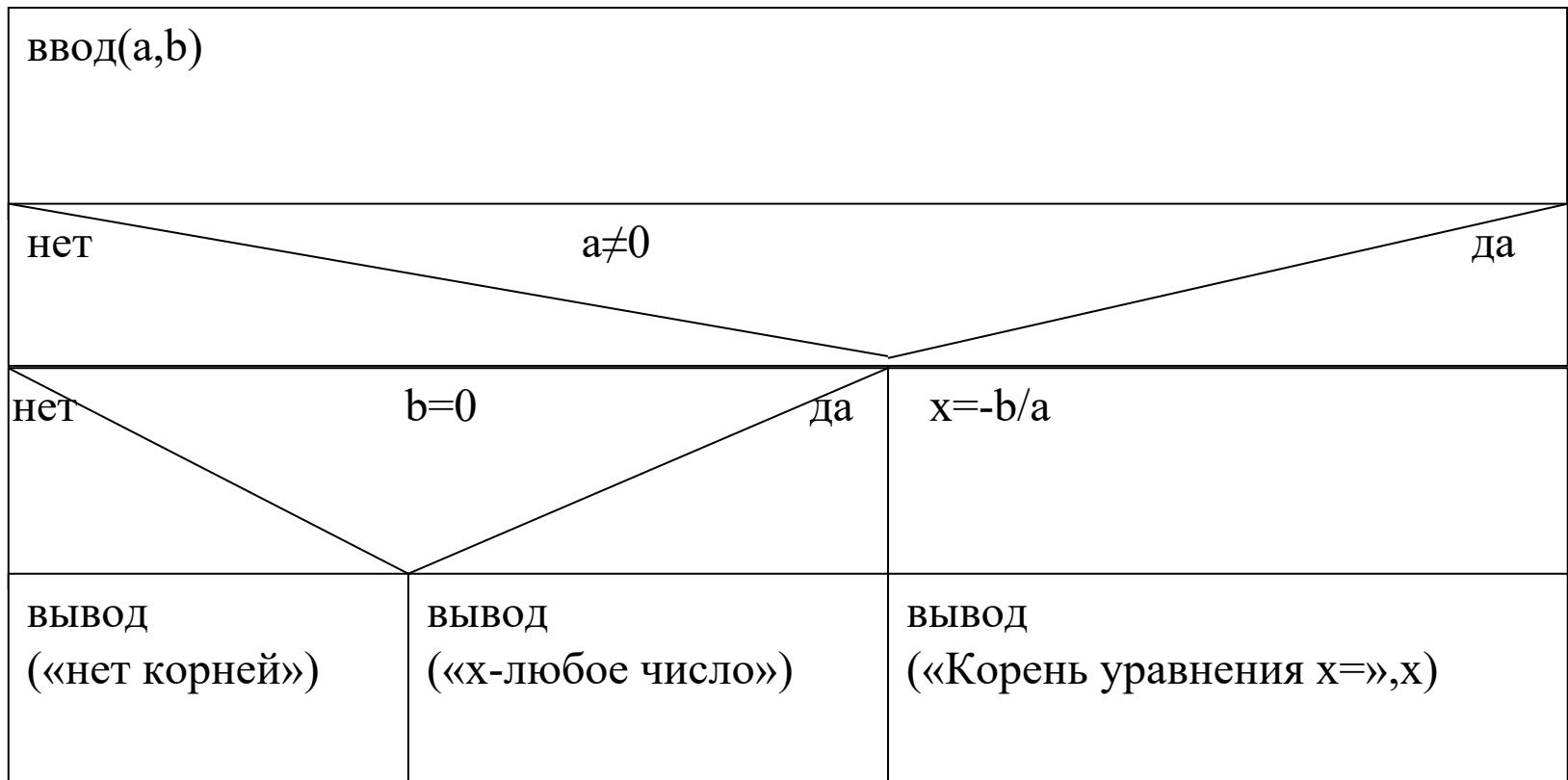
x

вещ

Алгоритм (блок-схема)



Алгоритм (диаграмма Нейсси-Шнейдермана)



Сравнение различных способов записи алгоритмов и программ

1. Наиболее компактны блок-схема и диаграмма Нейсси-Шнейдермана.
2. Блок-схема наглядна для небольших алгоритмов. Для больших задач это преимущество теряется.
3. При использовании блок-схем можно получить не структурный алгоритм.
4. Псевдокод достаточно компактен и нагляден. По псевдокоду удобнее всего кодировать (его конструкции близки к языкам программирования, особенно Pascal).

Свойства алгоритмов

- **Дискретность** — алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов.
- **Детерминированность** (определенность) - в каждый момент времени следующий шаг работы однозначно определяется состоянием системы.
- **Понятность** — алгоритм должен включать только те команды, которые доступны исполнителю и входят в его систему команд.
- **Завершаемость** (конечность) — при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
- **Массовость** (универсальность) - алгоритм должен быть применим к разным наборам исходных данных.
- **Результативность** — завершение алгоритма определенными результатами.
- Алгоритм **содержит ошибки**, если приводит к получению неправильных результатов либо не даёт результатов вовсе.
- Алгоритм **не содержит ошибок**, если он даёт правильные результаты для любых допустимых исходных данных.

Понятие переменной

Общий вид алгоритма

алг «название алгоритма»

нач

<команды>

кон

<перечисление переменных
с указанием их типов>

<имя>

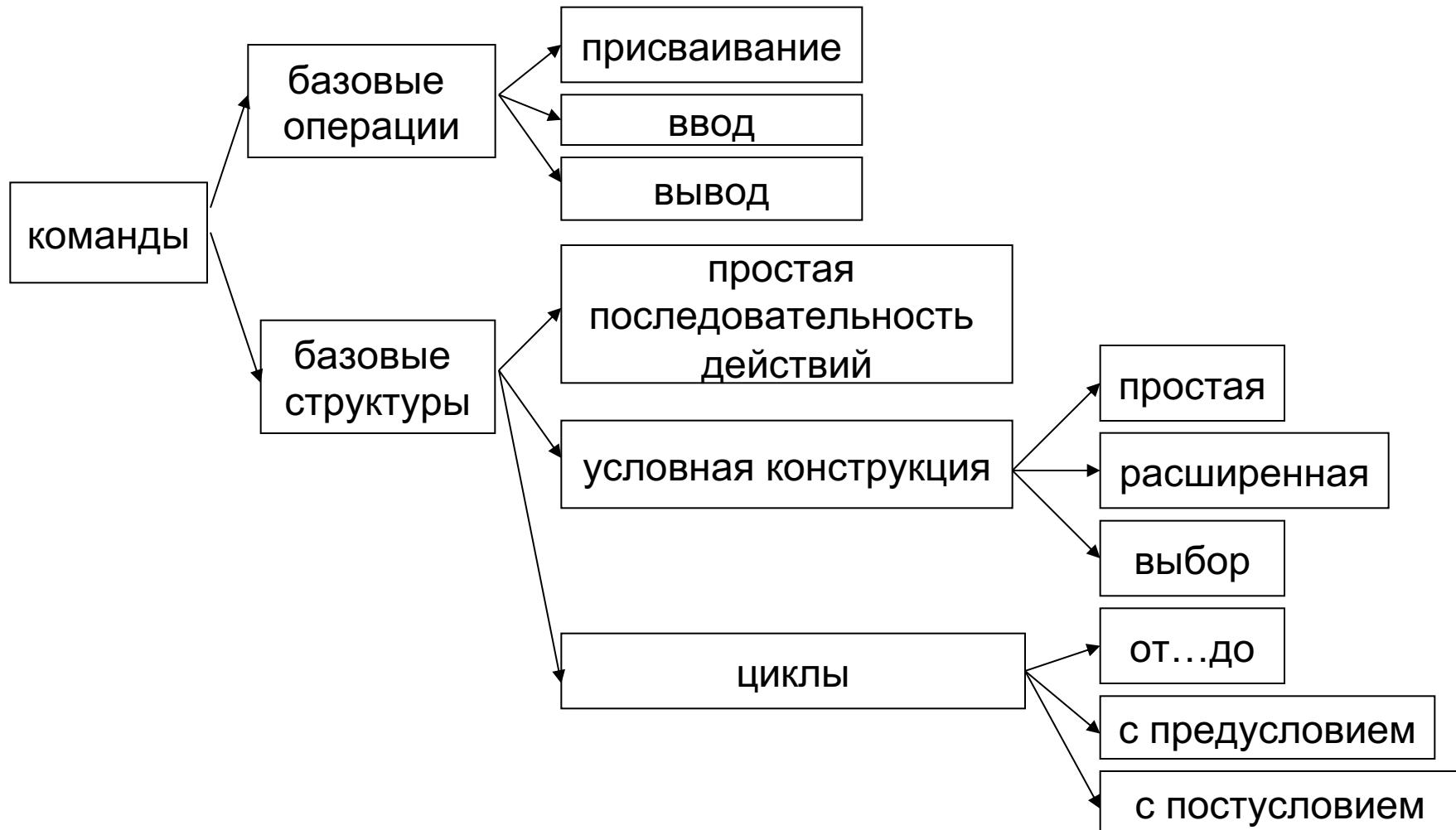
[<значение>]

<тип>

a - целая b – вещественная



Классификация команд



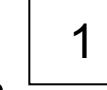
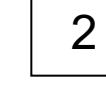
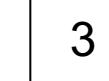
Операция присваивания

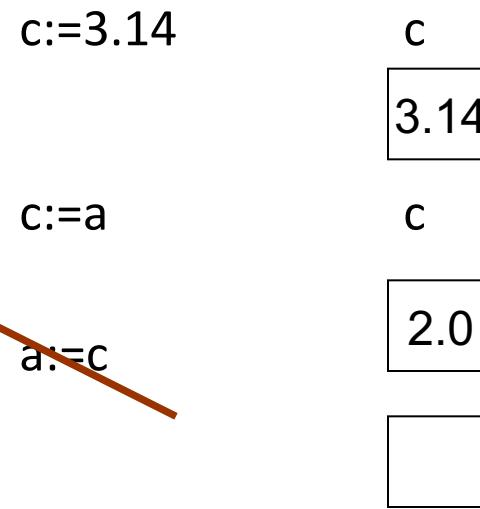
<имя переменной>:=<выражение>

Порядок выполнения

1. Вычисляется значение выражения, расположенного справа от знака :=.
2. Полученное значение присваивается переменной, имя которой расположено слева от знака :=.

Операция присваивания

a	b	c
		
цел	цел	вещ
	a	
$a := 1$		
	a	
$a := a + 1$		
	a	
$a := b$		
$b := a + 1$	b	
		a



Недопустимо – несоответствие типов.
Тип выражения должен b
типу переменной
или быть приводимым к нему.

Операция ввода

Общий вид:

ввод (<список ввода>)

1. ввод (a, b, c)

1 „13 „0 ↵

a b c

2. ввод (x, y)

15 ↵

x y ожидается ввод недостающих данных

3. ввод(d, e)

32 „0„17 ↵

d e лишние данные либо игнорируются, либо
используются при следующем вводе

Операция вывода

Общий вид

вывод (<список вывода>)

1. вывод ("a=", a, "b=", b)

```
a=1b=13
```

2. вывод ("a=", a, ", ", _b=", b)

```
a=1, _b=13
```

Форматы в списке вывода не отображаются

Постановка задачи

Задача: решить квадратное уравнение $ax^2 + bx + c = 0$

Дано: a, b, c -вещ.

Результат: «уравнение не квадратное»,

или x_1, x_2 -вещ,

или x_1 - вещ,

или сообщение «нет действительных корней».

При: -.

Связь: $ax_1^2 + bx_1 + c = 0$

$$ax_2^2 + bx_2 + c = 0$$

или

$$ax_1^2 + bx_1 + c = 0$$

Метод решения задачи

- При $a \neq 0$ $d = b^2 - 4ac.$

- При $d=0$ $x_1 = -b/(2a)$

- При $d \neq 0$

$$x_1 = \frac{-b + \sqrt{d}}{2a}$$

$$x_2 = \frac{-b - \sqrt{d}}{2a}$$

Внешняя спецификация

```
Решение квадратного уравнения  
Введите коэффициенты a, b, c  
<a> <b> <c>
```

При $a=0$

{ Уравнение не квадратное

иначе

При $d<0$

{ Нет действительных корней
иначе

При $d=0$

{ $x_1 = <<x_1>>$

иначе

{
 $x_1 = <<x_1>>$
 $x_2 = <<x_2>>$

Обозначения

- <> - ввод данных пользователем;
- <<>> - вывод результата;
- { - показывает связь условия и соответствующих действий;
- { }* - повторение действий.

Алгоритм

Алг «решение квадратного уравнения»

Нач вывод(«Решение квадратного уравнения »)

 вывод(«введите коэффициенты а, b, с»)

 ввод(а, b, c)

 если а=0 то

 вывод(«Уравнение не квадратное»)

 иначе

 d:=b²-4ac

 если d<0 то

 вывод(«нет действительных корней»)

 иначе

 если d=0 то

 x1:=-b/(2a)

 вывод(«x1=», x1)

 иначе

$$x1 := \frac{-b + \sqrt{d}}{2a} ; \quad x2 := \frac{-b - \sqrt{d}}{2a}$$

 вывод(«x1=», x1)

 вывод(«x2=», x2)

 всё

 всё

всё

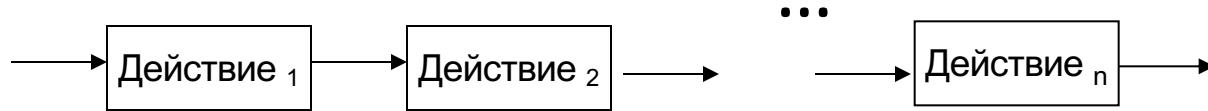
Кон

a	<input type="text"/>	вещ
b	<input type="text"/>	вещ
c	<input type="text"/>	вещ
d	<input type="text"/>	вещ
x1	<input type="text"/>	вещ
x2	<input type="text"/>	вещ

Основные алгоритмические конструкции

Простая последовательность действий

Блок-схема



Псевдокод

<действие₁>

<действие₂>

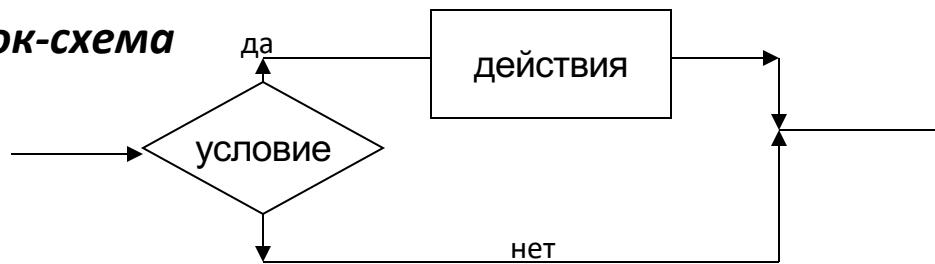
...

<действие_n>

Условная конструкция (выбор)

- Простейшая

Блок-схема



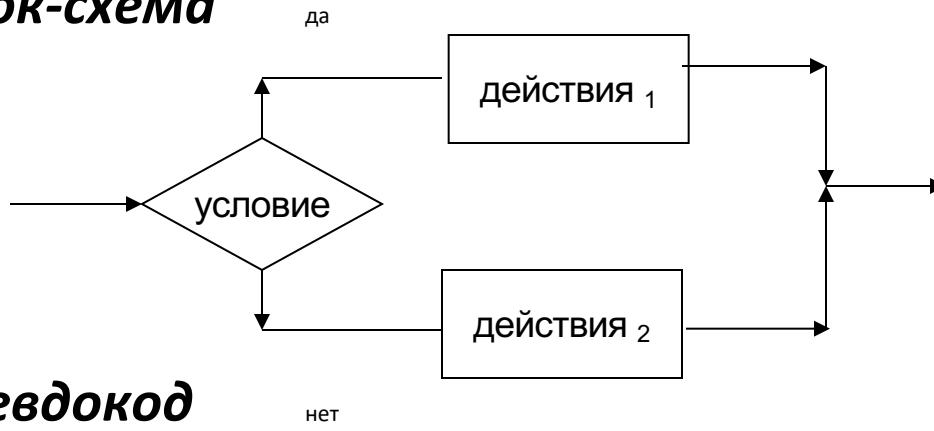
Псевдокод

```
если <условие> то  
    <действия>  
всё
```

Условная конструкция (выбор)

- Расширенная

Блок-схема



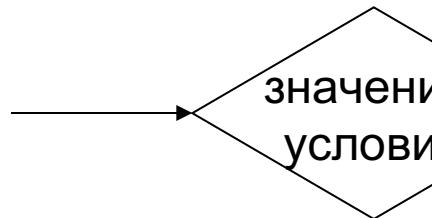
Псевдокод

```
если <условие> то  
    <действия 1>  
иначе  
    <действия 2>  
всё
```

Условная конструкция (выбор)

- Выбор

Блок-схема



Псевдокод
выбор (n)

$n=1: <$ действия $_1>$

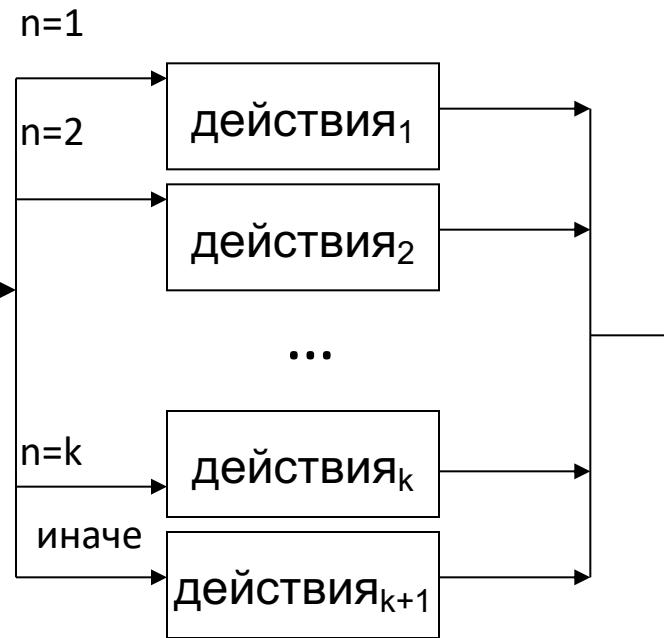
$n=2: <$ действия $_2>$

...

$n=k: <$ действия $_k>$

[иначе $<$ действия $_{k+1}>$]

квыбор



Циклы

- От ... до (со счетчиком)



Псевдокод

цикл от i:=<н.з.> до <к.з.> [шаг <приращение>]
<действия>

кц

, где

i – переменная (счетчик) цикла,

<н.з.> - начальное значение счетчика,

<к.з.> - конечное значение счетчика,

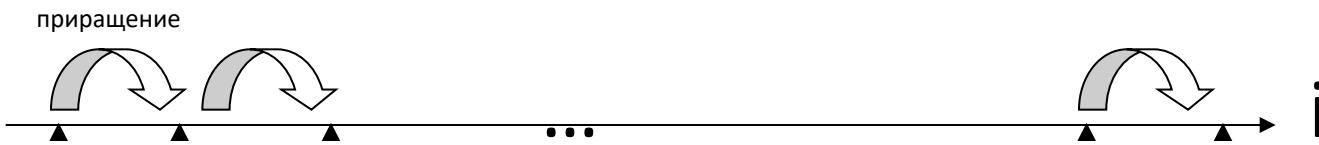
<приращение> - шаг, с которым изменяется значение счетчика.

Если шаг не указан, то он равен 1.

Циклы

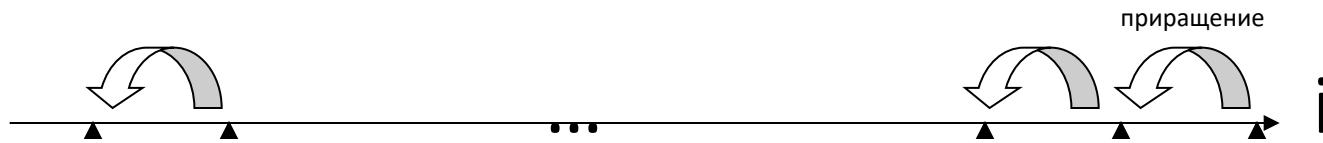
- Цикл от... до

при $\langle \text{приращение} \rangle > 0$



$\langle \text{Н.з.} \rangle$

при $\langle \text{приращение} \rangle < 0$



$\langle \text{К.з.} \rangle$

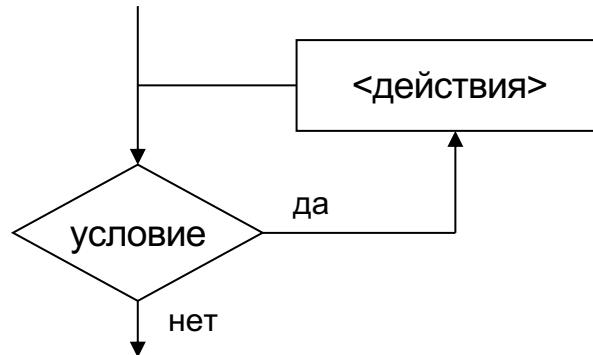
$\langle \text{К.з.} \rangle$

$\langle \text{Н.з.} \rangle$

Циклы

- Цикл-пока (с предусловием)

Блок-схема



Псевдокод

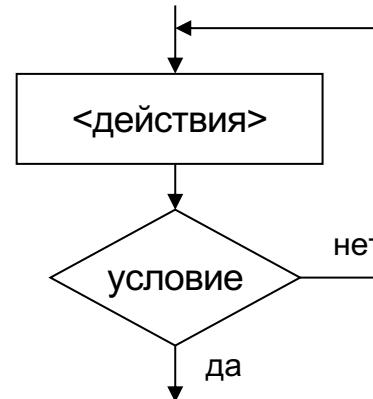
цикл-пока <условие>
 <действия>

кц

Выполнение цикла продолжается пока условие истинно и завершается, когда условие станет ложным.

- Цикл-до (с постусловием)

Блок-схема



Псевдокод

цикл

 <действия>

до <условие>

кц

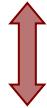
Выполнение цикла продолжается пока условие ложно и завершается, когда условие станет истинным.

Циклы с предусловием и с постусловием являются взаимозаменяемыми

цикл-пока <условие>

<действия>

кц



если <условие> то

цикл

<действия>

до **не** <условие>

кц

всё

цикл

<действия>

до <условие>

кц



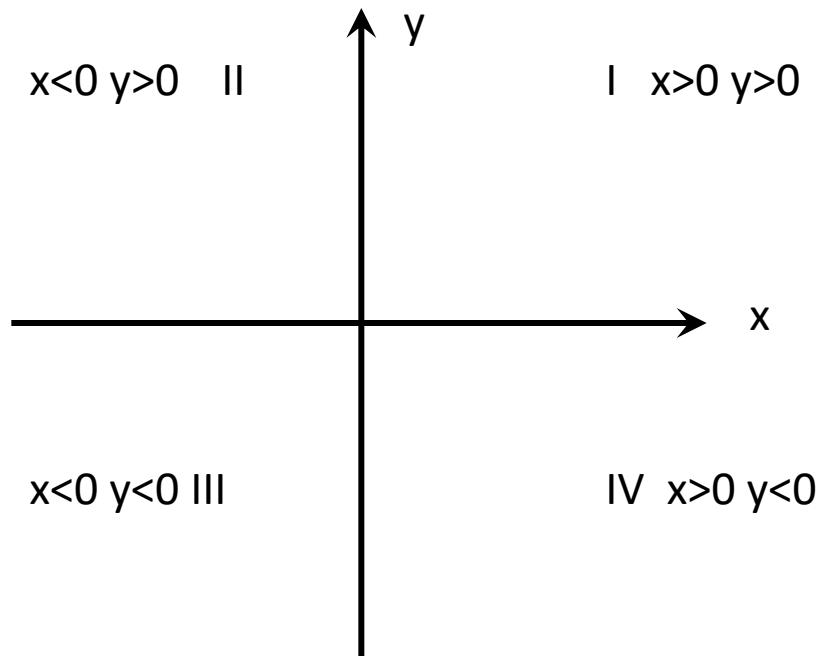
<действия>

цикл-пока **не** <условие>

<действия>

кц

Пример. Вычислить номер четверти, которой принадлежит точка с заданными координатами x , y . Если точка расположена на осях координат, то вывести соответствующее сообщение.



Постановка задачи

Дано: x, y - вещ

Результат: n или сообщение «точка на оси»

При : -

Связь: $n=0$ если $x=0$ или $y=0$

$n=1$ если $x>0$ и $y>0$

$n=2$ если $x<0$ и $y>0$

$n=3$ если $x<0$ и $y<0$

$n=4$ если $x>0$ и $y<0$

Алг "номер четверти"

нач

вывод ("номер четверти")

вывод ("введите координаты
точки")

ввод(x,y)

n:=0

если x>0 и y>0 то

n:=1

иначе если x<0 и y>0 то

n:=2

иначе если x<0 и y<0 то

n:=3

иначе если x>0 и

y<0 то

n:=4

всё

всё

всё

всё

если n=0 то

вывод("точка на оси")

иначе

вывод("номер четверти",n)

всё

кон

Язык программирования С

Язык Си был создан в 1972г. Деннисом Ритчи как инструментальное средство для написания операционной системы UNIX. Прообразом для Си послужил язык Би. Си предназначен для профессиональных программистов.

Си = язык + библиотека функций

Достоинства Си:

1. Современность. Его структура побуждает программиста использовать исходящее проектирование, структурное программирование, пошаговую разработку модулей. Результатом такого подхода является надежная и читаемая программа.
2. Эффективность. Структура языка позволяет использовать все возможности компьютера.
3. Гибкость и мощность. На Си написано большинство операционных систем. Программы, написанные на Си, используются для решения физических и технических проблем и даже для производства мультиплексионных фильмов.
4. Мобильность. Си - программы идут на разных ЭВМ.
5. Компактность и быстрота исполнения программ.
6. Удобство. Он достаточно структурирован, чтобы поддерживать хороший стиль программирования, и вместе с тем не связывать вас с ограничениями.

Недостаток Си: Повышенная ответственность программиста. Легко допустить трудно обнаруживаемую ошибку.

Структура языка программирования

Основные элементы языка

- **алфавит** (прописные и строчные буквы различаются (TIME и time - разные идентификаторы);
- **идентификаторы** (содержат символы подчеркивания (_), 0-9, a-z, A-Z, начинаются с буквы)(31 первых символов значащие по стандарту ANSI);
- **ключевые слова** (записываются строчными буквами);
- **операции**;
- **выражения** (Си позволяет выполнять явное преобразование типов и имеет возможность их смешивания):
- **операторы:**
 - присваивания;
 - описания управляющих структур;
 - ввода-вывода (это набор функций);
- **структуры данных;**
- **объявление типов данных** определяет область возможных значений, форму представления, точность, размер;
- **способ выделения памяти** (статический либо динамический);
- **подпрограммы** (имеется возможность рекурсии);
- **библиотеки** стандартных функций.

Форматный ввод/вывод

Для использования форматного ввода-вывода необходимо подключить библиотеку (заголовочный файл)

#include <stdio.h>

Форматный вывод

printf (<управляющая строка>, <арг. 1>, <арг. 2>, ...);

где

<арг. 1>, <арг. 2>. ... - выводимые параметры (переменные, константы, выражения)

<управляющая строка> - строка символов, показывающая, как должны быть напечатаны параметры. Управляющая строка - фраза в кавычках, содержащая спецификации для вывода:

- %d - десятичного целого числа;
- %c - одного символа;
- %s - строки символов;
- %f - числа с плавающей точкой .

Спецификация формата

При использовании формата можно указать ширину полей.
Для типа float можно указать общую ширину поля вывода и
число символов после точки.

```
int x,y;
```

```
float z;
```

```
x=0; y=125; z=3.14;
```

```
printf("x=%d;\ny=%5d\nz=%8.3f\n",x,y,z);
```

```
x=0;
```

```
y = _ _ 125
```

```
z=_ _ _ 3.140
```

Форматный ввод

Для функции `scanf()` указывается управляющая строка и список аргументов. Аргументы этой функции - **указатели** на переменные.

Два правила:

1. Для ввода некоторого значения перед именем переменной требуется написать символ & (т.е. указать адрес соответствующей области).
2. Для ввода строковой переменной использовать символ & не нужно.

```
#include <stdio.h>
```

```
int main()
```

```
{int age;
```

```
    printf ("How old are you?\n");
```

```
    scanf ("%d", &age);
```

```
    printf ("Wonderful! You are %d years old\n ", age);
```

```
    return 0;
```

```
}
```

Функция `scanf()` при чтении вводимой строки помещает в конец нуль-символ '\0'. Эта функция осуществляет ввод символов до тех пор, пока не встретит разделитель (пробел, символ табуляции или перевода строки).

Структура простой программы на Си

- В каждой программе обязательно присутствует главная функция (с именем `main()`), которая выполняется первой.
- Каждый оператор заканчивается «`;`», которая является частью оператора, а не разделителем, как в Паскале.
- После `#include`, `#define`, `main() ";"` не ставится.

<Подключение библиотек>

```
int main() //заголовок  
{<операторы объявления  
переменных>}
```

**<операторы присваивания,
ввода/вывода, управляющие
структуры >**

```
return 0;  
}
```

Типы данных

Различают простые и составные типы.

Простые:

- символьный (1 байт);
- целый (2 байта или 4 байта);
- вещественный (4 байта или 8 байтов);
- указатели.

Составные:

- структуры (записи);
- объединения.

Каждая простая переменная должна быть отнесена к одному из основных типов. Это делается с помощью **объявления переменных**. Цель объявления - определить:

- **форму представления** (с фиксированной или плавающей точкой);
- **точность** (число знаков);
- **способ выделения памяти** (статический, автоматический (до исполнения, в процессе исполнения));
- **размер памяти** и наличие или отсутствие знакового разряда (для типов с плавающей точкой – число разрядов для мантиссы и порядка).

Пример описания простых переменных

- int i, j, k;
 - long i1, abc;
 - short x, var;
 - unsigned Z, Z1;
 - float r1,r2;
 - char bukva;
- целые

Замечания:

- Всегда выдерживается соотношение размеров $\text{short} \leq \text{int} \leq \text{long}$.
- Данные типа **char** рассматриваются как **целые**.
- Использование **целых без знака** (для счетчиков) увеличивает максимальное целое, например, при 16 битах - до 65535 (вместо 32767).
- В Си отсутствует булевский тип (в C++ есть **bool**).

Определение размеров памяти для хранения данных

```
#include <stdio.h>
int main()
{
    printf("char ≡ %d bits\n", 8 *
        sizeof(char));
    printf("unsigned char ≡ %d
        bits\n", 8 * sizeof(unsigned
        char));
    //далее -аналогично
    return 0;
}
```

char ≡ 8 bits
unsigned char ≡ 8 bits
int ≡ 32 bits
unsigned int ≡ 32 bits
long int ≡ 32 bits
short int ≡ 16 bits
double ≡ 64 bits
float ≡ 32 bits
long double ≡ 128 bits

Операция **sizeof** возвращает число байтов, занимаемых операндом (константой, типом или переменной).

Константы

1. Целые: 243 -5, первой цифрой не должен быть 0.
 - Длинное целое: 243123L 12I = 12 (десятичное) (максимальное значение для 4 байт ≈ 2 млрд)
 - Восьмеричное: 0243 012341 (если целое начинается с нуля, оно интерпретируется как восьмеричное число) 020=16; 012=10 .
 - Шестнадцатеричное: 0X243 0x243L (начинающееся с символов 0x или 0X) 0X20=32.
2. Вещественные: 2.73 2E-5 .31E5 300. . В Паскале, если есть точка, то должны быть цифры слева и справа от нее. В Си нет такого ограничения.

В Си все вещественные константы представляются числом с плавающей точкой двойной точности, 8 байт, т.е. имеющим тип double.

В используемой версии компилятора для отделения дробной части числа от целой при вводе данных используется запятая, а в остальных случаях **точка**.

3. Символьные: 'A' '.' - один символ в апострофах (представляются как данные типа int, т.е. хранится код символа).

Специальные управляющие символьные константы

'\n' – новая строка

'\t' – табуляция

'\b' – возврат на шаг назад

'\r' – возврат каретки

'\\' – обратный слеш

'\'' – апостроф '

'\"' – кавычки "

'\0' – нуль-символ

4. Строковые

Строковая константа - последовательность символов, заключенная в кавычки. Например, “Это строка”.

В языке Си **нет** специального типа для описания строк.

Строки представляются в виде массива элементов типа **char**. Последним элементом массива является нуль-символ '\0', который компилятор автоматически помещает в конец строки. Нуль-символ не выводится на печать и в таблице кодов ASCII имеет номер **0**.

0	1	2	3	4	5	6	7	8	9	10
Э	т	о		с	т	р	о	к	а	\0

Описания с начальным присваиванием

Язык Си позволяет определить начальное значение переменной при ее описании.

```
int i=25;
```

```
float a=2.5, c=1E-7;
```

```
char C='c', lett='z';
```

Операции в Си

Операции в Си по количеству operandов разделяются на

- унарные (1);
- бинарные (2);
- тернарные (3).

Число operandов указано в скобках.

Унарные операции имеют высший приоритет и выполняются справа налево, бинарные – слева направо.

Унарные

-	арифметическое отрицание	-2.4
!	логическое отрицание (not) !(x>0)	
*	разадресация	*a
&	взятие адреса	&x
+	унарный плюс	+2
sizeof	размер памяти	sizeof(int)

Бинарные (в порядке убывания приоритета)

Арифметические

*	умножение	a *b
/	деление	x /10
%	(остаток от деления целых чисел)	z % 2
+	сложение	x + 2
-	вычитание	3 – 5

Операции отношения

>	больше	$z > 0$
<	меньше	$x < y$
\geq	больше или равно	$4 \geq 8$
\leq	меньше или равно	$0 \leq x$
$= =$	равно (сравнение)	$x == 5$
$!=$	не равно	$s != 0$

Логические (имеют более низкий приоритет, чем операции отношения)

$\&\&$ - логическое "и" (and) $a != 0 \&\& d > 0$

$\| \|$ - логическое "или" (or) $x == 1 \| \| x > y$

Примечание. В отличие от Паскаля, в выражении

$x - a > b \&\& b > c \| \| b > 10$

скобки не нужны.

Операции присваивания

= это присваивание. (Соответствует := в Паскале).

Операции увеличения (инкремент) и уменьшения (декремент) на единицу

- $v++$; – увеличение v на 1 **после** использования v (постфиксная форма),
 - $++v$; – увеличение v на 1 **перед** использованием v (префиксная форма),
 - $v--$; $--v$; – уменьшение на 1,
- где v – переменная (обычно целого типа).

Пример. Префиксная и постфиксная форма инкремента и декремента.

```
#include <stdio.h>
int main()
{int a=1, b=1, aplus, plusb;

aplus=a++; //aplus=a; a++;
//увеличивается после присваивания
plusb=++b; //++b; plusb=b;
//увеличивается до присваивания
printf("a=%d, aplus=%d\n", a, aplus);
printf("b=%d, plusb=%d\n", b, plusb);
return 0;
}
```

a=2, aplus=1
b=2, plusb=2

a	aplus
1	1
b	plusb
1	2

Операции с присваиванием [*=, /=, +=, -=, %=]

a+=1; соответствует **a=a+1;**

Множественные присваивания выполняются справа налево.

x=y=++z; соответствует **{z=z+1;y=z;x=y;}**

Операция «запятая» позволяет вместо составного оператора, содержащего только присваивания, использовать простой. Вычисления выполняются слева направо.

x=y, x++, z-=x; соответствует **{x=y; x++; z=z-x; }**

Наглядность программы снижается (но код укорачивается).

Не рекомендуется применять операции увеличения или уменьшения к переменной, которая входит в выражение более одного раза.

Операция приведения

Для приведения выражения к определённому типу используется операция приведения:

(тип) выражение

Она имеет наивысший приоритет, как все унарные операции.

```
int mice;
```

```
    mice=1.6+1.7;      //результат mice=3
```

```
    mice=(int) 1.6+(int) 1.7; //результат mice=2
```

Во втором операторе 1.6 и 1.7 преобразуются в целые числа 1, mice=1+1=2.

Математические функции

Для использования этих функций подключается математическая библиотека:

```
#include <math.h>
```

```
double x, y;      int n; //объявленные переменные
```

- $\sqrt{x} = \text{sqrt}(x)$
 - $x^y = \text{pow}(x, y)$
 - $|x| = \text{fabs}(x)$
 - $|n| = \text{abs}(n)$
 - $e^x = \text{exp}(x)$
 - $\ln x = \text{log}(x)$
 - $\lg x = \text{log10}(x)$
 - $\arctg \frac{x}{y} = \text{atan}(x/y) \quad (-\pi \quad +\pi)$
 - $\sin x = \text{sin}(x)$
 - $\cos x = \text{cos}(x)$
 - $\tg x = \text{tan}(x)$
 - $\arcsin x = \text{asin}(x)$
 - $\arccos x = \text{acos}(x)$
 - $\operatorname{arctgx} = \text{atan}(x)$
- (значение в интервале $-\pi/2; \pi/2$)

Например, рассмотрим кодирование формулы

$$z = \sqrt[3]{\ln|x - y^2|} + \frac{1}{\lg 7 + \tan 3};$$

На языке Си:

```
z=pow(log(fabs(x-pow(y,2.))), (1./3.))+  
1./(log10(7.)+tan(3.));
```

Простые и составные операторы

- Если ветвь условной конструкции или тело цикла содержит несколько операторов, то они объединяются в составной оператор при помощи “{” и “}”.
- Begin в Паскале соответствует “{” в Си, End - “}”.

Пример. Нахождение суммы первых 20 натуральных чисел и вывод сумм на экран.

```
#include <stdio.h>

int main()
{int sum=0; //сумма
int c=0; //счетчик

while (c++<20) //увеличение после сравнения
{
    sum=sum+c; // или sum+=c
    printf ("sum=%d\n", sum);
}
return 0;
}
```

Условная конструкция.

Различают три формы условной конструкции
(пример конструкции в Паскале):

1. простейшая (`if ... then...`);
2. расширенная (`if ... then ... else ...`);
3. выбор (`case ... of...`).

Простейшая условная конструкция.

Псевдокод :

если <условие> то
 <оператор>

всё

Си

if (<выражение>)
 <оператор>

В отличие от Паскаля выражение может иметь любой тип.

Важно: 0, '\0' и NULL считаются ложью, остальные значения - истиной.

Расширенная условная конструкция

Псевдокод :

если <условие> то

 <оператор1>

иначе

 <оператор2>

всё

Си

if(<выражение>)

 <оператор1>

else <оператор2>

Примеры.

```
if(a>b&&b>c)
```

```
f=x*x-1;
```

```
else
```

```
f=x+1;
```

‘;’ перед **else** является
частью оператора
присваивания в ветви
if.

```
if(x>5 || x<0)
```

```
{
```

```
k++;
```

```
y[k]=x;
```

```
}
```

```
else
```

```
{
```

```
n++;
```

```
z[n]=x;
```

```
}
```

Отличия от Паскаля:

1. Выражение обязательно заключается в скобки.
2. ";" может стоять перед **else**, если только нет составного оператора. (";" - часть <оператора1>).
3. Другой приоритет вычисления логического выражения:
 - сначала операции отношения;
 - затем логические операции !, &&, || (не, и, или).

Приоритет операций отношения меньше, чем у операций "+" и "-" и больше, чем у операции присваивания:
 $x > y + 2$ то же, что и $x > (y + 2)$.

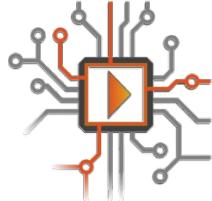


Допускается вложенность операторов `if`. Если нет составного оператора, `else` относится к ближайшему `if`.

Рассмотрим пример.

```
if (number>6)
if (number<12)
    printf ("Конец игры!\n");
else
    printf ("Потеря хода!");
```

Число	Результат
5	«Нет результата»
10	Конец игры
15	Потеря хода



Если необходимо, чтобы `else` соответствовал первому `if`, добавим `{ }`.

Рассмотрим пример.

```
if (number>6)
{
    if (number<12)
        printf ("Конец игры!\n");
}
else
    printf ("Потеря хода!");
```

Число	Результат
5	Потеря хода
10	Конец игры
15	«Нет результата»

Рассмотрим программу решения квадратного уравнения $ax^2+bx+c=0$.

```
#include <stdio.h>
#include <math.h>
int main()
{
float a,b,c,d,x1,x2;

printf("solve ax^2+bx+c=0\n");
printf("input a,b,c\n");
scanf("%f%f%f",&a,&b,&c);
if (a==0.0) printf("linear !\n");
else
{
d=b*b-4.0*a*c;
printf("\nd=%8.3f\n",d);
if (d<0) printf("no real solutions !\n");
else
{
    if (d==0.0)
        { x1=-b/(2.0*a);
        printf("one !\n x=%8.3f\n",x1);
        }
    else
        {
        x1=(-b+sqrt(d))/(2.0*a);
        x2=(-b-sqrt(d))/(2.0*a);
        printf("two !\n x1=%8.3f",x1);
        printf("\n x2=%8.3f\n",x2);
        }
    }
return 0;
}
```

Операция условия - сокращённый способ записи if-else (тернарная).

В общем виде условное выражение записывается следующим образом:

(<условие>) ? <значение если истинно> : <значение если ложно>;

Например:

x= (y<0) ? -y : y;

эквивалентно:

if (y<0)

 x = -y;

else

 x = y;

Условное выражение удобно использовать в тех случаях, когда некоторой переменной надо присвоить одно из двух возможных значений, например:

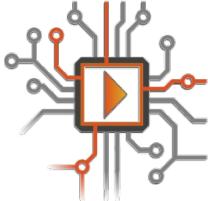
max = (a > b)?a:b;

Множественный выбор: switch и break

```
switch (<выражение>) /*выражение  
может быть типа int или char*/  
{  
    case <константа 1>: <операторы 1>  
        //операторы могут отсутствовать  
    case <константа 2>: <операторы 2>  
        //константы типа int или char  
        ...  
    default: <операторы>  
        //ветвь не обязательна  
}
```

Если значение выражения совпадает с одной из констант, выполняются операторы, расположенные после соответствующей константы.

Если подходящей метки не найдется, то, если существует строка с меткой "default", будет выполняться оператор, помеченный этой меткой. В противном случае произойдет переход к оператору, расположенному за оператором switch.



В каждой последовательности операторов последним должен быть оператор **break**.

Выполнение оператора **break** осуществляет выход из оператора **switch** и переход к следующему за ним оператору.

При отсутствии оператора **break** будут выполнены все операторы, начиная с помеченного данной меткой и заканчивая оператором **default**.

```
char ch='1';
switch (ch)
{
    case '1': printf ("один \n");
    case '2': printf ("два \n");
    default : printf ("три \n");
}
```



В качестве меток-констант используются выражения типа **int** или **char**.

В качестве метки запрещается использовать переменную.

Можно пометить оператор несколькими метками одновременно. Например,

```
case 'E':  
case 'e':    printf ("ель\n"); break;
```

Пример. Игра в города

```
#include <iostream>
#include <stdio.h>
int main() { char ch; setlocale(LC_ALL, "RUS");
printf ("Введите букву а, б или в. \n");
printf ("Я назову город на эту букву\n");
scanf("%c", &ch);
switch (ch) {
    case 'а': printf ("\n Ашхабад"); break;
    case 'б': printf ("\n Белгород"); break;
    case 'в': printf ("\n Воронеж"); break;
    default: printf ("\n неизвестная буква");
}
return 0;
}
```

Правила выбора условных операторов

1. Выбор из двух возможностей - выполнить оператор или пропустить его - оператор **if**.
2. Выбор одного из двух вариантов - **if...else**.
3. Выбор одного из нескольких - **else-if, switch**.

Если выбор вариантов основывается на вычислении значения переменной или выражения типа **float**, то **switch** применить **нельзя**.

Если значения переменной попадают в некоторый диапазон, использовать **switch** неудобно. Лучше записать:

```
if (i <1000 && i>2).
```

Циклы

Существует три базовых структуры цикла ([пример в Паскале](#)):

- цикл-пока (с предусловием) (**while ... do...**);
- цикл-до (с постусловием) (**repeat ...until ...**);
- цикл от-до (со счетчиком) (**for ... to/downto do ...**).

Цикл-пока (while)

Псевдокод

цикл-пока <условие>
 <действия>

кц

Си

while (<выражение>) <оператор>

, где

<выражение> - любого типа.

Пока значение выражения отлично от 0 (т.е.истинно), повторяется выполнение оператора в теле цикла.

Оператор может быть простым или составным.

while является циклом с предусловием, поэтому возможно, что он не выполнится ни разу.

Пример 1. Напечатать целые числа от 1 до 100 и их квадраты.

```
/*Здесь и в следующих примерах пропущены операторы
#include <iostream>
#include <stdio.h>
```

Русская локаль считается установленной, т.е. в начале главной функции
есть строка

```
setlocale(LC_ALL,"RUS");
*/
```

```
int main()
{
int n=0;

printf("число квадрат");
while(n++<100)
    printf("%6d %6d\n",n,n*n);
return 0;
}
```

Пример 2. Сколько членов гармонического ряда $S=1+1/2+1/3+\dots+1/n$ надо взять, чтобы получить сумму, большую числа **dano**?

```
int main()
{
int i=0;
float dano, S=0.0;

printf ("введите число"); scanf("%f",&dano);
while (dano<=0.0)
{
    printf("повторите ввод");
    scanf ("%f", &dano);
}
while (S<=dano)
    S+=1.0/(float)(++i);
printf ("число членов ряда=%d\n",i);
return 0;
}
```

Цикл for

Псевдокод

цикл от i:=<н.з.> до <к.з.> [шаг <приращение>]
<действия>

КЦ

Си

for (<выражение 1>; <выражение 2>; <выражение 3>)
<оператор>

, где

- <выражение 1> – инициализирующее: вычисляется один раз до начала цикла;
- <выражение 2> – проверяющее: вычисляется перед каждым выполнением оператора. Тело цикла выполняется, если значение проверяемого выражения истина (или не равно нулю);
- <выражение 3> – корректирующее: вычисляется после каждого выполнения тела цикла.

Перед первым выполнением цикла проверяется <выражение 2>, т.е. тело цикла может не выполниться ни разу!

Любое выражение, а также <оператор> может отсутствовать, но точка с запятой сохраняется.

Оператор **for**

```
for(<выражение 1>; <выражение 2>; <выражение 3>)  
    <оператор>
```

эквивалентен следующей последовательности операторов:

```
<выражение 1>  
while(<выражение 2>)  
{  
    <оператор>  
    <выражение 3>  
}
```

Пример 1. Найти сумму n членов гармонического ряда $S=1+1/2+1/3+\dots+1/n$.

a) $S=0.0;$

```
for(i=1;i<=n;i++)
```

```
S=S+1.0/(float)i;
```

b) $\text{for } (i=1, S=0.0; i<=n; i++)$

```
S+=1.0/(float)i;
```

Операция «запятая» связывает два выражения в одно и гарантирует, что самое левое будет вычисляться первым. Обычно используется для включения дополнительной информации в спецификацию цикла **for**, например:

<выражение 1>, < выражение 2>

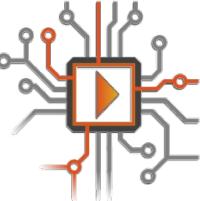
c) `for (i=1, S=0.0; i<=n; S+=1.0/(float)i++);`
`//Здесь пустой цикл.`

Это не очень хороший стиль программирования - лучше не смешивать процесс изменения переменной цикла с алгебраическими вычислениями.

Пример 2. Найти минимальное n , при котором $S=1+1/2+1/3+\dots+1/n > \text{dano}$.

`for (n=0, S=0.0; S<=dano; S+=1.0/++n);`

В операторе **for** проверяемое выражение не обязательно использует параметр цикла.



Пример 3. Счет в порядке убывания

```
for(n=10;n>0;n--)  
    printf ("%d секунд !\n", n);  
printf("пуск!\n");
```

Можно использовать любое значение шага цикла, например:

```
for(n=2; n<60;n+=13)  
    printf("%d\n",n);
```

На экране получим 2 15 28 41 54.

Пример 4. Переменная цикла может быть не только числом, но и символом.

```
for(ch='a';ch<='z';ch++)  
printf ("Величина кода для %c = %d\n", ch, ch);
```

При выполнении этого оператора будут выведены на печать все буквы от **a** до **z** и их коды ASCII.

Пример 5. Можно пропустить одно или более выражений, но при этом нельзя пропустить символ “;”.

1. ans=2; for (n=3; ans<=25;) ans=ans*n;
2. for(); printf ("работаю\n");

Тело этого цикла будет выполнять бесконечное число раз, поскольку пустое условие всегда считается истинным.

Следующие записи эквивалентны:

while (<выражение>) <оператор>

и

for (; <выражение> ;) <оператор>

Применение операторов **for** или **while** - дело вкуса. Цикл **for** предпочтительнее, когда в цикле используется инициализация и коррекция переменной, в остальных случаях лучше использовать **while**.

Возможно изменение параметров, входящих в проверяющее и корректирующее выражения, в теле цикла.

Цикл с постусловием do...while

Псевдо~~код~~

цикл

<действия>

до <условие>

кц

Си

do

<оператор>

while(<выражение>);

Тело цикла всегда выполняется хотя бы один раз. Выполнение цикла продолжается до тех пор, пока выражение не станет ложным (равным 0). Т.е. выражение обозначает условие ПРОДОЛЖЕНИЯ цикла.

В Паскале - наоборот, цикл продолжается пока выражение не станет истинным.

Пример. Найти минимальное число членов гармонического ряда с $S > \text{dano}$

```
i = 0; S=0.0;  
do  
    S+=1.0/(float)++i;  
while (S<= dano);
```

При помощи этого цикла можно организовать ввод данных с проверкой их правильности, например.

```
int a, b, x;  
do  
{puts("input a,b (a<=b)");  
 x=scanf("%d%d", &a,&b);  
 while(getchar()!='\n'); //очистка входного потока  
 }while(x<2||a>b);
```

Пример использования цикла-до: метод половинного деления

Задача. Найти корень уравнения $f(x)=0$ на $[a, b]$ с точностью ϵ методом половинного деления (предполагается, что на этом отрезке существует только один корень).

Пример использования цикла-до: метод половинного деления

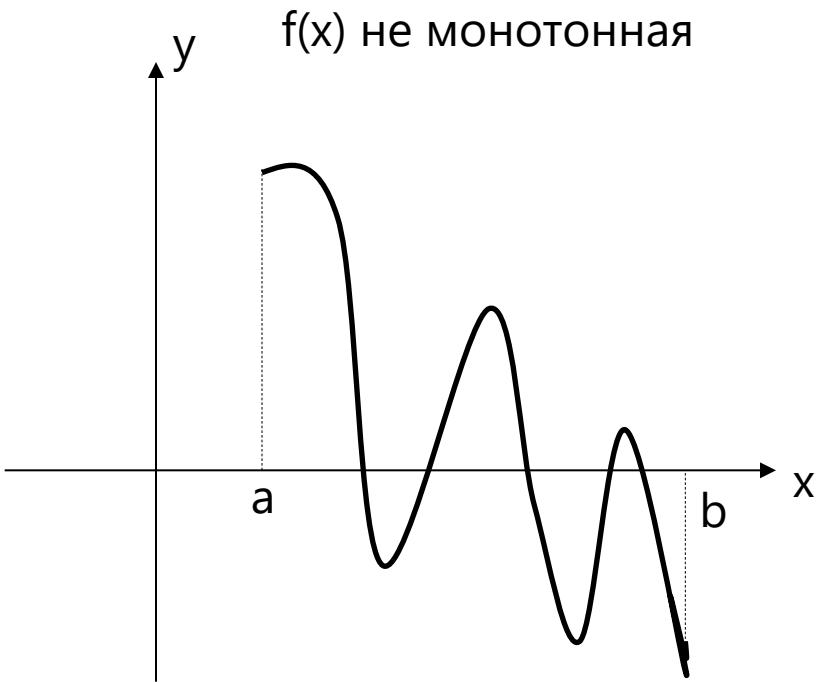
Задача. Найти корень уравнения $f(x)=0$ на $[a, b]$ с точностью ϵ методом половинного деления (предполагается, что на этом отрезке существует только один корень).

Для того, чтобы существовал только один корень, функция должна быть монотонна и непрерывна на этом отрезке. Знаки функции на концах отрезка должны быть различны.

Пример использования цикла-до: метод половинного деления

Задача. Найти корень уравнения $f(x)=0$ на $[a, b]$ с точностью ϵ методом половинного деления (предполагается, что на этом отрезке существует только один корень).

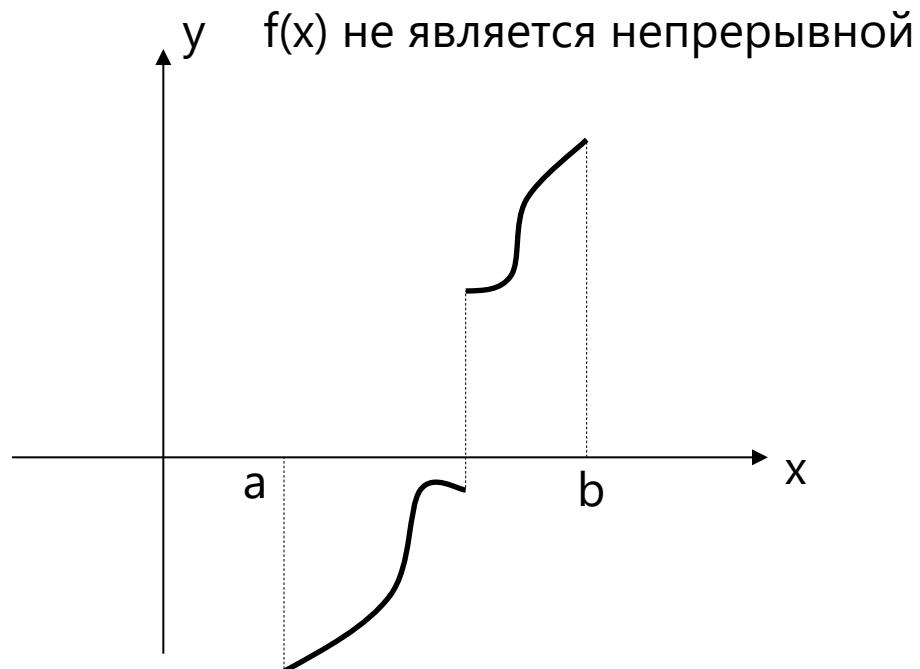
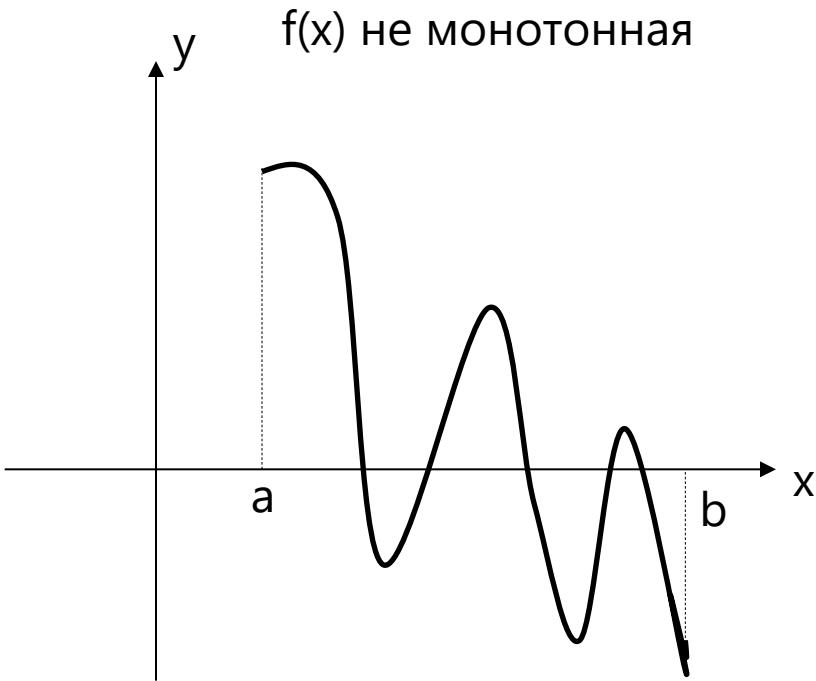
Для того, чтобы существовал только один корень, функция должна быть монотонна и непрерывна на этом отрезке. Знаки функции на концах отрезка должны быть различны.



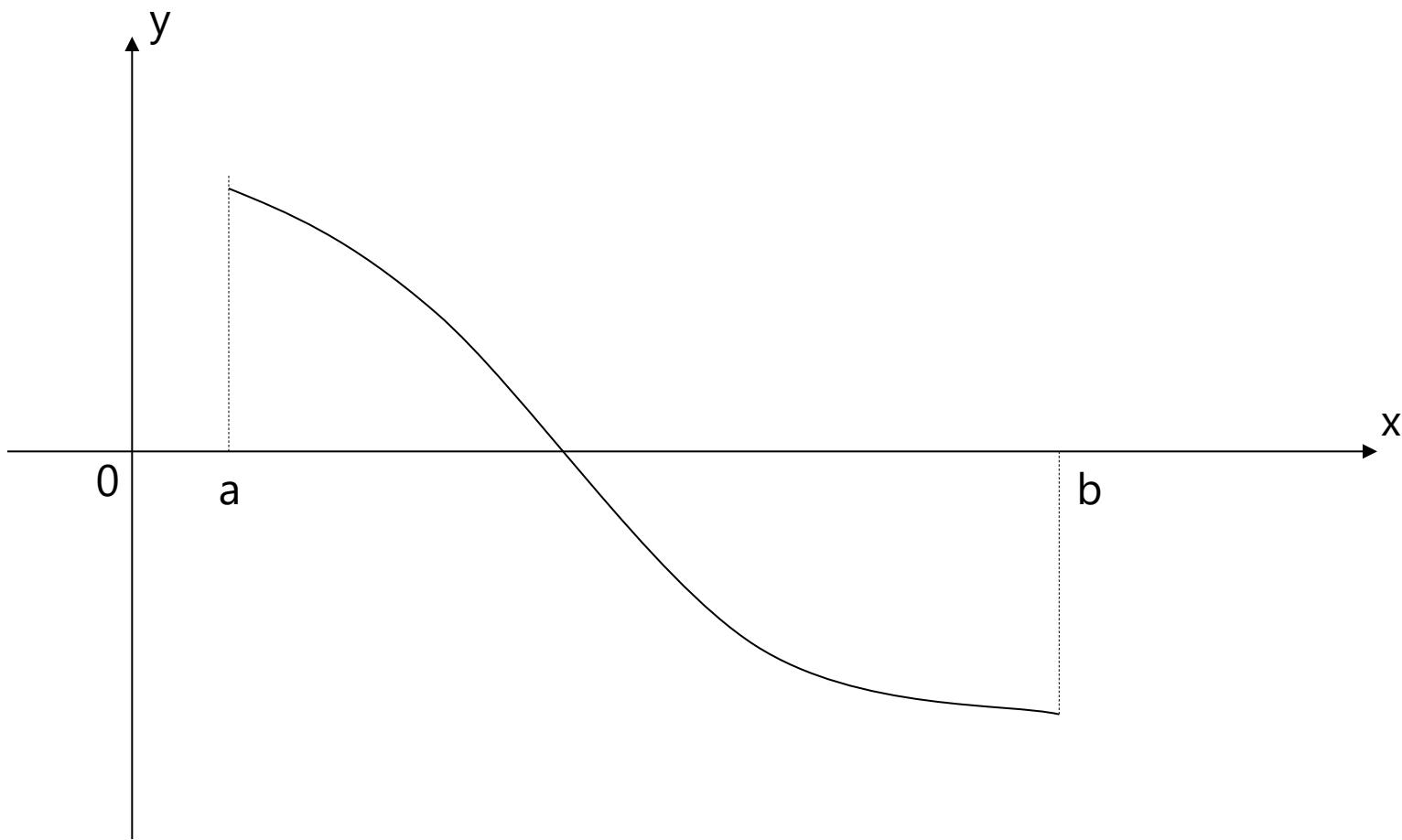
Пример использования цикла-до: метод половинного деления

Задача. Найти корень уравнения $f(x)=0$ на $[a, b]$ с точностью ε методом половинного деления (предполагается, что на этом отрезке существует только один корень).

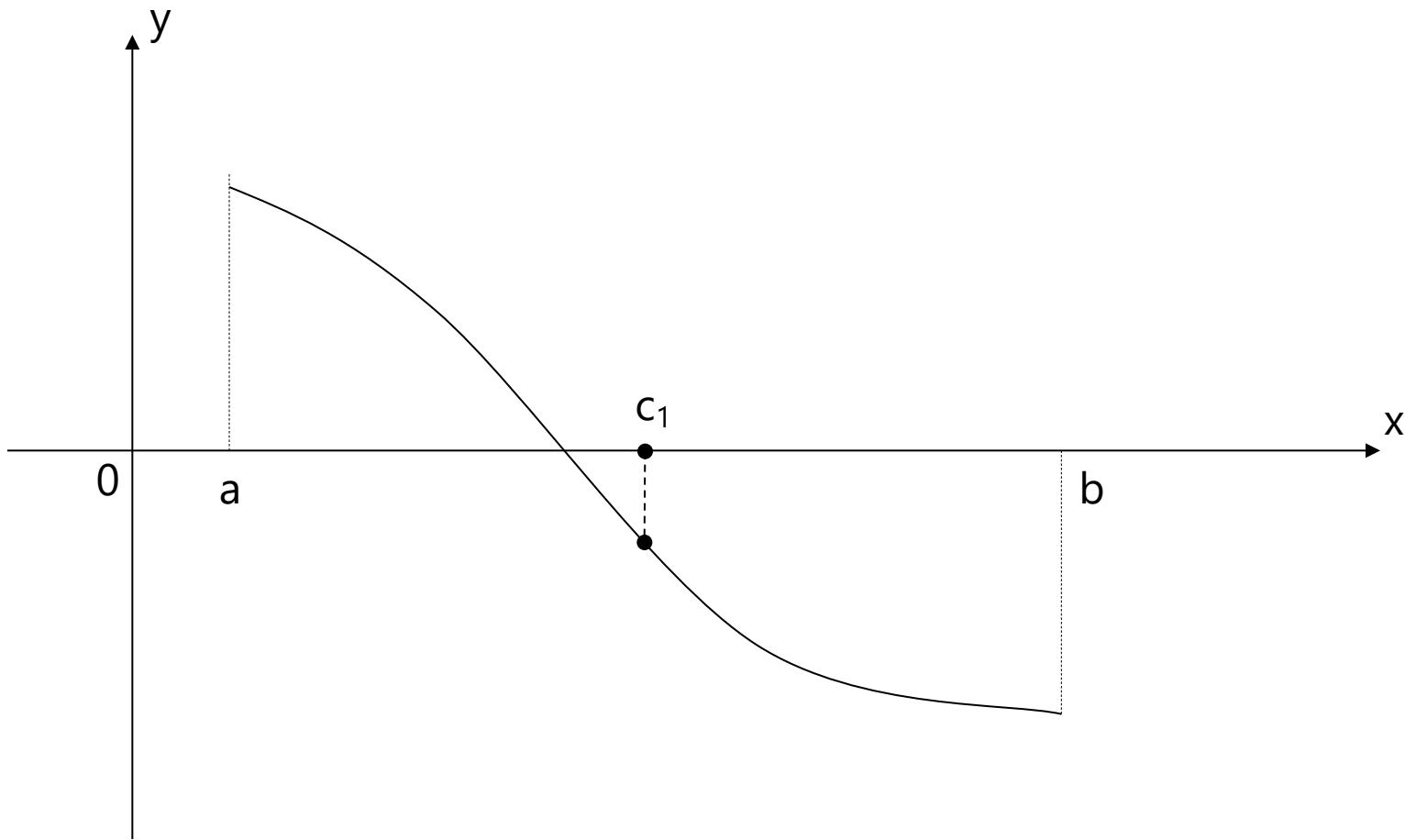
Для того, чтобы существовал только один корень, функция должна быть монотонна и непрерывна на этом отрезке. Знаки функции на концах отрезка должны быть различны.



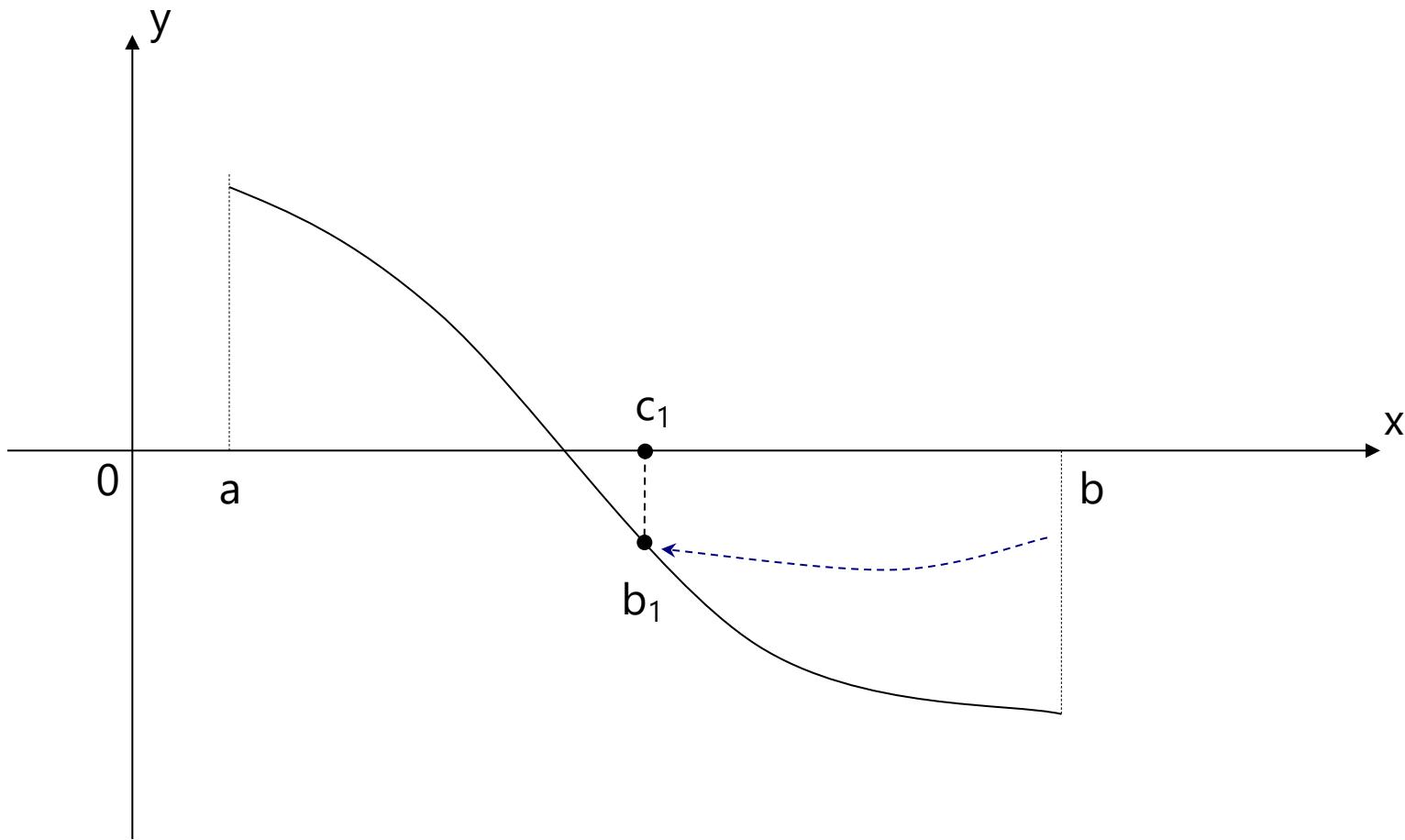
Описание работы метода



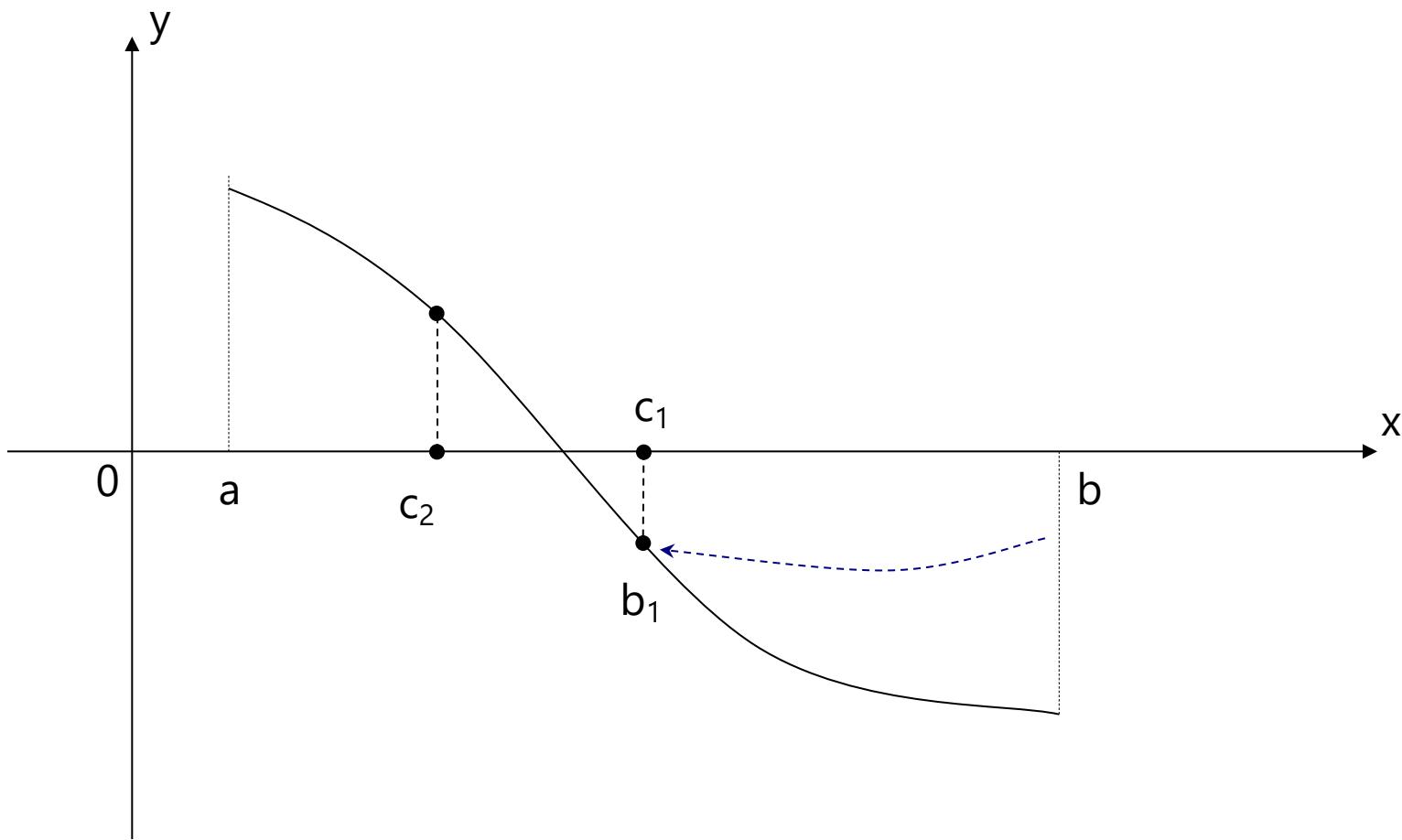
Описание работы метода



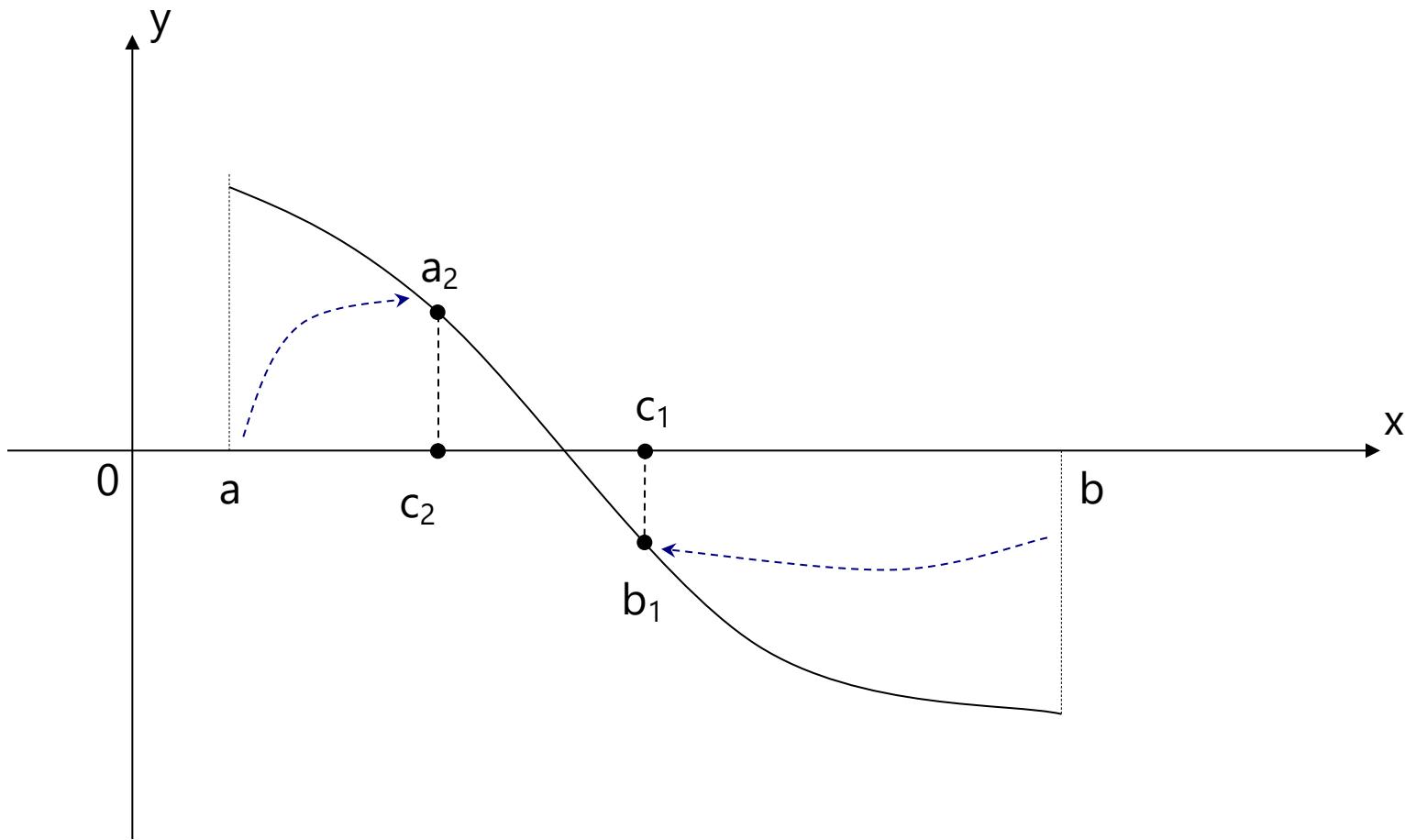
Описание работы метода



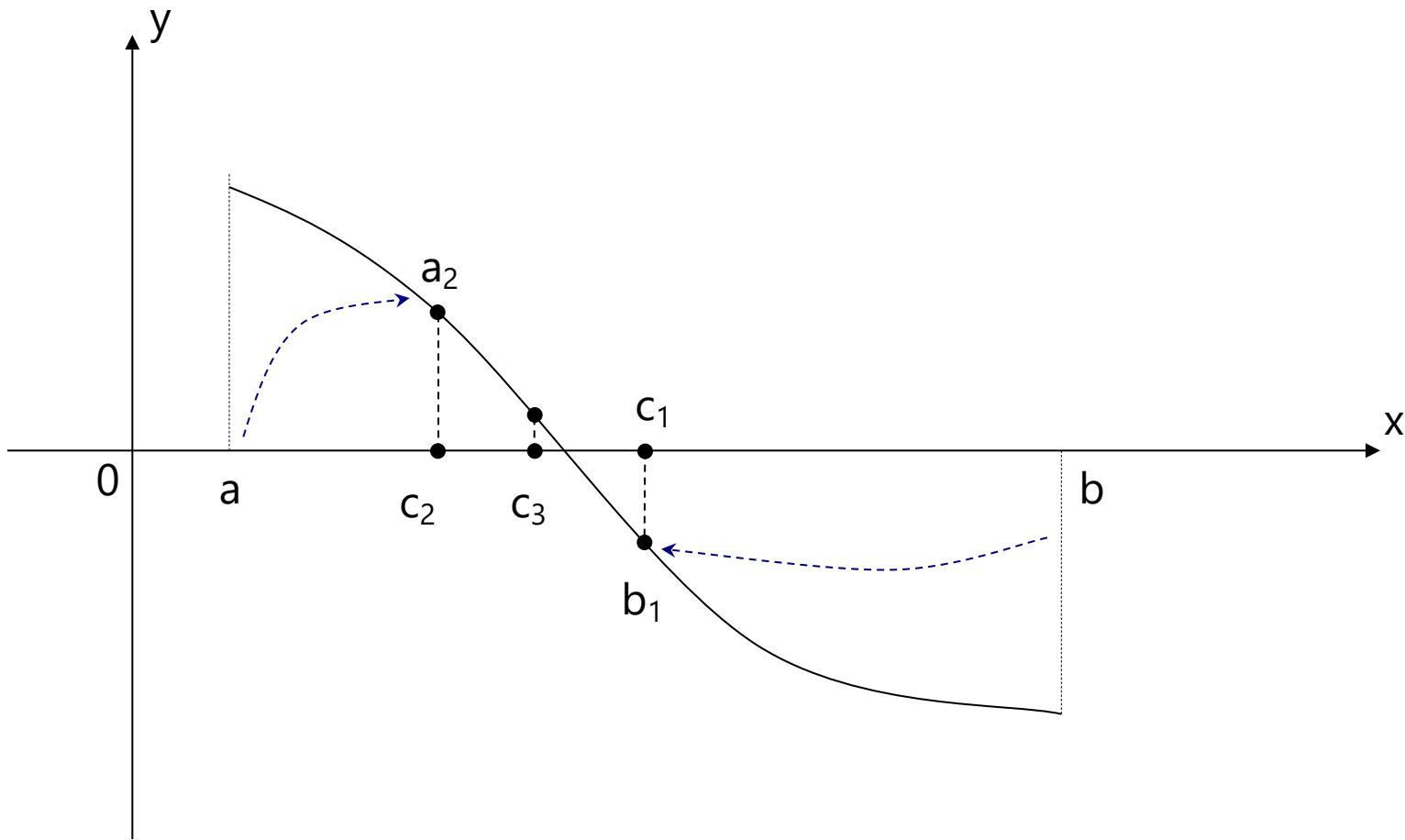
Описание работы метода



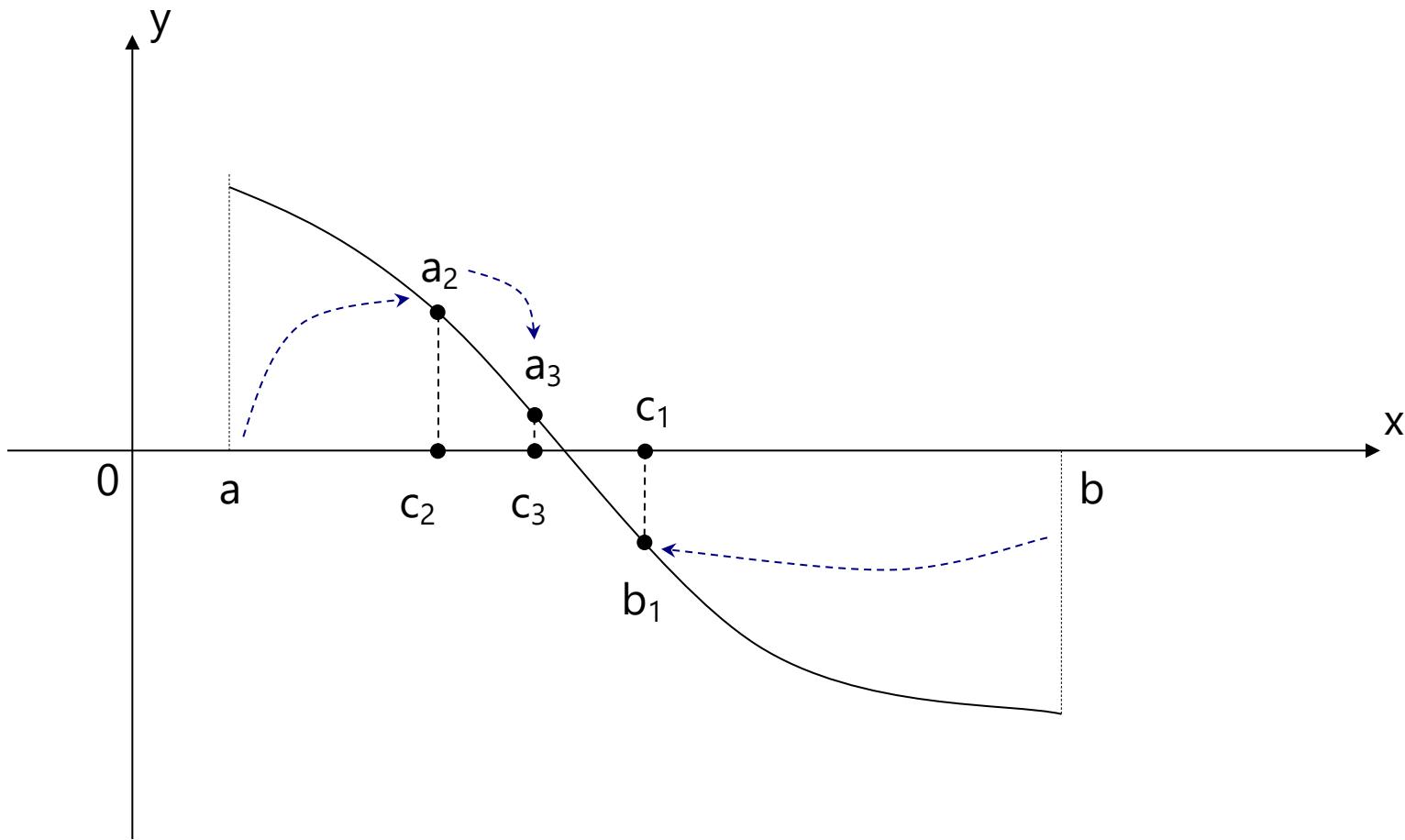
Описание работы метода



Описание работы метода



Описание работы метода



Постановка задачи:

Дано: a, b, eps - вещ.

Результат: x - вещ.

При: $a < b, \text{eps} > 0, f(a) * f(b) < 0.$

Связь: $f(x) = 0$ с точностью eps .

При $|f(x)| < \text{eps}$ считаем что $x=a$.

Алгоритм (общий случай):

Алг "Метод половинного деления"
нач
вывод("Метод половинного деления")
цикл
вывод("Границы отрезка [a,b] - ")
ввод(a, b)
если a >= b то
 вывод("Неверные данные : a >= b")
всё
если f(a) * f(b) >= 0 то
 вывод ("Нет корня или их несколько")
всё
до f(a)*f(b) < 0 и a < b кц
цикл
вывод ("Ведите точность eps:")
ввод (eps)
до eps > 0 кц

ЦИКЛ
c := (a+b) / 2
если f(a) * f(c) < 0 то
 b := c
иначе
 a := c
всё
до |f(a)| ≤ eps кц

вывод ("Корень x= ", a)
кон

В качестве примера рассмотрим функцию:
 $\cos(x) - x = 0$.

Программа

```
#include<stdio.h>
#include<math.h>
int main()
{float a,b,c,eps;
printf("Метод половинного
деления");
//ввод данных
do{
    printf(«Введите границы[a,b] - ");
    scanf("%f%f",&a, &b);
    if (a >= b)
        printf(«Неверные данные: a
>= b\n");
    if ((cos(a)-a)*(cos(b)-b) >= 0)
        printf («На этом отрезке нет
корня\n");
}while((cos(a)-a)*(cos(b)-b) >= 0 ||
a >= b);
```

```
do{
    printf(«Введите точность eps :");
    scanf("%f",&eps);}
while (eps <= 0);
//вычисления
do{
    c = (a+b) / 2.0;
    if ((cos(a)-a)*(cos(c)-c) < 0)
        b = c;
    else
        a = c;}
while (fabs(cos(a)-a)> eps);
//вывод результата
printf(«Корень x =%12.8f", a );
return 0;
}
```

Программа с проверкой правильности ввода

```
#include<stdio.h>
#include<math.h>
int main()
{float a,b,c,eps; int num;
printf("Метод половинного деления");
//ввод данных
do{
    printf("Введите границы[a,b] - ");
    num=scanf("%f%f",&a, &b);
    if (num!=2)
    { printf("Недостаточно данных");
        while(getchar()!='\n');}
    }
    if (a >= b)
        printf("Неверные данные: a >= b");
    if ((cos(a)-a)*(cos(b)-b) >= 0)
        printf ("На этом отрезке нет корня");
}while((cos(a)-a)*(cos(b)-b) >= 0 || a >= b);
```

```
do{
    printf("Введите точностьeps :");
    num=scanf("%f",&eps);
    if (!num)
    { printf("Недостаточно данных\n");
        while(getchar()!='\n'); }
    }
while (eps <= 0);
//вычисления
do{
    c = (a+b) / 2.0;
    if ((cos(a)-a)*(cos(c)-c) < 0)
        b = c;
    else
        a = c;}
while (fabs(cos(a)-a)> eps);
//вывод результата
printf("Корень x =%12.8f", a );
return 0;
```

Пример 6. Вложенные циклы.

Вычислить все совершенные числа, меньшие или равные заданному числу $smax$.

Совершенное число равно сумме своих делителей, исключая делитель, равный самому числу. Например,

$$6=1+2+3.$$

```
int main()
{int n, sum, del, smax;

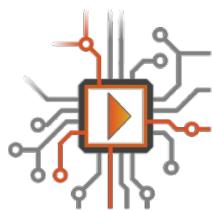
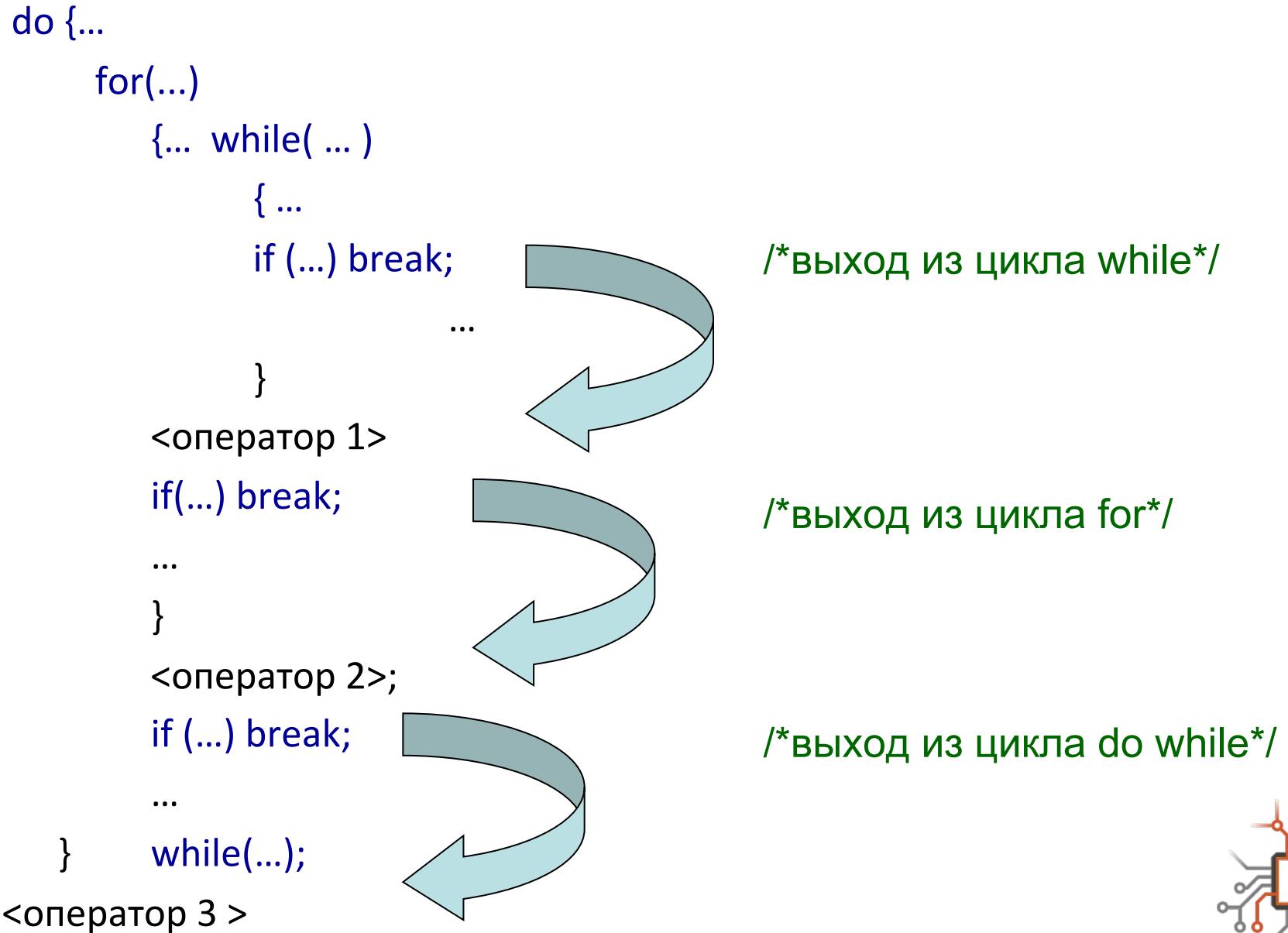
printf ("введите предел");
scanf ("%d", &smax);
printf ("совершенное число\n");
for(n=6; n<=smax; n++)
{
    sum= 1;
    for(del=2; del<n; del++)
        if(n%del==0) sum+=del;
    if (sum==n) printf("%5d",n);
}
return 0;
}
```

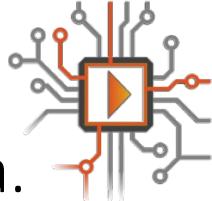
Управляющие операторы break, continue, goto

По возможности следует избегать их использования. Все эти операторы предназначены для безусловного перехода.

Частое и необоснованное применение этих операторов - признак низкой квалификации программиста и слабого владения структурным программированием.

break - выход из ближайшего цикла любого вида или switch и переход к следующему оператору программы.





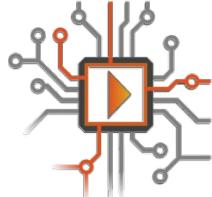
continue - окончание текущей итерации данного цикла.

В циклах **while** и **do ... while** происходит переход к проверке условия продолжения цикла.

В цикле **for** – переход к вычислению корректирующего выражения, а затем к проверке.

```
for (<выражение 1>;  
      <выражение2>; <выражение3>)  
{  
    while(...)  
    { ...  
        continue;  
        ...  
    }  
    ...  
    continue;  
    ...  
}
```

A diagram illustrating the control flow of a 'for' loop. A large blue arrow originates from the word 'continue' in the first iteration of the loop body and points upwards towards the start of the loop's body. Another smaller blue arrow originates from the word 'continue' inside the loop body and also points upwards towards its own start. This visualizes how the 'continue' statement exits the current iteration and loops back to the beginning of the body to start the next iteration.



Оператор goto

goto метка;

Метка должна быть идентификатором, например: **goto m1;**

Допускается использовать в одном случае - выход из вложенного набора циклов при обнаружении каких-либо ошибок.

```
for (...)  
    for (...)  
        for (...)  
        {  
            if (<ошибка>)  
                goto Code;  
            <операторы>  
        }
```

Code:<операторы для исправления ошибки>

Задача 1. Вычислить число чётных, нечётных, кратных 3-м и 4-м среди первых 1000 членов ряда Фибоначчи.

Ряд Фибоначчи: 1, 1, 2, 3, 5...

Каждый последующий член ряда равен сумме двух предыдущих.

$$\begin{cases} F(1) = F(2) = 1; \\ F(k) = F(k-1) + F(k-2), \quad k > 2. \end{cases}$$

Обозначим:

k_1 – количество нечётных;

k_2 – количество чётных;

k_3 – количество кратных 3;

k_4 – количество кратных 4

чисел Фибоначчи.

```
#include <iostream>
#include <stdio.h>
#define KOL 1000

int main()
{setlocale(LC_ALL, "RUS");
 int f1, f2, f3, k1, k2, k3, k4, i;

f1=f2=1;
k1=2;
k2=k3=k4=0;
for (i=3; i<=KOL; i++)
{
    f3=f1+f2, f1=f2,      f2=f3;
    if (f3%2==0)
        {k2++;
         if (f3%4==0)k4++;
        }
    else  k1++;
    if(f3%3==0) k3++;
}
printf("Среди первых %d членов
ряда Фибоначчи нашлось :\n",
KOL);
printf("Нечетных: %d\n", k1);
printf("Четных:    %d\n", k2);
printf("Кратных 3: %d\n", k3);
printf("Кратных 4: %d\n", k4);
return 0;
}
```

Также можно использовать оператор `continue`, который передаёт управление на начало следующей итерации ближайшего цикла:

```
if (!(f3%3)) k3++;//f3%3==0
if (f3%2)//нечетное
{
    k1++;
    continue;
}
k2++;
if (!(f3%4)) k4++;
```


Итерационные алгоритмы

Итерационные алгоритмы

- **Итерация** (лат. *iteratio* — повторение) в математике - одно из ряда повторений какой-либо математической операции, использующее результат предыдущей аналогичной операции.

Итерационные алгоритмы

- **Итерация** (лат. *iteratio* — повторение) в математике - одно из ряда повторений какой-либо математической операции, использующее результат предыдущей аналогичной операции.
- **Итерация** в программировании - это организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя (т.е. нет рекурсии).

Итерационные алгоритмы

- **Итерация** (лат. *iteratio* — повторение) в математике - одно из ряда повторений какой-либо математической операции, использующее результат предыдущей аналогичной операции.
- **Итерация** в программировании - это организация обработки данных, при которой действия повторяются многократно, не приводя при этом к вызовам самих себя (т.е. нет рекурсии).
- Один шаг цикла также называется **итерацией**.

Задача 1. Вычислить сумму членов ряда

$$z = \cos(x) + \frac{\cos(2x)}{4} + \dots + \frac{\cos(nx)}{n^2},$$

до тех пор, пока не выполнится условие $\frac{\cos(nx)}{n^2} \leq Eps.$

Задача 1. Вычислить сумму членов ряда

$$z = \cos(x) + \frac{\cos(2x)}{4} + \dots + \frac{\cos(nx)}{n^2},$$

до тех пор, пока не выполнится условие $\frac{\cos(nx)}{n^2} \leq Eps.$

Дано: x – вещ., Eps – вещ.

Задача 1. Вычислить сумму членов ряда

$$z = \cos(x) + \frac{\cos(2x)}{4} + \dots + \frac{\cos(nx)}{n^2},$$

до тех пор, пока не выполнится условие $\frac{\cos(nx)}{n^2} \leq Eps.$

Дано: x – вещ., Eps – вещ.

Результат: z – вещ.

Задача 1. Вычислить сумму членов ряда

$$z = \cos(x) + \frac{\cos(2x)}{4} + \dots + \frac{\cos(nx)}{n^2},$$

до тех пор, пока не выполнится условие $\frac{\cos(nx)}{n^2} \leq Eps.$

Дано: x – вещ., Eps – вещ.

Результат: z – вещ.

При : $Eps > 0$, $x \neq 0$

Задача 1. Вычислить сумму членов ряда

$$z = \cos(x) + \frac{\cos(2x)}{4} + \dots + \frac{\cos(nx)}{n^2},$$

до тех пор, пока не выполнится условие $\frac{\cos(nx)}{n^2} \leq Eps.$

Дано: x – вещ., Eps – вещ.

Результат: z – вещ.

При : $Eps > 0$, $x \neq 0$

Связь: см. формулу в условии.

Алгоритм

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

z:=0 {сумма членов ряда}

i:=1 {номер слагаемого}

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

z:=0 {сумма членов ряда}

i:=1 {номер слагаемого}

цикл

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

$z := 0$ {сумма членов ряда}

$i := 1$ {номер слагаемого}

цикл

$slag := \cos(ix)/i^2$ {слагаемое}

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

Алгоритм

алг "итерация 1"

нач

ввод(x, Eps)

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация "

$z=0$

нач

ввод(x, Eps)

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z=\cos(x)$

$z:=z+slag$ {вычисление
суммы}

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z=\cos(x)$

$z:=z+slag$ {вычисление
суммы}

i=2

$i:=i+1$ {увеличение номера
шага}

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z=\cos(x)$

$z:=z+slag$ {вычисление
суммы}

i=2

$i:=i+1$ {увеличение номера
шага}

$slag=\cos(2x)/4$

до $slag \leq Eps$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z=\cos(x)$

$z:=z+slag$ {вычисление
суммы}

i=2

$i:=i+1$ {увеличение номера
шага}

$slag=\cos(2x)/4$

до $slag \leq Eps$

$z=\cos(x) + \cos(2x)/4$

кц

вывод(z)

кон

Трассировка

алг "итерация 1"

$z=0$

нач

ввод(x, Eps)

i=1

$z:=0$ {сумма членов ряда}

$i:=1$ {номер слагаемого}

$slag=\cos(x)$

цикл

$slag:=\cos(ix)/i^2$ {слагаемое}

$z=\cos(x)$

$z:=z+slag$ {вычисление
суммы}

i=2

$i:=i+1$ {увеличение номера
шага}

$slag=\cos(2x)/4$

до $slag \leq Eps$

$z=\cos(x) + \cos(2x)/4$

кц

вывод(z)

И т.д.

кон

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$y_1 = x^n; \quad y_2 = \frac{1}{2}x^{n-1}; \quad y_3 = \frac{1}{3}x^{n-2};$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{до } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; \end{aligned}$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{до } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{до } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

$$\frac{y_i}{y_{i-1}} =$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

$$\frac{y_i}{y_{i-1}} = \frac{1}{i}x^{n-i+1} : \frac{1}{i-1}x^{n-i+2} =$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{до } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

$$\frac{y_i}{y_{i-1}} = \frac{1}{i}x^{n-i+1} : \frac{1}{i-1}x^{n-i+2} = \frac{i-1}{i} \times \frac{x^{n-i+1}}{x^{n-i+2}} =$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{do } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

$$\frac{y_i}{y_{i-1}} = \frac{1}{i}x^{n-i+1} : \frac{1}{i-1}x^{n-i+2} = \frac{i-1}{i} \times \frac{x^{n-i+1}}{x^{n-i+2}} = \frac{i-1}{ix}.$$

Задача 2. Вычислить сумму членов ряда

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \quad \text{до } y_i < Eps.$$

Здесь формула для члена ряда не дана. Выведем ее:

$$\begin{aligned} y_1 &= x^n; & y_2 &= \frac{1}{2}x^{n-1}; & y_3 &= \frac{1}{3}x^{n-2}; \\ y_i &= \frac{1}{i}x^{n-i+1}; & y_{i-1} &= \frac{1}{i-1}x^{n-i+2}. \end{aligned}$$

Для того, чтобы выразить слагаемое через предыдущее:

$$\frac{y_i}{y_{i-1}} = \frac{1}{i}x^{n-i+1} : \frac{1}{i-1}x^{n-i+2} = \frac{i-1}{i} \times \frac{x^{n-i+1}}{x^{n-i+2}} = \frac{i-1}{ix}.$$

$$y_i = y_{i-1} \times \frac{i-1}{ix}.$$

Дано: x – вещ., Eps – вещ., n – цел.

Результат: y – вещ.

При : $Eps > 0$, $x \neq 0$

Связь:

$$y = x^n + \frac{1}{2}x^{n-1} + \dots + \frac{1}{n}x + \frac{1}{n+1} + \frac{1}{n+2}x^{-1} + \dots, \text{do } y_i < Eps.$$

алг “итерация 2”

нач

ввод (x, Eps, n)

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

цикл

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

цикл

slag:=slag*(i-1)/(ix) {слагаемое}

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

цикл

slag:=slag*(i-1)/(ix) {слагаемое}

y:=slag+y {вычисление суммы}

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y{ слагаемое}

i:=2 {номер слагаемого}

цикл

slag:=slag*(i-1)/(ix) {слагаемое}

y:=slag+y {вычисление суммы}

i:=i+1 {увеличение номера шага}

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

цикл

 slag:=slag*(i-1)/(ix) {слагаемое}

 y:=slag+y {вычисление суммы}

 i:=i+1 {увеличение номера шага}

до slag<Eps

кц

алг “итерация 2”

нач

ввод (x, Eps, n)

y:=xⁿ {сумма членов ряда}

slag:=y {слагаемое}

i:=2 {номер слагаемого}

цикл

 slag:=slag*(i-1)/(ix) {слагаемое}

 y:=slag+y {вычисление суммы}

 i:=i+1 {увеличение номера шага}

до slag<Eps

кц

вывод(y)

кон

Задача 3. Вычислить сумму членов ряда

$$z = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots + (-1)^n \frac{x^{2n}}{(2n)!},$$

До тех пор пока не выполнится условие: $|a_i + a_{i+1}| < Eps$,

где $a_i = (-1)^i \frac{x^{2i}}{(2i)!}$,

используя цикл с предусловием и цикл с постусловием.

Выведем рекуррентное отношение

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

$$\frac{a_i}{a_{i-1}}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

$$\frac{a_i}{a_{i-1}} = \frac{(-1)^i \cdot x^{2i}}{(2i)!} \cdot \frac{(2i-2)!}{(-1)^{i-1} \cdot x^{2i-2}}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

$$\frac{a_i}{a_{i-1}} = \frac{(-1)^i \cdot x^{2i}}{(2i)!} \cdot \frac{(2i-2)!}{(-1)^{i-1} \cdot x^{2i-2}} = -\frac{x^2 \cdot (2i-2)!}{(2i)!}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$
$$\frac{a_i}{a_{i-1}} = \frac{(-1)^i \cdot x^{2i}}{(2i)!} \cdot \frac{(2i-2)!}{(-1)^{i-1} \cdot x^{2i-2}} = -\frac{x^2 \cdot (2i-2)!}{(2i)!} =$$
$$= -\frac{x^2 \cdot 1 \cdot 2 \cdot \dots \cdot (2i-2)}{1 \cdot 2 \cdot \dots \cdot (2i-2)(2i-1)(2i)}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$
$$\frac{a_i}{a_{i-1}} = \frac{(-1)^i \cdot x^{2i}}{(2i)!} \cdot \frac{(2i-2)!}{(-1)^{i-1} \cdot x^{2i-2}} = -\frac{x^2 \cdot (2i-2)!}{(2i)!} =$$
$$= -\frac{x^2 \cdot 1 \cdot 2 \cdot \dots \cdot (2i-2)}{1 \cdot 2 \cdot \dots \cdot (2i-2)(2i-1)(2i)} = -\frac{x^2}{(2i-1)(2i)}$$

Выведем рекуррентное отношение

$$a_i = (-1)^i \frac{x^{2i}}{(2i)!}; \quad a_{i-1} = (-1)^{i-1} \frac{x^{2(i-1)}}{(2(i-1))!} = (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

$$\frac{a_i}{a_{i-1}} = \frac{(-1)^i \cdot x^{2i}}{(2i)!} \cdot \frac{(2i-2)!}{(-1)^{i-1} \cdot x^{2i-2}} = -\frac{x^2 \cdot (2i-2)!}{(2i)!} =$$

$$= -\frac{x^2 \cdot 1 \cdot 2 \cdot \dots \cdot (2i-2)}{1 \cdot 2 \cdot \dots \cdot (2i-2)(2i-1)(2i)} = -\frac{x^2}{(2i-1)(2i)}$$

$$a_i = a_{i-1} \cdot \left(-\frac{x^2}{(2i-1)(2i)} \right)$$

Алгоритм

алг "постусловие"

нач

ввод(x, eps)

$z := 1$ {сумма}

$\text{curr} := 1$ {текущее слагаемое}

$i := 1$ {номер члена ряда}

цикл

$\text{pred} := \text{curr}$ {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z := z + \text{curr}$

$i := i + 1$

до $|\text{curr} + \text{pred}| < \text{eps}$

кц

вывод(z)

кон

Трассировка

алг "постусловие"

нач

ввод(x, ϵ)

$z := 1$ {сумма}

$curr := 1$ {текущее слагаемое}

$i := 1$ {номер члена ряда}

цикл

$pred := curr$ {предыдущее слагаемое}

$$curr := pred \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z := z + curr$

$i := i + 1$

до $|curr + pred| < \epsilon$

кц

 вывод(z)

кон

Трассировка

алг "постусловие"

$z=1;$

нач

ввод(x, eps)

$z := 1$ {сумма}

$\text{curr} := 1$ {текущее слагаемое}

$i := 1$ {номер члена ряда}

цикл

$\text{pred} := \text{curr}$ {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z := z + \text{curr}$

$i := i + 1$

до $|\text{curr} + \text{pred}| < \text{eps}$

кц

вывод(z)

кон

Трассировка

алг "постусловие"

$z=1;$

нач

ввод(x, eps)

$\text{curr}=1;$

$z:=1$ {сумма}

$\text{curr}:=1$ {текущее слагаемое}

$i:=1$ {номер члена ряда}

цикл

$\text{pred}:=\text{curr}$ {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z:=z+\text{curr}$

$i:=i+1$

до $|\text{curr}+\text{pred}| < \text{eps}$

кц

вывод(z)

кон

Трассировка

алг "постусловие"

$z=1;$

нач

ввод(x, ϵ)

$curr=1;$

$z:=1$ {сумма}

i=1

$curr:=1$ {текущее слагаемое}

$i:=1$ {номер члена ряда}

цикл

$pred:=curr$ {предыдущее слагаемое}

$$curr := pred \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z:=z+curr$

$i:=i+1$

до $|curr+pred| < \epsilon$

кц

вывод(z)

кон

Трассировка

алг "постусловие"

$z=1;$

нач

ввод(x, ϵ)

$curr=1;$

$z:=1$ {сумма}

i=1

$curr:=1$ {текущее слагаемое}

$pred=curr=1;$

$i:=1$ {номер члена ряда}

цикл

$pred:=curr$ {предыдущее слагаемое}

$$curr := pred \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

$z:=z+curr$

$i:=i+1$

до $|curr+pred| < \epsilon$

кц

вывод(z)

кон

Трассировка

алг "постусловие"
нач
 ввод(x, eps)
 z:=1 {сумма}
 curr:=1 {текущее слагаемое}
 i:=1 {номер члена ряда}
цикл
 pred:=curr {предыдущее слагаемое}
 curr := $pred \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$
 z:=z+curr
 i:=i+1
до |curr+pred|<eps
кц
 вывод(z)
кон

z=1;
curr=1;
i=1
pred=curr=1;
curr=-x²/2; z=1- x²/2

Трассировка

алг "постусловие"

нач

ввод(x, eps)

z:=1 {сумма}

curr:=1 {текущее слагаемое}

i:=1 {номер члена ряда}

цикл

pred:=curr {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

z:=z+curr

i:=i+1

до |curr+pred|<eps

кц

вывод(z)

кон

z=1;

curr=1;

i=1

pred=curr=1;

curr=-x²/2; z=1- x²/2

i=2

Трассировка

алг "постусловие"

нач

ввод(x, eps)

z:=1 {сумма}

curr:=1 {текущее слагаемое}

i:=1 {номер члена ряда}

цикл

pred:=curr {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

z:=z+curr

i:=i+1

до |curr+pred|<eps

кц

вывод(z)

кон

z=1;

curr=1;

i=1

pred=curr=1;

curr=-x²/2; z=1- x²/2

i=2

pred=curr =-x²/2;

Трассировка

алг "постусловие"

нач

ввод(x, eps)

z:=1 {сумма}

curr:=1 {текущее слагаемое}

i:=1 {номер члена ряда}

цикл

pred:=curr {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

z:=z+curr

i:=i+1

до |curr+pred|<eps

кц

вывод(z)

кон

z=1;

curr=1;

i=1

pred=curr=1;

curr=-x²/2; z=1- x²/2

i=2

pred=curr =-x²/2;

$$\text{curr} = \frac{-x^2}{2} \cdot \frac{(-x^2)}{3 \cdot 4} = \frac{x^4}{4!}$$

Трассировка

алг "постусловие"

нач

ввод(x, eps)

z:=1 {сумма}

curr:=1 {текущее слагаемое}

i:=1 {номер члена ряда}

цикл

pred:=curr {предыдущее слагаемое}

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

z:=z+curr

i:=i+1

до |curr+pred|<eps

кц

вывод(z)

кон

z=1;

curr=1;

i=1

pred=curr=1;

curr=-x²/2; z=1- x²/2

i=2

pred=curr =-x²/2;

$$\text{curr} = \frac{-x^2}{2} \cdot \frac{(-x^2)}{3 \cdot 4} = \frac{x^4}{4!}$$

z=1- x²/2+x⁴/4!

алг “предусловие”

нач

ввод(x, eps)

z:=1-x²/2 {сумма}

pred:=1 {предыдущее слагаемое}

curr:=-x²/2 {текущее слагаемое}

i:=2 {номер члена ряда}

цикл-пока | curr+pred | ≥ eps

 pred:=curr

$$\text{curr} := \text{pred} \cdot \left(-\frac{x^2}{2i(2i-1)} \right)$$

 z:=z+curr

 i:=i+1

кц

 вывод(z)

кон

Задача. Вычислить сумму членов ряда

$$s = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + \frac{x^{2n+1}}{(2n+1)!} + \dots$$

До тех пор пока не выполнится условие $|a_i - a_{i+1}| < Eps$,

используя цикл с предусловием.

Задача 1. Найти максимальную цифру заданного натурального числа. Написать постановку задачи и программу.

Дано: r-вещ

Результат: max-цел

При: $r > 0$; $r \leq \text{maxint}$; $r = [r]$

Связь: $n = [r]$

$n = \overline{z_{k-1} z_{k-2} \dots z_1 z_0}$, где z_i – цифры числа r ,

$i = \overline{0, k-1}$

$\exists m=0, k-1: \forall i=0, k-1 z_m \geq z_i, \text{max}=z_m$

```
#include <stdio.h>
#include<limits.h>
int main()// определение максимальной цифры числа
{float r;
int n,d,max,k;
do
{
    printf("Введите натуральное n ");
    k=scanf("%f", &r);
    if(k) n=(int)r;
    while (getchar()!='\n');
}
while (r<=0 || r>INT_MAX || r!=n || !k);
max=0;
while (n>0)
{ d=n % 10; // вычисляем последнюю цифру
  if (d>max ) max=d;
  n=(int)(n / 10); //отбрасываем последнюю цифру
}
printf("max = %d\n", max);
return 0;
}
```

Массивы

Массив - пронумерованная поименованная совокупность данных одного типа. Массив имеет имя и тип элементов.

Например,

a [] [] [] [] [] цел
0 1 2 3 4

- одномерный целочисленный массив из 5 элементов (а – имя массива).

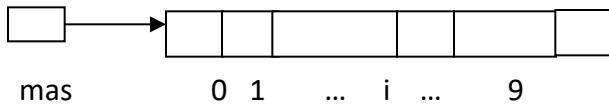
Нумерация элементов массива в С начинается с 0. Число элементов массива в С при статическом выделении памяти задается при описании массива и в дальнейшем не изменяется.

Для обращения к элементам массива используется можно использовать имя массива и номер элемента, например, $a[i]$. При этом i – константа или переменная соответствующего типа.

Массивы и матрицы

Имя массива – указатель на его первый элемент, т.е. записи `mas` и `&mas[0]` эквивалентны.

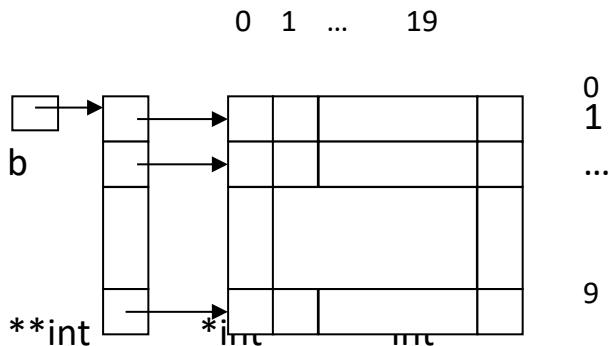
```
int mas[10];
```



Значение i -ого элемента массива можно представить двумя способами:
`mas[i]`; или `*(mas+i)`.

Для матрицы при объявлении указывается число строк и столбцов.
Нумерация элементов также начинается с нуля.

```
int b[10][20].
```



Обращение к элементу
`b[i][j]`

`*(b[i]+j)`
`*(*(b+i)+j)`

Алгоритмы с досрочным выходом из цикла

В задачах с досрочным выходом из цикла всегда используется целый тип, так как сравнивать на равенство можно только целые числа. Для вещественного типа значения всегда задаются с определенной точностью.

Досрочный выход из цикла подразумевает, что цикл может закончиться раньше, чем будет просмотрен весь массив.

Для досрочного выхода используются циклы с предусловием и с постусловием.

Задача 2. Дан массив $X[1:n]$ и число A. Если в массиве есть хотя бы один элемент, равный заданному числу, то вычислить сумму всех элементов массива, расположенных после него. В противном случае вывести сообщение об отсутствии суммы.

Дано: n-цел, $X[1:n]$ -цел, A-цел

Результат: S-цел или сообщение «нет суммы»

При : $n \in \mathbb{N}$, $n \leq L_{\max}$

Связь: $\exists k: k=1, \underline{n-1}: X[k]=A, \nexists t: t=1, \underline{k-1}: X[t]=A.$

$$S = \sum_{i=k+1}^n X[i].$$

алг «досрочный выход 1»

нач

ввод(n , $X[1:n]$, A)

flag:=false {признак наличия
 $X[i]=A$ }

$i:=1$ {номер элемента}

цикл-пока $i < n$ и flag=false {не
дошли до конца массива и не
нашли элемента, равного
заданному}

если $X[i]=A$ то

$k:=i+1$ {запоминаем номер
элемента, с которого начнется
суммирование}

flag:=true

{элемент найден}

иначе $i:=i+1$

всё

кц

если flag=false то

вывод(«нет суммы»)

иначе

$s:=0$

цикл от $i:=k$ до n

$s:=s+X[i]$

кц

вывод(s)

всё

кон

Для досрочного выхода из цикла можно использовать другое условие.

i:=1 {номер элемента}

цикл пока i<n и X[i]≠A {не дошли до конца массива и не нашли элемента, равного заданному}

i:=i+1

кц

если i=n то

вывод(«нет суммы»)

иначе

{вычисление суммы начиная с i+1-го элемента}

всё

алг «без досрочного выхода »

нач

ввод(n , $X[1:n]$, A)

$flag := false$

$s := 0$ {сумма}

цикл от $i := 1$ до $n - 1$

если $flag = true$ то

{суммирование происходит
когда нужный элемент уже
найден}

$s := s + X[i]$

иначе

если $X[i] = A$ то

$flag := true$

{элемент найден}

всё

всё

кц

если $flag = false$ то

вывод(«нет суммы»)

иначе

$s := s + X[n]$ {прибавляем
последний элемент}

вывод(s)

всё

кон

Задача 3. Найти номера первого и второго отрицательного элемента одномерного массива. Поменять местами значения этих элементов.

```
#include <iostream>
#include <stdio.h>
int main ()
{setlocale(LC_ALL,"RUS");
int a[10],na,i,n1,n2,k;
printf ("введите длину массива A:");scanf("%d",&na);
printf ("введите массив A\n");
for (i=0;i<na;i++)  scanf("%d",&a[i]);
n1=n2=-1; //инициализация номеров
// поиск номеров двух первых отрицательных элементов
for(i=0;i<na&&n2== -1;i++)
// цикл завершится когда будет найден второй отрицательный элемент
if (a[i]<0)
if(n1== -1)
n1=i;
else
n2=i;
```

```
// перестановка и вывод результата
if (n2== -1)
    printf(" Нет двух отрицательных элементов");
else
{printf("1 отрицательный %d\n 2 отрицательный%d\n",a[n1],a[n2]);
 k=a[n1],a[n1]=a[n2],a[n2]=k;
printf ("Получен массив A\n");
for (i=0;i<na;i++)
    printf("%5d",a[i]);}
return 0;
}
```

Задача 5. Даны матрица $A[1:na, 1:ma]$ и массив $B[1:nb]$. В каждой строке матрицы поменять местами первый четный и последний нечетный элемент, если оба эти элемента присутствуют в одномерном массиве.

Дано : na-цел, ma-цел, $A[1:na, 1:ma]$ -цел, nb-цел, $B[1:nb]$ -цел

Результат: $A'[1:na, 1:ma]$ -цел или сообщения('нет обмена в ', i, ' строке')

При: $na \in N$, $nb \in N$, $ma \in N$, $ma \leq lmax$, $na \leq lmax$, $nb \leq lmax$

Связь: $\forall i=1,na:$

$\exists j1:A[i,j1] : 2 \text{ и } \exists j: 1 \leq j < j1: A[i,j] : 2$

(т.е. $j1$ – № столбца, содержащего первый чётный элемент i -й строки) :

$\exists j2:\underline{\text{не}}(A[i,j2] : 2) \text{ и } \exists j: j2 < j \leq ma : \underline{\text{не}}(A[i,j] : 2):$

(т.е. $j2$ – № столбца, содержащего последний нечётный элемент i -й строки)

$\exists x,y: B[x]=A[i,j1], B[y]=A[i,j2]: x=1,nb, y=1,nb:$

$c=A[i,j1]; A[i,j1]=A[i,j2]; A[i,j2]=c.$ — —

алг "обмен"

нач

ввод(на, ма, A[1..n,1..ma],
nb, B[1..nb])

цикл от i:=1 до на

j1:=0;

j2:=0;

цикл от j:=1 до ма

если A[i,j] : 2 то

если j1=0 то

j1:=j

всё

иначе

j2:=j

всё

кц

если j1=0 или j2=0 то

вывод('нет обмена в ',

i, ' строке')

иначе

flag1:=false;

flag2:=false;

t:=1;

цикл-пока t<=nb и

((flag1=false) или (flag2=false))

если A[i,j1]=b[t] то flag1:=true всё

если A[i,j2]=b[t] то flag2:=true всё

t:=t+1

кц

если flag1 и flag2 то

c:=A[i,j1]

A[i,j1]:=A[i,j2]

A[i,j2]:=c

иначе

вывод('нет обмена в ',i,' строке')

всё

всё

кц

вывод(a[1..na,1..ma])

кон

```
#include <stdio.h>
#define lmax 20
int main()
{int a[lmax][lmax], b[lmax], flag1, flag2, na, ma, nb, t, i, j,
 k, c, j1, j2;
 printf("Введите число строк и столбцов матрицы a\n");
 do{k=scanf("%d%d",&na,&ma);
 while(getchar()!='\n') ;
 }while(na<=0||ma<=0||na>=lmax||ma>=lmax||k!=2);
 printf("Введите матрицу %d на %d\n",na,ma);
 for( i=1;i<=na;i++)
 for( j=1;j<=ma;j++)
 scanf("%d",&a[i][j]);
 printf("Введите длину массива b\n");
 do{k=scanf("%d",&nb);
 while(getchar()!='\n') ;
 } while(nb>=lmax||nb<0||k!=1);
 printf(" Введите массив b из %d элементов\n", nb);
 for( i=1;i<=nb;i++) scanf("%d",&b[i]);
```

```
for( i=1;i<=na;i++)
{
    j1=j2=0;
    for( j=1;j<=ma;j++)
        if (a[i][j] % 2==0 )
        { if (j1==0) j1=j;
        }
        else j2=j;
    if (j1==0 || j2==0)
        printf("нет перестановки в %d строке\n",i);
    else {flag1=flag2=0; t=1;
        while (t<=nb&&(flag1==0 || flag2==0))
    {
        if (b[t]==a[i][j1])
            flag1=1;
        if (b[t]==a[i][j2])
            flag2=1;
        t=t+1;
    }
}
```

```
    if (flag1==false || flag2==false)
        printf("нет перестановки в %d строке\n",i);
    else { c=a[i][j1];
            a[i][j1]=a[i][j2];
            a[i][j2]=c;
        }
    }

printf("Матрица после перестановки\n");
for( i=1;i<=na;i++)
{
    for( j=1;j<=ma;j++)
        printf("%8d",a[i][j]);
    printf("\n");
}
return 0;
}
```


Задача . Вычислить произведение двух матриц $A[1:n, 1:m]$ и $B[1:m, 1:L]$, получив матрицу $C[1:n, 1:L]$. Написать постановку задачи, алгоритм .
Используется формула:

$$C[i, k] = \sum_{j=1}^m A[i, j] * B[j, k].$$

$A[1:n, 1:m]$

1	2	3	1
4	0	1	2

$b[1:m, 1:L]$

1	0	1
1	1	1
0	1	0

1	1	0

$C[1:n, 1:L]$

4	6	

$$c[1,1] = 1*1 + 2*1 + 3*0 + 1*1 = 4$$

$$c[1,2] = 1*0 + 2*1 + 3*1 + 1*1 = 6$$

etc

Постановка задачи

Дано: n-цел, m-цел, L-цел, A[1:n, 1:m]-вещ,
B[1:m, 1:L]-вещ

Результат: C[1:n, 1:L]-вещ

При : n ∈ N, n <= lmax, m ∈ N, m <= lmax, L ∈ N,
L <= lmax

Связь: $\forall i=1, \overline{n}, \forall k=1, \overline{L}$

$$C[i, k] = \sum_{j=1}^m A[i, j] * B[j, k].$$

алг «умножение матриц»

нач

ввод($n, m, L, A[1:n, 1:m], B[1:m, 1:L]$)

цикл от $i:=1$ до n

цикл от $k:=1$ до L

$C[i,k]:=0$

цикл от $j:=1$ до m

$C[i,k]:=C[i,k]+A[i,j]*B[j,k]$

кц

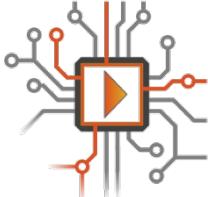
кц

вывод($C[1:n, 1:L]$)

кон

```
#include <stdio.h>
#define lmax 20
int main()
{float a[lmax][lmax], b[lmax][lmax], c[lmax][lmax];
Int i, j, k, l, m, n;
    printf(«Ведите n,m,l \n»);
    do{k=scanf("%d%d%d",&n,&m,&l);
        while(getchar()!='\n') ;
    }while(n<=0||m<=0||l<=0||n>=lmax||m>=lmax||l>=lmax|
|k!=3);
    printf(«введите матрицу a %d на %d\n»,n,m);
    for( i=1;i<=n;i++)
        for( j=1;j<=m;j++)
            scanf("%f",&a[i][j]);
    //ввод матрицы b аналогично
```

```
for( i=1;i<=n;i++)
    for( k=1;k<=l;k++)
    {
        c[i][k]=0;
        for( j=1;j<=m;j++)
            c[i][k]=c[i][k]+a[i][j]*b[j][k];
    }
printf("«Матрица C\n");
for( i=1;i<=n;i++)
{
    for( k=1;k<=l;k++)
        printf ("%8/3f",c[i][k]);
    printf("\n");
}
return 0;    }
```



Алгоритмы сортировки

Сортировка – перестановка элементов массива в определенном порядке.

Исходный массив

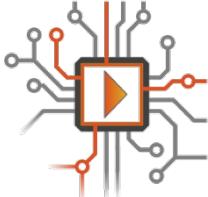
A	2	-3	0	5	3	1
---	---	----	---	---	---	---

По возрастанию

A	-3	0	1	2	3	5
---	----	---	---	---	---	---

По убыванию

A	5	3	2	1	0	-3
---	---	---	---	---	---	----



Метод установки

Каждый элемент массива последовательно сравнивается со всеми последующими. При необходимости элементы меняются местами. На каждое место в массиве последовательно устанавливается нужный элемент. Рассмотрим сортировку по возрастанию.

2	-3	0	5	3	1
---	----	---	---	---	---

-3	2	0	5	3	1
----	---	---	---	---	---

-3	0	2	5	3	1
----	---	---	---	---	---

-3	0	1	5	3	2
----	---	---	---	---	---

-3	0	1	3	5	2
----	---	---	---	---	---

-3	0	1	2	5	3
----	---	---	---	---	---

-3	0	1	2	3	5
----	---	---	---	---	---

Постановка задачи

Дано: n -цел, $A[1:n]$ -вещ

Результат: $A'[1:n]$ -вещ

При : $n \in N$, $n \leq l_{max}$

Связь: $\forall i=1, \overline{n-1} : A'[i] \leq A'[i+1].$

алг «сортировка методом установки по возрастанию»

нач

ввод(n , $A[1:n]$)

цикл от $i:=1$ до $n-1$ {что сравниваем}

цикл от $j:=i+1$ до n {с чем сравниваем}

если $A[i] > A[j]$ то

$C := A[i]$

$A[i] := A[j]$

$A[j] := C$ {перестановка}

всё

кц

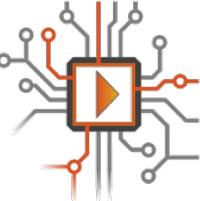
кц

вывод($A[1:n]$)

кон

Особенности

- Сортировка по убыванию отличается от сортировки по возрастанию только знаком («меньше» вместо «больше»).
- Если массив изначально упорядочен, то метод установки производит то же количество сравнений что и в неупорядоченном массиве. Число сравнений в этом методе зависит только от длины массива, а не от расположения элементов массива



Метод «пузырька»

Попарно сравниваются соседние элементы массива. Если элементы в паре расположены в неверном порядке, то они меняются местами.

Если при очередном просмотре массива были перестановки, то процесс сравнения пар будет выполняться еще раз (после того, как очередное сравнение пар завершится).

Сортировка закончится тогда, когда при очередном просмотре пар не будет ни одной перестановки. Рассмотрим сортировку по возрастанию.

2	-3	0	5	3	1
---	----	---	---	---	---

-3	2	0	5	3	1
----	---	---	---	---	---

-3	0	2	5	3	1
----	---	---	---	---	---

-3	0	2	3	5	1
----	---	---	---	---	---

-3	0	2	3	1	5
----	---	---	---	---	---

-3	0	2	1	3	5
----	---	---	---	---	---

-3	0	1	2	3	5
----	---	---	---	---	---

алг «сортировка методом «пузырька» по возрастанию»

нач

ввод(n , $A[1:n]$)

цикл

flag:=true {признак завершения сортировки}

цикл от $i:=1$ до $n-1$ {номер первого элемента пары}

если $A[i] > A[i+1]$ то

$C := A[i]$

$A[i] := A[i+1]$

$A[i+1] := C$ {перестановка}

flag:=false {перестановка была}

всё

кц

до flag[=true]

кц

вывод($A[1:n]$)

кон

Сравнение методов

- Метод установки проще.
- Метод «пузырька» дает меньшее число сравнений, особенно если исходный массив частично упорядочен.
- В методе установки уже упорядоченные элементы располагаются в начале массива, а в методе «пузырька» - в конце массива.

Задача. Дан массив $A[1:n]$. Найти в нем k максимальных элементов, используя методы сортировки.

Задача. Дан массив $A[1:n]$. Найти в нем k максимальных элементов, используя методы сортировки.

алг «сортировка методом установки по убыванию»

нач

ввод($n, k, A[1:n]$)

цикл от $i:=1$ до k {что сравниваем}

цикл от $j:=i+1$ до n {с чем сравниваем}

если $A[i] < A[j]$ то

$C := A[i]$

$A[i] := A[j]$

$A[j] := C$ {перестановка}

всё

кц

кц

вывод($A[1:k]$)

кон

Также можно использовать метод «пузырька». Сортируем элементы по возрастанию, производя не более k просмотров всех пар. Затем выводим k последних элементов массива.

Также можно использовать метод «пузырька». Сортируем элементы по возрастанию, производя не более k просмотров всех пар. Затем выводим k последних элементов массива.

алг «сортировка методом «пузырька» по возрастанию»

нач

ввод($n, k, A[1:n]$)

$j:=0$

цикл

flag:=true {признак завершения сортировки}

цикл от $i:=1$ до $n-1$ {номер первого элемента пары}

если $A[i] > A[i+1]$ то

$C:=A[i]$

$A[i]:=A[i+1]$

$A[i+1]:=C$ {перестановка}

flag:=false {перестановка была}

всё

кц

$j:=j+1$

до flag[=true] или $j=k$

кц

вывод($A[n-k+1:n]$)

кон

Использование методов сортировки при обработке матриц



Задача. Данна целочисленная матрица $a[1:n][1:m]$. Упорядочить элементы каждой строки по возрастанию. В полученной матрице расположить строки по убыванию сумм элементов. Для наглядности изображен массив сумм строк матрицы.

Исходная матрица

1	0	-3	9	1
3	-4	6	2	0
4	3	2	1	7

После первой сортировки

-3	0	1	1	9	8
-4	0	2	3	6	7
1	2	3	4	7	17

После второй сортировки

1	2	3	4	7	17
-3	0	1	1	9	8
-4	0	2	3	6	7

```
#include<stdio.h>
#define nmax 20
int main()
{int a[nmax][nmax],sum[nmax],i,j,k,n,m,b, flag;
 printf(«Введите n,m \n»);
 do{
     k=scanf("%d%d",&n,&m);
     while(getchar()!='\n') ;
     }while(n<=0 || m<=0 || n>=nmax || m>=nmax || k!=2);
     printf(«Введите матрицу а %d на %d\n»,n,m);
     for( i=1;i<=n;i++)
     for( j=1;j<=m;j++)
         scanf("%d",&a[i][j]);
```

```
//сортировка элементов строк по возрастанию //(установка)
for(i=1;i<=n;i++)
    for( j=1;j<m;j++)
        for( k=j+1;k<=m;k++)
            if (a[i][j]>a[i][k] )
            {
                b=a[i][j];a[i][j]=a[i][k];a[i][k]=b;
            }
printf(«Матрица после первой сортировки:\n»);
for( i=1;i<=n;i++)
{
    for( j=1;j<=m;j++)
        printf("%8d",a[i][j]);
        printf("\n");
}
```

```
for( i=1;i<=n;i++) //вычисление сумм элементов строк
{
    sum[i]=0;
    for( j=1;j<=n;j++)
        sum[i]=sum[i]+a[i][j];
}
//сортировка массива сумм и матрицы пузырьком
do{  flag=1;
    for( i=1;i<n;i++)
        if (sum[i]<sum[i+1])
            { b=sum[i]; sum[i]=sum[i+1]; sum[i+1]=b; flag=0;
            for( j=1;j<=m;j++)
                { b=a[i][j];
                a[i][j]=a[i+1][j];
                a[i+1][j]=b;
                }
            }
    }
while(!flag);
```

```
printf("матрица после второй сортировки:\n");
for( i=1;i<=n;i++)
{
    for( j=1;j<=m;j++)
        printf("%8d",a[i][j]);
    printf("\n");
}
return 0;
}
```

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$

1	2	4	1	2	3	1
---	---	---	---	---	---	---

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td></tr></table>	1						
1								

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td></tr></table>	1	2					
1	2							

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td><td>1</td></tr></table>	1	2	1				
1	2	1						

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td><td>1</td><td>2</td></tr></table>	1	2	1	2			
1	2	1	2					

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td><td>1</td><td>2</td><td>1</td></tr></table>	1	2	1	2	1		
1	2	1	2	1				

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td><td>1</td><td>2</td><td>1</td></tr></table>	1	2	1	2	1		
1	2	1	2	1				

Постановка задачи

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	1	2	4	1	2	3	1
$b[1:nb]$	1	2	1	2	1		

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	<table border="1"><tr><td>1</td><td>2</td><td>4</td><td>1</td><td>2</td><td>3</td><td>1</td></tr></table>	1	2	4	1	2	3	1
1	2	4	1	2	3	1		
$b[1:nb]$	<table border="1"><tr><td>1</td><td>2</td><td>1</td><td>2</td><td>1</td></tr></table>	1	2	1	2	1		
1	2	1	2	1				

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$, или сообщение “нет массива b ”

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	1	2	4	1	2	3	1
$b[1:nb]$	1	2	1	2	1		

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$, или сообщение “нет массива b ”

При: $na \in N$, $na <= lmax$

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все повторяющиеся элементы массива a .

Например,

$a[1:na]$	1	2	4	1	2	3	1
$b[1:nb]$	1	2	1	2	1		

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$, или сообщение “нет массива b ”

При: $na \in N$, $na \leq l_{max}$

Связь: $\forall \overline{i=1,na} : \exists \overline{j=1,na} : i \neq j$ и $a[i] = a[j]$,
 $\exists \overline{k=1,nb} : b[k] = a[i]$.

алг “новый массив из повторяющихся элементов исходного массива”

алг "новый массив из повторяющихся элементов исходного массива"

нач

ввод(на, а[1:на])

nb:=0 {длина массива b}

алг "новый массив из повторяющихся элементов исходного массива"

нач

ввод(на, a[1:на])

nb:=0 {длина массива b}

цикл от i:=1 до на

{проверка повторения a[i] }

j:=1 {текущий номер элемента в массиве a }

алг "новый массив из повторяющихся элементов исходного массива"

нач

ввод(на, а[1:на])

nb:=0 {длина массива b}

цикл от i:=1 до на

{проверка повторения a[i] }

j:=1 {текущий номер элемента в массиве a }

цикл-пока j ≤ на и (a[i] ≠ a[j] или i=j)

{не дошли до конца массива и не нашли элемента, равного текущему элементу массива, но расположенного на другом месте}

алг "новый массив из повторяющихся элементов исходного массива"

нач

ввод(на, а[1:на])

nb:=0 {длина массива b}

цикл от i:=1 до на

{проверка повторения а[i] }

j:=1 {текущий номер элемента в массиве а }

цикл-пока j≤ на и (а[i]≠а[j] или i=j)

{не дошли до конца массива и не нашли элемента, равного текущему элементу массива, но расположенного на другом месте}

j:=j+1 {берем следующий элемент массива}

КЦ

если $j \leq n_a$ то { $a[i]$ повторяется}

{запишем $a[i]$ в массив b }

$nb := nb + 1$

$b[nb] := a[i]$

всё

если $j \leq n_a$ то { $a[i]$ повторяется}

{запишем $a[i]$ в массив b }

$n_b := n_b + 1$

$b[n_b] := a[i]$

всё

КЦ

если $j \leq n_a$ то { $a[i]$ повторяется}

{запишем $a[i]$ в массив b }

$nb := nb + 1$

$b[nb] := a[i]$

всё

КЦ

если $nb = 0$ то вывод("нет массива b ")

если $j \leq na$ то { $a[i]$ повторяется}

{запишем $a[i]$ в массив b }

$nb := nb + 1$

$b[nb] := a[i]$

всё

КЦ

если $nb = 0$ то вывод("нет массива b ")

иначе

вывод ($b[1:nb]$)

всё

если $j \leq na$ то { $a[i]$ повторяется}

{запишем $a[i]$ в массив b }

$nb := nb + 1$

$b[nb] := a[i]$

всё

КЦ

если $nb = 0$ то вывод("нет массива b ")

иначе

вывод ($b[1:nb]$)

всё

КОН

- Если необходимо проверить, что элемент не повторяется в массиве, то заменяем только условие, расположенное после цикла-пока на если $j > i$ на то { $a[i]$ не повторяется}.

- Если необходимо проверить, что элемент не повторяется в массиве, то заменяем только условие, расположенное после цикла-пока на если $j > n$ на то { $a[i]$ не повторяется}.
- Нельзя заменить условие $j \leq n$ на $a[i] = a[j]$, т.к. в цикле-пока возможен выход индекса за границы массива.

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1
-----------	---

$kol[1:nb]$	1
-------------	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2
-----------	---	---

$kol[1:nb]$	1	1
-------------	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4
-----------	---	---	---

$kol[1:nb]$	1	1	1
-------------	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4
-----------	---	---	---

$kol[1:nb]$	2	1	1
-------------	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4
-----------	---	---	---

$kol[1:nb]$	2	2	1
-------------	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	2	2	1	1
-------------	---	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	1	1
-------------	---	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Результат: $b[1:nb]$, $kol[1:nb]$

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Результат: $b[1:nb]$, $kol[1:nb]$

При: $na \in N$, $na <= lmax$

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Результат: $b[1:nb]$, $kol[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Связь: $\forall i=1, na \exists k=1, nb: b[k]=a[i],$

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Результат: $b[1:nb]$, $kol[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Связь: $\forall i=1,na \exists k=1,nb: b[k]=a[i],$
 $\forall m=\overline{1,nb}, j=\overline{1,nb}: m \neq j \ b[m] \neq b[j]$

Задача. Дан массив $a[1:na]$. Подсчитать число повторений каждого элемента массива a .

Сформируем два новых массива. Массив $b[1:nb]$ содержит все различные элементы исходного массива a , а массив $kol[1:nb]$ содержит число повторений каждого элемента массива b .

Например:

$a[1:na]$	1	2	4	1	2	3	1	4
-----------	---	---	---	---	---	---	---	---

$b[1:nb]$	1	2	4	3
-----------	---	---	---	---

$kol[1:nb]$	3	2	2	1
-------------	---	---	---	---

Дано: na – цел., $a[1:na]$ - цел.

Результат: $b[1:nb]$, $kol[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Связь: $\forall i=1,na \exists k=1,nb: b[k]=a[i],$

$\forall m=\overline{1,nb}, j=\overline{1,nb}: m \neq j \quad b[m] \neq b[j]$

$\forall t=1,nb kol[t]=0, \forall L=1,na: a[L]=b[t], kol[t]=kol[t]+1$

Алгоритм

Алгоритм

алг “число повторений”

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массивов b и kol }

Алгоритм

алг “число повторений”

нач

ввод(на, а[1:на])

nb:=0 {длина массивов b и kol}

цикл от i:=1 до на

{проверка присутствия a[i] в массиве b}

j:=1 {текущий номер элемента в массиве b }

Алгоритм

алг “число повторений”

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массивов b и kol }

цикл от $i := 1$ до na

{проверка присутствия $a[i]$ в массиве b }

$j := 1$ {текущий номер элемента в массиве b }

цикл-пока $a[i] \neq b[j]$ и $j \leq nb$

$j := j + 1$

кц

если $j > n_b$ то { $a[i]$ отсутствует в b }

если $j > n_b$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

если $j > nb$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

$nb := nb + 1$

$b[nb] := a[i]$

$kol[nb] := 1$

если $j > nb$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

$nb := nb + 1$

$b[nb] := a[i]$

$kol[nb] := 1$

иначе

{цикл завершен при $a[i] = b[j]$. Увеличим число повторов на соответствующем месте}

если $j > nb$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

$nb := nb + 1$

$b[nb] := a[i]$

$kol[nb] := 1$

иначе

{цикл завершен при $a[i] = b[j]$. Увеличим число повторов на соответствующем месте}

$kol[j] := kol[j] + 1$

если $j > nb$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

$nb := nb + 1$

$b[nb] := a[i]$

$kol[nb] := 1$

иначе

{цикл завершен при $a[i] = b[j]$. Увеличим число повторов на соответствующем месте}

$kol[j] := kol[j] + 1$

всё

если $j > nb$ то { $a[i]$ отсутствует в b }

{запишем $a[i]$ в массив b (он повторяется 1 раз)}

$nb := nb + 1$

$b[nb] := a[i]$

$kol[nb] := 1$

иначе

{цикл завершен при $a[i] = b[j]$. Увеличим число повторов на соответствующем месте}

$kol[j] := kol[j] + 1$

всё

КЦ

вывод ($b[1:nb]$, $kol[1:nb]$)

КОН

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

$b[1:nb]$

1

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

$b[1:nb]$

1	4
---	---

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

$b[1:nb]$

1	4	2
---	---	---

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

$b[1:nb]$

1	4	2	0
---	---	---	---

Задача. Дан массив $a[1:na]$. Сформировать массив $b[1:nb]$, содержащий все различные элементы массива a по одному разу.

Например,

$a[1:na]$

1	4	2	1	2	0	-3	0
---	---	---	---	---	---	----	---

$b[1:nb]$

1	4	2	0	-3
---	---	---	---	----

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Постановка задачи

Дано: na – цел., a[1:na] – цел.

Результат: b[1:nb]

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Связь:

$$\forall i = \overline{1, na} \exists k = \overline{1, nb} : b[k] = a[i],$$

Постановка задачи

Дано: na – цел., $a[1:na]$ – цел.

Результат: $b[1:nb]$

При: $na \in \mathbb{N}$, $na \leq l_{max}$

Связь:

$$\forall \overline{i=1,na} \exists \overline{k=1,nb} : b[k] = a[i], \\ t = \overline{1,nb}, j = \overline{1,nb} : t \neq j \quad b[t] \neq b[j].$$

Алгоритм

алг “новый массив из различных элементов исходного массива”

Алгоритм

алг “новый массив из различных элементов исходного массива”

нач

ввод(на, а[1:на])

nb:=0 {длина массива b}

Алгоритм

алг “новый массив из различных элементов исходного массива”

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массива b }

цикл от $i := 1$ до na

{проверка присутствия $a[i]$ в массиве b }

$j := 1$ {текущий номер элемента в массиве b }

Алгоритм

алг “новый массив из различных элементов исходного массива”

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массива b }

цикл от $i := 1$ до na

{проверка присутствия $a[i]$ в массиве b }

$j := 1$ {текущий номер элемента в массиве b }

цикл-пока $a[i] \neq b[j]$ и $j \leq nb$

$j := j + 1$

кц

Алгоритм

алг "новый массив из различных элементов исходного массива"

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массива b }

цикл от $i := 1$ до na

{проверка присутствия $a[i]$ в массиве b }

$j := 1$ {текущий номер элемента в массиве b }

цикл-пока $a[i] \neq b[j]$ и $j \leq nb$

$j := j + 1$

кц

если $j > nb$ то { $a[i]$ отсутствует в b }

{увеличим длину массива b и запишем $a[i]$ в этот массив}

$nb := nb + 1$

$b[nb] := a[i]$

всё

Алгоритм

алг "новый массив из различных элементов исходного массива"

нач

ввод(na , $a[1:na]$)

$nb := 0$ {длина массива b }

цикл от $i := 1$ до na

{проверка присутствия $a[i]$ в массиве b }

$j := 1$ {текущий номер элемента в массиве b }

цикл-пока $a[i] \neq b[j]$ и $j \leq nb$

$j := j + 1$

кц

если $j > nb$ то { $a[i]$ отсутствует в b }

{увеличим длину массива b и запишем $a[i]$ в этот массив}

$nb := nb + 1$

$b[nb] := a[i]$

всё

кц

вывод ($b[1:nb]$)

кон

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,

$Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

$B[0:L2-1]$ (второй новый массив)

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

$B[0:L2-1]$ (второй новый массив)

2

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

$B[0:L2-1]$ (второй новый массив)

2	-3
---	----

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

1

$B[0:L2-1]$ (второй новый массив)

2	-3
---	----

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

1	2
---	---

$B[0:L2-1]$ (второй новый массив)

2	-3
---	----

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

1	2
---	---

$B[0:L2-1]$ (второй новый массив)

2	-3	8
---	----	---

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

1	2
---	---

$B[0:L2-1]$ (второй новый массив)

2	-3	8	4
---	----	---	---

Задача 1. Дан массив $Y[0:m-1]$. Вычислить два новых массива. Первый содержит элементы исходного массива, удовлетворяющие условию $y[i-1] < y[i] < y[i+1]$, а второй - все остальные.

Например, пусть $m=7$,
 $Y[0:m-1]$ (исходный массив)

2	-3	1	2	8	4	0
---	----	---	---	---	---	---

$A[0:L1-1]$ (первый новый массив)

1	2
---	---

$B[0:L2-1]$ (второй новый массив)

2	-3	8	4	0
---	----	---	---	---

Постановка задачи

Постановка задачи

Дано: m – цел., $Y[0:m-1]$ – вещ.

Постановка задачи

Дано: m – цел., $Y[0:m-1]$ – вещ.

Результат: $A[0:L1-1]$ или сообщение “нет массива A”
 $B[0:L2-1]$

Постановка задачи

Дано: m – цел., $Y[0:m-1]$ – вещ.

Результат: $A[0:L1-1]$ или сообщение “нет массива A”
 $B[0:L2-1]$

При: $m \in \mathbb{N}$, $m \leq L_{\max}$

Постановка задачи

Дано: m – цел., $Y[0:m-1]$ – вещ.

Результат: $A[0:L1-1]$ или сообщение “нет массива A”
 $B[0:L2-1]$

При: $m \in \mathbb{N}$, $m \leq L_{\max}$

Связь: $\forall i = \overline{1, m-2} \ \exists j = \overline{0, L1-1}: A[j] = Y[i]$, если
 $y[i-1] < y[i] < y[i+1]$;

Постановка задачи

Дано: m – цел., $Y[0:m-1]$ – вещ.

Результат: $A[0:L1-1]$ или сообщение “нет массива A”
 $B[0:L2-1]$

При: $m \in \mathbb{N}$, $m \leq L_{\max}$

Связь: $\forall i = \overline{1, m-2} \exists j = \overline{0, L1-1}: A[j] = Y[i]$, если

$y[i-1] < y[i] < y[i+1];$

$\forall i = \overline{0, m-1} \exists k = \overline{0, L2-1}: B[k] = Y[i]$, если $y[i-1] \geq y[i]$

или $y[i] \geq y[i+1]$,

алг "два новых массива
неизвестной длины"

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]; L2 := 1$

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]; L2 := 1$

цикл от $i := 1$ до $m-2$

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]; L2 := 1$

цикл от $i := 1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1] := Y[i]$

$L1 := L1 + 1$

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0] ; L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1+1$

иначе

$B[L2]:=Y[i]$

$L2:=L2+1$

алг "два новых массива
неизвестной длины"

нач

ввод ($m, Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0] ; L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1 + 1$

иначе

$B[L2]:=Y[i]$

$L2:=L2 + 1$

всё

КЦ

алг "два новых массива
неизвестной длины"

если $m > 1$ то

нач

ввод ($m, Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]$; $L2 := 1$

цикл от $i := 1$ до $m - 2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1] := Y[i]$

$L1 := L1 + 1$

иначе

$B[L2] := Y[i]$

$L2 := L2 + 1$

всё

КЦ

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]$; $L2 := 1$

цикл от $i := 1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1] := Y[i]$

$L1 := L1 + 1$

иначе

$B[L2] := Y[i]$

$L2 := L2 + 1$

всё

КЦ

если $m > 1$ то

$B[L2] := Y[m-1]$

$L2 := L2 + 1$

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]$; $L2 := 1$

цикл от $i := 1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1] := Y[i]$

$L1 := L1 + 1$

иначе

$B[L2] := Y[i]$

$L2 := L2 + 1$

всё

КЦ

если $m > 1$ то

$B[L2] := Y[m-1]$

$L2 := L2 + 1$

всё

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0]$; $L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и
 $Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1+1$

иначе

$B[L2]:=Y[i]$

$L2:=L2+1$

всё

КЦ

если $m>1$ то

$B[L2]:=Y[m-1]$

$L2:=L2+1$

всё

если $L1=0$ то

вывод ("нет массива A")

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0]$; $L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1+1$

иначе

$B[L2]:=Y[i]$

$L2:=L2+1$

всё

КЦ

если $m>1$ то

$B[L2]:=Y[m-1]$

$L2:=L2+1$

всё

если $L1=0$ то

вывод ("нет массива A")

иначе

вывод ($A[0:L1-1]$)

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0]$; $L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1+1$

иначе

$B[L2]:=Y[i]$

$L2:=L2+1$

всё

КЦ

если $m>1$ то

$B[L2]:=Y[m-1]$

$L2:=L2+1$

всё

если $L1=0$ то

вывод ("нет массива A")

иначе

вывод ($A[0:L1-1]$)

всё

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1:=0$ {длина массива A}

$L2:=0$ {длина массива B}

$B[L2]:=Y[0]$; $L2:=1$

цикл от $i:=1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1]:=Y[i]$

$L1:=L1+1$

иначе

$B[L2]:=Y[i]$

$L2:=L2+1$

всё

КЦ

если $m>1$ то

$B[L2]:=Y[m-1]$

$L2:=L2+1$

всё

если $L1=0$ то

вывод ("нет массива A")

иначе

вывод ($A[0:L1-1]$)

всё

вывод ($B[0:L2-1]$)

алг "два новых массива
неизвестной длины"

нач

ввод (m , $Y[0:m-1]$)

$L1 := 0$ {длина массива A}

$L2 := 0$ {длина массива B}

$B[L2] := Y[0]$; $L2 := 1$

цикл от $i := 1$ до $m-2$

если $Y[i-1] < Y[i]$ и

$Y[i] < Y[i+1]$ то

$A[L1] := Y[i]$

$L1 := L1 + 1$

иначе

$B[L2] := Y[i]$

$L2 := L2 + 1$

всё

КЦ

если $m > 1$ то

$B[L2] := Y[m-1]$

$L2 := L2 + 1$

всё

если $L1 = 0$ то

вывод ("нет массива A")

иначе

вывод ($A[0:L1-1]$)

всё

вывод ($B[0:L2-1]$)

кон

```
#include <stdio.h>
#define lmax 50
int main()
{float A[lmax], B[lmax], Y[lmax];
int i, L1,L2,m;

printf("input 0<m<=%d ", lmax);
scanf("%d", &m);
printf("input array y\n");
for (i=0; i<m; i++) scanf("%f", &Y[i]);
L1=0; //the length of A
L2=0; //the length of B
B[L2]=Y[0]; L2=1;
for(i=1;i<=m-2;i++)
    if( Y[i-1]<Y[i] && Y[i]<Y[i+1])
        A[L1]=Y[i],           L1=L1+1 ;
    else
        B[L2++]=Y[i];
```

```
if( m>1)
    B[L2++]=Y[m-1];

if (L1==0)
    printf ("Array A is empty\n");
else
{
    printf("Array a\n");
    for (i=0; i<L1; i++)      printf("%8.3f", A[i]);
    printf("\n");
}
printf("Array b\n");
for (i=0; i<L2; i++)  printf("%8.3f", B[i]);
printf("\n");
return 0;
}
```

Пусть определена переменная типа указатель на величину некоторого типа, например:

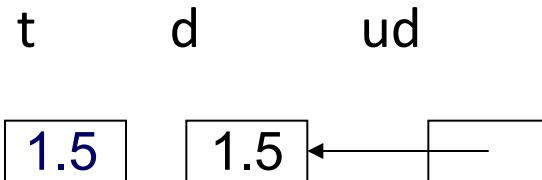
```
int *ukaz;  
char *uk;
```

Значением переменной типа **указатель** является **адрес**. С помощью операции взятия содержимого по адресу (`*ukaz`) можно получить значение, хранящееся в той ячейке памяти, на которую в данный момент показывает `ukaz` (переменная типа указатель).

Каждая переменная размещается в памяти, начиная с некоторого байта. Операция `&<имя переменной>` определяет адрес этой переменной. Например,

```
float t, d=1.5, *ud;
```

Выполним присваивание
`ud=&d;`



Тогда одинаковый эффект дают следующие операторы:

`t=d;`

`t=*ud; //берется значение по адресу ud`

Варианты ввода одномерного массива

```
int n, a[10], i, *uk;
```

Сначала вводится длина массива

```
printf ("Введите длину массива A: ");  
scanf ("%d", &n);
```

Затем можно использовать следующие варианты
ввода:

- `for (i=0; i<n; i++) scanf("%d", &a[i]);`
- `for (i=0; i<n; i++) scanf("%d", a+i);`
- `for (i=0; i<n; scanf("%d", &a[i++]));`
- `for (i=0; i<n; scanf("%d", a+i++));`
- `for (i=0, uk=a; i<n; i++) scanf("%d", uk++);`
- `for(uk=a; uk<a+n; uk++) scanf("%d", uk);`

Удаление путем сдвига

Задача 1. Удалить из массива $A[0:k-1]$ все элементы, удовлетворяющие условию $B < A[i] \leq L$, $i=0, k-1$ путем сдвига, т.е. без формирования нового массива.

Алг «удаление путем сдвига»

Нач

ввод(k , $A[0:k-1]$, B , L)

$n := 0$ {количество оставшихся
элементов массива}

цикл от $i := 0$ до $k-1$

если $A[i] > L$ или

$A[i] \leq B$ то {остается}

$A[n] := A[i]$

$n := n + 1$

всё

кц

если $n = 0$ то

вывод(«полное
удаление»)

иначе

если $n = k$ то

вывод(«нет
удаления»)

иначе

вывод($A[0:n-1]$)

всё

всё

кон

Закодируем данный алгоритм с использованием указателей

```
#include <stdio.h>
int main()
{float a[50], *i, b,l; int k, n;

printf("Введите 0<k<=50 k = ");
scanf("%d", &k);
printf("Введите элементы массива\n");
for (i=a; i<a+k; i++)
    scanf("%f", i);
do{
    printf("Введите b, l (b<l) "); n=scanf("%f%f", &b,&l);
    while(getchar()!='\n');
}
while(n<2 || b>=l);
for (i=a,n=0; i<a+k; i++)
    if (*i>l || *i<=b) //элемент остается
        *(a+n++)=*i;
```

```
if (n==0) printf ("Полное удаление");
else
    if(n==k) printf ("Нет удаления");
    else
        {printf("Массив А после удаления: ");
         for (i=a; i<a+n; i++)
             printf("%10.5f", *i);
         printf ("\n");
        }
return 0;
}
```

Задача 2. Найти адреса первого и второго отрицательного элемента одномерного массива. Поменять местами значения этих элементов.

```
#include <iostream>
#include <stdio.h>
int main ()
{setlocale(LC_ALL,"RUS");
int a[10],na,*u1,*u2,k,*t;
//u1 и u2 - указатели на два первых отрицательных элемента
//t - указатель на текущий элемент массива
printf ("введите длину массива A:"); scanf("%d",&na);
printf ("введите массив A\n");
for (t=a;t<a+na;t++) scanf("%d",t);
u1=u2=NULL; //инициализация указателей
for(t=a;t<a+na&&u2==NULL;t++) //поиск адресов
// цикл завершится когда будет найден второй отрицательный //элемент
if (*t<0)
if(!u1)
u1=t;
else
u2=t;
```

```
// перестановка и вывод результата
if (!u2)
    printf(" Нет двух отрицательных элементов");
else
{printf("1 отрицательный %d\n 2 отрицательный%d\n",*u1,*u2);
 k=*u1,*u1=*u2,*u2=k;
 printf ("Получен массив A\n");
 for (t=a;t<a+na;)
    printf("%5d",*t++);
 return 0;
}
```

Задача 3. Даны два одномерных массива целых чисел А и В.
Сформировать массив С, состоящий из повторяющихся
элементов массива А, отсутствующих в массиве В.

1. Есть новый массив

A
1 2 1 2 2 3 4

B
1 3 0 -5

C
2 2 2

2. Нет нового массива

A
1 2 3 3 3 4

B
1 2 3 4

```
/* Сформировать массив C, состоящий из повторяющихся
элементов массива A, отсутствующих в массиве B. */
```

```
#include <stdio.h>
#include <iostream>
//для setlocale
int main ()
{ setlocale(LC_ALL,"RUS");
int a[10], b[10], c[10], na, nb, nc=0, i, j;
printf ("введите длину массива A:");
scanf("%d",&na);
printf ("введите массив A\n");
for (i=0;i<na;i++)
    scanf("%d",&a[i]);
//ввод массива b аналогично
for(i=0;i<na;i++)
{ //проверка повторения a[i]
    for(j=0;j<na&&(a[i]!=a[j] || i==j);j++);
        if (j<na) //повторяется
            { //проверка отсутствия a[i] в b
                for (j=0;j<nb&&a[i]!=b[j];j++);
                //цикл закрыт
                if (j==nb) //условие отсутствия
                    c[nc++]=a[i];
            }
        }
//вывод результата
if (nc==0)
    printf("массив C пуст\n");
else
{
    printf("Массив C");
    for (i=0;i<nc;i++)
        printf("%7d",c[i]);
    printf ("\n");
}
return 0;
}
```

```
/* Сформировать массив C, состоящий из повторяющихся элементов массива A,
отсутствующих в массиве B. Используем указатель для обращения к элементам */
```

```
#include <stdio.h>
#include <iostream>
int main ()
{ setlocale(LC_ALL,"RUS");
int a[10], b[10], c[10], na, nb, nc=0, *ua,
    *ua1, *ub, *uc=c;
printf ("введите длину массива A:");
scanf("%d",&na);
printf ("введите массив A\n");
for (ua=a;ua<a+na;ua++)
    scanf("%d",ua);
//ввод массива b аналогично
for(ua=a;ua<a+na;ua++)
{ //проверка повторения *ua
    for(ua1=a;ua1<a+na&&(*ua!=*ua1) ||
        ua==ua1);ua1++);
```

```
if (ua1<a+na) //повторяется
{ //проверка отсутствия *ua в b
for (ub=b;ub<b+nb&&*ua!=*ub;ub++);
if (ub==b+nb) //условие отсутствия
    *uc=*ua, uc++, nc++;
}
//вывод результата
if (nc==0)
printf("массив C пуст\n");
else
{ printf("Массив C");
for (uc=c;uc<c+nc;uc++) printf("%7d",*uc);
printf ("\n");
}
return 0;}
```

Задача 4. В данной целочисленной матрице поменять местами первый положительный и последний отрицательный элемент. Вычислять адреса элементов.

```
#include <iostream>
#include <stdio.h>
int main()
{ int a[10][10],n,m,i,j,*u1,*u2,c;
setlocale(LC_ALL,"RUS");
do
{ printf("Введите количество строк"); scanf("%d",&n);
}
while (n<=0| |n>10);
do
{ printf("Введите количество столбцов"); scanf("%d",&m);
}
while (m<=0| |m>10);
```

```
printf("Введите матрицу\n");
for (i=0;i<n;i++)
    for(j=0;j<m;j++)
        scanf("%d",&a[i][j]);
u1=u2=NULL;
for (i=0;i<n;i++)
    for(j=0;j<m;j++)
    {
        if (a[i][j]>0&&u1==NULL)
            u1=&a[i][j];
        if (a[i][j]<0)
            u2=&a[i][j];
    }
```

```
if (!u1)
    printf("нет положительных");
else
    if (!u2)
        printf("нет отрицательных");
    else
    {   c=*u1,*u1=*u2,*u2=c;
        for (i=0;i<n;i++)
        {   for(j=0;j<m;j++)
            printf("%3d",a[i][j]);
            printf("\n");
        }
    }
return 0;
}
```

Определение индексов элементов матрицы, расположенных на, над и под главной и побочной диагоналями

Главная диагональ

0	1	...	n-2	n-1
0				
1				
...				
n-2				
n-1				

[j]);

```
printf("%7d",a[i][j]);
```

```
printf("на главной i=j\n");
for (i=0;i<n;i++)
    printf("\n");
printf("над главной\n");
for (i=0;i<n-1;i++)
    for (j=i+1;j<n;j++)
        printf("\n");
printf("под главной\n");
for (i=1;i<n;i++)
    for (j=0;j<i;j++)
        printf("\n");
```

```
printf("\n");
```



Побочная диагональ

0	1	...	$n-2$	$n-1$
0				
1				
\cdots				
$n-2$				
$n-1$				

```
printf("на побочной  i+j=n-1");
for (i=0;i<n;i++)
    printf("%7d",a[i][n-1-i]);
printf("\n");

printf("над побочной");
for (i=0;i<n-1;i++)
    for (j=0;j<n-i-1;j++)
        printf("%7d",a[i][j]);
printf("\n");

printf("под побочной");
for (i=1;i<n;i++)
    for (j=n-i;j<n;j++)
        printf("%7d",a[i][j]);
printf("\n");
```



Форматный ввод и вывод. Подробности

Опции вывода (флаги) могут появляться в любом порядке. После опций можно указать минимальную ширину поля вывода. Плюс имеет приоритет перед пробелом.

Флаг (опция вывода)	Результат
нет флага	обычная печать, правое выравнивание, пробелы для заполнения
-	левое выравнивание
0	дополнение нулями слева
+	вывод символа плюс для положительных чисел
пробел	невидимый пробел

Примеры форматного вывода

```
ch='A'; strcpy(string,"ABCDEFGH");// #include<string.h>
i=1; j=-1; e=12345; n=1234567; x=57; y=-12345678E-3;

printf("%-5d; %-5d", i,j);//выравнивание влево
//1_____-1_____
printf("%05d; %05d", i,j);//заполнение лидирующими нулями
//00001;-0001
printf("%+5d; %+5d", i,j);//печать знака принудительно
//__+1;__-1
printf("%+-5d; %+-5d", i,j);//знак и выравнивание влево
//+1_____-1_____
printf("%_5d; %_5d", 0,j);//невидимый плюс, 0 счит. положит
//0_____-1_____
.
```

```
printf("%_05d; %_05d", 0,j);//невид. плюс, заполнение 0-ми
//_0000;-0001
printf("%+05d; %+05d", i,j);//принуд. плюс, заполнение 0-ми
//+0001;-0001

printf("\n%.3s __ e is equal __%7d\n __ N = %-10ld",string, e, n);
//ABC __ e is equal ___12345
 //__ N = 1234567
//%.3s – ввод строки не длиннее 3 символов
//%-10ld – выравнивание по левому краю
printf("\nX = %e ___ Y = %f;\n", x, y);
//X = 5.7000000e+001 ___ Y = -12345.678000;
```

Код	Значение
%c	Считать один символ
%d	Считать десятичное число целого типа
%i	Считать десятичное число целого типа
%e	Считать число с плавающей запятой
%f	Считать число с плавающей запятой
%g	Считать число с плавающей запятой
%o	Считать восьмеричное число
%s	Считать строку
%x	Считать шестнадцатеричное число
%p	Считать указатель
%n	Принимает целое значение, равное количеству считанных до текущего момента символов
%u	Считывает беззнаковое целое
%[]	Просматривает набор символов

Наличие обычного символа заставляет `scanf()` считать и отбросить соответствующий символ. Например, формат "%d,%d" заставляет `scanf()` считать целое число, считать и отбросить запятую и затем считать еще одно целое число. Если указанный символ не обнаружен во входном потоке, `scanf()` останавливается.

Элементы вводимых данных должны разделяться пробелами, знаками табуляции или новой строки.

Знаки пунктуации, такие как запятая, точка с запятой и т.п., не считаются разделителями. Это значит, что для оператора

```
scanf("%d%d", &r, &c);
```

последовательность 10 20 будет воспринята, а последовательность 10,20 — нет.

- Знак *, помещенный после % и перед спецификатором формата, считывает данные указанного типа, но подавляет их присваивание. Таким образом, код
`scanf ("%d%*c%d", &x, &y);`
при вводе последовательности 10/20 присваивает значение 10 переменной x, отбрасывает символ / и присваивает значение 20 переменной y.
- Командами форматирования может задаваться модификатор максимальной ширины поля. Например, если необходимо считать не больше, чем 20 символов в массив address, следует написать
`scanf ("%20s", address);`
- Если входной поток содержал больше 20 символов, то при последующем вызове функция ввода начнет ввод с того места, где был остановлен ввод при текущем обращении. Ввод поля может быть прерван и до достижения максимальной длины поля, если встретится разделитель. В этом случае scanf() переходит к следующему полю.

Примеры форматного ввода

```
/* описание переменных */
char ch, string[20]; int i, j, e, *u;
long n;
float x;
double y;
```

Дана входная строка: ABC65432X1234.

1. scanf("ABC%ld%c%d", &n, &ch, &e);
printf("n=%ld; ch=%c; e=%d\n", n, ch, e);

//n=65432; ch=X; e=1234

//Ввод n осуществляется до первого символа,
отличающегося от цифры.

2. scanf("AB%c%f%*c%d", &ch, &x, &e);
printf("ch=%c; x=%8.3f; e=%d\n", ch, x, e);
//ch='C'; x=65432.000; e=1234;
//Символы "%*c" обозначают пропуск символа при вводе.

Дана входная строка: ABC65432X1234.

```
3. scanf("%3s%2d%3d%c%2f",string, &i, &j, &ch, &x);
printf("string=%s; i=%d; j=%5d; ch=%3c;
x=%f\n",string,i,j,ch,x);
//string=ABC; i=65; j=_ _432; ch=_ _X; x=12.000000;
(Остальные символы игнорируются).
```

Задача 7. Данна последовательность целых чисел A[0:n-1]. Найти длину максимальной последовательности из нулей и начало этой последовательности. Используем указатели и форматный ввод-вывод

Обозначения:

dtp – длина текущей последовательности из нулей

maxdp – максимальная длина последовательности

prmax – начало максимальной последовательности из нулей

nte - номер текущего элемента

//Алгоритм – вычислительная часть

maxdp:=0; dtp:=0;

цикл от nte:=0 до n-1

если a[nte]=0 то dtp:=dtp+1;

иначе

если dtp>maxdp то maxdp:=dtp; prmax:=nte-dtp;

всё

dtp:=0

всё

КЦ

если dtp>maxdp то maxdp:=dtp; prmax:=nte-dtp;

всё

```
#include <iostream>
#include <stdio.h>
int main()
{
    setlocale(LC_ALL, "RUS");
    int a[100], n, *ua, dtp, maxdp, npmax;
    printf("Введите 0<n<=100 n = "); scanf("%d", &n);
    printf("Введите элементы массива\n");
    for (ua=a; ua<a+n; ua++) scanf("%d", ua);
    maxdp=dtp=0;
    for (ua=a; ua<a+n; ua++)
        if (*ua==0) dtp++;
        else
    {
        if (dtp>maxdp) maxdp = dtp, npmax = (ua-a)-dtp;
        dtp=0;
    }
    if (dtp>maxdp) maxdp = dtp, npmax = (ua-a)-dtp;
    if (!maxdp) printf("Нет нулей");
    else
        printf("Maxdp=%d\nnpmax=%d\n ", maxdp ,npmax);
    return 0;
}
```

Пример. Ввод матрицы. Суммирование элементов матрицы различными способами.

```
int
    a[10][20],n,m,i,j,*ui,*uj,(*u)[20],
    s;

printf ("введите n и m :");
scanf("%d%d",&n,&m);
printf ("введите матрицу A\n");
for (i=0;i<n;i++)
for (j=0;j<m;j++)
scanf("%d",&a[i][j]);
//1
for (i=0,s=0;i<n;i++)
for (j=0;j<m;j++)
    s+=a[i][j];
printf("s=%10d ",s);
```

```
//2
for(ui=*a,s=0;ui<*a+n*20;ui+=20)
    for (uj=ui;uj<m+ui;uj++)
        s+=*uj;
printf("s=%10d ",s);

//3
for (u=a,s=0;u<a+n;u++)
    for (uj=*u;uj<*u+m;uj++)
        s+=*uj;
printf("s=%10d ",s);
//4 вариант обхода матрицы --//см.
задачу 3
```

Задача 2. Создать массив, содержащий суммы элементов строк заданной матрицы.

```
#include<iostream>
using namespace std;
int main()
{setlocale(LC_ALL,"RUS");
int a[10][20],n,m,i,j,*uj,b[10],*ub;

//ввод матрицы см. выше

for (i=0,ub=b;i<n;i++,ub++)
    for (uj=a[i],*ub=0;uj<a[i]+m;uj++)
        *ub+=*uj;

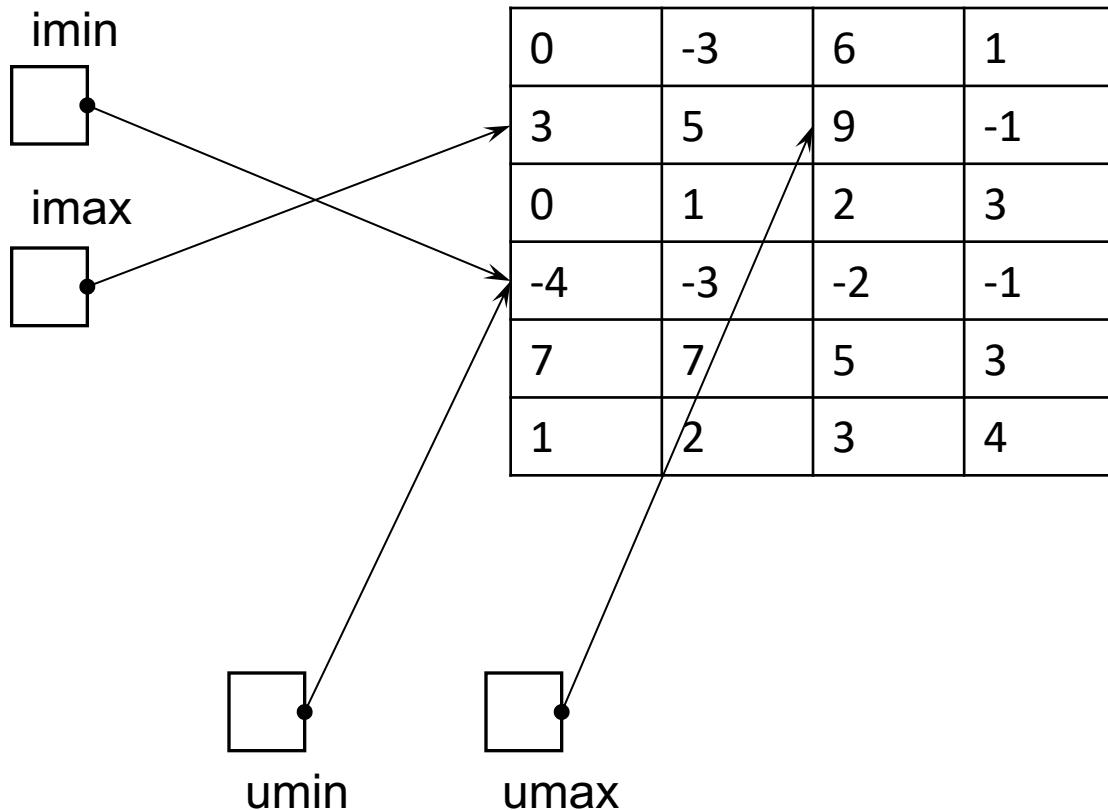
//вывод массива см. выше
return (0);
}
```

Задача 4. Создать массив, содержащий произведения элементов столбцов заданной матрицы.

```
#include<iostream>
using namespace std;
int main()
{setlocale(LC_ALL,"RUS");
int a[10][20],n,m,*ui,*uj,b[20],*ub;
// uj – указатель на начало столбца,
//ui – на элемент матрицы
//ввод матрицы см. выше
    for (uj=*a,ub=b;uj<*a+m;uj++,ub++)
        for (ui=uj,*ub=1; ui<a[n-1]+m;ui+=20)
            *ub=*ui*(*ub);
    //вывод массива см. выше
return (0);
}
```

Задача 1. Поменять местами строки матрицы, содержащие минимальный и максимальный элемент.

$a[0:n-1][0:m-1]$



```
#include <stdio.h>
#include <iostream>
int main ()
{
int a[10][20],n,m,i,*ui,*uj,*umin,*umax,*imin,*imax,b;
// imin,imax – указатели на начало строки с мин. и с макс.
//ввод матрицы см. выше
for (i=0,umin=umax=imin=imax=*a;i<n;i++)
    for (uj=a[i];uj<a[i]+m;uj++)
        { if(*uj<*umin) umin=uj,imin=a[i];
          if(*uj>*umax) umax=uj,imax=a[i];
        }
printf("min=%10d\nmax=%10d\n ",*umin,*umax);
if (imin==imax) printf ("No changes");
else
{
    for(ui=imin,uj=imax;ui<imin+m;ui++,uj++)
        {b=*ui,*ui=*uj,*uj=b;}
    //вывод матрицы см. выше
}
return (0);
}
```


Контрольные точки второго модуля

- 3 лабораторные работы, защита отчетов как в 1 модуле; (5 баллов)
- 1 контрольная работа (письменно) (3 балла);
- контроль усвоения материала на лекциях и семинарах (2 балла);
- экзамен (тест).

Dead-lines для лабораторных работ

1. 4 занятие;
2. 8 занятие;
3. 12 занятие

(считываются занятия обеих подгрупп по порядку, указанному в расписании).

Для вычисления оценки текущего контроля по дисциплине используется следующая таблица.

	Работа на семинарском занятии	Работа на лекции	Выполнение лабораторного практикума	Контрольная работа
1 модуль	1	2	7 (3+4)	
2 модуль	1	1	5(2+2+1)	3

В скобках указано распределение баллов по лабораторным работам.

Оценка за текущий контроль в 1 и 2 модуле учитывает результаты студента следующим образом.

Модуль 1. $O_{текущая\ 1} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски}$.

Модуль 2. $O_{текущая\ 2} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски} + O_{контр.\ работы}$.

Все оценки рассматриваются без округления.

Результирующая оценка учитывает оценки модулей. Промежуточная оценка за 1 и 2 модуль вычисляется по формуле

$O_{промежуточная\ 1\ и\ 2} = 0,3 * O_{текущая\ 1} + 0,5 * O_{текущая\ 2} + 0,2 * O_{экзамен\ 2\ модуль}$,

где $O_{текущая\ 1}$, $O_{текущая\ 2}$ – оценки текущего контроля 1, 2 модуля, без округления.

Округление производится один раз, после вычисления промежуточной оценки, по правилам арифметики.

Экзаменационная оценка не является блокирующей.
Промежуточная оценка за 1 и 2 модуль не может превышать 10 баллов, в случае превышения ставится промежуточная оценка 10 баллов.

Результирующая оценка за дисциплину вычисляется по формуле

$$O_{\text{результативная}} = 0,4 \cdot O_{\text{промежуточная 1 и 2}} + 0,6 \cdot O_{\text{промежуточная 3 и 4}}$$

Округление производится по правилам арифметики.

В диплом выставляется результирующая оценка.

- Ни один из элементов текущего контроля не является блокирующим.
- На некоторых семинарах и лекциях проводится тест или проверочная работа. Каждый вид работы оценивается от 1 до 4 баллов. В итоговую оценку эти баллы входят с коэффициентом, получаемым делением числа занятий, на которых проводилось оценивание, на общее количество занятий.
- При пропуске лекции или семинарского занятия по любой причине студент не может решить дополнительное задание для компенсации баллов, которые он мог бы получить на этом занятии.
- Кроме того, преподаватель может оценивать дополнительными баллами ответ студента у доски (максимум 2 балла) и активное участие в решении задач семинаров (например, выявление и исправление неточностей и ошибок в алгоритмах и при кодировании программ, внесение усовершенствований в алгоритм и т.п.) (максимум по 0.2 балла за каждый ответ).

- Для каждой лабораторной работы устанавливается срок защиты отчета (в 1 модуле на 2 и 4 занятиях, считая пары отдельно для каждой подгруппы). При своевременной защите работа оценивается полученным баллом, при опоздании на 1 неделю балл снижается на 40%, при опоздании на 2 недели балл снижается на 60% от полученной оценки. При опоздании более чем на 2 недели работа не оценивается.
- В случае пропуска занятий по уважительной причине (обязательно предоставление справки) срок сдачи лабораторной работы может быть перенесен на соответствующее количество рабочих дней.
- В случае пропуска занятий по уважительной причине (обязательно предоставление справки) предоставляется дополнительное время для написания контрольной работы (единственная дата переписывания заранее сообщается через старосту).
- Переписывание контрольной работы с целью повышения полученной оценки не допускается.

Начиная с л.р. 4 для подсчета результатов сдачи лабораторных работ используется следующая таблица.

Раздел	Максимальная оценка	Итоговая оценка
Работа программы	1	
Тесты	1	
Правильность алгоритма	3	
Ответы на вопросы	2	
Дополнительное задание	3	
Итого		

Количество столбцов «Итоговая оценка» равно числу программ, которые должны быть разработаны при выполнении соответствующей лабораторной работы.

Функции в языке Си

Функцией называется логически завершённый фрагмент кода программы, оформленный по специальным правилам.

Использование функций позволяет упростить отладку программ и избежать повторения однотипных фрагментов кода.

Программа на Си представляет собой совокупность функций, одна из которых – функция `main()` – главная – всегда выполняется первой. Остальные функции могут находиться как до главной, так и после неё. При этом если вызываемая функция идёт после вызывающей, то предварительно должен быть указан прототип вызываемой функции – заголовок с добавлением точки с запятой.

Си допускает вложенность функций и рекурсивный вызов (вызов функцией самой себя).



Описание функции

[<тип результата>] <имя функции>([<список формальных параметров с указанием их типов>])

```
{  
<операторы>  
}
```

Если тип функции не указан, то тип ее результата `int`. Если тип результата отличен от `void`, то в теле функции должен быть хотя бы один оператор

```
return <выражение>;
```

Вызов функции

[<имя переменной>=]<имя функции>([<список фактических параметров без указания типов>]);

Имя переменной не указывается, если тип функции `void`. Тип переменной должен соответствовать типу результата функции или быть приводимым к нему.

Пример 1.

```
void hello()  
{  
printf("Hello");  
}  
//вызов  
hello();
```

void означает, что функция не возвращает значения при помощи return.

() означают, что у функции нет входных параметров.

Пример 2. Суммирование двух чисел

```
int sum(int a, int b)
{
    int s ;
    s=a+b;
    return (s);
}
/* тело функции можно
заменить на return a+b;*/
```

Оператор `return`
возвращает результат
функции.

Вызов функции

```
int x, y, s;
s=sum (x, y);
```

`x, y` – фактические
параметры.

Вызов функции может
осуществляться как из
выражения, так и
самостоятельно:

```
y=2-sum(x, y);
s=sum (x+2, 3) ;
sum(x,10);
/* правильно, но
бессмысленно*/
```

Оператор `return` передаёт в вызывающую функцию только **одну** величину.

Передача параметров в приведенном примере осуществляется по значению. Это значит, что в вызываемой функции для каждого формального параметра выделяется область памяти, куда копируются значения фактических параметров при вызове.

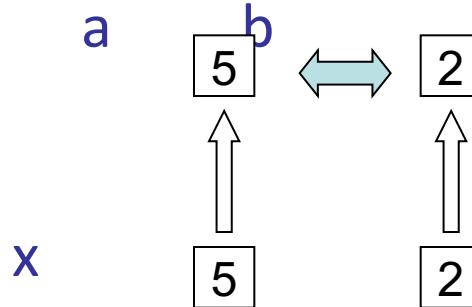
В подпрограмме все операции осуществляются только с **копиями** переменных. Для получения результатов из подпрограммы-функции следует использовать указатели в качестве формальных параметров.

Рассмотрим пример – функцию, которая меняет местами два числа.

Вариант 1.

до перестановки x=5, y=2
после перестановки x=5, y=2

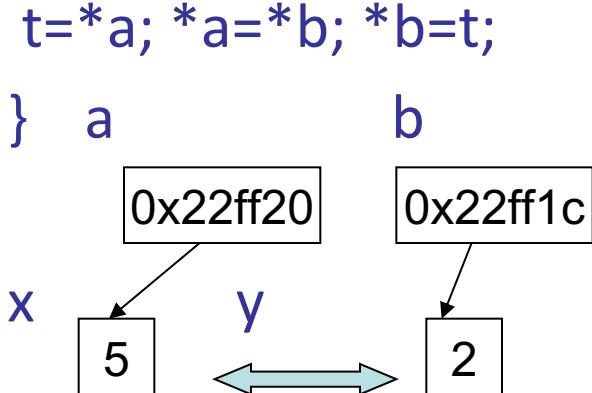
```
void f1(int a, int b)
{int t; //для перестановки
t=a; a=b; b=t;
}
int main()
{int x=5, y=2;
printf("до перестановки x=%d, y=%d\n",x, y);
f1(x, y);
printf("после перестановки x=%d, y=%d\n",x, y);
return 0;
}
```



При передаче параметров по значению во временной памяти создаются копии параметров. После выхода из функции копии уничтожаются и память освобождается.

Вариант 2. Передача параметров по адресу. Этот вариант правильный – происходит перестановка значений.

```
void f2(int *a, int *b)  
{  
int t;  
//перестановка значений
```



```
//вызов  
int main()  
{int x=5, y=2;  
printf("до перестановки x=%d,  
y=%d\n",x, y);  
f2(&x, &y);  
printf("после перестановки  
x=%d, y=%d\n",x, y);  
return 0;  
}
```

до перестановки x=5, y=2
после перестановки x=2, y=5

По адресу необходимо передавать

- переменные, значения которых изменяются в функции;
- переменные, занимающие большой объем памяти – массивы, структуры и т.п.

Особенности работы функций

- Если функция возвращает значение через `return`, то ее можно использовать в выражениях и операторе вывода, например

```
n=a+b+sum(x,y)*5;  
printf("%d", sum(x,y));
```

- Не следует возвращать из функции адреса локальных переменных, так как по завершении работы функции их значения уничтожаются и память освобождается.
- Если параметр передается по значению, то соответствующим ему фактическим параметром может быть константа, переменная или выражение, например

```
s=sum(a, 3);  
s=sum(b, a*3);
```

При передаче параметра по адресу фактическим параметром может быть только переменная соответствующего типа.

Функция ввода одномерного целочисленного массива и его длины. Результат функции – длина введенного массива.

```
int inp_arr(int b[])
{int nb, i;
printf(" Введите длину массива:");
scanf("%d", &nb);
printf(" Введите массив из %d элементов:\n",nb);
for(i=0; i<nb; i++)
    scanf("%d", b+i);
return nb;
}
//вызов
int b[20],nb;
nb=inp_arr(b);
```

Задача 2. Вставить элементы массива В в массив А после первого четного элемента этого массива с использованием указателей для обращения к элементам. Вычисления оформить в виде в функции

```
#include <stdio.h>
int* vstavka(int a[],int *na, int b[], int nb); // прототип функции
int main() {int a[20], b[10], na, nb, *up;
    printf("Введите длину массива A: na = "); scanf("%d",&na);
    printf("Введите элементы массива A:\n");
    for (up = a; up < a+na; up++)
        scanf("%d",up);
    printf("Введите длину массива B: nb = "); scanf("%d",&nb);
    printf("Введите элементы массива B:\n");
    for (up = b; up < b+nb; up++)
        scanf("%d",up);}
```

```
if (vstavka(a, &na, b, nb) == NULL) printf(" "Вставки нет");
else {
    printf(" "Массив А после вставки:" );
    for (up = a; up < a+na; up++) printf(" %6d", *up);
} return 0;
} //Ф-я возвращает адрес первого четного элемента в массиве а
int* vstavka(int a[], int *na, int b[], int nb) {
    int *u1, *ua, *ub;
    for (ua = a, u1 = NULL; ua < a + *na && u1 == NULL; ua++)
        if (*ua % 2 == 0)      u1 = ua;
    if (u1) {
        for (ua = a + (*na) - 1; ua > u1; ua--)      *(ua+nb) = *ua;
        for (ub = b; ub < b+nb; ub++)
            *(++ua) = *ub;
        *na = *na + nb;
    }
    return u1;
}
```

Задача 5. Дан массив. Переставить его элементы так, чтобы сначала расположились все неотрицательные элементы, а затем отрицательные. Порядок среди отрицательных и неотрицательных элементов должен быть сохранён. Дополнительный массив не использовать.

```
nc:=-1 //начало цепочки отрицательных элементов
цикл от i:=0 до n-1
    если a[i]>=0 то
        если nc<>-1 то /* уже есть отрицательные */
            b:=a[i]
            цикл от j:=i до nc+1 шаг -1
/* сдвиг неотрицательных вправо, т.е. освобождение места в конце
последовательности неотрицательных */
            a[j]:=a[j-1]
        кц
        a[nc]:=b
        nc:=nc+1
    всё
    иначе
        если nc=-1 то      nc:=i  всё
        всё
    кц
```

Рассмотрим массив:

-1 2 -3 5 0

i = 0
nc = 0
i = 1
1 2 -3 5 0
b = 2
-1 -1 -3 5 0
a[nc] = 2 , nc=1
2 -1 -3 5 0
i = 2
2 -1 -3 5 0

i = 3
2 -1 -3 5 0
b = 5
2 -1 -1 -3 0
a[nc] = 5 , nc=2
2 5 -1 -3 0
i = 4
2 5 -1 -3 0
b = 0
2 5 -1 -1 -3
a[nc] = 0 , nc=3
2 5 0 1 -3

```
#include <iostream> //для setlocale
#include <stdio.h> //стандартный ввод и вывод
//используем прототип функции перестановки
void sort(int n,int a[]);

int main()
{setlocale(LC_ALL, ".1251");
int a[50], n, i;
printf("Введите 0<n<=50 n = "); scanf("%d", &n);
printf("Введите элементы массива\n");
for (i=0; i<n; i++) scanf("%d", a+i);
sort(n,a); //вызов функции сортировки
printf("Преобразованный массив:\n");
for (i=0; i<n; i++) printf("%7d", a[i]);
printf('\n');
return 0; //признак успешного завершения программы
}
```

▲

```
void sort(int n, int a[])      //функция сортировки
{   int b, i, nc=-1, j;      //описание переменных

for (i=0; i<n; i++)
    if (a[i]>=0)
    {
        if (nc!=-1) /* сдвиг вправо, начиная с последнего */
        {
            for (b=a[i], j=i; j>=nc+1; j--)
                a[j]=a[j-1];
            a[nc++]=b;
        }
    }
else
    if (nc== -1) nc=i;
}
```

Δ

Задача . Написать программу для вычисления числа Фибоначчи с заданным номером.

Вычисления оформить в виде рекурсивной функции, ввод номера числа и вывод результата – в главной программе.

Числа Фибоначчи вычисляются по формуле:

$$F(n) = \begin{cases} 1, & n = 1, n = 2 \\ F(n-1) + F(n-2), & n > 2, n \in N \end{cases}$$

```
#include <stdio.h>
#include <limits.h>
float fib(int n)
{
    if(n==1||n==2) return 1;
    else return fib(n-1)+fib(n-2);
}
int main()
{
float r;
int n;
do{
    printf("enter n");
    scanf("%f",&r);
}while (r<=0||r>INT_MAX||r!=(int)r);
n=(int)r;
printf("Fibonacci(%d)=%10.0f \n",n,fib(n));
return 0;
}
```


Двумерные массивы. Ввод и вывод



```
void vv_matr(int a[][10], int *n, int *m) //или int (*a)[10]
{ //передаем массив указателей на строки, n и m – по адресу
int i, j;
printf("n = "); scanf("%d", n);
printf("m = "); scanf("%d", m);
printf("Введите матрицу размера %d на %d \n", *n, *m);
for(i=0; i<*n; i++)
    for(j=0; j<*m; j++)
        scanf("%d", a[i]+j);
}
```

//передаем n и m по значению

```
void viv_matr(int a[][10], int n, int m) {int i, j;
for(i=0; i<n; i++)
{
    for(j=0; j<m; j++)
        printf("%4d", a[i][j]);
    printf("\n");
}
}
```

```
.....  
int main()
{int a[10][10], n, m;
vv_matr(a,&n,&m);
viv_matr(a,n,m);
return 0;
}
```

Задача 1. Написать программу формирования одномерного массива В, содержащего k наибольших среди элементов квадратной целочисленной матрицы A[n][n], расположенных выше главной диагонали. Использовать функции для

- ввода исходных данных;
- вычисления В;
- вывода результата.

.

Обобщенный алгоритм:

ввод исходных данных;

инициализация $b[i]=INT_MIN$;

перебор элементов $a[i][j]$ выше главной
диагонали

если $a[i][j] >$ минимального в В то

замена минимального элемента В на $a[i][j]$

поиск нового минимального элемента в В

всё

Задача 1. Написать программу с использованием подпрограммы для вычисления одномерного массива, содержащего заданное число наибольших элементов среди элементов квадратной целочисленной матрицы, расположенных выше главной диагонали.

$A[0:n-1, 0:n-1]$, $k=3$ $B[0:k-1]$

1	2	3	5
0	4	7	-9
2	3	4	1
1	2	5	3

INT_MIN	INT_MIN	INT_MIN
2	INT_MIN	INT_MIN
2	3	INT_MIN
2	3	5
7	3	5

```
#include <stdio.h>
#include <limits.h>
void in_matr(int *kb, int *na, int ma[][20])
{  int maxkb, res, i, j, n;
do
    printf("Введите 0<n<=20 n = ");
    while (!scanf("%d", na) || *na<=0 || *na>20);
    maxkb=(*na**na-*na)/2;
    do    printf("Введите 0<k<=%d k = ", maxkb);
    while (!scanf("%d", kb) || *kb<=0 || *kb>maxkb);
    n=*na;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            do
                { printf("Введите a[%d][%d] = ", i, j);
                  if (!(res=scanf("%d", ma[i]+j)))  printf("Ошибка!\n");
                  while(getchar()!='\n');
                }
            while (!res);
    }
}
```

```
void form(int kb, int na, int ma[][][20], int mb[])
{
    int i, j, l, imin;

    /* Инициализация В */
    for (i=0; i<kb; mb[i++]=INT_MIN);
    imin=0;
    for (i=0; i<na-1; i++)
        for (j=i+1; j<na; j++)
            if (ma[i][j]>mb[imin])
            {
                mb[imin]=ma[i][j];
                /*Поиск минимума в В*/
                for (l=0; l<kb; l++)
                    if (mb[l]<mb[imin]) imin=l;
            }
}
```

```
void out(int b[ ], int k)
{
    int i;
    for (i=0; i<k; i++)
        printf("%7d", b[i]);
}
```

```
int main()
{
    int a[20][20], b[200], k, n;
    in_matr(&k, &n, a);
    form(k, n, a, b);
    out(b,k);
    return 0;
}
```

Оператор `return` в языке С обеспечивает досрочный выход из функции.

Задача 2. Бинарный (двоичный) поиск номера элемента в массиве, упорядоченном по возрастанию.

На каждом шаге количество рассматриваемых элементов уменьшается в 2 раза.

```
//двоичный поиск в массиве, упорядоченном по возрастанию
int search(int a[ ], int n, int number)
{int left, right, middle;
left=0; //левая граница
right=n-1; //правая граница
middle=(left+right)/2; //середина
while(left<=right)
{ if(a[middle]==number) return middle; //элемент найден
//если элемент левее середины, уменьшаем правую границу
if(a[middle]>number)right=middle-1;
//иначе увеличиваем левую границу
else left=middle+1;
//ищем середину подмассива с новыми границами
middle=(left+right)/2;
}
return -1; } //элемент не найден
```

```
int main()
{int i, n, a[20], pos, num,*ua;

printf("Введите 0<n<=100 n = "); scanf("%d", &na);
printf("Введите элементы массива\n");
for (ua=a;ua< a+na; ua++)
    do scanf("%d", ua);
    while (*ua<*(ua-1));
printf("Введите число для поиска"); scanf("%d", &num);
pos = search(a, n, num) ;
if (pos== -1) printf(" не найдено" );
else printf(" номер равен:%d", pos);
return 0;
}
```

Задача 3. Упорядочить элементы матрицы по всем строкам (по возрастанию). Дополнительный массив не использовать.
Вычисления оформить как отдельную функцию.

Метод:

Рассматриваем матрицу как одномерный массив, содержащий $n*m$ элементов. В сортировке «пузырьком» сравниваем 2 соседних элемента: $a[i][j]$ и следующий за ним $a[i1][j1]$. Индексы $i1$ и $j1$ вычисляются в зависимости от местоположения $a[i][j]$ в матрице:

если $j=m-1$ /* $a[i][j]$ находится в последнем столбце i -ой строки */

то $i1:=i+1; j1:=0;$ /* в качестве $a[i1][j1]$ берётся первый элемент следующей строки */

иначе $j1:=j+1; i1:=i;$ /* в качестве $a[i1][j1]$ берётся следующий элемент i -ой строки */

все

Кодируем с использованием индексов

Алгоритм решения задачи (вычислительная часть)

f:=1; m1:=n*m-1;

Цикл-пока (f)

 f:=0; m2:=m1; i:=0; j:=0;

цикл от k:=0 до m2-1

/* определение пары сравниемых элементов, т.е. вычисление
индексов следующего элемента i1, j1; сравниваем a[i][j] с a[i1][j1] */

если j=m-1 то

 i1:=i+1; j1:=0;

a[0:n-1][0:m-1]

иначе

 j1:=j+1; i1:=i; всё

если a[i][j]>a[i1][j1] то

 b:=a[i][j];

 a[i][j]:=a[i1][j1];

 a[i1][j1]:=b;

 f:=1; m1:=k; всё

 i:=i1; j:=j1;

кц

кц

	0	1	...	m-1
0				
1				
...				
n-1				

```
#include<stdio.h>
int sort(int n, int m, int a[][20])
{  int m1, m2, i , k, j, i1, j1, b;
   int f=1, f1=0;
   m1 = n*m-1;
   while(f)
   { f =0;
     i = j = 0;      m2 = m1;
     for (k = 0; k < m2; k++) {
       if (j == m-1)
       { i1 = i + 1;          j1 = 0;
         }
       else
       { j1 = j +1;          i1 = i;
         }
     }
   }
}
```

```
if (a[i][j] > a[i1][j1])
{ b = a[i][j];
  a[i][j] = a[i1][j1];
  a[i1][j1] = b;
  f = 1;
  f1=1;
  m1 = k;
}
i = i1;
j = j1;
}
return f1;
}
```

```
int main()
{
int n, m, i = 0, j;
int f1;
int a[20][20];
printf("Кол-во строк: ");
scanf("%d", &n); // n =
printf("Кол-во столбцов: ");
scanf ("%d", &m); // m =
printf("Ведите элементы матрицы : ");
for (i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        scanf("%d", (*(a+i)+j));
f1=sort(n,m,a);
```

```
if(f1)
{printf("Матрица после сортировки\n");
 for ( i = 0; i < n; i++)
{
    printf("\n");
    for (j = 0;j< m; j++)
        printf("%7d", *(*(a+i)+j));
}
else printf("Сортировки не было\n");
return 0;
}
```

Если указатель изменяется в функции, то необходимо передавать указатель на указатель.

Задача 4. Поменять местами два первых четных элемента матрицы. Для поиска и обмена использовать отдельные функции.

```
#include <stdio.h>
void adress(int a[ ][10],int n,int m, int **u1, int **u2)
{int i,*j;
 /*u1 - указатель на первый четный элемент.*u2 - на второй четный
 //элемент
*u1=*u2=NULL; //инициализация указателей
for(i=0;i<n&&*u2==NULL;i++) //поиск адресов первых двух четных
 //элементов
for(j=a[i];j<a[i]+m&&*u2==NULL;j++)
 if (!(*j%2))
 {if (*u1==NULL)
  *u1=j;
 else *u2=j;
}
}
```

```
//перестановка
void changes(int *u1, int *u2)
{
int t; //переменная для перестановки
t=*u1;
*u1=*u2;
*u2=t;
}
void in_matr(int a[ ][10],int*n,int*m); //прототип, текст см. выше
void out_matr(int a[ ][10],int n,int m)
{int *i,*j;
printf (" матрица A\n");
for (i=*a;i<*a+(n*10);i+=10)
{for(j=i;j<i+m;j++)
 printf("%7d",*j);
printf("\n");
}
}
```

```
int main ()  
{int a[10][10], n, m, *u1, *u2;  
  
vvod(a, &n, &m);  
adress(a, n, m, &u1, &u2);  
if(u2)  
{changes (u1,u2);  
 out_matr(a,n,m);  
}  
else printf("нет перестановки");  
 return 0;  
}
```


Распределение памяти. Классы памяти

Существует два основных способа распределения памяти: **статическое** и **динамическое** распределение.

При **статическом** распределении память под переменные выделяется до начала выполнения программы (на этапе компиляции или редактирования связей).

Достоинство: повышение эффективности программ (возрастает быстродействие).

Недостатки:

- увеличение требуемого объёма памяти;
- сложность рекурсии (надо создавать копии переменных);
- фиксированные размеры массивов.

При **динамическом** распределении память выделяется во время исполнения программы.

Класс памяти определяет область действия переменной и продолжительность её существования в памяти. Он устанавливается при описании переменной с соответствующим словом. Память при **описании** переменной не выделяется, но выделяется при **определении** переменной.

Свойства классов памяти

Класс памяти	Ключевое слово	Продолжительность существования	Область действия
автоматический	auto	временно	локальная
регистровый	register	временно	локальная
статический	static	постоянно	локальная
внешний	extern	постоянно	глобальная (все функции)
внешний статический	extern static	постоянно	глобальная (один файл)

Классы памяти, перечисленные выше линии, описываются внутри функции, перечисленные ниже линии - вне функции. Каждая переменная имеет тип и принадлежит к некоторому классу памяти.

Область видимости и время жизни переменных

1. **Область видимости** – это место в программе, где можно использовать переменные.

Если переменные декларируются в разных областях видимости, им могут быть присвоены одинаковые идентификаторы. В одной области видимости имена переменных должны быть разными.

2. **Время жизни** – промежуток времени, в течение которого существует переменная.

Локальные переменные (автоматические). Это переменные, которые декларируются **внутри** блока, помечаемого фигурными скобками. При входе программы в блок, отмеченный фигурными скобками, если в блоке имеются продекларированные переменные, они будут созданы. Когда программа будет покидать блок, произойдет удаление созданных в блоке переменных. Время жизни локальных переменных определяется временем нахождения программы в блоке, а область видимости – область блока. Локальные переменные не инициализируются без явного указания инициализации. Их используют только для временного хранения информации, обычно в рамках функции.

Глобальные переменные – это переменные, которые декларируются за рамками функций. Они живут ровно столько, сколько выполняется программа и использовать их можно в любом месте программы.

Все глобальные переменные находятся в одной области видимости, поэтому имена глобальных переменных в программе должны быть разными и не совпадать с именами функций, поскольку функции также находятся в глобальной области видимости.

Глобальные переменные компилятор автоматически инициализирует нулем.

В программах необходимо избегать использования глобальных переменных, т. к. значение такой переменной может быть изменено функциями программы, даже когда программист этого не ожидает. Это может приводить к тяжело отслеживаемым ошибкам в программе.

Автоматические переменные

По умолчанию переменные, описанные внутри функции, являются автоматическими. Переменная начинает существовать при вызове функции, содержащей эту переменную. Когда функция завершает свою работу, автоматическая переменная исчезает. Область её действия ограничена блоком в фигурных скобках.

Внешние переменные

Переменные, определённые вне функции являются внешними и имеют глобальную область действия. Внешние переменные, определённые раньше функции, доступны ей, даже если они не описаны внутри неё. Атрибут `extern` (ссылка на внешние переменные), описание может быть где угодно. Внешняя переменная - описана вне функций. Такая переменная может быть использована в любой функции, даже если эта переменная определена позже описания функции (или даже в другом файле).

Пример:

```
int err; char ch; //определения переменных
```

```
int main ()
```

```
{ extern int err; extern char ch;} /* эту группу описаний можно  
опустить если исходные определения появляются в том же файле и  
перед функцией, которая их использует */
```

```
/* file1.c */
```

```
static int m; /* глобальная статическая переменная */
int i,j; //глобальные
float a[20];
main ()
{
    /*описание – память не выделяется*/
    extern int i; //не обязательно
    extern int k; //обязательно
}
int f1()
{
    int i,j; //локальные
}
int k; /* глобальная переменная, определение после функции */
```

```
/* file2.c */
```

```
//описания
extern int i;
extern float a[];
f2()
{
    int f1(); /* описание прототипа
функции */
    f1(); /* вызов функции из другого
файла */
}
f3()
{
    extern int j; /*описание обязательн
о*/
}
```

Статические переменные

Они имеют такую же область действия, как автоматические переменные, но их значения **не исчезают** когда содержащая их функция закончит свою работу. Компилятор хранит эти значения от одного вызова функции до другого. **Например:**

```
int main()
{
void trt(); /* функция trt() описана по
зже, поэтому надо записать её прото-
тип в main*/
int c;
for (c=1;c<=3;c++)
{
printf("Итерация %d: ",c);
trt();
}
return 0;
}
```

```
void trt()
{
int f=1;
static int st=1;
printf("f=%d и st=%d\n",f++,st++);
}
```

результат:

Итерация 1: f=1 и st=1 Итерац
ия 2 : f=1 и st=2 Итерация 3 : f=
1 и st=3

Спецификатор **static** можно применять как для **глобальных**, так и для **локальных** переменных.

Для **глобальных** переменных спецификатор **static** сужает область видимости таких переменных до рамок файла, в котором они декларируются. Статические глобальные переменные могут быть полезны для того, чтобы избежать конфликтов имен в глобальной

области при создании программы несколькими программистами, а так же для создания функций (библиотеки функций), использующих глобальные переменные, которые планируется задействовать в разных программах.

Для **локальных** переменных спецификатор **static** продлевает время жизни такой переменной до конца жизни программы.

Внешние статические переменные

Внешняя статическая переменная описывается вне функции.

```
static r=1;
```

```
Rand() {...} //r - внешняя статическая для Rand
```

Разница между внешней переменной и внешней статической переменной заключается в области действия. Обычная внешняя переменная может использоваться функциями в **любом файле**. Внешняя статическая переменная может использоваться только функциями **того же самого файла**, причем после определения переменной.

Регистровые переменные

Регистровые переменные аналогичны автоматическим. Они создаются следующим образом:

```
main ()  
{ register int quick; }
```

Регистровые переменные запоминаются в регистрах ЦП. Доступ к ним выполняется быстрее. Операция **&** не применима к регистровым переменным.

Стандарты C89 и C99 декларируют "доступ к объекту так быстро, как только возможно".

Пример использования регистровых переменных - функция возведения целого числа m в степень e. e, m и result многократно используются для решения задачи в теле цикла.

```
int pwr(register int m, register int e)  
{  
register int result;  
result = 1;  
for(; e; e--) result *= m ;  
return result ;  
}
```

Операции с указателями

Язык Си предлагает пять основных операций, которые можно применить к указателям. Пусть имеется описание:

```
static int u[]={10,20,30}; //размер определяется автоматически  
int *ptr1,*ptr2, x, y;
```

Допустимы следующие операции:

1. присваивание `ptr1=u;`
2. определение значения `*ptr1=100; //только после присваивания`
3. получение адреса указателя `&ptr1;`
4. увеличение указателя `ptr1++;`
5. разность двух указателей (ссылающихся на элементы одного и того же массива) определяет расстояние, на котором элементы находятся друг от друга.

Правильно

```
ptr1++;  
x++;  
ptr2++;  
ptr2=u+1;
```

Неправильно

```
u++;  
3++;  
ptr2=u++;  
x=y+3++;
```

Различие объявлений

1. `int *u1;` //указатель на int
2. `int (*u2)[20];` //указатель на строку из 20 int
3. `int *u3[20];` //массив из 20 указателей на тип int

Увеличение (`u1++`) осуществляет переход к следующему целому (адрес увеличивается на 4).

(`u2++`) увеличивает адрес на 80 (20×4).

Задача 1. Дан одномерный целочисленный массив. Сформировать из его элементов два новых массива. Первый содержит четные элементы исходного массива, второй состоит из элементов с нечетными номерами. Для обращения к элементам массивов использовать указатели.

```
#include <stdio.h>
int main ()
{
int i,a[10],na,b[10],nb=0,c[10],nc=0,*ua,*ub,*uc; //ua,ub,uc - указатели //на текущий элемент
printf ("введите длину массива A:");scanf("%d",&na);
printf ("введите массив A\n");
for (ua=a;ua<a+na;ua++) scanf("%d",ua);
//инициализация указателей - настроены на начало массивов
for(ua=a, ub=b,uc=c;ua<a+na;ua++) //формирование массивов
{
if (!(*ua%2)) //элемент четный
{*ub=*ua;ub++;nb++;}
if ((ua-a)%2) //номер нечетный
{*uc=*ua;uc++;nc++;}
}
```

```
if(nb==0)
    printf(" Нет массива четных элементов\n");
else
{
    printf("Массив В ");
    for (int i=0;i<nb;i++)
        printf("%7d",b[i]);
    printf ("\n");
}
if(nc==0)
    printf(" Нет массива элементов с нечетными номерами\n");
else
{
    printf("Массив С ");
    for (uc=c;uc<nc+c;uc++)
        printf("%7d",*uc);
    printf ("\n");
}
return 0;
}
```

#define - макрос

Это директива, которая применяется при определении символьных констант, идентификаторов, макрофункций. Макросы, по-английски macro, бывают двух типов: подобные функциям и подобные объектам.

Подобные объектам

```
#define MaxLength 50
```

```
#define StrChange “My string”
```

Если строка-подстановка не помещается в строку файла, то в качестве знака переноса применяется обратный слэш.

```
#define Alphabet “qwertyuiopasdfghjklzxcvbnm\
```

```
QWERTYUIOPASDFGHJKLZXCVBNM”
```

Макросы, подобные объектам, не принимают параметров.

Подобные функциям

```
#include <stdio.h>
```

```
#define CircleLength(r) (6.28318 * (r))
```

```
int main()
{
    printf("\nCircle length = %.2f\n\n", CircleLength(12));
}
```

После имени макроса в макроопределении перед первой открывающей скобкой не должно быть пробела.

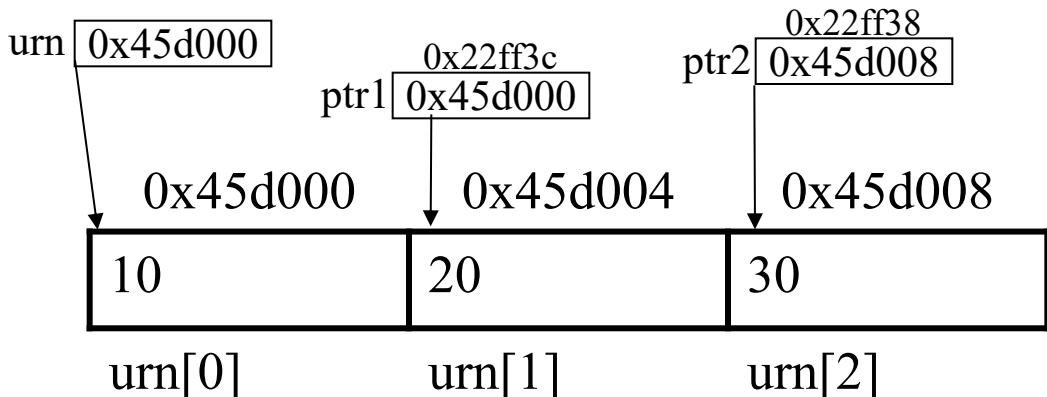
Пример:

```
#define PR(x) printf ("x=0x%0x, *x=%d, &x=0x%0x\n",x, *x, &x)
```

/* печатает адрес, значение,
находящееся по этому адресу и
адрес самого указателя*/

```
int main()
{ static int urn[]={10,20,30};
int *ptr1,*ptr2;
```

```
ptr1=urn; PR(ptr1) ;
ptr1++; PR(ptr1);
ptr2=&urn[2];PR(ptr2);
++ptr2; /* выходит за пределы
массива */
PR(ptr2);
return 0;
}
```



Результат:

```
x=0x45d000, *x=10, &x=0x22ff3c
x=0x45d004, *x=20, &x=0x22ff3c
x=0x45d008, *x=30, &x=0x22ff38
x=0x45d00c, *x=1, &x=0x22ff38
```

Задача. Вычислить сумму четных элементов, расположенныхных на побочной диагонали матрицы.

Существует два случая

$$m > n \\ i = \overline{1, n}$$

$$m \leq n \\ i = \overline{1, m} \\ k = \min(n, m) \\ i = \overline{1, k}$$

Элемент побочной диагонали
 $A[i, m + 1 - i]$

Постановка задачи

Дано: n – цел., m – цел., $A[1:n, 1:m]$ – цел.

Результат: sum или сообщение «нет
четных»

При: $n \in \mathbb{N}$, $n \leq lmax$, $m \in \mathbb{N}$, $m \leq lmax$

Связь:

$$k = \min(n, m)$$

$$sum = \sum_{i=1}^k A[i, m-i+1], где A[i, m-i+1] \vdots 2.$$

алг “сумма четных элементов на побочной диагонали”

нач

ввод($n, m, A[1:n, 1:m]$)

если $n < m$ то

$k := n$

иначе

$k := m$

всё

sum := 0; flag := false

цикл от $i := 1$ до k

если $A[i, m-i+1] \neq 0$ то

 sum := sum + $A[i, m-i+1]$

 flag := true

всё

кц

если не flag то

 вывод(“нет четных”)

иначе

 вывод(sum)

всё

кон

Не рекомендуется использовать (а тем более изменять) в функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}

int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Не рекомендуется использовать (а тем более изменять) в функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
```

```
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6
b=5
```

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
```

Результат

```
b=6
b=5
```

Трассировка

```
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6
b=5
```

Трассировка

a

```
1
```

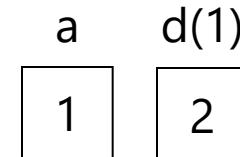
Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6
b=5
```

Трассировка



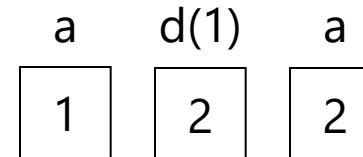
Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка



Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)
1	2	2	4

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

a
1

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

a	d(a)
1	2

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

a	d(a)	a
1	2	2

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

a	d(a)	a	d(1)
1	2	2	3

Не рекомендуется использовать (а тем более изменять) в процедурах и функциях глобальные переменные, так как это уменьшает универсальность подпрограмм и приводит к неожиданным посторонним эффектам.

```
#include<stdio.h>
int a,b; //global variables
int d(int x)
{
    int e;
    e=a+x;
    a=a+1;
    return e;
}
int main()
{
    a=1; b=d(1)+d(a);
    printf("b=%d\n",b);
    a=1; b=d(a)+d(1);
    printf("b=%d\n",b);
    return 0;
}
```

Результат

```
b=6  
b=5
```

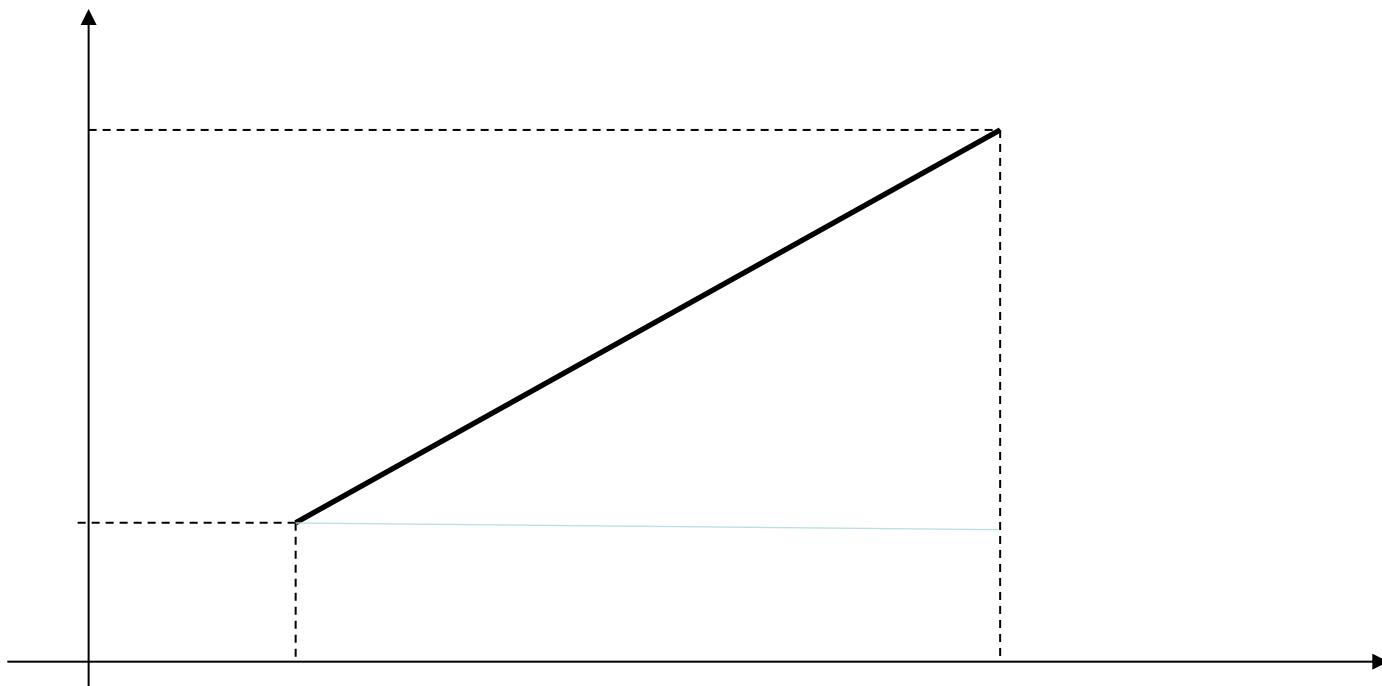
Трассировка

a	d(1)	a	d(a)	b
1	2	2	4	6

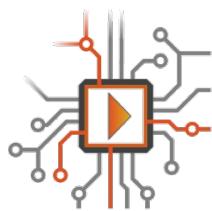
a	d(a)	a	d(1)	b
1	2	2	3	5

Пример решения задачи

Задача 1. Дано множество точек, заданных своими координатами: $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$. Найти координаты всех точек, являющихся вершинами прямоугольных треугольников.



$$L(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



Пример решения задачи

Задача 1. Дано множество точек, заданных своими координатами: $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$. Найти координаты всех точек, являющихся вершинами прямоугольных треугольников.

Постановка задачи:

Дано: n -цел, $x[0:n-1]$ -вещ, $y[0:n-1]$ -вещ

- Результат :** $\text{Res} = \{((x[i], y[i]), (x[j], y[j]), (x[k], y[k]))\}$ или
сообщение "треугольники не найдены"

При: $n \in \mathbb{N}$, $n > 2$, $n \leq \max n$.

Связь

$$L((x[i], y[i]), (x[j], y[j])) = \sqrt{((x[i]-x[j])^2 + (y[i]-y[j])^2)}$$

$$a = L((x[i], y[i]), (x[j], y[j])), b = L((x[i], y[i]), (x[k], y[k])),$$

$$c = L((x[k], y[k]), (x[j], y[j]))$$

$$\forall a \in R \quad \forall b \in R \quad \forall c \in R: a > 0 \text{ и } b > 0 \text{ и } c > 0 \text{ и } a+b > c \text{ и } \\ a+c > b \text{ и } b+c > a \quad tr(a, b, c) = true$$

$$\forall a \in R \quad \forall b \in R \quad \forall c \in R: a^2 + c^2 = b^2 \text{ или } a^2 + c^2 = b^2 \text{ или } b^2 + c^2 = a^2$$

$$pr(a, b, c) = true$$

$$\forall i=0, n-3 \quad \forall j=i+1, n-2 \quad \forall k=j+1, n-1: tr(a, b, c) = true \text{ и }$$

$$pr(a, b, c) = true \quad triangle ((x[i], y[i]), (x[j], y[j]), (x[k], y[k])) = true$$

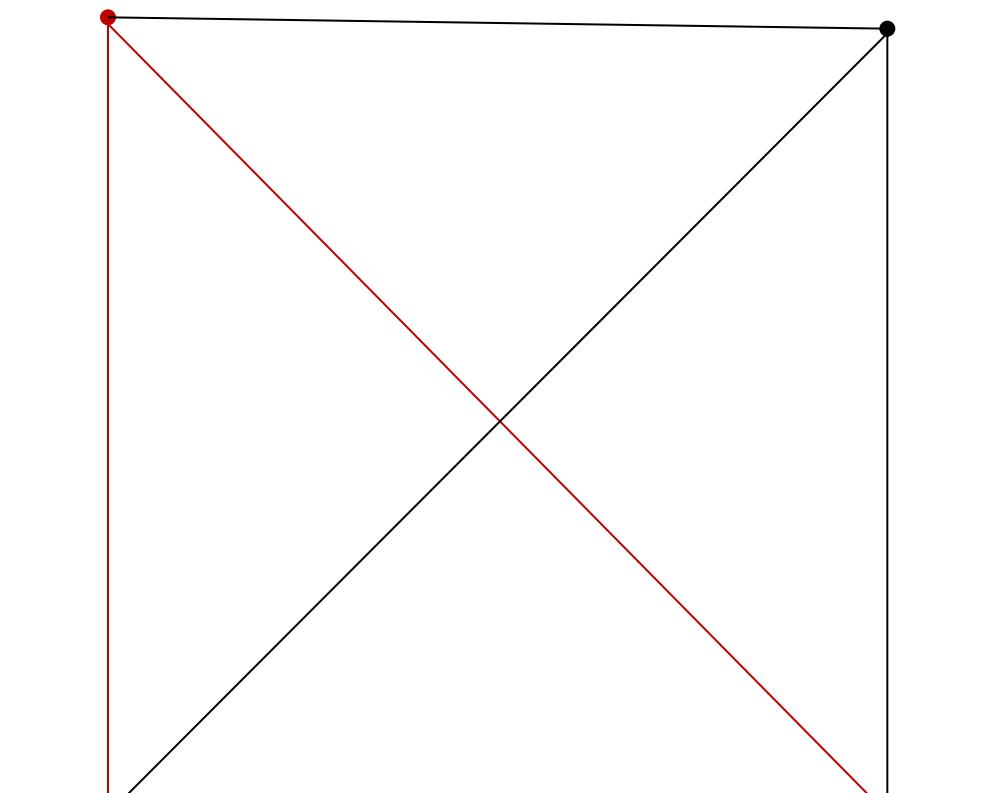
$$\forall i=0, n-3 \quad \forall j=i+1, n-2 \quad \forall k=j+1, n-1:$$

$$triangle ((x[i], y[i]), (x[j], y[j]), (x[k], y[k])) = true$$

$$((x[i], y[i]), (x[j], y[j])), (x[k], y[k])) \in Res$$

(x_1, y_1)

(x_2, y_2)



(x_3, y_3)

(x_4, y_4)

Внешняя спецификация

Поиск прямоугольных треугольников

Введите количество точек

$3 \leq n \leq \langle\langle \text{maxn} \rangle\rangle <n>$

до $n \geq 3$ и $n \leq \text{maxn}$

Введите координаты точек :

для $i=0, n-1$

$\langle x[i] \rangle \langle y[i] \rangle$

При $\text{flag}=\text{true}$

для $i=0, n-1$

найден прямоугольный треугольник с вершинами в точках:

$x_1 = \langle\langle x[i] \rangle\rangle \quad y_1 = \langle\langle y[i] \rangle\rangle$

$x_2 = \langle\langle x[j] \rangle\rangle \quad y_2 = \langle\langle y[j] \rangle\rangle$

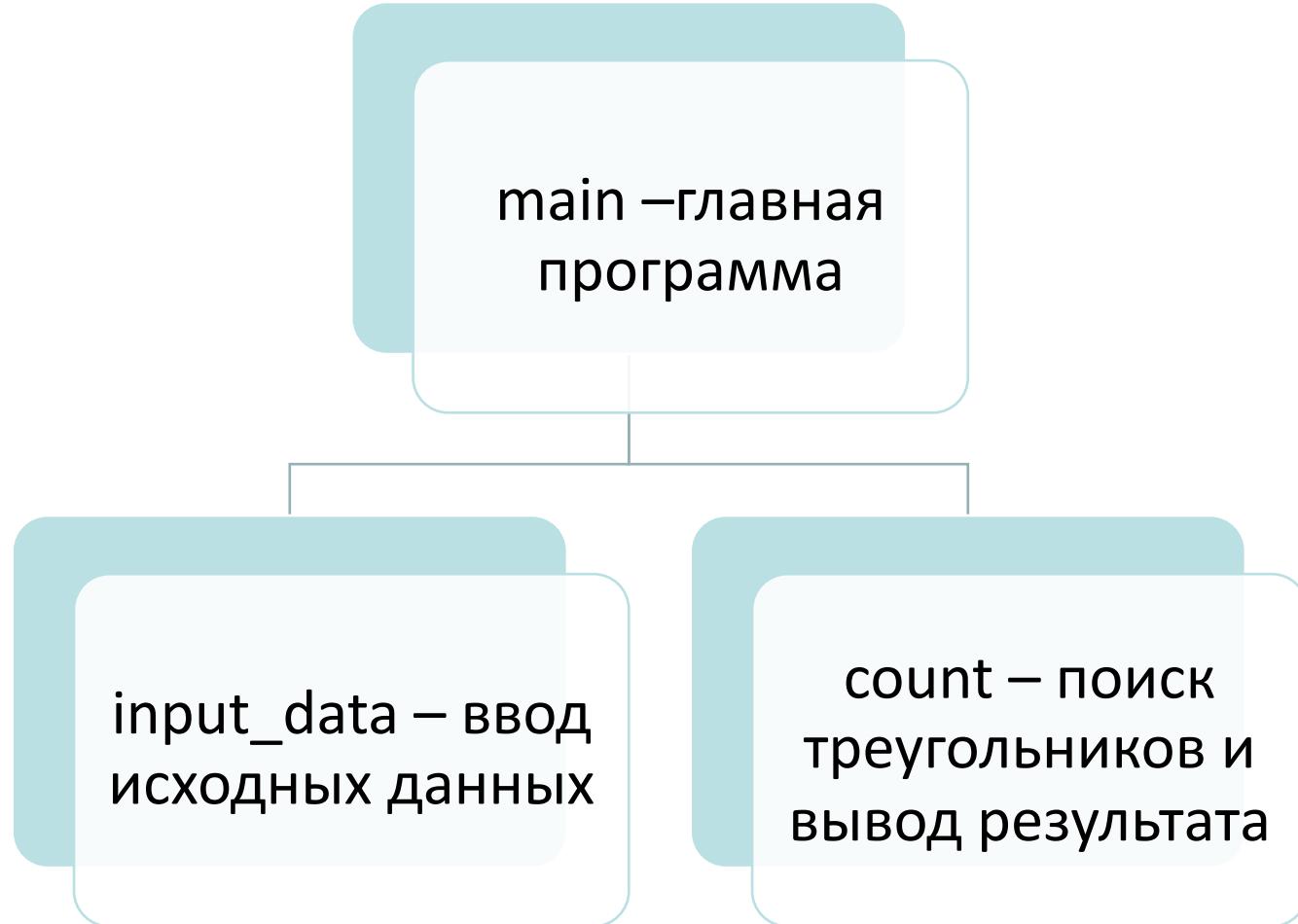
$x_2 = \langle\langle x[k] \rangle\rangle \quad y_2 = \langle\langle y[k] \rangle\rangle$

При $\text{triangle}(x, y, n, i, j, k)$

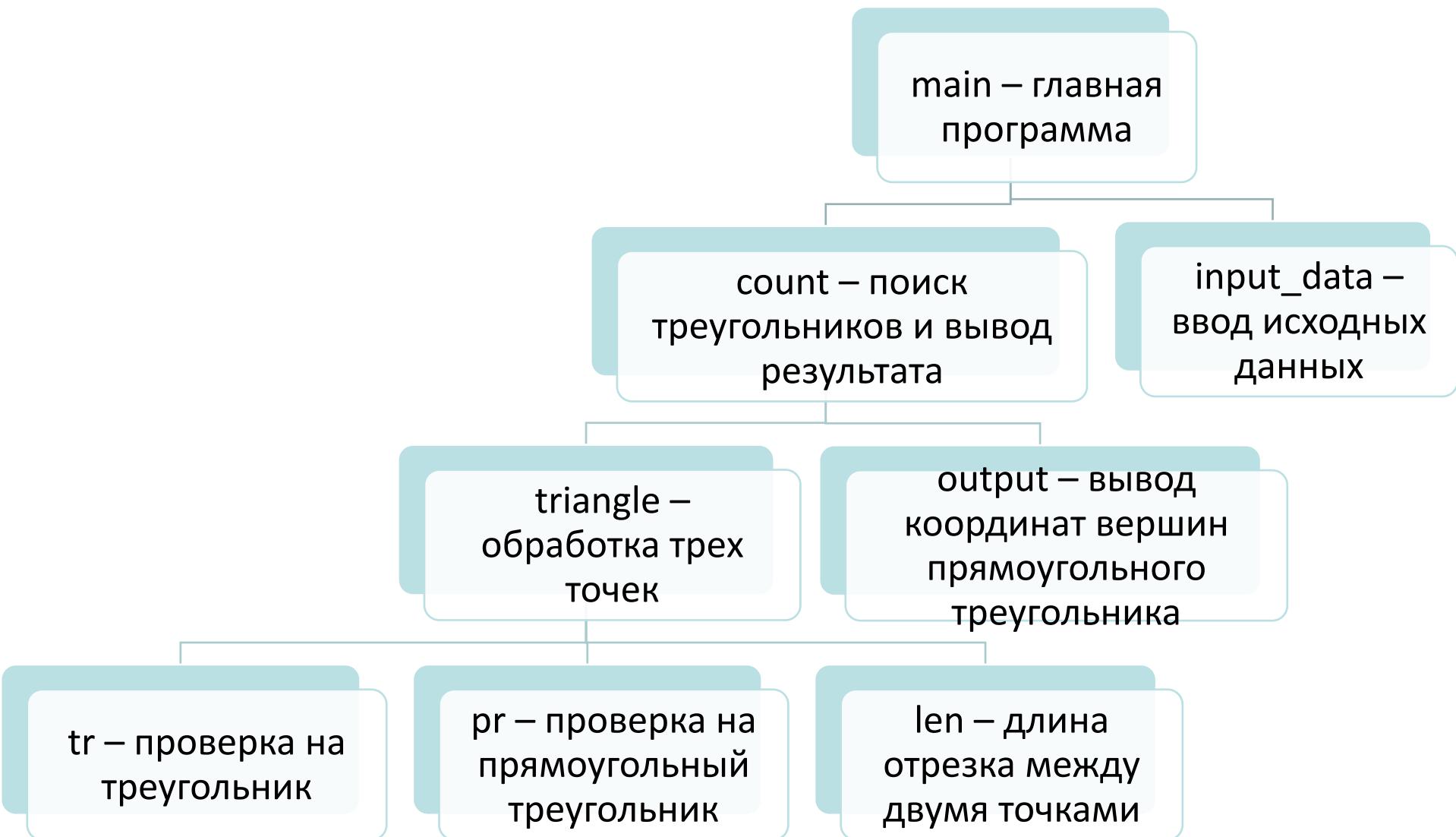
иначе

Треугольники не найдены

Схема взаимодействия подпрограмм



Дальнейшая детализация схемы



Алгоритм

алг “Прямоугольные треугольники”

нач

 ввод_данных (n, x[0:n-1], y[0:n-1])

 подсчет (n, x[0:n-1], y[0:n-1])

кон

алг “вывод_рез”

 вход: x1 –вещ., y1 –вещ., x2 –вещ., y2 –
 вещ. ., x3 –вещ., y3 –вещ.

нач

 вывод (“найден прямоугольный
 треугольник с вершинами в точках::”)

 вывод (“x1 = ”, x1, “y1 = ”, y1)

 вывод (“x2 = ”, x2, “y2 = ”, y2)

 вывод (“x3 = ”, x3, “y3 = ”, y3)

кон

алг “ввод_данных”

 выход: n – цел., x[0:n-1]-вещ., y[0:n-1] –
 вещ.

нач

 вывод (“Поиск прямоугольных
 треугольников”)

цикл

 вывод (“Ведите количество
 элементов массива 3<= n <= ”,
 maxn)

 ввод (n)

 до n > 2 и n <= maxn

кц

 вывод(“Ведите координаты
 точек:”)

цикл от i := 0 до n-1

 ввод (x[i], y[i])

кц

кон

алг “расстояние между точками”

вход : x_1 – вещ., y_1 – вещ. , x_2 – вещ. , y_2 – вещ.

выход : len – вещ.

нач

$$Len := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

кон

алг “ проверка на прямоугольный треугольник”

вход : a – вещ., b – вещ. , c – вещ.

выход : $result$ – логич.

нач

$result := false;$

если $|a^2 - b^2 + c^2| < 0.1$ или $|b^2 - a^2 + c^2| < 0.1$ или $|b^2 - c^2 + a^2| < 0.1$ то

$result := true;$

всё

кон

алг “ проверка на треугольник”

вход : а – вещ., b – вещ. , с – вещ.

выход : result – логич.

нач result := true;

если(а < 0.1) или (b < 0.1) или (c < 0.1) то result := false

всё

если |а-б+с|<0.1 или |б-а+с|<0.1 или |б-с+а| < 0.1 то

result := false всё

кон

алг “обработка трех точек”

вход x[0:n-1] – вещ., y[0:n-1] – вещ. , n – цел. , i – цел. , j – цел. , k – цел. ,

выход result – логич.

нач result := false

l1 := len(x[i], y[i], x[j], y[j]); l2 := len(x[i], y[i], x[k], y[k])

l3 := len(x[j], y[j], x[k], y[k])

если tr(l1, l2, l3) то

если (pr(l1, l2, l3) или pr(l1, l3, l2) или pr(l2, l1, l3)

или pr(l2, l3, l1) или pr(l3, l1, l2) или pr(l3, l2, l1)) то

result := true;

всё

кон

алг “перебор координат и поиск треугольников”

вход $x[0:n-1]$ – веществ., $y[0:n-1]$ – веществ. , n – цел.

выход –

нач

flag := false

цикл от $i := 0$ до $n - 3$

цикл от $j := i + 1$ до $n - 2$

цикл от $k := j + 1$ до $n - 1$

если triangle(x, y, n, i, j, k) то

flag := true;

вызов процедуры

output($x[i], y[i], x[j], y[j], x[k], y[k]$)

всё

кц

кц

если не (flag) то вывод (“треугольники не найдены”)

всё

кон

Алг “основная программа”

нач

вызов функции input_data(n, x, y); {ввод массивов координат}

вызов функции count(n, x, y); {поиск треугольников и вывод
результатов}

кон

Программа

```
#include<stdio.h>
#include<math.h>
#define maxn 20

//вывод результата
void output(float x1,float y1, float x2, float y2, float x3, float y3)
{
    printf("найден прямоугольный треугольник с вершинами в точках:\n");
    printf("x1 = %8.3f y1 = %8.3f\n",x1, y1);
    printf("x2 %8.3f y2 = %8.3f\n",x2, y2);
    printf("x3 = %8.3f y3 = %8.3f\n", x3,y3);
}
```

```
//ввод данных
```

```
void input_data(int * n, float x[], float y[] )
```

```
{
```

```
int i;
```

```
printf("Поиск прямоугольных треугольников\n");
```

```
do{
```

```
    printf("Введите количество точек 3 <= n <= %d\n", maxn);
```

```
    scanf("%d",n);
```

```
}
```

```
while (*n <3 || *n > maxn);
```

```
printf("Введите координаты точек :\n");
```

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

```
    scanf("%f%f",&x[i], &y[i] );
```

```
}
```

```
//расстояние между точками
float len(float x1, float y1, float x2, float y2)
{
    return sqrt(pow(x1 - x2,2) + pow(y1 - y2,2));
}

//проверка на треугольник
int tr(float a, float b, float c)
{
    int result = 1;
    if(a < 0.1 || b < 0.1 || c < 0.1) result = false;//длина стороны =0
    if(fabs(a - b + c) < 0.1 || fabs(b - a + c) < 0.1 || fabs(c - b + a) < 0.1) result =
0;// не выполняется неравенство треугольника
    return result;
}
```

//проверка на прямоугольный треугольник

```
int pr(float a, float b, float c)
{
    int result = 0;
    //неизвестно, какая из сторон гипотенуза, но можно проверить все //сочетания,
    //точность вычислений 0.1
    if((fabs(pow(a,2) - pow(b,2) + pow(c,2)) < 0.1) or (fabs(pow(b,2) - pow(a,2) +
    pow(c,2)) < 0.1) or (fabs(pow(b,2) - pow(c,2) + pow(a,2)) < 0.1)) result = 1;
    return result;
}
```

//обработка трех точек

```
int triangle(float x[], float y[], int n, int i, int j, int k)
{
    float l1, l2, l3;
    int result = 0;

    l1 = len(x[i], y[i], x[j], y[j]);
    l2 = len(x[i], y[i], x[k], y[k]);
    l3 = len(x[j], y[j], x[k], y[k]);
    if(tr(l1, l2, l3)) //треугольник
        if(pr(l1, l2, l3) || pr(l1, l3, l2) || pr(l2, l1, l3) || pr(l2, l3, l1) || pr(l3, l1, l2) || pr(l3, l2, l1) ) //прямоугольный
            result = 1;
    return result;
};
```

```
//перебор координат и поиск треугольников
void count(int n,float x[], float y[])
{
int i, j, k, flag;
flag = 0;
for (i = 0;i< n - 2; i++)
    for (j = i + 1;j< n - 1;j++)
        for (k = j + 1 ;k<n;k++)
            if (triangle(x, y, n, i, j, k) )
{
    flag = 1;
    output(x[i], y[i], x[j], y[j], x[k], y[k]);
}
if (!flag) printf("треугольники не найдены\n");
}
```

```
//основная программа
int main()
{
    float x[maxn],y[maxn];
    int n;

    input_data(&n, x, y); //ввод массивов координат
    count(n, x, y); //поиск треугольников и вывод результатов
    return 0;
}
```

Символьный тип данных

Тип `char` используется для представления одного символа. Значением объекта типа `char` является код, соответствующий данному символу. Значения переменных типа `char` заключаются в ‘’ (не в “ ”).

Для ввода и вывода значений этого типа используется формат `%c`. При использовании формата `%d` будет выводиться код символа.

Операции над типом char

1. Присваивание

char c;

c=' '; c='\n'; c='\\';

2. Сравнение (сравниваются коды символов).

Используются знаки ==, !=, >, <, >=, <=.

'a'<'b' - истина, 'B'=='b' - ложь

3. Арифметические

+,- изменяют код символа соответствующим образом

char c1='B', c2;

c2=c1+1; //c2='C';

c2=c1-1; //c2='A';

c2='A'+3; //c2='D';

Ввод и вывод данных типа char

В stdio.h определены функции

- int getchar (void); – ввод одного символа из входного потока;
- int putchar (int c); – вывод одного символа в выходной поток.

При успешном вводе/выводе возвращается код символа в виде целого без знака, при ошибке или конце файла - EOF.

EOF определен в stdio.h как константа (#define EOF -1).

Для генерации EOF с клавиатуры используется **Ctrl+Z**.

Ввод и эхо-печать символов:

```
int main ()  
{ int ch;  
while ((ch=getchar())!=EOF) putchar (ch);  
return 0;  
}
```

При выполнении этой программы вводимые символы помещаются в область временной памяти, называемую «буфером». Нажатие клавиши **Enter** приводит к тому, что блок символов (или один символ) становится доступным программе.

Ввод блока осуществляется быстрее и имеется возможность коррекции вводимой строки.

Рассмотрим результат работы нашей программы.

Интересно работает [Enter]

Интересно работает

Наша программа [Enter]

Наша программа

[CTRL+Z] [Enter]

В системе небуферизованного ввода при вводе символов вплоть до признака EOF все они немедленно отображаются на экране (эхопечать). Дублируются также пробелы:

ИИннттеерреесснноо

Инициализация символьных строк

1. `char name[7]={'C','a','ш','а','\0'};`
`/* остальные 2 элемента=='\0' */`
2. `char name[7]="Саша";`
`/* строковая константа - последние два элемента нулевые */`
3. `char name[]="Саша";`
`/* строковая константа - массив размером 5 (размер определяется автоматически) и в конец строки автоматически заносится нуль-символ*/`

Обработка символьных строк

Примеры использования указателя при работе со строками.

Пусть имеется описание:

```
char ch,*uc;
```

Тогда:

1. uc=name; //uc=&name[0];
2. ch=*name; //ch=name[0]; ch='C';
3. ch=*(name+1); //ch=name[1]; ch='a';
4. ch=*uc; //ch=name[0]; ch='C';
5. ch=*(++uc); //ch=name[1]; ch='a'; uc=&name[1];

6. char name[]="Саша";
7. uc=name++; //недопустимо, т.к. адрес начала массива есть
//указатель-константа

Массивы символьных строк

Массив символьных строк можно описать как двумерный массив символов, либо как одномерный массив указателей.

Двумерный массив символов (с инициализацией):

```
static char names[3][7] = { "Саша", "Маша", "Виктор"};
```



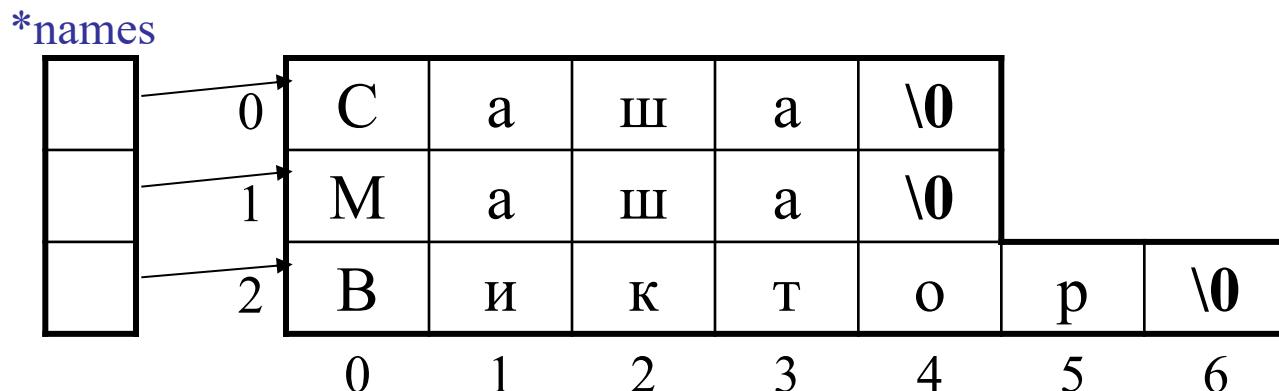
Если массив статический или внешний, то его можно инициализировать. Остальные элементы массива (не инициализированные) - нуль-символы.

```
char ch;  
ch=names[0][0]; //ch='C';  
ch=*names[1]; //ch='M';
```

Одномерный массив указателей

```
static char *names[3] = { "Саша", "Маша", "Виктор"};
```

При инициализации строки размещаются в памяти, а в массиве names запоминаются только адреса размещения. Память выделяется точно по длине строки.



Операции

```
char ch;  
ch=names[0][0]; //ch='C'  
ch=*names[1]; //ch='M'  
char *c1; c1=names[2];
```

Переменная `c1` получает значение адреса, начиная с которого в памяти хранится строка "Виктор". Сама строка не копируется.

Отличия описаний:

- `char names[3][7];`
/* выделяет $3 \times 7 = 21$ ячейку для хранения символов (1 ячейка=1 байт); */
- `char *names[3];`
/* выделяет 3 ячейки для хранения указателей на символ (1 ячейка=4 байта). */

Ввод-вывод строк

1. Функции `scanf()` и `printf()`.

Спецификация `%s` используется для чтения **слов**.

```
char name1[20],name2[10];  
int a,b;
```

входная строка: " _Маша_1_Саша"

```
scanf("%s",name1);  
printf("введенное имя:%s\n",name1);
```

name1: "Маша\0"

введённое имя:Маша

Начальные пустые символы не учитываются, считывание осуществляется до первого пробельного символа (' ', '\n', '\t').

Если хотим прочитать и "Саша":

```
b=scanf ("%s%d%s" , name1, &a, name2);
```

name1="Маша"

a=1

name2="Саша"

b=3 - число прочитанных аргументов

Замечание: Перед вводом строки необходимо выделять для неё память, например:

```
char *name;
```

```
scanf("%s", name);
```

нельзя, т.к. указатель **name** не определён. Необходимо предварительно объявить

```
char str[81];
```

Затем

```
name=str; scanf("%s", name);
```


2. Функции **gets** и **puts** - ввод строк из стандартного входного файла, вывод строк в стандартный выходной файл.

Ввод строк: gets ()

Интерфейс описан в файле `<stdio.h>`.

В отличие от `scanf("%s", str);` **gets()** выполняет ввод строк, а не слов.

Формат:

`char *gets(char *s); //возвращает указатель на тип char.`

`char name1[20];`

`gets(name1); //функция возвращает тот же указатель name1`

Функция **gets()** читает символы до тех пор, пока не встретится символ новой строки, который создается при нажатии клавиши **Enter**.
Функция читает все символы до (не включая) символа новой строки, присоединяет к ним нуль-символ и передаёт строку вызывающей программе. Затем символ новой строки пропускается.

Пусть входная строка: " _ _ Маша _ 1 _ Саша\n"

Тогда `name1`: " _ _ Маша _ 1 _ Саша\0"

Функция **gets()** получает значение, равное указателю на введённую строку. Присваивание обязательно, если использовать значение, возвращённое функцией **gets()**:

```
char *ukaz;  
ukaz= gets(name1);  
printf ("%6s%6s\n" , ukaz, name1) ;
```

```
..Masha ..Masha
```

Если ввод выполняется корректно, то функция **gets()** возвращает указатель на считанную строку. Если ввод не произошел или **gets()** встречает символ EOF, она возвращает NULL. Поэтому данная функция удобна для использования в конструкциях:

```
while (*gets(name)!=NULL);
```

Вывод строк puts ()

Функции **puts()** и **printf()** – предназначены для вывода строк. Функция **puts()** имеет только один аргумент, являющийся указателем на строку.

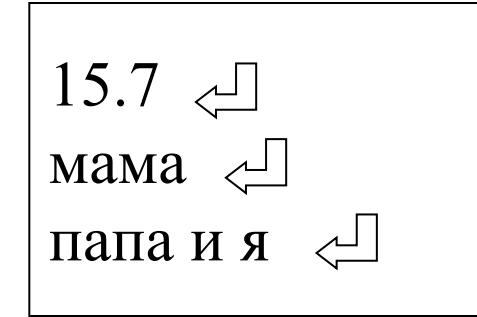
`char* puts(char* s); /* выводит символьную строку, которая определяется указателем s, и осуществляет переход на новую строку*/`

Результат: EOF (-1), если произошла ошибка, иначе – ненулевое значение. Символы переписываются в стандартный выходной поток и строка дополняется символом '\n';

```
puts(name1); // " _ _ Маша _ 1 _ Саша"  
puts ("Маша" ); // "Маша"  
puts(&name1[4]); // "ша _ 1 _ Саша"  
puts(name1); // эквивалентно printf("%s\n",name1);
```

При совместном использовании функций `gets()` и `scanf()` может потребоваться очистка входной строки. Например,

```
int main ()  
{  
float a;  
char str1[81], str2[81];
```



```
scanf("%f",&a) ;  
while (getchar()!='\n'); //очистка входного потока  
gets (str1); gets(str2) ;  
printf("a=%6.2f\n", a);  
printf("str1=%s\n", str1) ;  
printf("str2=%s\n", str2) ;  
return 0;  
}
```

Строковые функции

Строковые функции поддерживаются <string.h>.

1. Длина строки

```
int strlen(char *s);  
k=strlen("Маша"); //k=4
```

2. . Сцепление строк (конкатенация)

- char * strcat(char *s1, char *s2);

Копия второй строки присоединяется к концу первой и это объединение становится новой первой строкой. Вторая строка при этом не меняется.

Функция:

- char *strncat(char *s1, char *s2, unsigned maxlen);

Осуществляет конкатенацию не более чем первых maxlen символов второй строки.

3. Сравнение строк

- `int strcmp(char *s1, char *s2);`

Значением функции является разность между кодами первых различающихся символов в сравниваемых строках. Если строки равны, то результат 0.

```
i=strcmp("alpha", "beta"); // i=-1
```

```
i=strcmp("cccc", "a"); // i=2
```

```
i=strcmp("123", "123"); // i=0
```

- `int strncmp(char *s1, char *s2, unsigned maxlen);`

сравнивает только первые `maxlen` символов.

4. Копирование строки

- `char *strcpy(char *s1, char *s2);`

Осуществляется копирование второй строки по адресу первой.
Результат — адрес первой строки.

```
s2[ ]="Маша";
```

```
s1=s2; //копирование указателя
```

```
strcpy(s1, s2); //копирование строки
```

- `char *strncpy(char *s1, char *s2, unsigned maxlen);`

выполняет копирование только первых `maxlen` символов.

5. Вхождение символа в строку

- `char *strchr (char *str, char ch) ;`

Возвращает адрес первого появления заданного символа ch в строке str.

```
puts(strchr("Строка", 'о')); //ока
```

Если символ не найден, то возвращает NULL.

- `char *strstr(char *s1, char *s2);`

Ищет первое вхождение первой строки во вторую.

Рассмотрим несколько примеров обработки символьных строк. Считаем, что для всех примеров объявлена символьная строка и подключены библиотеки, приведенные ниже. Остальные переменные будут объявлены в примерах при вызове функций.

```
#include<stdio.h>
#include <ctype.h>
#include<string.h>
//тело функции
int main()
{ char s[81];
puts("Input one string");
gets(s);
//вызов функции и вывод ее результатов
return 0;
}
```

//число точек в начале строки

```
int kol(char*s)
{ int k=0;
for(;*s&&*s=='.';s++)k++;
return(k);
}
```

//то же, через указатель

```
int kol1(char*s)
{ char *b=s;//указатель на
//начало строки
for(;*s&&*s=='.';s++);
return(s-b);
}
```

```
int k;
k=kol(s);
if (k==0)
    puts("No points");
else
    printf("k=%d\n",k);
```

```
//число латинских букв в конце //строки
int kollet(char*s)
{char *e,*t; /*указатели на конец строки и
текущий символ*/
for(t=s;*t;t++); //т в конце строки, на '\0'
t--; //перед '\0'
/*другой вариант - через библиотечную
функцию*/
t=s+ strlen(s)-1; e=t;
while(t>=s&&isalpha(*t))t--;
//т на не букве
return(e-t); /*разность указателей –
число*/
}
```

```
//вызов
int k=kollet(s);
if (k==0)
    puts("No alpha at the
end");
else printf("k=%d\n",k);
```

```
//вставить 0 после каждой 1
int ins(char *s) /*вернет 1 если вставка
была. 0 если нет*/
{ int f=0; char *t;
while(*s)
{ if(*s=='1')
  { //освобождаем место для //вставки
    /*сдвигаем символы на 1 вправо
    начиная с конца, включая нуль-
    символ*/
  t=s+strlen(s);
  while (t>s)
    *(t+1)=*t,t--; /*t и s указывают на
один и тот же адрес*/
  *(++s)='0';
  f=1;
}
s++;
}
return f;
}
```

```
//вызов
if (!ins(s)) puts("No insert");
else
printf ("new string after
insert: %s",s);
```

```
//замена лат. букв на
//цифры - А на 0 и т.д.
int zam(char *s) /*1 - замена
была. 0- не было */
{int f=0;
for(;*s;s++)
if (*s>='A'&&*s<='J')
{f=1;*s=*s-'A'+'0';
}
return f;
}
```

```
//вызов
if(!zam(s)) puts("No
changes");
else printf("new string
after changes: %s",s);
```

```
//длина и начало самой длинной цепочки из
//единиц
void max1(char *s, char **b,int *maxlen)//указатель
    на начало и длина
{ int len; *b=NULL; //цепочки нет
*maxlen=len=0; //инициализация максимальной //и
    текущей длины
for(;*s;s++)
if (*s=='1') len++; //считаем единицы
else
{if (len>*maxlen) //текущая длина больше
    //максимальной
    {*maxlen=len;      *b=s-len; //адрес начала
     //цепочки
    }
len=0; //начало новой цепочки
}
//если цепочка в конце строки
if (len>*maxlen)
{ *maxlen=len;
  *b=s-len;
}
}
```

```
char *b;
int maxlen;
max1(s,&b,&maxlen);
/*указатель на начало
и длина*/
if(!b) //b==NULL
puts(" No 1");
/*выводим номер
начальной
позиции*/
else printf("begin at =
%d length = %d\n",b-
s,maxlen);
```

```
/* Определение первого вхождения строки str2 в строку str1,
начиная с n-ого символа str1. Если вхождение невозможno или не
установлено, то вернуть NULL.*/
#include <iostream>
#include <string.h>
#include <stdio.h>
char* search(int n, char* s1, char* s2); //прототип
int main()
{ setlocale(LC_ALL,"RUS");
    char str1[20], str2[20], *u; int n;
    printf("Введите первую строку "); gets(str1);
    printf("Введите вторую строку "); gets(str2);
    printf("Введите положение символа: ");
    scanf("%d",&n);
    u=search(n,str1,str2);
    if (!u) puts("Символ не найден");
    else printf("Адрес %0x, значение - %c, номер - %d", u, *u,u-str1);
    return 0;
}
```

```
char* search(int n, char* s1, char* s2)
{
    int L1,L2,i,j;
    int f=0;
    L1=strlen(s1);
    L2=strlen(s2);
    if (n>=0 && n<L1 && n+L2<=L1)
        for (i=n;(!f) && i<L1-L2+1;i++)
            for (j=0,f=1;j<L2 && f;j++)
                if (s2[j]!=s1[i+j])
                    f=0;
    if (f) return(s1+i-1);
    else return(NULL);
}
```


Пример. Проверка правильности ввода данных. Вводим целое положительное число. Используем форматный ввод данных.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
```

```
//проверка правильности ввода целого числа
```

```
#define CLR while (getchar()!='\n')
//макрос - пропуск символов до конца строки
```

```
int read(void)
{ char str[80];
    int n, k, f=0; //f=0 при ошибке ввода. f=1 при верном
//вводе
    do
    {printf("введите целое число\n");
        k=scanf("%s", str);
        if(k)
            {if(strcmp(str,"0")==0) n=0,f=1; //ввод нуля
             else //число не вещественное и строка не пустая
                 if(strcmp(str,"")!=0&&!strchr(str,','))
                     {char *s=str;
                      while(isdigit(*s)) // пропуск цифр
                          s++;
```

```
if(!*s) //пропустили все цифры и дошли до конца строки
    { n=atoi(str); //преобразование строки в целое число
        if(n)    f=1; } // n!=0
    }
}
while (!f); //пока не введено верное целое число
CLR; //очистка входного потока
return n;
}
```

```
int main()
{
    int n=read();
    printf("n=%d\n",n);
    return 0;
}
```

`ctype.h`—заголовочный файл стандартной библиотеки языка программирования С, содержащий объявления функций для классификации и преобразования отдельных символов.

Все перечисленные ниже функции имеют прототипы вида:

`int имя_функции(int);`

Все эти функции принимают в качестве аргумента `int` - код символа и возвращают значение типа `int`, которое может представлять или другой символ (в случае функций преобразования), или логическое значение. 0 означает «Ложь», а отличное от нуля значение— «Истина».

Функции классификации

Эти функции проверяют, является ли аргумент буквой или цифрой, пробелом или табуляцией, управляющим символом, десятичным числом, печатным символом (кроме пробела), символом в нижнем регистре, печатным символом (в том числе пробелом), пробелом, символом в верхнем регистре или шестнадцатиричным числом.

Для работы с этими функциями необходимо подключить `<ctype.h>`.

```
int isalnum(int c); /*Если аргумент функции является либо латинской буквой, либо цифрой, она возвращает ненулевое значение.*/
```

```
int isalpha(int c); /*Возвращает ненулевое значение, если её аргумент является латинской буквой, в противном случае возвращается нуль.*/
```

```
int isblank(int c); /* Возвращает true, если с - пробел или
горизонтальная табуляция. */

int iscntrl(int c); /* Возвращает true, если с - управляющий
символ, такой как <Ctrl+B>. */

int isdigit(int c); /* Возвращает ненулевое значение, если её
аргумент является десятичной цифрой, в противном случае
возвращается нуль. */

int isgraph(int c); /* Возвращает true, если с - печатаемый
символ, отличный от пробела. */

int islower(int c); /* Возвращает true, если с - символ нижнего
регистра. */

int isprint(int c); /* Возвращает true, если с — печатаемый
символ. */

int ispunct(int c); /* Возвращает true, если с - знак препинания
(любой печатаемый символ, отличный от пробела или
алфавитно-цифрового символа). */
```

```
int isspace(int c); /* Возвращает true, если с — пробельный  
символ: пробел, новая строка, возврат каретки,  
вертикальная табуляция, горизонтальная табуляция. */  
int isupper(int c); /* Возвращает true, если с - символ верхнего  
регистра. */  
int isxdigit(int c); /* Возвращает true, если с —  
шестнадцатиричная цифра. */
```

Функции преобразования

Функции преобразуют символ с в нижний или верхний регистр, если это возможно. В противном случае они возвращают неизменённое значение.

Для работы с этими функциями необходимо подключить `<ctype.h>`.

```
int toupper(int c); /* переводит латинские буквы нижнего регистра в верхний регистр. */
```

```
int tolower(int c); /* переводит латинские буквы верхнего регистра в нижний регистр. */
```

Задача. Дан массив символьных строк
(ввод производится до пустой строки).

1. Выделить из каждой строки подстроки, заключенные в круглые скобки. Вложенность скобок не учитывается.
2. Найти подстроку, содержащую максимальную последовательность из единиц, расположенных подряд.
3. Удвоить каждую цифру в исходной строке.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void search(char str[][50], int n, int *n1, char str1[][50], int num[])
{
    int i;
    char *s,*s1;
    *n1=0;
    for (i=0;i<n;i++)
    {
        s=str[i];
        while (*s)
            if ((*s)=='(')
            {
                s1=++s;
                while (*s&&(*s)!=')')
                    s++;
                *n1=*n1+1;
                str1[*n1]=*s1;
            }
    }
}
```

```
if (*s=='')&&s!=s1)
{
    num[*n1]=i;
    strncpy(str1[*n1],s1,s-s1);
    str1[*n1][s-s1]='\0';
    (*n1)++;
}
else
    s++;
}
}
```

```
int find (char str1[][50], int n1)
{
    int i, i1=-1, maxi=0;
    char *s, *s1;
    for (i=0; i<n1; i++)
    {
        s=str1[i];
        while (*s)
            if (*s=='1')
            {
                s1=s;
                while (*s && *s=='1')
                    s++;
                if (s-s1>maxi)
                {
                    maxi=s-s1;
                    i1=i;
                }
            }
        else
            s++;
    }
    return i1;
}
```

```
int doubling (char *s)
{
    int flag=0;
    char *t;
    while (*s)
        if (isdigit(*s))
        {
            flag=1;
            t=s+strlen(s);
            while (t>=s)
                *(t+1)=*t,t--;
            s=s+2;
        }
        else
            s++;
    return flag;
}
```

```
int main()
{
    char str[20][50], str1[40][50];
    int l1=0, l2, i, i1, nom[40];
    puts("Enter array of strings");
    while(l1<20&&*gets(str[l1])) l1++;
    if(!l1)
        puts("Empty array!");
    else
    {
        search(str,l1,&l2,str1,nom);
        if(!l2)
            puts("No substrings");
        else
        {
            puts("Substrings found:");
            for(i=0;i<l2;i++)
                puts(str1[i]);
            i1=find(str1,l2);
            if(i1==-1)
                puts("No '\'1\' in substrings");
        }
    }
}
```

```
else
{
    puts("Substring found:");
    puts(str1[i1]);
    puts(str[nom[i1]]);
    puts("Will be changed");
    if(doubling(str[nom[i1]]))
    {
        puts("Changed string");
        puts(str[nom[i1]]);
    }
    else
        puts("String was not changed");
}
}
return 0;
}
```

```
#include <stdio.h>
#include <iostream>

int main()
{ puts("System default locale\n");
  for (int i=128; i<256; i++) printf("%c %d | ",i,i);
  printf("\n\n\n");
  setlocale(LC_ALL,".1251");
  puts("Locale 1251\n");
  for(int i=128; i<256; i++)
  {
    if (i==149) printf(" ");
    printf("%c %d | ",i,i);
  }
  return 0;
}
```

System default locale

А 128	Б 129	В 130	Г 131	Д 132	Е 133	Ж 134	З 135	И 136	И 137
К 138	Л 139	М 140	Н 141	О 142	П 143	Р 144	С 145	Т 146	Ү 147
Ф 148	Х 149	Ц 150	Ч 151	Ш 152	Щ 153	Ь 154	Һ 155	Ь 156	Э 157
Ю 158	Я 159	а 160	е 161	в 162	г 163	д 164	е 165	ж 166	з 167
и 168	й 169	к 170	л 171	м 172	н 173	օ 174	ո 175	ղ 176	ղ 177
՚ 178	՚ 179	՚ 180	՚ 181	՚ 182	՚ 183	՚ 184	՚ 185	՚ 186	՚ 187
՚ 188	՚ 189	՚ 190	՚ 191	՚ 192	՚ 193	՚ 194	՚ 195	՚ 196	՚ 197
՚ 198	՚ 199	՚ 200	՚ 201	՚ 202	՚ 203	՚ 204	՚ 205	՚ 206	՚ 207
՚ 208	՚ 209	՚ 210	՚ 211	՚ 212	՚ 213	՚ 214	՚ 215	՚ 216	՚ 217
՚ 218	՚ 219	՚ 220	՚ 221	՚ 222	՚ 223	՚ 224	՚ 225	՚ 226	՚ 227
՚ 228	՚ 229	՚ 230	՚ 231	՚ 232	՚ 233	՚ 234	՚ 235	՚ 236	՚ 237
՚ 238	՚ 239	՚ 240	՚ 241	՚ 242	՚ 243	՚ 244	՚ 245	՚ 246	՚ 247
՚ 248	՚ 249	՚ 250	՚ 251	՚ 252	՚ 253	՚ 254	՚ 255		

Locale 1251

՞ 128	՞ 129	՞ 130	՞ 131	՞ 132	՞ 133	՞ 134	՞ 135	՞ 136	՞ 137
՞ 138	՞ 139	՞ 140	՞ 141	՞ 142	՞ 143	՞ 144	՞ 145	՞ 146	՞ 147
՞ 148	՞ 149	՞ 150	՞ 151	՞ 152	՞ 153	՞ 154	՞ 155	՞ 156	՞ 157
՞ 158	՞ 159	՞ 160	՞ 161	՞ 162	՞ 163	՞ 164	՞ 165	՞ 166	՞ 167
՞ 168	՞ 169	՞ 170	՞ 171	՞ 172	՞ 173	՞ 174	՞ 175	՞ 176	՞ 177
՞ 178	՞ 179	՞ 180	՞ 181	՞ 182	՞ 183	՞ 184	՞ 185	՞ 186	՞ 187
՞ 188	՞ 189	՞ 190	՞ 191	՞ 192	՞ 193	՞ 194	՞ 195	՞ 196	՞ 197
՞ 198	՞ 199	՞ 200	՞ 201	՞ 202	՞ 203	՞ 204	՞ 205	՞ 206	՞ 207
՞ 208	՞ 209	՞ 210	՞ 211	՞ 212	՞ 213	՞ 214	՞ 215	՞ 216	՞ 217
՞ 218	՞ 219	՞ 220	՞ 221	՞ 222	՞ 223	՞ 224	՞ 225	՞ 226	՞ 227
՞ 228	՞ 229	՞ 230	՞ 231	՞ 232	՞ 233	՞ 234	՞ 235	՞ 236	՞ 237
՞ 238	՞ 239	՞ 240	՞ 241	՞ 242	՞ 243	՞ 244	՞ 245	՞ 246	՞ 247
՞ 248	՞ 249	՞ 250	՞ 251	՞ 252	՞ 253	՞ 254	՞ 255		

Задача 1. Даны символьная строка, в которой слова разделены одним или несколькими пробелами.

Оставить по одному пробелу между словами и сформировать массив, содержащий все различные слова полученной строки по одному разу.

При помощи первой функции удаляем пробелы.

Во второй функции формируем массив различных слов.

Выводим их на экран в главной функции.

```
#include <stdio.h>
#include <string.h>
int delblanc(char*s)
{char *t; int f=0;
while(*(s+1))
if (*s==' '&&*(s+1)==' ')
{for (t=s;*t;t++)
*t=*(t+1);
*t=*(t+1); //копируем конец
f=1;
}
else s++;
return f;
}
```

```
int slova(char *s,char sub[][10]) /*вернет число различных слов*/
{
    int i,k=0; char *s1,sl[10];
    while(*s)
        if (*s!=' ') { s1=s;
            while (*s&&*s!=' ') s++; /*указатель на пробеле после слова*/
            strncpy(sl,s1,s-s1); //кандидат в новые слова
            sl[s-s1]='\0'; /*записываем конец для сравнения слова с другими*/
            for (i=0;i<k&&strcmp(sl,sub[i])!=0;i++);
                //проверяем отсутствие слова в массиве
            if (i==k) //его нет
                strcpy(sub[k++],sl); //записываем слово в массив
            }else s++;
        }
    return(k);
}
```

```
int main()
{char s[81],sub[10][10];
int i,k,f;
    puts("enter string");  gets(s);
    f=delblanc(s);
if (!f) puts ("No changes");
else { puts("new string");  puts(s);}
k= slova(s,sub);
if (k==0)    puts("no words");
else
{ puts("Words");
for (i=0;i<k;i++)  puts(sub[i]);
}
return 0;
}
```

Задача 2. Дан массив символьных строк (ввод осуществляется до тех пор, пока не будет введена пустая строка). Заменить цифровые подстроки, заключённые в круглые скобки, звёздочками, число которых равно первой цифре в заменяемой подстроке.

```
#include <stdio.h>
#include <ctype.h> /* Содержит прототипы функций для
проверки символов */

/*максимальное число строк и максимальная длина строки*/
#define ML 81
#define MK 20

int changestr(char *s); //прототип функции
```

```
int main()
{
char s[MK][ML];
    int n=0, i;
    char *ua;
    printf("Введите строки\n");
    while (n<MK&&(ua=gets(s[n]))&&*ua!='\0') n++;
    if (!n) printf("Массив пуст\n");
    else{
        printf("Измененные строки\n");
        for (i=0;i<n;i++){
            if(change(s[i])) puts(s[i]);
            else printf("Строка под номером %d не изменила
сь\n",i);
        }
    }
    return 0;
}
```

```
int changestr(char *s)
{char *s1, *s2; int n, i ,f=0;
 for (; *s; s++)
 if (*s=='(')
 { for (s1=++s; isdigit(*s); s++)
      if (*s==')' && s1!=s) /*Замена цифровой подстроки */
 { n=*s1-'0'; /* Вычисляемое значение типа int */
   for (s2=s1; *s; *s2++=*s++);
     /* Удаление подстроки из цифр
 */
   *s2='\0'; f=1;
   for (; s1<=s2; s2--) *(s2+n)=*s2; /* Сдвиг вправо для n
 звёздочек */
   for (i=0; i<n; i++) /* Вставка n звёздочек */
     *s1++='*';
   s=s1;
 }
 } return f;
}
```

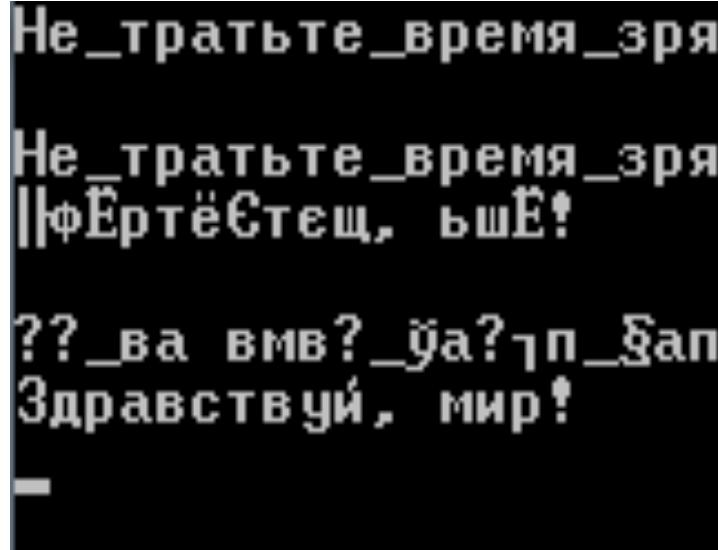


```
#include <iostream>
using namespace std;

int main ()
{
char sInput[200] = "Здравствуй, мир!";
char sOutput[200];

cin >> sOutput;
cout << endl << sOutput << endl;
cout << sInput << endl;
setlocale(LC_ALL,".1251"); // ru - russian
cout << endl << sOutput << endl;
cout << sInput << endl;
return 0;
}
```

В дальнейшем рекомендуется для вывода сообщений использовать английский язык.



```
Не_тратьте_время_зря
Не_тратьте_время_зря |ФЕРТЁСТЕЩ, ЬШЁ!
??_ва_вмв?_уа?_п_зап
Здравствуй, мир!
```

```
#include <iostream>
#include <windows.h>
using namespace std;

int main ()
{
    setlocale(LC_ALL, ".1251");
    //для использования кириллицы (из библиотеки <iostream>)
    SetConsoleCP(1251);
    //для потокового вывода с использованием русского языка
    SetConsoleOutputCP(1251);
    char sInput[200] = "Здравствуй, мир!";
    char sOutput[200];

    cin >> sOutput;
    cout << endl << sOutput << endl;
    cout << sInput << endl;
    return 0;
}
```

Необходимо выбрать шрифт Lucida console

Написать программу, которая в заданной символьной строке удаляет путём сдвига все подстроки из латинских букв, заключённые в круглые скобки и имеющие длину более 5 символов. Скобки тоже удалить.

Тестовый пример: строка

абв(84abc))абсабв(tptptp)абв))((абсЮgh))(+

Преобразуется в

абв(84abc))абсабвабв))((абсЮgh))(+

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int newstr(char* s);
int main()
{char str[181];    int f;
 printf("Введите строку \n");
 gets(str);
 f=newstr(str);
 if (f)
 {  puts("Измененная строка ");
    puts(str);
 }
 else
    printf("Не было изменений ");
 return 0;
}
```

```
int newstr(char* s)
{  char *u=s,*un,*u1;  int flag=0;
   while (*u)
   {    if (*u=='(')
        {      un=u;
          u1=u+1;
          while (isalpha(*u1)&&*u1)           u1++;
          if (*u1==')'&&(u1-un-1>5))
          { flag=1;
            u1++;
            while (*u1)
              *un++=*u1++;
            *un='\0';
          }
        else u++;
      }
    else u++;
  }
  return(flag);
}
```

Написать программу, которая в заданной символьной строке сортирует цифры по возрастанию. Порядок остальных символов не изменяется.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int newstr(char* s);
int main()
{char str[181];    int f;
 printf("Введите строку \n");
 gets(str);
 f=newstr(str);
 if (f)
 {  puts("Измененная строка ");
    puts(str);
 }
 else
    printf("Не было изменений ");
 return 0;
}
```

```
int newstr(char* s)
{char*i,*j,k;
int f=0;
for(i=s;i<s+strlen(s)-1;i++)
if (isdigit(*i))
    for(j=i+1;j<s+strlen(s);j++)
        if(isdigit(*j)&&*i>*j)
            {k=*i;*i=*j;*j=k;
             f=1;      }
return f;
}
```

Написать программу, которая сортирует массив строк в порядке возрастания их длин. Используем метод пузырька.

```
/* сортируем строки */
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#define SIZE 81 /*максимальная длина строки */
```

```
#define LIM 20 /*максимальное число строк */
```

```
void stsrt(char strings[][81],int num);
```

```
//прототип функции сортировки
```

```
int main( )
{ char input[LIM][SIZE]; /* массив строк */
int ct = 0; /* счетчик строк */
int k; /* переменная цикла */
printf("Введите строки, конец ввода – пустая строка \n");
while (*gets(input[ct]) && *input[ct]&&ct<LIM) ct++;
puts(" Исходный массив ");
for(k = 0; k < ct; k++)
    puts(input[k]);
stsrt(input, ct); /* вызов функции сортировки*/
puts(" Массив после сортировки");
for(k = 0; k < ct; k++)
    puts(input[k]);
return 0;
}
```

```
/* сортировка строк */
void stsrt(char strings[][81],int num)
{ char temp[81]; //подстрока для перестановки
int i, f;
do
{f=1;
for(i = 0; i < num-1; i++)
if(strlen(strings[i])> strlen(strings[i+1]))
{ f=0;
strcpy(temp , strings [i]);
strcpy(strings [i], strings [i+1]);
strcpy(strings [i+1], temp);
}
while (!f);
}
```

Написать программу, которая сортирует массив строк в порядке возрастания кодов. Для перестановки использовать указатели на строки.

```
/*сортировка строк с использованием массива указателей */  
  
#include <stdio.h>  
#include <string.h>  
  
#define SIZE 81 /*максимальная длина строки */  
#define LIM 20 /*максимальное число строк */  
  
void stsrt(char *strings[],int num); //прототип функции сортировки  
int main( )  
{  
    static char input[LIM][SIZE]; /*массив строк */  
    char *ptstr[LIM]; /* массив указателей на строки */  
    int ct = 0; /* число строк */  
    int k; /* переменная цикла*/  
  
    printf(" Введите строки, конец ввода – пустая строка");  
    while (*gets(input[ct]) && *input[ct]&&ct<LIM) ct++;
```

```
puts("Исходный массив строк \n ");
for(k = 0; k < ct; k++)
{
    puts(input[k]);
    ptstr[k] = input[k]; /*запоминаем адрес начала строки */
}
stsrt(ptstr, ct); /* сортировка */
puts(«Массив после сортировки \n »);
for(k = 0; k < ct; k++)
    puts(ptstr[k]); /*выводится в порядке адресов */
return 0;
}
```

```
/* сортировка установкой */  
  
void stsrt(char *strings[],int num)  
{ char *temp;  
 int i, j;  
  
 for(i = 0; i < num-1; i++)  
     for(j = i + 1; j < num; j++)  
         if(strcmp(strings[i], strings[j]) > 0) // перестановка указателей  
             { temp = strings [i];  
              strings [i] = strings [j];  
              strings [j] = temp;  
             }  
 }
```


Структуры

Структура как и массив, относится к составным типам данных. Это объект, состоящий из нескольких элементов (аналог в Паскале — запись, состоящая из полей). Элементами структуры могут являться объекты любого типа, в том числе массивы, структуры.

Объявление структурного типа

Информация о книге (указаны типы и названия полей):

author	name	year	price
Автор	Название	Год	Цена
строка	строка	число	число

```
struct book
{
    char author[20], name[60];
    int year, price;
};
```

Тип может быть локальным (внутри функции) или глобальным (вне функции). Описание типа не вызывает выделения памяти.

Определение структурных переменных

Определим структурную переменную kn1:

```
struct book kn1;  
// book - тип переменной; kn1 — имя переменной;  
stuct book *uk_book;  
/*uk_book - указатель на структуру book
```

Определение структурной переменной без предварительного объявления типа:

```
struct numbers  
{  
int a; float b;  
} z1, z2;  
  
//z1, z2 - две переменных типа struct numbers;
```

Принадлежность к внешнему типу определяется местом объявления структурной переменной, а не типа.

```
struct book kn1={"Иванов Н.И.", "физика", 2009, 175}; //инициализация
```

Для доступа к элементам структуры используются составные имена:
<имя переменной>.<имя элемента>

kn1.author //имя символьного массива[20] – адрес

kn1.name //имя символьного массива[60]

kn1.year, kn1.price //имена переменных типа int

&kn1.price //адрес

Эти имена могут быть использованы так же, как и обычные имена переменных этого же типа, например:

```
kn1.author[6] = 'a';
```

```
scanf("%d", &kn1.price);
```

```
kn1.year = 2013 ;
```

```
* (kn1. author+1) = 'p' ;
```

```
gets(kn1.name);
```

Массивы структур

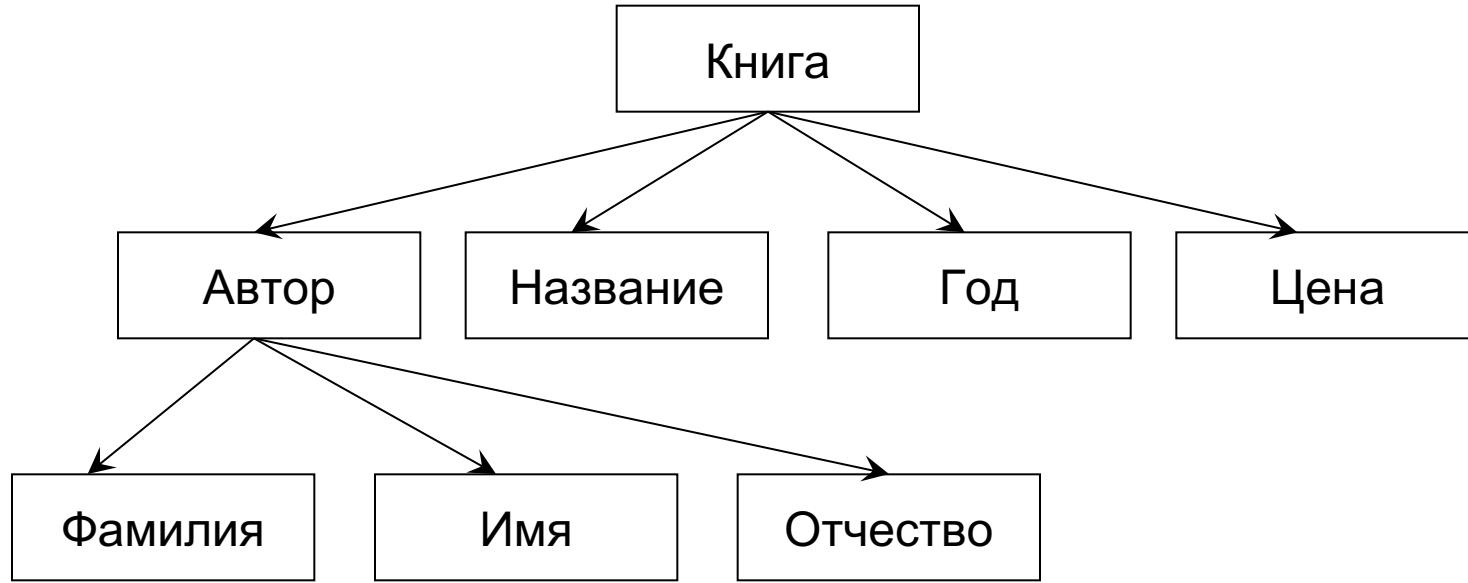
```
struct book kns[3]; // массив из трех структур  
kns[0].author //указатель на массив;  
kns[1].year //второй элемент первой структуры;  
kns[2].name[0] //нулевой символ названия.
```

Номер элемента массива всегда указывается после имени.

	Автор	Название	Год	Цена
0	[REDACTED]			
1	[REDACTED]			
2	[REDACTED]			

Вложенные структуры

Элементом структуры может быть другая структура, например



```
struct author
{
    char fam[15];
    char im[10];
    char ot[15];
};
```

```
struct book1
{
    struct author avt; /* Вложенная структура */
    char name[60];
    int year, price;
};
```

Указатели на структуры

Использование указателей на структуры удобно по трем причинам:

- так же, как и указатели на массивы, они проще в использовании, чем сами массивы;
- во многих способах представления данных используются структуры, содержащие указатели на другие структуры;
- указатель на структуру удобно передавать в функцию.

Определение переменной и инициализация

```
/* инициализация */  
struct book1 kn09 = { {"Иван", "Иванович", "Иванов"}, "физика", 2009, 175  
};  
Доступ: kn09.avt.fam[1]='р'; //Иранов
```

```
struct author avt1;  
struct author *ukavt;  
struct book1 *ukkanig; /* объявление двух указателей на структуры */
```

Все операции с указателями, используемые для обычных переменных, можно применять и к структурным переменным, например

```
ukavt=&avt1;  
ukkanig=&kn09;
```

Имя структуры - это не адрес!

```
struct book1 kni[100]; //массив структур  
ukkanig=kni;    ⇔    ukkanig=&kni[0];  
ukkanig=kni+1;   ⇔    ukkanig=&kni[1];
```

Прибавление "1" к указателю увеличивает его значение (адрес) на число байтов, которое занимает соответствующий тип.

```
ukkanig++;    ⇔    ukkanig=&kni[2]; *ukkanig=kni[2];
```

Операции над структурами

Присваивание: struct book kn1,kn2; kn2=kn1;

Сравнение структур - сравниваются поля.

Доступ к элементу структуры выполняется при помощи указателя:

```
struct avtor *ukavt, avt1;  
//ukavt - указатель на структуру,  
//avt1 - переменная структурного типа  
ukavt=&avt1; //указатель ссылается на avt1  
avt1.im [0] ='B';
```

С помощью указателя ukavt можно получить доступ к элементу одним из двух способов:

1. Операция присоединения

```
ukavt=&avt1; ukavt->im[0]= 'a';
```

Структурный указатель, за которым следует операция ->, работает так же, как имя структуры с последующей операцией «точка». Нельзя записать ukavt. im [0] т.к. ukavt - не является именем структуры.

ukavt - указатель,

ukavt->im [0] - элемент структуры, на которую настроен указатель (элемент имеет тип char).

2. Составное имя

```
(*ukavt ).im[0]= 'E';
```

Круглые скобки необходимы, т.к. операция "." имеет более высокий приоритет, чем "*".

Содержимое по адресу ukavt:

```
ukavt=&avt1; *ukavt=*(&avt1);
```

```
//т.е. *ukavt=avt1;
```

Передача информации о структурах в функцию

Структуру можно использовать в качестве формального параметра функции.

1. Можно передавать в качестве фактического параметра элемент структуры или его адрес.

```
#include <stdio.h>
```

```
struct pr
{
    int a;
    float b;
};
```

```
int ab (int a, float *b)
{
    /* присвоение значения */
    *b=2.5*(float)a;
    return (2*a);
}
```

```
int main()
{
    int c; struct pr pr1;
    scanf ("%d", &pr1.a);
    /* адрес элемента */
    c=ab(pr1.a, &(pr1.b));
    printf("result= %d b=%10.5f\n",c,pr1.b);
    return 0;
}
```

2. Можно использовать адрес структуры в качестве фактического параметра.

Объявим шаблон и переменные одновременно в задаче определения остатка от деления целых чисел.

```
struct sab
{ int a,b; } pr1 = {23,3};

int ab1(struct sab *prim) //т.е. prim - указатель на структуру pr1
{
    return (prim->a%prim->b); //остаток от деления
}
```

```
int main()
{
printf("Result=%d", ab1(&pr1) );
return 0;
}
```

3. Можно использовать имя массива структур в качестве фактического параметра (то есть адрес первого элемента массива).

```
#include <stdio.h>

/* найдём сумму всех элементов
всех структур*/
struct pr
{
    int a; float b;
};

//глобальная

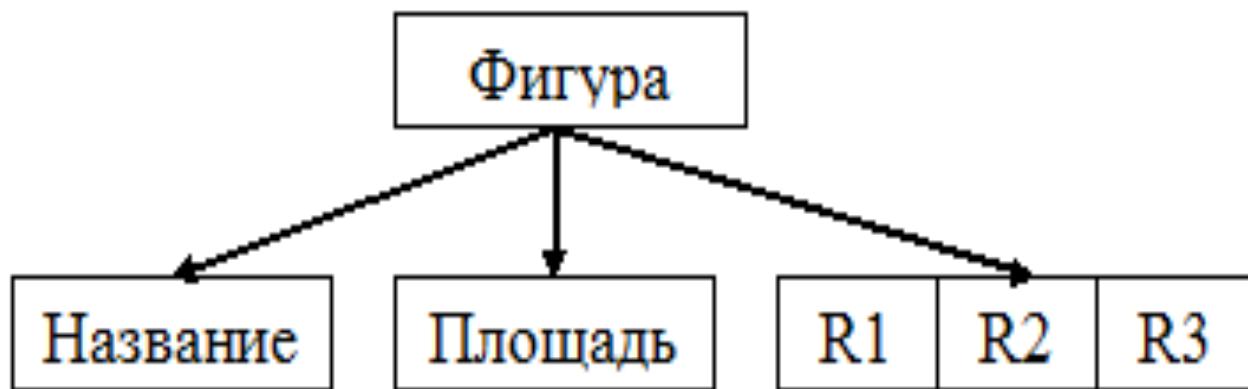
int main()
{ //в массиве 10 структур
int i;
float ab2(struct pr *prim);
/* прототип функции*/
struct pr pr2[10];
```

```
puts("Input twenty numbers");
for(i=0; i<10; i++)
    scanf("%d %f",&pr2[i].a, &pr2[i].b);
printf( "sum=%f", ab2 (pr2) );
/* pr2 — имя массива структур, адрес
первой структуры. */
return 0;
}

float ab2(struct pr *prim)
{
    int i;
    float sum=0;
    for(i=0; i<10; i++, prim++)
        sum+=prim->a+prim->b;
    return (sum);
}
```

Задача 1. Дан массив структур с информацией о геометрических фигурах.

Подсчитать число треугольников, найти треугольник с максимальной площадью (его номер) и треугольник с максимальной стороной.



```
#include <string.h>
#include <math.h>
#include <stdio.h>
struct figura /* структурный тип*/
{
    char name[15];
    float sq, r[3];
};
int treug(struct figura fig[], int kfig, int *np, int *ns, float *maxp, float *maxs);
//треугольник с максимальной площадью и стороной
float square(float r[]); //площадь треугольника
int triangle(float r[]); //проверка фигуры на треугольник
int main()
{struct figura fi[100];
    int kfi, i, nsq, nst, kt;
    float maxsq, maxst;
```

```
printf("Количество фигур = "); scanf("%d", &kfi);
printf("Введите параметры фигур в установленном порядке:\n");
printf("сторона1 сторона2 сторона3\n");
for (i=0; i<kfi; i++)
{strcpy(fi[i].name,"");
 fi[i].sq=0;
 scanf("%f %f %f", &fi[i].r[0], &fi[i].r[1],&fi[i].r[2]);
 if(triangle(fi[i].r))
 { strcpy(fi[i].name,"triangle");
   fi[i].sq=square(fi[i].r);
 }
}
printf("Число треугольников = %d\n", kt=treug(fi, kfi, &nsq, &nst, &maxsq,
 &maxst));
if (kt)
{
printf("Максимальная площадь = %f; номер = %d\n", maxsq, nsq);
printf("Максимальная сторона = %f; номер = %d\n", maxst, nst);
}
return 0;
}
```

```
float square(float r[])
{
    float p, s;
    p=(r[0]+r[1]+r[2])/2.0;
    s=sqrt(p*(p-r[0])*(p-r[1])*(p-r[2]));
    return s;
}

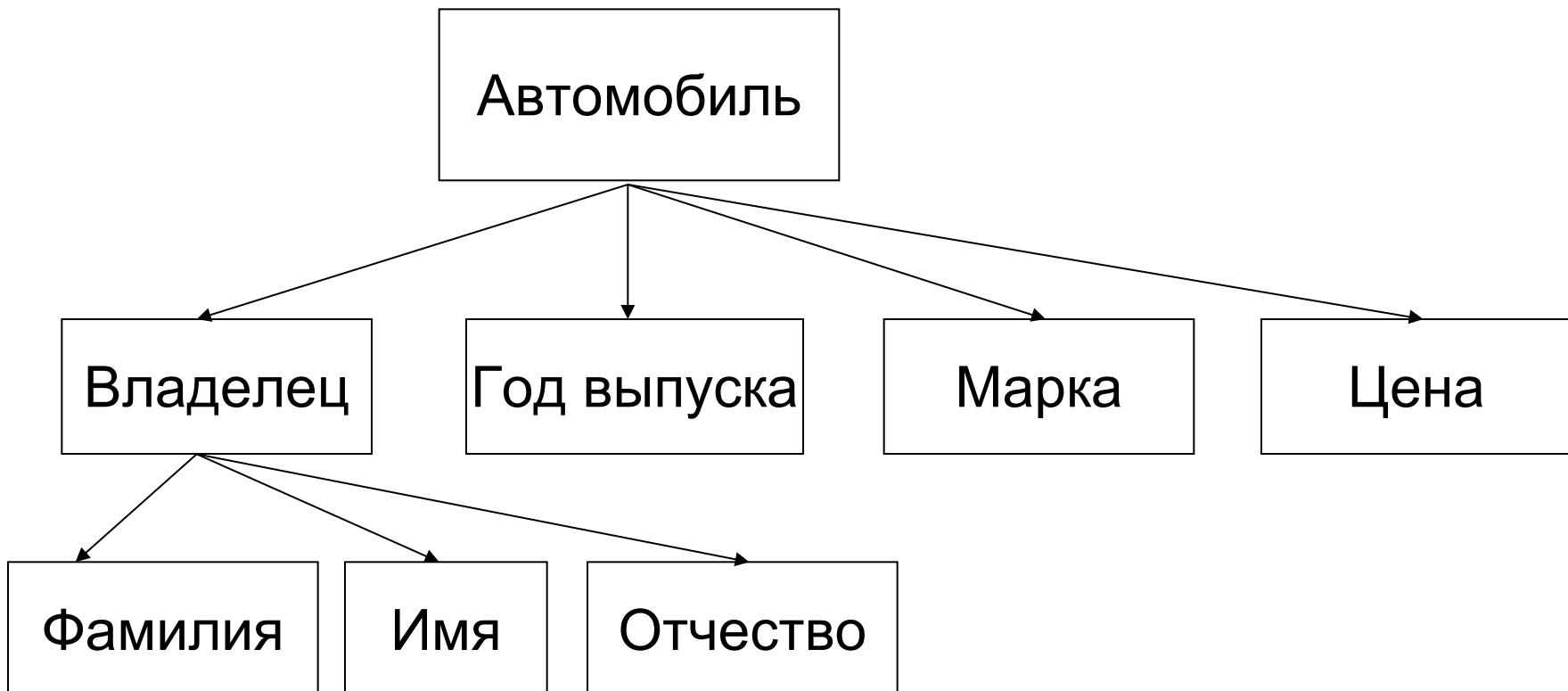
int triangle(float r[])
{
    int exist=1;
    if (r[0]<=0 || r[1]<=0 || r[2]<=0)
        exist=0;
    else
        if(r[0]>=r[1]+r[2] || r[1]>=r[2]+r[0] || r[2]>=r[1]+r[0])
            exist=0;
    return exist;
}
```

```
int treug(struct figura fig[], int kfig, int *np, int *ns, float *maxp, float
          *maxs)
{
    int i, j, kt;

    *maxp=*maxs=kt=0;
    for (i=0; i<kfig; i++, fig++)
        if (!strcmp(fig->name, "triangle"))
            {kt++;
             if (fig->sq>*maxp)          *np=kt-1, *maxp=fig->sq;
              for (j=0; j<3; j++)
                  if (fig->r[j]>*maxs) *ns=kt-1, *maxs=fig->r[j];

            }
    return (kt);
}
```

Задача 1. Ввести в массив данные о структурах (количество структур известно). Считаем, что полных тезок среди владельцев нет.



- Ввести данные с клавиатуры, проверив их допустимость.
- Найти различных владельцев автомобилей с ценой менее 500 000 р.
- Найти марки с максимальным числом выпущенных автомобилей.
- Найти автомобили с минимальной ценой (сформировать массив указателей на них).
- Найти годы выпуска, в которых было произведено максимальное число различных марок автомобилей.

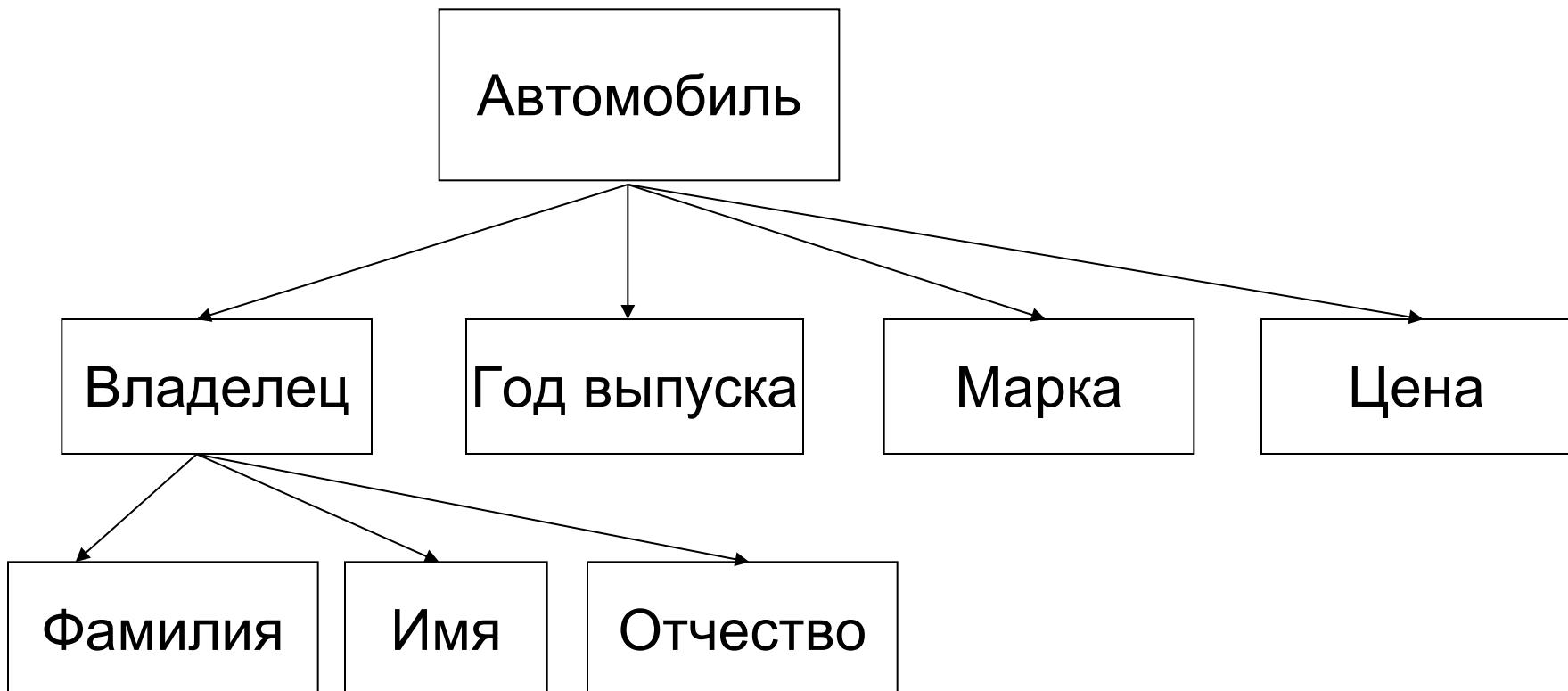
```
#include <iostream>
#include <stdio.h>
#include "time.h"
#include <string.h>
#include <math.h>
using namespace std;
#define lmax 100
#define CLR while (getchar() != '\n')
// структурный тип для владельца
struct own {
    char surname[lmax];
    char name[lmax];
    char g_name[lmax];
};
```

```
//структурный тип для автомобиля
struct auto_ {
    struct own id;
    int year;
    char model[lmax];
    int price;
};

//проверка правильности введенного года
//вернет 1 если год в границах от 1886 до текущего года
int check_time(int year_pr) {
    int m = 0;

    time_t t;
    tm *aTm = localtime(&t);
    int te = aTm -> tm_year + 1900;
    if (te >= year_pr && year_pr >= 1886)
        m = 1;
    return m;
}
```


Задача 1. Ввести в массив данные о структурах (количество структур известно). Считаем, что полных тезок среди владельцев нет.



- Ввести данные с клавиатуры, проверив их допустимость.
- Найти различных владельцев автомобилей с ценой менее 500 000 р.
- Найти марки с максимальным числом выпущенных автомобилей.
- Найти автомобили с минимальной ценой (сформировать массив указателей на них).
- Найти годы выпуска, в которых было произведено максимальное число различных марок автомобилей.

```
#include <iostream>
#include <stdio.h>
#include "time.h"
#include <string.h>
#include <math.h>
using namespace std;
#define lmax 100
#define CLR while (getchar() != '\n')
// структурный тип для владельца
struct own {
    char surname[lmax];
    char name[lmax];
    char g_name[lmax];
};
```

```
//структурный тип для автомобиля
struct auto_ {
    struct own id;
    int year;
    char model[lmax];
    int price;
};

//проверка правильности введенного года
//вернет 1 если год в границах от 1886 до текущего года
int check_time(int year_pr) {
    int m = 0;

    time_t t;
    tm *aTm = localtime(&t);
    int te = aTm -> tm_year + 1900;
    if (te >= year_pr && year_pr >= 1886)
        m = 1;
    return m;
}
```

```
//ввод массива автомобилей
```

```
void input(struct auto_ at[], int *k) {
```

```
    int ret_code;
```

```
    float check_y;
```

```
//ввод длины массива
```

```
do {
```

```
    printf("Number of structures:\n");
```

```
    ret_code = scanf("%d", k);
```

```
    CLR;
```

```
}
```

```
while (ret_code != 1 || *k <=0 || *k >lmax);
```

```
for (int i = 0; i < *k; i++) {
```

```
    printf(" Car # %d\n", i+1 );
```

```
    printf(" Surname: " );
```

```
    gets(at[i].id.surname);
```

```
    printf(" Name: " );
```

```
    gets(at[i].id.name);
```

```
    printf(" Given name: " );
```

```
    gets(at[i].id.g_name);
```

```
int m = 0;
do {
    printf("Year:");
    ret_code = scanf("%f", &check_y);
    m = check_time((int)check_y);
    CLR;
}
while (round(check_y) != check_y || ret_code != 1 || m == 0);
at[i].year = round(check_y);
printf( "Brand:" );
gets(at[i].model);
do {
    printf("Price:");
    ret_code = scanf("%d", &at[i].price);
    CLR;
}
while (ret_code != 1 || at[i].price <=0);
}
```

```
//вывод массива автомобилей
void output(struct auto_ at[], int k) {
    for (int i = 0; i < k; i++) {
        printf("Car #%-d\n", i+1);
        printf("Surname:%s\nName:%s\n", at[i].id.surname, at[i].id.name);
        printf("Given name:%s\nYear:%d\n", at[i].id.g_name, at[i].year);
        printf("Model:%s\nPrice:%d\n", at[i].model, at[i].price);
    }
}

//поиск различных владельцев автомобилей с ценой менее 500 000
int Search (struct auto_ at[], int k, struct own founds[]) {
    int f=0;
    int j=0;
    for (int i=0; i<k; i++)
        if (at[i].price<500000){ //проверка отсутствия владельца в массиве
            for (j=0; j<f && (strcmp(at[i].id.name, founds[j].name)
                || strcmp(at[i].id.g_name, founds[j].g_name)
                || strcmp(at[i].id.surname, founds[j].surname)); j++);
            founds[j].name = at[i].id.name;
            founds[j].g_name = at[i].id.g_name;
            founds[j].surname = at[i].id.surname;
            founds[j].year = at[i].year;
            founds[j].model = at[i].model;
            founds[j].price = at[i].price;
            f++;
        }
}
```

```
if (j==f) { founds[f]=at[i].id; f++;
}
}
return f;
}

//марки с максимальным числом выпущенных автомобилей
void vv(struct auto_ at[], int k, char marki[][lmax], int *n)
{ int i, j;
int num[lmax];
int max = 0;
for (i = 0, *n = 0; i < k; i++)
{ //проверка присутствия маркив массиве различных марок
    for (j = 0; j < *n && strcmp(at[i].model, marki[j]); j++);
    if (j == *n) //это новая марка, заносим ее в массив
    {
        strcpy(marki[*n], at[i].model);
        num[(*n)++] = 1;
    }
}
```

```
else //эта марка уже была, увеличиваем число повторов
    num[j]++;
}
//ищем максимальное число марок
for (i = 0; i < *n; i++)
    if (num[i] > max)
        max = num[i];
//выводим все марки с максимальным числом автомобилей
printf("Marks with maximum number of cars:\n" );
for (i = 0; i < *n; i++)
    if (num[i] == max)
        puts(marki[i]);
}
int poisk_min(struct auto_ at[], int k, struct auto_ *ukaz[])
{
    int i, nu=0, min=at[0].price;
    //поиск минимальной цены
    for (i=1; i<k; i++)
        if (at[i].price < min)
            min=at[i].price;
```

```
for (i=0; i<k; i++)
    if (at[i].price == min)
        ukaz[nu]=&at[i], nu++;
return nu;
}

//вывод автомобилей с минимальной ценой
void vv_uk (struct auto_ *ukaz[], int nu)
{
printf(" Cars with the lowest prices\n");
int i;
for (i=0; i< nu; i++)
{
    printf("Car # %d\n", i+1);
    printf("Surname:%s\nName:%s\n", ukaz[i]->id.surname, ukaz[i]->id.name);
    printf("Given name:%s\nYear:%d\n", ukaz[i]->id.g_name,ukaz[i]->year);
    printf("Model:%s\nPrice:%d\n", ukaz[i]->model, ukaz[i]->price);
}
}
```

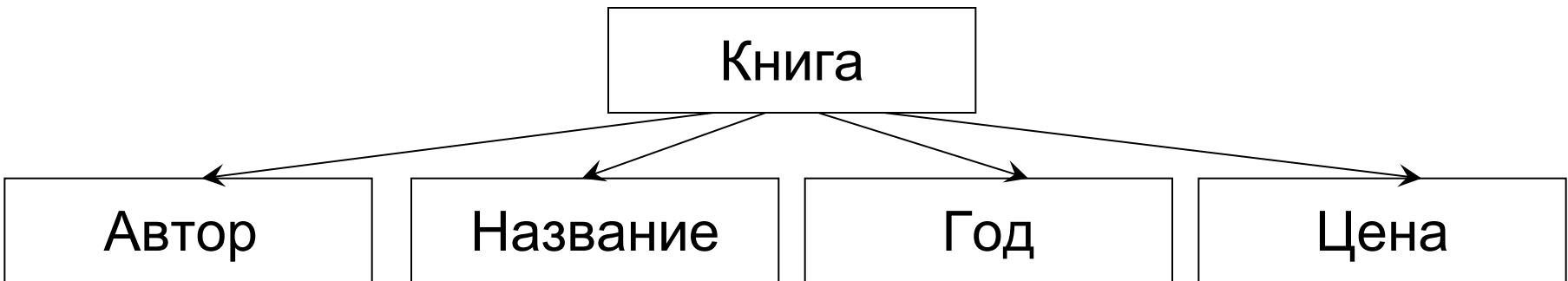
```
// годы выпуска с максимальным числом различных марок автомобилей
void yearswmax(struct auto_ at[], int k, int years[], int nmarks[], char m[][lmax][lmax],
    int *ny)
{ /* years[0:ny-1] - массив различных лет
   nmarks[0:ny-1]- количество различных марок, выпущенных в соответствующем
   году
   m[0:ny-1][0:nmarks[i]] - массив марок, выпущенных в соответствующем году */
    int i, j, l, max;
    *ny=0;
    for (i = 0;i < k;i++)
    { //проверка присутствия текущего года в массиве различных лет
        for (j = 0;j < *ny && at[i].year!=years[j]; j++);
        if (j==*ny) //такого года еще нет
        {
            years[*ny]=at[i].year; //заносим год в массив различных лет
            nmarks[*ny]=1; //одна марка
            strcpy(m[*ny][0],at[i].model); //заносим ее в массив марок
            (*ny)++;
        }
    }
}
```

```
else
{
    //такой год уже был, проверяем, была ли уже такая марка в этом году
    for (l = 0; l < nmarks[j] && strcmp(at[i].model,m[j][l]);l++);
    if (l == nmarks[j]) //такой марки не было
    {
        //заносим ее в массив марок и увеличиваем его длину
        strcpy(m[j][nmarks[j]],at[i].model);
        nmarks[j]++;
    }
}
//ищем максимальное число марок
for (i = 0, max=0;i < *ny ;i++)
    if (nmarks[i] > max)
        max = nmarks[i];
//выводим годы с этим числом
printf("These are years with max number of cars\n");
for (i = 0; i < *ny;i++)      if(nmarks[i]==max)      printf("%5d",years[i]);
}
```

```
int main()
{
    struct auto_ at[lmax], *ukaz[lmax];
    struct own founds[lmax];
    int f;
    int k, nu;
    //ввод массива автомобилей и вывод его на экран
    input(at, &k);
    output(at, k);
    char marki[lmax][lmax];
    int n;
    //поиск различных владельцев автомобилей с ценой менее 500 000
    f=Search(at,k,founds);
    if (!f) printf("No car owners with cars under 500,000!!!!");
    else
        for(int i=0; i<f; i++)
            printf("OWNER %d\n%s\n%s\n%s\n",i+1,founds[i].name,
                founds[i].g_name,founds[i].surname);
```

```
// марки с максимальным числом выпущенных автомобилей
vv(at, k, marki, &n);
//автомобили с минимальной ценой (формируем массив указателей)
nu=poisk_min(at,k,ukaz);
vv_uk(ukaz, nu);
//годы выпуска с максимальным числом различных автомобилей
int years[lmax], nmarks[lmax], ny;
char m[lmax][lmax][lmax];
yearswmax(at,k,years,nmarks,m,&ny);
return 0;
}
```

Задача. Даны структуры вида



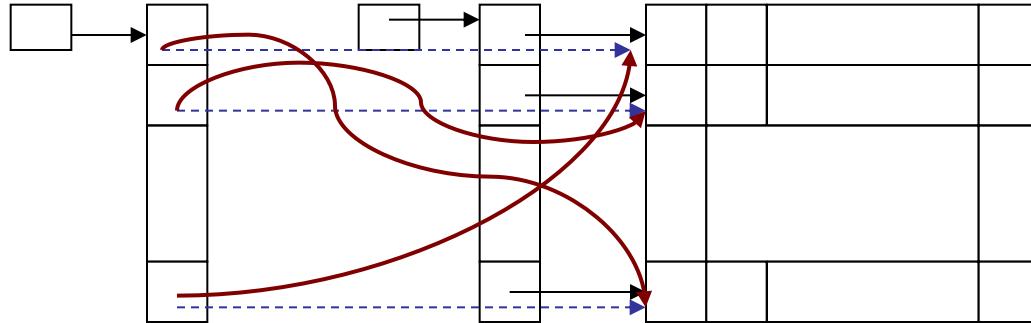
Из этих структур сформирован массив

```
struct book{  
    char author[20], title[60];  
    int year, price;  
};
```

Требуется упорядочить заданный массив структур вида KNIGA (каталог) в порядке возрастания года издания.

Примечание: В данном алгоритме производится только сортировка массива указателей на структуры, но сами информационные поля не переставляются.

ukn cat



Данные будут выводиться на экран в той
последовательности, в которой показывает измененный
массив указателей.

```
/*-----Функция вывода массива книг-----*/
void output(struct book *cat[], int kol)
//передаем массив указателей на структуры
{  int i;char ch;
   printf("Ordered catalog of books:\n");
   for (i=0; i<kol; i++)
   {
      printf("Book # %d\n", i+1);
      printf("Author.....%s\n", cat[i]->author);
      printf("Title.....%s\n", cat[i]->title);
      printf("Year of publishing..%d\n", cat[i]->year);
      printf("Price.....%d\n", cat[i]->price);
      puts("Press any key");
      while((ch=getchar())!='\n');
   }
}
```

```
void sort(struct book cat[], int kol) //Функция сортировки
{ int i, j;
book *ukn[lmax], *cat1; /* *ukn[500] - массив указателей */
for(i=0; i<kol; i++) ukn[i]=cat+i;
//Производится только сортировка указателей
for(i=0; i<kol-1; i++)
    for(j=i+1; j<kol; j++)
        if (ukn[i]->year>ukn[j]->year)
            cat1=ukn[i], ukn[i]=ukn[j], ukn[j]=cat1;
output(ukn, kol);
}
```

```
int main()
{
struct book cat[lmax];
int kol;
readcat(&kol, cat); //аналогично задаче 3 из сем.9
sort(cat, kol);
return 0;
}
```

Текстовые файлы

Текстовый файл - последовательность символов (кодов), разделенная на строки.

c	1	...	\n	1	2	...	\n	a	c	2	7	8	\n	EOF
0				1					...							

EOF – специальный символ для проверки и обозначения конца файла. Определен в заголовочном файле stdio.h; ввод с клавиатуры этого символа соответствует нажатию клавиш CTRL+Z.

Возможны следующие операции:

`int ch;`

`ch = EOF; //присваивание`

`if (ch == EOF) //проверка конца файла`

Для работы с файлом необходим объект FILE. Этот объект хранит идентификатор файлового потока и информацию, которая нужна, чтобы им управлять, включая указатель на его буфер, индикатор позиции в файле и индикаторы состояния.

Объект FILE является структурой, но к его полям нет свободного доступа.

`FILE *fl;`

Стандартные указатели на файлы определены в `<stdio.h>`

`FILE *stdin, *stdout, *stderr ;`

`stdin` - стандартный входной поток;

`stdout` - стандартный выходной поток;

`stderr` - стандартный поток вывода ошибок.

```
#include <stdio.h>

int main() {
    int a, b;

    fprintf(stdout, "Enter two numbers\n");
    fscanf(stdin, "%d", &a);
    fscanf(stdin, "%d", &b);
    if (b == 0) {
        fprintf(stderr, "Error: divide by zero");
    } else {
        fprintf(stdout, "a/b= %.3f", (float) a / (float) b);
    }
    return 0;
}
```

Открытие файла. При открытии указывается способ использования файла (считывание, запись, добавление). Функция открытия файла `fopen()` :

`FILE *fopen(char *filename, char type);`

Функция `fopen()` возвращает указатель на файл (если файл нельзя открыть, то результат равен `NULL`).

Функция `fopen()` имеет два аргумента:

filename – имя открываемого файла (может содержать путь);

type - способ использования (строка).

- “`r`”. Чтение. Файл должен существовать.
- “`w`”. Запись нового файла. Если файл с таким именем уже существует, то его содержимое будет потеряно.
- “`a`”. Запись в конец файла. Файл создаётся, если не существовал.
- “`r+`”. Чтение и обновление. Можно как читать, так и писать. Файл должен существовать.
- “`w+`”. Запись и обновление. Создаётся новый файл. Если файл с таким именем уже существует, то его содержимое будет потеряно. Можно как писать, так и читать.
- “`a+`”. Запись в конец и обновление. Если файл не существовал, то будет создан новый.

```
FILE *fl ;  
fl = fopen("prog1.txt", "w");
```

Здесь prog1.txt - имя файла.

Закрытие файла осуществляется функцией fclose(), аргумент которой - указатель:

```
int fclose(FILE *fl);  
fclose (fl) ;
```

При нормальном закрытии файла функция возвращает 0, в противном случае возвращается EOF .

Функция **fcloseall ()** осуществляет закрытие всех потоков, освобождает буферы ввода/вывода, куда данные записываются перед их пересылкой в файл на диск. Буферы используются для ускорения обмена ОЗУ с дисками.

void rewind (FILE *fl) – устанавливает указатель в начало файла.

ФУНКЦИИ ВВОДА И ВЫВОДА

1. `int fgetc(FILE *fl)` ; - чтение одного символа из файла fl.
Возвращает код введенного символа. Если достигнут конец файла или произошла ошибка, то вернет EOF.
2. `int fputc(int c, FILE *fl);` - вывод символа с кодом c в файл fl.

Форматный ввод/вывод

3. `int fscanf(FILE *fl, "форматы", <список аргументов>);`
4. `int fprintf (FILE *fl, "форматы", <список аргументов>);`

Обе функции возвращают количество успешно обработанных аргументов, при ошибке возвращается EOF.

Ввод строки

```
char *fgets(char *str, int n, FILE *fl);
```

Функция fgets () считывает из файла fl в строку str символы до тех пор, пока не будет выполнено одно из условий:

1. начнется новая строка, т.е. встретится символ '\n';
2. будет достигнут конец файла (EOF);
3. условия 1 и 2 не выполнены, но прочитано n-1 символов.

После считывания строка дополняется нуль символом '\0'. Если при чтении встретился символ конца строки, то он переносится в строку str и нуль-символ записывается за ним. Если считывание прошло успешно, то возвращается адрес строки str, в противном случае - NULL.

Напомним, функция gets () заменяет символ '\n' на '\0'. Считывание символов осуществляется из стандартного входного потока stdin. Если входной поток прерывается символом перехода на новую строку '\n', то он отбрасывается и не попадает в строку str.

Пример: Символы переписываются функцией puts () в стандартный выходной поток stdout, строка str дополняется символом конца строки '\n'.

```
char str[10];
```

```
FILE *fl;
```

```
fl=fopen("f1.txt","r"); //fl:
```

```
fgets(str,3,fl); //str:
```

```
puts(str);
```

```
fgets(str,10,fl); //str:
```

```
puts(str);
```

a	b	c	d	\n
---	---	---	---	----

a	b	\0
---	---	----

ab

a	b	c	d	\n	\0
---	---	---	---	----	----

abcd

Вывод строки в файл

```
int fputs (char *str, FILE *fl);
```

Строка str, ограниченная символом '\0', переписывается в файл fl, причем символ '\0' **отбрасывается**.

```
fputs ("abcd",fl) ;  
fputs("ef\n",fl);
```

abcdef\n

Результат в файле fl:

В отличие от puts () функция fputs не добавляет символ '\n' в файл.

Определение конца файла

```
int feof(FILE *fl);
```

Выдает истинное значение, если при чтении достигнут конец файла, в противном случае - нулевое.

Пример 1. Дан файл целых чисел. Числа в файле расположены по одному на каждой строке. Прочитать данные в массив (количество чисел не превышает заданного числа).

```
#include <stdlib.h> //для exit()
void getArray(char *filename, int *na, int a[])
//на входе строка с именем файла,
{ //на выходе – массив и его длина
FILE *f; char c;
if (!(f=fopen(filename,"r")))
{   puts("File not found");
exit(1);
}
*na=0;
while(!feof(f))
if ((fscanf(f,"%d",&a[*na]))==1) (*na)++;
else
while((c=fgetc(f))!='\n'&&c);
//пропуск символов до конца строки или EOF
fclose(f);
}
```


Пример 2. Прочитать матрицу из файла (в первой строке файла записаны число строк и столбцов матрицы).

```
void getMatrix(char *filename, int
    *na, int *ma, int a[][10])
/*на входе строка с именем
файла, на выходе – матрица и
число строк и столбцов в ней*/
{
FILE *f;
int i,j;
if (!(f=fopen(filename,"r")))
{
    puts("File not found");
    exit(1);
}
```

```
fscanf(f,"%d%d",na,ma);
for (i=0;i<*na;i++)
    for (j=0;j<*ma;j++)
        if ((fscanf(f,"%d",&a[i][j]))!=1)
        {
            printf("Error in data:
a[%d][%d]",i,j);
            exit(1);
        }
fclose(f);
}
```

Пример 3: Написать функцию формирования текстового файла, состоящего из строк, вводимых с клавиатуры.

```
#include <stdio.h>
#include <string.h>

void writetext(char *fname)
/*на входе строка с именем
файла*/
{
FILE *fout;
char str[81];

printf("Input strings\n When string is
empty, input is over\n");
fout=fopen(fname,"w");
while(gets(str)!=NULL&&
strcmp(str,"")!=0)
```

```
/* Пока не встретится EOF или не будет
прочитана пустая строка */

{
fputs (str,fout);
fputc('\n',fout); /* признак конца строки
- fputs() отбрасывает '\0' и в отличие
от puts не добавляет \n */

}
fclose(fout);
printf("Result is in file %s\n",fname);
}

//Вызов :
writetext("output.txt");
```

Пример 4: Вывести на экран содержимое текстового файла с именем "fl.txt".

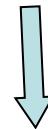
```
#include <stdio.h>
#define Max_Len 81
void deln(char *s);

int main()
{ FILE *fl;
char str[Max_Len];

fl=fopen("fl.txt", "r");
while(fgets(str,81,fl)!=NULL)
/*чтение файла построчно */
{deln(str); //замена '\n' на'\0'
 puts(str);}
fclose(fl);
return(0);
}
```

```
/*-----Функция замены в строке си
мвола '\n' на '\0'-----*/
void deln(char *s)
{ for (; *s && *s!='\n'; s++);
  *s='\0';
}
```

Иванов А.А. \n \0



Иванов А.А. \0 \0

Установка позиции в текстовом файле ввода-вывода

Указатель текущего байта (счетчик байтов)



Рассмотренные функции осуществляют ввод/вывод, начиная с текущей позиции в файле. При открытии файла счетчик указывает на нулевой байт. Функция `fseek()` позволяет установить позицию в файле, с которой будет осуществляться последующая операция ввода-вывода. Нумерация позиций в файле начинается с 0.

```
int fseek(FILE *fl, long delta, int code);
```

fl - указатель на файл;

delta - перемещение от начальной точки (шаг):

- >0 - движение вперед;
- <0 - движение назад;

code - код начальной точки:

- 0 - начало файла (SEEK_SET);
- 1 - текущая позиция файла (SEEK_CUR);
- 2 - конец файла (SEEK_END).

Результат:

0 - успешное завершение операции;

-1 (EOF) - ошибка (выход за пределы файла).

Если в файле размещены элементы фиксированной длины, то доступ к ним возможен по номеру элемента, как в массиве.

Запись:

```
for(i=0;i<n;i++)fprintf(fl,"%10d",a[i]); //10 байтов на элемент
```

Чтение:

```
if(fseek(fl,10*i,0)==0) fscanf(fl,"%d", &a[i]);  
else printf("Слишком большой номер");
```

```
long ftell(FILE *fl);
```

fl - указатель на файл. Возвращает количество символов в файле.

```
#include <stdio.h>
int main()
{FILE* fp = fopen("my_file.txt","w");
If (fp != NULL)  fprintf(fp,"It is an example using fseek and ftell functions.");
fclose(fp);
fp = fopen("my_file.txt","r");
if(fp != NULL)
{char ch;
fseek(fp,0L,SEEK_END); //указатель позиции помещается в конец файла
long length = ftell(fp); //количество символов
printf("length = %ld\n",length);
for(int i = 1;i <= length;i++)
{fseek(fp,-i,SEEK_END); /*смещает указатель позиции на –i символов
    относительно конца файла*/
ch = getc(fp);putchar(ch);
}
}
fclose(fp);
return 0;}
```

```
length = 49
.snoitcnuf lletf dna keesf gnisu elpmaxe na si tI
```

Для обработки ошибок ввода-вывода удобно использовать функцию

`void exit(int status).`

Прототип: `stdlib.h.`

Описание:

Функция `exit()` вызывает немедленное нормальное завершение программы.

По соглашению, если значением параметра `status` является 0, то предполагается нормальное завершение программы. Ненулевое значение используется для указания ошибки. Вызов функции `exit()` осуществляет очистку буферов файлов, закрывает все открытые файлы

Задача 1. Прочитать квадратную матрицу из файла(в первой строке файла записано число строк матрицы). Сформировать одномерный массив, содержащий элементы главной и побочной диагонали матрицы без повторения. Результат вывести в файл.

```
#include <stdio.h>
#include <stdlib.h>
//ввод имени файла
void getname (char * type, char *filename)
{ printf("Input file name for: %s : ",type); gets(filename);}
void getdata(char *filename, int *na, int a[][10])
{ FILE *f;    int i,j;
if (!(f=fopen(filename,"r")))
{   puts("File not found"); exit(1);}
fscanf(f,"%d",na);
for (i=0;i<*na;i++)
    for (j=0;j<*na;j++)
        if ((fscanf(f,"%d",&a[i][j]))!=1) {printf ("Error in data
a[%d][%d]",i,j);exit(1);}
fclose(f);
}
```

```
void putdata(int na, int a[][10])
{
int i,j;

printf(" Matrix [%d*%d] :\n",na,na);
for(i=0;i<na;i++)
{   for(j=0;j<na;j++)
    printf("%7d",a[i][j]);
    printf("\n");
}
}
```

```
int inb(int e,int b[],int nb)
{ int i;

for (i=0;i<nb&&b[i]!=e;i++);
if (i<nb) return 0;
else return 1;
}
```

```
void makearray (int na,int (*a)[10], int *nb, int b[])
{
    int i ;

    *nb=0;
    for(i=0;i<na;i++)
    { if (inb(a[i][i],b,*nb))b[(*nb)++]=a[i][i];
      if (inb(a[i][na-i-1],b,*nb))b[(*nb)++]=a[i][na-i-1];
    }
}

void makefile(char *filename,int nb,int b[])
{int i;
FILE *f=fopen(filename,"w");

    for (i=0; i<nb; i++)
        fprintf(f,"%5d", b[i]);
fclose(f);

printf("Result in %s\n",filename);
}
```

```
void putfile(char *filename)
{int t;
 FILE *f;

if (!(f=fopen(filename,"r")))
{
    puts("File not found");
    exit(1);
}
puts("Result ");
while(!feof(f))
{
    fscanf(f,"%d",&t);
    printf("%7d", t);
}
puts("");
fclose(f);
}
```

```
int main()
{
char filename1[20], filename2[20];
int a[10][10], na, b[20],nb;

getname ("Input", filename1);
getname ("Output", filename2);
getdata(filename1, &na, a);
printf("Data");
putdata(na,a);
makearray(na, a,&nb,b);
makefile(filename2,nb,b);
putfile(filename2);
return 0;
}
```

Задача 2. Слова в заданном файле разделены пробелами. Количество слов неизвестно. Отсортировать их в лексикографическом порядке и записать результат в другой файл, каждое слово - с новой строки.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void sort(char *stroki[], int kol) //stroki - массив указателей на слова
{   char *work;   int i, j;
    for (i=0; i<kol-1; i++)
        for (j=i+1; j<kol; j++)
            if (strcmp(stroki[i], stroki[j])>0)
                work=stroki[i], stroki[i]=stroki[j], stroki[j]=work;
}

void deln(char *s)
{   for (; *s && *s!='\n'; s++);
    *s='\0';
}
```

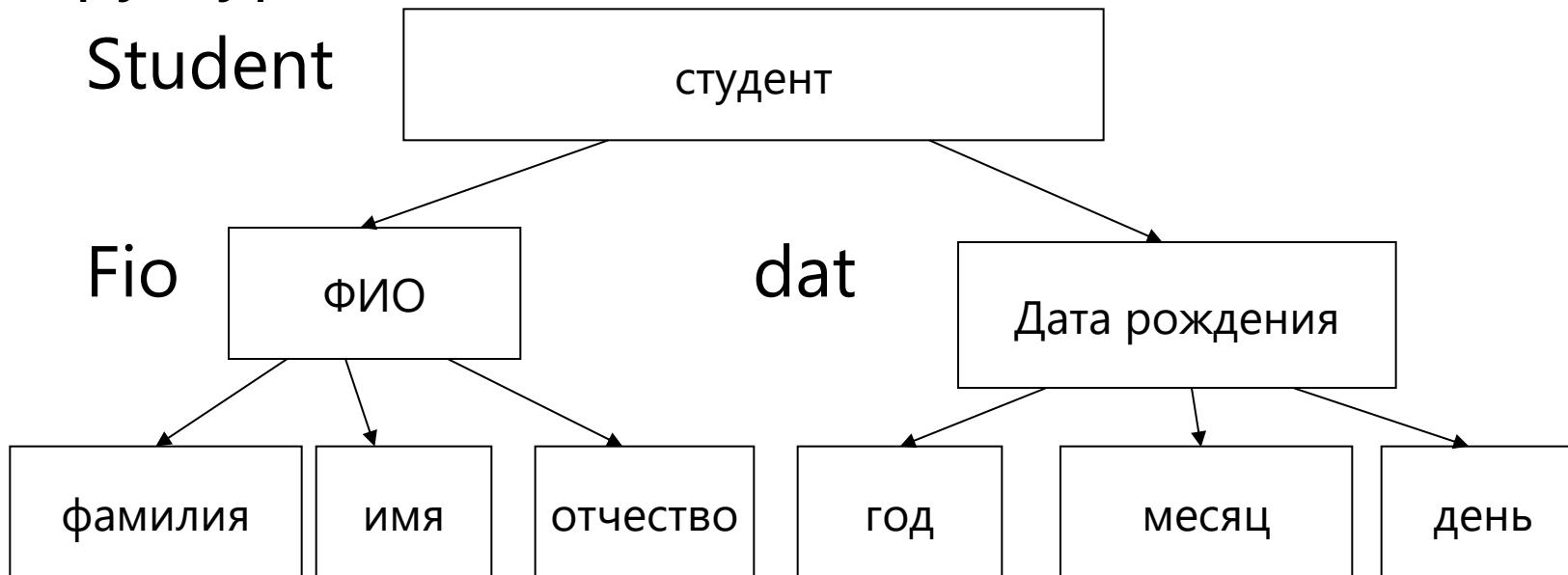
```
int main()
{
    FILE *fin, *fout;
    char slova[100][20], *uks[100]; /* uks - массив указателей */
    int i, t, ks=0;
    char filename[81];

    printf("Input file name with data ");
    fin=fopen(gets(filename), "r");
    if (!fin)
    {
        printf("File not found!\n");
        exit(1);
    }
    printf("Input file name for output");
    fout=fopen(gets(filename), "w");
```

```
while (fscanf(fin, "%s", slova[ks])!=EOF && ks<100)
{
    deln(slova[ks]);
    uks[ks]=slova[ks++];
}
/* Количество прочитанных слов равно ks */
sort(uks, ks);
for (i=0; i<ks; i++)
    fprintf(fout, "%s\n", uks[i]);
fclose(fin);
fclose(fout);
printf("Result in file %s\n", filename);
return 0;
}
```

Задача. Дан список группы, состоящий из структур вида:

Student



sn

fn

rn

year

month

day

Сформировать текстовый файл, содержащий эти данные. Найти самого молодого студента.

```
#include <stdio.h>
#include <string.h>
#define lmax 100
#define lmax 40

#define FCLR while (fgetc(f) != '\n')
#define CLR while (getchar() != '\n')

struct student {
    struct {
        char sn[lmax];
        char fn[lmax];
        char pn[lmax];
    }fio;
    struct {
        int year, month, day;
    }dat;
};

};
```

```
int young(int n, struct student gr[])
{int i, i1, pr, max;

max=0;
i1=0;
for(i=0;i<n;i++)
{
    pr= gr[i].dat.year*10000+gr[i].dat.month*100+gr[i].dat.day;
    if (pr > max )
    {
        max= pr;
        i1= i;
    }
}
return i1;
}
```

```
void make_file(int *n, struct student gr[], char filename[])
{int i; FILE *f=fopen(filename,"w");
 do{ puts("Enter number of students:"); scanf("%d",n); CLR;
 } while(*n <= 0 || *n > lmax);
 puts("Enter data");
 for(i=0;i<*n;i++)
 { puts("surname "); gets(gr[i].fio.sn); fprintf(f,"%s\n",gr[i].fio.sn);
  puts("name "); gets(gr[i].fio.fn); fprintf(f,"%s\n",gr[i].fio.fn);
  puts("givenname "); gets(gr[i].fio.bn); fprintf(f,"%s\n",gr[i].fio.bn);
  puts("year "); scanf("%d",&gr[i].dat.year);
   fprintf(f,"%d\n",gr[i].dat.year);
  puts("month "); scanf("%d",&gr[i].dat.month);
   fprintf(f,"%d\n",gr[i].dat.month);
  puts("day "); scanf("%d",&gr[i].dat.day);
  fprintf(f,"%d\n",gr[i].dat.day);
 CLR;
 }fclose(f);
}
```

```
void deln(char *s)
{ for (; *s && *s!='\n'; s++);
  *s='\0';
}
int get_data(char filename[], struct student gr[])
{FILE *f=fopen(filename,"r");
 int i = 0;
 do
 {   if(fgets(gr[i].fio.sn,lmax,f))
     {delen(gr[i].fio.sn);
      fgets(gr[i].fio.fn,lmax,f);delen(gr[i].fio.fn);
      fgets(gr[i].fio.pn,lmax,f);delen(gr[i].fio.pn);
      fscanf(f,"%d",&gr[i].dat.year);
      fscanf(f,"%d",&gr[i].dat.month);
      fscanf(f,"%d",&gr[i].dat.day);
      i++;FCLR;}
   } while (!feof(f));
 fclose(f);
 return i;
}
```

```
void output(struct student gr[], int n)
{ if (n==0)      puts("Empty array");
  else {      puts("Array of students:");
    for (int i=0;i<n;i++)
    { printf("Student number:%d\n",i+1);
      puts("surname "); puts(gr[i].fio.sn);
      puts("name "); puts(gr[i].fio.fn);
      puts("givenname "); puts(gr[i].fio.pn);
      puts("year "); printf("%d\n",gr[i].dat.year);
      puts("month "); printf("%d\n",gr[i].dat.month);
      puts("day "); printf("%d\n",gr[i].dat.day);
      puts("Press any key");      getchar();
    }
  }
}
```

```
int main()
{
struct student gr[Lmax];
char filename[lmax];
int i1, n;

puts("Enter file name");
gets(filename);

make_file(&n, gr, filename);
puts("File is ready");
n=get_data(filename,gr);
puts("File ");
output(gr,n);
i1 = young(n, gr);
puts("The yongest one");
puts( gr[i1].fio.sn); puts(gr[i1].fio.fn);puts( gr[i1].fio.bn);
return 0;
}
```


Контрольные точки второго модуля

- 3 лабораторные работы, защита отчетов как в 1 модуле; (5 баллов)
- 1 контрольная работа (письменно) (3 балла);
- контроль усвоения материала на лекциях и семинарах (2 балла);
- экзамен (тест).

Dead-lines для лабораторных работ

1. 4 занятие;
2. 8 занятие;
3. 12 занятие

(считываются занятия обеих подгрупп по порядку, указанному в расписании).

Для вычисления оценки текущего контроля по дисциплине используется следующая таблица.

	Работа на семинарском занятии	Работа на лекции	Выполнение лабораторного практикума	Контрольная работа
1 модуль	1	2	7 (3+4)	
2 модуль	1	1	5(2+2+1)	3

В скобках указано распределение баллов по лабораторным работам.

Оценка за текущий контроль в 1 и 2 модуле учитывает результаты студента следующим образом.

Модуль 1. $O_{текущая\ 1} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски}$.

Модуль 2. $O_{текущая\ 2} = O_{лекция} + O_{семинар} + O_{лаб.\ работы} + O_{ответы\ у\ доски} + O_{контр.\ работы}$.

Все оценки рассматриваются без округления.

Результирующая оценка учитывает оценки модулей. Промежуточная оценка за 1 и 2 модуль вычисляется по формуле

$O_{промежуточная\ 1\ и\ 2} = 0,3 * O_{текущая\ 1} + 0,5 * O_{текущая\ 2} + 0,2 * O_{экзамен\ 2\ модуль}$,

где $O_{текущая\ 1}$, $O_{текущая\ 2}$ – оценки текущего контроля 1, 2 модуля, без округления.

Округление производится один раз, после вычисления промежуточной оценки, по правилам арифметики.

Экзаменационная оценка не является блокирующей.
Промежуточная оценка за 1 и 2 модуль не может превышать 10 баллов, в случае превышения ставится промежуточная оценка 10 баллов.

Результирующая оценка за дисциплину вычисляется по формуле

$$O_{\text{результативная}} = 0,4 \cdot O_{\text{промежуточная 1 и 2}} + 0,6 \cdot O_{\text{промежуточная 3 и 4}}$$

Округление производится по правилам арифметики.

В диплом выставляется результирующая оценка.

ФУНКЦИИ динамического распределения памяти

Функция malloc ()

В языке Си имеются следующие способы выделения памяти :

- `char s1 []=" символьная константа";` будет выделена память для хранения строки (минимальное необходимое число ячеек);
- `int a[100];` будет выделено 100 ячеек памяти, каждая из которых предназначена для хранения целого числа.

Си позволяет распределять дополнительную память во время работы программы, используя функцию `malloc ()` .

`void *malloc(unsigned size);`

Функция выделяет `size` байтов. Возвращаемое значение - указатель на тип `void`. Нужно указать тип при выделении памяти

Для использования функций динамического распределения памяти подключается библиотека `malloc.h`.

Пример: выделить память для хранения строки и ввести ее.

- `char stroka [81]; //статическое выделение памяти`
`gets(stroka);`
- `char *stroka; //указатель на строку`
`stroka=(char*) malloc(81); //динамическое выделение памяти`
`gets(stroka);`

Чтобы получить указатель на тип, необходимо осуществить преобразование типа возвращаемого значения:

`(<новый тип> *)malloc(<size>);`

, где *size* – размер памяти, необходимой для размещения данных нового типа.

`void *calloc (unsigned n, unsigned size);`

Выделяется память для *n* элементов, каждый из которых занимает *size* байтов. Значения выделенных ячеек обнуляются.

Функция free ()

Функция освобождает память в процессе выполнения программы:

`free(void *p);`

Узнать фактический размер выделенной памяти мы можем с помощью функции `_msize`.

```
#include <stdio.h>
#include <malloc.h>
int main()
{unsigned long length = 0; //число выделенных байтов
 void* c_ptr = malloc(25); //выделение памяти
 if (c_ptr != NULL) //если память выделена
 { //вычисляем размер выделенной памяти
     length = _msize(c_ptr);
     printf("malloc_usable_size(c_ptr) == %i\n", length);
 } else
     printf("Memory allocation error\n");
 return 0;
}
```

Функция sizeof ()

sizeof(е); возвращает число байтов, требуемых для размещения данных типа е, где е – имя переменной или выражение.

sizeof('9') 1 байт //тип char

int i; 

sizeof(i) 4 байта

Если е определяет массив, то возвращается размер памяти, занимаемый всем массивом.

int mas[100];

sizeof(mas) 400 байт

Пример 

sizeof(mas) - число байтов для размещения всего массива;

sizeof(int) - число байтов для размещения одного элемента;

n=(int)sizeof (mas) / sizeof (int) - число элементов массива.

Массивы в языке Си

Нумерация элементов массива начинается с нуля. Имя массива – указатель на его первый элемент, т.е. `mas==&mas[0]`. При объявлении массива указывается его длина.

Например, `int mas[10];`



Значение i -ого элемента массива можно представить двумя способами: `mas[i]`; или `*(mas+i)`.

Пример. Динамическое выделение памяти для одномерного массива. Суммирование элементов массива.

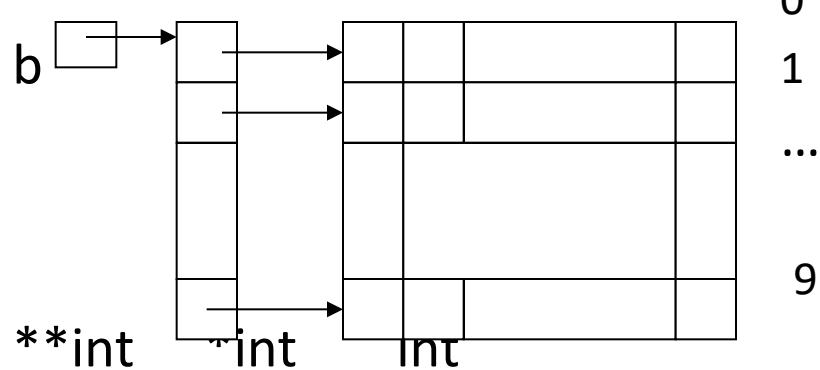
```
#include<stdio.h>
#include<malloc.h>
int main()
{int *a, summa=0;
unsigned k; //длина массива
int i;
printf("Enter the length of
array k>0 :");
scanf("%u",&k);
//выделение памяти
a=(unsigned int*)calloc(k,
sizeof(int));
printf("Input elements: ");
```

```
for(i=0;i<k;i++)
{printf("a[%d]=",i);
scanf("%d",a+i);
}
for (i=0;i<k;i++)
summa+=a[i];
printf("summa=%d\n",
summa);
free(a); /*освобождение
памяти*/
return(0);
}
```

Матрица в языке С

Для матрицы при объявлении указывается число строк и столбцов.

Нумерация элементов матрицы начинается с нуля.



0
1
...
9

`int b[10][20].`
Обращение к элементу:
`b[i][j]`
`*(b[i]+j)`
`*(*(b+i)+j)`

Пример. Динамическое выделение памяти для двумерного массива. Суммирование элементов матрицы.

```
#include<stdio.h>
#include<malloc.h>
int main()
{int **b,i,j,m,n,summa=0;
printf("Input n,m >0 ");
scanf("%d%d",&n,&m);
/*выделение памяти для массива указателей на строки матрицы*/
b=(int**)calloc(n,sizeof(int *));
for(i=0;i<n;i++)
{b[i]=(int*)calloc(m,sizeof(int)); /*выделение памяти для строки */
 for(j=0;j<m;j++)
 { printf("b[%d][%d]=",i,j); //ввод элемента матрицы
 scanf("%d",b[i]+j);
 }
}
}
```

```
for (i=0;i<n;i++)
    for(j=0;j<m;j++)
        summa+=b[i][j];
printf("summa=%d\n",summa);
//освобождение памяти
for(i=0;i<n;i++)
    /*освобождение памяти, выделенной для строки */
    free(b[i]);
/*освобождение памяти, выделенной для массива указателей
на строки */
free(b);
return(0); }
```

```
void *realloc(void *ptr, size_t newsize)
```

Прототип в stdlib.h.

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. Возвращается указатель на блок памяти, поскольку может возникнуть необходимость переместить блок при возрастании его размера. В таком случае содержимое старого блока копируется в новый блок и информация не теряется.

Если свободной памяти недостаточно для выделения в куче блока размером newsize, то возвращается нулевой указатель.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
int main()
{char *p;p = (char *) malloc(17); //выделение памяти
if(!p) {printf("Allocation error.");exit (1);}
strcpy(p, "This is 16 chars");
puts("Original string"); puts(p);
p = (char *) realloc (p,18); //добавляем место для 1 символа
if(!p) {printf("Allocation error.");exit (1);}
strcat (p, ".");
puts("Changed string");puts(p);
free(p); //освобождение памяти
return 0;
}
```

Динамические структуры данных

Динамические структуры данных - структуры, размер которых в программе явно не определяется. Память под элементы, входящие в эти структуры, выделяется в процессе работы программы.

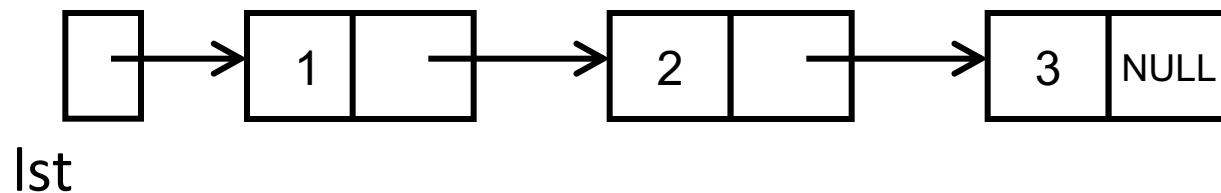
Пример динамических структур – связанные списки.

Основные типы списков уже изучались в языке Паскаль.

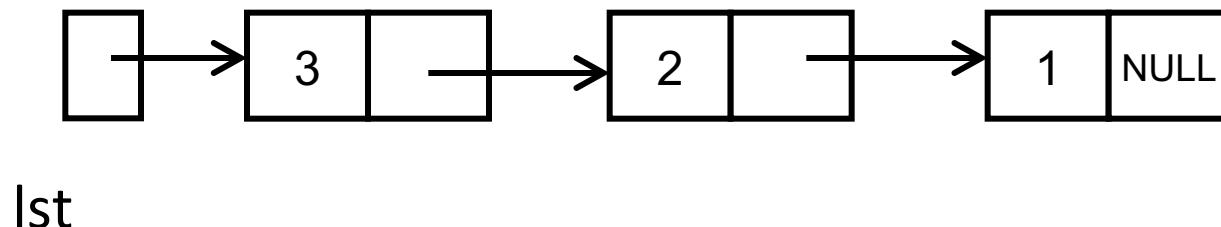
Пусть исходные данные для создания списка – числа 1, 2, 3 и 0 (0-признак окончания ввода данных).

1. Линейные односторонние списки

1. Очередь (FIFO – first in – first out)



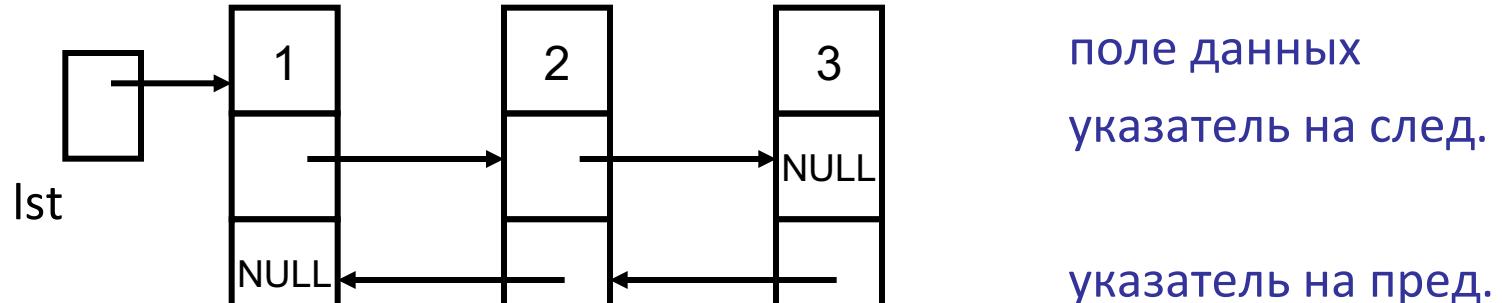
2. Стек (LIFO – last in – first out)



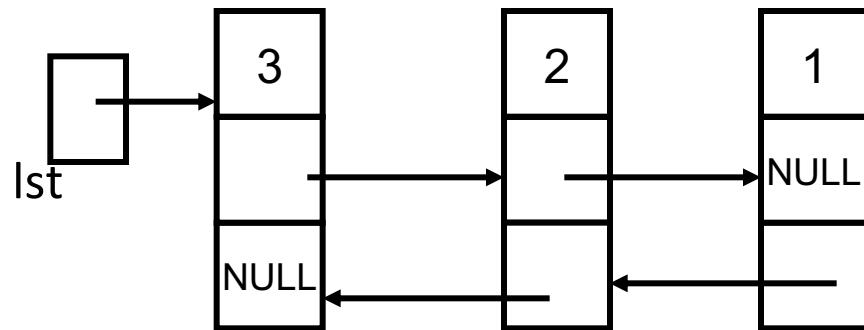
Элемент двунаправленного списка содержит указатель не только на следующий, но и на предыдущий элемент списка.

2. Линейные двунаправленные списки

1. Очередь



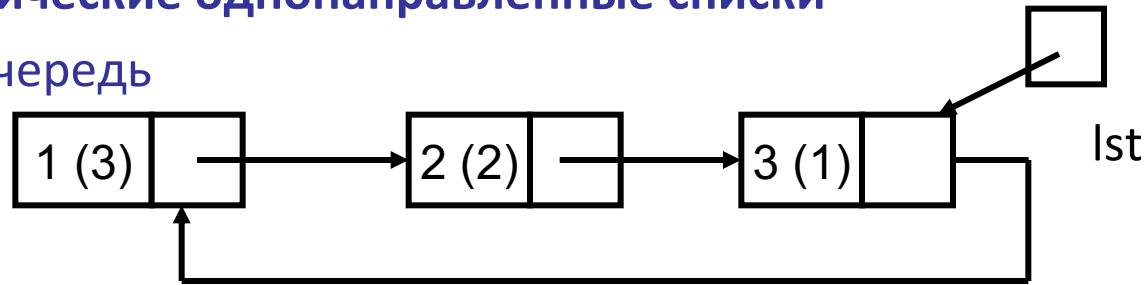
2. Стек



В циклическом списке последняя запись указывает на первую. Для циклического списка удобно возвращать адрес последней записи.

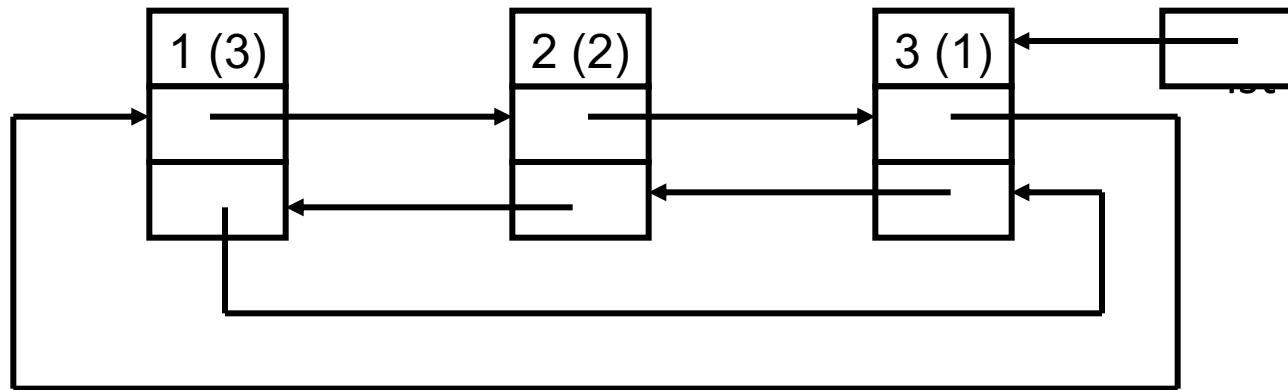
3. Циклические односторонние списки

1. Очередь



2. Стек отличается от очереди порядком расположения полей данных. Порядок данных для стека указан на предыдущем рис. в скобках.

4. Циклические двунаправленные списки – очередь (стек).

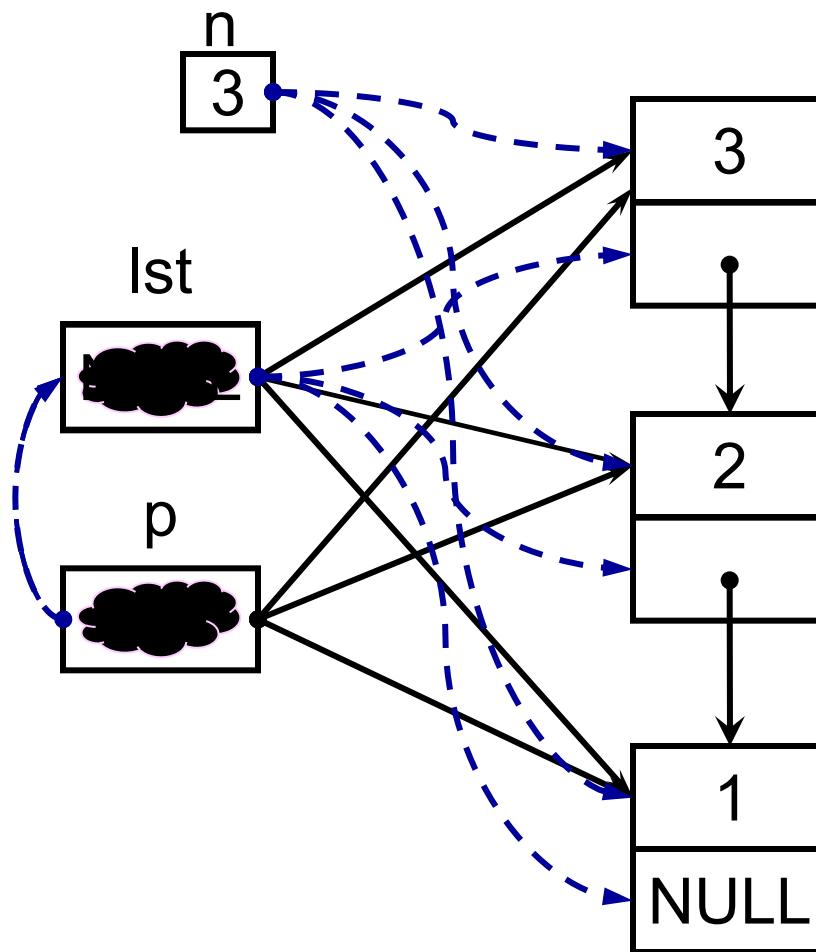


Пример 1: Написать программу, которая вводит целые положительные числа в стек, а затем вычисляет их сумму.

```
#include <stdio.h>
#include <malloc.h>
struct node
{struct node *next; int info; };

struct node *stack()
//формирование стека
{int n;
struct node *p, *lst; /* p -
указатель на текущую
добавляемую запись, lst -
указатель на предыдущую
запись (на вершину стека) */
```

```
lst=NULL; //пустой стек
printf("Enter positive integers\n");
while(scanf("%d",&n)==1&&n>0)
//ввод до Ctrl+Z или n<=0
{
p= (struct node*) malloc(sizeof(struct
node));
p->info=n;
p->next=lst;
lst=p;
}
return(lst);
/*вернет указатель на вершину
стека*/
}
```



```
int sumspisok(struct node *lst)//возвращает сумму элементов
//списка
{int sum=0;
node *p=lst; //указатель на текущую запись
while(p!=NULL) //лучше while(p)
{
    sum+=p->info;
    p=p->next;
}
return(sum);
}
//вывод списка на экран
void printspisok (struct node *lst)
{
struct node *p=lst;
while(p) { printf("%7d",p->info);
    p=p->next;
}
printf("\n");
}
```

```
//освобождение памяти
void free_memory(struct node *lst)
{
    struct node *now=lst, *next=lst;
    while (next)
    {next=now->next;
     free(now);
     now=next;
    }
    puts("\nNow memory is free");
}
int main()
{struct node*lst=stack();
 if(!lst) puts ("List is empty");
 else
 { puts("Original list");printspisok (lst);
  printf (" summ = %7d", sumspisok(lst) );
  free_memory(lst);
 }
 return 0;}
```

Создание очереди и вывод ее на экран

Задача 1. Сформировать линейную одностороннюю очередь. Вывести поля данных на экран, а затем освободить память, занятую списком.

Примечание. Используем три указателя:

add - указатель на текущую добавляемую запись;

first - указатель на первую запись;

last - на последнюю (к ней присоединяется новая).

При выводе используем указатель на начало списка `lst` в качестве указателя на текущую запись (т.к. в функцию передается копия адреса начала списка).

```
struct node *queue() /* очередь, вернет указатель на первую запись*/
{int n; struct node *add, *last, *first=NULL; /* Сначала список пуст */
printf("Enter positive integers\n");
if(!scanf("%d",&n)||n<=0) return first;
else{ first=(struct node*) malloc(sizeof(struct node));
//выделим память для первой записи
first->info=n; //занесем число в поле данных
last=first; //сделаем эту запись последней
while(scanf("%d",&n)==1&&n>0) //ввод до Ctrl+Z или n<=0
{ add=(struct node*) malloc(sizeof(struct node));
//память для добавляемой записи
add->info=n;last->next=add; //присоединим запись к списку
last=add; //сделаем эту запись последней
}
last->next=NULL; //это последняя запись в списке
return(first); //вернем адрес начала списка
} }
```

```
void output_list (struct node *lst) //вывод списка на экран
{ if(!lst) puts ("list is empty");
else
{ while(lst)
{ printf("%7d", lst->info);
lst=lst->next;
}
printf("\n");
}
}

void free_memory(struct node *lst) //освобождение памяти
{ struct node *now=lst, *next=lst;
while (next)
{next=now->next;
free(now);
now=next;  }
puts("\nNow memory is free");}

```


Задачи обработки связанных списков

Задача 1. Сортировка связанного списка по возрастанию методом установки.

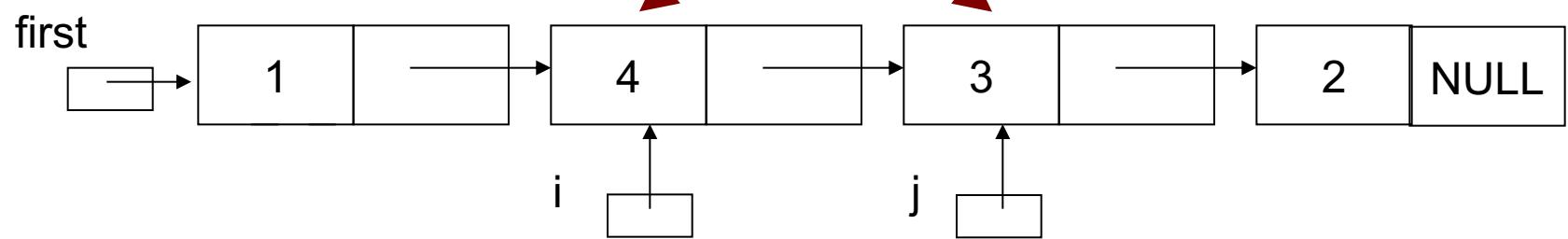
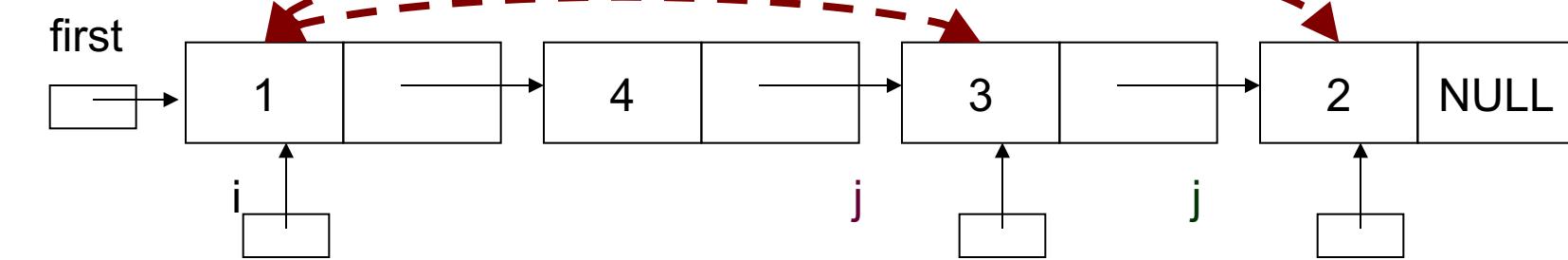
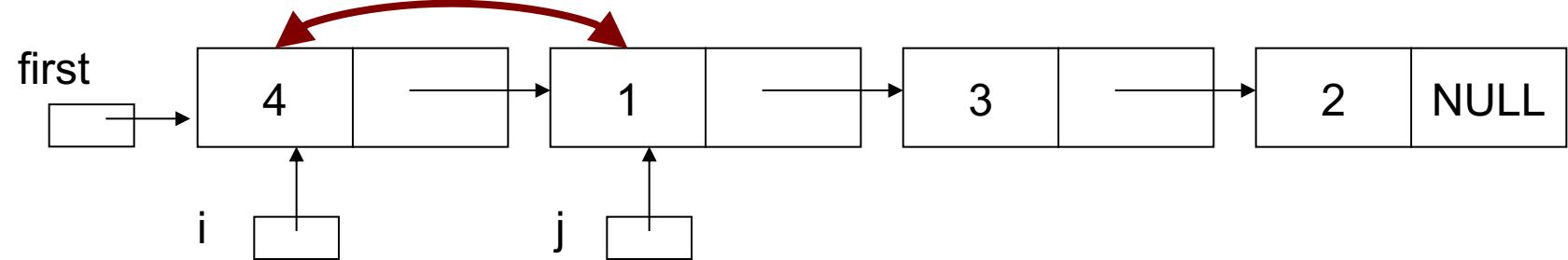
Примечание. После того как список сформирован, алгоритм его обработки не зависит от того, стек это или очередь.

Используем два указателя:

i – на элемент, который сравнивается;

j – на элемент, с которым происходит сравнение.

При перестановке меняем местами только поля данных, а не записи целиком, используя вспомогательную переменную целого типа c .



```
//установка, по возрастанию
void placing (struct node *lst)
{ int c; //для обмена значений полей
  struct node *i, *j; //указатели: какую запись сравниваем (i) и с //какой
  (j)

  i=lst;
  while(i->next!=NULL) //до предпоследней записи
  {j=i->next; //на следующей записи после i
   while(j!=NULL) //до последней записи
   {if(i->info>j->info)
     c=i->info,i->info=j->info,j->info=c; //перестановка
     j=j->next; //шаг внутреннего цикла
   }
   i=i->next; //шаг внешнего цикла
  }
}
```

Примечание. При сортировке списка эти методом указатель на начало списка не изменяется.

Задача 3. Сортировка связанного списка по убыванию методом пузырька. Используем 2 указателя: **curr** - на текущий элемент и **next** - на следующий. При перестановке снова меняем местами только поля данных, а не записи целиком.

```
void bubble (struct node *lst) //пузырек, по убыванию
{
    int c; //для обмена значений полей
    struct node *curr, *next; /*указатели: текущая запись и следующая за
    ней */
    int f; //признак завершения перестановок
    do{
        curr=lst;
        f=1;
        while(curr->next!=NULL) //до предпоследней записи
        {next=curr->next; //на следующей записи после curr
            if(curr->info<next->info) //перестановка
                c=curr->info,curr->info=next->info,next->info=c, f=0;
            curr=curr->next; //движение по списку
        }
    } while(!f); //пока есть перестановки
}
```

Задача 4. Найти значение максимального элемента списка..

```
int maxlist(struct node *lst)//возвращает максимальный из
//элементов списка – поле данных
{struct node*max=lst;
struct node *p=lst->next; //указатель на текущую запись
while(p!=NULL) //лучше while(p)
{ if(p->info>max->info) max=p;
p=p->next;
}
return(max->info);
}
```

```
int main()
{struct node*lst=stack();
 if(!lst) puts ("List is empty");
 else
{ puts("Original list");printspisok (lst);
 printf (" max = %7d", maxspisok(lst) ) ;
 free_memory(lst);
}
return 0;}
```

Задача 5. Найти максимальный нечетный элемент (функция возвращает указатель на структуру, либо NULL).

```
struct node* find(struct node*first)
{struct node *max,*curr;
max=NULL;curr=first;
while (curr)
{ if (curr->info % 2)
if (max==NULL| |curr->info>max->info)    max=curr;
curr=curr->next;
}
return max;
}
```

```
int main()
{struct node* first, *max;
first=stack();
if (first!=NULL)
{max=find(first);
 if (max==NULL) puts("Максимума не существует");
else printf("Max= %7d",max->info);
free_memory(lst);
}
else puts("List is empty");
return 0;
}
```

Задача 6. Сформировать новый стек из нечетных элементов исходного списка. Используем 2 указателя: **lst1** - на вершину формируемого списка и **p1** - на присоединяемый элемент. Для используемых в коде функций из предыдущих примеров приведены прототипы.

```
#include <stdio.h>
#include <malloc.h>
struct node
{struct node *next; int info; };

struct node *stack() ; // формирование стека - прототип
//вывод списка на экран - прототип
void printspisok (struct node *lst);
//освобождение памяти - прототип
void free_memory(struct node *lst);
```

```
// формирование нового стека
struct node* new_list(struct node *lst)
{struct node*lst1=NULL, *p1, *p=lst;
//указатель на текущую запись
while(p)
{ if(p->info%2)
 {p1=(struct node*) malloc(sizeof(struct node));
  p1->info=p->info;
  p1->next=lst1;
  lst1=p1;
 }
 p=p->next;
}
return(lst1);
}
```

```
int main()
{struct node*lst=stack(), *lst1;
 if(!lst) puts ("List is empty");
 else
 { puts("Original list");printspisok (lst);
 lst1=new_list(lst);
 if(!lst1) puts ("New list is empty");
 else
 {
 puts("New list");printspisok (lst1);
 free_memory(lst1);
 }
 puts("Original list");free_memory(lst);
}
return 0;
}
```

Работа над ошибками контрольной

Задание 1: Даны целочисленная матрица $C[0:N-1, 0:M-1]$ и целочисленный массив $D[0:K-1]$. Написать программу, которая сортирует по убыванию методом пузырька элементы тех строк матрицы C , номера которых присутствуют в массиве D . Вычисления оформить в виде функции с параметрами.

```
#include<stdio.h>
#define lmax 30
void sort(int n, int m, int k, int C[][][lmax], int D[], int *f1,int *f2)
{int i,j,x,y,flag;
 for( i=0;i<n; i++)
 {
    x=0;
    while (x<k && i!=D[x])    x=x+1;
    if (x<k)
    {  *f1=1;
       do
       { flag=1;
         for (j=0;j< m-1 ;j++)
         if (C[i][j]<C[i][j+1])
         {  y=C[i][j]; C[i][j]=C[i][j+1]; C[i][j+1]=y; flag=0;   *f2=1;  }
         }  while(! flag);
    }
}
,
```

```
int main()
{ int i, j, n, m, k, f1,f2;
int C[lmax][lmax], D[lmax];
printf("Input n, m for matrix C, 0<n<=%d, 0<m<=%d: ", lmax, lmax);
do           scanf("%d%d",&n, &m);
while (n<=0|| n>lmax|| m<=0||m>lmax);
printf("Input matrix:\n");
for (i=0;i< n ;i++)
    for (j=0;j< m ;j++)
        scanf("%d",&C[i][j]);
printf("Input the length for array D, 0<k<=30 ");
do       scanf("%d",&k);
while (k<=0|| k>lmax);
printf("Input array D:\n");
for (i=0;i< k ;i++)
    scanf("%d",&D[i]);
```

```
f1=0; f2=0;  
sort(n,m,k,C,D,&f1,&f2);  
if (f1 && f2)  
{  
    printf(" C: \n");  
    for (i=0;i< n ;i++)  
    {  
        for (j=0;j< m ;j++)  
            printf("%4d",C[i][j]);  
        printf("\n");  
    }  
  
    printf("Array D: \n");  
    for (i=0;i< k ;i++)  
        printf("%4d",D[i]);  
    printf("\n");  
}
```

```
if (!f1) printf("No such elements in D!\n");
if (!f2 && f1) printf("No changes in matrix.\n");
return 0;
}
```

Работа над ошибками контрольной

Задание 2: Данна матрица $Y[0:N-1,0:N-1]$ вещественного типа. Написать программу, которая обнуляет положительные элементы матрицы, расположенные ниже побочной диагонали, если среднее арифметическое элементов главной диагонали не меньше заданного числа А. Вычисления оформить в виде функции с параметрами.

```
#include<stdio.h>
#define Imax 30

int SrArf(int n, float Y[][][Imax], float A)
{int i;
float sr,s;

s=0;
for (i= 0;i< n; i++)
    s=s+Y[i][i];
sr=(float)(s/n);
if (sr>=A)  return 1;
else
return 0;
}
```

```
int main()
{ float Y[lmax][lmax];
int n,i,j, flag=0;
float A;

printf("Input size of matrix Y, 0<n<=30: ");
do
    scanf("%d", &n);
while (n<=0| |n>lmax);
printf("Input matrix Y: ");
for (i= 0;i< n; i++)
    for (j= 0;j< n; j++)
        scanf("%f",&Y[i][j]);

printf("Enter numberA: ");
scanf("%f",&A);
```

```
if (SrArf(n,Y,A) )
{   for (i=1;i<n; i++)
    for (j=n-i;j<n; j++)
        if (Y[i][j]>0)
        {   flag=1; Y[i][j]=0;           }
if (flag)
{ printf("Changed matrix: \n");
for (i= 0;i< n; i++)
    {for (j= 0;j< n; j++) printf("%10.5f",Y[i][j]);
printf("\n");          }
}
else printf("No changes!");
}
else
    printf("Middle does not satisfy the condition ...");
return 0;}
```