Управление программой

Артамонов Ю.Н.

Международный университет
природы, общества и человека "Дубна"
филиал Котельники

10 ноября 2017 г.

Содержание

Цикл for

Большинство программ содержат повторения, т.е. **циклы**. До сих пор мы изучили два вида повторений:

- повторение, управляемое счетчиком;
- повторение, управляемое контрольным значением (цикл с предусловием).

Однако для цикла, управляемого счетчиком в языке C предусмотрена специальная более удобная конструкция. Для повторения, управляемого счетчиком, требуется:

1. *Имя* управляющей переменной (счетчика).
2. *Начальное значение* управляющей переменной.
3. *Значения инкремента* (приращения со знаком плюс) или декремента (приращения со знаком минус), на которое управляющая переменная изменяет свое значение после каждой итерации цикла.
4. Условие, в котором происходит проверка на *конечное значение* управляющей переменной (т.е. должно ли продолжиться выполнение цикла).

Цикл **for** (продолжение)

Рассмотрим простую программу вывода чисел от 1 до 10. С использованием оператора while она могла бы выглядеть так:

```
#include <stdio.h>
main()
{
  int counter = 1; //объявляем имя переменной — пункт 1, устанавливаем
  начальное значение — пункт 2
  while (counter <= 10) //проверяет конечное условие — пункт 4
    {
      printf("%d\n",counter);
      counter++; //устанавливаем значение инкремента — пункт 3
    }
  return 0;
}
```

Цикл **for** (продолжение)

Еще короче она могла бы выглядеть так:

```
...
  int counter = 0;
  while (++counter <= 10) printf("%d\n", counter);
...
```

Этот код экономит один оператор, поскольку приращение выполняется непосредственно в условии структуры while перед его проверкой (3, 4 пункты объединяются). За счет этого отсутствует необходимость использования фигурных скобок, поскольку в теле цикла теперь выполняется один оператор. Написание кода в такой сжатой манере требует некоторой практики. Обратите внимание, что теперь счетчик нужно установить в 0. А в условии использовать оператор пред-инкремента (если использовать пост-инкремент, мы бы прошагали лишний раз до 11).

Цикл **for** (продолжение)

Структура оператора **for** объединяет в себе все 4 пункта. Её общий формат имеет вид:

for (Выражение1; Выражение2; Выражение3) оператор

Выражение1 объявляет и инициализирует переменную цикла.

Выражение2 является условием продолжения цикла.

Выражение3 служит для приращения управляющей переменной.

Тогда наша задача по выводу чисел от 1 до 10 может быть решена следующим образом:

```
#include <stdio.h>
main()
{
  int i;
  for (i=1; i<=10; i++) printf("%d\n",i);
  return 0;
}
```

Цикл **for** (продолжение)

Заметим, что в новых стандартах языка C (1999 и 2011 годов) допускается объединить объявление переменной с инициализацией. Но в этом случае нужно компилировать программу с дополнительным ключом -std=c99 или -std=c11.

```
#include <stdio.h>
main()
{
   for (int i=1; i<=10; i++) printf("%d\n",i);
   return 0;
}
```

Компилируем так: **cc name.c -std=c99** или **cc name.c -std=c11**.

Цикл **for** (примеры)

Следующие примеры демонстрируют способы изменения управляющей переменной в структуре for.

for (i = 1; i <= 100; i++) изменение i от 1 до 100 с приращением 1.

for (i = 100; i >= 1; i - -) изменение i от 100 до 1 с приращение -1 (декрементом 1).

for (i = 2; i <= 20; i += 3) шагаем по следующей последовательности: 2, 5, 8, 11, 14, 17, 20.

for (i = 99; i >= 0; i -= 11) шагаем по следующей последовательности: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

Цикл **for** (продолжение)

Обратите внимание, что тело структуры for можно объединять с его заголовком, используя запятую. Например, имеется код:

```
#include <stdio.h>
main()
{
   int sum = 0, number;
   for (number = 2; number <= 100; number += 2) sum += number;
   printf("%d", sum);
   return 0;
}
```

Это можно было бы выполнить так:

```
...
   int sum = 0, number;
   for (number = 2; number <= 100; sum += number, number += 2)
...
}
```

Цикл **for** - задача

Клиент открыл в банке 5-процентный сберегательный счет на 1000.00 \$. Рассчитайте и выведите сумму денег на счете на конец каждого года за 10 лет. Для этого используйте формулу сложных процентов:

$$a = p \cdot (1 + r)^t$$

p - первоначальный вклад;
r - годовая процентная ставка;
t - текущий расчетный год;
a - сумма на депозите на конец t-го года.

Цикл for (решение)

```
#include <stdio.h>
#include <math.h> //следует компилировать с ключом -lm
main()
{
  int t;
  double a, p = 1000, r = .05;
  printf("%4s%21s\n", "Год", "      Сумма на депозите");
  for (t = 1; t <= 10; t++)
    {
      a = p * pow(1 + r, t);
      printf("%4d%21.2f\n", t, a);
    }
  return 0;
}
```

Цикл **for** (анализ решения)

Структура for выполняет 10 раз цикл, изменяя при этом управляющую переменную t от 1 до 10. Функция **pow** реализует операцию возведения в степень, она содержится в специальной математической библиотеке **math**, поэтому следует ее подключить (для этого, кроме указания #include <math.h>, необходимо использовать специальный ключ при компиляции, например, **cc name.c -lm**. Функция pow должна принимать параметры типа **double** - это аналогично типу float для представления чисел с плавающей точкой, но большей размерности и с большей точностью.

Для вывода значения переменной a в программе используется спецификатор преобразования **%21.2f**. Это означает, что будет выводиться число с плавающей точкой с точностью два знака после точки. Число 21 обозначает *ширину поля*, в которое будет выводиться значение. Если количество отображаемых символов меньше ширины поля, то значение автоматически выравнивается по правому краю. Чтобы выровнять значение по левому краю, нужно использовать знак минус. Например, **%-6d**.

Структура повторения do/while

Структура повторения **do/while** подобна структуре while. Основное отличие - в структуре **do/while** условие на выполнение цикла проверяется после цикла, поэтому тело цикла будет выполнено по крайней мере один раз. Цикл **do/while** реализует так называемый *условный цикл с постусловием*. Общий вид этого оператора:

do { оператор } while (условие)

Если оператор один, то фигурные скобки можно не ставить. Однако настоятельно рекомендуется их ставить всегда в этом операторе, чтобы не спутать с оператором цикла while.

Пример использования do/while

```
#include <stdio.h>
main()
{
  int counter = 1;
  do
   {
     printf("%d", counter);
   } while (++counter <=10);
  return 0;
}
```

Задание 4.1

Написать программу, которая вычисляет произведение нечетных чисел от 1 до 15.

Задание 4.2

Написать программу вычисления факториала числа $n! = 1 \cdot 2 \cdot 3 \ldots \cdot n$.

Задание 4.3

Написать программу, которая выводит звездочками треугольник с высотой 7 звездочек:

```
*
**
***
****
*****
******
*******
******
*****
****
***
**
*
```

Задание 4.4

Написать программу, которая выводит звездочками треугольник с заданной высотой.

Задание 4.5

Вычислить значение числа π с заданной точностью 4 верных знака с использованием ряда:
$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} \cdots$
Сколько требуется взять членов этого ряда?

Множественный выбор switch

Иногда алгоритм содержит последовательность условий, когда происходит независимая проверка переменной или выражения на равенство каждому из целочисленных значений, и в зависимости от равенства одному из этих значений предпринимаются определенные действия. В языке C для обработки такого рода ситуаций предусмотрена структура множественного выбора switch.

Структура switch состоит из ряда меток **case** и необязательного блока **default**, каждая метка заканчивается оператором **break**.

Множественный выбор switch (продолжение)

Внешний вид структуры показан на рисунке.

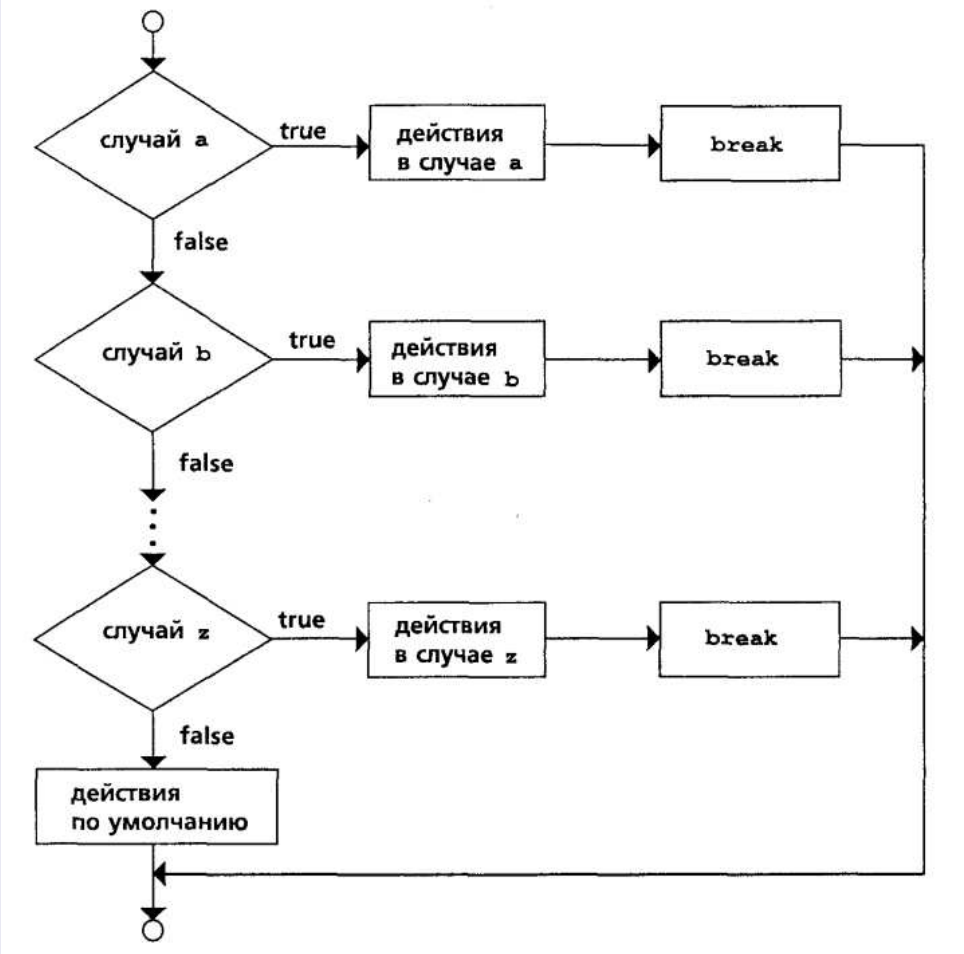

Рис. 1: Структура со множественным выбором switch

Пример использования switch

```
#include <stdio.h>
main()
{
  int s, counter=0;
  printf("Введите символ: ");
  while ((s = getchar()) != EOF)
    {
      switch (s)
       {
         case '0': case '1': case '2': case '3':
         case '4': case '5': case '6': case '7':
         case '8': case '9':
            printf("Цифра\n");
            counter++;
           break;
         case '\n': case ' ':
           break;
```

Пример использования switch

```
    default:
      printf("Введено некорректное значение\n");
      break;
    }
  }
  printf("Было введено %d цифр\n", counter);
  return 0;
}
```

Пояснения к коду

В программе в условии цикла ((s = getchar()) != EOF) сначала выполняется заключенное в скобки присваивание s = getchar(). Функция getchar() (из стандартной библиотеки ввода-вывода) считывает один символ с клавиатуры и сохраняет этот символ в целочисленной переменной s. Символы обычно хранятся в переменных типа char. Однако важной особенностью языка C является то, что символы могут храниться в любом целочисленном типе данных, поскольку они представляются в компьютере в виде 1-байтных целых чисел. Таким образом, мы можем обрабатывать символ либо как целое число, либо как символ в зависимости от его смысла. Например,

```
#include <stdio.h>
main()
{
  int s;
  s = getchar();
  printf("это символ (%c), а это его ASCII-код: (%d)", s,s);
  return 0;
}
```

Пояснения к коду

Вначале будет введен сам символ, затем его код.
Выражение ((s = getchar()) != EOF) демонстрирует еще одну особенность языка С. У присваивания (s = getchar()) есть значение, которое потом можно использовать. Например, это можно использовать для инициализации нескольких переменных одним значением:
a = b = c = 0;
Сначала оцениваается c = 0, затем переменной b присваивается значение c = 0, затем переменной a присваивается b = (c = 0), которое также равно нулю.
В программе значение (s = getchar()) сравнивается со значением EOF (это мнемоника (условное обозначение) символа конца файла). EOF - «end of file». Обычно EOF равен -1. Пользователь должен ввести системно-зависимую комбинацию клавиш, означающую «конец файла» - « у меня больше нет данных». В системах Unix и многих других индикатор EOF вводится нажатием <Ctrl-d>. В Microsoft этот индикатор вводится <Ctrl-z>.

Пояснения к коду

В теле цикла за ключевым словом switch следует имя переменной s. Она является управляющим выражением. Значение этого выражения сравнивается с каждой из меток case. Можно перечислять вместе все case, которые трубуют одинаковых действий (например, для распознавания цифры, мы перечислили все возможные случаи вместе: case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':). Если имеет место совпадение с соответствующим case, выполняются операторы для этого case. Оператор break вызывает передачу управления первому оператору после структуры switch. Необходимость этого оператора обусловлена тем, что без него метки case могут выполняться подряд. Если break вообще отсутствует, то всякий раз, когда в структуре switch будет соответствие какому-либо case, его операторы будут выполнены. Это соответствует стихотворению:
Вот дом, который построил Джек. А это пшеница, которая в тёмном чулане хранится в доме, который построил Джек... Структура switch отличается от всех других структур тем, что в блоке case последовательность операторов не требуется заключать в фигурные скобки.

Операторы **break, continue**

Данные операторы предназначены для изменения потока управления. Использование оператора **break** в структурах **while, for, do/while, switch** приводит к немедленному выходу из структуры. Обычным применением оператора **break** является досрочный выход из цикла или пропуск оставшихся операторов в switch. При выходе их структуры, выполнение программы продолжается с первого после этой структуры оператора.

Использование оператора **continue** в структурах **while, for, do/while** приводит к пропуску оставшихся операторов в теле этой структуры и выполнению следующей итерации цикла. В структурах **while, do/while** сразу же после выполнения оператора **continue** управление перейдет к проверке условия этих циклов. В структуре **for** выполняется выражение приращения, а затем проверяется условие продолжения цикла.

Рассмотрим примеры.

Пример break и for

```
#include <stdio.h>
main()
{
  int i;
  for ( i = 1; i <= 10; i++)
    {
      if ( i == 5 ) break;
      printf("%d\n",i);
    }
  return 0;
}
```

На самом деле цикл выполнится всего 4 раза.

Пример continue и for

```
#include <stdio.h>
main()
{
  int i;
  for ( i = 1; i <= 100; i++)
    {
      if ( i % 5 == 0 ) continue;
      printf("%d\n",i);
    }
  return 0;
}
```

Все числа, кратные пяти, напечатаны не будут.

Задание 4.6

Написать программу, которая с помощью операторов for выводит следующие последовательности:

- 1,2,3,4,5,6,7
- 3, 8, 13, 18, 23
- 20, 14, 8, 2, -4, -10
- 19, 27, 35, 43, 51

Задание 4.7

Написать программу, которая вычисляет и выводит сумму четных целых чисел от 2 до 30.

Задание 4.8

Написать программу управления банковским счетом:

- вначале на счете 0.00 руб.
- реализовать выбор и выполнение следующих действий: положить деньги на счет, снять деньги со счета, закрыть счет

Задание 4.9

Написать программу, которая загадывает случайное число от 1 до 100. Дает вам три попытки его отгадать.

Пример работы со случайными числами

```
/* Функция rand() генерирует последовательность псевдослучайных чисел. Всякий
  раз при ее вызове возвращается число в интервале от 0 до RAND_MAX-1 */
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    printf("RAND_MAX=%d\n", RAND_MAX);
    srand(time(NULL)); //Устанавливаем псевдослучайную последовательность
  по системному времени
    printf("%d\n ", rand()); //Случайное число от 0 до RAND_MAX-1
    printf("%d\n", rand() % 100); // Случайное число от 0 до 99
    return 0;
}
```

Задача

Рассмотрим следующую задачу:
Спросить у пользователя его температуру. Принять решение в зависимости от результата:
если температура ниже или равна 36.4, вывести сообщение «У Вас упадок сил, Вам нужно больше спать!»
если температура ниже 37 и выше 36.4, вывести сообщение «Судя по температуре, Вы здоровы.»
если температура выше или равна 37 но ниже 38, вывести сообщение «Вы заболели.»
если температура выше или равна 38, вывести сообщение «Вы заболели, нужно принять жаропонижающее средство.»
Рассмотрим различные способы решения этой задачи.

Неудачная реализация

```
#include <stdio.h>
main()
{
  float t;
  printf("Введите вашу температуру: ");
  scanf("%f", &t);
  if (t <= 36.4)
    printf("У Вас упадок сил, Вам нужно больше спать!\n");
  else if
    (t>36.4)
    {if (t<37)
printf("Судя по температуре, Вы здоровы.\n");
```

Неудачная реализация (продолжение)

```
      else if
(t >= 37)
{if (t <38)
    printf("Вы заболели.\n");
  else if
    (t>=38)
    printf("Вы заболели, нужно принять жаропонижающее средство.\n");
}
    }
  return 0;
}
```

Логика этой программы распознается с большим трудом.

Более удачная реализация

```
#include <stdio.h>
main()
{
  float t;
  printf("Введите вашу температуру: ");
  scanf("%f", &t);
  if (t <= 36.4) printf("У Вас упадок сил, Вам нужно больше спать!\n");
  if (t>36.4 && t<37) printf("Судя по температуре, Вы здоровы.\n");
  if (t >= 37 && t <38) printf("Вы заболели.\n");
  if (t>=38) printf("Вы заболели, нужно принять жаропонижающее средство.\
 n");
  return 0;
}
```

Более удачная реализация

Здесь становится все более ясно. Но после проверки условия, остальные условия также проложают проверяться. Например, если пользователь ввел $t = 36.6$, будет выведено сообщение: «Судя по температуре, Вы здоровы.», но программа не завершит свою работу, а продолжит проверять условия (t >= 37 && t <38), а потом и (t>=38), что является совершенно лишним и снижает эффективность программы.

Наиболее корректная реализация

```
#include <stdio.h>
main()
{
   float t;
   printf("Введите вашу температуру: ");
   scanf("%f", &t);
   if (t <= 36.4) printf("У Вас упадок сил, Вам нужно больше спать!\n");
   else if (t>36.4 && t<37) printf("Судя по температуре, Вы здоровы.\n")
   ;
   else if (t >= 37 && t <38) printf("Вы заболели.\n");
   else if (t>=38) printf("Вы заболели, нужно принять жаропонижающее
  средство.\n");
   return 0;
}
```

Если нет скобок, то else интерпретируется связанным с ближайшим if. Такое использование else if в C встречается довольно часто.

Логические операции

Как видно из примеров, сокращение программы стало возможным благодаря использованию логических операторов. В языке C предусмотрены логические операции, с помощью которых можно формировать сложные условия путем объединения более простых. **Логическими операциями** являются:

- && - логическое И;
- || - логическое Или;
- ! - логическое Не (отрицание).

Рассмотрим на примерах.

Логическая операция И

Если логическое выражение составлено из двух и более подвыражений и требуется, чтобы выражение было истинным, тогда и только тогда, когда истинно каждое подвыражение, то используют логическое И &&:
(a>0 && b != 0) будет отлично от нуля при истинности двух подвыражений: $a > 0, b \neq 0$;
(a>0 && b != 0 && c<-1) будет отлично от нуля при истинности трех подвыражений: $a > 0, b \neq 0, c < -1$.
Удобно обозначать работу логического оператора И с помощью так называемой *таблицы истинности*.

A	B	A && B
0	0	0
0	ненулевое	0
ненулевое	0	0
ненулевое	ненулевое	1

Логическая операция ИЛИ

Если логическое выражение составлено из двух и более подвыражений и требуется, чтобы выражение было истинным, если истинно (отлично от нуля) хотя бы одно из подвыражений, то используют операцию логическое Или ||:

(a>0 || b != 0) будет отлично от нуля при истинности хотя бы одного из подвыражений: $a > 0, b \neq 0$;

(a>0 && b != 0 && c<-1) будет отлично от нуля при истинности хотя бы одного из трех подвыражений: $a > 0, b \neq 0, c < -1$.

Таблица истинности логического оператора ИЛИ.

A	B	A || B
0	0	0
0	ненулевое	1
ненулевое	0	1
ненулевое	ненулевое	1

Операция && имеет более высокий приоритет, чем ||.

Логическая операция НЕ

Заметьте, логическое значение истина (TRUE) обозначается 1, а также любым отличным от нуля значением. Логическое значение ложь (FALSE), всегда обозначается нулем.

Если выражение составлено из одного подвыражения и требуется, чтобы выражение было истинным, если подвыражение ложно и наоборот, то используют логическую операцию Не !:

(!(b == 0)) будет отлично от нуля, если $b = 0$.

Таблица истинности логического оператора Не

A	**!A**
0	1
ненулевое	0

В большинстве случаев программист может избежать логического отрицания. Например, условие(!(b == 0)) эквивалентно (b != 0). Логическое отрицание имеет среди других логических операций наивысший приоритет.

Задание 4.10

Вычислить и вывести на экран

$$P = \prod_{k=1}^{N} \left(\frac{1}{k} + \frac{k+1}{k+2} \right)$$

Значение N ввести с клавиатуры.

Задание 4.11

Вычислить и вывести на экран

$$P = \prod_{k=1}^{N} \left(\frac{1}{k} + \frac{k+1}{k+2} \right)$$

Значение N определить из условия

$$|\frac{a_k}{P}| < 0.01, a_k = \frac{1}{k} + \frac{k+1}{k+2}$$

Задание 4.12

Вывести на экран таблицу ASCII символов:

	32	!	33	"	34	#	35	$	36		
%	37	&	38	’	39	(	40	)	41	*	42
+	43	,	44	–	45	.	46	/	47	0	48
1	49	2	50	3	51	4	52	5	53	6	54
7	55	8	56	9	57	:	58	;	59	<	60
=	61	>	62	?	63	@	64	A	65	B	66
C	67	D	68	E	69	F	70	G	71	H	72
I	73	J	74	K	75	L	76	M	77	N	78
O	79	P	80	Q	81	R	82	S	83	T	84
U	85	V	86	W	87	X	88	Y	89	Z	90
[	91	\	92	]	93	^	94	_	95	‘	96
a	97	b	98	c	99	d	100	e	101	f	102
g	103	h	104	i	105	j	106	k	107	l	108
m	109	n	110	o	111	p	112	q	113	r	114
s	115	t	116	u	117	v	118	w	119	x	120
y	121	z	122	{	123	|	124				

Задание 4.13

Даны два целых числа: D (день) и M (месяц), определяющие правильную дату. Вывести знак Зодиака, соответствующий этой дате: «Водолей» (20.1–18.2), «Рыбы» (19.2–20.3), «Овен» (21.3–19.4), «Телец» (20.4–20.5), «Близнецы» (21.5–21.6), «Рак» (22.6–22.7), «Лев» (23.7–22.8), «Дева» (23.8–22.9), «Весы» (23.9–22.10), «Скорпион» (23.10–22.11), «Стрелец» (23.11–21.12), «Козерог» (22.12–19.1).

Задание 4.14

В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и черный. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. По номеру года определить его название, если 1984 год — начало цикла: «год зеленой крысы».

Задание 4.15

Дано целое число $N > 0$. Найти квадрат данного числа, используя для его вычисления следующую формулу: $N^2 = 1 + 3 + 5 + \ldots + (2N - 1)$

Задание 4.16

Дано целое число $N > 1$. Последовательность чисел Фибоначчи F_k (целого типа) определяется следующим образом:
$F_1 = 1$,
$F_2 = 1$,
$F_k = F_{k-2} + F_{k-1}, k = 3, 4, \ldots$.
Вывести элементы $F_1, F_2, \ldots, F_n$.

Задание 4.17

Дано число $A > 1$. Вывести наибольшее из целых чисел K , для которых сумма $1 + 1/2 + \ldots + 1/K$ будет меньше A , и саму эту сумму.

Задание 4.18

Даны целые положительные числа A и B, $A > B$. Найти их наибольший общий делитель (НОД), используя алгоритм Евклида :
НОД(A , B) = НОД(B , A mod B), если B $\neq$ 0;
НОД(A , 0) = A.

Задание 4.19

Дано вещественное число ϵ. Последовательность вещественных чисел A_k определяется следующим образом:
$A_1 = 2,$
$A_k = 2 + 1/A_{k-1}, k = 2, 3, \ldots.$
Найти первый из номеров k , для которых выполняется условие
$|A_k - A_{k-1}| < \epsilon$
и вывести этот номер, а также числа A_{k-1}, A_k .