Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

(МТУСИ)



КАФЕДРА: Информационная безопасность

по дисциплине

«Программирование в системах информационной безопасности»

Лабораторная работа №12

Одиночное и множественное наследование. Виртуальные и чисто виртуальные функции. Абстрактные и конкретные классы

Выполнил:

студент группы БСУ1801 Убушуев Б.С

Проверил:

старший преподаватель кафедры ИБ

Барков В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2021

### ЗАДАНИЕ

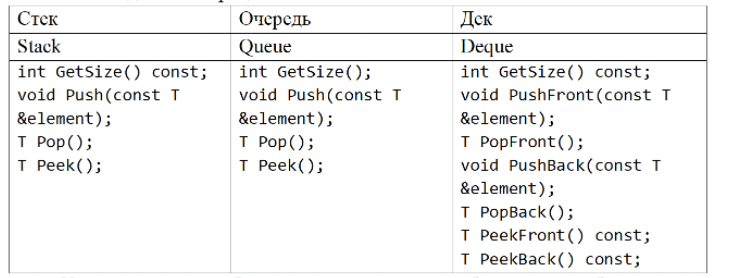
По номеру Вашего варианта выбрать задачу, решаемую в этой лабораторной работе, и выполнить для нее следующие задания.

**ИНДИВИДУАЛЬНЫЙ ВАРИАНТ ЗАДАНИЯ №25(10)**

Таблица 1 – Индивидуальные варианты заданий.

|  |
| --- |
|  |

#### ЗАДАНИЕ 1

Для динамической структуры данных, разработанной в предыдущей лабораторной работе (стек, очередь или дек) создать абстрактный класс, выделив в него необходимые операции.

Унаследовать разработанный в предыдущей лабораторной работе класс от созданного в этой работе абстрактного класса.

Разработать ещё одну реализацию динамической структуры данных, указанной в индивидуальном задании. Разработать соответствующий класс, унаследовав его от абстрактного класса и определив все требуемые операции. Предусмотреть конструкторы инициализации, копирования, перемещения, деструктор, функции вставки и удаления элемента, просмотра доступного элемента и функцию, проверяющую наличие элементов.

Перегрузить операции присваивания, перемещения и потокового вывода для вывода содержимого динамической структуры на экран.

Класс разработать в варианте шаблона.

Создать функцию, получающую указатель на базовый класс и демонстрирующую работу

Создать функцию, получающую ссылку на базовый класс и демонстрирующую работу.

Листинг 1 – Исходный код queue.h

|  |
| --- |
| #pragma once  template <typename T>  class DequeBase  {  public:  virtual int GetSize() const = 0;  virtual void PushFront(const T& element) = 0;  virtual T PopFront() = 0;  virtual void PushBack(const T& element) = 0;  virtual T PopBack() = 0;  virtual T PeekFront() const = 0;  virtual T PeekBack() const = 0;  }; |

Листинг 2 – Исходный код queue2.h

|  |
| --- |
| #pragma once  #include "QueueBase.h"  #include "Node.h"  template<typename T>  class Deque1 : public DequeBase<T>  {  private:  List<T>\* list;  int size;  public:  Deque1();  Deque1(const Deque1& other);  Deque1(Deque1&& other);  ~Deque1();  int GetSize() const;  void PushFront(const T& element);  void PushBack(const T& element);  T PopFront();  T PopBack();  T PeekFront() const;  T PeekBack() const;  Deque1& operator=(const Deque1& other);  Deque1& operator=(Deque1&& other);  template<typename T>  friend std::ostream& operator<< (std::ostream& os, const Deque1<T>& obj);  };  template<typename T>  Deque1<T>::Deque1()  {  list = new List<T>();  size = 0;  }  template<typename T>  Deque1<T>::Deque1(const Deque1& other)  {  list = new List<T>();  size = other.size;  for (int i = 0; i < other.size; i++)  {  this->list->push\_back(other.list->sel\_el(i));  }  }  template<typename T>  Deque1<T>::Deque1(Deque1&& other)  {  size = other.size;  list = other.list;  other.size = NULL;  other.list = nullptr;  }  template<typename T>  Deque1<T>::~Deque1() {  delete list;  size = 0;  }  template<typename T>  int Deque1<T>::GetSize() const { return size; }  template<typename T>  void Deque1<T>::PushFront(const T& element)  {  list->push\_front(element);  ++size;  }  template<typename T>  void Deque1<T>::PushBack(const T& element)  {  list->push\_back(element);  ++size;  }  template<typename T>  T Deque1<T>::PopFront()  {  --size;  return list->pop\_front();  }  template<typename T>  T Deque1<T>::PopBack()  {  --size;  return list->pop\_back();  }  template<typename T>  T Deque1<T>::PeekFront() const { return list->sel\_el(0); }  template<typename T>  T Deque1<T>::PeekBack() const { return list->sel\_el(size - 1); }  template<typename T>  Deque1<T>& Deque1<T>::operator=(const Deque1<T>& other)  {  if (this == &other)  {  return \*this;  }  list->clear();  size = other.size;  for (int i = 0; i < other.size; i++)  {  this->list->push\_back(other.list->sel\_el(i));  }  }  template<typename T>  Deque1<T>& Deque1<T>::operator=(Deque1<T>&& other)  {  if (this == &other)  {  return \*this;  }  list.clear();  size = other.size;  list = other.list;  other.size = NULL;  other.list = nullptr;  }  template<typename T>  std::ostream& operator<<(std::ostream& os, const Deque1<T>& obj)  {  for (size\_t i = 0; i < obj.size; i++)  {  os << obj.list->sel\_el(i) << " ";  }  os << std::endl;  return os;  } |

Листинг 3 – Исходный код Node.h

|  |
| --- |
| #pragma once  #include <iostream>  template<typename T>  class List  {  private:  class Node  {  public:  Node(T data = T(), Node\* pNext = nullptr);  Node\* pNext;  T data;  };  int size;  Node\* head;  Node\* tail;  public:  List();  ~List();  void push\_front(T data);  void insert(T data, int index);  void push\_back(T data);  T pop\_front();  T removeAT(int index);  T pop\_back();  void clear();  int getsize() const;  T sel\_el(const int index) const;  };  template<typename T>  List<T>::Node::Node(T data, Node\* pNext)  {  this->data = data;  this->pNext = pNext;  }  template<typename T>  List<T>::List()  {  size = 0;  head = nullptr;  tail = nullptr;  }  template<typename T>  List<T>::~List() { clear(); }  template<typename T>  void List<T>::push\_front(T data)  {  if (head == nullptr)  {  head = tail = new Node(data);  }  else  {  head = new Node(data, head);  }  size++;  }  template<typename T>  void List<T>::insert(T data, int index)  {  if (index > this->size - 1) {  std::cout << "Указанный индекс находится вне границ!" << std::endl;  return T();  }  else if (index == 0)  {  push\_front(data);  }  else  {  Node\* previous = this->head;  for (int i = 0; i < index - 1; i++)  {  previous = previous->pNext;  }  Node\* newNode = new Node(data, previous->pNext);  previous->pNext = newNode;  size++;  }  }  template<typename T>  void List<T>::push\_back(T data)  {  if (head == nullptr) {  head = tail = new Node(data);  }  else  {  Node\* current = this->tail;  current->pNext = new Node(data);  tail = current->pNext;  }  size++;  }  template<typename T>  T List<T>::pop\_front()  {  Node\* temp = head;  T data = temp->data;  head = head->pNext;  delete temp;  size--;  return data;  }  template<typename T>  T List<T>::removeAT(int index)  {  if (index > this->size - 1)  {  std::cout << "Указанный индекс находится вне границ!" << std::endl;  return T();  }  else if (index == 0)  {  return pop\_front();  }  else  {  Node\* previous = this->head;  for (int i = 0; i < index - 1; i++)  {  previous = previous->pNext;  }  Node\* toDelete = previous->pNext;  if (toDelete->pNext == nullptr)  tail = previous;  previous->pNext = toDelete->pNext;  T data = toDelete->data;  delete toDelete; size--;  return data;  }  }  template<typename T>  T List<T>::pop\_back() {  return removeAT(size - 1);  }  template<typename T>  void List<T>::clear()  {  while (size) {  pop\_front();  }  }  template<typename T>  int List<T>::getsize() const { return size; }  template<typename T>  T List<T>::sel\_el(const int index) const  {  if (index > this->size - 1)  {  std::cout << "Указанный индекс находится вне границ!" << std::endl;  return T();  }  if (index == 0)  return head->data;  else if (index == this->size - 1)  return tail->data;  else  {  Node\* current = head;  for (int i = 0; i < index; i++)  {  current = current->pNext;  }  return current->data;  }  } |
| Листинг 3 – Исходный код Node2.h |
| #pragma once  #include <iostream>  using namespace std;  template <typename T>  struct Elem  {  T data;  Elem\* next, \* prev;  };  template <typename T>  class ListD  {  Elem<T>\* Head;  Elem<T>\* Tail;  int Count;  public:  ListD();  ~ListD();  int GetSize();  void DelAll();  T Del(int pos = 0);  void PushBack(T n);  void PushFront(T n);  void PeekFront() const;  void PeekBack() const;  T PopFront();  T PopBack();  void Print();  void Print(int pos);  T sel\_el(const int index) const;    template <typename T>  friend ostream& operator << (ostream& stream, const ListD&);  };  template <typename T>  ListD<T>::ListD()  {  Head = Tail = NULL;  Count = 0;  }  template <typename T>  ListD<T>::~ListD()  {  DelAll();  }  template <typename T>  void ListD<T>::PushFront(T n)  {  Elem<T>\* temp = new Elem<T>;  temp->prev = 0;  temp->data = n;  temp->next = Head;  if (Head != 0)  Head->prev = temp;  if (Count == 0)  Head = Tail = temp;  else  Head = temp;  Count++;  }  template <typename T>  void ListD<T>::PushBack(T n)  {  Elem<T>\* temp = new Elem<T>;  temp->next = 0;  temp->data = n;  temp->prev = Tail;  if (Tail != 0)  Tail->next = temp;  if (Count == 0)  Head = Tail = temp;  else  Tail = temp;  Count++;  }  template <typename T>  T ListD<T>::Del(int pos)  {  if (pos == 0)  {  cout << "Input position: ";  cin >> pos;  }  if (pos < 1 || pos > Count)  {  cout << "Incorrect position !!!\n";  return T();  }  int i = 1;  Elem<T>\* Del = Head;  while (i < pos)  {  Del = Del->next;  i++;  }  Elem<T>\* PrevDel = Del->prev;  Elem<T>\* AfterDel = Del->next;  if (PrevDel != 0 && Count != 1)  PrevDel->next = AfterDel;  if (AfterDel != 0 && Count != 1)  AfterDel->prev = PrevDel;  if (pos == 1)  Head = AfterDel;  if (pos == Count)  Tail = PrevDel;  T data = Del->data;  delete Del;  Count--;  return data;  }  template <typename T>  void ListD<T>::Print(int pos)  {  if (pos < 1 || pos > Count)  {  cout << "Incorrect position !!!\n";  return;  }  Elem<T>\* temp;  if (pos <= Count / 2)  {  temp = Head;  int i = 1;  while (i < pos)  {  temp = temp->next;  i++;  }  }  else  {  temp = Tail;  int i = 1;  while (i <= Count - pos)  {  temp = temp->prev;  i++;  }  }  cout << pos << " element: ";  cout << temp->data << endl;  }  template <typename T>  void ListD<T>::Print()  {  if (Count != 0)  {  Elem<T>\* temp = Head;  cout << "( ";  while (temp->next != 0)  {  cout << temp->data << ", ";  temp = temp->next;  }  cout << temp->data << " )\n";  }  }  template <typename T>  void ListD<T>::DelAll()  {    while (Count != 0)  Del(1);  }  template <typename T>  int ListD<T>::GetSize()  {  return Count;  }  template <typename T>  void ListD<T>::PeekFront() const  {  Elem<T>\* temp = Head;  cout << temp->data << endl;  }  template <typename T>  void ListD<T>::PeekBack() const  {  Elem<T>\* temp = Tail;  cout << temp->data << endl;  }  template <typename T>  T ListD<T>::PopFront()  {  return Del(1);  }  template <typename T>  T ListD<T>::PopBack()  {  return Del(9);  }  template<typename T>  T ListD<T>::sel\_el(const int index) const  {  int indexmax = Count - 1, indexmin = 0;  indexmax -= index;  indexmin += index;  if (indexmin < indexmax || indexmin == indexmax)  {  int counter = 0;  Elem<T>\* current = Head;  while (current != nullptr)  {  if (counter == index)  {  return current->data;  }  current = current->next;  counter++;  }  }  else if (indexmin > indexmax)  {  int counter = Count - 1; /  Elem<T>\* current = Tail;  while (current != nullptr)  {  if (counter == index)  {  return current->data;  }  current = current->prev;  counter--;  }  }  }  template <typename T>  std::ostream& operator << (std::ostream& stream, const ListD<T>& L)  {  Elem<T>\* temp = L.Head;  while (temp != NULL)  {  stream << temp->data << "";  temp = temp->next;  }  return stream << endl;  } | |

#### 

Рисунок 1 –Выполнение lab6.exe

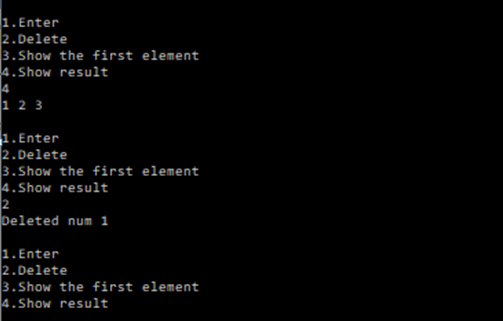
****

Рисунок 2 –Выполнение lab6.exe

**ВЫВОД**

В ходе данной работы получен навык создания базовых классов с виртуальными и чисто виртуальными функциями, а также производных классов с переопределением указанных функций; создания абстрактного класса, реализации динамической структуры данных, перегрузки операции присваивания, перемещения и потокового вывода для вывода содержимого динамической структуры на экран.