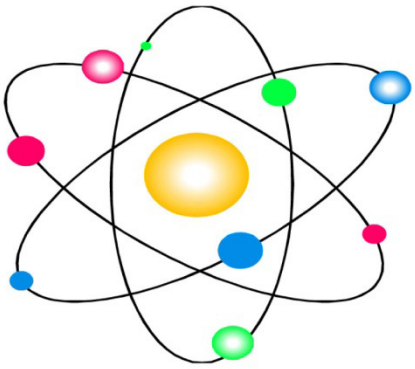
Министерство образования и науки Хабаровского края

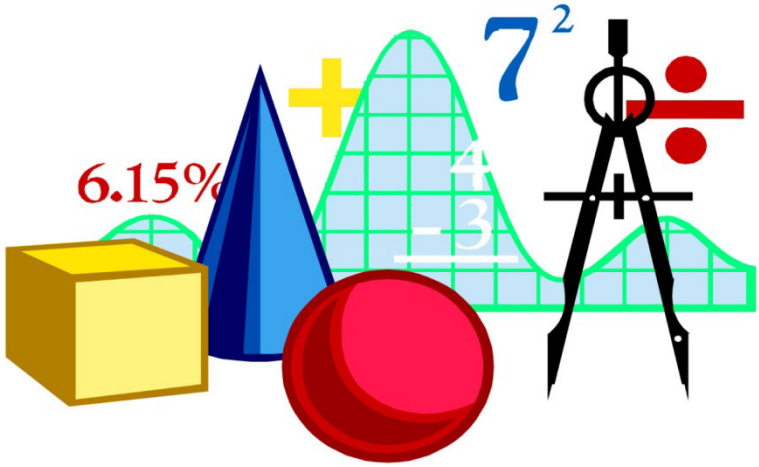
Краевое государственное бюджетное образовательное учреждение
дополнительного образования
«Хабаровский краевой центр развития творчества детей и юношества»

Хабаровская краевая заочная физико-математическая школа



Е.А. Редько, С.В. Летучий, А.В. Крупский.

ИНФОРМАТИКА:
методическое пособие



Хабаровск
2017

Е.А. Редько, С.В. Летучий, А.В. Крупский.

ИНФОРМАТИКА: методическое пособие

учебно-методическое пособие для студентов-бакалавров педагогических направлений подготовки и учащихся старших классов общеобразовательных учреждений

Подписано к печати 17.07.2017 г. Формат 60x84 1/16.
Тираж 100 экз.

КГБОУ ДО «Хабаровский краевой центр развития творчества детей и юношества».
680000, г. Хабаровск, ул. Комсомольская, 87

СОДЕРЖАНИЕ

Редько Е.А., старший преподаватель
кафедры математики и информационных технологий
ФГБОУ ВО ПИ ТОГУ, г. Хабаровск

ОРГАНИЗАЦИЯ ФАЙЛОВОГО ВВОДА-ВЫВОДА В РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

1. Общие понятия

Обязательным требованием к решению олимпиадных задач по информатике является чтение исходных данных из файла и запись результата решения в файл. Именно поэтому каждому начинающему участнику олимпиад по информатике в первую очередь необходимо научиться реализации файлового ввода-вывода в одном из языков программирования. В данном текстовом материале будут рассмотрены способы организации работы с файлами в языках *Pascal, C, C++, Python* на наиболее типовых примерах.

Файл – это способ хранения данных на физическом носителе и рассматривается с точки зрения операционной системы как последовательность байтов. Данные интерпретируются тем, кто их считывает. В программе файл связывается с файловой переменной, и все дальнейшие действия с ней реализуют операции с физическим файлом на диске.

Для организации обмена данными с файлом из программы необходимо выполнить последовательно 4 этапа:

1) создание файловой переменной;
2) связывание файловой переменной и физического файла на дисковом пространстве и открытие файла в определенном режиме;
3) обмен данными с файлом;
4) закрытие файла.

Рассмотрим, какие особенности при работе с файлами возникают в различных языках.

2. Основные этапы работы с файлами в языке С

В языке программирования *С* различают два вида файлов – текстовые и бинарные. Рассмотрим на отдельных примерах способы обработки бинарных и текстовых файлов. Различие между ними определяется вариантом интерпретации содержащихся в файле данных. Из бинарного файла можно читать данные различных типов: числовые (целые и вещественные), символьные, структуры. Тогда как из текстового файла можно читать только строки или символы. Для работы с файлами необходимо подключить библиотеку *stdio.h,* содержащую все средства стандартного ввода-вывода *(**ST**andar**DI**nput/**O**utput).*

Пример 1. Чтение целого числа из файла f1.txt и запись удвоенного значения в файл f2.txt.

```
#include <stdio.h>

int main()
{
	FILE *f_in, *f_out;  //1 этап
	int p;
	f_in=fopen("c:\\data\\f1.txt","rb");  //2 этап
	f_out=fopen("c:\\data\\f2.txt","wb");
	fscanf(f_in,"%d",&p);  //3 этап
	fprintf(f_out,"%d, ",2*p);
	close(f_in);  //4 этап
	close(f_out);
	return 0;
}
```

Комментарий к листингу программы примера 1:

1. **FILE** – это название структуры, содержащей информацию о файле.
2. Функция **fopen()** возвращает в указатель адрес файла (т.е. связывает физический файл и файловую переменную), если открытие файла состоялось. В противном случае в указатель будет записано значение **NULL**.
3. Параметрами функции **fopen()** являются две строковые константы: первая – полный путь к файлу, вторая – режим доступа и способ интерпретации. Режим доступа к файлу определяется следующими спецификаторами (таблица 1):

Таблица 1

"r"	Режим открытия файла для чтения. Файл должен существовать.
"w"	Режим создания пустого файла для записи. Если файл с таким именем уже существует его содержимое стирается, и файл рассматривается как новый пустой файл.
"a"	Дописать в файл. Операция добавления данных в конец файла. Файл создается, если он не существует.
"r+"	Режим открытия фала для обновления чтения и записи. Этот файл должен существовать.
"w+"	Создаёт пустой файл для чтения и записи. Если файл с таким именем уже существует его содержимое стирается, и файл рассматривается как новый пустой файл.
"a+"	Открыть файл для чтения и добавления данных. Все операции записи выполняются в конец файла, защищая предыдущее содержания файла от случайного изменения. Вы можете изменить позицию (FSEEK, перемотка назад) внутреннего указателя на любое место файла только для чтения, операции записи будет перемещать указатель в конец файла, и только после этого дописывать новую информацию. Файл создается, если он не существует.

Для того чтобы открыть двоичный файл, символ **b (binary)** должен быть включен в режим доступа. Этот дополнительный символ **b** может быть добавлен в конец строки, что даёт следующие режимы доступа к бинарным файлам: **rb, wb, ab, r+b, w+b, a+b** или может быть вставлен между буквой и знаком +, в случае со смешанными режимами: **rb+, wb+, ab+**.

4. Функции **fscanf()** и **fprintf()** организуют соответственно чтение из файла и запись в файл. Работают они точно так же, как функции **scanf()/printf()** за исключением того, что считывают/записывают информацию из потока/в поток, указанного первым аргументом, а не из **stdin/stdout** (стандартных потоков, связанных с клавиатурой и экраном).

Задания:

A. Предусмотрите в программе обработку ситуации, когда файл при открытии не найден. Программа должна выводить на экран сообщение об этом для пользователя.

B. Организуйте чтение из файла массива целых чисел.

C. Организуйте чтение из файла массива вещественных чисел.

D. Организуйте запись в файл f2.txtмаксимального из чисел, хранящихся в фале f1.txt.

E. Дан файл f, содержащий 20 чисел. Разделить его на два файла по 10 элементов каждый.

F. Дано два числовых файла одинаковой длины. Сформировать третий файл из по парных сумм элементов первого и второго файлов.

Пример 2. Чтение строки из файла f1.txt и запись первого и последнего символа в файл f2.txt.

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

int main()
{
        FILE *f_in, *f_out;  //1 этап
        char s[70];
        f_in=fopen("c:\\data\\f1.txt","rt");  //2 этап
        f_out=fopen("c:\\data\\f2.txt","wt");
        fgets(s,70,f_in); //3 этап
        fputc(s[0],f_out);
        fputc(s[strlen(s)-1],f_out);
        close(f_in);  //4 этап
        close(f_out);
        return 0;
}
```

Комментарий к листингу программы примера 2:

1. При открытии файлов функцией **fopen()** в спецификаторе режима доступа можно в явном виде указать, что файл будет интерпретироваться как текстовый, добавив символ **t**.
2. Чтение из файла строки, возможно содержащей пробел, необходимо осуществлять с помощью функции **fgets()**, которая признаком конца строки считает только символ начала новой строки.
3. В данном примере рассмотрена функция **fputc()**, которая используется для записи единичного символа в файл.

Задания:

A. Дан текстовый файл f1.txt. Вывести на экран содержимое файла построчно.

B. Дан файл f1. Переписать в файл f2 в «перевёрнутом» виде строки файла f1.

C. Дан файл f1. Создать новый файл, включая в него строки без первого слова.

D. Дан файл f2. Переписать его, занося лишь строки, начинающиеся с буквы «А».

E. Дан файл f1. Переписать его, исключая из строк заданное слово.

Пример 3. Запись в файл массива данных

```
#include <stdio.h>

int main()
        {
        FILE *f_out=fopen("c:\\data\\f1.txt", "wb");
        char mas[] = { 'w' , 't' , 'f' };
        fwrite(mas, 1, sizeof(mas), f_out);
        fclose (f_out);
        return 0;
        }
```

Комментарий к листингу программы примера 3:

1. Созданный файл f1.txt хранит содержимое массива **mas[].**
2. Для простоты, в массив записаны символы типа **char**, поэтому второй параметр функции **fwrite()** имеет значение =1 (байт) – такой объем занимает символ в памяти. Также записываемый массив может содержать любой другой тип данных.
3. Параметр **sizeof(mas)** — длина массива в байтах (в данном случае это три, так как массив состоит из трех элементов, каждый элемент занимает один байт).

Задание:

Записать в файл f1 сведения об именах и датах рождения учеников 7«А» класса. Переписать в файл f2 учеников, родившихся весной.

3. Основные этапы работы с файлами в языке С++

Для работы с файлами необходимо подключить библиотеку <*fstream*>, в которой определены *несколько* классов и подключены заголовочные файлы <*ifstream*> — файловый ввод и <*ofstream*> — файловый вывод.

Тогда для реализации 1-го этапа будем создавать объекты **разных** классов: *ofstream/ifstream*, в зависимости от дальнейших действий с файлом: запись/чтение.

Рассмотрим решение примеров 1-3 через объекты "файловых" классов.

Пример 1. Чтение целого числа из файла f1.txt и запись удвоенного значения в файл f2.txt.

Вариант 1.

```
#include <fstream>
using namespace std;

int main()
{
ifstreamf_in("c:\\data\\f1.txt");
ofstreamf_out("c:\\data\\f2.txt");
//1 и 2 этап
        int p;
        f_in>>p;  //3 этап
        f_out<<2*p;
        f_in.close();  //4 этап
        f_out.close();
        return 0;
        }
```

Вариант 2.

```
#include <fstream>
using namespace std;

int main()
{
        ifstreamf_in;   //1 этап
        ofstreamf_out;
f_in.open("c:\\data\\f1.txt");
f_out.open("c:\\data\\f2.txt");
//2 этап
        int p;
        f_in>>p;  //3 этап
        f_out<<2*p;
        f_in.close();  //4 этап
        f_out.close();
        return 0;
        }
```

Комментарий к листингу программы примера 1:

1. Вариант 1. Объявление объекта-файла и связывание с физическим файлом может быть реализовано одновременно (при этом происходит и открытие файла). Успешность открытия можно проверить функцией **is_open(),** которая возвращает логическое значение.
2. Вариант 2. Объявление объекта-файла можно выполнить отдельно. Тогда для связывания с физическим файлом и открытия необходимо использовать метод класса – **open().**
3. Чтение из потока и запись в поток выполняется с помощью перегруженных операторов << и >> (сдвиг влево/вправо).

Пример 2. Чтение слова и строки из файла f1.txt.

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

using namespace std;

int main()
{
        ifstreamf_in;   //1 этап
        f_in.open("c:\\data\\f1.txt");   //2 этап
        char s[70];
        f_in>>s;  //3 этап
        cout<<s<<endl;
        f_in.getline(s, 70);
        cout<<s<<endl;
        f_in.close();  //4 этап
        f_out.close();
        return 0;
        }
```

Комментарий к листингу программы примера 2:

1. Операторы << и >> интерпретируют данные в зависимости от объявленного типа: в примере 1 – число, в примере 2 – строка.
2. Первое считывание из файла оператором >> производит чтение СЛОВА (т.е. до пробела).
3. Второе считывание методом **getline()** будет читать оставшуюся часть строки длиной 70 символов, либо пока не встретит символ конца строки \n.

Задание:

Дан файл f2. Создать новый файл, включая в него строки без первого слова.

Пример 3. Чтение из файла f1.txt данных в массив. Запись в файл f2.txt массива чисел

```
#include <fstream>
#include <iostream>
using namespace std;

int main()
{
        ifstreamf_in("c:\\data\\f1.txt");
        ofstreamf_out("c:\\data\\f2.txt");
        int mas[70];
        inti=0;
        while (!f_in.eof())
{
```

```
        f_in>>mas[i];  //3 этап
        f_out<<mas[i]<<"\t";
        i++;
}
        f_in.close();  //4 этап
        f_out.close();
        return 0;
}
```

Комментарий к листингу программы примера 3:

Сравнивая примеры 3 и 4, можем сделать вывод, что для **строки** запись в файл может осуществляться как посимвольно: **f_out<<s[i]**, так и целиком: **f_out<<s**. В случае же массива элементов другого типа (не символьного) имя массива будет отображать только адрес. Поэтому запись уже сформированного в памяти массива в файл может быть реализована циклом:

for (inti=0; i<n; i++) f_out<<a[i]<<''\t'';

Задание:

Разработать программу, которая, используя операцию **sizeof()**, будет вычислять характеристики основных типов данных в*C++* и записывать их в файл. Характеристики: число байт, отводимое под тип данных; максимальное значение, которое может хранить определённый тип данных (либо диапазон).

4. Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе *ios_base* предусмотрены константы, которые определяют режим открытия файлов (см. таблицу 2).

Таблица 2

Константа	Описание
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции *open()*. **Рассмотрим эти два варианта,** открывая файл для добавления информации к концу файла:

a) ***ofstreamfout("cppstudio.txt", ios_base::app);***

b) ***fout.open("cppstudio.txt", ios_base::app);***

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **или** |, например: *ios_base::out* | *ios_base::trunc* — открытие файла для записи, предварительно очистив его.

Объекты класса ofstream, при связке с файлами по умолчанию содержат режимы открытия файлов *ios_base::out* | *ios_base::trunc*. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи.

Объекты класса *ifstream* связываясь с файлом, имеют по умолчанию режим открытия файла *ios_base::in* — файл открыт только для чтения.

Обратите внимание на то, что флаги *ate* и *app* по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг *app* позволяет производить запись только в конец файла, а флаг *ate* просто переставляет флаг в конец файла и не ограничивает места записи.

5. Основные этапы работы с файлами в языке Pascal

Все базовые средства работы с файлами в языке *Pascal* являются средствами самого языка и не требуют для своего использования подключения дополнительных модулей (библиотек).

В *Pascal* различают три типа файлов:

- текстовые (text);
- типизированные (fileof…);
- нет ипизированные (file).

В зависимости от указанного при описании типа файла, будут различаться способы обмена данными с ним.

Пример 1. Чтение целого числа из файла f1.txt и запись удвоенного значения в файл f2.txt.

```
program fvv_1;
var f1,f2: file of integer;  //1 этап
p,t:integer;
begin
        assign(f1,'c:\data\f1.txt');  //2.1 этапдля f1
        rewrite(f1);   //2.2 этапдля f1
        read(p);  //3 этапдля f1
        write(f1,p);
        close(f1);   //4 этапдля f1
        assign(f2,'c:\data\f2.txt');   //2.1 этапдля f2
        rewrite(f2);   //2.2 этапдля f1
        reset(f1);   //2.2 этапдля f2
        read(f1,t);   // 3 этапдля f1
        write('chisloizfajla f1:',t ,chr(13));
        write(f2,2*t);   //3 этапдля f2
        close(f1);  //4 этап
        close(f2);
```

```
        reset(f2);
        read(f2,p);
        write('chisloizfajla f2:',p);
        close(f2);
end.
```

Комментарий к листингу программы примера 1:

1. Проверить содержимое типизированного файла, открыв его в операционной системе, невозможно (интерпретация файла ОС отличается от способа представления данных). Поэтому будем использовать чтение содержимого файла и вывод на экран.
2. Реализация второго этапа требует применения двух процедур: **assign()** – связывает файловую переменную с физическим файлом на диске, и **reset()/rewrite()** – открывает для чтения/записи.
3. Чтение/запись в случае с типизированным файлом требует использования для обмена данными с программой переменной соответствующего типа: *в примере 1 объявлен файл целых чисел и для обмена данными с файлом используется переменная целого типа*.
4. Для чтения/записи используются стандартные операторы **read()/write()**, но необходимо ПЕРВОЙ в списке параметров указать файловую переменную.

Пример 2. Чтение строки из файла f1.txt и запись первого и последнего символа в файл f2.txt.

```
program fvv_2;
var f1,f2: text;  //1 этап
s, slovo, 11troke: string[70];
begin
        assign(f1,'c:\data\f1.txt');  //2.1 этапдля f1
        assign(f2,'c:\data\f2.txt');   //2.1 этапдля f2
        rewrite(f2);   //2.2 этапдля f1
        reset(f1);   //2.2 этапдля f2
        read(f1,s);   // 3 этапдля f1
        write(f2,s[1],s[length(s)]);   //3 этапдля f2
        close(f1);  //4 этап
        close(f2);
end.
```

Комментарий к листингу программы примера 2:

1. Текстовые файлы могут быть интерпретированы операционной системой именно так, как представлены в них данные – строки. Поэтому будем проверять работу программы, открывая файлы из ОС.
2. Чтение процедурой **read()** производится либо в соответствии с размерностью строки, указанной при описании (в примере длина строки составляет 70 символов), либо до символа конца строки.

Задания:

A. Дан файл f1. Переписать его, исключаяиз строк заданное слово.

B. Дан файл f1, ключевое слово a$ и число k . Переписать исходный файл, циклически сдвигая ключевое слово на k букв.

C. Дан файл f1. Переписать его, заменяя «abc» на «абв».

D. Дан файл f1. Переписать в «перевёрнутом» виде его строки.

E. Дан файл f1. Создать новый файл, включая в него все слова–перевёртыши из исходного файла.

F. Дан файл f1. Создать новый файл, включая в него строки без первого слова.

Пример 3. Чтение из файла f1.dat данных в массив. Запись в файл f2.dat массива чисел.

```
Program fvv_3;
type mas=array[1..100] of integer;
var f1,f2: file of mas;  //1 этап
a, t:mas;
n,i:integer;
begin
        writeln('введите размерность массива');
        readln(n);
        writeln('вводите элементы массива');
        for i:=1 to n do readln(a[i]);
        assign(f1,'c:\data\f1.dat');  //2.1 этапдля f1
        rewrite(f1);   //2.2 этапдля f1
        write(f1,a);
        close(f1);   //4 этапдля f1
        assign(f2,'c:\data\f2.dat');   //2.1 этапдля f2
        rewrite(f2);   //2.2 этапдля f1
        reset(f1);   //2.2 этапдля f2
        read(f1,t);   // 3 этапдля f1
        writeln('элементы массива из файла f1');
        for i:=1 to n do write(t[i] ,chr(13));
        write(f2,t);   //3 этапдля f2
        close(f1);  //4 этап
        close(f2);
end.
```

Комментарий к листингу программы примера 3:

Таким же образом в файл можно записывать и другие структуры данных.

Задания:

A. Записать в файл f1 сведения о книгах, изданных в период с 1998 по 2001г. в различных издательствах. В файл f2 переписать сведения о книгах, изданных в 2000 году.

В. Сформировать файл f1, содержащий названия команд – участниц чемпионата по футболу, и файл f2, содержащий результаты игр в виде матрицы (2 – выигрыш, 1 – ничья, 0 – проигрыш).

6. Основные этапы работы с файлами в языке PYTHON

В скриптовых языках *perl, php, python, ruby* отсутствует возможность потокового ввода данных, свойственного для языков программирования *Pascal,C* и *C++,* то есть в этих языках нельзя простыми методами считать из входного файла последовательность целых чисел (действительных чисел, строк и т.д.), если неизвестно, находятся ли эти числа в одной строке или в разных строках.

Для ввода данных в этих языках предлагается считывать содержимое всего файла сразу в строковую переменную, а затем использовать стандартную функцию *split* для разбивки этой переменной на поля.

Функция *split* игнорирует все начальные и конечные пробельные символы (то есть пробелы, символы табуляции и символы новой строки), а все прочие символы разбиваются на последовательности, состоящие из не пробельных символов, пробельные символы являются разделителями для полученных последовательностей. В результате получается список, состоящий из строк не пробельных символов, содержащихся в исходном файле. Например, если в исходном файле записана строка «1 2 abc» (количество пробелов неважно), то в полученном списке будет три элемента –«1», «2» и «abc».

Пример 1. Чтение двух чисел из файла f1.txt и запись их суммы в файл f2.txt.

```
deffvv():
        fin=open("c:\\data\\f1.txt","r")
        fout=open("c:\\data\\f2.txt","w")
        InputData = fin.read().split()
        Answer = int(InputData[0]) + int(InputData[1])
        print >>fout, Answer
        fin.close()
        fout.close()
```

Задание:

В файле mas.in записаны целые числа по модулю не превышающие 32000, в файл max_mas.out нужно вывести максимальное из них.

Пример 2. Чтение строки из файла f1.txt и запись ее в файл f2.txt.

```
deffvv():
        fin=open("c:\\data\\f1.txt","r")
        fout=open("c:\\data\\f2.txt","w")
```

```
s=fin.read()
print(s,file=fout)
fin.close()
fout.close()
```

Задание:

В файле text.in записаны строки, в файл rez.out нужно вывести самую длинную из них.

7. Справочник некоторых функций в языке C/C++

а) Прототип функцииfgets:

char *fgets(char *string, intnum, FILE *filestream);

Описание

Функция *fgets* считывает символы из потока и сохраняет их в виде строки в параметр *string* до тех пор пока не наступит конец строки или пока не будет достигнут конец файла.

Символ новой строки прекращает работу функции *fgets*, но он считается допустимым символом, и поэтому он копируется в строку *string*.

Нулевой символ автоматически добавляется в строку *str* после прочитанных символов, чтобы сигнализировать о конце строки.

Таблица 3. Параметры

string	Указатель на массив типа char, в который сохраняются считанные символы.
Num	Максимальное количество символов для чтения, включая нулевой символ.
filestream	Указатель на объект типа FILE, который идентифицирует поток, из которого считываются символы. Для чтения из стандартного ввода, stdin может быть использован в качестве этого параметра.

Возвращаемое значение

В случае успеха, функция возвращает указатель на string. Если конец файла был,достигнут и ни один символ не был прочитан, содержимое*string* остается неизменными и возвращается нулевой указатель. Если происходит ошибка, возвращается нулевой указатель.

Используйте функции *ferror* или *feof* для проверки, произошла ошибка или был достигнут конец файла.

б) Прототип функцииfread:

size_tfread(void *ptrvoid, size_t size, size_t count, FILE *filestream);

Описание

Функция *fread* считывает массив размером — *count* элементов, каждый из которых имеет размер *size* байт, из потока, и сохраняет его в блоке памяти, на который указывает *ptrvoid.*

Индикатор положения потока увеличивается на общее число записанных байтов. Общее количество успешно считанных байт *(count*).*

Таблица 4. Параметры

ptrvoid	Указатель на блок памяти, размер которого должен быть минимум (size*count) байт.
size	Размер в байтах каждого считываемого элемента.
count	Количество элементов, каждый из которых имеет размер size байт.
filestream	Указатель на объект типа FILE, который связан с потоком ввода.

Возвращаемое значение

Возвращается объект типа *size_t*, который содержит общее количество, успешно считанных, элементов. Если возвращаемое значение отличается от количества элементов, значит, произошла ошибка или был достигнут конец файла. Вы можете использовать функции *ferror()* или *feof()* для определения проблемы — произошла ошибка или был достигнут конец файла.

в) Прототип функциифwrite:

size_tfwrite(const void *ptrvoid, size_t size, size_t count, FILE *filestream);

Описание

Функция *fwrite* записывает массив размером — *count* элементов, каждый из которых имеет размер *size* байт, в блок памяти, на который указывает *ptrvoid* — текущая позиция в потоке.

Индикатор положения потока увеличивается на общее число записанных байтов. Общее количество записанных байт *(count*).*

Таблица 5. Параметры

ptrvoid	Указатель на массив элементов, которые необходимо записать в файл.
size	Размер в байтах каждого элемента массива.
count	Количество элементов, каждый из которых занимает size байт.
filestream	Указатель на объект типа FILE, который связан с потоком вывода.

Возвращаемое значение

Общее число элементов, которые успешно были записаны. Возвращаемое значение, в этом случае имеет тип данных *size_t.*

Если возвращаемое значение отличается от количества элементов, значит произошла ошибка.

8. Справочник используемых функций в языке Pascal

a) Функция определения достижения конца файла

Eof (<имя_ф_переменной>);

Название этой функции является сложносокращенным словом от **endoffile.** Значение этой функции имеет значение ***true,*** если конец файла уже достигнут, т.е. указатель стоит на позиции, следующей за последней компонентой файла. В противном случае значение функции – ***false.***

b) Изменение имени файла

Rename(<имя_ф_переменной>, <новое_имя_файла>);

Здесь новое_ имя_ файла – строковое выражение, содержащее новое имя файла, возможно с указанием пути доступа к нему.

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был открыт.

c) Уничтожение файла

Erase(<имя_ф_переменной>);

Перед выполнением этой процедуры необходимо закрыть файл, если он ранее был открыт.

d) Уничтожение части файла от текущей позиции указателя до конца

Truncate(<имя_ф_переменной>);

e) Открытие для добавления записей в конец файла

Append (<имя_ф_переменной>);

f) Функция, определяющая число компонентов в типизированном файле

fileSize(<имя_ф_переменной>);

g) Функция, значением которой является текущая позиция указателя в типизированном файле

filePos(<имя_ф_переменной>);

h) Процедура, смещающая указатель на компоненту типизированного файла с номером n

seek(<имя_ф_переменной>,n);

Так, процедура *seek(<имя_ф_переменной>,0)* установит указатель в начало файла, а процедура *seek(<имя_ф_переменной>)*, *FileSize(<имя_ф_переменной>)* установит указатель на признак конца файла.

i) Определение достижения конца строки в текстовом файле *EOLN(<имя_ф_переменной>)*

К текстовым файлам применимы процедуры *assign, reset, rewrite, read, write* и функция *eof*. Процедуры и функции *seek, filepos, filesize* к ним не применяются. При создании текстового файла в конце каждой записи (строки) ставится специальный признак *EOLN (end of line* – конец строки). Для определения достижения конца строки существует одноименная логическая функция *EOLN(<имя_ф_переменной>)*, которая принимает значение *true*, если конец строки достигнут.

Форма обращения к процедурам *write()* и *read()* для текстовых и типизированных файлов одинакова, но их использование принципиально различается.

Кроме процедур *read()* и *write()* при работе с текстовыми файлами используются их разновидности *readln()* и *writeln()*. Отличие заключается в том, что процедура *writeln()* после записи заданного списка записывает в файл специальный маркер конца строки. Этот признак воспринимается как переход к новой строке. Процедура *readln()* после считывания заданного списка ищет в файле следующий признак конца строки и подготавливается к чтению с начала следующей строки.

9. Список литературы для самостоятельного изучения

1. Бусько, В. Л. Основы алгоритмизации и программирования в среде Visual C++: лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. 1 – 2-го курсов всех спец. БГУИР / В. Л. Бусько, А. А. Навроцкий. – Минск: БГУИР, 2008. – 66 с.

2. Звягина А.С. Основы программирования на языке Pascal: Лабораторный практикум. Ч.9-10 / А.С. Звягина. – Хабаровск, изд-во ДВГГУ, 2007. – 23с.

3. Основы программирования на языках Си и C++ для начинающих // Электронный учебник. Режим доступа: http://cppstudio.com/

4. Федоров Д. Ю. Основы программирования на примере языка Python: учеб.пособие / Д. Ю. Федоров. – СПб., 2016. – 176с.

С.В. Летучий, ПДО объединений
«Робототехника», «Программирование»
КГБОУ ДО ХКЦРТДиЮ, ЦТТ

ПРЕДСТАВЛЕНИЕ ДАННЫХ В КОМПЬЮТЕРЕ. СИСТЕМА СЧИСЛЕНИЯ

1. Общие сведения

Любой компьютер предназначен для обработки ,преобразования и хранения данных. Для выполнения этих функций компьютер должен обладать некоторым способом представления этих данных. Представление данных заключается в преобразовании их в вид, удобный для последующей обработки либо пользователем, либо компьютером. Форма представления данных определяется их конечным предназначением. В зависимости от этого данные имеют внутреннее и внешнее представление.

Во внешнем представлении (для пользователей) все данные хранятся в виде файлов. Простейшими способами внешнего представления данных являются:

- вещественные и целые числа(числовые данные);
- последовательность символов (текст);
- изображение (графика, рисунки, схемы, фотографии).

Внутреннее преставление данных определяется физическими принципами, по которым происходит обмен сигналами между аппаратными средствами компьютера, принципами организации памяти, логикой работы компьютера. Любые данные для обработки компьютером представляются последовательностями двух целых чисел–единицы и нуля. Такая форма представления получила названия ***двоичной.*** Важным понятием при представлении данных в компьютере является система счисления.

2. Система счисления

Система счисления– это совокупность приемов и правил представления чисел с помощью символов, имеющих определенное количественное значение.

Различают позиционные системы счисления и непозиционные.

Непозиционная системы счисления– система, в которой символы, обозначающие то или иное количество, не меняют своего значения в зависимости от местоположения (позиции) в изображении числа.

Запись числа *A* в непозиционной системе счисления может быть представлена выражением:

$$A = D_1 + D_2 + \ldots + D_n = \sum_{i=1}^{n} D_i ,$$

где D_1, D_2, …,D_n– символы системы

Непозиционной системой счисления является самая простая система с одним символом (палочкой). Для изображения какого-либо числа в этой системе надо записать количество палочек, равное данному числу. Это система самая неэффективная, так как форма записи очень громоздка.

К непозиционной системе относится и римская, символы алфавита которой представлены ниже (таблица 1).

Таблица 1

Римскиецифры	I	V	X	L	C	D	M
Значение (обозначаемое количество)	1	5	10	50	100	500	1000

Так, например, в римской системе счисления в числе XXXII (тридцать два) значение цифры X в любой позиции равно десяти.

Запись чисел в данной системе счисления осуществляется по правилам:

1)если цифра слева меньше, чем цифра справа, толевая цифра вычитается из правой (IX:1<10,следовательно,10–1=9;XC:10<100,следовательно, 100– 10 = 90);

2)если цифра справа меньше или равна цифре слева, то эти цифры складываются (VII:5+1+1=7;XXXV: 10+10+10+5=35).

Так,число1984 в римской системе счисления имеет вид MCMLXXXIV (M – 1000, CM– 900, LXXX– 80, IV–4).

В римской системе нельзя записывать подряд 4 одинаковых цифр.

В общем случае непозиционные системы счисления характеризуются сложными способами записи чисел и правилами выполнения арифметических операций.

Позиционная система счисления –это система счисления, в которой значение цифры определяется ее местоположением (позицией) в изображении числа.

Алфавит позиционной системы счисления –упорядоченный набор символов (цифр){a_0,a_1,…,a_n},используемый для представления чисел в данной системе счисления.

Основание позиционной системы счисления –количество символов (цифр) алфавита q=n+1, используемых для изображения чисел в данной системе счисления.

Примером позиционной системы счисления является десятичная система счисления. Ее алфавит{0, 1,2, 3, 4,5, 6,7, 8, 9}. Основание q = 10.

Например, в десятичной системе счисления число 333 записывается с помощью одной цифры 3, назначении каждой цифры определяется ее местоположением в числе: первая тройка –число сотен в числе, вторая тройка– число десятков, последняя –число единиц.

За основание системы счисления можно принять любое натуральное число – два, три, четыре ит. д.

Обычно в качестве алфавита берутся последовательные целые числа от 0 до (q–1) включительно. В тех случаях, когда общепринятых (арабских) цифр не хватает для обозначения всех символов алфавита системы счисления с основанием q>10,используются буквенные обозначения цифр. Для примера в табл. 2 приведены алфавиты некоторых систем счисления.

Таблица 2

Система счисления	Основание	Алфавит системы счисления
Двоичная	2	0, 1
Троичная	3	0, 1, 3
Четверичная	4	0, 1, 2, 3
Пятеричная	5	0, 1, 2, 3, 4
Восьмеричная	8	0, 1, 2, 3, 4, 5, 6, 7
Десятичная	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Двенадцатеричная	12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
Шестнадцатеричная	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E,F

Для позиционной системы счисления справедливо равенство:

$$A_q = a_n q^n + a_{n-1} q^{n-1} + \dots + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + a_{-2} q^{-2} + \dots a_{-m} q^{-m} \qquad (1)$$

где A_q($A_q = a_n a_{n-1} \dots a_1 a_0, a_{-1} a_{-2} a_{-m}$)–любое число, записанное в системе счисления с основанием q;

a_i– цифры числа (i= n, n-1, …,1,0,-1, -2, -m); n +1 – число целых разрядов;

m– число дробных разрядов.

Равенство (1) называют *развернутой формой записи числа.*

Примеры. Записать числа$386,15_{10}101,11_2,561,42_8$, $6BF,A_{16}$ в развернутой форме. Согласно равенству (1) имеем:

$386,15_{10} = 3\cdot10^2 + 8\cdot10^1 + 6\cdot10^0 + 1\cdot10^{-1} + 5\cdot10^{-2}$

$101,11_2 = 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 1\cdot2^{-2}$

$561,423_8 = 5\cdot8^2 + 6\cdot8^1 + 1\cdot8^0 + 4\cdot8^{-1} + 2\cdot8^{-2} + 3\cdot8^{-3}$

$6BF,A_{16} = 6\cdot16^2 + B\cdot16^1 + F\cdot16^0 + A\cdot16^{-1}$

В вычислительной технике наибольшее распространение получили двоичная, восьмеричная, шестнадцатеричная системы счисления.

3. Перевод чисел в позиционных системах счисления

Приведем таблицу для перевода первых 16 чисел в различные системы счисления (табл. 3).

Таблица3

Десятичные числа q=10	Двоичные числа q=2	Восьмеричные числа q=8	Шестнадцатеричные числа q=16
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

3.1. Перевод чисел в десятичную систему счисления из системы счисления с основанием q

Перевод вещественного числа из десятичной системы счисления в систему счисления с основание mq осуществляется в два этапа. Переводится раздельно целая и дробная часть числа, а за тем при записи числа в новой системе счисления целая часть запятой (точкой) отделяется от дробной.

Перевод целых чисел из десятичной системы счисления в систему счисления с основанием q

Правило: Для перевода целого числа A из десятичной системы счисления в систему счисления с основанием q необходимо A разделить с остатком (нацело) на число q, записанное в десятичной системе. Затем неполное частное, полученное от такого деления, нужно снова разделить с остатком на q ит.д., пока последнее полученное неполное частное не станет равным нулю или меньше основания системы, в которую производится перевод. Представлением числа A в новой системе счисления будет последовательность остатков деления, изображенных q-ичной цифрой и записанных в порядке, обратном порядку их получения.

Примеры:

1. Перевести число405_{10} в двоичную систему счисления.

Число	Частное	Остаток
405:2=	202	0
202:2=	101	1
101:2=	50	0
50:2=	25	0
25:2=	12	1
12:2=	6	0
6:2=	3	0
3:2=	1	1
1:2=	0	1

Ответ: $405_{10}=110010101_2$.

2. Перевести число20959_{10}в шестнадцатеричную системусчисления.

Число	Частное	Остаток
20959:16=	1309	15
1309:16=	81	13
81:16=	5	1
5:16=	0	5

Ответ: $20959_{10}=51DF_{16}$.

Возможен еще более простой способ перевода из десятичной системы счисления в двоичную называемый вычитанием убывающих степеней - (в целом это применимо и к остальным системам). Приведем пример перевода числа 200 десятичной системы счисления в двоичную:

Первым действием необходимо найти такую степень числа два, которое будет ближайшим наименьшим к переводимому числу, в нашем случае это будет число 128, затем вписываем эти числа в строку 2 в порядке убывания:

1	200	72	-	-	8	-	-	-
2	128	64	32	16	8	4	2	1
3	72	8			0			
4	1	1	0	0	1	0	0	0

Там где вычитание возможно, в строке 3 записываем разницу и в строке 4 записываем единицу, если вычитание произведено и если вычитание не возможно, то 0.

Таким образом, мы получаем число:

11001000 – и сразу возможна проверка, сложим все весовые коды в разрядах, где стоят единицы, получим 128 + 64 + 8 = 200. Что собственно вытекает и из самой таблицы перевода.

3.2. Перевод чисел из двоичной системы счисления в системы с основанием $q=2^n$

Перевод чисел из двоичной системы в системы с основанием, равным степени двойки, выполняется по более простым правилам, чем с другим основанием.

Правило: Для перевода двоичного числа в систему с основание $mq=2^n$ нужно число разбить влево и вправо от запятой на группы по n цифр в каждой. Если в первой левой или последней правой группах окажется менее n цифр, то их необходимо дополнить слева и справа нулями. Затем для каждой группы, состоящей из n двоичных цифр, записать соответствующее число в системе счисления $q=2^n$.

Примеры:

1.Число10111111100000011_2 перевести в восьмеричную систему счисления.

$q = 8=2^3$ $n = 3$.

Заданное число разобьем справа на лево (с конца числа) на группы по 3 цифры (триады) и запишем соответствующие им числа в восьмеричной системе:

10111111100000011_2= 010 111111 100000 011 = 277403_8

↓ ↓ ↓ ↓ ↓ ↓

27 7 4 0 3

2.Число$110110111100011,11011_2$ перевести в шестнадцатеричную систему счисления.

$q = 16 = 2^4$, $n =4$.

Целую часть числа разобьем, справа на лево, а дробную–слева на право группы по 4цифры (тетрады, еще их называю nibble или полубайт),недостающие группы дополним нулями и запишем соответствующие им числа в шестнадцатеричной системе:

110110111100011= 00110110 1110 0011 =36E3

↓ ↓ ↓ ↓

3 6 E 3

3.3.Перевод чисел из систем счисления с основание $mq=2^n$ в двоичную систему

Правило: Для перевода числа из системы счисления с основанием $q=2^n$ в двоичную систему нужно каждую цифру числа заменить эквивалентным двоичным числом длиной n разрядов.

Примеры:

1. Число $537,45_8$ перевести в двоичную систему счисления.

$q = 8 = 2^3$ $n = 3$.

Заменим каждую цифру числа 537двоичным числом длиной три разряда (n= 3)

$536,45_8 = 101011110,100101_2$

(5→101, 3→011, 6→110, 4→100, 5→101)

2. Число $5F7,A23_{16}$ перевести в двоичную систему счисления.

$q = 16 = 2^4$ $n = 4$.

Заменим каждую цифру числа $5F7,A23_{16}$ двоичным числом длиной четыре разряда (n =4)

$5F7,A23_{16} = 0101\ 1111\ 0111,101000100011_2$

(5→0101, F→1111, 7→0111, A→1010, 2→0010, 3→0011)

4. Арифметические операции в позиционных системах счисления

Правила выполнения арифметических действий для всех позиционных систем счисления одинаковы и совпадают с правилами для десятичной системы счисления. При этом можно пользоваться таблицами сложения и умножения для системы счисления с основанием q.

Для q =2, 8 и16 таблицы сложения и умножения представлены ниже.

q = 2

a+b

a \ b	0	1
0	0	1
1	1	10

a · b

a \ b	0	1
0	0	0
1	0	1

q = 8

a+b

a \ b	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

a · b

a \ b	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

q = 16

a+b

a \ b	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

a · b

a \ b	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

4.1. Сложение

Если результат сложения двух цифр в системе счисления с основанием q, больше q (т.е.полученное число двузначное), то старшая цифра результата равна1.Таким образом, при сложении в следующий разряд может переходить только единица, а результат сложения в любом разряде будет меньше, чем q. Результат сложения двух положительных чисел имеет столько же значащих цифр, что и максимальное из двух слагаемых, либо на одну цифру больше, но этой цифрой может быть только единица.

Примеры. Сложить числа:

1. $100110011_2 + 1101001_2 = 110011100_2$
2. $723{,}3_8 + 467{,}53_8 = 1413{,}03_8$
3. $3B9{,}6_{16} + 78C{,}8_{16} = B45{,}E_{16}$

```
  1 0 0 1 1 0 0 1 1        7 2 3, 3        3 B 9, 6
+     1 1 0 1 0 0 1    +   4 6 7, 5 3    +   7 8 C, 8
-------------------      ---------------  ------------
  1 1 0 0 1 1 1 0 0        1 4 1 3, 0 3      B 4 5, E
```

4.2. Вычитание

Если необходимо вычесть из цифры a цифру b и a ≥ b, то в столбце b таблицы сложения ищем значение числа a. Самая левая цифра в строке, в которой найдено значение числа a, и будет результатом вычитания. Если же a < b, то нужно заимствовать единицу из левого разряда, поэтому в столбце ищем число 1a, и левая цифра в соответствующей строке будет результатом вычитания.

Примеры. Выполнить вычитание чисел:

1. $1100000{,}001_2 - 101101{,}1_2 = 110010{,}101_2$
2. $1510{,}2_8 - 1430{,}73_8 = 57{,}25_8$
3. $25E{,}D8_{16} - 171{,}6_{16} = ED{,}78_{16}$

```
  1 1 0 0 0 0 0, 0 0 1      _1 5 1 0, 2      _2 5 E, D 8
    1 0 1 1 0 1, 1          1 4 3 0, 7 3     1 7 1, 6
-------------------       ---------------  ------------
    1 1 0 0 1 0, 1 0 1          5 7, 2 5         E D, 7 8
```

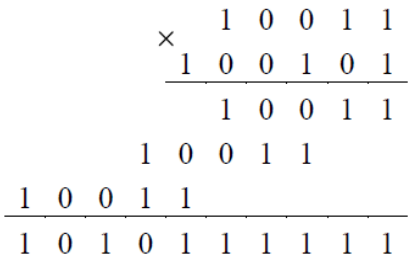
4.3. Умножение

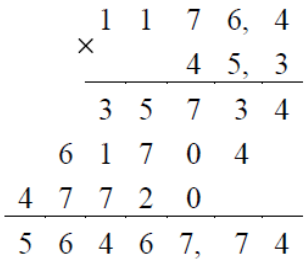
Умножение выполняется столбиком с использованием соответствующих таблиц умножения и сложения.

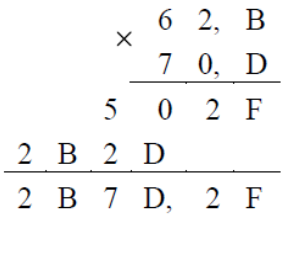
Отметим, что во всех позиционных системах счисления с любым основанием q умножение на числа вида q^m, где m – целое число, сводится просто к перенесению запятой умножаемого на m разрядов вправо или влево (в зависимости от знака m), так же, как и в десятичной системе счисления.

Примеры. Выполнить умножение чисел:

1. $10011_2 \cdot 100101_2 = 1010111111_2$
2. $1176{,}4_8 \cdot 45{,}3_8 = 56467{,}74_8$
3. $62{,}B_{16} \cdot 70{,}D_{16} = 2B7D{,}2F_{16}$





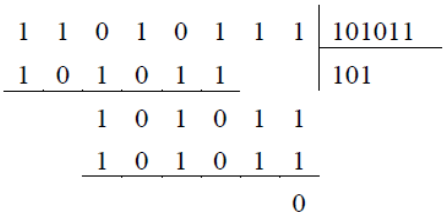


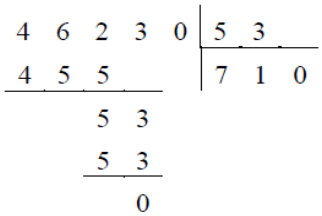
4.4. Деление

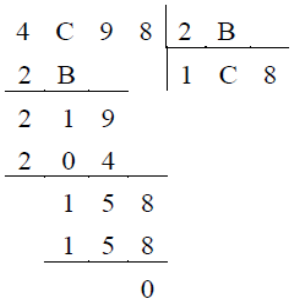
Как для умножения, так и для деления нужны обе таблицы – умножения и сложения в соответствующей системе счисления. Само деление выполняется уголком с последующим вычитание сомножителей.

Примеры. Выполнить деление:

1. $11010111_2 : 101011_2 = 101_2$
2. $46230_8 : 53_8 = 710_8$;
3. $4C98_{16} : 2B_{16} = 1C8_{16}$.







5. Задания для самостоятельного решения

1) Перевести числа из заданной системы счисления в десятичную: 11011_2;$0{,}1101_2$; $F0A9_{16}$; $46{,}05_7$; $471{,}31_8$.

2) Перевести числа 95 и 568,125 из десятичной системы счисления в двоичную, восьмеричную, шестнадцатеричную.

3) Число 10010111_2 перевести в четверичную систему счисления.

4) Упорядочить по убыванию числа: 55_7,55_{16}, 55_8.

5) Найти сумму и разность чисел$11001,11_2$ и $1010,011_2$в двоичной системе счисления.

6) Найти сумму и разность чисел $505C_{16}$и $5A6_{16}$ в шестнадцатеричной системе счисления.

7) Найти произведение чисел11_2 и 101_2 в двоичной системе счисления.

8) Найти значение выражения 2^4+ 2 +1 в двоичной системе счисления.

9) В восьмеричной системе счисления число представлено в виде10000_8. Выбрать правильный вариант представления в десятичной системе счисления.8^4, 8^5, 8·10000,8·5.

10) Найти значение числа1110010111_2 в шестнадцатеричной и восьмеричной системах счисления.

11) В какой системе счисления выполнены действия: 122 +2 = 201?

12) В какой системе счисления выполнены действия:127+2 = 131?

13) Число$10010111,100011_2$ перевести в восьмеричную и шестнадцатеричную системы счисления.

14) Число $2A_{16}$ перевести в восьмеричную систему счисления.

15) Число 23_xиз системы счисления с основанием x перевели в десятичную систему счисления иполучили19_{10}. Найти основание системы счислениях.

16) Число135_xиз системы счисления с основанием x перевели в десятичную систему счисления и получили59_{10}. Найти основание системы счисления x.

17) Обратный кодчисла X имеет значение 11100111_2. Найти его значение в десятичной системе счисления.

18) Найти дополнительный код в однобайтном формате числа 11_{10}.

19) Найти дополнительный код для числа X= -24_{10} в однобайтном формате.

20) Дополнительный код числа X имеет значение 10101101. Найти его значение в десятичной системе счисления.

21) Даны три числа33,66,88 в различных системах счисления. К этим числам прибавили по единице и получили во всех системах счисления 100. Найти значения всех этих чисел в десятичной системе счисления.

22) Задано число в шестнадцатеричной системе счисленияF023A9,12C4. Как изменится число, если в его представлении запятую перенести на два знака влево? На три знака вправо?

6. Индивидуальные задания

Задание 1. Переведите данные числа из двоичной системы счисления в десятичную.

Вариант	1	2	3	4	5
Числа	10101010101	101010001	10010010100	1101011000	110000100
	10011000	10101010011	1000000100	100010010010	100110000111
	10000010110	11010001000	1110000	10101000110	100100011000

Вариант	6	7	8	9	10
Числа	100101100010	110010010	10110010000	100100010001	10000101001
	1001000110	1100011000	11101100101	1000111001	10000000111
	11100110110	11000010000	11100010111	1101100011	1001110001

Вариант	11	12	13	14	15
Числа	1100100110	110000000	1000000010101	11110000100	10100110011
	1000010110	100101010110	100110011001	1100010001	100100100101
	10100010010	11101100001	1101100001	100101010001	100010010001

Вариант	16	17	18	19	20
Числа	1100110011	100110101	100000100	100110010110	100001111001
	1101100010	1010010011	10110011001	100100110010	100000010000
	10001000100	1000000100100	100000110111	110010000	1101000100

Задание 2. Переведите данные числа из десятичной системы счисления в двоичную.

Вариант	1	2	3	4	5	6	7	8	9	10
Числа	585	285	905	483	88	325	464	342	749	817
	673	846	504	412	153	112	652	758	691	661
	626	163	515	738	718	713	93	430	1039	491

Вариант	11	12	13	14	15	16	17	18	19	20
Числа	596	322	780	164	280	728	158	328	1026	853
	300	320	949	1020	700	383	177	537	725	135
	515	738	718	713	464	202	439	634	100	66

Задание 3. Переведите данные числа из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную системы счисления. Вещественные числа перевести в новую систему счисления с точностью до четвертого знака.

Вариант	1	2	3	4	5	6	7	8	9	10
Числа	860	250	759	216	530	945	287	485	639	618
	78,15	57,17	82,21	33,38	25,27	85,14	20,18	90,42	48,28	55,49

Вариант	11	12	13	14	15	16	17	18	19	20
Числа	772	233	218	898	557	737	575	563	453	572
	76,45	43,86	77,35	71,41	30,19	92,24	74,23	30,18	41,29	36,73

Задание4. Переведите числа из заданной системы счисления в десятичную.

Вариант	7	8	9	10	11	12
Числа	1416_8	2037_8	1720_8	1361_8	1315_8	1072_8
	$16D,8_{16}$	$196,B_{16}$	$14F,1_{16}$	$19A,6_{16}$	$14C,7_{16}$	$2A3,B_{16}$
	$11000,11_2$	$11010,011_2$	$10011,11_2$	$11101,011_2$	$11101,01_2$	$10110,11_2$

Вариант	1	2	3	4	5	6
Числа	1740_8	2671_8	1216_8	2534_8	1627_8	5142_8
	$A2C,8_{16}$	$48E,4_{16}$	$14F,A_{16}$	$3FD,9_{16}$	$553,E_{16}$	$19F,C_{16}$
	$10110,11_2$	$1011,101_2$	$11100,011_2$	$10101,01_2$	$11110,111_2$	$10001,01_2$

Вариант	13	14	15	16	17	18
Числа	1501_8	1510_8	3207_8	4510_8	1023_8	2506_8
	$3AB,A_{16}$	$1BA,5_{16}$	$186,C_{16}$	$24D,1_{16}$	$49A,C_{16}$	$15C,4_{16}$
	$10101,101_2$	$11010,11_2$	$11110,01_2$	$10010,11_2$	11001.011_2	$10101,111_2$

Вариант	19	20
Числа	1053_8	3260_8
	$2E3,D_{16}$	$32B,F_{16}$
	$10011,11_2$	$10001,111_2$

А.В. Крупский

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

1. Введение

Программы составляет программист. Рассмотрим, что же необходимо программисту для создания программы (рис.1).

Прежде чем создать программу, программист придумывает **алгоритм**.

Затем выбирает **исполнителя** своего алгоритма.

И в соответствии с **языком описания алгоритма**, понятному исполнителю, составляет **программу**.

Таким образом, мы наметили путь от алгоритма к программе.

Рис.1

Понятие алгоритма

Прежде чем что-нибудь сделать, надо составить **план**.

И в жизни мы все время составляем планы наших действий. Например, утром большинство из нас действует по такому плану: встать → одеться → умыться → позавтракать → выйти из дома в школу или на работу.

Такой план действий называют **алгоритмом**.

Алгоритм – это последовательность или план действий, которые нужно выполнить для решения определенной задачи.

Алгоритмизация – это техника разработки (составления) алгоритма для решения задач на ЭВМ.

Алгоритм состоит из отдельных шагов – команд. Для каждого шага алгоритма можно и нужно составить более **подробный план**.

Остановиться надо на таком плане, в котором исполнителю будет понятно, как выполнить каждый шаг.

Пример алгоритма кипячения чайника:

1. Проверить, есть ли в чайнике вода, если есть, то перейти к действию 2. Если нет, то налить воду в чайник и перейти к действию 2.
2. Зажечь плиту и перейти к действию 3.
3. Поставить чайник на плиту и перейти к действию 4.
4. Если чайник со свистком, ждать, пока он не засвистит, и перейти к действию 5. Если чайник без свистка, ждать до тех пор, пока из носика не пойдет пар, и перейти к действию 5.
5. Выключить плиту.

Понятие исполнителя алгоритма

Исполнитель – это тот, кто умеет понимать и выполнять некоторые команды.

Пример исполнителя алгоритма «Папа» (рис. 2).

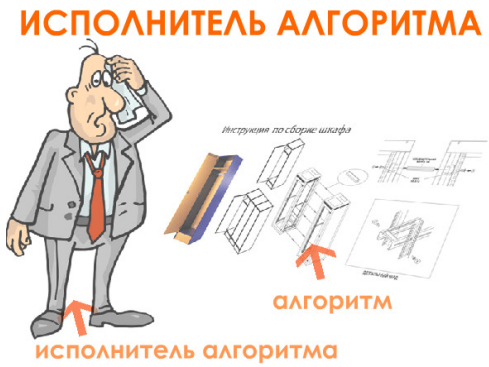


Рис. 2 ***Рис.3***

Классификация исполнителей: неформальный и формальный (рис. 3). В отличие от устройств (роботов и компьютеров), человек является **неформальным** исполнителем алгоритма. Что это означает? Во-первых, человек не выполняет алгоритм бездумно и формально. Соображения здравого смысла часто берут верх, какими бы строгими ни были инструкции. Во-вторых, человек, даже если он еще мал, обладает определенным жизненным опытом. И чем больше этот опыт, тем менее подробным может быть алгоритм. В-третьих, неформальному исполнителю известна **конечная цель,** и он к ней стремится. **Формальный** исполнитель не обладает ни жизненным опытом, ни здравым смыслом, не стремится к конечному результату и **все инструкции выполняет буквально**.

Система команд исполнителя

Команда – это указание исполнителю выполнить конкретное действие.

Для решения большинства задач недостаточно отдать одну команду исполнителю, надо составить для него алгоритм – план действий, состоящий из команд, которые ему понятны (входят в его **систему команд исполнителя**).

Система команд исполнителя (СКИ) – набор команд, понятных исполнителю. Исполнитель может выполнить только те команды, которые входят в его СКИ.

Исполнителями могут быть люди: ученик, рабочий, учитель; животные: дрессированные собака, кошка; машины: станки, роботы, компьютеры.

Язык описания алгоритма и программа

Исполнитель сможет выполнить алгоритм, если он ему **известен**, т.е. если алгоритм ему сообщили.

Для людей важнейшим способом общения является **язык**.

Для формального исполнителя язык общения не может быть многозначным. Для таких исполнителей разрабатывают и используют специальные **искусственные языки**, не допускающие различные толкования отдельных слов и выражений.

Исполнителю нужно сообщить четкий **алгоритм**, в соответствии с которым он будет действовать.

Программа – это алгоритм, записанный на языке исполнителя.

Программирование – это процесс перевода алгоритма на язык определенного исполнителя.

Так как **разные исполнители** владеют разными языками, а один и тот же алгоритм могут выполнять разные исполнители, то на основе одного алгоритма могут быть написаны **разные программы**.

В настоящее время существует **множество языков программирования: Си, Паскаль, Делфи, Си++, Вижуал Бейсик, Джава** и другие.

Чтобы **научиться** писать программы на том или ином языке, нужно изучить **алфавит, словарь и грамматические правила,** по которым строятся предложения в этом языке. При этом **не допускаются никакие отклонения** от правил написания слов и предложений. Иначе исполнитель просто **откажется** выполнять программу и выдаст ошибку.

Практическое задание

1) Придумайте и составьте словесный алгоритм из жизни (например, алгоритм уборки комнаты; алгоритм проведения выходного дня; алгоритм собирания ягод).

2) Ответьте на *контрольные вопросы:*

1. Кто составляет программы?
2. Опишите путь от алгоритма к программе.
3. Что такое «алгоритм»?
4. Что такое «исполнитель алгоритма»?
5. Приведите примеры формальных исполнителей алгоритма.
6. Приведите примеры неформальных исполнителей алгоритма.
7. Что такое «команда»?
8. Что такое «СКИ»?
9. Что такое «язык описания алгоритма»?
10. Приведите примеры языков программирования.
11. Что такое «программа»?
12. Что такое «программирование»?

2. Алгоритмы

Виды алгоритмов

Следуя определенным правилам, алгоритмы делят на следующие виды: линейные; условные; циклические; смешанные.

Линейный алгоритм

В линейном алгоритме команды выполняются последовательно, одна за другой.

Примером линейного алгоритма может служить алгоритм заваривания чая:

вскипятить воду
сполоснуть заварочный чайник горячей водой
насыпать заварку
залить заварку кипятком
закрыть чайник чем-нибудь теплым
подождать 5 минут
теперь можно пить чай

Условный алгоритм

Условные алгоритмы **всегда** содержат какое-либо **условие**, от выполнения которого зависят дальнейшие действия алгоритма. Условный алгоритм легко **определить** по ключевым словам: **если, то, иначе**.

Примером условного алгоритма может служить алгоритм перехода улицы:

подойти к пешеходному переходу
если есть светофор, то
ждать зеленого света
перейти улицу
иначе
ждать, пока слева не будет машин
перейти улицу до середины
ждать, пока справа не будет машин
перейти вторую половину улицы

Циклический алгоритм

Циклические алгоритмы основаны на каких-либо **повторяющихся действиях**. В циклическом алгоритме некоторые действия повторяются несколько раз. Существуют два вида циклических алгоритмов. В одном из них мы знаем заранее, сколько раз надо сделать эти действия, в другом мы должны остановиться лишь тогда, когда выполнится некоторое условие.

Примером цикла первого типа является наша жизнь в рабочие дни (с понедельника по субботу): мы выполняем 6 раз почти одни и те же действия.

```
/* Число шагов известно */
повторить 6 раз
        проснуться
        встать
        позавтракать
        пойти в школу
        вернуться домой
        пообедать
        сделать уроки
        поиграть в футбол
        лечь спать
/* программа на воскресенье */
  спать
  ...
```

Пример цикла второго типа – алгоритм распилки бревна: мы не можем заранее сказать, сколько раз нам надо провести пилой от себя и на себя, так как это зависит от плотности дерева, качества пилы и наших усилий. Однако

```
/* Число шагов неизвестно,
   но ограничено условием
*/
положить бревно на козлы
наметить место распила
пока полено не отвалится
        пилить от себя
        пилить на себя
положить полено в поленницу
```

мы точно знаем, что надо закончить работу, когда очередное отпиленное полено упадет на землю.

Свойства алгоритмов

К алгоритмам предъявляют специальные **требования**, которые необходимо соблюдать при их составлении.

Эти требования называют **свойствами** алгоритмов: понятность, дискретность, определенность, результативность, массовость.

Рассмотрим по порядку все свойства алгоритмов

Понятность алгоритма. Это свойство означает, что исполнитель алгоритма должен **понимать**, как его выполнять. То есть, имея алгоритм и какие-то исходные данные, исполнитель **должен знать**, как надо действовать для выполнения этого алгоритма. Машина – **формальный** исполнитель алгоритма. И она выполняет лишь те команды, которые ты даешь ей, нажимая на кнопки.

Дискретность алгоритма. Дискретность, или пошаговость, алгоритма означает, что алгоритм должен состоять из последовательного выполнения простых **отдельных шагов**. Свойство дискретности будет нарушено в том случае, если алгоритм будет представлен не в виде четких отдельных шагов, а в виде какого-либо повествовательного рассказа.

Определенность алгоритма. Это свойство означает, что каждый шаг алгоритма должен быть четким, однозначным и не оставлять исполнителю возможность самому принимать какие-то решения.

Результативность алгоритма. Результативность, или конечность, алгоритма состоит в том, что алгоритм должен приводить к нужному результату.

Массовость алгоритма. Свойство массовости означает, что алгоритм решения задачи разрабатывается в общем виде, то есть он должен быть применим к похожим задачам, различающимся лишь исходными данными.

Способы представления алгоритмов

Один и тот же алгоритм можно представить в одной из **трех форм**: словесная, графическая, программная

Словесная форма представления алгоритма отображает алгоритм в виде последовательности действий на обычном языке, например на русском.

Пример: Алгоритм нахождения среднего арифметического трех чисел: 1) сложить два числа; 2) к полученной сумме прибавить третье число; 3) разделить сумму чисел на 3.

Графический способ представления алгоритмов изображается в виде последовательности связанных между собой **блоков**, каждый из которых соответствует выполнению определенного действия. Такое графическое представление называют **схемой алгоритма**, или **блок-схемой**.

Пример блок-схемы алгоритма нахождения среднего арифметического трех чисел (рис. 4):

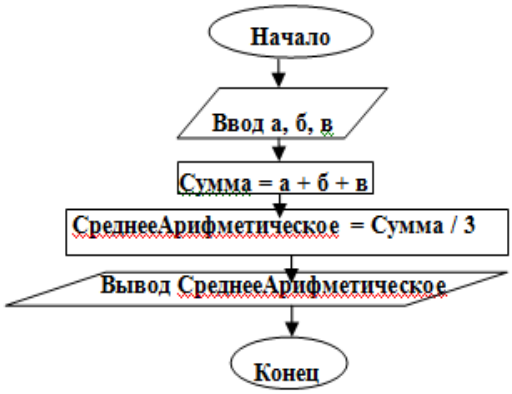


Рис. 4. Пример блок-схемы

Программный способ представления алгоритмов – это и есть самая настоящая **программа**, написанная на языке программирования.

Пример программы вычисления среднего арифметического трех чисел (рис. 5):

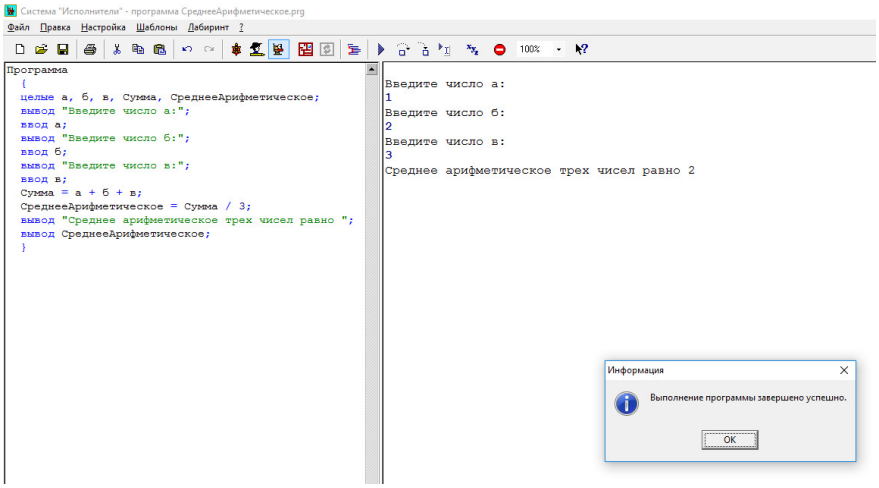


Рис.5. Фрагмент окна программы «Исполнители»

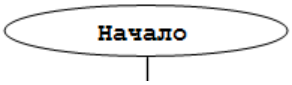
Блок-схемы алгоритмов

Блок-схема представляет собой алгоритм в виде последовательно соединенных блоков. Для того чтобы все программисты понимали, что за алгоритм представлен в виде блок-схемы, существует **специальные правила** записи блок-схем алгоритмов.

В блок-схеме каждому типу действия соответствует своя **геометрическая фигура**. Рассмотрим основные блоки блок-схем.

Блок начала алгоритма изображается в виде овала, внутри которого пишут слово **НАЧАЛО** (рис. 6).

Рис. 6



Блок начала алгоритма **всегда** имеет один выход и не имеет входов.

Блок ввода-вывода информации отображается в виде параллелограмма (рис. 7).

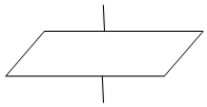


Рис. 7

Этот блок используют в том случае, когда необходимо ввести информацию в компьютер с клавиатуры (рис. 8) или вывести на экран монитора (рис. 9). Внутри блока записывают то, что необходимо ввести или вывести. Этот блок имеет один вход и один выход.

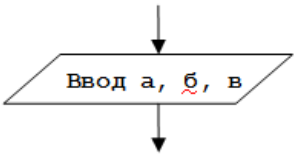


Рис. 8 Ввод информации с клавиатуры

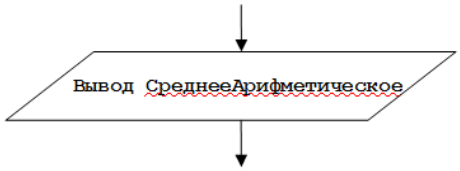


Рис. 9. Вывод информации на экран монитора

Блок действия отображается в виде прямоугольника (рис. 10). В этом блоке записываются действия, выполняющиеся в алгоритме. Обычно это выполнение каких-либо операций. Этот блок также имеет один вход и один выход.

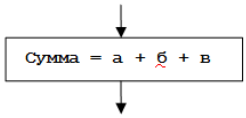


Рис. 10. Блок действия

Блок условия отображается в виде ромба (рис. 11). Внутри ромба записывают **условие**, которое нужно проверить. Этот блок имеет один вход и два выхода.

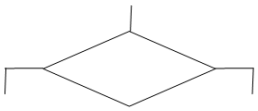
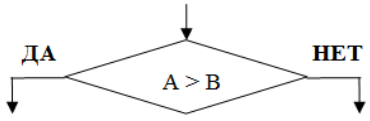
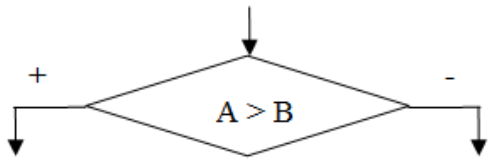


Рис.11. Блок условия





Над линиями выходов пишут слово «да», «нет» или ставят «+», «-» (рис. 12).

Рис.12

Таким образом, **если условие внутри ромба выполняется**, алгоритм следует дальше по ветке, соответствующей слову «да» или знаку «+».

А в том случае, **если условие не выполняется**, алгоритм продолжает выполняться по ветке «нет» или знаку «-».

Блок конца алгоритма изображается в виде овала, внутри которого пишут слово **КОНЕЦ** (рис. 13).

Рис.13. Блок конца

Это всегда завершает алгоритм и имеет один вход и не имеет выходов.

Примеры блок-схемы

Пример № 1. Построить блок-схему алгоритма вычисления выражения: 10+2*8-4=

Решение. Словесное описание алгоритма:

1. Вычислить произведение 2*8.
2. Вычислить сумму произведения 2*8 и 10 (16+10).
3. Вычислить разность полученной суммы и 4 (16-4).
4. Вывести ответ на экран.

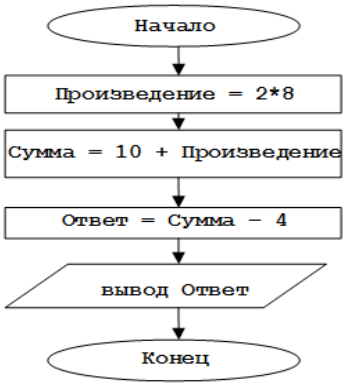
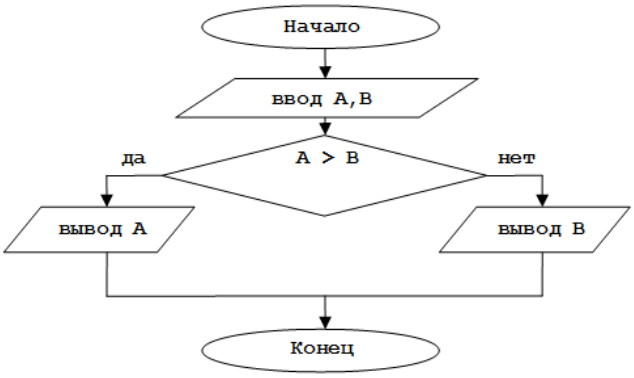


Рис. 14

Пример № 2. Ввести два натуральных числа A и B. Определить наибольшее число.



Решение.

Словесное описание алгоритма:

1. Ввести числа A и B.
2. Если A > B, то вывести на экран число A.
3. Иначе вывести на экран число B.

Рис. 15

Практическое задание

1) Составьте блок-схему задачи вычисления произведения трех чисел.
2) Ответьте на *контрольные вопросы:*
   1. Какие виды алгоритмов вы знаете?
   2. Линейный алгоритм. Приведите пример алгоритма.

3. Условный алгоритм. Приведите пример алгоритма.
4. Циклический алгоритм. Приведите пример алгоритма.
5. Способы представления алгоритма.
6. Блок-схема алгоритма.
7. Основные блоки блок-схем и их назначение.

3. Среда «Исполнители»

Знакомство со средой «Исполнители»

Среда имеет название **Исполнители**, так как в ней можно работать с тремя исполнителями: Роботом, Чертежником и Черепахой.

Каждый исполнитель имеет свою **систему команд** и выполняет программу, которая вводится в **текстовом редакторе**. При этом все действия исполнителя отображаются на экране.

Рассмотрим **интерфейс** системы «Исполнители» (рис. 16).

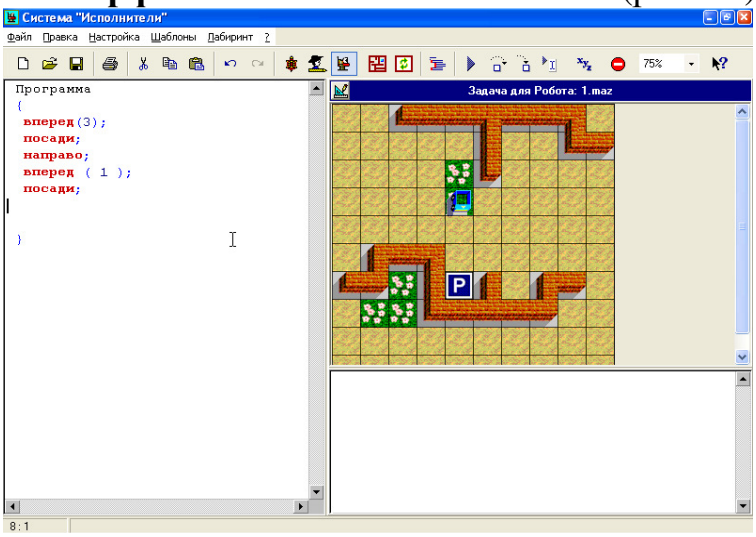


Рис. 16. Интерфейс системы «Исполнители»

Верхняя часть содержит **меню** и **панель инструментов** (рис. 17).

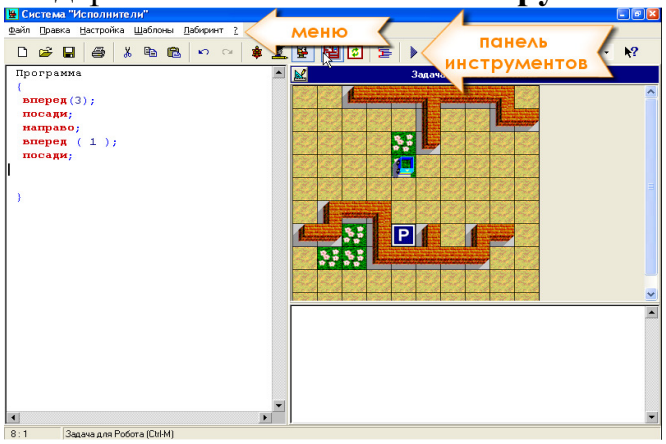


Рис. 17. Меню и панель инструментов

Основная часть окна разделена на три области:

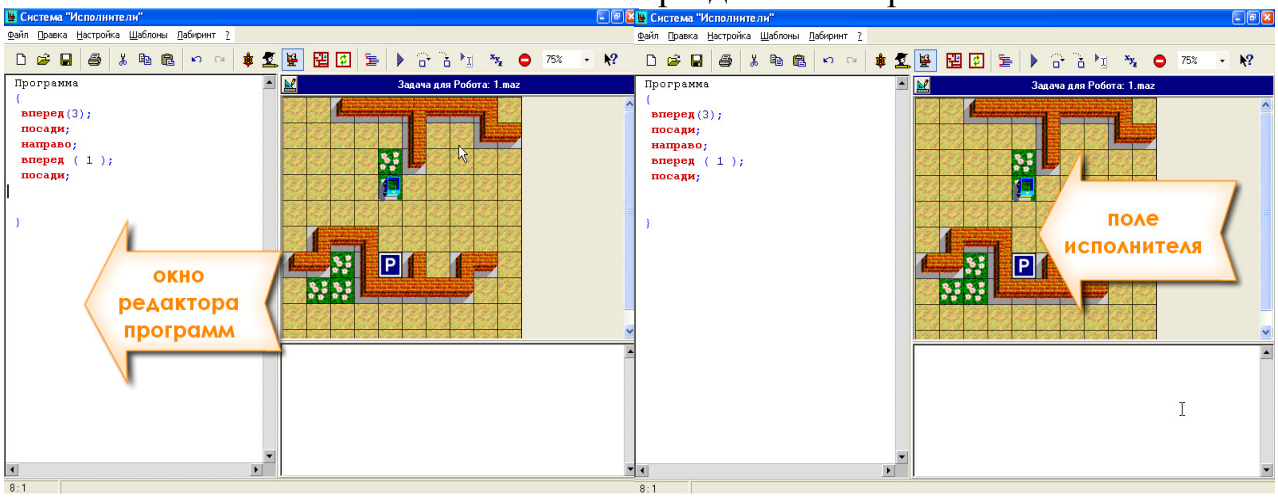


Рис. 18. Окно редактора программ ***Рис. 19. Поле исполнителя***

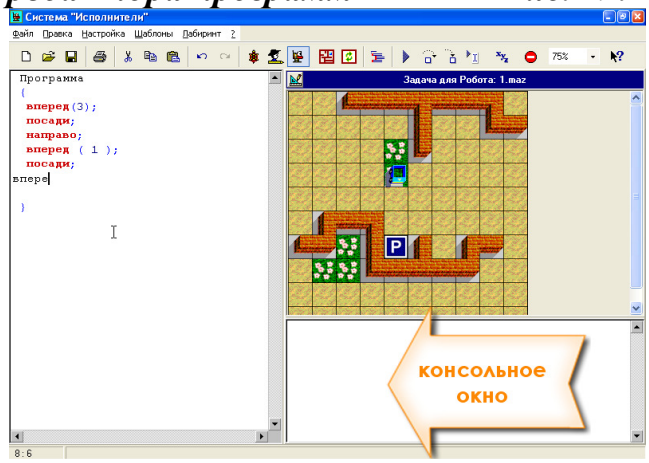


Рис. 20. Консольное окно

В **окне редактора** набирается текст программы для исполнителя на специальном языке программирования.

После этого надо запустить программу на выполнение. Для этого нужно нажать клавишу **F9** либо кнопку с треугольной стрелкой на панели инструментов (рис. 21).

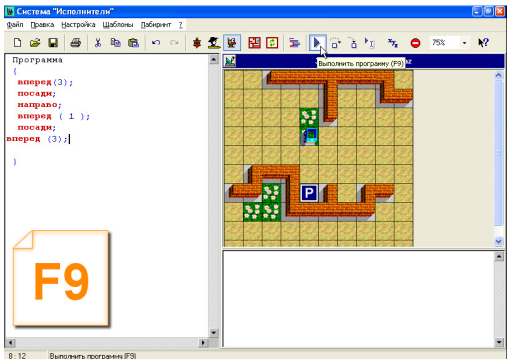


Рис. 21. Запуск программы на исполнение

Когда исполнитель выполняет программу, его действия отображаются на поле исполнителя.

Каждый исполнитель может делать что-то свое. **Черепаха и Чертежник** могут рисовать на белом листе, а **Робот** выполняет специальное задание по озеленению местности.

Помощь по работе системы «Исполнители» вызывается по кнопке **F1** (рис. 22).

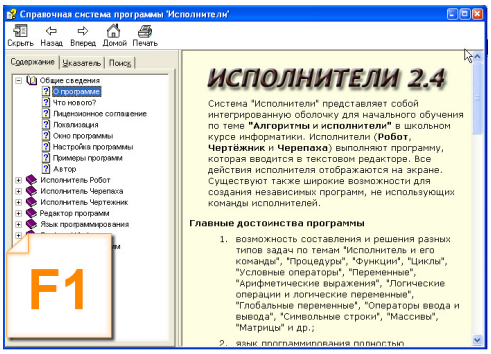


Рис. 22. Справочная система программы «Исполнители»

Если необходимо узнать о конкретном элементе главного окна, нужно щелкнуть по кнопке с вопросительным знаком или выбрать пункт меню «?» (рис. 23). При этом курсор изменит форму на вопрос, и, щелкнув на каком-то элементе, можно получить справку о нем.

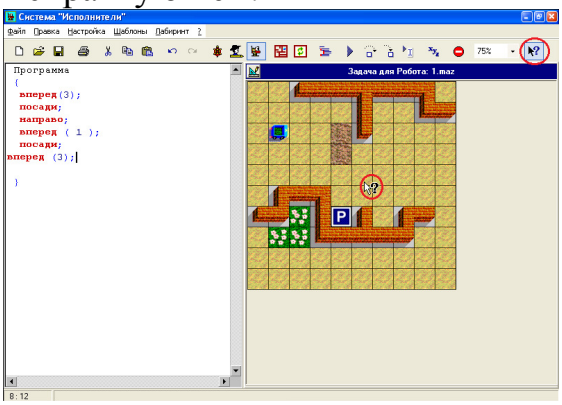


Рис.23. Справка по выбранному элементу

Окно текстовой консоли предназначено для ввода данных с клавиатуры и вывода текстовой информации.

Запустить систему «Исполнители» можно, выполнив **двойной щелчок мышью** на значке файла (рис. 24).

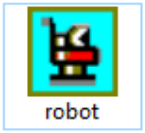


Рис.24. Значок файла системы «Исполнители»

Контрольные вопросы

1. Почему программа называется «система «Исполнители»?
2. Опишите интерфейс системы «Исполнители»?
3. Как запустить программу в системе «Исполнители»?
4. Как вызвать справку программы?
5. Как получить справку по отдельным элементам интерфейса программы?
6. Назначение окна текстовой консоли.

4. Примеры решения задач

Задача № 1. Получить реверсную запись трехзначного числа

Формулировка.

Сформировать число, представляющее собой реверсную (обратную в порядке следования разрядов) запись заданного трехзначного числа. Например, для числа 341 таким будет 143. Давайте разберемся с условием. В нашем случае с клавиатуры вводится некоторое трехзначное число (трехзначными называются числа, в записи которых три разряда (то есть три цифры), например: 115, 263, 749 и т.д.).

Нам необходимо получить в некоторой переменной число, которое будет представлять собой реверсную запись введенного числа. Другими словами, нам нужно перевернуть введенное число «задом наперед», представить результат в некоторой переменной и вывести его на экран.

Решение.

Определимся с выбором переменных и их количеством. Ясно, что одна переменная нужна для записи введенного числа с клавиатуры, мы обозначим ее как **n**.

Так как нам нужно переставить разряды числа **n** в некотором порядке, следует для каждого из них также предусмотреть отдельные переменные. Обозначим их как **a** (для разряда единиц), **b** (для разряда десятков) и **c** (для разряда сотен).

Теперь можно начать запись самого алгоритма. Будем разбирать его поэтапно:

1) Вводим число **n**.

2) Работаем с разрядами числа **n**. Как известно, последний разряд любого числа в десятичной системе счисления – это остаток от деления этого числа на 10. В терминах языка **системы «Исполнители»** это означает, что для получения разряда единиц нам необходимо присвоить переменной **a** остаток от деления числа **n** на 10. Этому шагу соответствует следующий оператор:

```
a = n % 10;
```

Получив разряд единиц, мы должны отбросить его, чтобы иметь возможность продолжить работу с разрядом десятков. Для этого разделим число **n** на 10. В терминах **системы «Исполнители»**, опять же, это означает: присвоить переменной **n** результат от деления без остатка числа **n** на 10. Это мы сделаем с помощью оператора

n = n / 10;

3) Очевидно, что после выполнения **п. 2** в переменной **n** будет храниться двухзначное число, состоящее из разряда сотен и разряда десятков исходного. Теперь, выполнив те же самые действия еще раз, мы получим разряд десятков исходного числа, но его уже нужно присваивать переменной **b**.

4) В результате в переменной **n** будет храниться однозначное число – разряд сотен исходного числа. Мы можем без дополнительных действий присвоить его переменной **c**.

5) Все полученные в переменных числа – однозначные. Теперь переменная **n** нам больше не нужна, и в ней нужно сформировать число-результат, в котором **a** будет находиться в разряде сотен, **b** – десятков, **c** – единиц. Легко понять, что для этого нам следует умножить **a** на 100, прибавить к полученному числу **b**, умноженное на 10 и **c** без изменения, и весь этот результат присвоить переменной **c**. Это можно записать так:

n = 100 * a + 10 * b + c;

6) Далее остается только вывести полученное число на экран.

Код:

```
Программа
  {
int n, a, b, c;
print "Введите трехзначное число:";
inputn;
a = n % 10;
  n = n / 10;
  b = n % 10;
  n = n / 10;
  c = n;
  n = 100 * a + 10 * b + c;
print "Реверсная запись введенного
числа: ";
print n;
  }
```

Задача № 2. Посчитать количество единичных битов числа

Формулировка.

Дано натуральное число меньше 16. Посчитать количество его единичных битов. Например, если дано число 9, запись которого в двоичной системе

счисления равна 1001_2 (подстрочная цифра 2 справа от числа означает, что оно записано в двоичной системе счисления), то количество его единичных битов равно 2.

Решение.

Нам необходима переменная для ввода с клавиатуры. Обозначим ее как **n**. Так как мы должны накапливать количество найденных битов, возникает потребность в еще одной переменной. Обозначим ее как **count** («*count*» в переводе с англ. означает «считать», «подсчет» и т.д.).

Переменные возьмем типа **int** (они могут принимать значения от 0 до 65535), и пусть в данном случае такой объем избыточен, но это не принципиально важно. Как же сосчитать количество битов во введенном числе? Ведь число же вводится в десятичной системе счисления, и его нужно переводить в двоичную?

На самом деле все гораздо проще. Здесь нам поможет одно интересное правило:

Остаток от деления любого десятичного числа x на число p дает нам разряд единиц числа x (его крайний разряд справа) в системе счисления с основанием p.

То есть, деля некоторое десятичное число, например, на 10, в остатке мы получаем разряд единиц этого числа в системе счисления с основанием 10. Возьмем, например, число 3468. Остаток от деления его на 10 равен 8, то есть разряду единиц этого числа.

Понятно, что такие же правила господствуют и в арифметике, в других системах счисления, в том числе в двоичной системе.

Предлагаю поэкспериментировать: запишите на бумаге десятичное число, затем, используя любой калькулятор с функцией перевода из одной системы счисления в другую, переведите это число в двоичную систему счисления и также запишите результат. Затем разделите исходное число на 2 и снова переведите в двоичную систему. Как оно изменилось в результате? Вполне очевидно, что у него пропал крайний разряд справа, или, как мы уже говорили ранее, разряд единиц. Но как это использовать для решения задачи? Воспользуемся тем, что в двоичной записи числа нет цифр, кроме 0 и 1.

Легко убедиться в том, что сложив все разряды двоичного числа, мы получаем как раз таки количество его единичных битов. Это значит, что вместо проверки значений разрядов двоичного представления числа мы можем прибавлять к счетчику сами эти разряды: если в разряде был 0, значение счетчика не изменится, а если 1, то увеличится на единицу.

Теперь, резюмируя вышеприведенный итог, можно поэтапно сформировать сам алгоритм:

1) Вводим число **n**.

2) Обнуляем счетчик разрядов **count**. Это делается потому, что значения всех переменных при запуске программы считаются неопределенными, и хотя в большинстве компиляторов **системы «Исполнители»** они обнуляются при запуске, все же считается признаком

«хорошего тона» в программировании обнулить значение переменной, которая будет изменяться в процессе работы без предварительного присваивания ей какого-либо значения.

3) Прибавляем к **count** разряд единиц в двоичной записи числа **n**, то есть остаток от деления **n** на 2:

count = count + n % 2;

Строго говоря, мы могли бы не прибавлять предыдущее значение переменной **count** к остатку от деления, так как оно все равно было нулевым. Но мы поступили так для того, чтобы сделать код более однородным, далее это будет видно.

Учтя разряд единиц в двоичной записи **n**, мы должны отбросить его, чтобы исследовать число далее. Для этого разделим **n** на 2. На языке **системы «Исполнители»** это будет выглядеть так:

n = n / 2;

4) Теперь нам нужно еще два раза повторить **n. 3** , после чего останется единственный двоичный разряд числа **n**, который можно просто прибавить к счетчику без каких-либо дополнений:

count = count + n;

5) В результате в переменной **count** будет храниться количество единичных разрядов в двоичной записи исходного числа. Осталось лишь вывести ее на экран.

Код:

```
Программа
  {
int n, count;
print "Введите число в диапазоне от 0
до 15:";
input n;
  count = 0;
  count = count + n % 2;
  n = n / 2;
  count = count + n % 2;
  n = n / 2;
  count = count + n % 2;
  n = n / 2;
  count = count + n;
  print "Количество единиц в
введенном числе:";
print count;
       }
```

Программа работает правильно на всех вариантах правильных исходных данных, в чем несложно убедиться с помощью простой проверки.

Задача № 3. Проверить, является ли четырехзначное число палиндромом

Формулировка.

Дано четырехзначное число. Проверить, является ли оно палиндромом.

Примечание***:*** ***Палиндромом*** называется число, слово или текст, одинаково читающиеся слева направо и справа налево. Например, в нашем случае это числа 1441, 5555, 7117 и т.д. Примеры других чисел-палиндромов произвольной десятичной разрядности, не относящиеся к решаемой задаче: 3, 787, 11, 91519 и т.д.

Решение.

Для ввода числа с клавиатуры будем использовать переменную **n**. Вводимое число принадлежит множеству натуральных чисел и четырехзначно, поэтому оно заведомо больше 255, так что будем использовать тип **int**.

Какими же свойствами обладают числа-палиндромы? Из указанных примеров легко увидеть, что в силу своей одинаковой «читаемости» с двух сторон в них равны первый и последний разряд, второй и предпоследний и т. д. вплоть до середины.

Причем если в числе нечетное количество разрядов, то серединную цифру можно не учитывать при проверке, так как при выполнении названного правила число является палиндромом вне зависимости от ее значения.

В нашей же задаче все даже несколько проще, так как на вход подается четырехзначное число. А это означает, что для решения задачи нам нужно лишь сравнить первую цифру числа с четвертой и вторую цифру с третьей.

Если выполняются оба эти равенства, то число – палиндром.

Остается только получить соответствующие разряды числа в отдельных переменных, а затем, используя условный оператор, проверить выполнение обоих равенств с помощью булевского (логического) выражения.

Однако не стоит спешить с решением. Может быть, мы сможем упростить выведенную схему?

Возьмем, например, уже упомянутое выше число 1441. Что будет, если разделить его на два числа двузначных числа, первое из которых будет содержать разряд тысяч и сотен исходного, а второе – разряд десятков и единиц исходного. Мы получим числа 14 и 41.

Теперь, если второе число заменить на его реверсную запись (это мы делали в **задаче 1**), мы получим два равных числа 14 и 14! Это преобразование вполне очевидно, так в силу того, что палиндром читается одинаково в обоих направлениях, он состоит из дважды повторяющейся комбинации цифр, и одна из копий просто повернута задом наперед.

Отсюда вывод: нужно разбить исходное число на два двузначных, одно из них реверсировать, а затем выполнить сравнение полученных чисел с помощью условного оператора **if**. Кстати, для получения реверсной записи второй

половины числа нам необходимо завести еще две переменные для сохранения используемых разрядов. Обозначим их как **a** и **b**, и будут они типа **int**.

Теперь опишем сам алгоритм:

1) Вводим число **n**.

2) Присваиваем разряд единиц числа **n** переменной **a**, затем отбрасываем его. После присваиваем разряд десятков **n** переменной **b** и также отбрасываем его:

a = n % 10;
n = n / 10;
b = n % 10;
n = n / 10;

3) Присваиваем переменной **a** число, представляющее собой реверсную запись хранящейся в переменных **a** и **b** второй части исходного числа **n** по уже известной формуле:

a = 10 * a + b;

4) Теперь мы можем использовать проверку булевского выражения равенства полученных чисел **n** и **a** с помощью оператора **if** и организовать вывод ответа с помощью ветвлений: *if (n == a) print «Да» elseprint «Нет»;*

Так как в условии задачи явно не сказано, в какой форме необходимо выводить ответ, мы будем считать логичным вывести его на интуитивно понятном пользователю уровне, доступном в средствах самого языка **системы «Исполнители»**.

Напомним, что с помощью оператора **print** можно выводить результат выражения булевского типа, причем при истинности этого выражения будет выведено слово «Истина», при ложности – слово «Ложь».

Тогда предыдущая конструкция с **if** может быть заменена на *Print (n == a);*

Код:

```
Программа
  {
intn, a, b;
print "Введите четырехзначное число:";
input n;
  a = n % 10;
  n = n / 10;
  b = n % 10;
n = n / 10;
a = 10 * a + b;
print "Введенное число является палиндромом? ";
  print (n == a);
      }
```

Задача № 4. Найти наибольший нетривиальный делитель натурального числа

Формулировка.

Дано натуральное число. Найти его наибольший нетривиальный делитель или вывести единицу, если такового нет.

Примечание 1: Делителем натурального числа **a** называется натуральное число **b**, на которое **a** делится без остатка. То есть выражение «**b** – делитель **a**» означает: **a / b = k**, причем **k** – натуральное число.

Примечание: нетривиальным делителем называется делитель, который отличен от 1 и от самого числа (так как на единицу и само на себя делится любое натуральное число).

Решение.

Пусть ввод с клавиатуры осуществляется в переменную **n**. Попробуем решить задачу перебором чисел. Для этого возьмем число, на единицу меньше **n,** и проверим, делится ли **n** на него. Если да, то выводим результат и выходим из цикла с помощью оператора **break**. Если нет, то снова уменьшаем число на 1 и продолжаем проверку.

Если у числа нет нетривиальных делителей, то на каком-то шаге проверка дойдет до единицы, на которую число гарантированно поделится, после чего будет выдан соответствующий условию ответ.

Хотя, если говорить точнее, следовало бы начать проверку с числа, равного **n / 2** (чтобы отбросить дробную часть при делении, если **n** нечетно), так как ни одно натуральное число не имеет делителей больших, чем половина этого числа.

В противном случае частное от деления должно быть натуральным числом между 1 и 2, которого просто не существует.

Алгоритм на естественном языке:

1) Ввод **n**.
2) Запуск цикла, при котором **i** изменяется от **n / 2** до 1. В цикле:

1. Если **n** делится на **i** (то есть остаток от деления числа **n** на **i** равен 0), то выводим **i** на экран и выходим из цикла с помощью **break**.

Код:

```
Программа
 {
inti, n;
print "Введите натуральное число:";
input n;
 for (i = n / 2; i>= 1; i = i - 1)
{
if (n % i == 0)
    {
print "Наибольший нетривиальный делитель введенного числа:";
print i;
     break;
     }
   }
     }
```

Задача № 5. Вывести на экран все простые числа до заданного

Формулировка.

Дано натуральное число. Вывести на экран все простые числа до заданного включительно.

Решение.

В решении данной задачи используется решение предыдущей.

Нам необходимо произвести тест простоты числа (обозначим его как **код 1**):

```
count = 0;
for (i = 1; i<= n; i = i + 1)
if (n % i == 0) count = count + 1;
print (count == 2);
```

Здесь **n** – проверяемое на простоту число. Напомним, что данный фрагмент кода в цикле проверяет, делится ли **n** на каждое **i** в отрезке от 1 до самого **n**, и если **n** делится на **i**, то увеличивает счетчик **count** на 1. Когда цикл заканчивается, на экран выводится результат проверки равенства счетчика числу 2.

В нашем же случае нужно провести проверку на простоту всех натуральных чисел от 1 до заданного числа (обозначим его как **n**).

Следовательно, мы должны поместить **код 1** в цикл по всем **k** от 1 до **n**.

Также в связи с этим необходимо заменить в **коде 1** идентификатор **n** на **k**, так как в данном решении проверяются на простоту все числа **k**.

Кроме того, теперь вместо вывода ответа о подтверждении/опровержении простоты **k** необходимо вывести само это число в случае простоты:

if (count == 2) print (k + " ");

Обобщая вышесказанное, приведем алгоритм на естественном языке:

1) Ввод **n**.
2) Запуск цикла, при котором **k** изменяется от 1до **n**. В цикле:
1. Обнуление переменной **count**.
2. Запуск цикла, при котором **i** изменяется от 1 до **k**. В цикле:
a. Если **k** делится на **i**, то увеличиваем значение переменной **count** на 1.
3. Если **count** = 2, выводим на экран число **k** и символ пробела.

Код:

```
Программа
 {
inti, k, n, count;
 print "Введите крайнюю верхнюю границу диапазона чисел:";
inputn;
println "Простые числа в диапазоне от 0 до заданного:";
for (k = 1; k <= n; k = k + 1)
   {
   count = 0;
   for (i = 1; i<= k; i = i + 1)
     if (k % i == 0)
       count = count + 1;
   if (count == 2)
println k;
   }
}
```

Вычислительная сложность.

В данной задаче мы впервые столкнулись с вложенным циклом (когда один цикл запускается внутри другого). Это означает, что наш алгоритм имеет нелинейную сложность (при которой количество выполненных операций равно размерности исходных данных (в нашем случае это **n** – количество чисел) плюс некоторое количество обязательных операторов).

Давайте посчитаем количество выполненных операций в частном случае.

Итак, пусть необходимо вывести все натуральные простые числа до числа 5. Очевидно, что проверка числа 1 пройдет в 1 + 2 шага, числа 2 – в 2 + 2 шага, числа 3 – в 3 + 2 шага и т.д. (прибавленная двойка к каждому числу – два обязательных оператора вне внутреннего цикла), так как мы проверяем делители во всем отрезке от 1 до проверяемого числа.

В итоге количество проведенных операций (выполненных операторов на низшем уровне) представляет собой сумму: 3 + 4 + 5 + 6 + 7 (также опущен обязательный оператор ввода вне главного (внешнего) цикла).

Очевидно, что при выводе всех простых чисел от 1 до **n** приведенная ранее сумма будет такой:

$$1 + 3 + 5 + 6 + ... + (\mathbf{n} - 1) + \mathbf{n} + (\mathbf{n} + 1) + (\mathbf{n} + 2),$$

где вместо многоточия нужно дописать все недостающие члены суммы. Очевидно, что это сумма первых (**n** + 2) членов арифметической прогрессии с вычтенным из нее числом 2.

Как известно, сумма **k** первых членов арифметической прогрессии выражена формулой:

$$S_k = \frac{a_1 + a_k}{2} k,$$

где $\mathbf{a_1}$ – первый член прогрессии, $\mathbf{a_k}$ – последний.

Подставляя в эту формулу наши исходные данные и учитывая недостающее до полной суммы число 2, получаем следующее выражение:

$$S_n = \frac{1+(n+2)}{2}(n+2) - 2 = \frac{(n+2)+(n+2)^2}{2} - \frac{4}{2} = \frac{n+2+n^2+4n+4-4}{2} = \frac{n^2+5n+2}{2} = \frac{1}{2}n^2 + \frac{5}{2}n + 1$$

Чтобы найти асимптотическую сложность алгоритма, отбросим коэффициенты при переменных и слагаемые с низшими степенями (оставив, соответственно, слагаемое с самой высокой степенью). При этом у нас остается член $\boldsymbol{n^2}$, значит, асимптотическая сложность алгоритма – $\mathbf{O(n^2)}$.

Конечно, в дальнейшем мы не будем так подробно находить асимптотическую сложность алгоритмов, а тем более вычислять количество требуемых операций, что интересно только теоретически.

На самом деле, конечно, нас интересует лишь порядок роста времени работы алгоритма (количества необходимых операций), который можно выявить из анализа вложенности циклов и некоторых других характеристик.

Задача № 6. Проверить, является ли заданное натуральное число совершенным

Формулировка.

Дано натуральное число. Проверить, является ли оно совершенным.

Примечание: совершенным числом называется натуральное число, равное сумме всех своих собственных делителей (то есть натуральных делителей, отличных от самого числа).

Например, 6 – совершенное число, оно имеет три собственных делителя: 1, 2, 3, и их сумма равна 1 + 2 + 3 = 6.

Решение.

Код основной части (назовем его **кодом 1**):

```
count = 0;
```

for (i = 1; i<= n; i = i + 1)
if (n % i == 0) count = count + 1;

Как видно, в этом цикле проверяется делимость числа **n** на все числа от 1 до **n**. Чтобы переделать этот код под текущую задачу, нужно вместо инкрементации (увеличения значения) переменной-счетчика прибавлять числовые значения самих делителей к некоторой переменной для хранения суммы (обычно ее мнемонически называют **sum**, что в пер. с англ. означает «сумма»). В связи с этим оператор *if (n % i == 0) count = count + 1;* в **коде 1** теперь уже будет выглядеть так: *if (n % i == 0) sum = sum + i;*

Кроме того, чтобы не учитывалось само число **n** при суммировании его делителей (насколько мы помним, этот делитель не учитывается в рамках определения совершенного числа), цикл должен продолжаться не до **n**, а до **n – 1**.

Правда, если говорить точнее, то цикл следовало бы проводить до **ndiv 2**, так как любое число **n** не может иметь больших делителей, иначе частное от деления должно быть несуществующим натуральным число между 1 и 2.

Единственное, что останется теперь сделать, – это вывести ответ, сравнив число **n** с суммой его делителей **sum** как результат булевского выражения через **print**:

print (n == sum);

Код:

```
Программа
  {
inti, n, sum;
  print "Введите натуральное число:";
inputn;
sum = 0;
  for (i = 1; i<= n / 2; i = i + 1)
    if (n % i == 0) sum = sum + i;
print "Является введенное числ совершенным?
";
  print (n == sum);
       }
```

Задача № 7. Проверить монотонность последовательности цифр числа

Формулировка.

Дано натуральное число **n**. Проверить, представляют ли цифры его восьмеричной записи строго монотонную последовательность. При этом последовательность из одной цифры считать строго монотонной.

Примечание: В математике строго возрастающие и строго убывающие последовательности называются строго монотонными. В строго возрастающей последовательности каждый следующий член **больше** предыдущего. Например: 1, 3, 4 ,7, 11, 18. В строго убывающей последовательности каждый следующий член **меньше** предыдущего. Например: 9, 8, 5, 1.

Решение.

Здесь нам нужно будет последовательно получить разряды восьмеричной записи числа, двигаясь по записи числа справа налево.

Как мы уже знаем, последний разряд числа в восьмеричной системе счисления есть остаток от деления этого числа на 8. Попытаемся определить несколько общих свойств строго возрастающих (обозначим пример как 1) и строго убывающих (обозначим как 2) последовательностей (для наглядности будем сразу брать восьмеричные последовательности):

1) 3, 4, 5, 8, 9, 11.

2) 8, 7, 3, 2, 0.

Для начала введем в рассмотрение некоторую формулу, обозначим ее как (I):

$$delta_i = a_i - a_{i+1},$$

где a_i – член заданной последовательности с индексом i. Нетрудно понять, что эта формула определяет разность между двумя соседними элементами: например, если $i = 5$ (то есть мы рассматриваем пятую разность), формула будет выглядеть так: $delta_5 = a_5 - a_6$.

При этом стоит учитывать множество всех значений, которые может принимать i. Например, для последовательности (1) i может принимать значения от 1 до 5 включительно, для последовательности (2) – от 1 до 4 включительно.

Легко проверить, что для всех других i формула (I) не имеет смысла, так как в ней должны участвовать несуществующие члены последовательности.

Найдем все $delta_i$ для последовательности (1): $delta_1 = 3 - 4 = -1$, $delta_2 = 4 - 5 = -1$, $delta_3 = 5 - 8 = -3$, $delta_4 = 8 - 9 = -1$, $delta_5 = 9 - 11 = -2$.

Как видим, они все отрицательны. Нетрудно догадаться, что это свойство сохраняется для всех строго возрастающих последовательностей.

Выпишем все $delta_i$ для последовательности (2), не расписывая при этом саму формулу: 1, 4, 1, 2. Видим, что все они положительны.

Кстати, весьма примечательно, что в математическом анализе определение монотонной функции дается в терминах, подобных используемым в нашей формуле (I), которая рассматривается там несколько более гибко, а $delta_i$ при этом называется *приращением* и имеет несколько иное обозначение.

Можно обобщить сказанное тем, что последовательность *приращений* показывает, на какую величину *уменьшается* каждый член исследуемой последовательности чисел, начиная с первого.

Понятно, что если каждый член числовой последовательности уменьшается на положительную величину, то эта последовательность строго убывает и т.д.

Из всех этих рассуждений делаем вывод о том, что числовая последовательность является строго монотонной (то есть строго возрастающей или строго убывающей) тогда и только тогда, когда $delta_i$ имеют один и тот же знак для всех i.

Таким образом, мы вывели понятие, которое можно проверить с помощью последовательности однотипных действий, то есть циклической обработки.

Теперь нам необходимо попробовать унифицировать проверку знако постоянства всех *delta* для последовательностей обоих видов. Для этого рассмотрим произведение каких-либо двух *delta* в последовательностях (1) и (2).

Примечательно, что оно положительно как произведение чисел одного знака. Таким образом, мы можем в любой последовательности взять $delta_1$ и получить произведения его со всеми остальными *delta* (обозначим эту формулу как (II)):

$$p = delta_1 * delta_i$$

Если все p положительны, то последовательность строго монотонна, а если же возникает хотя бы одно отрицательное произведение p, то условие монотонности нарушено.

Теперь перенесем эти рассуждения из математики в программирование и конкретизируем их на последовательность цифр восьмеричной записи числа.

Обозначим идентификатором **delta** результат вычисления формулы (I) для двух текущих соседних членов последовательности.

Каким же образом двигаться по разрядам числа **n** и обрабатывать все **delta**? Сначала мы можем получить последнюю цифру числа **n** (назовем ее **b**), отбросить ее и получить предпоследнюю цифру **n** (назовем ее **a**), отбросить ее тоже, а затем вычислить **delta = a – b**.

Кстати, отметим, что при таком подходе мы будем для всех i находить $delta_i$, двигаясь по ним справа налево.

Начальному фрагменту этих действий соответствует следующий код:

```
b = n % 8;
n = n / 8;
a = n % 8;
n = n / 8;
delta = a - b;
```

Теперь мы можем войти в цикл с предусловием **n<> 0**. В каждом шаге цикла мы должны присвоить переменной **b** число **a**, затем считать следующий разряд в **a** и отбросить этот разряд в **n**.

Таким способом мы «сдвигаем» текущую пару: например, на первом шаге в примере (2) мы до входа в цикл использовали бы цифры 2 (в переменной **a**) и 0 (в переменной **b**), затем при входе в цикл скопировали бы 2 в **b** и 3 в **a** – таким

образом, все было бы готово для исследования знака по произведению. В связи с этим основной цикл будет выглядеть так:

```
while (n != 0)
{
b = a;
a = n % 8;
n = n / 8;
...
}
```

На месте многоточия и будет проверка знако постоянства произведений. Воспользовавшись формулой (II), мы заменим $delta_i$ в качестве текущего на саму разность **a – b**, чтобы не задействовать дополнительную переменную. В итоге, если теперь **delta * (a – b) <= 0**, то выходим из цикла:

```
if (delta * (a - b) <= 0) break;
```

Теперь рассмотрим развитие событий по завершении цикла:

1) Если произойдет выход по завершении цикла, то есть «закончится» число в связи с превращением его в 0 на некотором шаге, то значения **a**, **b** и **delta** будут содержать значения, подтверждающие строгую монотонность последовательности, что можно проверить с помощью вывода значения булевского выражения на экран.

2) Если в теле цикла произошел выход через условный оператор, то эти переменные будут содержать значения, с помощью которых выявлено условие нарушения строгой монотонности. Это значит, что на выходе из цикла ответ можно выводить в формате:

```
println (delta * (a - b) > 0);
```

Код:

```
Программа
 {
int n, a, b, delta;
 print "Введите число:";
inputn;
b = n % 8;
 n = n / 8;
 a = n % 8;
 n = n / 8;
 delta = a - b;
 while (n <> 0)
  {
  b = a;
  a = n % 8;
  n = n / 8;
  if (delta * (a - b) <= 0)
   break;
  }
```

```
print
"Монотонностьчисла?";
  print (delta * (a - b) > 0);
       }
```

Кстати, что будет при вводе числа **n**, меньшего 8, ведь его восьмеричная запись однозначна? Хотя наша цепочка делений еще до входа в цикл требует двух цифр в записи числа.

Посмотрим, как поведет себя программа при вводе числа **n** из единственной цифры *k*: сначала *k* идет в **b** (строка 9), затем отбрасывается в **n** (строка 10), которое теперь равно 0, затем в **a** идет число 0, так как **0 mod 8 = 0** (строка 11).

При этом строка 12 уже ничего не дает, так как **n**, которое сейчас равно нулю, присваивается значение **0 div 8 = 0**.

Далее вычисляется **delta = –*k*** (строка 13), затем игнорируется цикл, так как **n = 0** (строки 14–19), затем выводится на экран значение выражения ***–k* * –*k*> 0** (строка 20), которое истинно для любого действительного *k* кроме нуля, который и не входит в условие нашей задачи, так как **n** натуральное.

Как видим, вырожденный случай прошел обработку «сам собой», и для него не пришлось разветвлять программу, что выражает несомненный плюс.

Задача № 8. Получить каноническое разложение числа на простые сомножители

Формулировка.

Дано натуральное число **n (n> 1)**. Получить его каноническое разложение на простые сомножители, то есть представить в виде произведения простых сомножителей. При этом в разложении допустимо указывать множитель 1. Например, 264 = 2 * 2 * 2 * 3 * 11 (программе допустимо выдать ответ 264 = 1 * 2 * 2 * 2 * 3 * 11).

Решение.

Данная задача имеет достаточно красивое решение.

Из *основной теоремы арифметики* известно, что для любого натурального числа больше 1 существует его каноническое разложение на простые сомножители, причем это разложение единственно с точностью до порядка следования множителей.

Например, 12 = 2 * 2 * 2 и 12 = 3 * 2 * 2 – это одинаковые разложения. Рассмотрим каноническую форму любого числа на конкретном примере.

Например, 264 = 2 * 2 * 2 * 3 * 11.

Каким образом можно выявить эту структуру?

Чтобы ответить на этот вопрос, вспомним изложенные в любом школьном курсе алгебры правила деления одночленов, представив, что числа в каноническом разложении являются переменными. Как известно, если

разделить выражение на переменную в некоторой степени, содержащуюся в этом выражении в той же степени, оно вычеркивается в ее записи.

То есть, если мы разделим 264 на 2, то в его каноническом разложении уйдет одна двойка. Затем мы можем проверить, делится ли снова получившееся частное на 2. Ответ будет положительным, но третий раз деление даст остаток.

Тогда нужно брать для рассмотрения следующее натуральное число 3 – на него частное разделится один раз. В итоге, проходя числовую прямую в положительном направлении, мы дойдем до числа 11, и после деления на 11 **n** станет равно 1, что будет говорить о необходимости закончить процедуру.

Почему при таком «вычеркивании» найденных сомножителей мы не получим делимостей на составные числа?

На самом деле, здесь все просто: любое составное число является произведением простых сомножителей, меньших его.

В итоге получается, что мы вычеркнем из **n** все сомножители любого составного числа, пока дойдем до него самого в цепочке делений.

Например, при таком переборе **n** никогда не разделится на 4, так как «по пути» к этому числу мы вычеркнем из **n** все сомножители-двойки.

Алгоритм на естественном языке:

1) Ввод **n**.
2) Присвоение переменной **p** числа 2.
3) Вывод числа **n**, знака равенства и единицы для оформления разложения.
4) Запуск цикла с предусловием **n<> 1**. В цикле:

1. Если **m % p = 0**, то вывести на экран знак умножения и переменную p, затем разделить **n** на **p**, иначе увеличить значение **i** на 1.

Код:

```
Программа
 {
intn, p;
print "Введите число:";
input n;
 p = 2;
 print n;
 print " = 1";
 while (n != 1)
  {
  if ((n % p) == 0)
   {
   print " * ";
   print p;
   n = n / p;
   }
  else
   p = p + 1;
  }
    }
```

Задача № 9. Сформировать число из двух заданных чередованием разрядов

Формулировка.

Даны два натуральных числа одинаковой десятичной разрядности. Сформировать из них третье число так, чтобы цифры первого числа стояли на нечетных местах третьего, а цифры второго – на четных. При этом порядки следования цифр сохраняются. Например, при вводе 1234 и 5678 программа должна выдать ответ 15263748 (для наглядности разряды обоих чисел выделены разными цветами).

Решение.

Так как у чисел (обозначим их **a** и **b**) одинаковая десятичная разрядность, крайняя справа цифра у третьего числа (**c**, которое поначалу должно быть равно 0) всегда будет на четном месте, так как при его формировании мы работаем с длинами **a** и **b** как с числами одной четности, сумма которых всегда четна, и длина **c** как раз и есть позиция крайней справа цифры. Это значит, что формирование **c** нужно в любом случае начинать с последнего разряда **b**.

При этом каждый взятый из **a** или **b** разряд мы должны сместить на необходимую позицию влево, чтобы добавлять разряды **c**, используя операцию сложения.

Мы сделаем это с помощью вспомогательной переменной **z**, которая перед входом в цикл будет равна 1. В цикле же она будет умножаться на последний добытый разряд **b** (при этом выражение **z * b % 10** нужно прибавить к **c**), затем умножить **z** на 10 и проделать то же самое с последним разрядом **a** и снова умножить **z** на 10.

Кстати, при этом нужно не забыть своевременно отбросить уже рассмотренные разряды чисел.

Так как разрядность чисел неизвестна, нам нужен цикл с предусловием. В силу одинаковой десятичной разрядности **a** и **b** мы можем сделать условие по обнулению любого из них, так как второе при этом также обнулится. Возьмем условие **a<> 0**.

Таким будет основной цикл:

```
while (a != 0)
{
c = c + z * (b % 10);
z = z * 10;
b = b / 10;
c = c + z * (a % 10);
z = z * 10;
a = a / 10;
}
```

В итоге конечное число **c** будет сформировано в таком виде (все направления справа налево): первая цифра **b**, первая цифра **a**, вторая цифра **b**, вторая цифра **a** и так далее до самых последних разрядов слева.

Кстати, скобки в двух операторах нужны для правильного понимания компилятором приоритета выполняемых арифметических операций.

Без них **z** умножится на соответствующее число, и остаток от деления именно этого числа прибавится к **c**, что неправильно.

Алгоритм на естественном языке:

1) Ввод **a** и **b**.
2) Обнуление переменной **c**.
3) Присвоение переменной **z** числа 1.
4) Запуск цикла с предусловием **a<> 0**. В цикле:
   1. Прибавляем последний разряд **b** в текущий разряд **c**, определяемый с помощью множителя **z**.
   2. Умножаем **z** на 10.
   3. Избавляемся от последнего разряда в **b**.
   4. Прибавляем последний разряд **a** в текущий разряд **c** с помощью множителя **z**.
   5. Умножаем **z** на 10.
   6. Избавляемся от последнего разряда в **a**.
5) Вывод **c**.

Код:

```
Программа
 {
int c, z, a, b;
print "Введите числа a и b:";
input a, b;
  c = 0;
  z = 1;
  while (a != 0)
    {
    c = c + z * (b % 10);
    z = z * 10;
    b = b / 10;
    c = c + z * (a % 10);
    z = z * 10;
a = a / 10;
    }
print "Сформированное число с чередованием разрядов:";
print c;
}
```

Задача № 10. Вычислить экспоненту с заданной точностью

Формулировка.

Дано действительное число **x**. Вычислить значение экспоненциальной функции (то есть показательной функции e^x, где e – математическая константа,

$e \approx 2,718281828459045$) в точке **x** с заданной точностью **eps** с помощью ряда Тейлора:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Примечание 1: показательными называются функции вида a^x, где a – некоторое действительное число, x – независимая переменная, являющаяся показателем степени.

Примечание 2: ряд Тейлора – это представление функции в виде суммы (возможно, бесконечной) некоторых других функций по особым правилам (требующим детального математического обоснования, что в данном случае нам не нужно).

Решение.

Не вникая в теоретическую часть и считая представленную формулу корректной, попробуем разобраться в том, что же нам необходимо сделать для того, чтобы решить эту задачу:

1) Нам дана некоторая точка на оси *Ox*, и мы должны вычислить значение функции e^x в этой точке. Допустим, если **x = 4**, то значение функции в этой точке будет равно $e^4 \approx 2,71828^4 \approx 54,598$.

2) При этом вычисление необходимо реализовать с помощью заданной бесконечной формулы, в которой прибавление каждого очередного слагаемого увеличивает точность результата.

3) Точность должна составить вещественное число **eps**, меньшее 1. Это означает, что когда очередное прибавляемое к сумме слагаемое будет меньше **eps**, необходимо завершить вычисление и выдать результат на экран.

Это условие обязательно выполнится, так как математически доказано, что каждое следующее слагаемое в ряде Тейлора меньше предыдущего, следовательно, бесконечная последовательность слагаемых – это бесконечно убывающая последовательность.

Теперь разберемся с вычислением самого ряда. Очевидно, что любое его слагаемое, начиная со второго, можно получить из предыдущего, умножив его на **x** и разделив на натуральное число, являющееся номером текущего шага при последовательном вычислении (примем во внимание то, что тогда шаги нужно нумеровать с нуля).

Значение **x** нам известно на любом шаге, а вот номер текущего шага (будем хранить его в переменной **n**) придется фиксировать.

Создадим вещественную переменную **expf** (от англ. *exponentialfunction* – экспоненциальная функция) для накопления суммы слагаемых.

Будем считать нулевой шаг уже выполненным, так как первое слагаемое в ряду – константа 1, и в связи с этим **expf** можно заранее проинициализировать числом 1:

```
expf = 1;
```

Так как мы начинаем вычисления не с нулевого, а с первого шага, то также нужно инициализировать значения **n** (числом 1, так как следующий шаг

будет первым) и **p** (в ней будет храниться значение последнего вычисленного слагаемого):

```
n = 1;
p = 1;
```

Теперь можно приступить к разработке цикла.

С учетом заданной точности **eps** условием его продолжения будет **abs(p) >= eps**, где **abs(p)** – модуль числа **p** (модуль нужен для того, чтобы не возникло ошибки, если введено отрицательное **x**).

В цикле необходимо домножить **p** на **x** и доделить его на текущий номер шага **n**, чтобы обеспечить реализацию факториала в знаменателе, после чего прибавить новое слагаемое **p** к результату **expf** и увеличить **n** для следующего шага:

```
while (abs(p) >= eps)
{
p = p * x / n;
expf = expf + p;
n = n + 1;
}
```

После выхода из цикла нужно осуществить форматированный вывод результата **expf** на экран с некоторым количеством цифр после точки, например, пятью.

Отметим, что если при этом введенное **eps** содержало меньше 5 цифр после точки, то сформированное значение **expf** будет, соответственно, неточным.

Код:

```
Программа
 {
 float x, eps, expf, p;
intn;
print "Введите число x:";
inputx;
print "Введите точность eps:";
input eps;
expf = 1;
 n = 1;
 p = 1;
 while (abs(p) >= eps)
  {
  p = p * x / n;
expf = expf + p;
  n = n + 1;
}
print "Экспонента числа равна ";
printexpf;
}
```

Для Заметок