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

Отчёт по лабораторной работе № 6

Выполнил:

Студент группы РИС-23-1б

Гордеев Василий Андреевич

Проверила:

Доцент кафедры ИТАС

О. А. Полякова

2024

1. Постановка задачи
   1. Определить класс-контейнер.
   2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
   3. Перегрузить операции, указанные в варианте.
   4. Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.
   5. Написать тестирующую программу, иллюстрирующую выполнение операций.

1. Анализ задачи
   1. Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу;
   2. int() – определение размера вектора;
   3. \* – пересечение множеств;
   4. -- - переход к предыдущему элементу ( с помощью класса-итератора).

3. Код

|  |
| --- |
| #pragma once  #include "Iterator.h"  #include <iostream>  #include <vector>  using namespace std;  struct Node {  public:  int data;  Node\* next;  Node\* prev;  };  class Vector {  private:  int size;  Node\* head;  Node\* tail;  friend ostream& operator <<(ostream& out, const Vector& list);  friend istream& operator >> (istream& in, const Vector& list);  Iterator begin;  Iterator end;  public:  void push\_back(int data) {  Node\* new\_node = new Node;  new\_node->data = data;  new\_node->next = nullptr;  if (this->head == nullptr)  {  this->head = new\_node;  this->tail = new\_node;  this->begin.elem = this->head;  this->end.elem = this->tail;  }  else  {  tail->next = new\_node;  new\_node->prev = tail;  tail = new\_node;  }  this->size++;  }  void push\_front(int data) {  Node\* new\_node = new Node;  new\_node->data = data;  if (this->head == nullptr)  {  this->head = new\_node;  this->tail = new\_node;  this->size++;  this->begin.elem = this->head;  this->end.elem = this->tail;  }  else  {  head->prev = new\_node;  new\_node->next = head;  head = new\_node;  this->size++;  this->begin.elem = this->head;  }  }  int front() {  return this->head->data;  }  int back() {  return this->tail->data;  }  int pop\_back() {  int temp;  if (this->tail != nullptr) {  Node\* current\_node = this->tail;  tail = current\_node->prev;  temp = current\_node->data;  tail->next = nullptr;  this->size--;  }  return temp;  }  int pop\_front() {  int temp;  if (this->head != nullptr) {  Node\* current\_node = this->head;  head = current\_node->next;  temp = current\_node->data;  head->prev = nullptr;  this->size--;  }  return temp;  }  bool is\_empty() {  return this->size == 0;  }  Vector& operator = (const Vector& list) {  cout << "Operator =" << endl;  if (this == &list) {  return \*this;  }  while (head != nullptr) {  Node\* temp = head;  head = head->next;  delete temp;  }  size = 0;  Node\* current\_node = list.head;  while (current\_node != nullptr) {  push\_back(current\_node->data);  current\_node = current\_node->next;  }  this->begin = list.begin;  this->end = list.end;  return \*this;  }  int& operator [](int index) {  if (index < this->size && index >= 0) {  Node\* current\_node = this->head;  for (int i = 0; i != index; i++) {  current\_node = current\_node->next;  }  return current\_node->data;  }  else {  cerr << "index out of range";  exit(0);  }  }  int operator () () {  return this->size;  }  Vector operator \* (Vector& list) {  int temp\_size;  if (this->size > list.size) {  temp\_size = list.size;  }  else  {  temp\_size = this->size;  }  Vector temp(temp\_size, 0);  for (int i = 0; i < temp\_size; i++) {  temp[i] = (\*this)[i] \* list[i];  }  return temp;  }  Iterator first() {  return this->begin;  }  Iterator last() {  return this->end;  }  Vector(int size, int data) {  this->size = size;  if (size > 0) {  Node\* node = new Node;  node->data = data;  this->head = node;  this->tail = node;  for (int i = 1; i < size; i++) {  Node\* newNode = new Node;  newNode->data = data;  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  }  tail->next = nullptr;  }  else  {  this->head = nullptr;  this->tail = nullptr;  }  this->begin.elem = this->head;  this->end.elem = this->tail;  }  Vector(const Vector& list)  {  this->head = nullptr;  this->tail = nullptr;  this->size = 0;  Node\* current\_node = list.head;  while (current\_node != nullptr) {  push\_back(current\_node->data);  current\_node = current\_node->next;  }  this->begin.elem = this->head;  this->end.elem = this->tail;  };  Vector(int size) {  this->size = size;  if (size > 0) {  Node\* node = new Node;  this->head = node;  this->tail = node;  for (int i = 1; i < size; i++) {  Node\* newNode = new Node;  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  }  tail->next = nullptr;  }  else  {  this->head = nullptr;  this->tail = nullptr;  }  this->begin.elem = this->head;  this->end.elem = this->tail;  };  ~Vector()  {  Node\* current\_node = head;  while (current\_node != nullptr) {  Node\* next = current\_node->next;  delete current\_node;  current\_node = next;  }  head = nullptr;  };  };  ostream& operator<<(ostream& out, const Vector& list) {  out << endl << "Elements of list" << endl;  Node\* current\_node = list.head;  while (current\_node != nullptr)  {  out << current\_node->data << " ";  current\_node = current\_node->next;  }  out << endl << "conclusion end" << endl;  return out;  }  istream& operator >> (istream& in, const Vector& list) {  cout << endl << "Enter element of list" << endl;  Node\* current\_node = list.head;  while (current\_node != nullptr) {  in >> current\_node->data;  current\_node = current\_node->next;  }  cout << endl << "Enter element stop" << endl;  return in;  } |

1. Реализация UML

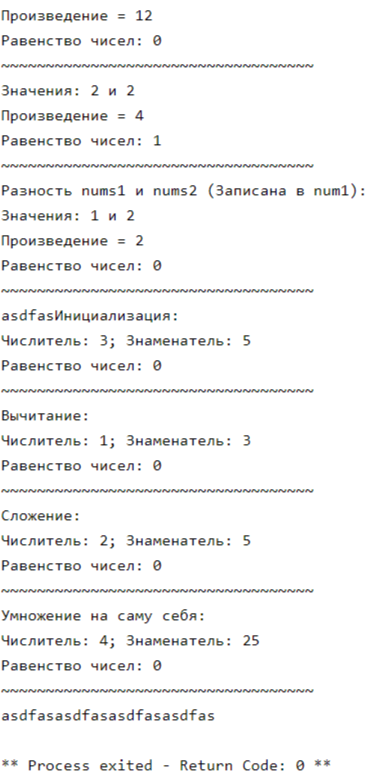
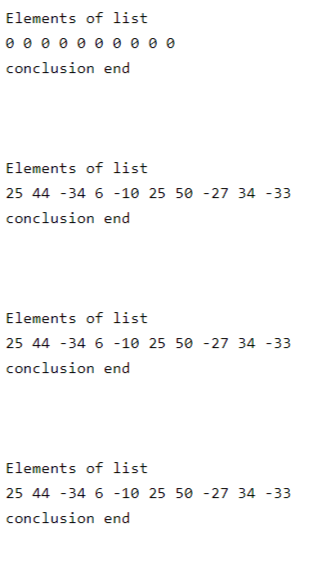
|  |
| --- |
| class Vector |
| int size;  Node\* head;  Node\* tail; |
| friend ostream& operator << (ostream& out, const Vector& list);  friend istream& operator >> (istream& in, const Vector& list);  Iterator begin;  Iterator end;  void push\_back(int data)  void push\_front(int data)  int front()  int back()  int pop\_back()  int pop\_front()  bool is\_empty()  int& operator [](int index)  int operator () ()  Vector operator \* (Vector& list)  Iterator first()  Iterator last()  Vector(int size, int data)  Vector(const Vector& list)  Vector(int size)  ~Vector() |

|  |
| --- |
| class Iterator |
| friend class Vector;  Node\* elem; |
| Iterator()  Iterator(const Iterator& iterator)  Iterator& operator = (const Iterator& iterator)  bool operator ==(const Iterator& iterator)  bool operator !=(const Iterator& iterator)  Iterator& operator++()  Iterator& operator--()  Iterator& operator+(const int number)const  Iterator& operator-(const int number)const  int& operator\* () |

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

|  |
| --- |
| class Vector |
| size\_t \_size; // Текущий размер вектора  size\_t \_capacity; // Емкость вектора (количество выделенной памяти)  T\* \_data; |
| Vector()  ~Vector()  void push\_back(const T& value)  T& operator[](size\_t index)  size\_t size() const |

1. Консоль:



Вывод: программа работает корректно и выдаёт требуемый результат.