МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Інститут математики, економіки та механіки

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

бакалавра

(освітньо-кваліфікаційний рівень)

на тему «Генератори псевдовипадкових чисел»
«Генераторы псевдослучайных чисел»
«Pseudorandom number generators»

Виконав: студент 5 курсу, групи ЗВ
напряму підготовки (спеціальності)
6.050102 комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

Токмовцев А.А.
(прізвище та ініціали)

Керівник Варбанець П.Д.
(прізвище та ініціали)

Рецензент Савастру О.В.
(прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№_____ від «___»_____р.

Завідувач кафедри
_____ Варбанець П.Д.
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол №___ від «___» _____р.
Оцінка _____
(за 4-х бальною шкалою, за шкалою ECTS, бал.)

Голова ЕК
_____ Арсірій О.О.
(підпис) (прізвище, ініціали)

Одеса - 2017

АНОТАЦІЯ

У даній дипломній роботі розглядається тема «Генератори псевдовипадкових чисел».

Метою роботи є програмна реалізація алгоритмів генераторів псевдовипадкових чисел.

У даній роботі описані основи теорії методів генерування псевдовипадкових чисел, їх перевірка, видозміна. Розглянуто наступні методи: метод Середин квадратів фон Неймана, лінійний конгруентний метод, квадратичний конгруентний метод, метод Фібоначчі з запізнюваннями і віднімання (адитивний генератор), інверсний конгруентний метод. Програмно реалізовано наступні методи: лінійний конгруентний метод, метод Фібоначчі з запізнюваннями і віднімання (адитивний генератор), інверсний конгруентний метод.

Програми виконують всі необхідні функції і дозволяють автоматизувати процес генерування псевдовипадкових чисел.

АННОТАЦИЯ

В данной дипломной работе рассматривается тема «Генераторы псевдослучайных чисел».

Целью работы является программная реализация алгоритмов генераторов псевдослучайных чисел.

В данной работе описаны основы теории методов генерирования псевдослучайных чисел, их проверка, видоизменение. Рассмотрены следующие методы: метод средин квадратов фон Неймана, линейный конгруэнтный метод, квадратичный конгруэнтный метод, метод Фибоначчи с запаздываниями и вычитаниями (аддитивный генератор), инверсный конгруэнтный метод. Программно реализовано следующие методы: линейный конгруэнтный метод, метод Фибоначчи с запаздываниями и вычитаниями (аддитивный генератор), инверсный конгруэнтный метод.

Программы выполняют все требуемые функции и позволяют автоматизировать процесс генерирования псевдослучайных чисел.

ABSTRACT

In this diploma work, the topic "Pseudorandom number generators" is considered.

The aim of the work is the software implementation of algorithms for pseudorandom number generators.

In this paper, the fundamentals of the theory of methods for generating pseudorandom numbers, their verification, and modification are described. The following methods are considered: the von Neumann square method, linear congruential method, quadratic congruential method, lagged Fibonacci sequences with subtraction (additive generator), inversive congruential method. The following methods are implemented programmatically: linear congruential method, lagged Fibonacci sequences with subtraction (additive generator), inversive congruential method.

The programs perform all the required functions and allow to automate the process of generating pseudorandom numbers.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6
1 ИССЛЕДОВАНИЕ МЕТОДОВ ГЕНЕРИРОВАНИЯ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ И ИХ ТЕСТИРОВАНИЕ 10
1.1 Генерирование равномерно распределенных случайных чисел 11
1.2 Линейный конгруэнтный метод 12
1.3 Другие методы 15
2 ИССЛЕДОВАНИЕ ПАРАМЕТРОВ ЛИНЕЙНО КОНГРУЭНТНОГО МЕТОДА 22
2.1 Выбор модуля 22
2.2 Выбор множителя 27
3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ 29
3.1 Реализация линейно конгруэнтного метода 29
3.2 Реализация метода Фибоначчи с запаздываниями и вычитаниями 30
3.3 Реализация линейно-инверсного конгруэнтного метода 31
ЗАКЛЮЧЕНИЕ 32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33
ДОПОЛНЕНИЕ А. КОД ПРОГРАММЫ РЕАЛИЗАЦИИ ЛИНЕЙНО КОНГРУЭНТНОГО МЕТОДА 34
ДОПОЛНЕНИЕ Б. КОД ПРОГРАММ РЕАЛИЗАЦИИ МЕТОДА ФИБОНАЧЧИ С ЗАПАЗДЫВАНИЯМИ И ВЫЧИТАНИЯМИ 36
ДОПОЛНЕНИЕ В. КОД ПРОГРАММЫ РЕАЛИЗАЦИИ ЛИНЕЙНО-ИНВЕРСНОГО КОНГРУЭНТНОГО МЕТОДА 42

ВВЕДЕНИЕ

Числа, которые выбираются случайным образом, находят множество полезных применений.

a) Моделирование. При использовании компьютера для моделирования естественных явлений случайные числа нужны для того, чтобы сделать эти модели похожими на реальные явления. Моделирование применяется во многих областях, начиная от исследований в ядерной физике (где частицы испытывают случайные столкновения) и заканчивая исследованием операций (где люди прибывают, например, в аэропорт через случайные промежутки времени).

b) Выборочный метод. Часто бывает невозможно исследовать все варианты, но случайная выборка обеспечивает понимание того, что можно назвать «типичным» поведением.

c) Численный анализ. Для решения сложных задач численного анализа была разработана остроумная техника, использующая случайные числа. Об этом написано несколько книг.

d) Компьютерное программирование. Случайные величины являются хорошим источником данных тестирования эффективности компьютерных алгоритмов. Более важно то, что они играют решающую роль при использовании рандомизированных алгоритмов, которые часто намного превосходят своих детерминированных двойников.

e) Принятие решений. Многие администраторы принимают решения, бросая монету, игральную кость либо каким-нибудь другим подобным способом. Иногда важно принять полностью «беспристрастное» решение. Случайность является также важной частью оптимальных стратегий в теории матричных игр.

f) Эстетика. Небольшая добавка случайности оживляет музыку и компьютерную графику.

g) Развлечения. Многие считают, что они замечательно проводят время, бросая игральные кости, тасуя колоду карт, вращая колесо рулетки и т.п. Такие традиционные способы использования случайных чисел получили название метод Монте-Карло. Это общее название всех алгоритмов, использующих случайные числа.

Те же, кто интересуются этой темой, постоянно вовлекаются в философские дискуссии о значении слова «случайный». Возникает вопрос «А что является не случайным числом?». Например, будет ли число 2 случайным? Охотнее говорят о последовательности независимых случайных чисел с заданным распределением, и это означает, если говорить не строго, что каждое число было получено случайно, не имея ничего общего с другими числами в последовательности, и что каждое число имеет заданную вероятность появления в любой заданной области значений.

Равномерным распределением на конечном множестве чисел (в дальнейшем – просто равномерным распределением) называется такое распределение, при котором любое из возможных чисел имеет одинаковую вероятность появления. Если не задано определенное распределение на конечном множестве чисел, что принято считать его равномерным.

Каждая из десяти цифр от 0 до 9 будет появляться примерно один раз из 10 в равномерной последовательности случайных цифр. Каждой паре двух последовательных цифр следует появиться один раз из ста и т.д. Однако если взять конкретную случайную последовательность длиной в миллион цифр, то она не всегда будет содержать 100 000 нулей, 100 000 единиц и т.д. Действительно, возможность появления такой последовательности незначительна; на самом деле, если рассматривать достаточно большую

совокупность таких последовательностей, то в среднем будет появляться 100 000 нулей, 100 000 единиц и т.д.

Любая конкретная последовательность, содержащая миллион цифр, так же вероятна, как и любая другая. Если мы выберем миллион цифр наудачу и если окажется, что первые 999 999 из них – нули, то вероятность того, что последняя цифра в этой последовательности – также нуль, все еще остается точно равной одной десятой в истинно случайной ситуации. Это утверждение большинству кажется парадоксальным, однако оно не противоречит реальности.

В течение многих лет те, кому случайные числа были необходимы для научной работы, вынуждены были таскать шары из урны, предварительно хорошо перемешав их, либо бросать игральные кости, либо раскладывать карты.

С тех пор были построены механические генераторы случайных чисел. Первая такая машина была использована в 1939 году М. Ж. Кендаллом (M. G. Kendall) и Б. Бабингтон-Смитом (B. Babington-Smith) для построения таблицы, содержащей 100 000 случайных цифр. Компьютер Ferranti Mark I, впервые запущенный в 1951 году, имел встроенную программу, использующую резисторный генератор шума, которая поставляла 20 случайных битов на сумматор. Этот метод был предложен А. М. Тьюрингом (A. M. Turing). В 1955 году RAND Corporation опубликовала широко используемые таблицы, в которых содержался миллион случайных цифр, полученных с помощью других специальных устройств. Известный генератор случайных чисел ERNIE применялся на протяжении многих лет для определения выигрышных номеров британской лотереи.

Короче говоря, после изобретения компьютеров начались исследования эффективного способа получения случайных чисел, встроенных программно в компьютеры. Можно было применять таблицы, но пользы от этого метода

было мало из-за ограниченной памяти компьютера и требуемого времени ввода, поэтому таблицы могли быть лишь слишком короткими; кроме того, было не особенно приятно составлять таблицы и пользоваться ими. Генератор ERNIE мог быть встроен в компьютер, как в Ferranti Mark I, но это оказалось неудобно, поскольку невозможно точно воспроизвести вычисления даже сразу по окончании работы программы; более того, такие генераторы часто давали сбои, что было крайне трудно обнаружить. Технологический прогресс позволил вернуться к использованию таблиц в 90-е годы, так как миллиарды тестированных случайных байтов можно было разместить на компакт-дисках.

Главной целью данной работы является программная реализация алгоритмов генераторов псевдослучайных чисел.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследование методов генерирования псевдослучайных чисел и их тестирование;
- исследование параметров линейно конгруэнтного метода;
- программная реализация алгоритмов.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы были программно реализованы алгоритмы генераторов псевдослучайных чисел.

Были реализованы: линейный конгруэнтный метод, метод Фибоначчи с запаздываниями и вычитаниями (аддитивный генератор), инверсный конгруэнтный метод.

Программы выполняют все требуемые функции и позволяют автоматизировать процесс генерирования псевдослучайных чисел.

Были исследованы методы генерирования псевдослучайных чисел и их тестирование.

Были исследованы параметры линейно конгруэнтного метода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кнут Д. Искусство программирования, том 2. Получисленные алгоритмы. Вильямс, Москва, Россия, 2001.

2. Lehmer D. Proceedings of a Second Symposium on Large Scale Digital Calculating Machinery. Harvard University Press, Cambridge, Massachusetts, USA, 1951, 141-146.

3. Gauss C. Disquisitiones Arithmeticae. Yale University Press, New Haven, Connecticut, USA, 1965, §90-92.

4. Green B., Smith K., Klem L. Empirical Tests of an Additive Random Number Generator. Journal of the Association for Computing Machinery, Vol. 6, New York, USA, 1959, 527-537.

5. Eichenauer J., Lehn J. A non-linear congruential pseudo random number generator. Statistische Hefte, Springer Berlin Heidelberg, Vol. 27, Berlin, Germany, 1986, 315-326.

6. Luscher M. A portable high-quality random number generator for lattice field theory simulations. Computer Physics Communications, Vol. 79, Elsevier Science, Amsterdam, Netherlands, 1994, 100-110.

7. Todd J., Taussky O. Generation and testing of pseudo-random numbers. Symposium on Monte Carlo methods, University of Florida, Wiley, New York, USA, 1956, 15-28.

8. Marsaglia G. A current view of random number generators. Computer science and statistics: Symposium on the Interface 16, Atlanta, Georgia, USA, March 1984, Elsevier Science, Amsterdam, Netherlands, 3-10.

9. Niederreiter H. Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992.

ДОПОЛНЕНИЕ А. КОД ПРОГРАММЫ РЕАЛИЗАЦИИ ЛИНЕЙНО КОНГРУЭНТНОГО МЕТОДА

```
#include "stdafx.h"
#include <iostream>
#include <clocale>
#include <ctime>
using namespace std;
long double congruential(unsigned long &);
// прототип функции генерации случайных чисел
     unsigned int m,
                 a,
                 c;
     int N;
     int main(int argc, char* argv[])
     {
          setlocale(LC_ALL, "Russian");
cout << "Укажите значение m (0 < m): ";
//Вывод информации на консоль
     cin >> m; //Присвоение значения пользователем через консоль
          while (m <= 0)
          {
               cout << "m должно быть больше нуля";
               cout << "\n";
               cout << "Укажите значение m (0 < m): ";
               cin >> m;
          }
               cout << "Укажите значение a (0 <= a < m): ";
               cin >> a;
          while (a > m || a < 0)
          {
               cout << "введенное a больше m или меньше нуля ";
               cout << "\n";
               cout << "Укажите значение a (0 <= a < m): ";
```

```
            cin >> a;
        }
        cout << "Укажите значение c (0 <= c < m): ";
        cin >> c;
        while (c > m || c < 0)
        {
            cout << "введенное c больше m или меньше нуля";
            cout << "\n";
            cout << "Укажите значение c (0 <= c < m): ";
            cin >> c;
        }
cout << "Укажите значение N (количество случайных чисел): ";
        cin >> N;
        while (N < 0)
        {
            cout << "введенное N меньше нуля";
            cout << "\n";
cout << "Укажите значение N (количество случайных чисел): ";
            cin >> N;
        }
        unsigned long x0; // начальное значение
cout << "Укажите значение x0(начальное значение(0 <= x0 < m)):";
        cin >> x0;
        while (x0 > unsigned int(m) || x0 < 0)
        {
            cout << "введенное x0 больше m или меньше нуля";
            cout << "\n";
cout << "Укажите значение x0(начальное значение(0 <= x0 < m)):";
            cin >> x0;     }
        unsigned int start_time =  clock(); // начальное время
        cout << (x0 / long double(m)) << ", ";
        for (int i = 0; i <= N; i++)
    cout << congruential(x0) << ", "; // генерация i-го числа
```

```
          cout << "\n";
          unsigned int end_time = clock(); // конечное время
          unsigned int search_time = end_time - start_time;
// искомое время
cout << "Время работы для вычислений данного объема(N) = " <<
search_time/1000.0 << " секунд" <<endl;
          system("pause");
          return 0;
     }
     long double congruential(unsigned long &x)
     // функция генерации случайных чисел
     {
          //cout << x << "\n"; // проверка корректности чисел
          x = ((a * x) + c) % m;
// формула линейного конгруэнтного метода генерации случайных
// чисел
          return (x / long double(m));
     }
```

ДОПОЛНЕНИЕ Б. КОД ПРОГРАММ РЕАЛИЗАЦИИ МЕТОДА ФИБОНАЧЧИ С ЗАПАЗДЫВАНИЯМИ И ВЫЧИТАНИЯМИ

1) для генерирования случайных целых чисел:

```
#include <stdio.h>
#define KK 100                     /* длинное запаздывание */
#define LL  37                    /* короткое запаздывание */
#define MM (1L<<30)                   /* модуль */
#define mod_diff(x,y) (((x)-(y))&(MM-1)) /* (x-y) mod MM */
long ran_x[KK];                    /* состояние генератора */
#ifdef __STDC__
void ran_array(long aa[],int n)
#else
void ran_array(aa,n)
  long *aa;
```

```
      int n;
    #endif
    {
      register int i,j;
      for (j=0;j<KK;j++) aa[j]=ran_x[j];
      for (;j<n;j++) aa[j]=mod_diff(aa[j-KK],aa[j-LL]);
for (i=0;i<LL;i++,j++) ran_x[i]=mod_diff(aa[j-KK],aa[j-LL]);
for (;i<KK;i++,j++) ran_x[i]=mod_diff(aa[j-KK],ran_x[i-LL]);
    }
    #define QUALITY 1009
    long ran_arr_buf[QUALITY];
    long ran_arr_dummy=-1, ran_arr_started=-1;
    long *ran_arr_ptr=&ran_arr_dummy; /* следующее случайное
число или -1 */
    #define TT  70   /* гарантирует разделение потоков */
    #define is_odd(x)  ((x)&1)   /* блок двоичных разрядов x */
    #ifdef __STDC__
    void ran_start(long seed)
    #else
    void ran_start(seed)
      long seed;
    #endif
    {
      register int t,j;
      long x[KK+KK-1];              /* подготовка буфера */
      register long ss=(seed+2)&(MM-2);
      for (j=0;j<KK;j++) {
        x[j]=ss;                      /* самозагрузка буфера */
        ss<<=1; if (ss>=MM) ss-=MM-2; /* циклический сдвиг 29
двоичных разрядов */
      }
      x[1]++;        /* делаем x[1] (и только x[1]) нечетным */
      for (ss=seed&(MM-1),t=TT-1; t; ) {
```

```
for (j=KK-1;j>0;j--) x[j+j]=x[j], x[j+j-1]=0; /* "квадрат" */
          for (j=KK+KK-2;j>=KK;j--)
            x[j-(KK-LL)]=mod_diff(x[j-(KK-LL)],x[j]),
            x[j-KK]=mod_diff(x[j-KK],x[j]);
          if (is_odd(ss)) {              /* "умножаем на z" */
            for (j=KK;j>0;j--)  x[j]=x[j-1];
            x[0]=x[KK];            /* сдвигаем буфер циклично */
            x[LL]=mod_diff(x[LL],x[KK]);
          }
          if (ss) ss>>=1; else t--;
        }
        for (j=0;j<LL;j++) ran_x[j+KK-LL]=x[j];
        for (;j<KK;j++) ran_x[j-LL]=x[j];
        for (j=0;j<10;j++) ran_array(x,KK+KK-1);
        ran_arr_ptr=&ran_arr_started;
      }
#define ran_arr_next() (*ran_arr_ptr>=0? *ran_arr_ptr++:
ran_arr_cycle())
      long ran_arr_cycle()
      {
        if (ran_arr_ptr==&ran_arr_dummy)
          ran_start(314159L);
        ran_array(ran_arr_buf,QUALITY);
        ran_arr_buf[KK]=-1;
        ran_arr_ptr=ran_arr_buf+1;
        return ran_arr_buf[0];
      }
      int main()
      {
      register int m; long a[2009];
        ran_start(310952L);
        for (m=0;m<=2009;m++) ran_array(a,1009);
        for (m=0;m<100;m++)
```

```
  {
       printf("%ld\n", a[m]);
  }
  ran_start(510952L);
  printf("\n");
  for (m=0;m<=1009;m++) ran_array(a,2009);
  for (m=0;m<100;m++)
  {
      printf("%ld\n", a[m]);
  }
  return 0;
}
```

2) для генерирования случайных дробей между 0 и 1:

```
#include <stdio.h>
#define KK 100                    /* длинное запаздывание */
#define LL  37                   /* короткое запаздывание */
#define mod_sum(x,y) (((x)+(y))-(int)((x)+(y)))   /* (x+y)
mod 1.0 */
double ran_u[KK];           /* состояние генератора */
#ifdef __STDC__
void ranf_array(double aa[], int n)
#else
void ranf_array(aa,n)
  double *aa;
  int n;
#endif
{
  register int i,j;
  for (j=0;j<KK;j++) aa[j]=ran_u[j];
  for (;j<n;j++) aa[j]=mod_sum(aa[j-KK],aa[j-LL]);
for (i=0;i<LL;i++,j++) ran_u[i]=mod_sum(aa[j-KK],aa[j-LL]);
for (;i<KK;i++,j++) ran_u[i]=mod_sum(aa[j-KK],ran_u[i-LL]);
}
```

```
#define QUALITY 1009
double ranf_arr_buf[QUALITY];
double ranf_arr_dummy=-1.0, ranf_arr_started=-1.0;
double *ranf_arr_ptr=&ranf_arr_dummy; /* следующее
случайное число или -1 */
#define TT  70   /* гарантирует разделение потоков */
#define is_odd(s) ((s)&1)
#ifdef __STDC__
void ranf_start(long seed)
#else
void ranf_start(seed)
  long seed;
#endif
{
  register int t,s,j;
  double u[KK+KK-1];
  double ulp=(1.0/(1L<<30))/(1L<<22);      /* от 2 к -52 */
  double ss=2.0*ulp*((seed&0x3fffffff)+2);
  for (j=0;j<KK;j++) {
    u[j]=ss;
    ss+=ss; if (ss>=1.0) ss-=1.0-2*ulp;  /* циклический
сдвиг на 51 двоичный разряд */
  }
  u[1]+=ulp; /* получаем u[1] (и только u[1]) "нечетное" */
  for (s=seed&0x3fffffff,t=TT-1; t; ) {
    for (j=KK-1;j>0;j--)
      u[j+j]=u[j],u[j+j-1]=0.0;  /* возведение в квадрат */
    for (j=KK+KK-2;j>=KK;j--) {
      u[j-(KK-LL)]=mod_sum(u[j-(KK-LL)],u[j]);
      u[j-KK]=mod_sum(u[j-KK],u[j]);
    }
    if (is_odd(s)) {                 /* "умножение на z" */
      for (j=KK;j>0;j--) u[j]=u[j-1];
```

```
            u[0]=u[KK];            /* циклический сдвиг буфера */
            u[LL]=mod_sum(u[LL],u[KK]);
          }
          if (s) s>>=1; else t--;
        }
        for (j=0;j<LL;j++) ran_u[j+KK-LL]=u[j];
        for (;j<KK;j++) ran_u[j-LL]=u[j];
        for (j=0;j<10;j++) ranf_array(u,KK+KK-1);
        ranf_arr_ptr=&ranf_arr_started;
      }
      #define ranf_arr_next() (*ranf_arr_ptr>=0? *ranf_arr_ptr++:
ranf_arr_cycle())
      double ranf_arr_cycle()
      {
        if (ranf_arr_ptr==&ranf_arr_dummy)
        ranf_start(314159L);
        ranf_array(ranf_arr_buf,QUALITY);
        ranf_arr_buf[KK]=-1;
        ranf_arr_ptr=ranf_arr_buf+1;
        return ranf_arr_buf[0];
      }
      int main()
      {
        register int m; double a[2009];
        ranf_start(310952);
        for (m=0;m<2009;m++) ranf_array(a,1009);
        for (m=0;m<KK;m++)
        {
             printf("%.20f\n", ran_u[m]);
        }
        ranf_start(510952);
        printf("\n");
        for (m=0;m<1009;m++) ranf_array(a,2009);
```

```
  for (m=0;m<KK;m++)
  {
      printf("%.20f\n", ran_u[m]);
  }
  return 0;
}
```

ДОПОЛНЕНИЕ В. КОД ПРОГРАММЫ РЕАЛИЗАЦИИ ЛИНЕЙНО-ИНВЕРСНОГО КОНГРУЭНТНОГО МЕТОДА

```
#include "stdafx.h"
#include <iostream>
#include <cmath>
#include <clocale>
#include <ctime>
using namespace std;
long double congruential(unsigned long &);
// прототип функции генерации случайных чисел
int isprostoe(int);
// прототип функции опередеделения простое ли число
     int p,
     m,
     a,
     b,
     c;
int main(int argc, char* argv[])
{
     setlocale(LC_ALL, "Russian");
cout << "Укажите значение p (нечётное простое): ";
//Вывод информации на консоль
cin >> p; //Присвоение значения пользователем через консоль
     while (isprostoe(p) == 0) // простое ли число введено
     {
          cout << "Число не простое";
          cout << "\n";
```

```
            cout << "Укажите значение p (нечётное простое): ";
            cin >> p;
        }
        cout << "Укажите значение m (натуральное): ";
        cin >> m;
    while (m < 1) // натуральное ли число введено
        {
            cout << "Число не натуральное";
            cout << "\n";
            cout << "Укажите значение m (натуральное): ";
            cin >> m;
        }
    cout << "Укажите значение a (a не должно делится на p): ";
        cin >> a;
        while (a % p == 0) // делится ли a на p
        {
            cout << "Число a делится на p";
            cout << "\n";
    cout << "Укажите значение a (a не должно делится на p): ";
            cin >> a;
        }
        cout << "Укажите значение b (b должно делится на p): ";
        cin >> b;
        while (b % p != 0) // делится ли b на p
        {
            cout << "Число b не делится на p";
            cout << "\n";
        cout << "Укажите значение b (b должно делится на p): ";
            cin >> b;
        }
cout << "Укажите значение c (c должно делится на p в кубе): ";
        cin >> c;
    while (c % (int)pow(p,3) != 0) // делится ли c на p в кубе
```

```
          {
               cout << "Число c не делится на p в кубе";
               cout << "\n";
cout << "Укажите значение c (c должно делится на p в кубе): ";
               cin >> c;
          }
          const int N = 2 * (int)pow(3,8);
        // количество случайных чисел (13122)
          unsigned long y0; // начальное значение
          cout << "Укажите значение y0 (начальное значение y): ";
          cin >> y0;
         while (y0 > (unsigned int)pow(p,m) || y0 < 0)
          {
               cout << "введенное y0 больше p^m или меньше нуля";
               cout << "\n";
cout << "Укажите значение y0(начальное значение(0<=y0<p^m)): ";
               cin >> y0;
              }
          unsigned int start_time =  clock(); // начальное время
          for (int i = 0; i <= N; i++)
      cout << congruential(y0) << "), "; // генерация i-го числа
          cout << "\n";
          unsigned int end_time = clock(); // конечное время
          unsigned int search_time = end_time - start_time;
          // искомое время
cout << "Время работы для вычислений данного объема(N) = " <<
search_time/1000.0 << " секунд" <<endl;
          system("pause");
          return 0;
     }
     long double congruential(unsigned long &y)
     // функция генерации случайных чисел
     {
```

```
            unsigned long d = (unsigned long)pow(y,2),
             mod = (unsigned long)pow(p,m);
            long double dmod = pow(p,m);
          //cout << y << "   "; // проверка корректности чисел
          cout << "(";
          cout << (y / dmod) << ", ";
          y = ( ( a / y ) + b + (c * d)) % mod;
          return (y / dmod);
     }
     int isprostoe(int pp) // возвращает 1, если p-простое и 0,
// если p не простое
     {
     int d;
     if (pp<2) return 0;
     if (pp==2) return 1; // 2 - простое
     if (pp%2==0) return 0; // четное -> не простое
     d=3; // начальный делитель
     while (d*d<=pp)
     {
          if (pp%d==0) return 0;
          d+=2; // следующий делитель
     }
     return 1; // делителей не найдено -> простое
     }
```