Лекция 5. Язык С. Ввод и вывод данных.

1. Форматированный ввод и вывод

В языке Си нет операторов ввода-вывода. Ввод и вывод информации осуществляется через функции стандартной библиотеки (stdlibc). Прототипы рассматриваемых функций находятся в файле stdio.h. Эта библиотека содержит прежде всего функции форматированного (текстового) ввода/вывода:

- scanf() — ввод информации с клавиатуры,
- printf() — вывод информации на экран,
- fprintf() – вывод информации в текстовый файл,
- sscanf() – ввод информации из символьного буфера,
- sprintf() – вывод информации в символьный буфер.

Также имеются специальные функции посимвольного ввода с клавиатуры и вывода на экран:
getchar(), getc(); putchar(), putc().
Также имеются специальные функции построкового ввода с клавиатуры и вывода на экран:
gets(), puts().

1.1. Вывод информации на экран

Функция printf() предназначена для форматированного вывода. Она переводит данные в символьное представление и выводит полученные изображения символов на экран. При этом у программиста имеется возможность форматировать данные, то есть влиять на их представление на экране.
Общая форма записи функции printf():

```
printf("Строка форматов", объект1, объект2, ..., объектN);
```

Строка форматов состоит из следующих элементов:

- управляющих символов;
- текста, представленного для непосредственного вывода;
- форматов, предназначенных для вывода значений переменных различных типов.

Объекты (константы и переменные) могут отсутствовать.

Управляющие символы не выводятся на экран, а управляют расположением выводимых символов. Отличительной чертой управляющего символа является наличие обратного слэша "\" перед ним.
Основные управляющие символы:

- "\n" — перевод строки;
- "\t" — горизонтальная табуляция;
- "\v" — вертикальная табуляция;
- "\b" — возврат на символ;
- "\r" — возврат на начало строки;
- "\a" — звуковой сигнал.

Форматы нужны для того, чтобы указывать вид, в котором информация будет выведена на экран. Отличительной чертой формата является наличие символа процент "%" перед ним:

- %d — целое число типа int со знаком в десятичной системе счисления;
- %u — целое число типа unsigned int в десятичной системе счисления;
- %x — целое число типа int со знаком в шестнадцатеричной системе счисления;
- %o — целое число типа int со знаком в восьмеричной системе счисления;
- %hd — целое число типа short со знаком в десятичной системе счисления;
- %hu — целое число типа unsigned short в десятичной системе счисления;
- %hx — целое число типа short со знаком в шестнадцатеричной системе счисления;
- %ld — целое число типа long int со знаком в десятичной системе счисления;
- %lu — целое число типа unsigned long int;
- %lx — целое число типа long int со знаком в шестнадцатеричной системе счисления;
- %f — вещественный формат (числа с плавающей точкой типа float);
- %lf — вещественный формат двойной точности (числа с плавающей точкой типа double);

- %Lf — вещественный формат четверной точности (числа с плавающей точкой типа long double);
- %e — вещественный формат в экспоненциальной форме (числа с плавающей точкой типа float в экспоненциальной форме);
- %c — символьный формат;
- %s — строковый формат.

Строка форматов содержит форматы для вывода значений. Каждый формат вывода начинается с символа "%". После строки форматов через запятую указываются имена переменных, которые необходимо вывести. Количество символов % в строке формата должно совпадать с количеством переменных для вывода. ***Тип каждого формата должен совпадать с типом переменной, которая будет выводиться на это место.*** Замещение форматов вывода значениями переменных происходит в порядке их следования.

Пример

```
#include <stdio.h>
int main()
{
 int a = 5;
 float x = 2.78;
 printf("a=%d\n", a);
 printf("x=%f\n", x);
 getchar();
 return 0;
}
```

Результат работы программы

```
a=5
x=2.780000
```

Тот же результат может быть получен с помощью одного вызова printf:

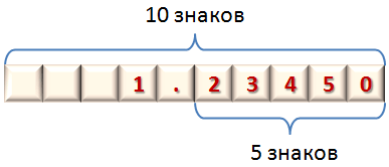
```
#include <stdio.h>
int main()
{
 int a = 5;
 float x = 2.78;
 printf("a=%d\nx=%f\n", a, x);
 getchar();
 return 0;
}
```

Табличный вывод

При указании формата можно явным образом указать общее количество знакомест и количество знакомест, занимаемых дробной частью:

```
#include <stdio.h>
int main()
{
 int a=5;
 float x = 1.2345;
 printf("Hello!\n");
 getchar();
 printf("a=%2d\n", a);
 getchar();
 printf("x=%10.5f\n", x);
 getchar();
 printf("Bye!\n");
```

```
 return 0;
}
```
Результат выполнения
```
x=   1.23450
```
В примере 10 — общее количество знакомест, отводимое под значение переменной; 5 — количество позиций после разделителя целой и дробной части (после десятичной точки). В указанном примере количество знакомест в выводимом числе меньше 10, поэтому свободные знакоместа слева от числа заполняются пробелами. Такой способ форматирования часто используется для построения таблиц.



1.2. Ввод данных с клавиатуры

Функция форматированного ввода данных с клавиатуры scanf() выполняет чтение данных, вводимых с клавиатуры, преобразует их во внутренний формат и передает вызывающей функции. При этом программист задает правила интерпретации входных данных с помощью спецификаций форматной строки. Общая форма записи функции scanf( ):
```
scanf ("Строка форматов", адрес1, адрес2,...);
```
Строка форматов аналогична функции printf(). Для формирования адреса переменной используется символ амперсанд "&": адрес = &объект

Строка форматов и список аргументов для функции обязательны.

Пример
```
#define _CRT_SECURE_NO_WARNINGS // использование scanf без ограничений
#include <stdio.h> // для использования функций ввода-вывода
#include <stdlib.h> // для перехода на русский язык
int main()
{
 float y;
 system("chcp 1251"); // переходим в консоли на русский язык
 system("cls");      // очищаем окно консоли
 printf("Введите y: "); // выводим сообщение
 scanf("%f", &y);     // вводим значения переменной y
 printf("Значение переменной y=%f", y); // выводим значение переменной y
 getchar(); getchar();
 return 0;
}
```
Результат работы программы:

Введите y: 1.345

Значение переменной y=1.345000

Функция scanf( ) является функцией незащищенного ввода, т.к. появилась она в ранних версиях языка Си. Поэтому чтобы разрешить работу данной функции в современных компиляторах необходимо в начало программы добавить строчку
```
#define _CRT_SECURE_NO_WARNINGS
```
Другой вариант — воспользоваться функцией защищенного ввода scanf_s( ), которая появилась несколько позже, но содержит тот же самый список параметров.
```
#include <stdio.h>
int main()
{
 int a;
 printf("a = ");
 scanf_s("%d", &a);
 printf("a = %d",a);
```

```
 getchar(); getchar();
 return 0;
}
```

1.3. Файловый ввод и вывод данных

Практически вся информация в запоминающих устройствах хранится в виде файлов.

Файл – именованная область внешней памяти, выделенная для хранения массива данных. Данные, содержащиеся в файлах, имеют самый разнообразный характер: программы на алгоритмическом или машинном языке; исходные данные для работы программ или результаты выполнения программ; произвольные тексты; графические изображения и т. п.

Каталог (***папка***, ***директория***) – именованная совокупность байтов на носителе информации, содержащая название подкаталогов и файлов, используется в файловой системе для упрощения организации файлов.

Файловой системой называется функциональная часть операционной системы, обеспечивающая выполнение операций над файлами. Примерами файловых систем в ОС Windows являются **FAT** (FAT – File Allocation Table, таблица размещения файлов – разновидности FAT12, FAT16, FAT32), **NTFS** (New Technology File System — файловая система новой технологии). В UNIX/Linux ОС используются другие файловые системы – **Ext2**,...,**Ext4**, **JFS** (журналируемая файловая система), **XFS** и др. Также используются ОС независимые файловые системы, например, **UDF** (Universal Disk Format, универсальный дисковый формат) — спецификация формата файловой системы, не зависящей от типа ОС и предназначенной для хранения файлов на оптических носителях.

Работа с файлами в языке Си

Для программиста файл представляется как последовательность считываемых или записываемых данных (в общем случае байтов информации). При открытии файла с ним связывается ***поток ввода-вывода***. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Когда поток открывается для ввода-вывода, он связывается со стандартной структурой типа FILE, которая определена в stdio.h. Структура FILE содержит необходимую информацию о файле.

Открытие файла осуществляется с помощью функции fopen(), которая возвращает указатель на структуру типа FILE, который можно использовать для последующих операций с файлом.

```
FILE *fopen(name, type);
```

name – имя открываемого файла (включая путь), "a.txt", "C:\WORK\a.txt", "$HOME/work/a.txt"

type — указатель на строку символов, определяющих способ доступа к файлу:

- "r" — открыть файл для чтения (файл должен существовать);
- "w" — открыть пустой файл для записи; если файл существует, то его содержимое теряется;
- "a" — открыть файл для записи в конец (для добавления); файл создается, если он не существует;
- "r+" — открыть файл для чтения и записи (файл должен существовать);
- "w+" — открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряется;
- "a+" — открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Возвращаемое значение функции fopen — указатель на открытый поток. Если обнаружена ошибка, то возвращается специальное значение NULL.

Функция fclose() закрывает поток или потоки, связанные с открытыми при помощи функции fopen() файлами. Закрываемый поток определяется аргументом функции fclose().

Возвращаемое значение: значение 0, если поток успешно закрыт; константа EOF, если произошла ошибка.

```
#include <stdio.h>
int main() {
 FILE *fp;
 char name[] = "my.txt";
 if (   (fp = fopen(name, "w"))  == NULL)
 {
```

```
  printf("Не удалось открыть файл\n");
  getchar();
  return 0;
 }
 printf("Файл открыт!\n");
 fprintf(fp,"Файл открыт!\n");
 fclose(fp);
 getchar();
 return 0;
}
```

Чтение символа из файла:

```
char fgetc(поток);
```

Аргументом функции является указатель на поток типа FILE. Функция возвращает код считанного символа. Если достигнут конец файла или возникла ошибка, возвращается константа EOF.

Запись символа в файл:

```
fputc(символ,поток);
```

Аргументами функции являются символ и указатель на поток типа FILE. Функция возвращает код считанного символа.

Функции fscanf() и fprintf() аналогичны функциям scanf() и printf(), но работают с файлами данных, и имеют первый аргумент — указатель на файл.

```
fscanf(поток, "Формат ввода", аргументы);
fprintf(поток, "Формат вывода", аргументы);
```

Ввод-вывод строк

Функции fgets() и fputs() предназначены для ввода-вывода строк, они являются аналогами функций gets() и puts() для работы с файлами.

```
fgets(УказательНаСтроку, КоличествоСимволов, поток);
```

Символы читаются из потока до тех пор, пока не будет прочитан символ новой строки "\n", который включается в строку, или пока не наступит конец потока EOF или не будет прочитано максимальное количество символов. Результат помещается в указатель на строку и заканчивается нуль- символом "\0". Функция возвращает адрес строки.

```
fputs(УказательНаСтроку,поток);
```

Копирует строку в поток с текущей позиции. Завершающий нуль- символ не копируется.

Пример Ввести число и сохранить его в файле s1.txt. Считать число из файла s1.txt, увеличить его на 3 и сохранить в файле s2.txt.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 FILE *S1, *S2;
 int x, y;
 system("chcp 1251");
 system("cls");
 printf("Введите число : ");
 scanf("%d", &x);
 S1 = fopen("S1.txt", "w");
 fprintf(S1, "%d", x);
 fclose(S1);
 S1 = fopen("S1.txt", "r");
 S2 = fopen("S2.txt", "w");
 fscanf(S1, "%d", &y);
 y += 3;
 fclose(S1);
```

```
 fprintf(S2, "%d\n", y);
 fclose(S2);
 return 0;
}
```

Результат выполнения — 2 файла

```
>cat S1.txt
3
>cat S2.txt
6
```

Ввод/вывод из строки/в строку.

Ввод или вывод можно осуществлять в оперативную память. Обычно это используется, когда необходимо сформировать логически законченный фрагмент текстовой информации. Для реализации такого способа ввода-вывода понадобится сначала определить в программе символьный буфер определенного размера. Далее можно пользоваться функциями:

```
sscanf(буфер, "Формат ввода", аргументы);
sprintf(буфер, "Формат вывода", аргументы);
```

Пример:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <strings.h>
int main()
{
 char str[80];
 int i;
 for(i=0; i<80; i++) str[i] = 0;
 sscanf("hello 1 2 3 4 5", "%s%d", str, &i);
 printf("%s %d\n", str, i);
 for(i=0; i<80; i++) str[i] = 0;
 sprintf(str,"%18.16lf",(4.0*atan(1.0)));
 printf("Computer's Pi = %s\n", str);
 return 0;
}
```

Стандартные потоки

stdin – поток ввода с клавиатуры;
stdout – поток вывода на экран;
stderr – поток вывода ошибок на экран.
Эквиваленты:

```
scanf("формат", аргументы)  <==>  fscanf(stdin, "формат", аргументы)
printf("формат", аргументы)  <==>  fprintf(stdout, "формат", аргументы)
printf("формат", аргументы)  <==>  fprintf(stderr, "формат", аргументы)
```

2. Неформатированный ввод и вывод

Стандарт языка Си описывает два вида файлов - текстовые и двоичные - хотя операционная система не требует их различать.

Текстовый файл - файл, содержащий текст, разбитый на строки при помощи разделяющего символа окончания строки (в Unix - одиночный символ перевода строки "\n"; в Microsoft Windows перед символом перевода строки следует знак возврата каретки "\r\n". При считывании байтов из текстового файла, символы конца строки обычно заменяются концом строки "\0" для упрощения последующе обработки. При записи текстового файла одиночный символ перевода строки перед записью заменяется специфичной для ОС последовательностью символов конца строки.

Двоичный файл - файл, из которого байты считываются и выводятся в <сыром> виде без какого-либо связывания (подстановки).

При открытии файла имеется возможность указать компилятору и ОС, что вводиться будут форматированные (текстовые) или неформатированные (двоичные, бинарные) данные. В первом случае специальный описатель не требуется. Во втором случае для этого зарезервировано значение параметра type равное "b", означающее двоичный режим ввода.

Для ввода/вывода двоичных данных в поток/из потока используются функции

```
size_t fread(void *ptrvoid, size_t size, size_t count, FILE * filestream );
size_t fwrite(const void *ptrvoid, size_t size, size_t count, FILE * filestream );
int fgetpos(FILE *stream, fpos_t *pos); - узнать позицию указателя
int fsetpos(FILE *stream, const fpos_t *pos); - установить указатель на позицию
int fseek(FILE *stream, long offset, int whence); - передвинуть указатель на несколько позиций
void rewind(FILE *stream); - перевод указателя на начало файла
int feof(FILE *stream);
int ferror(FILE *stream);
```

Пример:

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  FILE *fp;
  double d = 12.23;
 short  int i = 101;
  long l = 123023L;
  if ((fp=fopen("test", "wb+"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    exit(1);
  }
  fwrite(&d, sizeof(double), 1, fp);
  fwrite(&i, sizeof(short int), 1, fp);
  fwrite(&l, sizeof(long), 1, fp);
  rewind(fp);
  fread(&d, sizeof(double), 1, fp);
  fread(&i, sizeof(short int), 1, fp);
  fread(&l, sizeof(long), 1, fp);
  printf("%f %d %ld", d, i, l);
  fclose(fp);
  return 0;
}
```

3. Непотоковый ввод и вывод в файл

Наряду с потоковым вводом/выводом имеются функции примитивной работы с файлами. В этом случае используются функции open(), close(), read(), write(). В современной ситуации их использование не рекомендуется. Данные функции не включены в стандарт ANSI C.

4. Массивы и списки

1. Работа с массивами

Векторы, матрицы, тензоры

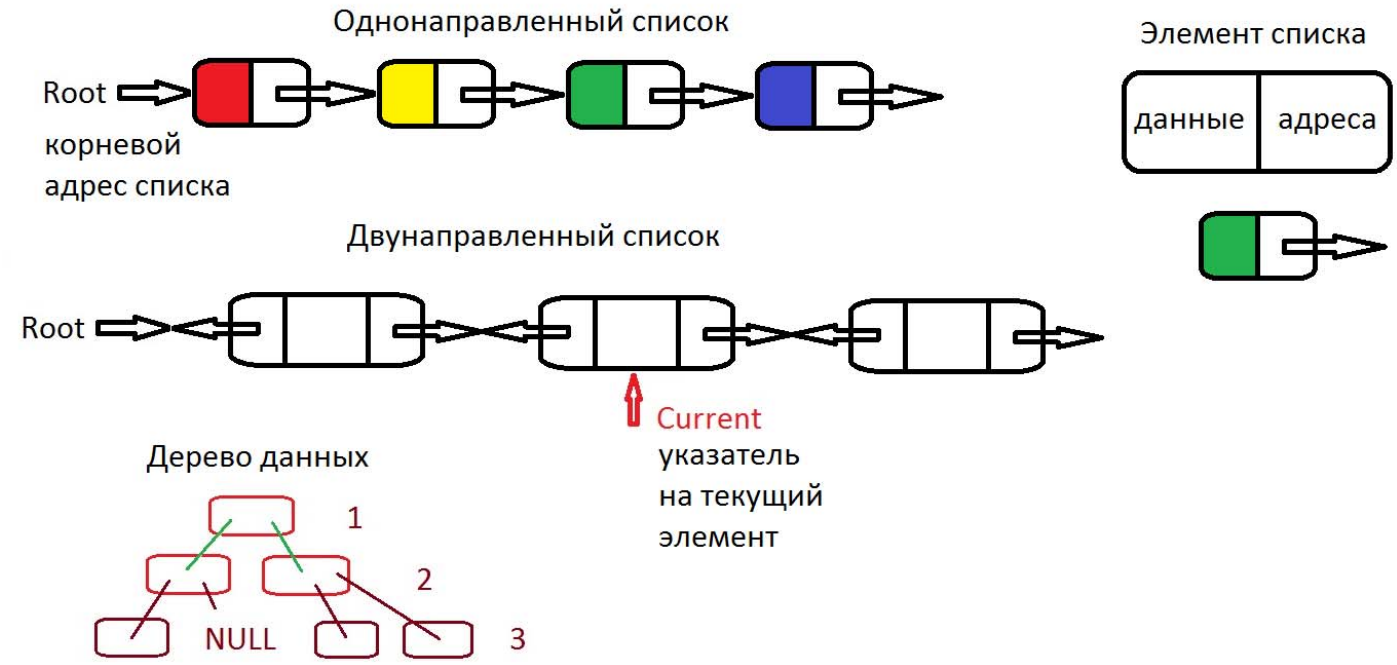
```
>cat massiv.c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
// Указатели на массивы
  double **A; // Матрица системы Ax=b
  double *B; // Вектор правой части
  double *X; // Вектор неизвестных
  double *C; // Матрица системы в развернутом в строку виде
  int i, j, ij, n;
  char ch = 0;

  Met:
  printf("Input massiv dimension ");
  scanf ("%d",&n);
  printf("n=%d\n", n);

  // Выделение памяти для массивов
  A = (double**)(malloc(sizeof(double*)*n));
  for (i=0; i<n; i++)
    A[i] = (double*)(calloc(n, sizeof(double)));

  B = (double*)(malloc(sizeof(double)*n));

  X = (double*)(calloc(n, sizeof(double)));
  C = (double*)(calloc(n*n, sizeof(double)));

  // Заполнение элементов матрицы
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      if (j==i) A[i][j] = 1.0*(i+1);
      else      A[i][j] = -0.1/(i+j);
      ij = i*n + j;
      C[ij] = A[i][j];
    }
  }

  // Заполнение элементов вектора правой части
  for (i = 0; i < n; i ++) {
    double el;
    printf("Input element with number %d = ", i);
    scanf("%lf",&el);
    B[i] = el;
  }

  // Вывод элементов матрицы
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      printf ("A[%d][%d]=%lf\n", i, j, A[i][j]);
    }
  }

// Вывод элементов вектора правой части
  for (i = 0; i < n; i ++) {
    printf ("B[%d]=%lf\n", i, B[i]);
  }

// Решение системы:
...

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

```
    printf ("X[%d]=%lf\n", i, X[i]);
  }

  for (i=0; i<n; i++) free(A[i]);
  free(A);

  free(B);
  free(X);
  free(C);

  printf("\n");
  ch = getchar();
  printf("Continue?(y|n): ");

  ch = getchar();
  printf("\n");

  if (ch == 'y') {
    goto Met;
  }

  return 0;
}
```

2. Списки и деревья данных

Список – набор данных, элементами которого являются данные и их адреса.

Дерево отличается от списка наличием нелинейной структуры связей (графа связей).



Пример

```
>cat lists.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>

typedef struct Abonent_t {
  char FirstName[20];
  char LastName[20];
  char PhoneCode[4];
  char PhoneNumber[11];
  struct Abonent_t* NextLeft;
  struct Abonent_t* NextRight;
} Abonent_t;
```

```
typedef struct Book_t {
  int Length;
  struct Abonent_t* First;
} Book_t;

void PrintBook(Book_t Book)
{
  int i;
  struct Abonent_t* Ab;
  struct Abonent_t  Abb;
  struct Abonent_t* Abbb = &Abb;

Ab->Next = NULL;

Abb.Next = NULL;
Abbb->Next = NULL;

  printf("Book length is %d\n", Book.Length);

  if (Book.Length > 0) {
    for (i=0; i<Book.Length; i++){
      if (i==0) Ab = Book.First;
      else      Ab = Ab->Next;
      printf("Abonent %d is %s %s %s %s\n",
        (i+1), Ab->FirstName, Ab->LastName, Ab->PhoneCode, Ab->PhoneNumber);
    }
  }

  return;
}

int main() {
  int i, n;

  Book_t Book;
  Abonent_t *Ab, *Ac;

  double ddd, fff;
  double *eee;
  double *aaa, *bbb;

eee = &ddd;
*eee = 12.34;

eee = &fff;
*eee = 35.67;

  Ab = (Abonent_t*)calloc(1, sizeof(Abonent_t));
  strncpy(Ab->FirstName,"Ivan",4);
  strncpy(Ab->LastName,"Petrov",6);
  strncpy(Ab->PhoneCode,"007",3);
  strncpy(Ab->PhoneNumber,"4953222231",10);
  Ab->Next = NULL;

  Book.Length = 1;
  Book.First  = Ab;

  PrintBook(Book);

  Ab->Next = (Abonent_t*)calloc(1, sizeof(Abonent_t));
  Ac = Ab->Next;
  strncpy(Ac->FirstName,"Nikolay",7);
  strncpy(Ac->LastName,"Sidorov",7);
  strncpy(Ac->PhoneCode,"007",3);
  strncpy(Ac->PhoneNumber,"4998445416",10);
  Ac->Next = NULL;

  Book.Length = 2;
```

```
  PrintBook(Book);

  Ac = (Abonent_t*)calloc(1, sizeof(Abonent_t));
  strncpy(Ac->FirstName,"Petr",4);
  strncpy(Ac->LastName,"Grigoriev",9);
  strncpy(Ac->PhoneCode,"001",3);
  strncpy(Ac->PhoneNumber,"3458995416",10);
  Ac->Next = NULL;

  Ab = Book.First; // first element
  Ab = Ab->Next;   // second element
  Ab->Next = Ac;   // link to third element

  Book.Length = 3;

  PrintBook(Book);

  Ac = NULL;
  for (i=0; i<Book.Length; i++){
    if (i==0) Ab = Book.First;
    else      Ab = Ab->Next;

    if ((i+1) == 2) {
      Ac->Next = Ab->Next;
      free(Ab);
      Ab = NULL;
      Book.Length = 2;
      break;
    }

    Ac = Ab; // Previous element
  }

  PrintBook(Book);

  return 0;
}
```

3. Использование массивов. Метод Гаусса
Задача

Пусть исходная система выглядит следующим образом:

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n & = & b_1 \\ \ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n & = & b_m \end{cases}$$

Её можно записать в матричном виде:

$$Ax = b,$$

где

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \ldots & & \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}. \quad (1)$$

Матрица A называется основной матрицей системы, b — столбцом свободных членов.

Необходимо найти решение – вектор **x**.
Идея Гаусса – с помощью эквивалентных линейных преобразований привести матрицу системы к верхнему треугольному виду:

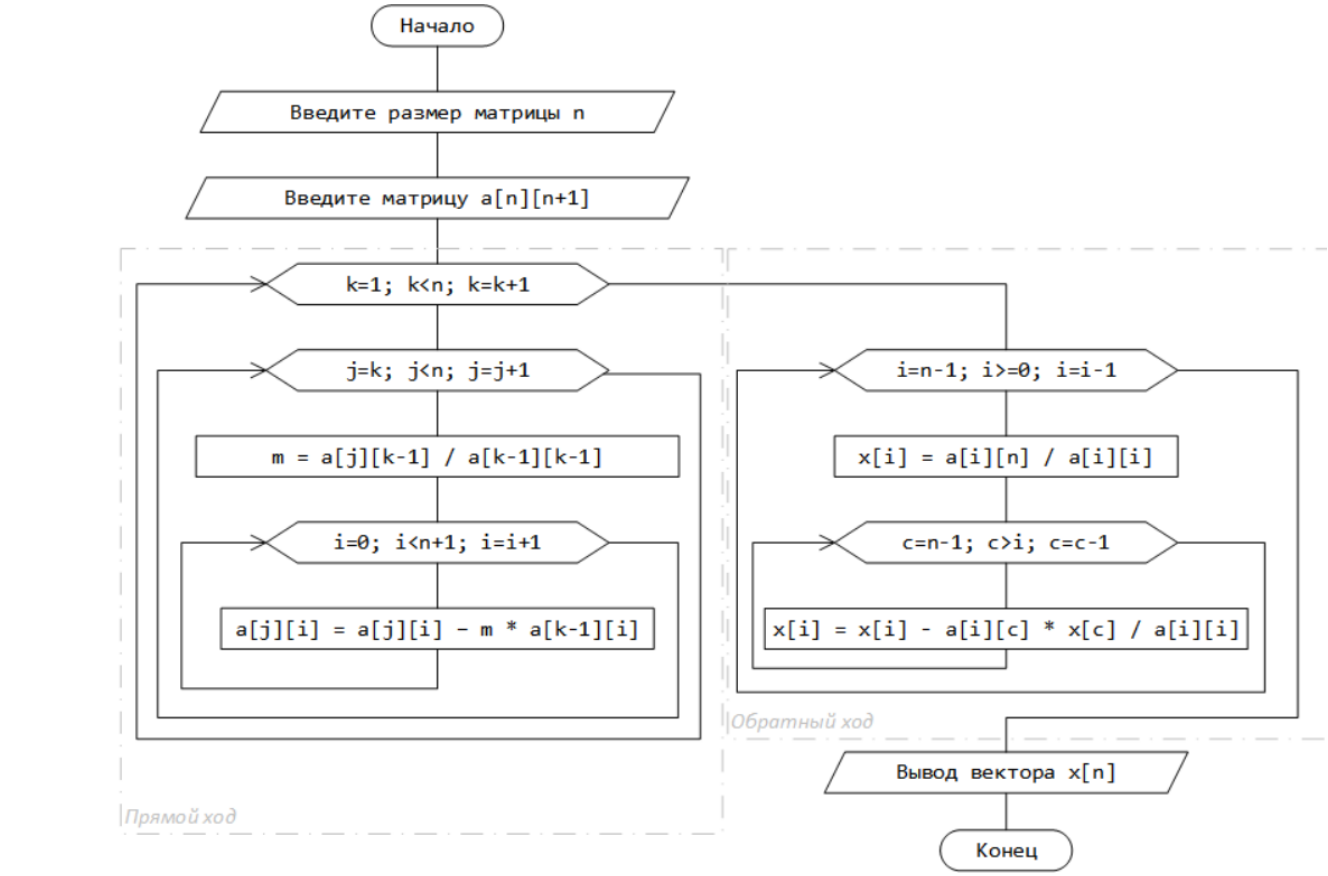
$$\begin{cases} \alpha_{1j_1}x_{j_1} + \alpha_{1j_2}x_{j_2} + \ldots + \alpha_{1j_r}x_{j_r} + \ldots + \alpha_{1j_n}x_{j_n} = \beta_1 \\ \alpha_{2j_2}x_{j_2} + \ldots + \alpha_{2j_r}x_{j_r} + \ldots + \alpha_{2j_n}x_{j_n} = \beta_2 \\ \ldots \\ \alpha_{rj_r}x_{j_r} + \ldots + \alpha_{rj_n}x_{j_n} = \beta_r \\ 0 = \beta_{r+1} \\ \ldots \\ 0 = \beta_m \end{cases}$$

Алгоритм Гаусса

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа.

На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к треугольной форме, либо устанавливают, что система несовместна. Для этого среди элементов первого столбца матрицы выбирают ненулевой, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают, пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.

На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по <ступенькам> наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.



Блок-схема алгоритма Гаусса.

Программа

```
>cat massiv2.c
#include <stdio.h>
#include <stdlib.h>

void print_system(int n, double **A, double *X);

int main ()
```

```
{
// Указатели на массивы
  double **A; // Матрица системы
  double *B; // Вектор правой части
  double *X; // Вектор неизвестных
  double *C; // Матрица системы в развернутом в строку виде
  int i, j, k, ij, n, ier;
  double s, p;
  char ch = 0;

  Met:
  printf("Input massiv dimension ");
  scanf ("%d",&n);
  printf("n=%d\n", n);

  // Выделение памяти для массивов
  A = (double**)(malloc(sizeof(double*)*n));
  for (i=0; i<n; i++)
    A[i] = (double*)(calloc(sizeof(double),n));

  B = (double*)(malloc(sizeof(double)*n));

  X = (double*)(calloc(sizeof(double),n));
  C = (double*)(calloc(sizeof(double*),n*n));

  // Заполнение элементов матрицы
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      if (j==i) A[i][j] = 1.0*(i+1);
      else      A[i][j] = -0.1/(i+j);
      ij = i*n + j;
      C[ij] = A[i][j];
    }
  }

  // Заполнение элементов вектора правой части
  for (i = 0; i < n; i ++) {
    double el;
    printf("Input element with number %d = ", i);
    scanf("%lf",&el);
    B[i] = el;
  }

  // Вывод элементов матрицы
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      printf ("A[%d][%d]=%lf\n", i, j, A[i][j]);
    }
  }

  // Вывод элементов вектора правой части
  for (i = 0; i < n; i ++) {
    printf ("B[%d]=%lf\n", i, B[i]);
  }
  printf("\n");

  // Решение системы:
  ier =0;

  for (i=0; i<n; i++) X[i] = B[i];
  print_system(n,A,X);

  for (i=0; i<n; i++) {
    s = A[i][i];
    if (s!=0) {
      s = 1.0/s;
      A[i][i] = 1.0;
      for (j=i+1; j<n; j++) A[i][j] *= s;
```

```
      X[i] *= s;
      if (i<n-1){
        for (k=i+1; k<n; k++){
          p = A[k][i];
          A[k][i] = 0;
          for (j=i+1; j<n; j++) A[k][j] -= (p*A[i][j]);
          X[k] -= (p*X[i]);
        }
      }
      print_system(n,A,X);
    }
    else {
      ier = -1;
      printf("Bad matrix!\n");
      break;
    }
  }

  if (ier == 0){
    for (i=(n-2); i>=0; i--) {
      s = 0;
      for (j=i+1; j<n; j++) s += (A[i][j]*X[j]);
      X[i] -= s;
    }
    print_system(n,A,X);
  }

  // Проверка решения:
  if (ier == 0){
    for (i=0; i<n; i++) {
      s = B[i];
      for (j=0; j<n; j++) {
        ij = i*n + j;
      s -= (C[ij]*X[j]);
      }
      printf("i=%d s=%le\n",i,s);
    }
  }

  // Освобождение памяти:
  for (i=0; i<n; i++) free(A[i]);
  free(A);
  free(B);
  free(X);
  free(C);

  printf("\n");
  ch = getchar();
  printf("Continue?(y|n): ");

  ch = getchar();
  printf("\n");

  if (ch == 'y') {
    goto Met;
  }

  return 0;
}

void print_system(int n, double **A, double *X)
{
  int i, j;
  for (i=0; i<n; i++){
    for (j=0; j<n; j++) printf("%14.6le ",A[i][j]);
    printf("%14.6le\n",X[i]);
  }
  printf("\n");
```

```
  return;
}
```