Урок 1

Простые алгоритмы

Программирование — увлекательное занятие. И дело не только в том, что оно оправдывает себя с экономической и научной точек зрения; оно может вызвать также эстетические переживания, подобные тем, которые испытывают творческие личности при написании музыки или стихов.

«Искусство программирования. Том 1», Дональд Кнут

Почему C

Всем, что делает компьютер, любыми устройствами, которые находятся внутри него или подключены к нему, можно управлять, кодируя программы на языке программирования, — именно они будут тянуть за нужные рычаги и нажимать нужные кнопки. Язык программирования C — наилучшее (и самое распространенное) средство программирования любого персонального компьютера. Может быть, C и не самый простой для понимания, но уж точно не самый сложный. Он потрясающе популярен и хорошо поддерживается на всех платформах, именно поэтому лучше всего изучить именно его.

Введение в C

Расстояние между точками

Написать программу, которая подсчитывает расстояние между точками с координатами x1, y1 и x2, y2 по формуле sqrt(pow(x1 − x2, 2) + pow(y1 − y2, 2)).

```
#include <stdio.h>
#include <math.h>
int main(int argc, const char *argv[])
{
    double x1;
    double y1;
    double x2;
    double y2;

    printf("Input x1:");
    scanf("%lf", &x1);
    printf("Input y1:");
    scanf("%lf", &y1);
    printf("Input x2:");
    scanf("%lf", &x2);
    printf("Input y2:");
    scanf("%lf", &y2);

    double dist = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
    printf("Distance: %lf", dist);

    return 0;
}
```

Использование функций. Периметр треугольника

Найти периметр треугольника, заданного координатами своих вершин (определить функцию для расчёта длины отрезка по координатам его вершин).

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

double distance(double x1, double y1, double x2, double y2)
{
    return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}

int main(int argc, const char *argv[])
{
    double x1;
    double y1;
    double x2;
    double y2;
    double x3;
    double y3;

    printf("Input x1:");
    scanf("%lf", &x1);
    printf("Input y1:");
    scanf("%lf", &y1);
    printf("Input x2:");
    scanf("%lf", &x2);
    printf("Input y2:");
    scanf("%lf", &y2);
    printf("Input x3:");
    scanf("%lf", &x3);
    printf("Input y3:");
    scanf("%lf", &y3);

    double len1 = distance(x1, y1, x2, y2);
    double len2 = distance(x1, y1, x3, y3);
    double len3 = distance(x2, y2, x3, y3);

    printf("Perimetr: %lf", len1 + len2 + len3);

    return 0;
}
```

Здесь для решения задачи мы написали функцию, в которую передаём 4 числа − координаты двух точек. Теперь давайте посмотрим, как можно улучшить решение задачи.

Структуры

Ещё раз вернёмся к задаче нахождения расстояния между двумя точками. В языках программирования давно существует инструмент, который позволяет объединять логически связанные данные в единое целое.

Рассмотрим решение задачи на нахождение расстояния между точками с использованием такого инструмента:

```
#include <stdio.h>
#include <math.h>
struct Point
{
    double x;
    double y;
};

double distance(struct Point A, struct Point B)
{
    return sqrt(pow(A.x - B.x, 2) + pow(A.y - B.y, 2));
}

int main(int argc, const char *argv[])
{
    struct Point p1;
    struct Point p2;

    printf("Input x1:");
    scanf("%lf", &p1.x);
    printf("Input y1:");
    scanf("%lf", &p1.y);
    printf("Input x2:");
    scanf("%lf", &p2.x);
    printf("Input y2:");
    scanf("%lf", &p2.y);

    printf("Distance: %lf", distance(p1, p2));

    return 0;
}
```

Здесь мы вводим новый тип данных Point. Для этого мы используем структуру.

```
struct Point
{
    double x;
    double y;
};
```

Программист уже оперирует не отдельными переменными x1, y1, x2, y2, а более общим понятием — точками. Введение нового типа Point придаёт программе больше осмысленности и в дальнейшем упрощает программирование.

В общем виде структуры описываются следующим образом:

```
struct имя_структуры
{
   поля_структуры
};
```

Если поля структуры имеют различные типы, то каждый тип описывается в отдельной строке.

Для использования нового типа в программе нужно перед названием структуры написать ключевое слово struct, далее – название структуры и имя переменной, которая будет выступать в качестве структуры.

```
struct Point p1;
```

Для доступа к полю структуры нужно написать её имя, поставить точку и написать имя поля:

```
p1.x
A.x
```

Ещё одно улучшение, которое позволяет нам сделать язык C, — избавиться от написания длинного struct Point, каждый раз, когда нам нужно описать переменную типа структуры.

Для этого есть возможность дать нашему типу другое имя, например, так:

```
typedef struct Point Point;
```

Теперь вместо

```
struct Point p1;
```

можно писать:

```
Point p1;
```

Конечно, мы можем создавать и более сложные структуры. Например, структура, описывающая человека:

```
struct Human
{
   char fName[50];    // Имя
   char lName[50];    // Фамилия
   int age;           // Возраст
   double weight;     // Вес
};
```

Функции позволяют логически разбить программу на подпрограммы. Выполнение домашней работы может выглядеть так:

```
#include <stdio.h>
void solution1();
void solution2();
void solution3();
void menu();

int main()
{
    int sel = 0;
    do
    {
        menu();
        scanf("%i", &sel);
        switch (sel)
        {
            case 1:
                solution1();
                break;
            case 2:
                solution2();
                break;
            case 3:
                solution3();
                break;
            case 0:
                printf("Bye-bye");
                break;
            default:
                printf("Wrong selected\n");
        }
    } while (sel != 0);
    return 0;
}
void solution1()
{
    printf("Solution 1\n");
    // Решение
}

void solution2()
{
    printf("Solution 2\n");
    // Решение
}

void solution3()
{
    printf("Solution 3\n");
    // Решение
}

void menu()
{
  printf("1 - task1\n");
  printf("2 - task2\n");
```

```
  printf("3 - task3\n");
  printf("0 - exit\n");
}
```

Функции позволяют программисту разбить большую программу на подпрограммы и концентрироваться на решении небольшой задачи. Решив одну задачу и убедившись, что это сделано верно, можно переходить к другой, также оформив её в виде функции.

В общем виде функции описываются следующим образом:

```
тип_возвращаемого_значения имя_функции(список_формальных_параметров)
{
   // Тело функции
}
```

Если функция ничего не возвращает, то тип возвращаемого значения будет void. Например:

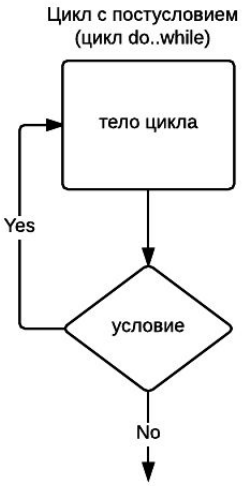
```
void myFunction()
{
   // Моя функция
}
```

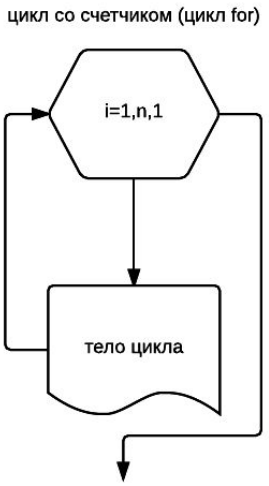
В C есть возможность описать прототип функции. Это даёт возможность вызывать функции до их описания. Например:

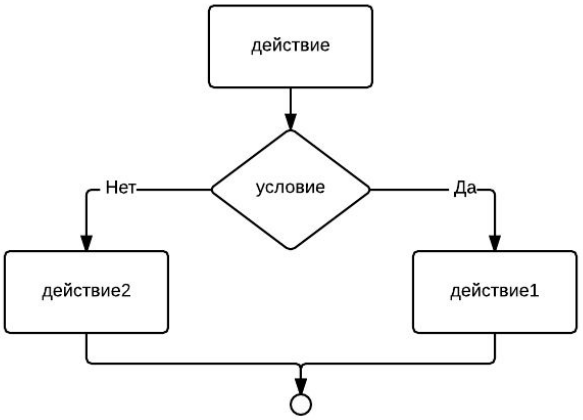
```
void myFunction1();  // Описали прототип функции

void myFunction2()
{
   myFunction1();    // Вызываем функцию до ее описания
}

void myFunction1()   // Описываем функцию
{
                     // Тело функции
}
```

Алгоритмы

Блоки и блок-схемы

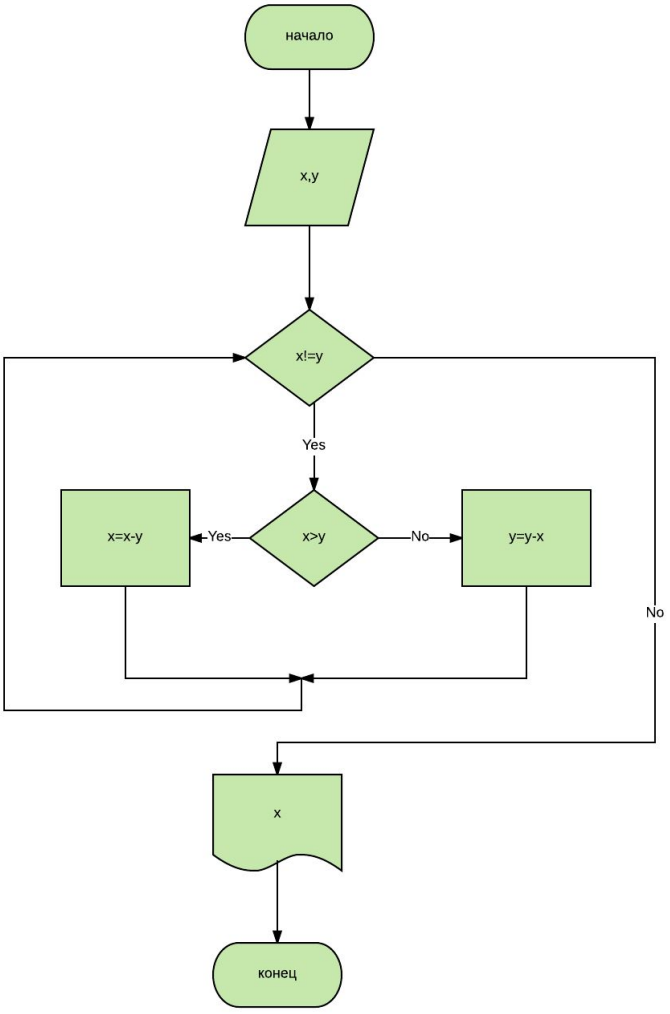






Определение наибольшего общего делителя. Алгоритм Эвклида

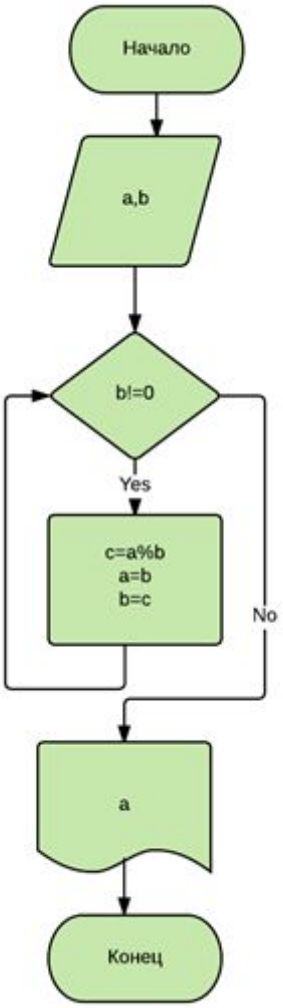
Пока a не равно b, вычитаем из большего числа меньшее.




```
#include <stdio.h>
// Определение наибольшего общего делителя. Алгоритм Эвклида

int main(int argc, const char *argv[])
{
    int a;
    int b;
    printf("%s", "Input a:");
    scanf("%d", &a);
    printf("%s", "Input b:");
    scanf("%d", &b);
    while (a != b)
        if (a > b)
            a = a - b;
        else
            b = b - a;
    printf("GCD: %i", a);
```

```
    return 0;
}
```

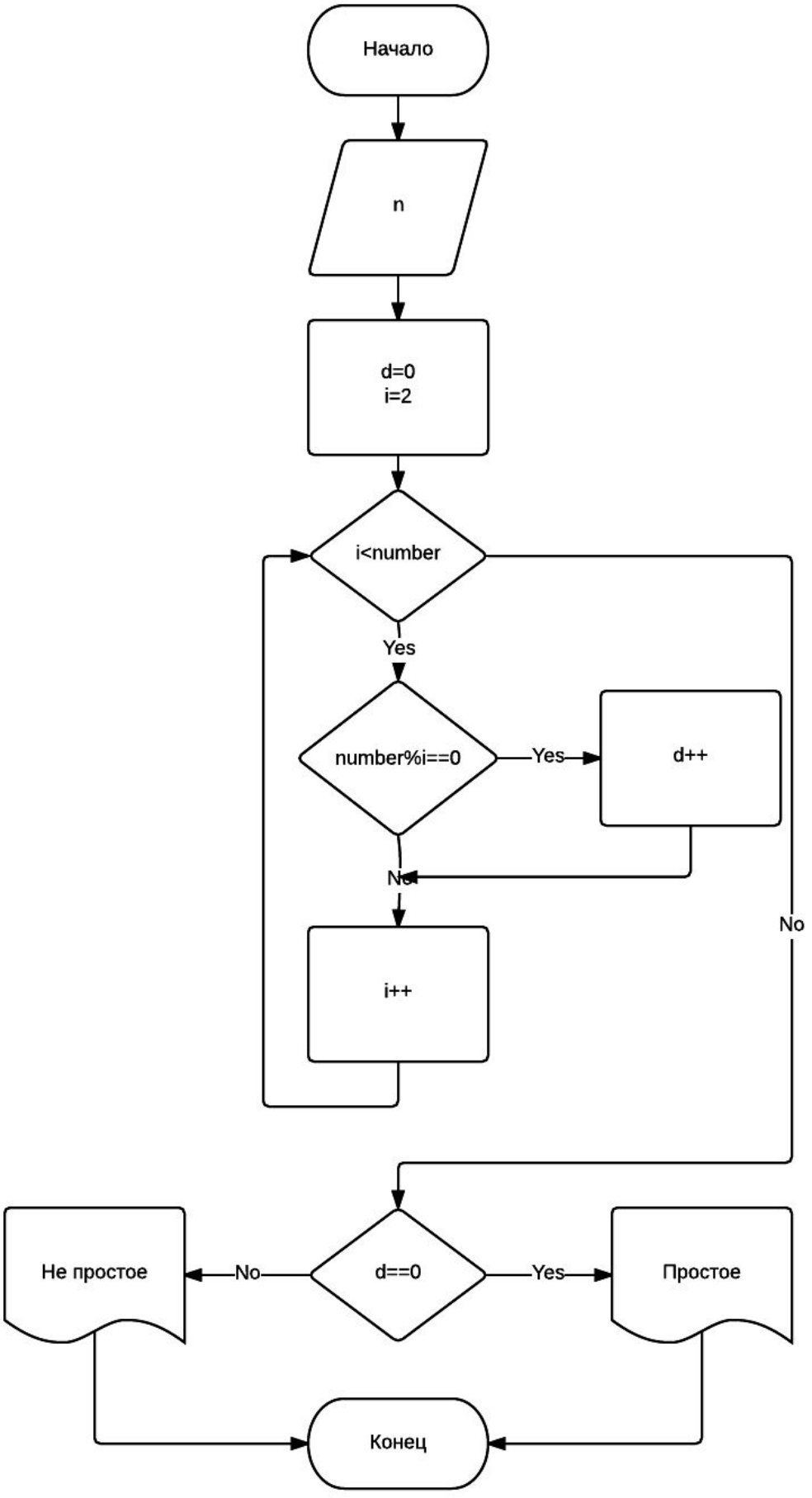
Ускоренный алгоритм Эвклида




```
#define T long

T gcd(T a, T b) {
  T c;

  while (b) {
     c = a % b;
     a = b;
     b = c;
  }

  return a;
}
```

Программа определения простоты числа

Простым называется число, у которого всего два делителя — 1 и оно само. Напишем программу, которая определяет, является ли число простым.

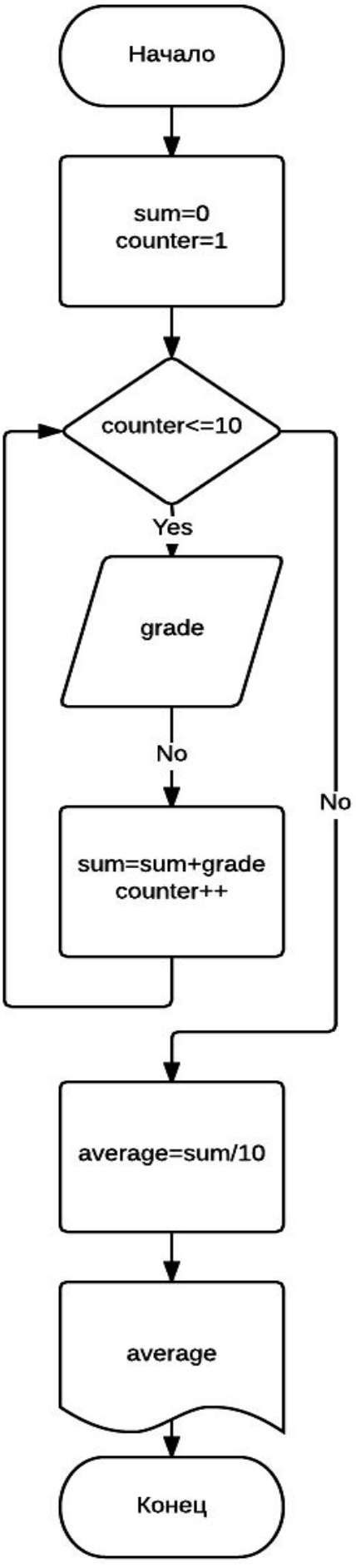

Начало
n
d=0
i=2
i<number
Yes
number%i==0
Yes
d++
No
i++
No
d==0
Yes
Простое
No
Не простое
Конец

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    int number;
    printf("Input number:");
    scanf("%d", &number);
    int d = 0;
    int i = 2;

    while (i < number)
    {
        if (number % i == 0) d++;
        i++;
    }

    if (d == 0)
        printf("Number is simple");
    else
        printf("Number is not simple");

    return 0;
}
```

В программе мы подсчитываем количество делителей числа number, начиная с цифры 2 до числа number — 1. Если делителей в этом промежутке не нашлось, то число простое. Это один из самых простых алгоритмов. Есть несколько способов сделать его более эффективным.

Определение средней оценки с помощью цикла while. Используем счётчик

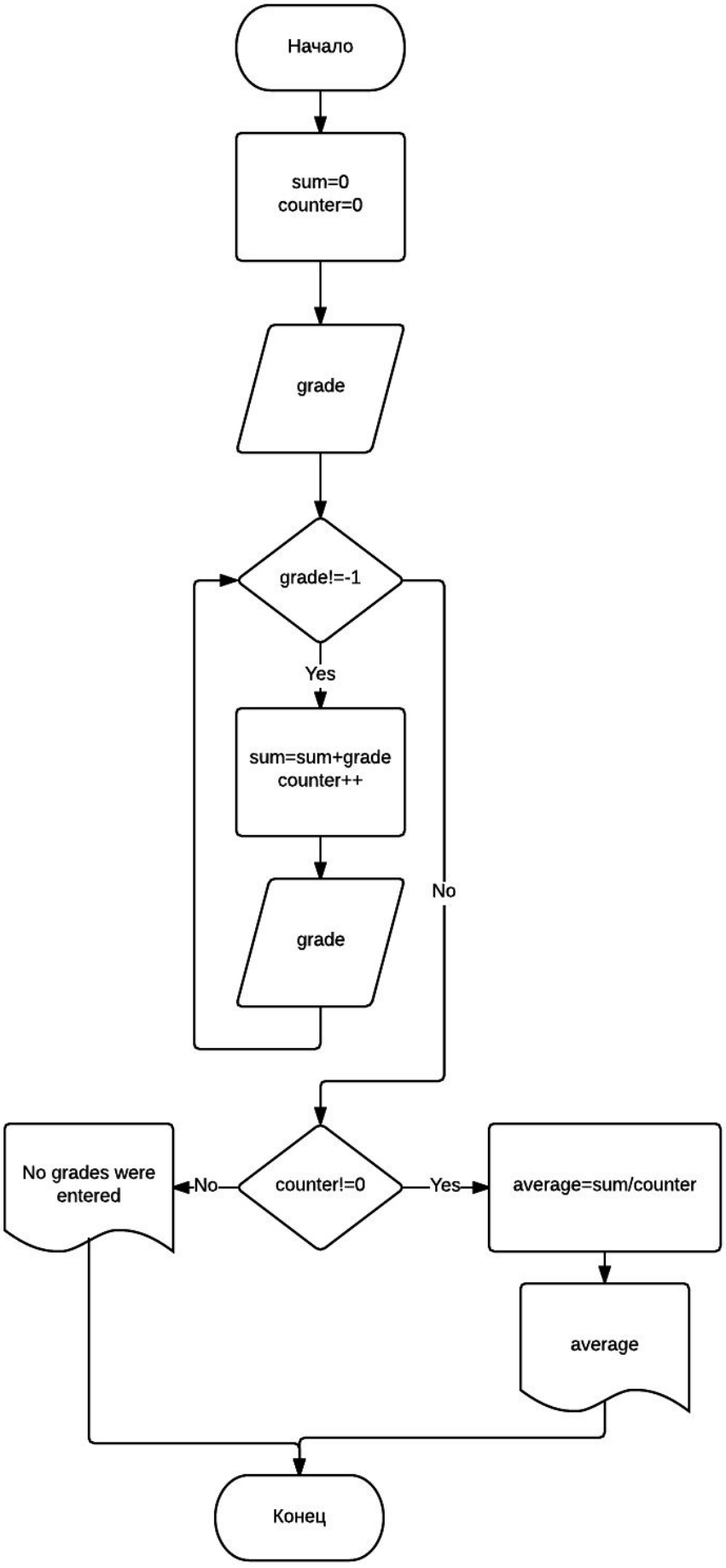
С клавиатуры вводится десять оценок учащихся. Задача — определить среднюю оценку и вывести её на экран. Для окончания ввода используем переменную счётчик.


Начало
sum=0
counter=1
counter<=10
Yes
grade
No
sum=sum+grade
counter++
No
average=sum/10
average
Конец

```c
#include <stdio.h>
// Вычисление средней оценки инструкцией повторения, управляемой счетчиком

int main(int argc, char *argv[])
{
    unsigned int counter;  // количество оценок
    int grade;             // значение оценки
    int sum;               // сумма оценок
    double average;        // средняя оценка

    sum = 0;
    counter = 1;

    while (counter <= 10)
    {
        printf("%s", "Enter grade:");
        scanf("%d", &grade);
        sum = sum + grade;
        counter++;
    }

    average = (double) sum/10;
    printf("Class average is %lf\n", average);

    return 0;
}
```

Определение средней оценки с помощью цикла while. Используем флаг

С клавиатуры вводятся оценки учащихся до тех пор, пока не будет введено число −1. Задача — определить среднюю оценку и вывести её на экран. Для окончания ввода используем переменную «флаг».

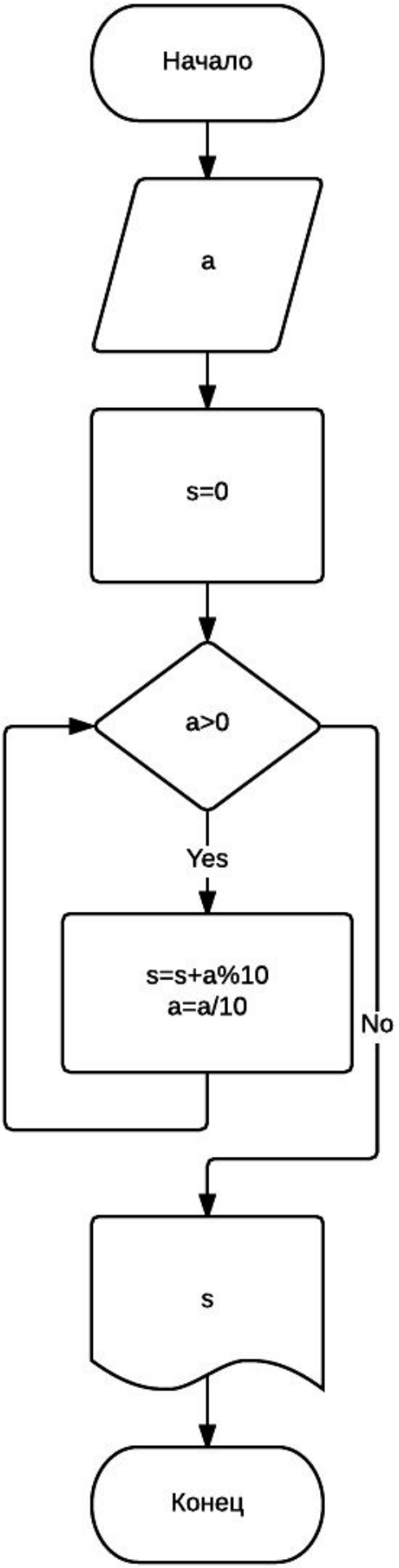

Начало
sum=0
counter=0
grade
grade!=-1
Yes
sum=sum+grade
counter++
grade
No
No grades were entered
No
counter!=0
Yes
average=sum/counter
average
Конец

```c
#include <stdio.h>
// Разработать программу вычисления средней оценки по классу для произвольного количества учащихся

int main(int argc, char *argv[])
{
    int counter = 0;   // количество оценок
    int grade = 0;     // значение оценки
    double sum = 0; // сумма оценок
    double average; //средняя оценка

    do
    {
        sum = sum + grade;
        counter++;
        printf("%s", "Enter grade (or -1 to):");
        scanf("%d", &grade);
    } while (grade != -1);

    if (counter != 0)
    {
        average = sum / counter;
        printf("Class average is %g\n", average);
    }
    else
    {
        puts("No grades were entered");
    }

    return 0;
}
```

Найти сумму цифр целого числа

Получить последнюю цифру можно, если найти остаток от деления числа на десять. В связи с этим для разложения числа на составляющие его цифры можно использовать следующий алгоритм:

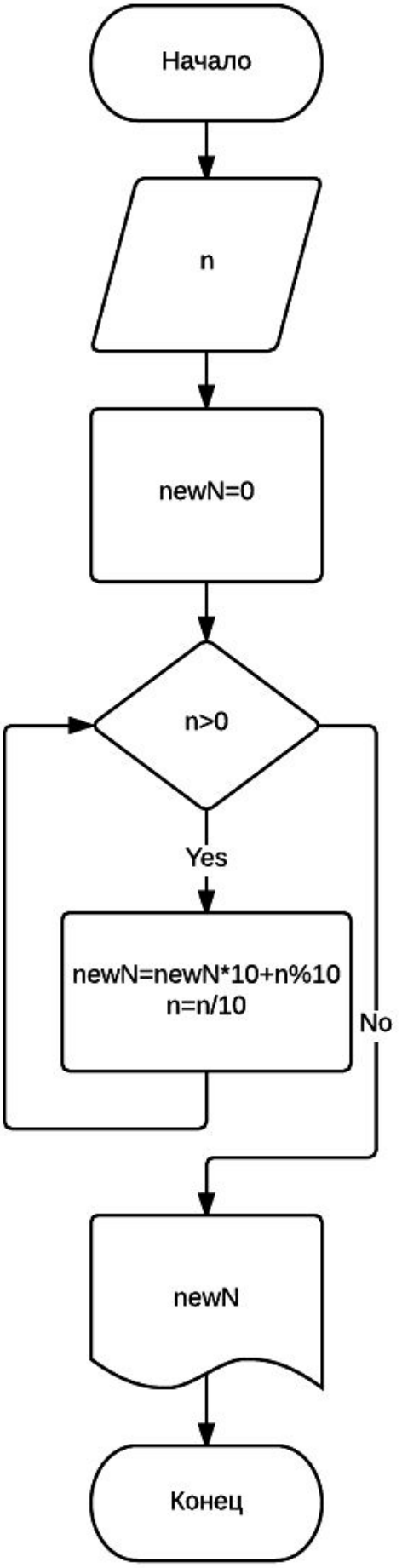
1. Находим остаток при делении числа A на десять, т.е. получаем крайнюю правую цифру числа.
2. Находим целую часть числа при делении A на десять, т.е. отбрасываем от числа A крайнюю правую цифру.
3. Если преобразованное A > 0, то переходим к пункту 1. Иначе число равно нулю, и отделять от него больше нечего.


Начало
a
s=0
a>0
Yes
s=s+a%10
a=a/10
No
s
Конец

```c
#include <stdio.h>
int sumDigit(long a)
{
    int result = 0;
    while (a > 0)
    {
        result = result + a % 10;
        a = a / 10;
    }
    return result;
}

int main(int argc, char *argv[])
{
    int n;
    printf("Input number:");
    scanf("%d", &n);
    printf("Sum digit: %d", sumDigit(n));

    return 0;
}
```

Функция переворота числа

Алгоритм переворота числа довольно простой. Достаточно отбрасывать от него последнюю цифру и прибавлять её к новому числу, только новое число нужно умножить на десять.


Начало
n
newN=0
n>0
Yes
newN=newN*10+n%10
n=n/10
No
newN
Конец

```c
#include <stdio.h>
long reverse(long n)
{
   long result = 0;
   while (n > 0)
   {
      result = result * 10 + n % 10;
      n /= 10;
   }
   return result;
}

int main(int argc, char *argv[])
{
   int n = 123;
   printf("%d %d\n", n, reverse(n));
   return 0;
}
```

Перевод из одной системы счисления в другую

Алгоритм перевода довольно прост. Мы делим число на основание системы счисления, в которую хотим его перевести, и записываем в новое число остатки от деления в обратном порядке, то есть с применением алгоритма переворота числа:

```c
   int result = 0;
   while (n > 0)
   {
      result = result * 10 + n % 2;
      n /= 2;
   }
```

Но в этом алгоритме есть ошибка: число перевёрнутое, и если остатки от деления — нули, то число будет переведено неправильно. Например:

n	result
16	0
8	0
4	0
2	0
1	**1**

Результат равен 1, хотя должно быть 10000. Для решения этой задачи можно использовать рекурсию, но мы рассмотрим рекурсивное решение позднее. Поэтому используем глобальный массив, в который запишем остатки от деления, а потом перевернём его при выводе.

```
#include <stdio.h>
#define ARR_SIZE 100

// Задаём значение первого элемента, чтобы обнулить остальные
int bin[ARR_SIZE] = {0};
int size = 0;    // количество реально задействованных элементов массива

void convertToBin(long n)
{
    int i=0;
    while (n > 0)
    {
        bin[i++] = n % 2;
        n /= 2;
    }
    size = i;
}
void binPrint()
{
    int i;
    for(i = size - 1; i >= 0; i--)
        printf("%d", bin[i]);
    printf("\n");
}
int main(int argc, char *argv[])
{
    int N = 3;
    convertToBin(N);
    binPrint();

    N = 1024;
    convertToBin(N);
    binPrint();

    return 0;
}
```

Функция возведения в степень a^b

Умножаем число a на само себя b раз, и с каждым проходом цикла уменьшаем b на единицу. Когда b будет равным нулю, условие станет ложным.

```
long power(int a, int b)
{
  long p = 1;
  while(b)
  {
     p = p * a;
     b--;
  }
}
```

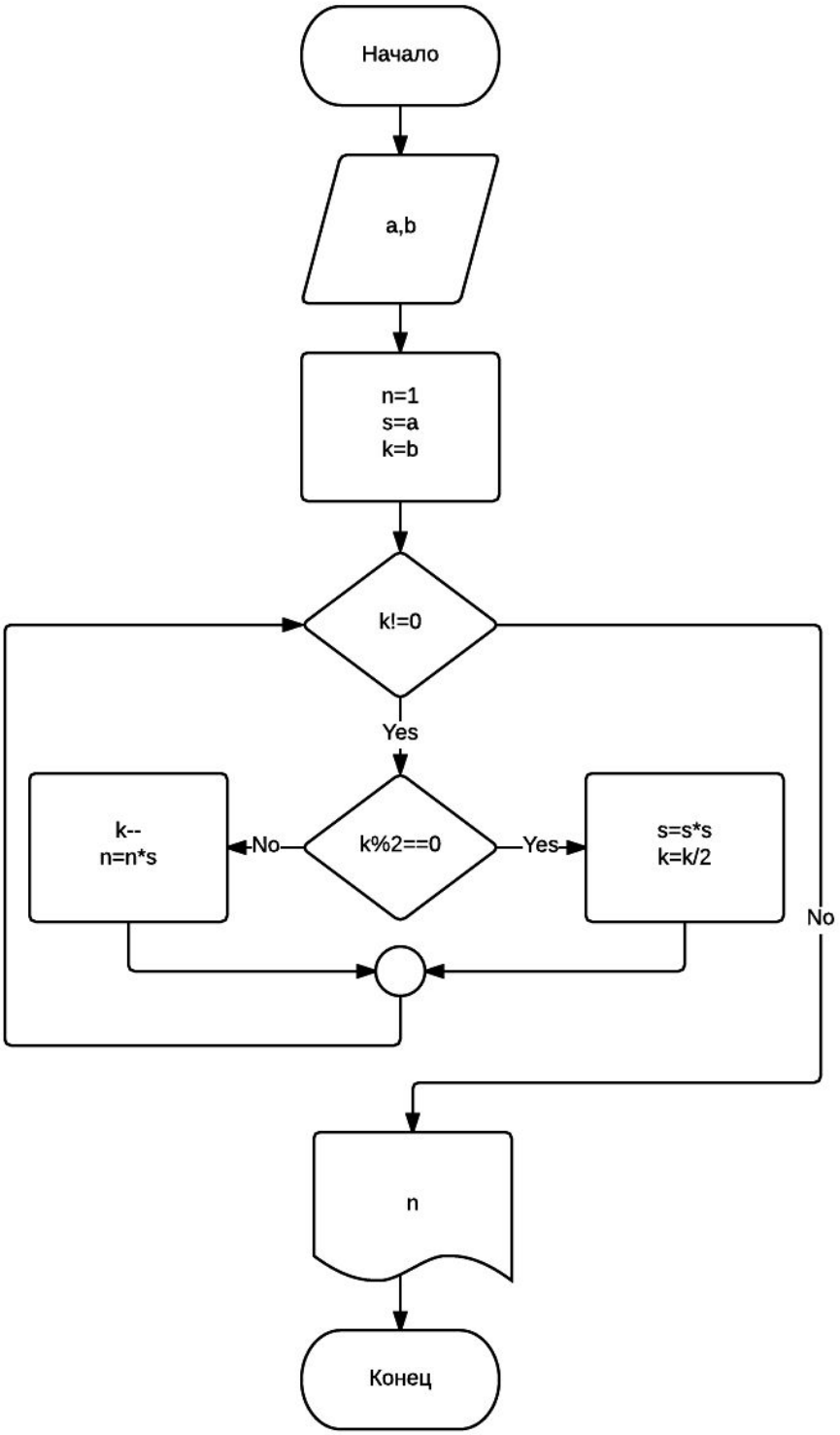
Функция ускоренного возведения в степень

Ускорим процесс возведения в степень, используя следующие формулы:

$A^{2xM} = (A^M)^2$

$A^{M+N} = A^M x A^N$

Первая формула позволяет быстро вычислить степень числа A, возводя его в квадрат, при условии, что степень четная. Вторая формула напоминает нам комбинировать различные степени.


Начало
a,b
n=1
s=a
k=b
k!=0
Yes
No
k%2==0
Yes
No
k--
n=n*s
s=s*s
k=k/2
n
Конец

```
long quickPow(int a, int b) {
	long n = 1;
	while (b) {
		if (b % 2) {
			b--;
			n *= a;
		} else {
			a *= a;
			b /= 2;
		}
	}
	return n;
}
```

Генератор псевдослучайных чисел (ГПСЧ)

Примером простого и общего метода создания псевдослучайных чисел является линейный конгруэнтный генератор, использующий следующую зависимость для формирования величин:

```
Xn + 1 = (A * Xn + B) % M
```

где A, B, и M — постоянные.

Величина X0 называется начальным числом. Она инициализирует генератор, поэтому различные величины X0 дают неодинаковые комбинации чисел.

Поскольку все значения в числовой последовательности берутся по модулю M, как только генератор достигает максимума, он производит число, полученное им ранее, а последовательность чисел повторяется с того места.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   int x, a, b, m;
   m = 100; // Вершина последовательности
   b = 3;
   a = 2;
   x = 1;
   int i;
   int modulus = 100;

   for (i = 0; i < modulus; i++)
   {
      x = (a * x + b) % m;
      printf("%d ", x);
   }

   return 0;
}
```

Некоторые алгоритмы ГПСЧ используют множественные линейные конгруэнтные генераторы с разными постоянными и производят выборку из величин, сгенерированных на каждом шаге. Это делается для того, чтобы полученные числа казались «более случайными», а период повторения последовательности увеличился. Однако и такие методы не являются истинно случайными.

Встроенный генератор случайных чисел

Большинство языков программирования обладает встроенными ГПСЧ. Эти генераторы в основном довольно быстрые и производят очень длинные последовательности чисел, прежде чем повториться. По этой причине их удобно использовать вместо того, чтобы писать собственные.

Пример использования функции rand():

```
#include <stdio.h>
#include <stdlib.h>       // Для использования генератора случайных чисел
int main(int argc, char *argv[])
{
   srand(time(NULL));   // Инициализация счетчика случайных чисел.
   printf("%d\n", rand());// возвращает псевдослучайное целое число в диапазоне int.
   printf("%d\n", rand() % 100);          //... от 0 до 99
   printf("%d\n", rand() % 100 + 1);      //... от 1 до 100
   printf("%d\n", rand() % 10 + 100);   //... от 100 до 109
}
```

Простейший подсчёт производительности программы

Производительность в чистом виде — это количество элементарных операций, которые необходимо выполнить для получения результата в зависимости от размера обрабатываемых данных. Для сортировки размер обрабатываемых данных — это длина сортируемого массива. В качестве элементарных операций разумно взять операции сравнения и присваивания. Их суммарное количество и будет характеризовать производительность.

Для простейшего подсчёта производительности можно ввести переменную, увеличивающуюся на единицу при каждой операции, которые мы хотим подсчитать. Легко определить количество операций сравнений в пузырьковой сортировке:

```
int count = 0;                  // Ввели счётчик количества операций
for(i = 0; i < N; i++)
   for(j = 0; j < N - 1; j++)
   {
      count++;
      if (a[j] > a[j + 1])
      {
         count++;
         swap(&a[j], &a[j + 1]);
      }
   }
puts("Array after sort");
Print(N, a);
printf("Count:%d", count); // Выводим счётчик на экран
```

Нахождение максимального числа. Заполнения массива из файла. Передача массива в качестве параметра функции

На примере решения задачи нахождения максимального числа рассмотрим, как считать массив из файла и передать внутрь функций.

Решим стандартную задачу. Дан файл с целыми числами. Из файла считывается массив целых чисел. Написать функции вывода массива на экран и нахождения максимального числа.

```
#include <stdio.h>
#define ARR_SIZE 100
void arrayPrint(int length, int *a)
{
    int i;
    for(i = 0; i < length; i++)
        printf("%6i", a[i]);
    printf("\n");
}
// Нахождение максимального числа в массиве
int findMax(int length, int *a)
{
// В качестве начального значения максимума берём первое число
    int result = a[0];
    int i;
// Просматриваем остальные числа
    for (i = 1; i < length; i++)
// если среди них есть число больше max, то берём его в качестве max
        if (a[i] > result) result = a[i];
// Возвращаем результат
    return result;
}
int main(int argc, char *argv[])
{
    int array[ARR_SIZE];
    int size = 0;
    FILE *in;
    in = fopen("D:\\temp\\data.txt", "r");
    if (in == NULL)
    {
        puts("Can't open file");
        return 1;
    }
    int data;
// Пока количество считанных данных больше нуля
    while(fscanf(in, "%d", &data) > 0)
    {
        array[size] = data;
        size++;
    }
    fclose(in);
    printf("Read %d records\n", size);
// Массив является указателем (хранит адрес), поэтому & не ставится
    arrayPrint(size, array);
    printf("Max = %d", findMax(size, array));
    return 0;
}
```

Функция fscanf() возвращает количество правильно считанных данных, поэтому используем возвращаемое значение в качестве условия цикла.

Нахождение простых чисел. Решето Эратосфена

Название «решето» метод получил, потому что, согласно легенде, Эратосфен писал числа на дощечке, покрытой воском, и прокалывал отверстия в тех местах, где были написаны составные числа. Поэтому дощечка являлась неким подобием решета, через которое «просеивались» все составные числа, а оставались только числа простые. Эратосфен дал таблицу простых чисел до 1000.

Алгоритм

Для нахождения всех простых чисел не больше заданного числа n, следуя методу Эратосфена, нужно выполнить следующие шаги:

1. Выписать подряд все целые числа от двух до n (2, 3, 4, …, n).
2. Присвоить переменной p значение 2 — первого простого числа.
3. Зачеркнуть в списке числа от $2p$ до n, считая шагами по p (это будут числа кратные p: $2p$, $3p$, $4p$, …).
4. Найти первое незачёркнутое число в списке, большее чем p, и присвоить значению переменной p это число.
5. Повторять шаги 3 и 4, пока возможно.

Теперь все незачёркнутые числа в списке — все простые числа от 2 до n.

На практике алгоритм можно улучшить следующим образом. На шаге 3 числа можно зачеркивать, начиная сразу с числа p^2, потому что все составные числа меньше него уже будут зачеркнуты к этому времени. И, соответственно, останавливать алгоритм можно, когда p^2 станет больше, чем n. Также все простые числа (кроме 2) — нечётные, и поэтому для них можно считать шагами по $2p$, начиная с p^2.

Домашнее задание

1. Ввести вес и рост человека. Рассчитать и вывести индекс массы тела по формуле I=m/(h*h); где m-масса тела в килограммах, h - рост в метрах.

2. Найти максимальное из четырех чисел. Массивы не использовать.

3. Написать программу обмена значениями двух целочисленных переменных:

   a. с использованием третьей переменной;

   b. *без использования третьей переменной.

4. Написать программу нахождения корней заданного квадратного уравнения.

5. С клавиатуры вводится номер месяца. Требуется определить, к какому времени года он относится.

6. Ввести возраст человека (от 1 до 150 лет) и вывести его вместе с последующим словом «год», «года» или «лет».

7. С клавиатуры вводятся числовые координаты двух полей шахматной доски (x1,y1,x2,y2). Требуется определить, относятся ли к поля к одному цвету или нет.

8. Ввести a и b и вывести квадраты и кубы чисел от a до b.

9. Даны целые положительные числа N и K. Используя только операции сложения и вычитания, найти частное от деления нацело N на K, а также остаток от этого деления.

10. Дано целое число N (> 0). С помощью операций деления нацело и взятия остатка от деления определить, имеются ли в записи числа N нечетные цифры. Если имеются, то вывести True, если нет — вывести False.

11. С клавиатуры вводятся числа, пока не будет введен 0. Подсчитать среднее арифметическое всех положительных четных чисел, оканчивающихся на 8.

12. Написать функцию нахождения максимального из трех чисел.

13. * Написать функцию, генерирующую случайное число от 1 до 100.

    с использованием стандартной функции rand()

    без использования стандартной функции rand()

14. * Автоморфные числа. Натуральное число называется автоморфным, если оно равно последним цифрам своего квадрата. Например, 25 \ :sup: `2` = 625. Напишите программу, которая вводит натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.

Записывайте в начало программы условие и свою фамилию. Все решения создавайте в одной программе. Со звёздочками выполняйте в том случае, если вы решили задачи без звёздочек.

Дополнительные материалы

1. Правила хорошего тона в программировании;
2. Индекс массы тела(BMI)

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Род Стивенс, Алгоритмы. Теория и практическое применение. Издательство «Э», 2016.
2. Никлаус Вирт, Алгоритмы и структуры данных. Новая версия для Оберона, ДМК, Москва, 2010.
3. Пол Дейтел, Харви Дейтел. C для программистов. С введением в C11, ДМК, Москва, 2014.