ДИСТАНЦИОННОЕ ОБУЧЕНИЕ

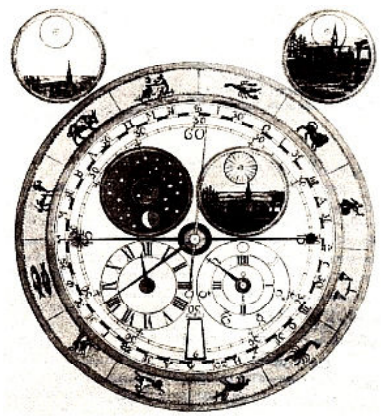
Бариев Руслан Радикович,
Свиридов Владислав Сергеевич

ЗАМЕТКИ О ПРОГРАММНОМ МОДУЛЕ «ЧАСЫ-КАЛЕНДАРЬ» КИО-2013

От редакции: На конкурсе «Конструируй, исследуй, оптимизируй» была предложена лаборатория, позволяющая создавать различные часовые механизмы. Для разных классов были предложены разные формулировки. Наиболее сложная была предложена ученикам старших классов. Участникам было предложено создать устройство, согласованно показывающее как суточное время, так и периоды солнечного года. Известно, что эта задача не простая и солнечный год представляется не целым числом суток. Авторы статьи проанализировали все допустимые условием задачи варианты шестереночных механизмов и представили лучшие решения. Также ими был обнаружен и проанализирован недостаток программного обеспечения, связанный с неточным определением малых погрешностей из-за округления в программе приближаемого числа. На диск к журналу помещена исправленная версия, однако текст статьи приводится в оригинале и демонстрирует метод анализа программы без знания её кодов.

Условие задачи. Часы – одно из самых удивительных устройств в истории техники. История механических часов начинается в Древней Греции. В Китае механические часы существовали уже более тысячи лет назад. Оттуда часы пришли в Европу, где усовершенствовались и обрели множество новых функций и форм.

Интересные часы изобрёл русский изобретатель-самородок Иван Петрович Кулибин. Сделанные им знаменитые карманные часы хранятся в Эрмитаже, другие – планетные часы – можно только представить на основании чертежей и рисунков, которые сохранились в его фонде. Плоский циферблат часов был снабжен 6 стрелками. Первая из них, имеющая на себе знак Солнца, обращалась вокруг циферблата за 365 дней. Она должна была показывать месяцы. Вторая стрелка со знаком Земли показывала начало весны, лета, осени и зимы; третья — дни недели, четвертая показывала часы; пятая — минуты; шестая — секунды.

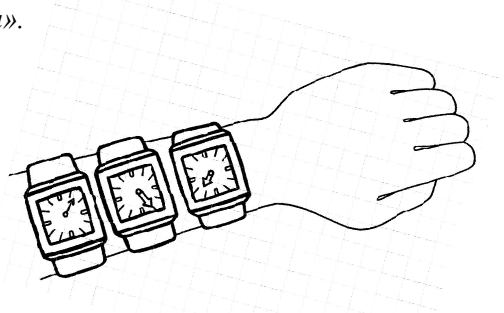


В задаче вам нужно будет собрать механизм часов-календаря так, чтобы в то время как большая стрелка проходила четырехнедельный круг (его можно было бы назвать лунным месяцем, но это будет неточно), маленькая передвигалась по годовому кругу. Сразу обращаем ваше внимание, что в году не 365 и не 366, а 365,24220… так называемых средних суток. Это связано с тем, что год – оборот Земли вокруг Солнца не равен целому числу суток – числу оборотов Земли вокруг своей оси. Поэтому сделать точные часы вам не удастся. Кроме того, у вас могут

получиться часы с двумя циферблатами (если второе колесо последней шестерни не будет сцеплено со вторым колесом на главной оси, то есть шестерни не образуют замкнутый контур). Также, может оказаться, что большая стрелка станет вращаться в обратном направлении. Это неплохо, если за каждый оборот большой стрелки будет отсчитываться ровно четыре недели, но лучшим считается решение с правильным направлением вращения. Если вращение стрелок согласовано не точно, то будет указана погрешность в процентах. Нужно, чтобы она стала как можно меньше. Ещё лучше, если вам удастся расположить шестерни часов так, чтобы обе стрелки были на главной оси. Среди таких решений лучшим будет то, у которого размер корпуса часов меньше (корпус – это круг с центром, совпадающим с центром ведущей шестерни, который появляется, как только стрелки оказываются на одной оси; размер корпуса определяется как площадь этого круга).

На заметку учителю. Известный петербургский математик Пафнутий Львович Чебышев занимался теорией механизмов и разработал теорию наилучших приближений, которая позволяет находить шестерни с минимальным числом зубьев, дающих наиболее точные передаточные отношения. Ребята могут почитать про это в Интернете по ключевым словам «цепные или непрерывные дроби».

«Задача про часы относится к разделу наилучших приближений и работам Чебышева», практически, говорит нам описание задачи. Но это далеко не так – разработанная Чебышевым теория наилучших приближений позволяет найти дробь, приближающую заданное вещественное число так, что не найдется лучшего приближения со знаменателем меньше либо равным знаменателю этой дроби. В данной задаче, во-первых, есть ограничения на число зубьев, во-вторых, шестеренок несколько. Для такой задачи теория пока не разработана. Поэтому мы попробуем решить задачу полным перебором. Таким образом, вся работа, которая требуется от нас, заключается в подборе определённого передаточного коэффициента, равного 365.2422/28 – количеству дней в году, делённому на количество дней в часовой неделе. То есть нам нужно провести факторизацию внутри условий самой задачи.



Шестерни можно представить в виде дробей. Искомый передаточный коэффициент получается перемножением первой и перевёрнутых (числитель на месте знаменателя и наоборот) остальных.

На рис. 1 шестеренка (1) связана с четырехнедельной стрелкой, шестеренка (2) с годичной. На рис. 3 шестеренка (1) связана с четырехнедельной стрелкой, шестеренка (6) с годичной.

На рис. 1 шестеренка (1) крутится быстрее шестеренки (2) в 16/8 раз, то есть быстрее в два раза. На рис. 2 шестеренка (1) крутится быстрее шестеренки (2) в 16/8 раз. Ше-

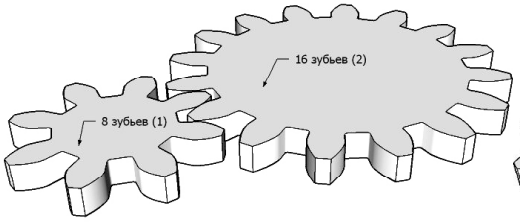


Рис. 1

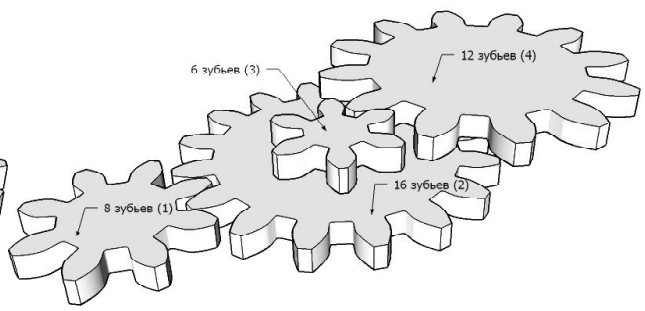


Рис. 2

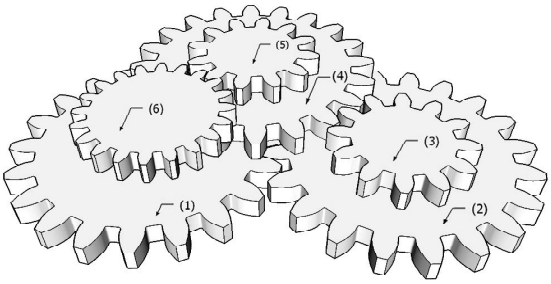


Рис. 3

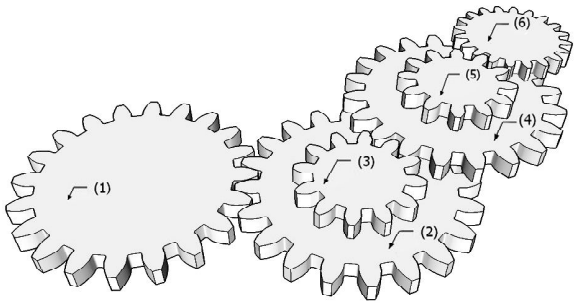


Рис. 4

стеренка (3) крутится быстрее шестеренки (4) в 12/6 раз. Шестеренки (3) и (2) намертво сцеплены, при этом получается, что шестеренка (1) крутится в (16/8) * (12/6) быстрее/медленнее шестеренки (4). Шестеренки (6) и (1) крутятся независимо друг от друга. Поэтому отношения скорости шестеренок (1) и (6) на рис. 3 можно считать как отношения шестеренок (1) и (6), как на рис. 4. Единственное отличие в том, что на рис. 3 стрелки находятся на одном циферблате, что нам и нужно.

Как и с Busy Beaver'ом, без определённого алгоритма действий (алгоритмы факторизации простого числа без серьёзных модификаций здесь не сработают) задачу легче «взять в лоб», просчитать на современном компьютере. Напишем перебирающую все возможные варианты для четырёх дробей программу. программу. Потребуется сделать более 10^{12} вычислений, что без оптимизации может занять несколько часов, так что полученные данные считываем во время отладки кода (см. листинг1).

После примерно 20 минут вычислений в памяти программы сохранится массив result, взяв элементы которого в виде { [0]/[4] [5]/[1] [6]/[2] [7]/[3] }, где [*n*] – номер элемента, мы получаем результат, показанный на рис. 5.

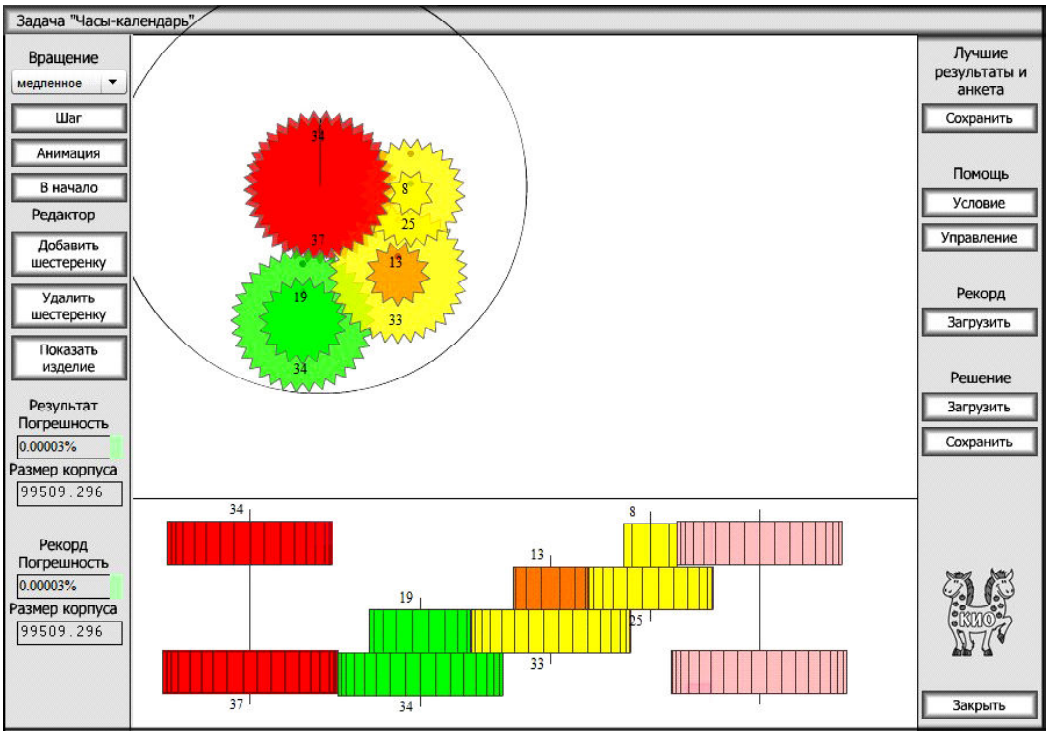


Рис. 5

Листинг 1

```
// Перебор значений, КИО-2
#include "stdafx.h"
#include <iostream>
#include <math.h>
using namespace std;

int main()
{
	double A,B,C,D,a,b,c,d, min=1000, count;
	double result[20];
	short i=0;

	A=B=C=D=a=b=c=d=8;
	while(d<41)
	{
		A++;
		if(A>40)
		{
			A=8;
			B++;
			if(B>39)
			{
				B=8;
				C++;
				if(C>39)
				{
					C=8;
					D++;
					if(D>39)
					{
						D=8;
						a++;
						if(a>39)
						{
							a=8;
							b++;
							if(b>39)
							{

								b=8;
								c++;
								if(c>39)
								{
									c=8;
									d++;
								}
							}
						}
					}
				}
			}
		}
		count = abs(1-(A*B*C*D)/(a*b*c*d*13.0443642857143))*100;
		if ( count < min)
		{
			min = count;
			result[0] = A;
			result[1] = B;
			result[2] = C;
			result[3] = D;
			result[4] = a;
			result[5] = b;
			result[6] = c;
			result[7] = d;
		}

	}
	for (i=0; i<20; i++)
		cout << result[i] << endl;
	system("pause");
	return 0;
}
```

РАСХОЖДЕНИЯ В РЕЗУЛЬТАТАХ

Полный просчет 4 шестеренок (дробей) занял 4–5 часов, так что новые полученные значения стали проверяться не только в КИО-программе, но и на калькуляторах. До определенного момента результаты были примерно равными, затем начиная резко отличаться. Изначально рассматривались 3 варианта: либо эталонное значение передаточного коэффициента взято неточное, либо алгоритм обработки чем-то отличается от выведенного, либо программа (КИО) неправильно округляет слишком малые величины. Сперва было решено остановиться на третьем. Для приведённого выше (рис. 5) решения погрешность по выведенной формуле считается

[0]	40.000000000000000
[1]	32.000000000000000
[2]	23.000000000000000
[3]	23.000000000000000
[4]	39.000000000000000
[5]	11.000000000000000
[6]	11.000000000000000
[7]	11.000000000000000

Рис. 7

$$\left|1-\frac{\frac{34}{37}\cdot\left(\frac{25}{8}\right)\cdot\left(\frac{33}{13}\right)\cdot\left(\frac{34}{19}\right)}{13.0443642857143}\right|\cdot 100 =$$

$$= 4.5962363171976\cdot 10^{-5}.$$

Или возьмём следующую комбинацию, которую выводит компьютер (рис. 6). Расчёт для этого случая:

$$\left|1-\frac{\frac{37}{21}\cdot\left(\frac{37}{21}\right)\cdot\left(\frac{32}{11}\right)\cdot\left(\frac{26}{18}\right)}{13.0443642857143}\right|\cdot 100 =$$

$$= 1.7497685145801\cdot 10^{-5}.$$

Лучший результат для 4 дробей после полного просчета (КИО снова показывает 0.00006 %) по выведенной формуле равен 0.000014 %. Так выглядело извлечение данных из программы (рис. 7).

Текущий лучший результат для 5 дробей (КИО выводит 0.00008 %):

$$\left|1-\frac{\frac{34}{37}\cdot\left(\frac{33}{23}\right)\cdot\left(\frac{32}{23}\right)\cdot\left(\frac{32}{9}\right)\cdot\left(\frac{32}{16}\right)}{\frac{365.2422}{28}}\right|\cdot 100 =$$

$$= 3.4588141589184\cdot 10^{-6}.$$

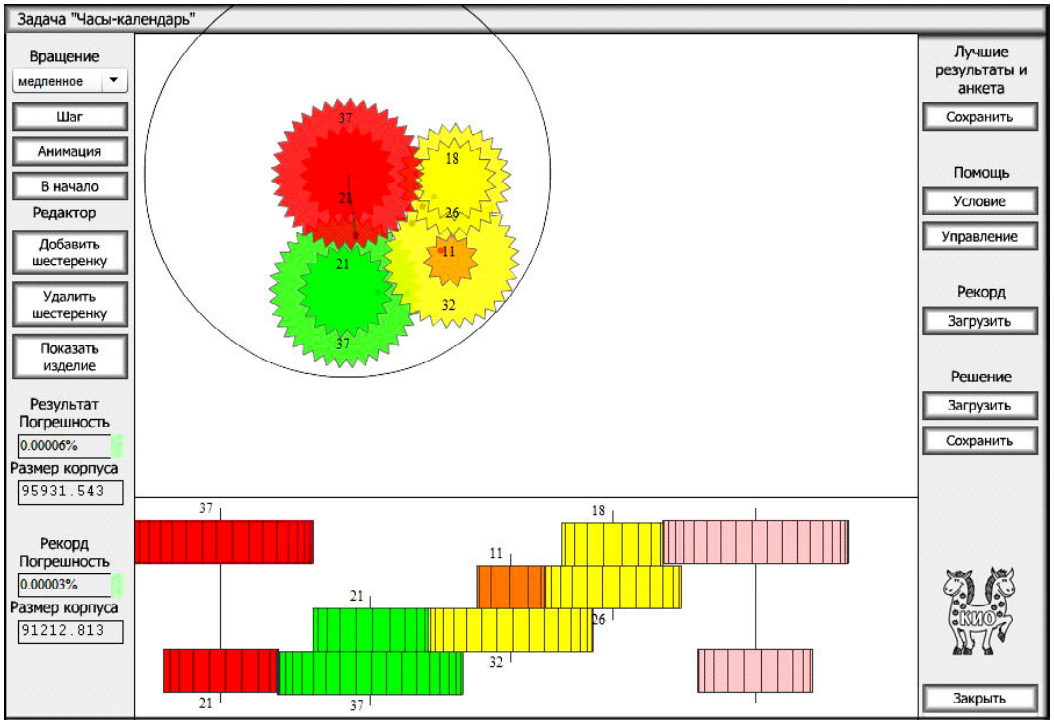


Рис. 6

Табл. 1

Количество дробей	Лучший результат погрешности (КИО)	Лучший результат погрешности (расчёт)
2	0.043%	0.043%
3	0.0004%	0.00039%
4	0.00003%	0.000014%

Табл. 1 – решения для 2, 3 и 4 дробей.

ДЕКОМПИЛЯЦИЯ

После не очень удачных попыток разобраться в ситуации мы попытались декомпилировать исходную программу, формула для расчёта оказалась той же (в другом виде, см. листинг 2). Сразу нашлась первая причина расхождения значений (см. листинг 3).

Вместе с тем дело может быть в особенности типа данных: `float` хранит числа до

Листинг 2

```
192  public function get absTransmissionError() : Number
193  {
194      return Math.abs(this.transmissionNumber - SettingsHolder.instance.levelImpl.correctRatio);
195  }// end function
196
197  public function get relTransmissionError() : Number
198  {
199      return 100 * (TransmissionMechanism.instance.absTransmissionError /
200  }// end function                              SettingsHolder.instance.levelImpl.correctRatio);
201
```

Листинг 3

```
        public function get correctRatio() : Number
        {
            return 28 / 365.243;
        }// end function
```

Листинг 4

```
202         private function get transmissionNumber() : Number
203         {
204             var _loc_1:Number = 1;
205             var _loc_2:Number = 1;
206             var _loc_3:* = this.leadingSimpleGear;
207             while (_loc_3 != null)
208             {
209
210                 if (_loc_3.getDrivenGear() != null && _loc_3.isCrossedWithDriven())
211                 {
212                     _loc_1 = _loc_1 * _loc_3.amountOfCogs;
213                     _loc_3 = _loc_3.getDrivenGear();
214                     _loc_2 = _loc_2 * _loc_3.amountOfCogs;
215                 }
216                 else
217                 {
218                     break;
219                 }
220                 if (_loc_3 != null && _loc_3.transferGear == this.firstGear)
221                 {
222                     break;
223                 }
224                 _loc_3 = _loc_3.other;
225             }
226             return _loc_1 / _loc_2;
227         }// end function
```

–149 степени двойки, но при выполнении арифметических операций точность (после 7 знака) понижается. **`transmissonNumber`** здесь – **`x`** в выведенной формуле (см. листинг 4). **`DrivenGear`** – переворот» дробей (листинг 5). И, наконец, сам вывод результата (листинг 6).

Попытка подстановки нового значения первого коэффициента выдала новые решения, снова не подходящие под результат

Листинг 5

```
76        public function getDrivenGear() : SimpleGear
77        {
78            if (this.transferGear.isFirst())
79            {
80                if (SettingsHolder.instance.isDownToUp() && this.part == LOWER_PART)
81                {
82                    return this.transferGear.getNextTransferGear().lowerGear;
83                }
84                if (!SettingsHolder.instance.isDownToUp() && this.part == UPPER_PART)
85                {
86                    return this.transferGear.getNextTransferGear().upperGear;
87                }
88            }
89            else if (SettingsHolder.instance.isDownToUp() && this.part == UPPER_PART)
90            {
91                if (this.transferGear.getNextTransferGear().isFirst())
92                {
93                    return this.transferGear.getNextTransferGear().upperGear;
94                }
95                return this.transferGear.getNextTransferGear().lowerGear;
96            }
97            else if (!SettingsHolder.instance.isDownToUp() && this.part == LOWER_PART)
98            {
99                if (this.transferGear.getNextTransferGear().isFirst())
100               {
101                   return this.transferGear.getNextTransferGear().lowerGear;
102               }
103               return this.transferGear.getNextTransferGear().upperGear;
104           }
105           return null;
106       }// end function
107
```

Листинг 6

```
116       public function getFormattedError(param1:Number) : String
117       {
118           var _loc_2:* = printf("%.10f", param1);
119           var _loc_3:* = _loc_2.substr(0, _loc_2.indexOf(".") + 3);
120           var _loc_4:* = _loc_2.indexOf(".") + 3;
121           while (_loc_4 < _loc_2.length)
122           {
123
124               if (_loc_2.charAt(_loc_4) == "0")
125               {
126                   _loc_3 = _loc_3 + "0";
127               }
128               else
129               {
130                   return _loc_3 + _loc_2.charAt(_loc_4) + "%";
131               }
132               _loc_4++;
133           }
134           return _loc_3 + "%";
135       }// end function
```

программы-КИО (рис. 8).

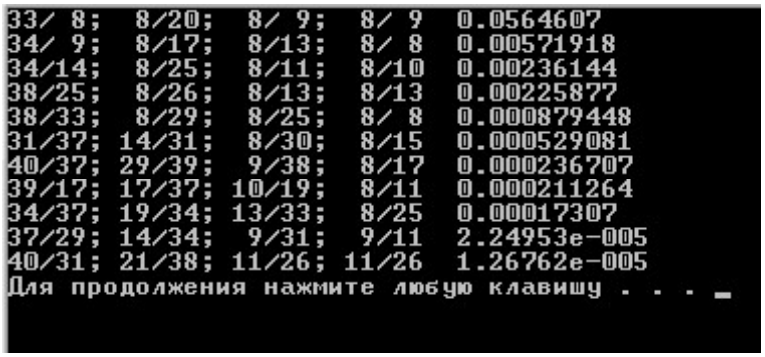


Рис. 8

ЗАКЛЮЧЕНИЕ

О несовпадении.

Возможны следующие варианты:

- Какая-либо ошибка в интерпретации кода и/или особенности типа данных (возможно, исправленной в компиляторе или в новой версии языка), формула нахождения передаточного коэффициента отличается от выведенной.
- Самый вероятный – неточный подсчёт (365.243 вместо 365.2422) + округление `float` или что-то другое в исходной программе даёт неверный ответ при высоких точностях.

Результаты приведены в табл. 2.

Табл. 2

	Кол-во дробей-шестеренок	Погрешность (округленная)	Лишний день раз в
КИО	4	0.000030%	9 132 года
Просчёт с неточным коэфф.	4	0.000012%	21 613 лет
Просчёт с точным коэфф.	5	0.000003%	79 210 лет

Бариев Руслан Радикович,
Свиридов Владислав Сергеевич,
студенты кафедры АСОИУ
направления «Информационные системы и технологии»
ФКТИ СПбГЭТУ «ЛЭТИ».