Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Отчёт по лабораторной работе № 4

Выполнил:

Студент группы РИС-23-1б

Гордеев Василий Андреевич

Проверила:

Доцент кафедры ИТАС

О. А. Полякова

2024

1. Постановка задачи
   1. Определить пользовательский класс.
   2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
   3. Определить в классе деструктор.
   4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
   5. Перегрузить операцию присваивания.
   6. Перегрузить операции ввода и вывода объектов с помощью потоков.
   7. Определить производный класс.
   8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
   9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

1. Анализ задачи
   1. Базовый класс:
   2. ПАРА\_ЧИСЕЛ (PAIR)
   3. Первое\_число (first) - int
   4. Второе\_число (second) – int
   5. Определить методы проверки на равенство и операцию перемножения полей.
   6. Реализовать операцию вычитания пар по формуле (a,b)-(c,d)=(a-b,c-d)
   7. Создать производный класс ПРОСТАЯ\_ДРОБЬ(RATIONAL), с полями Числитель и Знаменатель. Переопределить операцию вычитания и определить операции сложения и умножения простых дробей.

3. Код

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  template <typename T>  struct Node {  T data;  Node\* next\_ptr = nullptr;  };  template <typename T>  class List {  private:  Node<T>\* head\_node;  Node<T>\* tail\_node;  public:  List() {  this->head\_node = nullptr;  this->tail\_node = nullptr;  }  List(int size\_list) {  this->head\_node = nullptr;  this->tail\_node = nullptr;  for (int i = 0; i < size\_list; i++) {  this->insert\_back(0);  }  }  List(int size\_list, T data) {  this->head\_node = nullptr;  this->tail\_node = nullptr;  for (int i = 0; i < size\_list; i++) {  this->insert\_back(data);  }  }  ~List() {  }  void print\_list() {  if (this->head\_node == nullptr) { cout << " Empty list" << endl; return; }  else {  Node<T>\* curr = this->head\_node;  while (curr != nullptr) {  cout << curr->data << endl;  curr = curr->next\_ptr;  }  }  }  int size() {  int k = 0;  Node<T>\* curr = this->head\_node;  if (this->head\_node == nullptr) { return 0; }  while (curr != 0) {  curr = curr->next\_ptr;  k += 1;  }  return k;  }  void clear() {  while (this->head\_node != nullptr) {  Node<T>\* temp = this->head\_node;  this->head\_node = this->head\_node->next\_ptr;  delete temp;  }  head\_node = nullptr;  tail\_node = nullptr;  }  void insert\_position(T data, int pos) {  Node<T>\* new\_node = new Node<T>;  new\_node->data = data;  if (this->head\_node == nullptr) {  this->head\_node = new\_node;  this->tail\_node = new\_node;  return;  }  else {  if (pos <= 0) {  new\_node->next\_ptr = this->head\_node;  this->head\_node = new\_node;  return;  }  if (pos >= this->size()) {  this->tail\_node->next\_ptr = new\_node;  this->tail\_node = new\_node;  return;  }  Node<T>\* curr = this->head\_node;  for (int i = 0; i < pos - 1; i++) {  curr = curr->next\_ptr;  }  new\_node->next\_ptr = curr->next\_ptr;  curr->next\_ptr = new\_node;  }  }  void insert\_back(T data) {  Node<T>\* new\_node = new Node<T>;  new\_node->data = data;  if (this->head\_node == nullptr) {  this->head\_node = new\_node;  this->tail\_node = new\_node;  }  else {  this->tail\_node->next\_ptr = new\_node;  this->tail\_node = new\_node;  }  }  List<T> operator = (const List& other) {  this->clear();  Node<T>\* temp = other.head\_node;  while (temp != nullptr) {  this->insert\_back(temp->data);  temp = temp->next\_ptr;  }  return \*this;  }  T operator [](int index) {  if (this->head\_node == nullptr) {  cout << " Empty list!" << endl;  return 0;  }  else {  Node<T>\* temp = this->head\_node;  for (int i = 0; (i < index) && (temp->next\_ptr != nullptr); i++) {  temp = temp->next\_ptr;  }  if (temp == nullptr) {  cout << " Error with index!" << endl;  return 0;  }  return temp->data;  }  }  List<T> operator \* (List<T>& other\_list) {  if (other\_list.head\_node == nullptr) { return \*this; }  else { if (this->head\_node == nullptr) { return \*this; } }  Node<T>\* curr1 = this->head\_node;  Node<T>\* curr2 = other\_list.head\_node;  while (curr1 != nullptr && curr2 != nullptr) {  curr1->data \*= curr2->data;  curr1 = curr1->next\_ptr;  curr2 = curr2->next\_ptr;  }  return \*this;  }  List<T> operator \*= (List<T>& other\_list) {  if (other\_list.head\_node == nullptr) { return \*this; }  else { if (this->head\_node == nullptr) { return \*this; } }  Node<T>\* curr1 = this->head\_node;  Node<T>\* curr2 = other\_list.head\_node;  while (curr1 != nullptr && curr2 != nullptr) {  curr1->data \*= curr2->data;  curr1 = curr1->next\_ptr;  curr2 = curr2->next\_ptr;  }  return \*this;  }  }; |

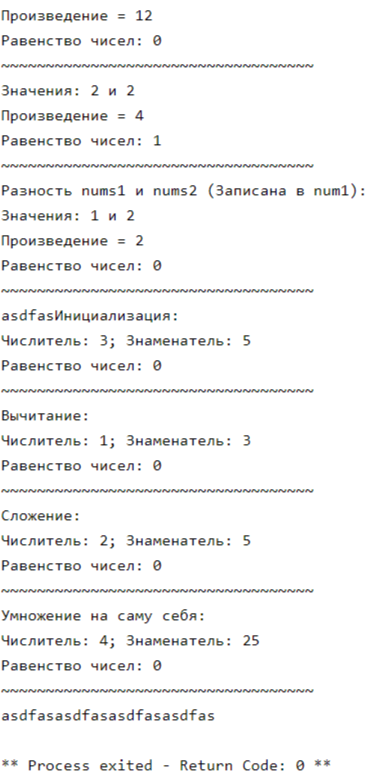
1. Реализация UML

|  |
| --- |
| class Numbers |
|  |
| numbers();  ~numbers();  virtual void print() = 0; |

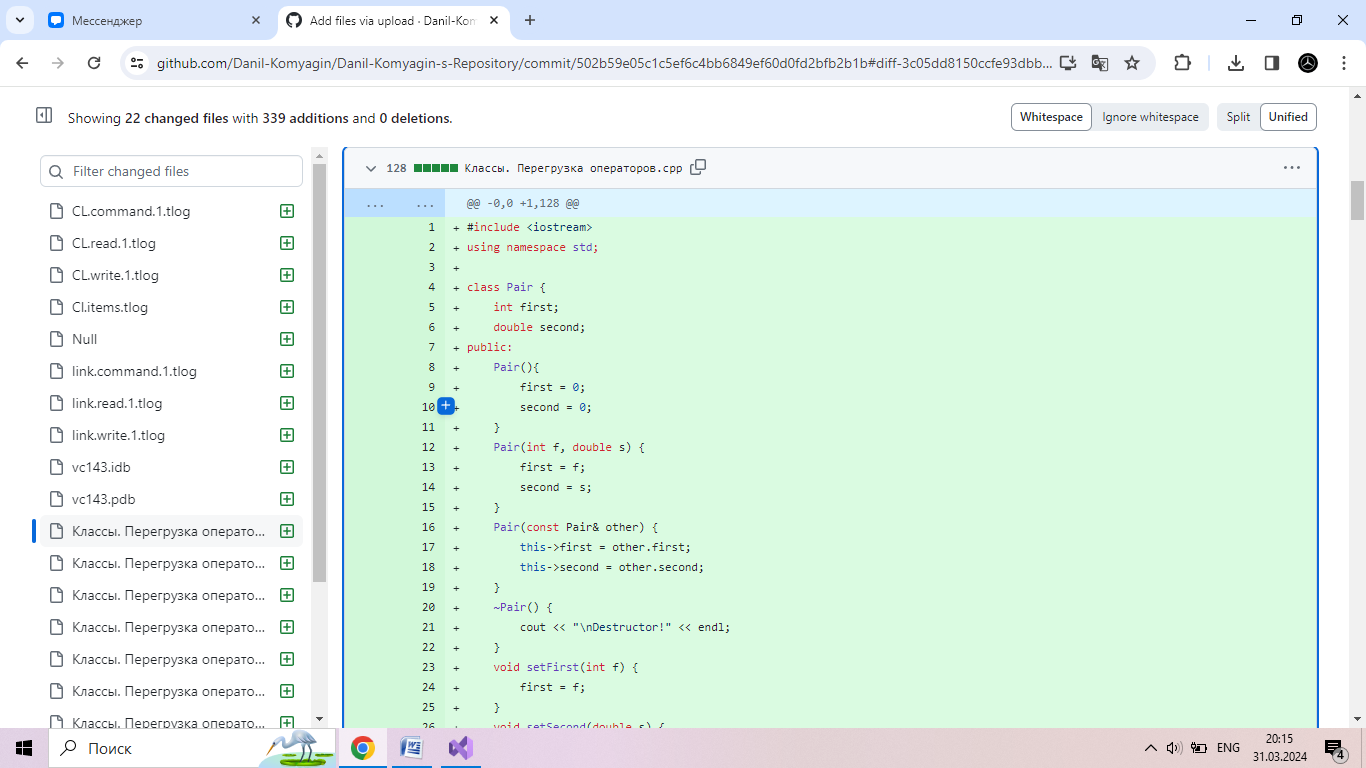
|  |
| --- |
| class Pair : Numbers |
| int first;  int second; |
| Pair();  Pair(int, int);  Pair(Pair&);  int getFirst();  int getSecond();  bool ecualityCheck();  int multiplication();  void print();  Pair& subtract(Pair&); |

|  |
| --- |
| class Rational : public Pair |
|  |
| Rational();  Rational(int, int);  Rational(Pair&);  Rational& subtract(Pair&);  Rational& addition(Pair&);  Rational& multiplication(Pair&);  void print(string); |

1. Консоль:



5. GitHub



Вывод: программа работает корректно и выдаёт требуемый результат.