

Подготовка школьников к олимпиадам по информатике: стартовый уровень

Учебно-методическое пособие



г. Хабаровск, 2021 г.

Печатается по решению
научно-методического совета
КГАОУ ДО РМЦ
протокол № 1 от 15.02.2021 г.

Подготовка школьников к олимпиадам по информатике: стартовый уровень:
учебно-методическое пособие / Е. А. Редько. – Хабаровск: КГАОУ ДО РМЦ, 2021.
– 60 с.

Ответственный редактор: М.В. Gladунова
Ответственный за выпуск: О.А. Наумова
Дизайн обложки: Ю.А. Лубашова

Учебно-методическое пособие «Подготовка школьников к олимпиадам по информатике: стартовый уровень» содержит комплекс учебных модулей, необходимых для изучения начинающими участниками олимпиад по информатике и программированию. В пособии дается краткий обзор конструкций языка программирования C++, обсуждаются идеи решения задач для начинающих, алгоритмы обработки целых чисел, алгоритмы работы с цифрами числа, идеи динамического программирования. В пособии представлены кратко теоретические факты; разбор готовых примеров; задания для индивидуальной работы. Иллюстрация примеров выполнена с использованием среды программирования Code::Blocks 13.12.

Издание будет полезно обучающимся, учителям общеобразовательных организаций, педагогам дополнительного образования, студентам-бакалаврам педагогических направлений подготовки.

СОДЕРЖАНИЕ

Введение.....	2
Глава 1. Примеры решения олимпиадных задач из категории «Для начинающих».....	3
Глава 2. Основные алгоритмы теории делимости.....	10
Понятие делимости.....	10
Простые числа.....	12
Решето Эратосфена.....	14
Основная теорема арифметики.....	15
Алгоритм Евклида.....	17
Задачи для самостоятельного решения.....	21
Глава 3. Системы счисления.....	23
Младшая цифра числа.....	23
Просмотр всех цифр числа в обратном порядке.....	24
Реверс цифр числа.....	27
Двоичная запись числа (обратный порядок цифр).....	28
Просмотр цифр числа в прямом порядке.....	29
Задачи для самостоятельного решения.....	32
Глава 4. Динамическое программирование.....	34
Последовательность Фибоначчи.....	34
Двумерная динамика.....	36
Размен монетами.....	38
Задачи для самостоятельного решения.....	40
Справочный материал по языку программирования C++.....	42
Основные сведения о среде программирования Code::Blocks.....	42
Структура программы.....	44
Организация ввода/вывода.....	46
Простейшие вычисления. Целочисленная арифметика.....	47
Логический тип.....	48
Ветвление в программе.....	49
Циклы (повторение действий в программе).....	51
Функции.....	52
Заключение.....	55
Список источников.....	56

ВВЕДЕНИЕ

Учебно-методическое пособие предназначено для сопровождения педагогической деятельности учителей общеобразовательных организаций, педагогов дополнительного образования в рамках начальной подготовки обучающихся к решению задач олимпиадной информатики. Пособие также может быть использовано студентами-бакалаврами педагогических направлений подготовки в рамках изучения дисциплин «Методика обучения информатике», «Методика преподавания информатики в профильных классах» и других.

Методическое пособие состоит из 5 глав. Первая глава посвящена разбору некоторых простых задач школьного и муниципального этапов ВСОШ по информатике, а также задач из категории «Для начинающих» на сайте «Школа программиста». Показано, что обучающемуся достаточно владеть стартовым уровнем в программировании, чтобы приступить к решению данной группы задач.

В главах 2, 3 и 4 обсуждаются соответственно алгоритмы обработки целых чисел, алгоритмы работы с цифрами числа, идеи динамического программирования, которые доступны для понимания на начальном уровне владения языком программирования. Изучение алгоритмов, изложенных в главах 2-4, позволит успешно решать задачи указанной темы, ряд которых приведён в конце каждой главы. Так как часть задач выбрана из архива на сайте «Школа программиста», то обучающийся может отправить своё решение в тестовую систему сайта и проверить правильность составленной программы.

Последняя глава содержит краткие сведения о синтаксических особенностях выбранного языка и служит справочным материалом при прочтении остальных глав. Автором выбран для изложения материала язык программирования C++, как современный, мощный, востребованный и один из основных в олимпиадном программировании.

Учебно-методическое пособие не ставит целью изложить абсолютно все вопросы выбранных разделов, а позволяет обучающимся овладеть стартовым уровнем и выбрать дальнейший вектор изучения структур данных и алгоритмов их обработки.

Глава 1. Примеры решения олимпиадных задач

из категории «Для начинающих»

В данной главе представлены идеи решения и листинги программ на языке C++ к некоторым простым задачам школьного и муниципального этапов ВСОШ по информатике, а также к задачам из категории «Для начинающих» из архива задач на сайте «Школа программиста». Решение подобных задач не требует знания «специальных» алгоритмов и опирается на некоторую математическую модель и/или понимание алгоритмических конструкций языка программирования.

▪ **Клумбы (школьный этап, 7-8 класс, 2018 год)**

В парке разбили n клумб, каждая из которых имеет форму прямоугольника со сторонами a на b метров. Расстояние между соседними клумбами — 1 метр. Вокруг и между клумбами проложили дорожку шириной 1 метр в форме извилистой линии (рис. 1). Определите площадь дорожки.

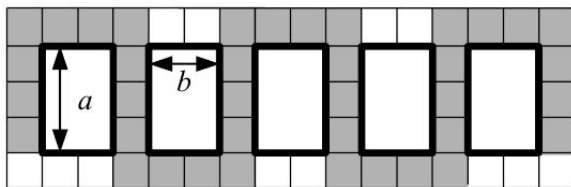


Рисунок 1. Схема спиральной дорожки в парке

На приведённом рисунке изображено $n = 5$ клумб размерами $a = 3$ и $b = 2$ метра. Клумбы изображены большими белыми прямоугольниками, дорожка закрашена серым цветом. Сторона одной клетки — 1 метр. В данном примере площадь дорожки равна 38 м^2 . Обратите внимание на концы дорожек.

Программа получает на вход три числа n — количество клумб, a , b — размеры 1 клумбы. Числа записаны в отдельных строках. Все числа — натуральные, не превосходящие 30 000.

Программа должна вывести одно целое число — длину спиральной дорожки.

Примеры входных и выходных данных

Ввод	Вывод
5	38
3	
2	

Решение

Для решения задачи отметим, что спиральную дорожку можно разбить на «горизонтальные» и «вертикальные» отрезки, причём длина каждого горизонтального отрезка — это ширина клумбы плюс два ($b + 2$), а длина каждого вертикального отрезка совпадает с длиной клумбы, то есть равна a (см. схему на рисунках 2-3).

Количество горизонтальных участков спиральной дорожки совпадает с количеством клумб (рис. 2), то есть равно n .

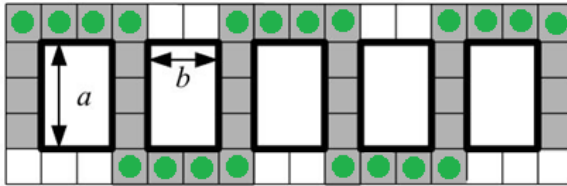


Рисунок 2. Горизонтальные отрезки спиральной дорожки

Количество вертикальных участков спиральной дорожки на одну больше, чем количество клумб (рис. 3), то есть равно $n + 1$.

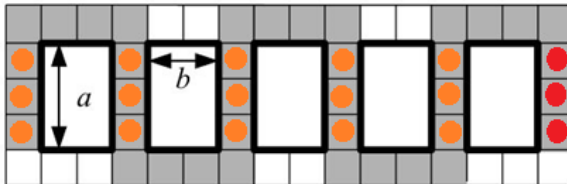


Рисунок 3. Вертикальные отрезки спиральной дорожки

Тогда суммарная длина всех отрезков спиральной дорожки может быть вычислена по формуле:

$$l = n \cdot (b + 2) + (n + 1) \cdot a$$

Приведём текст программы на языке C++ (рис. 4).

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

int main()
{
    int a,b,n;
    cin >> n >> a >> b;
    int L=n*(b+2)+(n+1)*a;
    cout << L;
    return 0 ;
}
```

Отметим, что задача максимально проста: требует минимальных сведений из языка программирования (структура программы, ввод/вывод данных, оператор присваивания); не предполагает выход за границы диапазона стандартного целого типа *int*, так как входные данные не превосходят 30000, а значит максимальный результат меньше, чем $2 \cdot 10^9$.

▪ **Конфеты (школьный этап, 9–11 класс, 2015 год)**

На столе стоят три вазы с конфетами. В левой вазе лежат *A* конфет, в средней вазе лежат *B* конфет, в правой вазе лежат *C* конфет. Лена съедает одну конфету из левой вазы, затем — одну конфету из средней вазы, затем из правой, средней, левой, средней, правой, средней и т. д. (слева направо, затем налево, опять направо и т.д.)

Если Лена хочет взять конфету из какой-то вазы, а конфет там нет, она расстраивается и идёт спать. Определите, сколько конфет съест Лена.

Программа получает на вход три целых неотрицательных числа *A*, *B*, *C* – количество конфет в левой, средней, правой вазе. Сумма трёх данных чисел не превосходит 2×10^9 .

Пример входных и выходных данных

Ввод	Вывод	Примечание
3 3 3	7	Лена съест конфеты из левой, средней, правой, средней, левой, средней, правой вазы. После этого она захочет съесть конфету из средней вазы, но в ней уже не осталось конфет.

Система оценивания

Решение, правильно работающее только для случаев, когда входные числа не превосходят 10, будет оцениваться в 40 баллов.

Решение, правильно работающее только для случаев, когда входные числа не превосходят 10000, будет оцениваться в 70 баллов (через цикл, моделируя процесс взятия конфет по одной).

Решение

Для получения решения заметим, что процесс взятия конфет содержит цикл «левая ваза, средняя ваза, правая ваза, средняя ваза», который затем повторяется. За один проход такого «цикла» число *A* уменьшается на 1, число *B* уменьшается на 2, число *C* уменьшается на 1. Посчитаем, сколько раз можно взять эти 4 конфеты: это — минимум из чисел *A*, $[B \div 2]$ и *C*. Запишем количество проходов цикла в переменную *k* и уменьшим значение переменных *A* и *C* на *k* (так как за один проход такого «цикла» числа *A* и *C* уменьшаются на 1), а значение переменной *B* на $2k$ (так как за один проход

«цикла» Лена дважды берёт конфету из средней вазы, то есть, число конфет уменьшается на 2). За k исполнений цикла суммарно будет взято $4k$ конфет.

Теперь посмотрим на остатки в вазах после того, как совершено k «проходов» по вазочкам с конфетами. Если $A = 0$, то нельзя на следующем шаге взять конфету из первой вазы, и ответом будет $4k$. Если $B = 0$, то будет взята конфета из первой вазы, но во второй вазе конфеты уже закончились, поэтому ответ будет $4k + 1$. Если же $C = 0$, то, аналогично, можно взять ещё две конфеты из левой и средней вазы, и ответ будет $4k + 2$. Наконец, если все эти условия не выполнены, то ответ будет $4k + 3$.

Решение программы на языке C++ (рис. 5).

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

int main()
{
    int A,B,C;
    cin >> A >> B >> C;
    int K=min(min(A,B/2),C);
    A-=K;
    B-=2*K;
    C-=K;
    if (A==0)
        cout << 4*K;
    else if (B==0)
        cout << 4*K+1;
    else if (C==0)
        cout << 4*K+2;
    else
        cout << 4*K+3;

    return 0 ;
}
```

Рисунок 5. Решение задачи «Конфеты»

▪ **Бесконечный пластилин (муниципальный этап, 7–8 класс, 2016 год)**

У малыша имеется $N \geq 1$ пластилина. Из него малыш катает шарики массой $K \geq 1$ каждый. После этого из каждого шарика мама лепит фигурки животных массой $M \geq 1$ каждая (из каждого шарика мама лепит максимально возможное количество фигурок). Если от шариков после этого что-то остаётся, то оставшийся пластилин возвращают в общую пластилиновую массу. Если общей пластилиновой массы, которая получилась, достаточно для одного шарика, то малыш снова катает шарики, а мама — фигурки животных, и т.д.

Напишите программу, которая вычислит, какое количество фигурок может быть получено из исходных N пластилина.

Формат входных данных

В единственной строке записаны через пробел три числа — N , K , M . Все числа натуральные и не превосходят 200. Гарантируется, что хотя бы одну фигурку слепить можно.

Формат выходных данных

Выведите одно число — количество фигурок, которое получится по такой технологии.

Примеры

Стандартный поток ввода	Стандартный поток вывода
752	2
952	4

Решение

Для определения количества шариков *balls*, которые катает малыш, выполним целочисленное деление N на K . Остаток от деления N на K сохраним в переменной r , чтобы добавить его к общей массе.

Такие же операции проделаем и для определения количества фигурок, которые лепит мама, учитывая при этом, что мама лепит фигурки из каждого шарика, скатанного малышом: т.е. K/M — это количество фигурок из одного шарика (целая часть от деления), тогда $(K/M)*balls$ — это общее количество фигурок, $(K\%M)*balls$ — остаток пластилина от шариков (остаток от деления).

Накопленный остаток пластилина требуется проверить — может ли малыш еще катать из него шарики? Если да, то рассмотренную выше последовательность действий повторяем (организуем цикл с предусловием).

Решение программы на языке C++ (рис. 6).

```

#include <iostream>
using namespace std;
int main()
{
    int N,K,M,balls,f,r;
    cin >> N >> K >> M;
    f=0;
    while (N>=K)
    {
        balls=N/K;
        r=N%K;
        f+=(K/M)*balls;
        r+=(K%M)*balls;
        N=r;
    }
    cout << f;
    return 0;
}

```

Рисунок 6. Листинг программы к задаче «Бесконечный пластилин»

▪ **Перепись (задача № 131 на сайте «Школа программиста»)**

В доме живёт N жильцов. Однажды решили провести перепись всех жильцов данного дома и составили список, в котором указали возраст и пол каждого жильца. Требуется найти номер самого старшего жителя мужского пола.

Входные данные

Во входном файле INPUT.TXT в первой строке задано натуральное число N — количество жильцов ($N \leq 100$). В последующих N строках располагается информация о всех жильцах: каждая строка содержит два целых числа: V и S — возраст и пол человека ($1 \leq V \leq 100$, $S = 0$ или 1). Мужскому полу соответствует значение $S = 1$, а женскому — $S = 0$.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать номер самого старшего мужчины в списке. Если таких жильцов несколько, то следует вывести наименьший номер. Если жильцов мужского пола нет, то выведите — 1.

Примеры

Стандартный поток ввода	Стандартный поток вывода
4 25 1 70 1 100 0 3 1	2
2 25 0 25 1	2

Решение

Решение данной задачи реализуется в виде однопроходного алгоритма с поиском максимума в заданной последовательности значений, причём количество обрабатываемых значений известно и равно N , значит, можно использовать цикл с параметром *for*, изменяя номера людей в списке с 1-го до -го.

Минимально возможный возраст для человека в списке начинается с 1 года, значит, будущему максимуму необходимо присвоить значение на 1 меньше, то есть 0.

Искомый номер самого старшего жителя мужского пола обозначим k и зададим начальное значение $k = -1$, что позволит правильно вывести результат в случае, если мужчин в списке не было, и, следовательно, в цикле не произошло замены переменной k на некоторый реальный номер.

Таким образом, задав начальные значения, запускаем цикл и на каждой итерации считываем очередную пару чисел в переменные v и s . Проверяем для них условия: первое условие о том, что проверяется персона мужского пола, может быть записана развернуто $s == 1$, но можно и сократить до s , так как единица (обозначающая мужчин) будет интерпретироваться как истинное значение.

Решение программы на языке C++ (рис. 7).

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int v,s,k=-1, max_v=0;
    for (int i=1; i<=n; i++){
        cin >> v >> s;
        if (s and v>max_v){
            max_v=v;
            k=i;
        }
    }
    cout << k;
    return 0 ;
}
```

Рисунок 7. Листинг программы к задаче «Перепись»

Таким образом, приведённые примеры задач разного типа учащиеся могут успешно решать при условии владения навыками использования операторов ввода/вывода, ветвления, циклов (краткий справочный материал по синтаксису языка представлен в конце пособия).

Глава 2. Основные алгоритмы теории делимости

Теория чисел относится к фундаментальным разделам математики. Вместе с тем ряд её задач имеет самое непосредственное отношение к практической деятельности, в первую очередь, к запросам криптографии. Криптографические алгоритмы шифрования с открытым ключом значительно стимулировали исследования классических задач теории чисел, приведя в ряде случаев к их решению, а также стали источником постановки новых фундаментальных проблем.

В настоящей главе будет уделено внимание алгоритмическим вопросам теории чисел, доступным для изучения школьниками, предложена реализация алгоритмов на языке C++. Вам предстоит знакомство со следующими алгоритмами:

- проверка числа на простоту;
- решето Эратосфена для получения значений простых чисел в указанном диапазоне $[1, N]$;
- разложение числа на простые множители;
- поиск НОД (наибольшего общего делителя).

Рассмотренные алгоритмы находят широкое применение в решении олимпиадных задач по информатике.

❖ Понятие делимости

Отношение делимости рассматривается в теории чисел. Теория чисел — важный раздел математики, изучающий свойства ЦЕЛЫХ чисел.

Определение 1. Говорят, что m делит n , если $m > 0$ и отношение $\frac{n}{m}$ представляет собой целое число, то есть существует целое k такое, что $n = k \cdot m$.

Обозначение: $m \setminus n$

Примеры:

- 1) Пусть $n = 12, m = 3$. Тогда $3 \setminus 12$ (число 3 делит число 12), так как $3 > 0$ и $12/3 = 4$ — целое.
- 2) Пусть $n = -21, m = 7$. Тогда $7 \setminus (-21)$ (число 7 делит число -21), так как $7 > 0$ и $-21/7 = -3$ — целое.

Определение 2. Говорят, что n кратно m , если отношение n/m представляет собой целое число, то есть $n = m \cdot k$ для некоторого целого k .

Возникает вопрос: чем отличаются понятия «делит» и «кратно»?

Ответ простой — в случае кратности m не обязательно быть положительным.

Примеры:

- 1) Про числа $n = -21$, $m = 7$ можно сказать: -21 кратно 7 , а также 7 делит -21 .
- 2) Пусть $n = 33$, $m = -11$. Тогда мы можем сказать, что 33 кратно -11 . Но недопустимо говорить, что -11 делит 33 , так как $-11 < 0$.

Итак, если m делит n , то можно определить целое число k такое, что $n = m \cdot k$. Число k называют целой частью от деления.

Если же число m не делит число n , то возникает понятие деления с остатком.

Пусть a и b — целые числа, причём b отлично от 0 (нуля). Деление с остатком a («делимого») на b («делитель») означает нахождение таких целых чисел q и r , что выполняется равенство: $a = b \cdot q + r$.

Таким образом, результатами деления с остатком являются *два целых числа*:

- q называется целой частью или неполным частным от деления,
- r называется остатком от деления.

На остаток налагается дополнительное условие: *остаток от деления должен быть неотрицательным числом* и по абсолютной величине меньше делителя. Это условие обеспечивает однозначность результатов деления с остатком для всех целых чисел, то есть существует единственное решение уравнения $a = b \cdot q + r$ при заданных выше условиях.

Примеры:

- 1) Пусть $a = 22$, $b = 5$. Тогда, выполнив деление с остатком, получим $22 = 5 \cdot 4 + 2$. То есть, целая часть от деления — $q = 4$, остаток от деления — $r = 2$.
- 2) Пусть $a = -22$, $b = 5$. Тогда, выполнив деление с остатком, получим $-22 = 5 \cdot (-5) + 3$. То есть, целая часть от деления — $q = -5$, остаток от деления — $r = 3$.

Упражнения

- 1) Выполните целочисленное деление для следующих чисел (запишите оба результата: q и r)
 - a) $a = 94$, $b = 33$;
 - b) $a = -94$, $b = 33$;
 - c) $a = -7$, $b = -13$;
 - d) $a = 90$, $b = -7$;
 - e) $a = 84$, $b = 7$.

2) Подберите пары «формулировка – термин»:

Результат операции деления с остатком a на b	b делит a
Операция нахождения для заданных целых чисел a и b (b не равно нулю) таких целых значений q и r ($0 \leq r < b$), что $a = b \cdot q + r$	Неполное частное
Если остаток от целочисленного деления a на b равен нулю, то...	Остаток от деления
Результат операции деления с остатком a на b , для которого выполняется условие $0 \leq r < b$	Деление с остатком

❖ Простые числа

В данном пункте рассмотрим понятие простого числа и алгоритм проверки на простоту. Понятие простого числа опирается на понятие отношения делимости.

Определение 1. Число p называется простым, если оно имеет ровно два делителя: единицу (1) и самого себя.

Определение 2. Число, не являющееся простым, называется составным.

Примечание: число 1 не относится ни к простым, ни к составным!

Наиболее очевидный алгоритм проверки того, что заданное число p является простым, это — **переборный алгоритм**. В переборном алгоритме рассматриваются последовательно все числа (d), которые могут быть потенциальными делителями исходного числа p . Для каждого такого числа проводится проверка — равен ли нулю остаток от деления p на d . Если хотя бы для одного из делителей выполнилось деление нацело, то число p не является простым.

Определим диапазон возможных делителей числа p .

Самый очевидный набор — это диапазон от 2 до $p - 1$. Можно ли его сократить? Конечно, можно, и самый простой вариант — сократить в два раза, то есть, рассмотреть возможные делители числа p среди чисел от 2 до $p/2$. Однако, есть и более оптимальный вариант, который для правой границы диапазона использует \sqrt{p} .

Докажем, что, если число p составное, то найдется хотя бы один делитель, не превосходящий \sqrt{p} :

Пусть у числа p есть делитель d такой, что $d > \sqrt{p}$. Тогда существует такое целое число k , что $p = d \cdot k$, или $k = \frac{p}{d}$. Докажем, что $k < \sqrt{p}$. Предположим противное, то есть, $k \geq \sqrt{p}$. Тогда из того, что $d > \sqrt{p}$ и $k \geq \sqrt{p}$ следует, что $d \cdot k > \sqrt{p} \cdot \sqrt{p}$ или $d \cdot k > p$. Но $d \cdot k = p$. Получили противоречие. Следовательно, $k < \sqrt{p}$. Другими словами, если у числа p есть делитель, больший \sqrt{p} , то найдется и делитель, меньший \sqrt{p} . Значит, если для p нет делителей от 2 до \sqrt{p} , то оно простое.

Представим рассмотренный алгоритм на языке C++ (рис.8).

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

int main()
{
    int64_t p;
    cin >> p;
    bool isPrime=true;
    for (int i=2; i*i<=p; i++)
        if (p%i==0) isPrime=false;
    if (isPrime) cout << p << " - is a prime number" << endl;
    else cout << p << " - is a composite number" << endl;
    return 0;
}
```

Рисунок 8. Переборный алгоритм проверки числа на простоту

В практических задачах данный алгоритм применяется редко ввиду его большой асимптотической сложности $O(\sqrt{n} \cdot \log^2 n)$, однако его применение оправдано в случае, если проверяемые числа относительно невелики, так как данный алгоритм довольно легко реализуем.

Упражнения

- 1) Пусть необходимо проверить, является ли число 625 простым или составным. Определите верхнюю границу такого диапазона делителей для числа 625, которого будет достаточно для ответа на поставленный вопрос.
- 2) Алгоритм, проверяющий, является ли данное число p простым или нет, перебирает в цикле возможные делители (i) из наименьшего диапазона, достаточного для ответа на поставленный вопрос. Этот диапазон задаётся от 2 до корня из числа p . Как в программе задать условие, по которому i не выйдет за верхнюю границу цикла, не используя функцию вычисления корня из p ? Запишите правильное выражение условия.

❖ Решето Эратосфена

Решето Эратосфена — это алгоритм, позволяющий найти все простые числа в отрезке $[1..n]$.

Идея проста — запишем ряд чисел $[1..n]$, и будем вычеркивать сначала все числа, делящиеся на 2, кроме самого числа 2, затем делящиеся на 3, кроме самого числа 3, затем на 5, затем на 7, 11, и все остальные простые до n .

Алгоритм приписывают древнегреческому математику Эратосфену Киренскому. Как и во многих случаях, здесь название алгоритма говорит о принципе его работы, то есть, решето подразумевает фильтрацию, в данном случае фильтрацию всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные (составные) исключаются.

Визуализацию алгоритма можно посмотреть на Википедии (*ссылка приведена в списке источников*).

Сложность алгоритма $O(n * \log(\log(n)))$. Доказательство можно прочитать в статье «Решето Эратосфена» на сайте Иванова Максима (*ссылка приведена в списке источников*).

Приведём реализацию алгоритма на языке C++ (*рис.9*).

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

int main()
{
    bool p[1000];
    int n;
    cin>>n;
    for (int i=0; i<=n; i++) p[i]=1;
    p[0]=p[1]=0;
    for (int i=2; i*i<=n; i++)
        if (p[i]==1)
            for (int j=i*i; j<=n; j+=i) p[j]=0;

    for (int i=0; i<=n; i++)
        if (p[i]==1) cout<<i<<" ";
    return 0;
}
```

Рисунок 9. Реализация алгоритма решета Эратосфена на языке C/C++

В массиве p для каждого i -го элемента (числа i) будет храниться одно из двух значений: 1 или 0. Единица будет признаком того, что число i простое, ноль — признаком составного числа i .

Первый цикл записывает для каждого числа i единицу (начальные установки, пока число не вычеркнуто). Следующий проход по массиву проверяет, является ли число простым (то есть, его не вычеркнули $p[i] == 1$) и вычеркивает все последующие числа j , кратные i (j получено как $i^2 + i$).

Упражнения

- 1) Выполняется алгоритм решета Эратосфена. Пусть переменная i изначально равна двум — первому простому числу. Числа от $i \cdot i$ до n , считая шагами по i , вычеркнуты: элементы массива помечены нулём. Каким будет следующее не зачёркнутое число (другими словами, каким будет индекс следующего элемента массива, не помеченного нулём)?
- 2) Выполняется алгоритм решета Эратосфена для $n = 20$. Пусть переменная i на текущем шаге равна трём (3). Числа от $i \cdot i$ до n , считая шагами по i вычеркнуты. На языке типов данных программы: элементы массива помечены нулём. Запишите индексы тех элементов массива, которые будут помечены нулём?
- 3) Выполняется алгоритм решета Эратосфена для $n = 300$. Пусть переменная i на текущем шаге равна 13. Числа от $i \cdot i$ до n , считая шагами по i вычеркивают. На языке типов данных программы: элементы массива помечают нулём. Запишите индекс того элемента массива, который будет первым помечен нулём?
- 4) Выполняется алгоритм решета Эратосфена для $n = 350$. Каким будет последнее простое число (в программе — переменная i), для которого будет производиться вычеркивание кратных ему чисел?

❖ Основная теорема арифметики

Теорема

Всякое целое число N , отличное от нуля, может быть представлено в виде произведения простых чисел, причём такое представление единственно с точностью до порядка сомножителей и их знаков.

Указанная теорема содержит два утверждения:

- 1) о существовании представления произвольного целого числа в виде произведения простых чисел;
- 2) о единственности такого представления.

Пример 1 (иллюстрирует возможность разложения)

Пусть дано число 420. Его разложение на простые множители будет выглядеть следующим образом:

$$420 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7.$$

Пример 2 (иллюстрирует понятие неразличимости сомножителей и порядка)

Пусть дано число 18. Выполним для него два разложения:

$$18 = 2 \cdot 3 \cdot 3$$
$$18 = (-3) \cdot (-2) \cdot 3$$

Два этих разложения считают неразличимыми, так как не имеет значения порядок множителей и их знаки (смена знака должна производиться одновременно для двух множителей).

Алгоритм разложения на простые множители может быть реализован следующим образом (*рис.10*): для введённого числа n (для которого будет производиться разложение на множители) задаём p — простой делитель, первое значение $p = 2$; далее, пока n нацело делится на p , выводим значение множителя p на экран и изменяем значение числа n , разделив нацело на p . Если n не делится нацело на p , то переходим к следующему делителю, увеличивая p на 1. Процесс продолжаем до тех пор, пока n не станет равно 1 (единице).

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

int main()
{
    int n, p=2;
    cin>>n;
    cout<<n<<"=";
    while (n!=1)
    {
        while (n%p==0)
        {
            n/=p;
            cout<<p<<"*";
        }
        p++;
    }
    return 0;
}
```

Рисунок 10. Разложение числа n на простые множители

Упражнение

В представленном алгоритме разложения на простые множители символ умножения (*) выводится на каждой итерации цикла, пока n делится на p . Вам необходимо самостоятельно доработать программу так, чтобы

после последнего простого множителя в разложении n символ умножения не выводился.

❖ Алгоритм Евклида

Вспомним понятие **делителя** для целого числа n : говорят, что m делит n , если $m > 0$ и отношение n/m представляет собой целое число.

Общим делителем натуральных чисел a и b называется число, на которое делятся оба числа a и b .

Наибольший общий делитель обозначается НОД (a, b) и определяется как наибольшее из общих делителей.

Для нахождения НОД (a, b) в школьном курсе математики используют разложение каждого из чисел a и b на простые множители.

Пример: найти НОД (2002, 420).

Решение:

- 1) разложим 2002 на простые множители 2) разложим 420 на простые множители

2002	2
1001	7
143	11
13	13
1	
завершено	

420	2
210	2
105	3
35	5
7	7
1	
завершено	

- 3) Выпишем разложения для обоих чисел:

$$2002 = 2 \cdot 7 \cdot 11 \cdot 13$$

$$420 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7$$

- 4) Выберем совпадающие множители: 2 и 7. Тогда НОД(2002, 42) = $2 \cdot 7 = 14$

Ответ: НОД(2002, 42) = 14

Существенный недостаток этого способа в том, что разложить большое число на простые множители не так просто, а, точнее, не так быстро.

Евклид в 7 книге «Начал» описывает алгоритм нахождения «общей меры двух чисел». Алгоритм описан геометрически, как нахождение общей меры двух отрезков. Он сводится к «последовательному отнятию» от большего отрезка

меньшего отрезка. Теперь этот алгоритм известен как **алгоритм Евклида** для нахождения наибольшего общего делителя двух натуральных чисел.

Визуализацию нахождения НОД(15, 24) можно посмотреть на сайте «Математические этюды» (ссылка указана в списке источников).

Рассмотрим идею модифицированного алгоритма Евклида, в котором от многократного вычитания выполнен переход к взятию остатка от деления (в дальнейших рассуждениях будем считать, что $a > b$).

Алгоритм Евклида опирается на тот факт, что если $\text{НОД}(a, b) = d$, то и $\text{НОД}(a \bmod b, b) = d$, где \bmod — общепринятое обозначение остатка от деления a на b .

Действительно, пусть a не делится на b нацело. Тогда существует два числа k и r , причём $r > 0$, такие, что возможно представление: $a = k \cdot b + r$.

Зная, что $\text{НОД}(a, b) = d$, то есть $d \mid a$ (d делит a) и $d \mid b$ (d делит b), мы должны требовать, что бы и второе слагаемое (r) правой части равенства $a = q \cdot b + r$ делилось на d .

А это и есть то число, которое мы ранее обозначили через $a \bmod b$.

Тогда, если r (или $a \bmod b$) не равно нулю ($r \neq 0$), то рассуждения можно продолжить (рис. 11).

$a = kb + r_1$, где $0 < r_1 \leq b - 1$	$\text{НОД}(a; b) = \text{НОД}(b; r_1)$
$b = k_1 r_1 + r_2$, где $0 < r_2 \leq r_1 - 1$	$\text{НОД}(r_1; b) = \text{НОД}(r_1; r_2)$
$r_1 = k_2 r_2 + r_3$, где $0 < r_3 \leq r_2 - 1$	$\text{НОД}(r_1; r_2) = \text{НОД}(r_2; r_3)$
...	...
$r_{n-1} = k_n r_n + r_{n+1}$, где $0 < r_{n+1} \leq r_n - 1$	$\text{НОД}(r_{n-1}; r_n) = \text{НОД}(r_n; r_{n+1})$
$r_n = k_{n+1} r_{n+1}$	$\text{НОД}(r_n; r_{n+1}) = \text{НОД}(r_{n+1}, 0) = r_{n+1}$

Рисунок 11. Шаги алгоритма Евклида

В итоге однажды мы получим нуль в качестве одного из остатков, т.к. остатки постоянно уменьшаются, а уменьшаться бесконечно они не могут (это целые неотрицательные числа).

И тогда последний, отличный от нуля остаток, и будет наибольшим общим делителем исходных чисел.

Пример: найти НОД (2002, 420).

Решение (рис. 12):

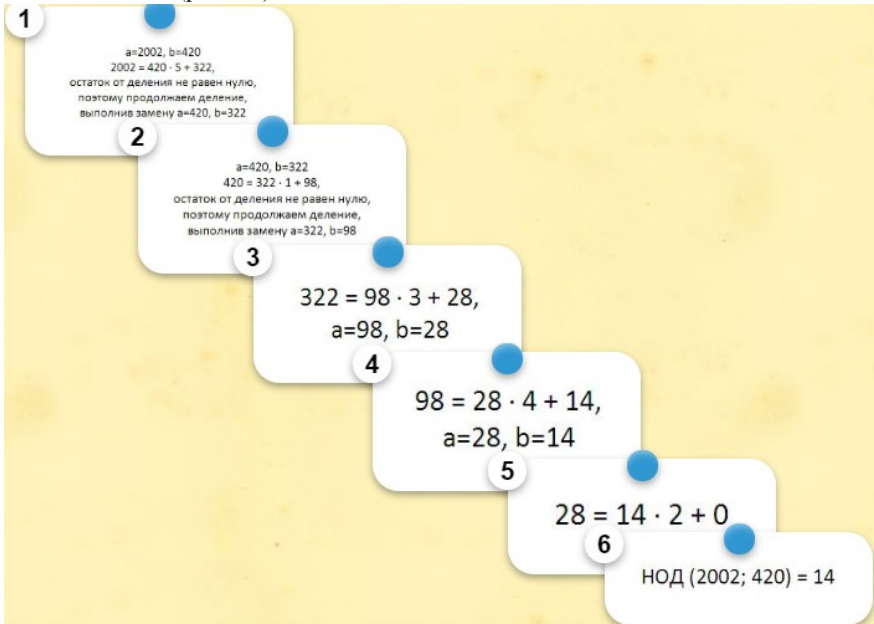


Рисунок 12. Шаги алгоритма Евклида для исходных чисел 2002 и 420

Алгоритм Евклида можно реализовать в виде рекурсивной функции, передавая ей в качестве измененных параметров b и $a \bmod b$ (рис. 13).

```
#include <iostream>
using namespace std;
int nod(int a, int b)
{
    if (b!=0) return nod(b, a%b);
    else return a;
}
int main()
{
    int a,b,d;
    cin>>a>>b;
    cout<<"NOD="<<nod(a, b);
    return 0;
}
```

Рисунок 13. Рекурсивная реализация алгоритма Евклида на языке C/C++

Рекурсия наглядно демонстрирует переход к следующему шагу алгоритма Евклида. Однако, можно обойтись и без неё. Приведём для сравнения не рекурсивную реализацию алгоритма Евклида (рис. 14).

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

int main()
{
    int a,b,d;
    cin>>a>>b;
    while (a!=0 && b!=0)
        if (a>b) a%=b;
        else b%=a;
    d=a+b;
    cout<<"NOD="<<d;
    return 0;
}
```

Рисунок 14. Итеративная реализация алгоритма Евклида (с использованием цикла)

❖ Задачи для самостоятельного решения

1. Дано некоторое натуральное число X . Найти ближайшее к нему простое число.
2. Два нечётных простых числа, отличающиеся на 2, называются близнецами. Например, числа 5 и 7. Напишите программу, которая будет находить все числа-близнецы на отрезке $[2; 1000]$.
3. Два нечётных простых числа, отличающиеся на 2, называются близнецами. Например, числа 5 и 7. Определить, являются ли числа, находящиеся рядом с заданным чётным числом, близнецами.
4. Совершенным числом называется число, равное сумме своих делителей, меньших его самого. Например, $28=1+2+4+7+14$. Определите, является ли данное натуральное число совершенным.
5. Совершенным числом называется число, равное сумме своих делителей, меньших его самого. Например, $28=1+2+4+7+14$. Найти 3 совершенных числа, больших заданного M .
6. Совершенным числом называется число, равное сумме своих делителей, меньших его самого. Например, $28=1+2+4+7+14$. Вывести ближайшее к данному числу N совершенное число.
7. Число называется совершенным, если оно равно сумме всех своих делителей, меньших его самого. Требуется найти все совершенные числа от M до N (задача № 364 из архива задач сайта «Школа программиста», требует «предподсчёта» с последующим хранением совершенных чисел в массиве).
8. Найти в интервале $[1, 1000]$ число, имеющее наибольшее количество делителей.
9. Два различных натуральных числа называются дружественными, если первое из них равно сумме делителей второго числа, за исключением самого второго числа, а второе равно сумме делителей первого числа, за исключением самого первого числа. Требуется найти все пары дружественных чисел, оба из которых принадлежат промежутку от M до N (задача № 371 из архива задач сайта «Школа программиста», требует «предподсчёта» с последующим хранением дружественных чисел в массиве).
10. Даны две рациональные дроби: a/b и c/d . Сложите их и результат представьте в виде несократимой дроби m/n .
11. Необходимо вывести все простые числа от M до N включительно (задача № 349 из архива задач сайта «Школа программиста», требует использования решета Эратосфена).
12. Написать программу, определяющую наименьшее общее кратное (НОК) чисел a и b (задача № 14 из архива задач сайта «Школа программиста»).
13. Катя решила пригласить к себе в гости n друзей. Так как её друзья очень любят фрукты, то в качестве угощения для них она купила m одинаковых

апельсинов. Она хочет разрезать каждый апельсин на одинаковое число равных долек так, чтобы их можно было распределить между гостями (сама Катя апельсины есть не будет), и всем гостям досталось поровну долек. Напишите программу, которая вычисляет минимальное количество долек, на которое необходимо разрезать каждый апельсин, чтобы были выполнены указанные выше условия (задача № 394 из архива задач сайта «Школа программиста»).

14. Известно, что любое чётное число, большее 2, представимо в виде суммы двух простых чисел, причём таких разложений может быть несколько. Впервые гипотезу о существовании данного разложения сформулировал математик Х. Гольдбах. Требуется написать программу, производящую согласно утверждению Гольдбаха, разложение заданного чётного числа. Из всех пар простых чисел, сумма которых равна заданному числу, требуется найти пару, содержащую наименьшее простое число (задача № 323 из архива задач сайта «Школа программиста»), требует использования решета Эратосфена).
15. Требуется вывести представление целого числа N в виде произведения простых чисел (задача № 354 из архива задач сайта «Школа программиста»).
16. Вася учится в третьем классе и сейчас он проходит тему «Простые дроби с натуральными числителем и знаменателем». Оказывается, что дробь называется правильной, если её числитель меньше знаменателя, и несократимой, если числитель и знаменатель являются взаимно простыми. Вася очень любит математику, и поэтому дома он решает много задач. В данный момент Вася ищет наибольшую правильную несократимую дробь, у которой сумма числителя и знаменателя равна N . Требуется написать программу, которая поможет Васе решить эту задачу (задача № 352 из архива задач сайта «Школа программиста»).
17. Имеются гири с массами: 1 г , 2 г , ..., $N\text{ г}$. Требуется написать программу, распределяющую эти гири на максимально возможное количество пар так, чтобы суммарный вес гирь в каждой паре выражался простым числом (задача № 448 из архива задач сайта «Школа программиста»).
18. Дано натуральное число N . Требуется представить его в виде суммы двух натуральных чисел A и B таких, что НОД (наибольший общий делитель) чисел A и B максимален (задача № 255 из архива задач сайта «Школа программиста»).

Работая с целыми числами устно или в тетради, мы легко можем «посмотреть» на отдельную цифру числа — оценить её значение, сделать вывод о её чётности, даже просто сложить все цифры.

Когда же мы «доверяем» программе обработку целочисленных значений, то помещаем обрабатываемое число из входного потока в переменную — ячейку памяти с именем. В этом случае «увидеть» отдельную цифру «напрямую» не представляется возможным — можно оперировать только значением числа, выполнять с ним различные арифметические операции.

Таким образом, «просмотр» цифр числа становится самостоятельной задачей, требующей написания программы по определённому алгоритму. Назначение данной главы — рассмотреть на примерах типовые задачи обработки цифр целого числа. Параграфы в главе будем задавать как формулировки задач.

Все рассмотренные далее алгоритмы опираются на развёрнутую форму записи числа и схему Горнера для вычисления значения многочлена или числа в произвольной системе счисления.

Напомним, что если число A записано в системе счисления с основанием d как $\overline{\alpha_n \alpha_{n-1} \dots \alpha_2 \alpha_1 \alpha_0}$, то значение A равно:

$$A = \alpha_n \cdot d^n + \alpha_{n-1} \cdot d^{n-1} + \dots + \alpha_2 \cdot d^2 + \alpha_1 \cdot d^1 + \alpha_0 \cdot d^0.$$

Тогда для вычисления значения A можно использовать алгоритм схемы Горнера сложности $O(n)$. Покажем последовательные действия, которые приводят к данному алгоритму:

$$\begin{aligned} A &= (\alpha_n \cdot d^{n-1} + \alpha_{n-1} \cdot d^{n-2} + \dots + \alpha_3 \cdot d^2 + \alpha_2 \cdot d + \alpha_1) \cdot d + \alpha_0 = \\ &= ((\alpha_n \cdot d^{n-2} + \alpha_{n-1} \cdot d^{n-3} + \dots + \alpha_3 \cdot d^1 + \alpha_2) \cdot d + \alpha_1) \cdot d + \alpha_0 = \dots \\ &= (((\alpha_n \cdot d + \alpha_{n-1}) \cdot d + \dots + \alpha_3) \cdot d + \alpha_2) \cdot d + \alpha_1) \cdot d + \alpha_0. \end{aligned}$$

Пример схемы Горнера для числа 3594:

$$3594_{10} = ((3 \cdot 10 + 5) \cdot 10 + 9) \cdot 10 + 4.$$

❖ Младшая цифра числа

Пример 1. Для заданного целого числа вывести на экран значение его младшей цифры (в разряде единиц).

Рассмотрим, в качестве примера, число 1854. Согласно развёрнутой форме записи числа, получим: $1854 = 1 \cdot 10^3 + 8 \cdot 10^2 + 5 \cdot 10 + 4$. Так как нас интересует только последняя цифра 4, и мы увидели её в качестве самостоятельного слагаемого, скомпонуем остальные слагаемые следующим образом: $1854 = 185 \cdot 10 + 4$. Полученная запись соответствует разложению числа 1854 при целочисленном делении на 10.

Итак, идея решения проста и опирается на целочисленное деление: если a — делимое, b — делитель, то существуют числа: целое q и неотрицательное целое r , такие, что:

$$a = b \cdot q + r,$$

где q называют **целой частью от деления**, r — **остатком от деления**.

А теперь вспомним, что в языках программирования есть две различные операции деления для целых переменных:

- целая часть от деления (в C++ обозначается /),
- остаток от деления (в C++ обозначается %).

То есть, при делении числа 1854 на 10, получим $q = 1854/10 = 185$, $r = 1854 \% 10 = 4$. В данном примере нам понадобится только второй результат — остаток от деления (рис. 15).

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int d = n % 10;
    cout << d;
    return 0;
}
```

Рисунок 15. Отделение младшей цифры десятичной записи числа

Комментарии к программе (рис. 15): исходное число хранится в переменной n и вводится с клавиатуры; для вычисления младшей цифры числа n введена вспомогательная переменная d (от слова digit — цифра), которой присваивается результат операции «остаток от деления» исходного числа n на 10; далее значение d выводится на экран.

❖ Просмотр всех цифр числа в обратном порядке

Пример 2. Для заданного целого числа вывести на экран значения всех его цифр (в обратном порядке, разделяя пробелом).

Обобщим приём получения младшей цифры числа. Для этого будем «избавляться» от младшей цифры в записи числа n после того, как её значение выведено на экран. Ранее мы заметили, что при делении числа 1854 на 10, получаем целую часть $q = 1854/10 = 185$ — именно это число необходимо для повторного отделения младшей цифры, а значит, его надо записать в переменную

n . Итак, для числа $n = 1854$ получим следующую последовательность однотипных действий (Таблица 1).

Таблица 1. Пошаговое выполнение действий для отделения цифр числа

Номер шага	Действия	Значение переменной n	Значение, выводимое на экран
		1854	
1	<code>cout<< n % 10<< "\n"; n /= 10;</code>	185	4
2	<code>cout<< n % 10<< "\n"; n /= 10;</code>	18	5
3	<code>cout<< n % 10<< "\n"; n /= 10;</code>	1	8
4	<code>cout<< n % 10<< "\n"; n /= 10;</code>	0	1

После того, как значение переменной n стало равным нулю, деление остановим (все цифры в десятичной записи числа получены). Листинг программы (рис 16).

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    while (n > 0)
    {
        cout << n % 10 << " ";
        n /= 10;
    }
    return 0;
}
```

Рисунок 16. Отделение последовательно всех цифр десятичной записи числа, начиная с младшей цифры

Примечание: рассмотренный алгоритм является базовым при решении множества задач, в которых требуется ответить на вопрос о значениях цифр заданного целого числа. Например: все ли цифры числа нечётные/чётные; есть ли в

записи числа хоть одна чётная/нечётная цифра; сколько цифр 7 (или любая другая) встречается в десятичной записи числа и так далее. Решение таких задач требует проверки того или иного условия для каждой вычисленной цифры и фиксации положительного ответа на вопрос, например, через увеличение счетчика, или перевода логической переменной в противоположное значение.

В качестве примера рассмотрим листинг программы, которая определяет, есть ли в десятичной записи числа хоть одна чётная цифра (рис. 17).

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    bool t = false;
    while (n > 0)
    {
        if (n % 2 == 0) t = true;
        n /= 10;
    }
    if (t) cout << "YES";
    else cout << "NO";
    return 0;
}
```

Рисунок 17. Отделение последовательно всех цифр десятичной записи числа, начиная с младшей цифры, с проверкой четности

Комментарии к программе (рис. 17):

- до цикла логической переменной t задано значение $false$, которое будем в дальнейшем интерпретировать как отсутствие чётных цифр в записи числа;
- цикл $while()$ позволяет работать с числом, пока оно не станет равным нулю;
- в теле цикла (где проверяем каждую цифру на чётность) можно было бы написать более очевидное условие $(n \% 10) \% 2 == 0$, по которому видно, что сначала отделяется цифра числа: $n \% 10$, а далее её значение проверяется на чётность (равенство нулю остатка от деления на 2). Но с математической точки зрения такая запись будет избыточной, так как чётность самого числа обеспечивается как раз чётностью младшей цифры. Итак, если число, а значит, и младшая цифра чётны, то значение логической переменной становится равным $true$;
- алгоритм можно усовершенствовать, если добавить оператор выхода из цикла $break$ в случае истинности условия $n \% 2 == 0$, чтобы не проверять все остальные цифры (ведь по условию задачи достаточно найти хотя бы одну чётную цифру);

- после проверки всех цифр и выхода из цикла, условным оператором проверяем значение логической переменной t — если оно равно *true* (то есть, встретилась хотя бы одна чётная цифра), то выводим на экран ответ YES, иначе — NO.

❖ Реверс цифр числа

Пример 3. По заданному целому числу получить новое число, в котором порядок цифр изменён на противоположный.

При решении данной задачи важно понимать, что требуется не просто увидеть на экране реверсную запись числа, а именно записать в новую переменную число, в котором порядок цифр изменён на противоположный.

Для этого будем использовать дополнительную переменную, например, *revers*, значение которой будет формироваться постепенно, на каждой итерации цикла.

Допустим, что исходное число $n = 6$ однозначное, тогда наш цикл отделит одну цифру 6, значение которой и надо записать в *revers*. Задача на этом будет решена.

Теперь представим, что к числу n дописали слева ещё одну цифру — разряд десятков, $n = 56$. Это значит, что в числе *revers* старое значение, равное 6, должно «перейти» в разряд десятков, или увеличиться в десять раз, после чего нужно просто прибавить значение ещё одной, полученной из числа n , цифры 5.

Эти же рассуждения будут верны для последовательного просмотра цифр исходного числа справа налево (рис. 18).

n=3256				revers=0												
3	2	5	6				0	+	6	=				6	→	
3	2	5		→	6	0	+	5	=				6	5	→	
3	2			→	6	5	0	+	2	=			6	5	2	→
3				→	6	5	2	0	+	3	=		6	5	2	3

Рисунок 18. Пошаговое формирование числа с противоположным порядком цифр

Далее можно действовать аналогично — получая из числа n значение цифры из каждого следующего разряда, необходимо снова увеличивать в 10 раз значение *revers* и прибавлять текущую цифру числа n .

Листинг программы (рис. 19).

```

#include <iostream>
using namespace std;
int main()
{
    int n, revers = 0;
    cin >> n;
    while (n > 0)
    {
        int d = n % 10;
        revers = revers * 10 + d;
        n /= 10;
    }
    cout << revers;
    return 0;
}

```

Рисунок 19. Листинг программы пошагового формирования числа *revers* с порядком цифр, обратным по отношению к исходному числу *n*

❖ Двоичная запись числа (обратный порядок цифр)

Пример 4. Для заданного целого числа вывести на экран значения всех его цифр в двоичной записи (в обратном порядке).

Решение данной задачи отличается от примера 2 только делителем — если для получения цифр в десятичной записи числа использовалась операция целочисленного деления на 10 (рис. 16), то для получения двоичных цифр будем делить, соответственно, на 2 (рис. 20).

```

#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    while (n > 0)
    {
        cout << n % 2 << " ";
        n /= 2;
    }
    return 0;
}

```

Рисунок 20. Отделение последовательно всех цифр двоичной записи числа, начиная с младшей цифры

Таким образом, видим, что алгоритм просмотра цифр (рис. 16, 20) легко обобщается на случай произвольной системы счисления.

Примечание 1 (словесный алгоритм перевода десятичного числа в систему счисления с основанием d): для перевода десятичного числа в систему счисления с основанием d нужно делить это число на d , фиксируя остаток на каждом шаге, пока не получится ноль. Затем выписать найденные остатки в обратном порядке. [Поляков К.Ю., 8 класс]

Примечание 2: те, кто знаком с операциями побитового сдвига, могут заменить целочисленное деление на 2 ($n/=2$) сдвигом вправо на 1 бит: $n = n >> 1$. Тогда переменную n необходимо объявить как беззнаковую (*unsigned int*), так как не всегда гарантируется корректная работа побитовых операторов с целочисленными типами *signed* в C++.

❖ Просмотр цифр числа в прямом порядке

Пример 5. Просмотр цифр числа в прямом порядке (слева направо).

До сих пор мы решали задачи, отделяя цифры числа последовательно от младшей к старшей (справа налево). Как же быть, если необходимо просматривать цифры в прямом порядке — от старшей к младшей?

Подходы к решению данной задачи различны, причём некоторые требуют использования дополнительно структур данных. Рассмотрим два варианта решения задачи — с предварительным вычислением количества цифр числа и с использованием структуры данных «стек».

Вариант 1. Выполним «проход» по десятичным цифрам числа дважды — сначала для того, чтобы определить k — максимальную степень десяти, которая в этом числе содержится, и второй раз непосредственно для получения значения каждой цифры (рис. 21).

Важно позаботиться о дубликаты значения исходного числа n , так как «проход» по числу организован с делением его до тех пор, пока не будет получен ноль.

Итак, из листинга видим (рис. 21), что первый цикл *while()* работает с дубликатом числа в переменной $n2$ и, «отбрасывая» в очередной раз цифру числа $n2$, увеличивает k в 10 раз. Так, например, для двухзначного числа в переменной $n2$ значение $k = 10$, для трёхзначного числа в переменной $n2$ значение $k = 100$, для четырёхзначного числа в переменной $n2$ значение $k = 1000$ и так далее.

Следующий цикл *while()* предназначен для получения значения каждой цифры в записи числа. Но в данном случае старшая цифра (крайняя левая) — это целая часть от деления на k . Например, для числа 1854 на первой итерации цикла получим цифру из разряда «тысячи», разделив исходное число на 1000. Для следующей итерации цикла необходимо «отбросить» старшую цифру, понизив разрядность исходного числа, и также уменьшить в 10 раз k .

```

#include <iostream>
using namespace std;
int main()
{
    int n, n2, k = 1;
    cin >> n;
    n2 = n;
    while (n2 > 10)
    {
        k *= 10;
        n2 /= 10;
    }
    while (n > 0)
    {
        cout << n/k << " ";
        n %= k;
        k /= 10;
    }
    return 0;
}

```

Рисунок 21. Отделение последовательно всех десятичных цифр числа, начиная со старшей цифры

Пошаговое выполнение алгоритма для числа $n = 1854$ (рис. 22).

До цикла $n=1854$					$k=1000$					На экран	Уменьшаем k и n	
1	8	5	4	/	1	0	0	0	=	1	$n\%=k$ $k/=10$	→
→	8	5	4	/	→	1	0	0	=	8	$n\%=k$ $k/=10$	→
	→	5	4	/	→	1	0	=	5	$n\%=k$ $k/=10$	→	
		→	4	/	→	1	=	4				

Рисунок 22. Пример получения всех десятичных цифр числа 1854, начиная со старшей цифры

Вариант 2.

Идея решения проста, но требует вспомогательных средств, а именно некоторой структуры данных, которая будет хранить цифры, прежде чем мы выведем их на экран. В качестве такой вспомогательной структуры данных можно использовать массив, строку, стек.

Используя тот же цикл, что и в программе решения задачи 2 (рис. 16), будем последовательно отделять цифры числа, начиная с младшей, и сохранять их в структуре данных. Тогда в результате работы цикла мы получим массив/строку/стек цифр. Остаётся только вывести их в обратном порядке. И именно стек имеет в данном случае преимущество при выборе структуры данных.

Так как хранение данных в стеке организовано по принципу *LIFO* «Последний пришёл, первый вышел» (*LastIn – FirstOut*), то на экране увидим цифры в прямом порядке — от старшей к младшей.

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    int n, d=10;
    stack<int> digits;
    cin>>n;
    while (n>0)
    {
        digits.push(n%d);
        n/=d;
    }
    while (!digits.empty())
    {
        cout<<digits.top()<<" ";
        digits.pop();
    }
    return 0;
}
```

Рисунок 23. Использование стека для хранения цифр числа и дальнейшей их обработки в прямом порядке — от старшей к младшей

Обратите внимание, что в этой программе (рис. 23) мы не использовали в операциях целочисленного деления константу 10, а ввели переменную d , значение которой равно 10, и все необходимые операции выполняли уже с этой переменной: $n\%d, n/=d$. Именно этот приём позволит нам далее реализовывать алгоритмы перевода десятичного числа в произвольную систему счисления.

Действительно, придавая переменной d различные значения (уже отличные от 10), мы будем получать в качестве результата работы программы запись числа в новой системе счисления.

❖ Задачи для самостоятельного решения

1. Найдите количество единиц двоичной записи заданного числа N (задача № 22 из архива задач сайта «Школа программиста»).
2. Задано натуральное число N . Необходимо перевести его в k -ичную систему счисления ($k \leq 55$). Для обозначения цифр в случае, если $k > 9$, используйте дополнительно буквы латинского алфавита, например, $A = 10$, $B = 11$, $C = 12$ для 13-ричной системы счисления.
3. Наш любимый сисадмин Алексей установил новую ОС семейства Unix. Основные её особенности — это стабильность, надёжность, гибкость, масштабируемость и огромное количество идущего в стандартной поставке программного обеспечения. Одна из таких встроенных программ предназначена для сложения чисел, представленных в троичной системе счисления. Вы понимаете то, что Костя — известный тестер, и делом чести для него является найти ошибку в реализации столь сложной задачи. Помогите ему — напишите свою, абсолютно безошибочную версию «троичного калькулятора» (задача № 941 из архива задач сайта «Школа программиста»).
4. Задано натуральное число N . Необходимо найти разность между произведением и суммой его цифр в десятичной системе счисления.
5. Задано натуральное число N . Необходимо перевести его в k -ичную систему счисления и найти разность между произведением и суммой его цифр в этой системе счисления (задача № 59 из архива задач сайта «Школа программиста»)
6. Задана последовательность (S_1, S_2, \dots, S_N) целых чисел. Найдите элемент этой последовательности с наибольшей суммой цифр, выведите его значение и номер в исходной последовательности. Если таких элементов несколько, выберите наибольший из них.
7. Число Армстронга (также самовлюблённое число) — натуральное число, которое в данной системе счисления равно сумме своих цифр, возведённых в степень, равную количеству его цифр. Например, десятичное число 153 — число Армстронга, потому что $1^3 + 5^3 + 3^3 = 153$. Найдите все трёхзначные числа Армстронга.
8. Задано число в k -ичной системе счисления. Найти десятичное значение числа, используя схему Горнера.
9. Задано натуральное число N . Необходимо найти разность между произведением и суммой его цифр в пятеричной системе счисления.
10. Задана последовательность (S_1, S_2, \dots, S_N) целых чисел. Найдите элемент этой последовательности с наименьшей суммой цифр, выведите на экран его значение и номер в исходной последовательности. Если таких элементов несколько, выберите наибольший из них.

11. Число Армстронга (также самовлюблённое число) — натуральное число, которое в данной системе счисления равно сумме своих цифр, возведённых в степень, равную количеству его цифр. Например, десятичное число 153 — число Армстронга, потому что $1^3 + 5^3 + 3^3 = 153$. Найдите все четырёхзначные числа Армстронга.
12. Задано натуральное число N . Необходимо найти значение S^k , где S — сумма цифр числа N , k — количество его цифр.

Глава 4. Динамическое программирование

Динамическим программированием называется метод, который позволяет решать переборные задачи, опираясь на уже решённые подзадачи меньшего размера. При этом часто появляется необходимость, чтобы все решения можно было запомнить в таблице (т.е. данные, посчитанные однажды, не пересчитываются). Идеи динамического программирования очень похожи на идеи метода доказательства по индукции, известного из школьной программы математики. Сформулируем необходимые требования:

1. Существует известные решения для задачи с малой размерностью (аналогично проверке для минимального параметра в методе математической индукции).
2. Существует способ выразить решение задачи через подзадачи (возможно, отличные от исходной задачи) меньшей размерности. Это больше всего напоминает рекуррентное соотношение в математике.
3. Все решения подзадач должны запоминаться в таблице.

Требование меньшей размерности означает, что значение параметра (или одного из параметров, если их несколько) в подзадаче должно быть меньше, чем значение параметра в задаче, и в итоге последовательность подзадач должна сходиться к известным решениям (нулевому параметру, например). Если параметры подзадач уходят в бесконечность или возникает круговая зависимость, это означает, что метод динамического программирования неприменим. В некоторых случаях разумно подсчитать сумму всех параметров и требовать, чтобы на каждом шаге эта сумма уменьшалась (однако это не всегда возможно).

Задачи динамического программирования делятся на два типа: оптимизация целевой функции и подсчёт количества вариантов решения. В первом случае нам необходимо выбрать лучшее среди всех решений подзадач, во втором — просуммировать все количества решений подзадач. В случае если ищется решение, иногда бывает необходимо подсчитать не только сам ответ, но и восстановить решение полностью (записать, какие действия выполнялись на каждом шаге).

Итак, для использования метода динамического программирования нужно вывести рекуррентную формулу, связывающую решение задачи с решением подобных задач меньшей размерности, и определить простые базовые случаи.

❖ Последовательность Фибоначчи

Последовательность чисел Фибоначчи может быть проиллюстрирована *задачей о кроликах*: человек посадил пару кроликов в загон, окружённый со всех сторон стеной. Сколько пар кроликов за год может произвести на свет эта пара, если известно, что каждый месяц, начиная со второго, каждая пара кроликов производит на свет одну пару?

Решение:

Проведем рассуждения для общего случая. Считать кроликов будем парами. В произвольном k -ом месяце к каждой паре кроликов, которая уже посчитана в предыдущем $(k-1)$ -м месяце, потомство могут принести все пары кроликов, возраста не меньше двух месяцев, то есть посчитанных в $(k-2)$ -м месяце. Таким образом, общее число пар кроликов составит $P(k) = P(k-1) + P(k-2)$.

Теперь вернёмся к исходным данным: в первый месяц количество кроликов составляет одну пару $P(1) = 1$; а во второй месяц к этой паре должна добавиться еще пара кроликов $P(2) = 2$.

Примечание: уже для второго месяца количество пар кроликов можно рассчитать по рекуррентной формуле, если принять $P(0) = 1$ — количество пар в потомстве первой пары.

Алгоритм решения может быть реализован тремя способами:

- a) рекурсивной функцией (рис. 24);
- b) итеративно, с хранением вычисленных значений в массиве (рис. 25);
- c) итеративно, без использования массива, с пересчётом трёх переменных (рис. 26).

```
#include <iostream>
using namespace std;
int P(int k)
{
    if (k==0 || k==1) return 1;
    else return P(k-1)+P(k-2);
}
int main()
{
    cout<<P(12) - 1;
    return 0;
}
```

Рисунок 24. Реализация решения задачи о кроликах с использованием рекурсивной функции

```

#include <iostream>
using namespace std;
int main()
{
    int P[13];
    P[0]=1;
    P[1]=1;
    for (int k=2; k<=12; k++) P[k]=P[k-1]+P[k-2];
    cout<<P[12]-1;
    return 0;
}

```

Рисунок 25. Итеративная реализация решения задачи о кроликах с хранением вычисленных значений в массиве

```

#include <iostream>
using namespace std;
int main()
{
    int P0=1, P1=1, Pk;
    for (int k=2; k<=12; k++)
    {
        Pk=P0+P1;
        P0=P1;
        P1=Pk;
    }
    cout<<Pk-1;
    return 0;
}

```

Рисунок 26. Итеративная реализация решения задачи о кроликах с пересчётом трёх переменных

❖ Двумерная динамика

Задача

На прямоугольном поле $n \times m$, разбитом на ячейки-квадратики, живёт робот. Его задача состоит в прохождении этого поля от левого верхнего угла до правого нижнего (рис. 27). За 1 шаг робот может выполнять одну из двух команд: сместиться на 1 клетку вправо, или сместиться на 1 клетку вниз. Сколько возможных маршрутов для робота можно составить?

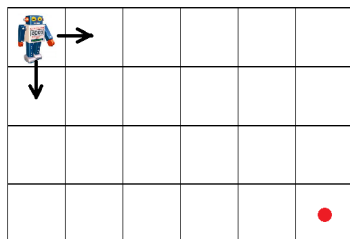


Рисунок 27. Поле робота

Решение

Принцип решения по-прежнему опирается на рассуждения для произвольной клетки. На клетку (x, y) можно попасть из клеток: $(x - 1, y)$ — левее текущей, или $(x, y - 1)$ — выше текущей. Рекуррентное соотношение выглядит так: $R(x, y) = R(x - 1, y) + R(x, y - 1)$. Дополнительно нужно учесть граничные значения: все клетки вида $(1, y)$ и $(x, 1)$ имеют только один маршрут — по прямой вниз или по прямой вправо. Для этих крайних клеток количество вариантов равно единице.

Очевидно, что все промежуточные вычисления будут записываться в двумерный массив. Ответом на задачу станет значение в правой нижней ячейке двумерного массива.

Приведём решение задачи на языке C++ (рис. 28).

```

#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int n,m;
    cin>>n>>m;
    vector <vector<int> > R(n);
    for (int i=0; i<R.size(); i++)
        R[i].resize(m);
    for(int i=0; i<n; i++)
        for(int j=0; j<m; j++)
        {
            if (i==0 || j==0) R[i][j]=1;
            else
                R[i][j]=R[i-1][j]+R[i][j-1];
        }
    cout<<R[n-1][m-1];
    return 0;
}

```

Рисунок 28. Решение «базовой» задачи на двумерную динамику

❖ Размен монетами

Задача

Пусть имеется множество номиналов монет $penny = \{p_1, p_2, \dots, p_n\}$ и денежная сумма S . Требуется набрать сумму S , используя, как можно, меньше монет. Количество монет одного номинала не ограничено.

Примеры:

- 1) Для размена суммы 12 монетами $penny = \{1, 2, 5\}$ достаточно трёх монет:
 $5 + 5 + 2 = 12$.
- 2) Если $penny = \{1, 3, 4\}$, то сумму $S = 6$ можно набрать двумя монетами:
 $3 + 3 = 6$.
- 3) Если $penny = \{1, 3, 4\}$, то сумму $S = 10$ можно набрать тремя монетами:
 $4 + 3 + 3 = 10$.

Примеры 2 и 3 наглядно демонстрируют, что жадный алгоритм для решения данной задачи не подходит.

Действительно, для второго примера согласно жадному алгоритму первой монетой будет 4. Но тогда оставшуюся сумму ($6 - 4 = 2$) можно набрать только двумя монетами, достоинством 1. Таким образом, получим 3 монеты для размена: $4 + 1 + 1 = 6$, а это не минимальное количество.

Самостоятельно покажите, какой результат даст жадный алгоритм для исходных данных третьего примера.

Решение

Воспользовавшись идеей динамического программирования, построим алгоритм, близкий к полному перебору, но при этом эффективный, а именно: будем брать каждую монету p_i и рассматривать результат решения уже для суммы $S - p_i$, выбирая лучшее (минимальное) решение.

Наиболее очевидной является рекурсивная реализация данной идеи (рис. 29).

Базой рекурсии является значение $change(0) = 0$, так как для размена нулевой суммы ни одна монета не требуется.

Также нужно учесть отрицательную денежную сумму, которая может возникать при рекурсивном обращении со значением $S - p_i$, так как номинал монеты может превысить оставшуюся для размена сумму.


```

#include <iostream>
#include <climits>
#include <set>
using namespace std;
set<int> penny;
int change(int s)
{
    if (s<0) return INT_MAX-1;
    if (s==0) return 0;
    int min_change=INT_MAX-1;
    for (auto p: penny)
        min_change=min(change(s-p)+1, min_change);
    return min_change;
}
int main()
{
    int n,p,S;
    cin>>S>>n;
    for (int i=0; i<n; i++){
        cin>>p;
        penny.insert(p);
    }
    cout<<change(S);
    return 0;
}

```

Рисунок 29. Решение задачи о размене суммы заданным набором монет через рекурсивную функцию

Отметим, что приведённая рекурсивная реализация решения задачи не эффективна, так как, во-первых, перебирает все возможные способы размена суммы, во-вторых, по несколько раз рассчитывает одни и те же значения, встретившиеся в различных комбинациях размена.

Воспользуемся требованием динамического программирования — запоминать вычисленные значения в массиве для последующего использования (рис. 30).

Фактически в массиве будут записаны количества разменных монет для каждой из сумм (i) от нулевой до S . Так как все денежные суммы перебираются подряд, то проверка условия $i - p \geq 0$ в такой реализации решения позволяет не выйти за границы массива. Для каждой последующей денежной суммы выбор наилучшего размена (наименьшего по количеству монет) опирается на те значения, которые можно получить из текущей суммы, вычитая достоинство одной из возможных монет p , то есть $change[i - p]$. Цикл `for (auto: penny)` перебирает все монеты в множестве.

```

#include <iostream>
#include <set>
#include <vector>
#include <climits>
using namespace std;

int main()
{
    set<int> penny;
    int p, n, S;
    cin >> S >> n;
    for (int i=1; i<=n; i++) {
        cin >> p;
        penny.insert(p);
    }
    vector<int> change(S+1);
    change[0]=0;
    for (int i=1; i<=S; i++)
    {
        change[i]=INT_MAX-1;
        for (auto p: penny)
            if (i-p>=0)
                change[i]=min(change[i], change[i-p]+1);
    }
    cout << change[S];
    return 0;
}

```

Рисунок 30. Решение задачи о размене суммы заданным набором монет с хранением вычисленных значений в массиве

Задание: самостоятельно модифицируйте алгоритм так, чтобы вывести в качестве результата не только количество монет, но и набор монет, составляющих оптимальное решение.

❖ Задачи для самостоятельного решения

1. Пицца — любимое лакомство Васи, он постоянно покупает и с удовольствием употребляет различные сорта этого великолепного блюда. Однажды, в очередной раз разрезая круглую пиццу на несколько частей, Вася задумался: на какое максимальное количество частей можно разрезать пиццу за N прямых разрезов? Помогите Васе решить эту задачу, определив максимальное число не обязательно равных кусков, которые может получить Вася, разрезая пиццу таким образом. (задача № 554 из архива задач сайта «Школа программиста»)
2. В прямоугольной таблице $N \times M$ (в каждой клетке которой записано некоторое число) в начале игрок находится в левой верхней клетке. За один ход ему разрешается перемещаться в соседнюю клетку либо вправо, либо

- вниз (влево и вверх перемещаться запрещено). При проходе через клетку с игрока берут столько у.е., какое число записано в этой клетке (деньги берут также за первую и последнюю клетки его пути). Требуется найти минимальную сумму у.е., заплатив которую игрок может попасть в правый нижний угол (задача № 120 из архива задач сайта «Школа программиста»).
3. При переработке радиоактивных материалов образуются отходы двух видов — особо опасные (тип А) и неопасные (тип В). Для их хранения используются одинаковые контейнеры. После помещения отходов в контейнеры последние укладываются вертикальной стопкой. Стопка считается взрывоопасной, если в ней подряд идёт более одного контейнера типа А. Стопка считается безопасной, если она не является взрывоопасной. Для заданного количества контейнеров N определить количество возможных типов безопасных стопок.
 4. Мальчик подошёл к платной лестнице. Чтобы наступить на любую ступеньку, нужно заплатить указанную на ней сумму. Мальчик умеет перешагивать на следующую ступеньку, либо перепрыгивать через ступеньку. Требуется узнать, какая наименьшая сумма понадобится мальчику, чтобы добраться до верхней ступеньки.
 5. Вова стоит перед лесенкой из N ступеней. На каждой из ступеней написаны произвольные целые числа. Первым шагом Вова может перейти на первую ступень или, перепрыгнув через первую, сразу оказаться на второй. Также он поступает и дальше, пока не достигнет N -ой ступени. Посчитаем сумму всех чисел, написанных на ступенях, через которые прошёл Вова. Требуется написать программу, которая определит оптимальный маршрут Вовы, при котором, шагая, он получит наибольшую сумму (задача № 329 из архива задач сайта «Школа программиста»).
 6. На прямой дощечке вбиты гвоздики. Любые два гвоздика можно соединить ниточкой. Требуется соединить некоторые пары гвоздиков ниточками так, чтобы к каждому гвоздику была привязана хотя бы одна ниточка, а суммарная длина всех ниточек была минимальна (задача № 121 из архива задач сайта «Школа программиста»).
 7. Организовать вывод на экран треугольника Паскаля до N -й строки.

❖ Основные сведения о среде программирования Code::Blocks

Code::Blocks (<http://www.codeblocks.org>) — свободная кросс-платформенная интегрированная среда разработки (программирования), написанная на C++. Code::Blocks поддерживает языки C и C++.

В состав Code::Blocks входят (*рис. 31*):

- **редактор исходных текстов** с подсветкой синтаксиса и функцией автозаполнения (*autocomplete*) — подсказки имен функций и переменных при наборе первых букв имени;

- **менеджер проектов** — позволяет быстро переключаться между открытыми проектами, выбирать для редактирования необходимые файлы, включённые в проект;

- **компилятор**, а точнее, набор компиляторов GCC (GNU Compiler Collection), являющийся свободным программным обеспечением;

- **компоновщик** (служит для автоматизации сборки) — связывает объектный код с кодами отсутствующих функций, чтобы создать исполняемый загрузочный модуль (без пропущенных частей). В результате работы компоновщика создаётся файл с расширением `.exe` (для Windows), либо `.out` (для Linux);

- **отладчик** (*debugger*) — служит для тестирования программы и выявления в ней ошибок. Отладчик позволяет выполнять трассировку (пошаговое выполнение), отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и так далее.

Исходные тексты, относящиеся к одной единице компоновки (исполняемый модуль, библиотека), составляют проект (*project*) Code::Blocks. По каждому проекту могут быть заданы параметры трансляции компоновки компилятором GCC для отладочной (*Debug*) и рабочей (*Release*) конфигураций.

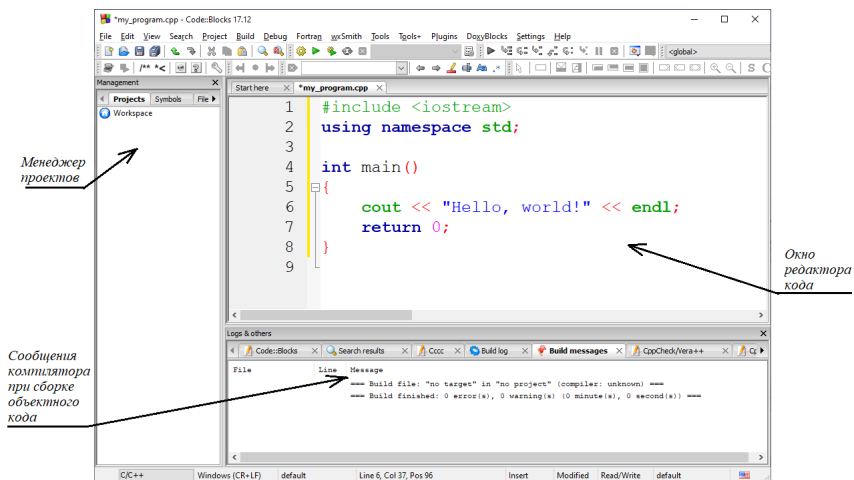


Рисунок 31. Интерфейс и некоторые компоненты среды разработки Code::Blocks 13.12

В случае написания небольших программ достаточно создавать не проект, а файл с расширением `.cpp`, который будет содержать текст программы на языке C++.

Последовательность шагов при этом будет следующей:

- 1) Выполнить команду `File → New → Emptyfile` для создания пустого файла (рис. 32)

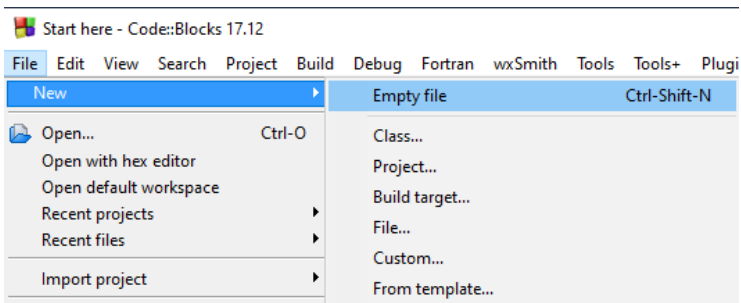


Рисунок 32. Выбор действия через главное меню программы Code::Blocks для создания пустого файла

- 2) Сохранить файл с расширением `.cpp` в выбранный каталог командой `File → Save file`. Обратите внимание на то, что задать расширение придётся полностью самостоятельно (рис. 33).

Полное имя
файла с
расширением
cpp

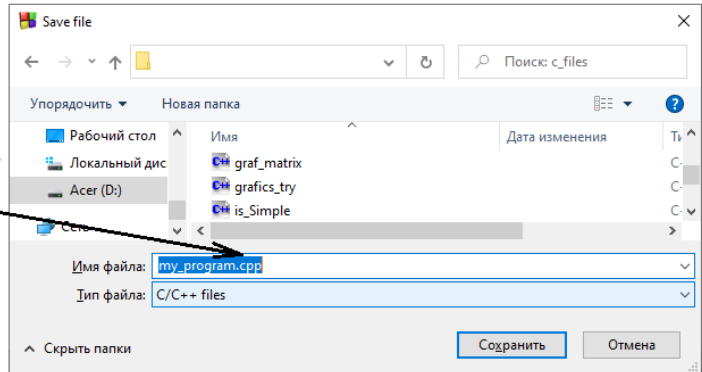


Рисунок 33. Сохранение файла

Далее можно приступить к написанию кода программы. Обратите внимание, что простейшая программа на языке C++ имеет некоторый шаблон. Все элементы этого шаблона обязательны для корректного запуска программы. Рассмотрим далее структуру шаблона программы на языке C++.

❖ Структура программы

В шаблон программы включены следующие строки (рис. 31):

<code>#include <iostream></code>	— директива препроцессора <code>#include</code> , позволяющая подключать заголовочные файлы библиотек. Библиотека <code>iostream</code> (<code>input/outputstream</code>) содержит функции, организующие обмен данными с базовыми средствами ввода/вывода: клавиатурой, монитором, принтером, файлом.
<code>using namespace std;</code>	— использование стандартного пространства имён.
<code>intmain()</code>	— заголовок главной функции. Главная функция программы — это блок программного кода с внешними атрибутами функции, который выполняется при запуске программы. Имя функции — <code>main</code> (главный). Список аргументов функции (в круглых скобках) пуст. Тип возвращаемого результата — <code>int</code> (<code>integer</code> , целый).




{...}

— операторные скобки, в которых заключено тело функции, то есть последовательность операторов. Операторные скобки являются неотъемлемой частью синтаксиса функции.


return 0;




— оператор возврата. Завершает выполнение функции main() с возвратом значения 0 — нуль, определяемого стандартом (для функции main) как код успешного завершения.

Для запуска программы на выполнение в меню среды программирования Code::Blocks содержится пункт Build (рис. 34). Рассмотрим следующие команды меню Build:

- a.  Build — сборка проекта.
- b.  Run — выполнение исполняемого файла (объектного кода) последней версии компиляции. Если в программном коде были исправлены ошибки, но проект не «пересобрали», то он выполнится в прежней версии, с ошибками.
- c.  Buildandrun — одновременная сборка и выполнение программного кода.

Рекомендации:

- 1) используйте команду  Buildandrun, если текст программы был отредактирован.
- 2) запомните **горячие клавиши**, позволяющие выполнять рассмотренные выше команды с клавиатуры:

 Build	Ctrl+F9
 Run	Ctrl+F10
 Buildandrun	F9

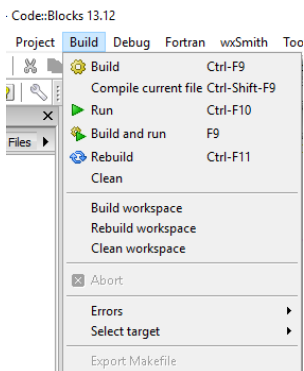


Рисунок 34. Меню Build для сборки проекта

Результатом выполнения будет запуск консольного окна (рис. 35) и вывод строки `Hello world!` (если в теле главной функции было обращение к консоли вывода — `cout`, с передачей в поток вывода текстовой строки, заключённой в кавычки, как показано на рис. 31). Также в консольном окне можно увидеть результат, с которым функция `main()` завершила свою работу: `Process returned 0 (0x0) execution time : 1.188 s` (процесс вернул нуль, то есть главная функция выполнялась без ошибок).

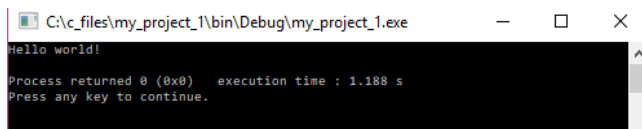


Рисунок 35. Консольное окно (результат работы программы)

❖ Организация ввода/вывода

- **Консоль вывода** (экран) `cout` — это объект, поток вывода, который расшифровывается как «*console output*», то есть «вывод на экран». Если мы хотим показать на экране какой-нибудь текст, то должны поместить его в этот поток. Операция `<<` как раз и означает «поместить». Она похожа на стрелку, это мнемоническое правило легко запомнить. Наконец, следом идёт текст в кавычках — тот, который выводится на экран. Не забывайте ставить точку с запятой там, где она есть в примерах кода.

- **Консоль ввода** (клавиатура). Значения переменных можно считывать с клавиатуры. Для этого по аналогии с потоком вывода `cout` используется поток `cin` (*console input*). Это тот механизм, по которому программа получает данные от пользователя. Как и в случае `cout`, вводимые значения разделяются оператором `>>` (стрелочками), только теперь они направлены вправо, так как мы забираем данные из потока `cin`. Приведём пример программы, которая считывает два числа *a* и *b* и выводит их сумму (рис. 36).


```

#include <iostream>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b;
    cout << a + b;
    return 0;
}

```

Рисунок 36. Пример работы с потоками ввода (cin) и вывода (cout)

Обратите внимание, что при чтении из cin, если мы хотим считать несколько переменных, то при перечислении их следует разделять оператором >> (стрелочками).

Есть и другой вариант чтения нескольких переменных. Строку

```
cin>>a>>b;
```

можно заменить строками

```
cin>>a;
```

```
cin>>b;
```

которые будут делать то же самое.

❖ Простейшие вычисления. Целочисленная арифметика

Для решения задач, требующих выполнения операций над целыми числами, в языке C++ есть базовый тип int (не единственный, но в данном методическом пособии разбираются задачи только с целыми числами).

Таблица 2. Перечень арифметических операций и их обозначений

Знак операции	Значение
+	сложение
-	вычитание (либо унарная операция изменения знака)
/	деление (для целых (int) операндов выполняется с отбрасыванием остатка)
*	умножение
%	остаток от деления целочисленных операндов со знаком первого операнда (деление по модулю)

Операндами традиционных арифметических операций (+, -, *, /) могут быть константы, переменные, идентификаторы функций, элементы массивов, указатели, любые арифметические выражения.

Порядок выполнения операций (приоритет):

1. Выражения в круглых скобках.

2. Функции (стандартные математические, функции пользователя).
3. Умножение и деление (*, /, %).
4. Сложение и вычитание (+, -).

Порядок выполнения операций можно определять круглыми скобками, тогда выражение в скобках выполняется в первую очередь (слева направо).

Унарные операции + и - обладают самым высоким приоритетом, определены только для целых и вещественных операндов, «+» носит только информационный характер, «-» меняет знак значения операнда на противоположный (не адресная операция).

Таким образом, так как операции *, /, % обладают высшим приоритетом над операциями +, -, то при записи сложных выражений нужно использовать общепринятые математические правила. Например, запись на языке программирования $x + y * z - a/b$ соответствует математической записи $x + (y * z) - (a/b)$.

Особого внимания заслуживают операции деления — в целочисленной арифметике их две, как и результатов целочисленного деления: неполное целое и остаток от деления. Так, если значения переменных a и b равны соответственно 16 и 3, то переменная $q = a / b$ получит значение, равное 5, а значение переменной $r = a \% b$ будет равно 1, так как $16 = 3 \cdot 5 + 1$.

❖ Логический тип

В языке C++ для логических значений существует специальный тип — *bool*. Допустимыми значениями этого типа являются только *true* и *false*, при этом других значений у переменной данного типа быть не может. Переменная типа *bool* занимает в памяти ровно 1 байт.

Совместимость с типом *int*: тип *bool* совместим с типом *int* по присваиванию в обе стороны. При этом *true* переходит в 1, *false* — в 0. При обратном присваивании любое число, не равное нулю, переходит в *true*, 0 — в *false*. Если использовать *bool* в арифметическом выражении, то оно будет переведено в *int*: $bool + bool = int$. Надо понимать, что в C++ логический и целочисленный тип — это разные типы.

Логические выражения

Составить выражение, имеющее логическое значение, можно с использованием операторов сравнения и логических операторов.

1) Операторы сравнения

Язык C++ имеет 6 различных операторов сравнения в своём арсенале.

- | | |
|--------|---|
| A < B | – сравнивает две переменные и возвращает <i>true</i> , если A строго меньше B |
| A > B | – возвращает <i>true</i> , если A строго больше B |
| A == B | – сравнивает на равенство переменные A и B |

- A!=B – проверяет переменные A и B на неравенство
- A>=B – нестрогое неравенство. Возвращает true, если A больше или равно B
- A<=B – нестрогое неравенство. Возвращает true, если A меньше или равно B

Примеры:

- bool r; – создали переменные, с которыми будем
- int a = 5, b = 7; работать
- r=a>b; – r содержит false, поскольку 5 < 7
- r=a<=b – r содержит true
- r=a<=5 – r равен true
- r = b == 9 – r содержит false, поскольку 7 не равно 9

2) Логические операторы

Для комбинации сразу нескольких логических выражений мы должны использовать один или несколько логических операторов (записаны в порядке убывания приоритета):

- !A – эквивалент «НЕ». Данный оператор инвертирует значение A. То есть, если A == true, то он вернет false и наоборот
- AandB – эквивалент логической операции «И». Соответственно возвращает true, если A и B являются истиной одновременно
- AorB – эквивалент логического «ИЛИ». Вернет true, если хотя бы одно из выражений является истинным

Примеры:

- bool t, r; – создали переменные с которыми будем работать
- t=3>=3and5!=4; – t содержит true, так как первое сравнение 3>=3 — истина. И второе сравнение 5! = 4 — истина
- r=trueorfalse; – r содержит true, так как один из операндов true

Следует помнить, что логический оператор НЕ имеет очень высокий уровень приоритета. Поэтому, например, выражение !x == y обрабатывается как (!x) == y.

❖ Ветвление в программе

Иногда нам нужно выполнить различные действия в зависимости от условий, либо в случае невыполнения условия вообще ничего не делать.

Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных вариантов вычислительного процесса. Каждый подобный путь называется ветвью алгоритма. Признаком разветвляющегося алгоритма является наличие операций проверки условия. Для проверки условия в языке C++ используется условный оператор *if*.

В языке C/C++ имеют отражение обе формы ветвления: полное и неполное (рис. 37-38). Соответственно получаем два вида условного оператора.



Рисунок 37. Блок-схема неполного ветвления



Рисунок 38. Блок-схема полного ветвления

Синтаксис неполного оператора условного выполнения:

```
If(условие){
    Блок операций;
}
```

Здесь условием, как правило, является логическое выражение или выражение отношения. Если выражение в скобках истинно (*true*, или не ноль), то выполняется блок операторов, заключённых в операторные скобки, иначе он игнорируется.

Примеры записи:

```
if (x>0) s+=x;    – суммирование только положительных значений x
if (i!=1) {      – используем операторные скобки {}, чтобы при
    j++;           условии, что i не равно 1, выполнились два
    s=1;          действия: j увеличилось на 1 и s стало равным 1
}
```

Предпочтительным считается вариант использования операторных скобок.

Синтаксис полного оператора условного выполнения:

```
If(условие){
    Блок операций1;
}
else{
    Блок операций2;
}
```

Если выражение в скобках истинно (*true*, или не ноль), то выполняется *Блок операций 1*, иначе — *Блок операций 2*.

Задача:

Дано целое число A . Определить, каким является число — чётным или нечётным.

Решение:

Число является чётным, если остаток при делении его на 2 равен 0 (нулю). Так как в задаче требуется выполнить *одно из двух* возможных действий (либо вывести сообщение о том, что число чётное, либо вывести сообщение, что число нечётное), будем использовать для её решения полный условный оператор (*if-else*).

Условие выбора действия зависит от остатка деления на 2 для исходного числа, который будем сравнивать с нулем: сравнение $a \% 2 == 0$ принимает истинное значение (*true*), когда a чётное (рис. 39).

```
#include <iostream>
using namespace std;
int main()
{
    int a;
    cin >> a;
    if (a % 2 == 0) cout << "The number is even";
        else cout << "The number is odd";
    return 0;
}
```

Рисунок 39. Листинг программы для определения чётности введенного числа

❖ Циклы (повторение действий в программе)

В языке программирования C++ существует три вида цикла: с параметром, с предусловием, с постусловием.

1) Общий вид оператора цикла с предусловием:

while (<выражение>) <тело_цикла>;

Если выражение в скобках — истина (TRUE, не равно 0), то выполняется тело_цикла. Это повторяется до тех пор, пока выражение не примет значение ложь (FALSE, или 0). В этом случае выполняется оператор, следующий за while. Если выражение в скобках ложно (равно 0), то цикл не выполнится ни разу.

Тело_цикла может быть простым или составным оператором. Под составным оператором понимается последовательность операторов, заключённых в операторные скобки { }.

Примечание: среди этих операторов могут быть операторы continue — переход к следующей итерации цикла и break — выход из цикла.

2) Общий вид записи:

do<тело_цикла>**while** (<выражение>;

Тело_цикла будет выполняться до тех пор, пока выражение истинно. Всё, что говорилось выше справедливо и здесь, за исключением того, что данный цикл всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять ещё раз.

3) Оператор **цикла с параметром** имеет общий вид:

```
for (<выражение1>; <выражение2>; <выражение3>) {<тело_цикла>}
```

Цикл *for* эквивалентен последовательности инструкций:

```
<выражение1>;  
while (<выражение2>)  
{  
    <тело_цикла>  
    <выражение3>;  
}
```

где выражение 1 — инициализация счётчика (начальное значение), выражение 2 — условие продолжения счета, выражение 3 — увеличение счётчика. Выражения 1, 2 и 3 могут отсутствовать (пустые выражения), но символы «;» опускать нельзя.

Например, для суммирования первых *N* натуральных чисел можно записать:

```
sum = 0;  
for (i=1; i<=N; i++) sum+=i;
```

Операция «запятая» чаще всего используется в операторе *for*. Она позволяет включать в его спецификацию несколько инициализирующих выражений. Предыдущий пример можно записать в виде:

```
for ( sum=0, i=1; i<=N; sum+= i, i++ );
```

❖ Функции

Выделение подпрограмм при решении задач позволяет не только улучшить «читабельность» программы и сократить количество строк кода, но и сформировать пользовательские библиотеки подпрограмм для многократного их использования в различных программах.

Подпрограмму можно определить как именованную часть компьютерной программы, содержащую группу операторов (набор действий) для реализации решения конкретной задачи.

Подпрограмма является самостоятельной единицей программного кода. Многие программисты предпочитают думать о подпрограмме, как о «чёрном ящике», представленном в терминах информации, которая поступает на вход этого ящика (входные параметры функции), и значением или действием, которое он производит (действия, возвращаемый результат). В модели «чёрного ящика» абстрагируются от того, что происходит внутри «чёрного ящика».

Каждая функция в языке C/C++ должна быть определена, то есть должны быть указаны:

- 1) тип возвращаемого значения;
- 2) имя функции;
- 3) информация о формальных аргументах (параметрах);
- 4) тело функции.

Пример:

Выполнить описание функции, выполняющей перевод массы, выраженной в килограммах и граммах, в граммы.

```
int to_gramm(int kg, int g)
{
    int gramm;
    gramm=kg*1000+g;
    return gramm;
}
```

Рисунок 40. Описание функции to_gramm()

Выделим элементы синтаксиса в описании функции рассмотренного примера (рис. 40) — таблица 3:

Таблица 3. Элементы синтаксиса в описании функции to_gramm()

Типвозвращаемого значения	int
Имя функции	to_gramm()
Список формальных параметров	(int kg, int g)
Тело функции	{ int gramm; gramm=kg*1000+g; return gramm; }
Возвращаемое значение	gramm

В описании функции задается список **формальных параметров**. Каждый параметр, описанный в этом списке, является локальным по отношению к описываемой функции, то есть на него можно сослаться по его имени из данной функции, но не из основной программы.

После описания функции её, как лексическую единицу, можно использовать в выражениях наряду со стандартными функциями. Функция реализует алгоритм (активируется) только при **вызове** её из тела основной программы (*main*) или из другой уже вызванной функции.

При вызове функции указывается имя функции и конкретные параметры (их называют *фактические параметры*), необходимые для вычисления функции, например, `to_gramm(2, 537)`.

При оформлении текста программы рекомендуется размещать описание всех вспомогательных функций после описания главной функции (*main*). Но учитывая принцип работы компилятора, согласно которому текст программы анализируется при компиляции сверху вниз, возникает ошибка компиляции при попытке вызова вспомогательной функции из тела главной функции, если описание функции располагается ниже по тексту программы.

Выходом из данной ситуации является использование **прототипа** для предварительного включения в текст программы сигнатуры функции — описания её заголовка.

Компилятор использует прототип функции для проверки того, что вызов функции имеет корректный тип возвращаемых данных, корректное число аргументов, корректный тип аргументов и правильный порядок следования аргументов.

Таким образом, с функцией в языке C/C++ связано три понятия:

- прототип — указывается вместо описания функции до места вызова. Прототип фактически повторяет заголовок функции. При этом список параметров допускает упрощения: достаточно указать типы параметров в соответствии с их количеством, не указывая имена;

- описание — состоит из заголовка и тела;

- вызов — оформляется в виде самостоятельного оператора (если у функции нет возвращаемого значения) или в виде компонента выражения (если функция возвращает значение).

Примечание: в языке C/C++ все функции по отношению друг к другу являются внешними. Для того, чтобы одна функция была доступна внутри другой, необходимо, чтобы до её вызова компилятору было известно имя такой функции. Поэтому при разработке программы необходимо учитывать порядок включения прототипов и описаний функций.

Рассмотренные в данной главе общие сведения о синтаксических особенностях языка C/C++ не раскрывают в полной мере всю мощь данного языка программирования, а дают лишь возможность сориентироваться в прочтении алгоритмов других глав.

Заключение

В методическом пособии приведены теоретические основы и примеры решения задач, с которых целесообразно начинать знакомство с олимпиадным программированием.

Для каждой задачи в пособии излагается идея решения в объёме, достаточном для её выполнения самостоятельно на любом языке программирования, при этом автор сопровождает каждую задачу листингом на языке C++. Самостоятельное решение задач, приведённых в конце каждой главы, позволит обучающимся закрепить теоретические знания и приобрести навыки решения практических задач изучаемой темы. Пособие позволяет обучающимся овладеть стартовым уровнем и выбрать дальнейший вектор изучения структур данных и алгоритмов их обработки.

Методическое пособие рекомендуется начинающим участникам олимпиад по информатике и программированию, а также может быть использовано студентами-бакалаврами педагогических направлений подготовки, изучающими дисциплины, связанные с методикой обучения информатике. Учебно-методическое пособие также может быть использовано учителями общеобразовательных организаций, педагогами дополнительного образования в рамках начальной подготовки обучающихся к решению задач олимпиадной информатики.

СПИСОК ИСТОЧНИКОВ

1. Алгоритмы компьютерной арифметики : учебное пособие / С.М. Окулов, А.В. Лялин, О.А. Пестов, Е.В. Разова. — 3-е изд., электрон. — Москва : Лаборатория знаний, 2020. — 288 с. — (Развитие интеллекта школьников). — ISBN 978-5-00101-657-1. — Текст : электронный. — URL: <https://znanium.com/catalog/product/1094343> (дата обращения: 01.10.2021). — Режим доступа: по подписке.
2. Антти Лааксонен Олимпиадное программирование. / пер. с англ. А.А. Слинкин. — М.: ДМК Пресс. — 2018. — 300 с.: ил.
3. Архив задач // «Школа программиста» — сайт Красноярского краевого Дворца пионеров. — URL: <https://acmp.ru/index.asp?main=tasks>
4. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы. : Пер. с англ. : М. : Издательский дом «Вильямс», 2001. — 384 с. : ил.
5. Беликов Д.А., Каминская Е.В. Информатика. Системы счисления: учебно-методический комплекс. [Электронный ресурс]. — Томск. — 2007. URL: <https://ido.tsu.ru/schools/physmat/data/res/informatika2/>
6. Визуализация алгоритма решета Эратосфена // Свободная энциклопедия «Википедия». — URL: https://upload.wikimedia.org/wikipedia/commons/thumb/8/8c/New_Animation_Sieve_of_Eratosthenes.gif/400px-New_Animation_Sieve_of_Eratosthenes.gif
7. Гашков С.Б. Системы счисления и их применение // Серия: Библиотека «Математическое просвещение»). — М.: МЦНМО. — 2004. — 52 с.
8. Динамическое программирование / Школа программиста [Электронный ресурс]. — URL: https://acmp.ru/article.asp?id_sec=1&id_text=1331
9. Динамическое программирование / Дистанционная подготовка по информатике Московского центра непрерывного математического образования [Электронный ресурс]. — URL: <https://informatics.mccme.ru/course/view.php?id=9>
10. Динамическое программирование для начинающих / Troger – издание о разработке и обо всём, что с ней связано [Электронный ресурс]. — URL: <https://tproger.ru/articles/dynprog-starters/>
11. Ишмухаметов Ш.Т. Методы факторизации натуральных чисел: учебное пособие / Ш.Т. Ишмухаметов. — Казань: Казан. ун. 2011. — 190 с.
12. Лапшева Л.Е., Пономаренко В.И. Системы счисления для профильной информатики // [Электронный ресурс]. URL: <http://window.edu.ru/catalog/pdf2txt/842/54842/26646>

- 13.Меньшиков Ф. В. Олимпиадные задачи по программированию (+CD). — СПб.: Питер. — 2006. — 315 с: ил.
- 14.Наибольший общий делитель // Математические этюды. — URL:<https://www.etudes.ru/ru/etudes/greatest-common-divisor/>
- 15.Окулов С. М. Дискретная математика. Теория и практика решения задач по информатике : учебное пособие / С. М. Окулов. — М.: БИНОМ. Лаборатория знаний, 2008. — 422 с. : ил. — (Педагогическое образование).
- 16.Решето Эратосфена // сайт Иванова Максима. — URL:https://e-maxx.ru/algo/eratosthenes_sieve
- 17.Скиена С. Алгоритмы. Руководство по разработке. — 2-е изд.: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 720 с.: ил.
- 18.Фомин С.В. Системы счисления // Серия «Популярные лекции по математике». — М.: Наука. — Вып. 40. — 1987. — С.48. URL:<https://math.ru/lib/plm/40>
- 19.Шень А. Программирование: теоремы и задачи. — 6-е изд., дополненное. — М.: МЦНМО, 2017. — 320 с.: ил

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

Подготовка школьников к олимпиадам по информатике:
стартовый уровень.
Учебно-методическое пособие

Краевое государственное автономное образовательное учреждение
дополнительного образования «Центр развития творчества детей
(Региональный модельный центр дополнительного образования детей
Хабаровского края)»

680000, г. Хабаровск, ул. Комсомольская, 87
тел. / факс: (4212) 30-57-13
Инстаграм: @dop.obrazovanie27
e-mail: yung_khb@mail.ru
<http://www.kcdod.khb.ru>

Подписано в печать: 13.12.2021
Тираж: 30 экз.

Методические материалы размещены на сайте КГАОУ ДО РМЦ



физкультурно-спортивная



туристско-краеведческая



художественно-эстетическая



естественнонаучная



техническая



социально-гуманитарная