МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет автоматики и вычислительной техники

Кафедра АСУ

**ОТЧЕТ**

по лабораторной работе №3

«**Наследование. Полиморфизм подтипов. Контейнеры»**

Дисциплина «Программирование», 2 семестр

Преподаватель: **Попов Е.А.**

Группа: АВТ - 113

Студенты:

**Войнова А.М.**

**Томилов М.Д.**

**Осокин Д.М.**

г. Новосибирск

2022 год

**Задание**

Реализуйте класс, моделирующий заданный тип объектов предметной области.

**4. Генераторы последовательностей чисел.**Разработать классы СлучайныйГенератор и ГенераторСПостояннымШагом, выдающий следующее число, отличающееся от предыдущего на заданный постоянный шаг. У генераторов должен быть метод СледующееЧисло, а также метод установки предыдущего числа (сброса в это состояние). У каждого объекта классов генераторов должно быть имя в виде текстовой строки. Добавьте каждому генератору метод возврата среднего последних N выданных чисел. N передаётся в конструктор генератора. Если ещё не были сгенерированы хотя бы N чисел, то возможны три варианта поведения: возврат среднего арифметического от сгенерированных чисел, которых меньше N; выброс исключения; возврат NotANumber.

Разработайте класс СоставнойГенератор, выдающий среднее арифметическое чисел, сгенерированных входящими в его состав СлучайнымиГенераторами и ГенераторамиСПостояннымШагом. Реализуйте расчёт среднего арифметического по последним N выданным числам.

Выполнить перепроектирование (рефакторинг) исходного кода предыдущей ЛР: определить базовые классы, построить иерархии наследования, объединить/заменить старые методы. При работе с объектами одной иерархии классов вы должны пользоваться как можно более абстрактным классом из имеющихся, а поэтому и полиморфизмом подтипов. Теперь вместо ручного хранения объектов в массивах и управления памятью используйте контейнеры (коллекции).

**Тестовые данные и ожидаемые результаты.**

|  |  |  |
| --- | --- | --- |
| № | Ввод для СлучайногоГенератора | Ожидаемый результат |
| 1 | Сгенерировать 10 чисел | 61  2  70  26  14  21  38  35  5  66 |
| 2 | Среднее Арифметическое последних 4 чисел | 36 |
| 3 | Вернуть предпоследнее значение | 5 |

|  |  |  |
| --- | --- | --- |
| № | Ввод для ГенератораСПостояннымШагом | Ожидаемый результат |
| 1 | Сгенерировать 10 чисел | 85  87  89  91  93  95  97  99  101  103 |
| 2 | Среднее Арифметическое последних 4 чисел | 100 |
| 3 | Вернуть предпоследнее значение | 101 |

|  |  |  |
| --- | --- | --- |
| № | Ввод для СоставногоГенератора | Ожидаемый результат |
| 1 | Сгенерировать число на основе ГСЧ и ГсПШ | (Cумма чисел ГСЧ + сумма чисел ГсПШ ) / кол-во всех чисел |
| 2 | Среднее Арифметическое последних N чисел | Среднее Арифметическое число последнего добавленного генератора |

**Проектирование свойств класса.**

1. Класс CоставнойГенератор (CompositeGen)

|  |  |  |
| --- | --- | --- |
| **Элементы класса:** | **Типы данных:** | **Ограничения:** |
| Averarr | int | Не может быть равно 0 |
| RGarr | std::unordered\_map <std::string, std::shared\_ptr<RandomGen> | динамический массив для генератора рандомных чисел |
| CGarr | std::unordered\_map <std::string, std::shared\_ptr<ConststepGen> | динамический массив для генератора с постоянным шагом |
| counter | int | Не может быть равно 0 |

1. Класс ГенераторСПостояннымШагом (ConststepGen)

|  |  |  |
| --- | --- | --- |
| **Элементы класса:** | **Типы данных:** | **Ограничения:** |
| step | int | Не может быть равно 0 |

1. Абстрактный класс (PrototypeGen.h)

|  |  |  |
| --- | --- | --- |
| **Элементы класса:** | **Типы данных:** | **Ограничения:** |
| name | string | Имя не может быть пустой строкой |
| numbers | int | Нет ограничений |
| N | int | N>1 |
| Shortage | Enum ShortageOfNumbers | Элементы перечисления |

**Разработка конструкторов.**

1. Класс CоставнойГенератор (CompositeGen)

Конструктор с изначальными значениями

CompositeGen(std::string = "No name", std::string ="1", int = 0)

1. Класс ГенераторСПостояннымШагом (ConststepGen)

Конструктор с изначальными значениями

ConststepGen(std::string = "No name", std::string = "1", int = 1)

1. Класс СлучайныйГенератор (RandomGen)

Конструктор с изначальными значениями

RandomGen(std::string = "No name", std::string = "1")

1. Абстрактный класс (RrototypeGen.h)

Конструктор с изначальными значениями

PrototypeGen(std::string = "No name", std::string = 1);

**Проектирование методов класса.**

1. Класс CоставнойГенератор (CompositeGen)

|  |  |  |
| --- | --- | --- |
| **Функция:** | **Список параметров:** | **Тип возвращаемого значения:** |
| PushRG | Const string, int | Void |
| NextCG | String, int, int | Void |
| DeleteGen | string | Void |
| GetNum | - | int |
| Average | - | Int |

1. Класс ГенераторСПостояннымШагом (ConststepGen)

|  |  |  |
| --- | --- | --- |
| **Функция:** | **Список параметров:** | **Тип возвращаемого значения:** |
| SetStep | int | Void |
| GetStep | - | int |
| PushNum | - | int |

1. Класс СлучайныйГенератор (RandomGen)

|  |  |  |
| --- | --- | --- |
| **Функция:** | **Список параметров:** | **Тип возвращаемого значения:** |
| PushNum | string | Void |

1. Абстрактный класс (RrototypeGen.h)

|  |  |  |
| --- | --- | --- |
| **Функция:** | **Список параметров:** | **Тип возвращаемого значения:** |
| SetName | string | Void |
| SetN | String | Void |
| SetPrev | Int | Void |
| GetPrev | - | Int |
| Getsum | - | Int |
| Average | - | Virtual int |

**Обоснование выбора типа коллекции.**

List доступ к случайному элементу не требуется,последовательный доступ не усложняет использование в виду использования перебора для получения суммы,достаточно доступа к последнему и предыдущим, через перебор до N-ого элемента с конца

Map состоит из пары ключ-значение, имеет быстрый поиск по ключу,что подходит под задачу

Forward\_list доступ к случайному элементу не требуется,последовательный доступ не усложняет использование в виду использования перебора для получения суммы

**Перечень исключительных ситуаций.**

invalid\_argument используется при неудовлетворении аргумента определённым критериям

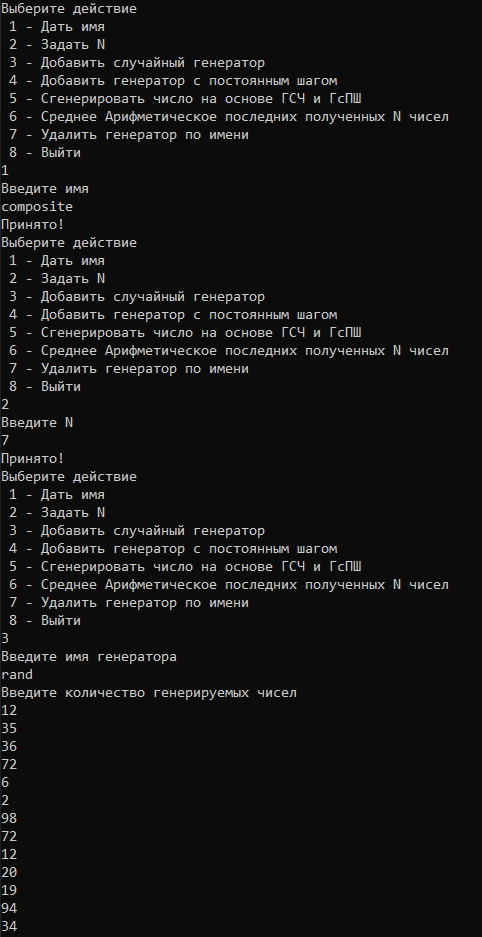
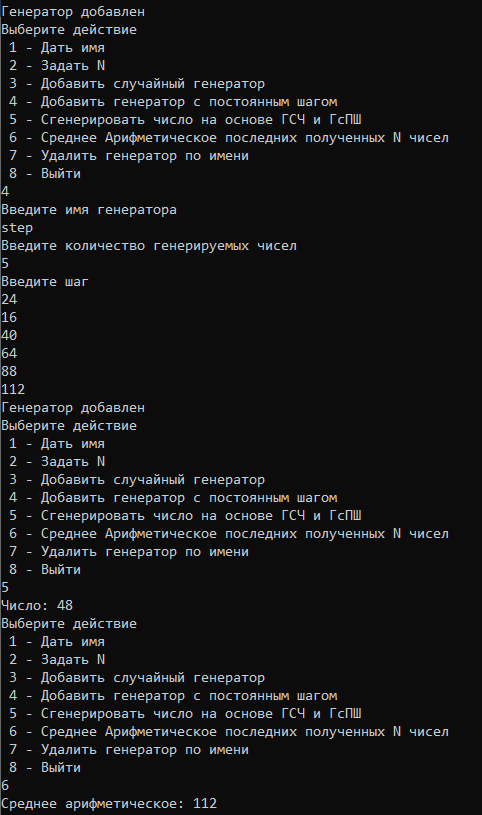
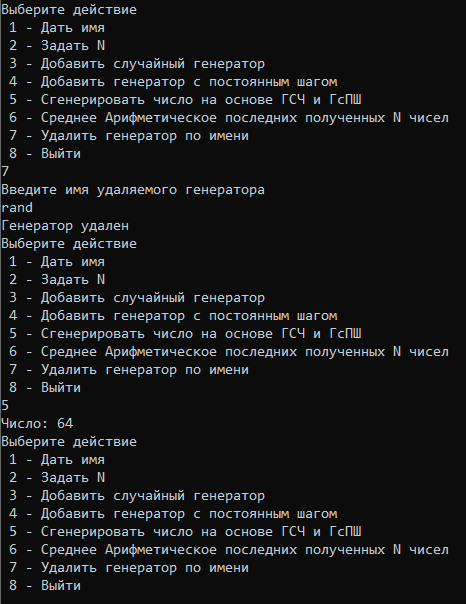
Out\_of\_range используется при попытке обратиться к несуществующим элементам списка

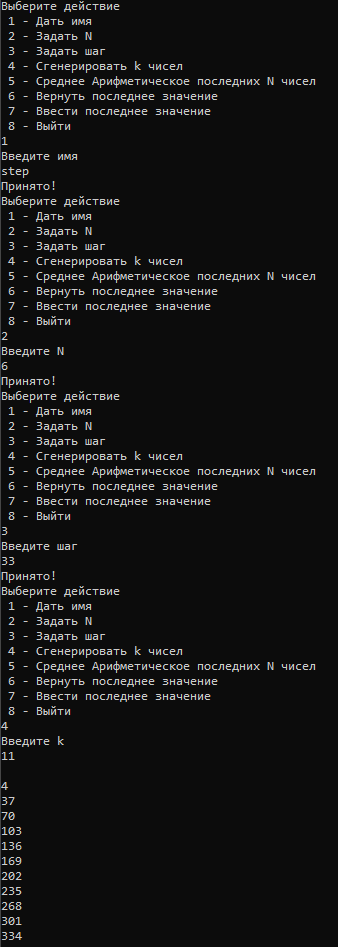
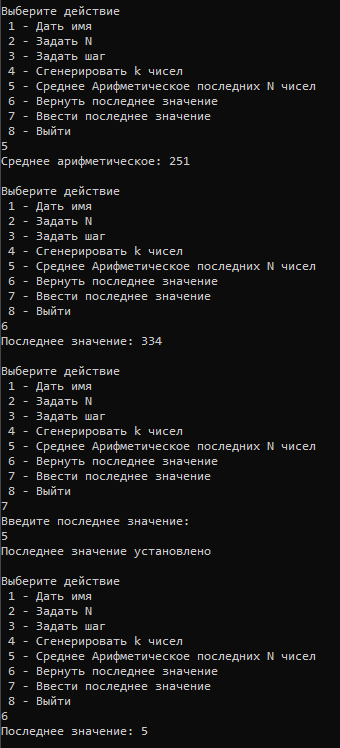
Описаные условия являются следствием неправильной логики в рамках программы

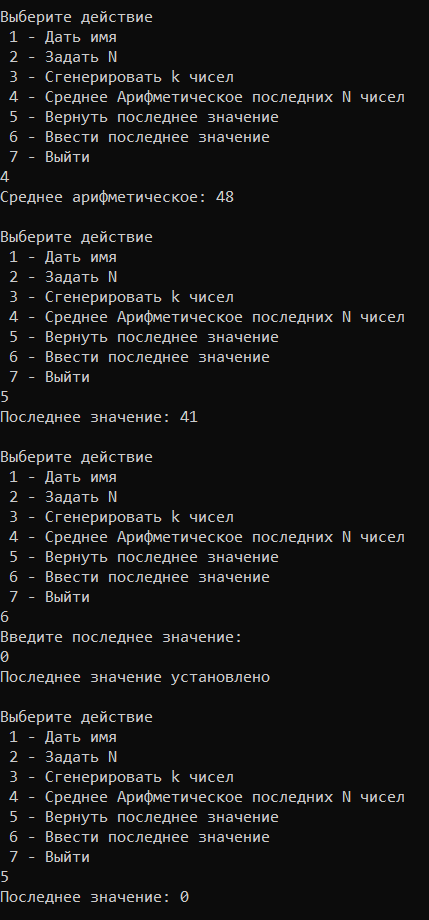
**Проектирование главной функции main.**

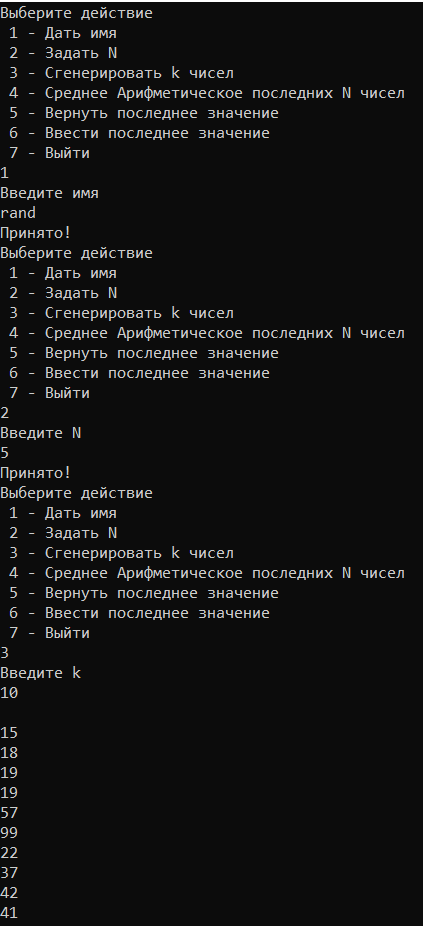
Пользователю предлагают работу с одним из предложенных генераторов. После выбора пользователя, программа предлагает варианты взаимодействия с выбранным генератором. Есть общие варианты взаимодействия (“Дать имя”, ”Задать N”,”Среднее Арифметическое последних N чисел”, …) с генераторами, так и действия свойственные только определенным генераторам.(“Задать шаг” в генераторе ConststepGen)

**Тестирование программы.**

1. ****Класс CоставнойГенератор (CompositeGen)
2. Класс ГенераторСПостояннымШагом (ConststepGen)

1. ****Класс СлучайныйГенератор (RandomGen)



**Вывод.**

В данной лабораторной работе были углублены знания, полученные в ходе лекционных и практических занятий. Получены навыки самостоятельной работы по формализации поставленной задачи, программированию, тестированию и отладке задач на ЭВМ. Получены новые навыки при разработке объектно-ориентированных программ. Разработана универсальная программа.

Проект отвечает тем требованиям, которые были предъявлены к нему в ходе разработки.

Обработка исключений делает код более читабельным и элегантным. Поэтому актуальной является необходимость умения грамотно использовать конструкции обработки.

**Листинг программы с комментариями**

|  |
| --- |
| **Source.cpp** |
| #include "RandomGen.h"  #include "ConststepGen.h"  #include "CompositeGen.h"  void Menu() {  int Gen, action, k, step, prev;  std::string N;  std::string name;  RandomGen Rand;  ConststepGen Const;  CompositeGen Comp;  std::cout << "Генераторы случайных чисел\nВыберите генератор : \n";  std::cout << "(1) Случайный Генератор\n(2) ГенераторСПостояннымШагом\n(3) СоставнойГенератор\n";  std::cin >> Gen;  system("cls");  switch (Gen) {  case 1:  while (1) {  std::cout << "Выберите действие\n 1 – Дать имя\n 2 – Задать N\n 3 – Сгенерировать k чисел\n 4 – Среднее Арифметическое последних N чисел\n";  std::cout << " 5 – Вернуть последнее значение\n 6 – Ввести последнее значение\n 7 – Выйти\n";  std::cin >> action;  switch (action) {  default:  std::cout << "Ошибка\n" << std::endl;  break;  case 1:  std::cout << "Введите имя\n";  std::cin >> name;  try {  Rand.SetName(name);  std::cout << "Принято!\n";  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 2:  std::cout << "Введите N\n";  std::cin >> N;  try {  Rand.SetN(N);    }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Принято!\n";  break;  case 3:  std::cout << "Введите k\n";  std::cin >> k;  if (k < 1) {  std::cout << "k должно быть больше 1\n";  break;  }  std::cout << std::endl;  for (int i = 0; i < k; i++)  std::cout << Rand.PushNum() << std::endl;  std::cout << std::endl;  break;  case 4:  try {  Rand.Average();  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Среднее арифметическое: ";  std::cout << Rand.Average() << std::endl << std::endl;  break;  case 5:  std::cout << "Последнее значение: ";  std::cout << Rand.GetPrev() << std::endl << std::endl;  break;  case 6:  std::cout << "Введите последнее значение:\n";  std::cin >> prev;    try {  Rand.SetPrev(prev);  std::cout << "Последнее значение установлено\n" << std::endl;  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 7:  return;  }  }  break;  case 2:  while (1) {  std::cout << "Выберите действие\n 1 – Дать имя\n 2 – Задать N\n 3 – Задать шаг\n 4 – Сгенерировать k чисел\n 5 – Среднее Арифметическое последних N чисел\n";  std::cout << " 6 – Вернуть последнее значение\n 7 – Ввести последнее значение\n 8 – Выйти\n";  std::cin >> action;  switch (action) {  default:  std::cout << "Ошибка\n" << std::endl;  break;  case 1:  std::cout << "Введите имя\n";  std::cin >> name;  try {  Const.SetName(name);  std::cout << "Принято!\n";  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 2:  std::cout << "Введите N\n";  std::cin >> N;  try {  Const.SetN(N);    }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Принято!\n";  break;  case 3:  std::cout << "Введите шаг\n";  std::cin >> step;  try {  Const.SetStep(step);  std::cout << "Принято!\n";  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 4:  std::cout << "Введите k\n";  std::cin >> k;  if (k < 1) {  std::cout << "k должно быть больше 1\n";  break;  }  std::cout << std::endl;  for (int i = 0; i < k; i++)  std::cout << Const.PushNum() << std::endl;  std::cout << std::endl;  break;  case 5:  std::cout << "Среднее арифметическое: ";  std::cout << Const.Average() << std::endl << std::endl;  break;  case 6:  std::cout << "Последнее значение: ";  std::cout << Const.GetPrev() << std::endl << std::endl;  break;  case 7:  std::cout << "Введите последнее значение:\n";  std::cin >> prev;  try {  Const.SetPrev(prev);  std::cout << "Последнее значение установлено\n" << std::endl;  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 8:  return;  }  }  break;  case 3:  while (1) {  std::cout << "Выберите действие\n 1 – Дать имя\n 2 – Задать N\n 3 – Добавить случайный генератор\n 4 – Добавить генератор с постоянным шагом\n 5 – Сгенерировать число на основе ГСЧ и ГсПШ\n 6 – Среднее Арифметическое последних полученных N чисел\n";  std::cout << " 7 – Удалить генератор по имени\n 8 – Выйти\n";  std::cin >> action;  switch (action) {  default:  std::cout << "Ошибка!\n" << std::endl;  break;  case 1:  std::cout << "Введите имя\n";  std::cin >> name;  try {  Comp.SetName(name);  std::cout << "Принято!\n";  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  break;  case 2:  std::cout << "Введите N\n";  std::cin >> N;  try {  Comp.SetN(N);    }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Принято!\n";  break;  case 3:  std::cout << "Введите имя генератора\n";  std::cin >> name;  std::cout << "Введите количество генерируемых чисел\n";  std::cin >> k;  try {  Comp.PushRG(name, k);  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Генератор добавлен\n";  break;  case 4:  std::cout << "Введите имя генератора\n";  std::cin >> name;  std::cout << "Введите количество генерируемых чисел\n";  std::cin >> k;  std::cout << "Введите шаг\n";  std::cin >> step;  try {  Comp.PushCG(name, k, step);  }  catch (const std::exception& ex) {  std::cout << ex.what();  return;  }  std::cout << "Генератор добавлен\n";  break;  case 5:  std::cout << "Число: ";  std::cout << Comp.GetNum() << std::endl;  break;  case 6:  std::cout << "Среднее арифметическое: ";  std::cout << Comp.Average() << std::endl;  break;  case 7:  std::cout << "Введите имя удаляемого генератора\n";  std::cin >> name;  Comp.DeleteGen(name);  std::cout << "Генератор удален\n";  break;  case 8:  return;  }  }  break;  default:  std::cout << "Ошибка\n" << std::endl;  }  }  int main() {  setlocale(LC\_ALL, "RU");  system("cls");  Menu();  } |
| **ConststepGen.cpp** |
| #include "ConststepGen.h"  #include "PrototypeGen.h"  ConststepGen::ConststepGen(std::string \_name, std::string \_N, int \_step):PrototypeGen(\_name, \_N) ///конструктор  {  SetStep(\_step);  }  //ConststepGen::ConststepGen(ConststepGen& source)  //{  // step = source.step;  //}  //  //ConststepGen& ConststepGen::operator=(const ConststepGen& source)  //{  // step = source.step;  // return \*this;  //}  //  //ConststepGen::ConststepGen(ConststepGen&& other) noexcept  // :step(std::exchange(other.step, NULL)) {}  //  //ConststepGen& ConststepGen::operator=(ConststepGen&& other) noexcept  //{  // std::swap(step, other.step);  // return \*this;  //}  void ConststepGen::SetStep(int \_step)  {  if (\_step == 0)  throw std::invalid\_argument("Шаг не может быть равен 0");  else  step = \_step;  }  int ConststepGen::GetStep()  {  return step;  }  int ConststepGen::PushNum()  {  std::random\_device rd;  std::mt19937 mersenne(rd());  if (numbers.size() == 0)  numbers.push\_back(mersenne() % 100);  else  numbers.push\_back(numbers.back() + step);  return numbers.back();  } |
| **CompositeGen.cpp** |
| #include "CompositeGen.h"  CompositeGen::CompositeGen(std::string \_name,std::string \_N, int \_counter) :PrototypeGen(\_name, \_N)  {  counter = \_counter;  }  int CompositeGen::GetNum()  {  int sum = 0, size = 0;  for (auto it = RGarr.begin(); it != RGarr.end(); it++) {  for (auto iter = (it->second)->numbers.begin(); iter != it->second->numbers.end(); iter++) {  size++;  sum += \*iter;  }  }  for (auto it = CGarr.begin(); it != CGarr.end(); ++it) {  for (auto iter = it->second->numbers.begin(); iter != it->second->numbers.end(); iter++) {  size++;  sum += \*iter;  }  }  return size==0?0:sum / size;  }  void CompositeGen::PushRG( std::string \_name, int \_size)  {  if(\_size<0)  throw std::invalid\_argument("Количество чисел отрицательное");  auto temp =std::make\_shared<RandomGen>() ;  temp.get()->SetName(\_name);  for (int i = 0; i < \_size; i++) {  int t = temp.get()->PushNum();  std::cout << t << std::endl;  Averarr.push\_front(t);  if (counter <= N)  counter++;  else  Averarr.erase\_after(std::next(Averarr.begin(), N));  }  RGarr.insert(std::make\_pair(\_name, temp));  }  void CompositeGen::PushCG(std::string \_name, int \_size,int \_step)  {  if (\_size < 0)  throw std::invalid\_argument("Количество чисел отрицательное");  auto temp = std::make\_shared<ConststepGen>();  temp.get()->SetName(\_name);  temp.get()->SetStep(\_step);  for (int i = 0; i < \_size; i++) {  int t = temp.get()->PushNum();  std::cout << t << std::endl;  Averarr.push\_front(t);  if (counter <= N)  counter++;  else  Averarr.erase\_after(std::next(Averarr.begin(), N));  }  CGarr.insert(std::make\_pair(\_name, temp));  }  void CompositeGen::DeleteGen(std::string \_name)  {  if(RGarr.end() != RGarr.find(\_name))  RGarr.erase(\_name);  if (CGarr.end() != CGarr.find(\_name))  CGarr.erase(\_name);  }  int CompositeGen::Average()  {  int sum = 0,size = N;  if (Shortage == Count\_Generated) {  size = 0;  for (auto iter = Averarr.begin(); iter != Averarr.end(); iter++) {  size++;  sum += \*iter;  }  }  if(Shortage==Less)  for (auto iter = Averarr.begin(); iter!=Averarr.end(); iter++)  sum += \*iter;  return size==0?0:sum / size;  } |
| **RandomGen.cpp** |
| #include "RandomGen.h"  RandomGen::RandomGen(std::string \_name, std::string \_N) :PrototypeGen(\_name,\_N){}  int RandomGen::PushNum() {  std::random\_device rd;  std::mt19937 mersenne(rd());  numbers.push\_back(mersenne() % 100);  return numbers.back();  } |
| **PrototypeGen.cpp** |
| #include "PrototypeGen.h"  #include <locale.h>  PrototypeGen::PrototypeGen(std::string \_name, std::string \_N)  {  SetName(\_name);  SetN(\_N);  }  //PrototypeGen::PrototypeGen(PrototypeGen& source)  //{  // this->name = source.name;  // this->N = source.N;  // this->numbers = source.numbers;  //  //}  //  //PrototypeGen& PrototypeGen::operator=(const PrototypeGen& source)  //{  // if (this != &source) {  // this->name = source.name;  // this->N = source.N;  // this->numbers = source.numbers;  // }  // return \*this;  //}  //  //PrototypeGen::PrototypeGen(PrototypeGen&& other) noexcept :name(other.name), N(other.N), numbers(other.numbers)  //{  // other.name = "";  // other.N = 0;  // other.numbers = {};  //}  //  //PrototypeGen& PrototypeGen::operator=(PrototypeGen&& other) noexcept  //{  // if (this == &other)  // return \*this;  // std::swap(name, other.name);  // std::swap(N, other.N);  // std::swap(numbers, other.numbers);  // return \*this;  //}  void PrototypeGen::SetName(std::string \_name)  {  if (\_name == "")  throw std::invalid\_argument("Имя не задано");  else  this->name = \_name;  }  void PrototypeGen::SetN(std::string \_N) {  for (int i = 0; i < \_N.size(); i++)  if (!(\_N[i] >= '0' || \_N[i] <= '9' || \_N[i] == '-'))  Shortage = NotANumber;  if (Shortage != NotANumber) {  int temp = std::stoi(\_N);  if (temp <= 0)  Shortage = Exception;  else {  N = temp;  if (temp > numbers.size())  Shortage = Count\_Generated;  else  Shortage = Less;  }  }  if(Shortage== NotANumber)  throw std::invalid\_argument("Введено не число");  if (Shortage == Exception)  throw std::invalid\_argument("N меньше 1");  }  int PrototypeGen::Average() {  int sum = 0;  if (Shortage== Count\_Generated)  {  for (auto iter = std::prev(numbers.end(),N) /\*- N\*/; iter != numbers.end(); iter++)  sum += \*iter;  return sum / N;  }  if(Shortage==Less)  {  for (auto iter = numbers.begin(); iter != numbers.end(); iter++)  sum += \*iter;  return sum / numbers.size();  }  else  if (Shortage==Exception)  throw std::invalid\_argument("N не может быть меньше единицы ");  else  throw std::out\_of\_range("Числа не были сгенерированы");  }  void PrototypeGen::SetPrev(int prev) {  if (numbers.size() > 0)  numbers.back() = prev;  else  throw std::out\_of\_range("Числа не были сгенерированы");  }  int PrototypeGen::GetPrev() {  if (numbers.size() > 0)  return numbers.back();  else  throw std::out\_of\_range("Числа не были сгенерированы");  } |
| **ConststepGen.h** |
| #pragma once  #include <string>  #include <iostream>  #include <random>  #include <vector>  #include "PrototypeGen.h"  class ConststepGen : public PrototypeGen  {  public:  friend class CompositeGen;  ConststepGen(std::string = "No name", std::string = "1", int = 1); //конструктор  //ConststepGen(ConststepGen& source); //копирование инициализация  //ConststepGen& operator = (const ConststepGen& source); // присваивание копирование  //ConststepGen(ConststepGen&& other) noexcept;// перемещаем конструктор  //ConststepGen& operator=(ConststepGen&& other) noexcept; // перемещаем присвоение  void SetStep(int);  int GetStep();  int PushNum();  //~ConststepGen(){}  private:  int step;  }; |
| **CompositeGen.h** |
| #pragma once  #include "RandomGen.h"  #include "ConststepGen.h"  #include <unordered\_map>  #include <forward\_list>  class CompositeGen : public PrototypeGen  {  public:  CompositeGen(std::string = "No name",std::string ="1", int = 0);  int GetNum();  void PushRG(const std::string, int);  void PushCG(std::string, int, int);  void DeleteGen(std::string);  int Average();  private:  std::unordered\_map <std::string, std::shared\_ptr<RandomGen> > RGarr;  std:: unordered\_map <std::string, std::shared\_ptr<ConststepGen> > CGarr;  std::forward\_list<int> Averarr;  int counter;  }; |
| **RandomGen.h** |
| #pragma once  #include <random>  #include "PrototypeGen.h"  class RandomGen : public PrototypeGen  {  public:    RandomGen(std::string = "No name", std::string = "1");//конструктор  //RandomGen(RandomGen& source);//копирование инициализация  //RandomGen& operator= (const RandomGen& source);// присваивание копирование  //RandomGen(RandomGen&& other) noexcept;// перемещаем конструктор  //RandomGen& operator=(RandomGen&& other) noexcept; // перемещаем присвоение  int PushNum();  //~RandomGen(){}  }; |
| **PrototypeGen.h** |
| #pragma once  #include <iostream>  #include <list>  #include <string>  enum ShortageOfNumbers { Count\_Generated, Exception,NotANumber,Less };  class PrototypeGen  {  friend class CompositeGen;  public:  PrototypeGen(std::string = "No name", std::string = "1");  //PrototypeGen(PrototypeGen& source);  //PrototypeGen& operator=(const PrototypeGen& source);  //PrototypeGen(PrototypeGen&& other) noexcept;  //PrototypeGen& operator=(PrototypeGen&& other) noexcept;  void SetName(std::string);  void SetN(std::string);  void SetPrev(int);  int GetPrev();  virtual int Average();    protected:  std::string name;  std::list <int> numbers;  int N;  ShortageOfNumbers Shortage;  }; |