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»

Кафедра «Информационная безопасность»

Лабораторная №5

Выполнила:

студентка группы БП31801

Зелик М.Б.

Вариант №10

Москва, 2021

# Цель работы

# Изучить динамические структуры данных, овладеть навыками создания конструкторов копирования, перемещения, деструкторов, перегрузки операций копирования и перемещения

# Задание

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

|  |  |  |
| --- | --- | --- |
| Стек | Очередь | Дек |
| int GetSize() const;  void Push(const T &element);  T Pop();  T Peek(); | int GetSize();  void Push(const T &element);  T Pop();  T Peek(); | int GetSize() const;  void PushFront(const T &element);  T PopFront();  void PushBack(const T &element);  T PopBack();  T PeekFront() const;  T PeekBack() const; |

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.

**Индивидуальные варианты заданий**

|  |  |  |
| --- | --- | --- |
| 10 | Очередь на основе однонаправленного списка | QueueBasedOnUnidirectionalLinkedList |

**Выполнение**

Листинг 1 – Исходный код файла Source.cpp

#include <iostream>

#include "QueueBase.h"

#include "Queue1.h"

#include "Queue2.h"

using namespace std;

template<typename T>

void MyFuncWithPointer(QueueBase<T>\* s)

{

cout << s->Peek() << endl;

}

template<typename T>

void MyFuncWithReference(QueueBase<T>& s)

{

cout << s.Peek() << endl;

}

int main()

{

setlocale(LC\_ALL, "rus");

Queue1<int> d1;

d1.Push(2);

d1.Push(1);

d1.Push(10);

cout << d1;

cout << "Первый элемент с начала : " << d1.Peek() << "; Всего элементов : " << d1.GetSize() << endl;

Queue2<int> d2;

d2.Push(2);

d2.Push(1);

cout << d2;

cout << "Первый элемент с начала : " << d2.Peek() << "; Всего элементов : " << d2.GetSize() << endl;

cout << "Вызов функции с аргументом указателя" << endl;

cout << "Элементы из первой очереди" << endl;

MyFuncWithPointer(&d1);

cout << "Элементы из второй очереди" << endl;

MyFuncWithPointer(&d2);

cout << "Вызов функции с аргументом ссылки" << endl;

cout << "Элементы из первой очереди" << endl;

MyFuncWithReference(d1);

cout << "Элементы из второй очереди" << endl;

MyFuncWithReference(d2);

return 0;

}

Листинг 2 – Исходный код файла Queue1.h

#pragma once

#include "QueueBase.h"

#include "ListS.h"

template<typename T>

class Queue1 : public QueueBase<T>

{

private:

List<T>\* list;

int size;

public:

Queue1();

Queue1(const Queue1& other);

Queue1(Queue1&& other);

~Queue1();

int GetSize() const;

void Push(const T& element);

T Pop();

T Peek() const;

Queue1& operator=(const Queue1& other);

Queue1& operator=(Queue1&& other);

template<typename T>

friend std::ostream& operator<< (std::ostream& os, const Queue1<T>& obj);

};

template<typename T>

Queue1<T>::Queue1()

{

list = new List<T>();

size = 0;

}

template<typename T>

Queue1<T>::Queue1(const Queue1& other)

{

list = new List<T>();

size = other.size;

for (int i = 0; i < other.size; i++)

{

this->list->push(other.list->sel\_el(i));

}

}

template<typename T>

Queque1<T>::Queue1(Queue1&& other)

{

size = other.size;

list = other.list;

other.size = NULL;

other.list = nullptr;

}

template<typename T>

Queue1<T>::~Queue1() {

delete list;

size = 0;

}

template<typename T>

int Queue1<T>::GetSize() const { return size; }

template<typename T>

void Queue1<T>::Push(const T& element)

{

list->push(element);

++size;

}

template<typename T>

T Queue1<T>::Pop()

{

--size;

return list->pop();

}

template<typename T>

T Queue1<T>::Peek() const { return list->sel\_el(0); }

template<typename T>

Queue1<T>& Queue1<T>::operator=(const Queue1<T>& other)

{

if (this == &other)

{

return \*this;

}

list->clear();

size = other.size;

for (int i = 0; i < other.size; i++)

{

this->list->push(other.list->sel\_el(i));

}

}

template<typename T>

Queue1<T>& Queue1<T>::operator=(Queue1<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 Queue1<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 файла Queue2.h

#pragma once

#include "QueueBase.h"

#include "ListD.h"

template<typename T>

class Queue2 : public QueueBase<T>

{

private:

ListD<T>\* list2;

int size;

public:

Queue2();

Queue2(const Queue2& other);

Queue2(Queue2&& other);

~Queue2();

int GetSize() const;

void Push(const T& element);

T Pop();

T Peek() const;

Queue2& operator=(const Queue2& other);

Queue2& operator=(Queue2&& other);

template<typename T>

friend std::ostream& operator<< (std::ostream& os, const Queue2<T>& obj);

};

template<typename T>

Queue2<T>::Queue2()

{

list2 = new ListD<T>();

size = 0;

}

template<typename T>

Queue2<T>::Queue2(const Queue2& other)

{

list2 = new ListD<T>();

size = other.size;

for (int i = 0; i < other.size; i++)

{

this->list2->Push(other.list2->sel\_el(i));

}

}

template<typename T>

Queue2<T>::Queue2(Queue2&& other)

{

size = other.size;

list2 = other.list2;

other.size = NULL;

other.list2 = nullptr;

}

template<typename T>

Queue2<T>::~Queue2() {

delete list2;

size = 0;

}

template<typename T>

int Queue2<T>::GetSize() const { return size; }

template<typename T>

void Queue2<T>::Push(const T& element)

{

list2->Push(element);

++size;

}

template<typename T>

T Queue2<T>::Pop()

{

--size;

return list2->Pop();

}

template<typename T>

T Queue2<T>::Peek() const { return list2->sel\_el(0); }

template<typename T>

Queue2<T>& Queue2<T>::operator=(const Queue2<T>& other)

{

if (this == &other)

{

return \*this;

}

list2->clear();

size = other.size;

for (int i = 0; i < other.size; i++)

{

this->list2->push(other.list2->sel\_el(i));

}

}

template<typename T>

Queue2<T>& Queue2<T>::operator=(Queue2<T>&& other)

{

if (this == &other)

{

return \*this;

}

list2.clear();

size = other.size;

list2 = other.list2;

other.size = NULL;

other.list2 = nullptr;

}

template<typename T>

std::ostream& operator<<(std::ostream& os, const Queue2<T>& obj)

{

for (size\_t i = 0; i < obj.size; i++)

{

os << obj.list2->sel\_el(i) << " ";

}

os << std::endl;

return os;

}

Листинг 4 – Исходный код файла QueueBase.h

#pragma once

template <typename T>

class QueueBase

{

public:

virtual int GetSize() const = 0;

virtual T Pop() = 0;

virtual void Push(const T& element) = 0;

virtual T Peek() const = 0;

};

Листинг 5 – Исходный код файла ListD.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 Push(T n);

void Peek() const;

T Pop();

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>::Push(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>::Peek() const

{

Elem<T>\* temp = Head;

cout << temp->data << endl;

}

template <typename T>

T ListD<T>::Pop()

{

return Del(1);

}

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;

}

Листинг 6 – Исходный код файла ListS.h

#pragma once

#include <iostream>

template<typename T>

class List // Linked list

{

private:

class Node // хранилище

{

public:

Node(T data = T(), Node\* pNext = nullptr); // конструктор

Node\* pNext; // переменная адреса

T data; // информ

};

int size; // кол-во эл

Node\* head; //начало строки

Node\* tail; // конец списка

public:

List(); // конструктор

~List(); // деструктор

void insert(T data, int index); // добавление элемента в указанное место

void push(T data); //добавление в конец

T pop(); // удаление первого

T removeAT(int index); // удаление в указанном месте

void clear(); // удаление всех

int getsize() const; // кол-во элементов

T sel\_el(const int index) const; // возвращает элемент по индексу

};

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>::insert(T data, int index)

{

if (index > this->size - 1) //

{

std::cout << "Указанный индекс находится вне границ!" << std::endl;

return T();

}

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(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()

{

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();

}

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>

void List<T>::clear()

{

while (size)

{

pop();

}

}

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;

}

}

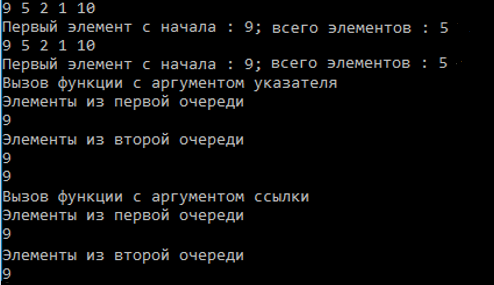


Рисунок 1 – Выполнение программы