Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №1**по дисциплине «Структуры и организация данных»  
на тему «Линейные структуры данных»

Вариант 12

Студент: Костров Г. Ю.  
Группа: О712Б  
   
Преподаватель: Палехова О.А

Санкт-Петербург  
2022 г.

Уровень сложности – **повышенный**. данные хранятся в бинарном файле записей, а для обработки считываются в двусвязный линейный список (если файл не существует, то создается пустой список). При выходе из программы обработанные данные сохраняются в том же файле. Имя файла с данными должно передаваться программе при ее запуске (через параметры функции main()). Если параметры пользователем при запуске программы не заданы, имя файла вводится с клавиатуры. Элементами данных должны являться записи с вариантами (вариативные поля придумать самостоятельно)**.** Обязательные операции: добавление элемента в упорядоченный список с сохранением упорядоченности (ключевое поле выбрать самостоятельно), просмотр списка в прямом и обратном направлении, удаление произвольного элемента списка, поиск в соответствии с индивидуальным вариантом. Вывод данных осуществлять постранично в табличном виде с графлением визуально подходящими символами, предусмотреть листание в прямом и обратном направлении.

Вариант № 12(индивидуальный)

Поля данных: название набора, артикул набора, год выпуска набора, количество минифигурок в наборе, количество деталей в наборе, стоимость набора на момент его выпуска, количество наборов на складе, вариативные поля: 1) возрастная группа набора, 2) ROI\*. Поиск наборов соответствующего года.

\*ROI(return on investment) - окупаемость инвестиций финансовый коэффициент, иллюстрирующий уровень доходности или убыточности бизнеса, учитывая сумму сделанных в этот бизнес инвестиций

Созданные типы данных:

struct LEGO { // Структура данных

char title[30]; // Название набора

int itemNumber; // Артикул набора

int yearRelease; // Год выпуска(упорядочивание)

int minifigures; // Количество минифигурок

int parts; // Количество деталей

int price; // Стоимость набора на момент выхода

int count; // Количество наборов на складе

int type; // Набор относится к линейке Звёздных войн(да или нет)

union {

char age[10]; // Возрастное ограничение набора

int ROI; // Окупаемость инвестиций в набор

} variation;

};

template<class T> // Структура узла

class Node {

public:

T data;

Node\* pPrev, \* pNext;

};

template<class T>

class List {//Структура хранения указателей на первый и последний элемент списка + его размер

private:

int size;

Node<T> \*head;

Node<T> \*tail;

};

Используемые функции:

1. void writeFile(const char\*, List<T> &); //Запись данных в файл

Получает: строку в формате си-строка, ссылку на объект двусвязного списка

Выдает: -

2. void readFile(string, List<T> &); // Чтение данных из файла

Получает: строку с названием файла, ссылку на объект двусвязного списка

Выдает: -

3. void printTable(Node<T> \*); // Вывод таблицы с данными

Получает: указатель на голову или хвост двусвязного списка

Выдает: -

4. void deleteSet(); // удаление элемента из списка

метод класса List

5. void printList(); // Вывод списка с функцией листания

метод класса List

5. void search(); // Поиск определённого элемента в списке

метод класса List

7. void editSetInfo(); // Изменение параметров элемента в списке

метод класса List

8. void addSet(); // Добавление элемента в список

метод класса List

9. void pushBack(T); // Добавление элемента в конец двусвязного списка

Получает: Данные, которые необходимо поместить в конец двусвязного списка

Выдает: -

10. void pushFront(T); // Добавление элемента в конец двусвязного списка

Получает: Данные, которые необходимо поместить в начало двусвязного списка

Выдает: -

11. void popFront(); // Удаление элемента с начала двусвязного списка

12. void popBack(); // Удаление элемента с конца двусвязного списка

13. void clear(); // Очистка двусвязного списка от всех элементов

14. void insert(T); // Добавление элемента в двусвязный список с сохранением упорядоченности

Получает: Данные, которые необходимо поместить в двусвязный список

Выдает: -

15. void remove(Node<T> \*); // Удаление определённого элемента двусвязного списка

Получает: Указатель на «Ноду», которую необходимо удалить

Выдает: -

16. bool isEmpty() // Проверка на пустоту двусвязного списка

Получает: -

Выдает: True(если список пустой) / False(если список не пустой)

17. void setData(Node<T> \*) ; // изменение данных опеределённого элемента двусвязного списка

Получает: Указатель на «Ноду», данные которой необходимо изменить

Выдает: –

Текст программы:

// файл main.h

#include "string"

using namespace std;

// Данные хранящиеся в элементе связного списка

struct LEGO {

char title[30];

int itemNumber;

int yearRelease;

int minifigures;

int parts;

int price;

int count;

int type;

union {

char age[10];

int ROI;

} variation;

};

// Элемент связного списка

template<class T>

class Node {

public:

Node(T data = T(), Node \*pPrev = nullptr, Node \*pNext = nullptr) {

this->data = data;

this->pNext = pNext;

this->pPrev = pPrev;

}

T data;

Node\* pPrev, \* pNext;

};

// Связный список

template<class T>

class List {

public:

List();

~List();

// Методы

void pushBack(T data);

void pushFront(T data);

void popFront();

void popBack();

void clear();

void insert(T value);

void remove(Node<T> \*set);

void sort();

void printTable(Node<T> \*direction);

bool isEmpty() {

if (head == nullptr) {

cout << "There are no entries..." << endl; system("pause");

}

return head == nullptr;

}

// сеттеры

void setData(Node<T> \*set);

// перегрузка оператора [] - получение свойства data по индексу

T& operator[] (const int index);

// взаимодействие с файлами

void writeFile(const char\* fileName, List<T> &lst);

void readFile(string fileName, List<T> &lst);

// взаимодействие с консолью(менюшка)

void deleteSet();

void printList();

void search();

void editSetInfo();

void addSet();

private:

int size;

Node<T> \*head;

Node<T> \*tail;

};

// Файл main.cpp

#include "iostream"

#include "string"

#include "fstream"

#include "iomanip"

#include "main.h"

#include "..\..\helpFunctions\functions.cpp"

using namespace std;

// Конструктор

template<class T>

List<T>::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

// Деструктор

template <class T>

List<T>::~List() {

clear();

}

// Методы

// Добавление элемента в конец списка

template <class T>

void List<T>::pushBack(T data) {

// Если список пустой

if (head == nullptr) {

// Создаём первый элемент

head = tail = new Node<T>(data);

}

else {

// Ищем последний элемент

Node<T> \*pCur = head;

while (pCur->pNext != nullptr) {

pCur = pCur->pNext;

}

// Добавляем новый элемент

pCur->pNext = tail = new Node<T>(data, pCur);

}

size++;

}

// Добавление элемента в начало списка

template <class T>

void List<T>::pushFront(T data) {

// Если список пустой

if (head == nullptr) {

// Создаём первый элемент

head = tail = new Node<T>(data);

}

else {

// Создаём новый элемент

Node<T> \*pNew = new Node<T>(data, nullptr, head);

head->pPrev = pNew;

// Перемещаем указатель на новый элемент

head = pNew;

}

size++;

}

// Удаление первого элемента

template <class T>

void List<T>::popFront() {

Node<T> \*pCur= head;

if (head == nullptr) {

cout << "Empty list..." << endl;

return;

} else if (size == 1) {

head = head->pNext;

tail = nullptr;

} else {

head = head->pNext;

head->pPrev = nullptr;

}

delete pCur;

size--;

}

// Удаление последнего элемента

template <class T>

void List<T>::popBack() {

Node<T> \*pCur = tail;

if (head == nullptr) {

cout << "Empty list..." << endl;

return;

} else if (size == 1) {

head = head->pNext;

tail = nullptr;

} else {

tail = tail->pPrev;

tail->pNext = nullptr;

}

delete pCur;

size--;

}

// Очистка списка

template <class T>

void List<T>::clear() {

while (size) {

popFront();

}

}

// Вставка элемента в список

template <class T>

void List<T>::insert(T value) {

// pCur - указатель на элемент, который стоит на месте индекса

// Мы его обязаны сместить вперед, а на его место поставить новый элемент

Node<T> \*pCur = head;

while(pCur != nullptr ) {

if (value.yearRelease <= pCur->data.yearRelease) {

break;

}

pCur = pCur->pNext;

}

if (pCur == nullptr) { // Если хотим вставить в начало

pushFront(value);

} else if (pCur->pPrev == nullptr) { // Если хотим вставить в начало

pushFront(value);

}else if (pCur->pNext == nullptr) { // Если хотим вставить в конец

pushBack(value);

} else {

// Создаём указатель на новый элемент

Node<T> \*pNew = new Node<T>(value, pCur->pPrev, pCur);

pCur->pPrev->pNext = pNew;

pCur->pPrev = pNew;

}

size++;

}

// Удаление элемента по индексу

template <class T>

void List<T>::remove(Node<T> \*pCur) {

if (pCur->pPrev == nullptr) {

popFront();

} else if (pCur->pNext == nullptr) {

popBack();

} else {

// pCur - указатель на элемент, который стоит на месте индекса

pCur->pPrev->pNext = pCur->pNext;

pCur->pNext->pPrev = pCur->pPrev;

delete pCur;

size--;

}

}

// Сортировка списка

template <class T>

void List<T>::sort() {

Node<T> \*pStart = head;

for (int j = 0; j < size; j++) {

pStart = head;

for(int i = 0; i < j; i++) {

pStart = pStart->pNext;

}

Node<T> \*pMax = pStart;

Node<T> \*pCur = pStart;

while (pCur) {

if (pCur->data.yearRelease >= pMax->data.yearRelease) {

pMax = pCur;

}

pCur = pCur->pNext;

}

pushFront(pMax->data);

remove(pMax);

}

}

// перегрузка оператора [] для доступа к элементам списка

template<class T>

T& List<T>::operator[] (const int index) {

// Ищем нужный элемент

Node<T> \*pCur = head; // временный указатель

for (int i = 0; i < index; i++) {

pCur = pCur->pNext;

}

return pCur->data;

}

// функци для записи данных в бинарный файл

template<class T>

void List<T>::writeFile(const char\* fileName, List<T> &lst) {

ofstream file("newFile", ios::out | ios::binary);

Node<T> \*pCur = lst.head;

while (pCur != nullptr) {

file.write((char\*)&pCur->data, sizeof(T));

pCur = pCur->pNext;

}

file.close();

std::remove(fileName);

rename("newFile", fileName);

}

// функция доя чтения бинарного файла

template<class T>

void List<T>::readFile(string fileName, List<T> &lst) {

ifstream file(fileName, ios::in | ios::binary);

if(file.is\_open()) {

file.seekg(0, file.end);

int length = (int)file.tellg();

file.seekg(0, file.beg);

while (file.tellg() < length) {

T data;

file.read((char\*)&data, sizeof(T));

lst.pushBack(data);

}

}

file.close();

}

// Добавление элемента в список

template<class T>

void List<T>::addSet() {

T newSet;

// Ввод данных

{cout << "Set title:\n"; cin.getline(newSet.title, 30);

cout << "Set number:\n"; newSet.itemNumber = getVariant(100000);

cout << "Set year release:\n"; newSet.yearRelease = getVariant(2022);

cout << "minifigures count:\n"; newSet.minifigures = getVariant(50);

cout << "Parts count:\n"; newSet.parts = getVariant(20000);

cout << "Price:\n"; newSet.price = getVariant(1000000);

cout << "Sets count:\n"; newSet.count = getVariant(10000);

cout << "Is your set released under the Star Wars franchise?\n1- Yes\n2 -No\n>"; newSet.type = getVariant(2);

if (newSet.type == 1) {

cout << "Set ROI of the set\n"; newSet.variation.ROI = getVariant(1000);

} else {

cout << "Set consumer age:\n"; cin.getline(newSet.variation.age, 10);

}}

insert(newSet);

}

// Печать списка на консоль

template<class T>

void List<T>::printList() {

if (isEmpty()) { return;}

int variant = 0;

do {

system("cls");

string menu[] = {

"Choose list visualization:",

"1. Print all sets from the beggining",

"2. Print all sets from the end"};

printMenu(menu, 3);

exitMenu(3);

variant = getVariant(3);

system("cls");

switch (variant) {

case 1:

printTable(head);

break;

case 2:

printTable(tail);

break;

default:

break;

}

} while (variant != 3);

}

// вывод данных в виде таблицы

template<class T>

void List<T>::printTable (Node<T> \*direction) {

int variant = 0;

int N = 0;

int total = 0;

Node<T> \*pCur = direction;

do {

int c = 0;

system("cls");

cout << "| N | Set title | Set Number | Year release | Minifigures | Parts count | Price | Count | ROI/Consumer age |" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl;

while (c != 5 && pCur != nullptr) {

cout << "|" << setw(4) << ++N << " | ";

cout << setw(22) << pCur->data.title << " | ";

cout << setw(10) << pCur->data.itemNumber << " | ";

cout << setw(12) << pCur->data.yearRelease << " | ";

cout << setw(11) << pCur->data.minifigures << " | ";

cout << setw(11) << pCur->data.parts << " | ";

cout << setw(7) << pCur->data.price << " | ";

cout << setw(7) << pCur->data.count << " | " ;

if (pCur->data.type == 1) {

cout << setw(15) << pCur->data.variation.ROI << "% | " << endl;

} else {

cout << setw(16) << pCur->data.variation.age << " | " << endl;

}

if (direction == head) {

pCur = pCur->pNext;

} else {

pCur = pCur->pPrev;

}

c++;

}

total += c;

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl << endl;

string menu[] = {

"1. Forward",

"2. Back"};

printMenu(menu, 2);

exitMenu(3);

variant = getVariant(3);

switch (variant) {

case 1:

if (total == size) {

N = total = 0;

pCur = direction;

break;

}

break;

case 2:

if (total == size) {

N = total =0;

pCur = direction;

break;

}

break;

default:

break;

}

} while(variant != 3);

}

// Изменение данных элемента

template<class T>

void List<T>::editSetInfo () {

if (isEmpty()) { return;}

do {

cout << "Enter set number: ";

int number = getVariant(100000);

Node<T> \*pCur = head;

while (pCur != nullptr) {

if (pCur->data.itemNumber == number) {

setData(pCur);

return;

}

pCur = pCur->pNext;

}

cout << "Set not found" << endl;

cout << "1. exit" << endl;

cout << "2. try again" << endl;

int variant = getVariant(2);

if (variant == 1) {

return;

}

system("cls"); // очищаем экран

} while(true);

}

// Сеттер данных с меню

template<class T>

void List<T>::setData(Node<T> \*pCur) {

int variant = 0;

int N = 0;

do {

system("cls");

// вывод элемента

{

cout << "| N | Set title | Set Number | Year release | Minifigures | Parts count | Price | Count | ROI/Consumer age |" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl;

cout << "|" << setw(4) << ++N << " | ";

cout << setw(22) << pCur->data.title << " | ";

cout << setw(10) << pCur->data.itemNumber << " | ";

cout << setw(12) << pCur->data.yearRelease << " | ";

cout << setw(11) << pCur->data.minifigures << " | ";

cout << setw(11) << pCur->data.parts << " | ";

cout << setw(7) << pCur->data.price << " | ";

cout << setw(7) << pCur->data.count << " | " ;

if (pCur->data.type == 1) {

cout << setw(15) << pCur->data.variation.ROI << "% | " << endl;

} else {

cout << setw(16) << pCur->data.variation.age << " | " << endl;

}

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl;

}

string variation;

if(pCur->data.type == 1) {

string variation = "8. Star Wars variation";

} else {

string variation = "8. ROI variation";

}

string menu[] = {

"Choose set data to change:",

"1. Title",

"2. Set number",

"3. Year release",

"4. Minifig count",

"5. Parts count",

"6. Set price",

"7. Sets count",

variation,

};

printMenu(menu, 9);

exitMenu(9);

variant = getVariant(9);

system("cls");

cout << "Enter new value: ";

switch (variant)

{

case 1:

cin.getline(pCur->data.title, 30); break;

case 2:

pCur->data.itemNumber = getVariant(100000); break;

case 3:

pCur->data.yearRelease = getVariant(2022); break;

case 4:

pCur->data.minifigures = getVariant(50); break;

case 5:

pCur->data.parts = getVariant(20000); break;

case 6:

pCur->data.price = getVariant(1000000); break;

case 7:

pCur->data.count = getVariant(10000); break;

case 8:

if (pCur->data.type == 1) {

pCur->data.variation.ROI = getVariant(1000);

} else {

cin.getline(pCur->data.variation.age, 10);

}

break;

}

} while (variant != 9);

}

// Удаление элемента

template<class T>

void List<T>::deleteSet() {

if (isEmpty()) { return;}

do {

cout << "Enter set number: ";

int number = getVariant(100000);

Node<T> \*pCur = head;

while (pCur != nullptr) {

if (pCur->data.itemNumber == number) {

remove(pCur);

cout << "Set deleted" << endl;

system("pause");

return;

}

pCur = pCur->pNext;

}

cout << "Set not found" << endl;

cout << "1. exit" << endl;

cout << "2. try again" << endl;

int variant = getVariant(2);

if (variant == 1) {

return;

}

system("cls"); // очищаем экран

} while (true);

}

// Поиск элемента

template<class T>

void List<T>::search() {

if (isEmpty()) { return;}

do {

cout << "Enter year release: ";

int year = getVariant(2022);

system("cls"); // очищаем экран

int flag = 1;

int N = 0;

Node<T> \*pCur = head;

while (pCur != nullptr) {

if (pCur->data.yearRelease == year) {

if (flag == 1) {

cout << "| N | Set title | Set Number | Year release | Minifigures | Parts count | Price | Count | ROI/Consumer age |" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl;

}

flag = 0;

cout << "|" << setw(4) << ++N << " | ";

cout << setw(22) << pCur->data.title << " | ";

cout << setw(10) << pCur->data.itemNumber << " | ";

cout << setw(12) << pCur->data.yearRelease << " | ";

cout << setw(11) << pCur->data.minifigures << " | ";

cout << setw(11) << pCur->data.parts << " | ";

cout << setw(7) << pCur->data.price << " | ";

cout << setw(7) << pCur->data.count << " | ";

if (pCur->data.type == 1) {

cout << setw(15) << pCur->data.variation.ROI << "% | " << endl;

} else {

cout << setw(16) << pCur->data.variation.age << " | " << endl;

}

}

pCur = pCur->pNext;

}

if (flag == 0) {

cout << "-------------------------------------------------------------------------------------------------------------------------------" << endl << endl;

system("pause");

return;

}

cout << "Sets not found" << endl;

cout << "1. exit" << endl;

cout << "2. try again" << endl;

int variant = getVariant(2);

if (variant == 1) {

return;

}

system("cls"); // очищаем экран

} while (true);

}

int main(int argc, char\* argv[])

{

// Чтение агрументов командной строки

string fileName;

{

if (argc == 2)

fileName.assign(argv[1]); // получаем имя файла из командной строки

else if (argc == 1) {

cout << "Enter fileName bin\*:\n";

getline(cin, fileName);

} else {

cout << "Too many arguments :(\n";

return 1;

}

}

List<LEGO> lst;

LEGO newSet;

// Чтение из файла

lst.readFile(fileName, lst);

int variant = 0;

do {

system("cls");

string menu[] = {

"1. Add set",

"2. Edit set information",

"3. Print list",

"4. Delete set",

"5. Clear list",

"6. Search"};

printMenu(menu, 6);

exitMenu(7);

variant = getVariant(7);

system("cls");

switch (variant) {

case 1:

lst.addSet(); break;

case 2:

lst.editSetInfo(); break;

case 3:

lst.printList(); break;

case 4:

lst.deleteSet(); break;

case 5:

lst.clear(); break;

case 6:

lst.search(); break;

default:

break;

}

} while (variant != 7);

// Запись в файл

lst.writeFile(fileName.c\_str(), lst);

return 0;

}

Результаты работы программы:



Рисунок 1 – ввод названия файла

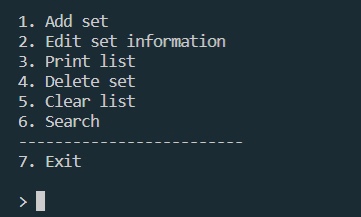


Рисунок 2 – основное меню программы

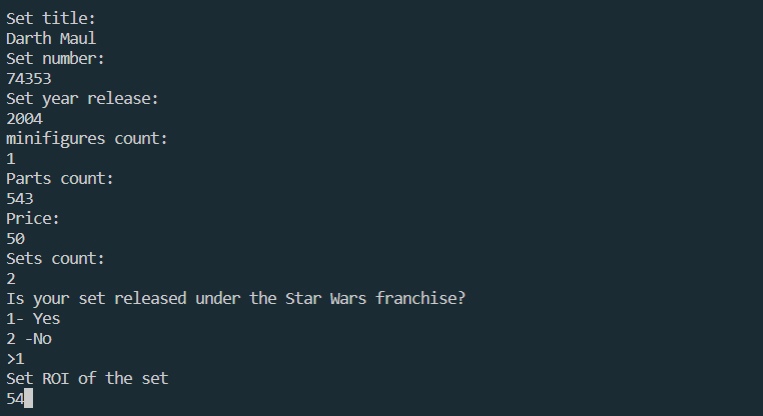


Рисунок 3 – добавление элемента в список



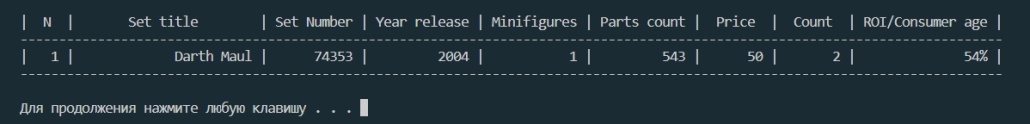


Рисунок 4, 5 – Поиск элементов в списке

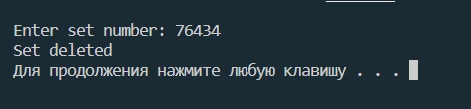


Рисунок 5 – Удаление элемента из списка

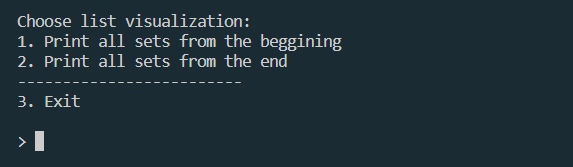


Рисунок 6 – Выбор вывода данных в виде таблицы

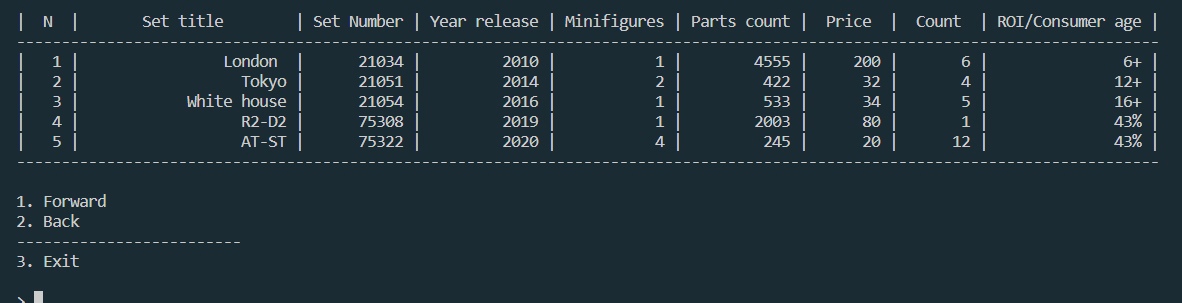


Рисунок 7 – Вывод данных в виде таблицы

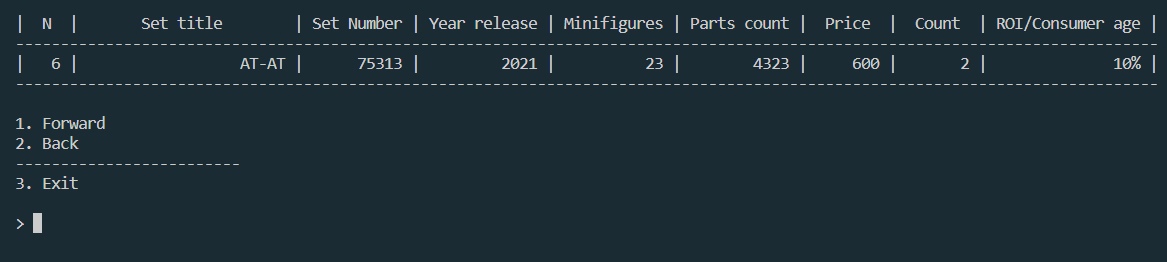


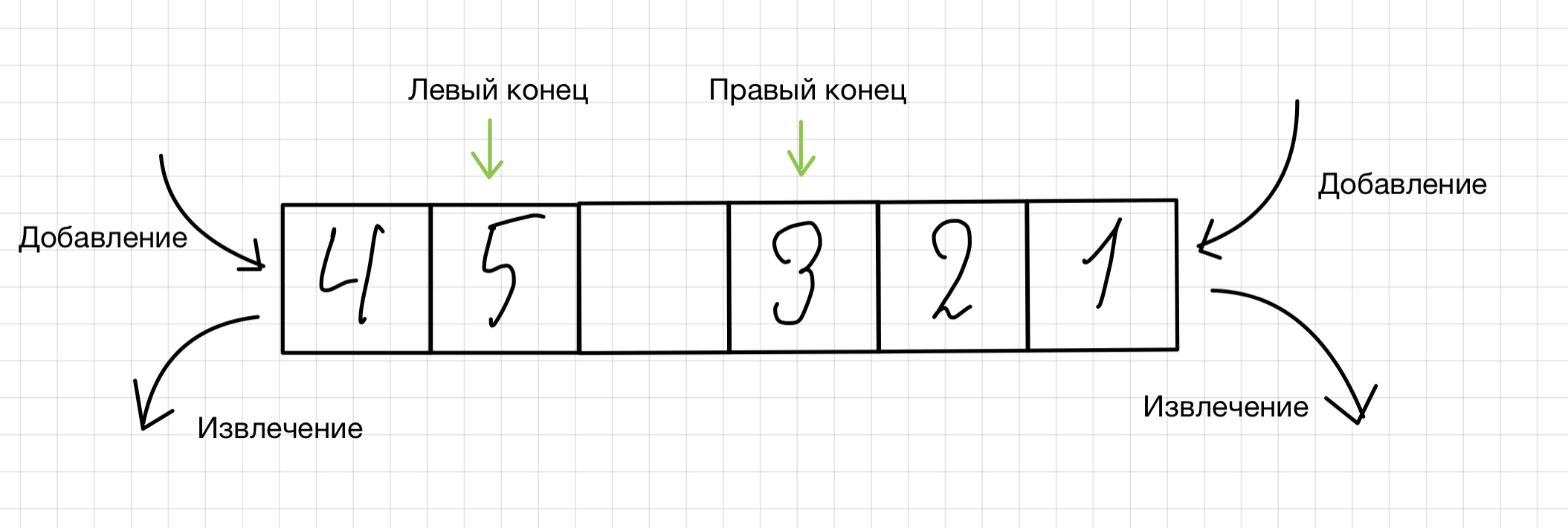
Рисунок 8 – Листание таблицы

1. Структура данных – дек.

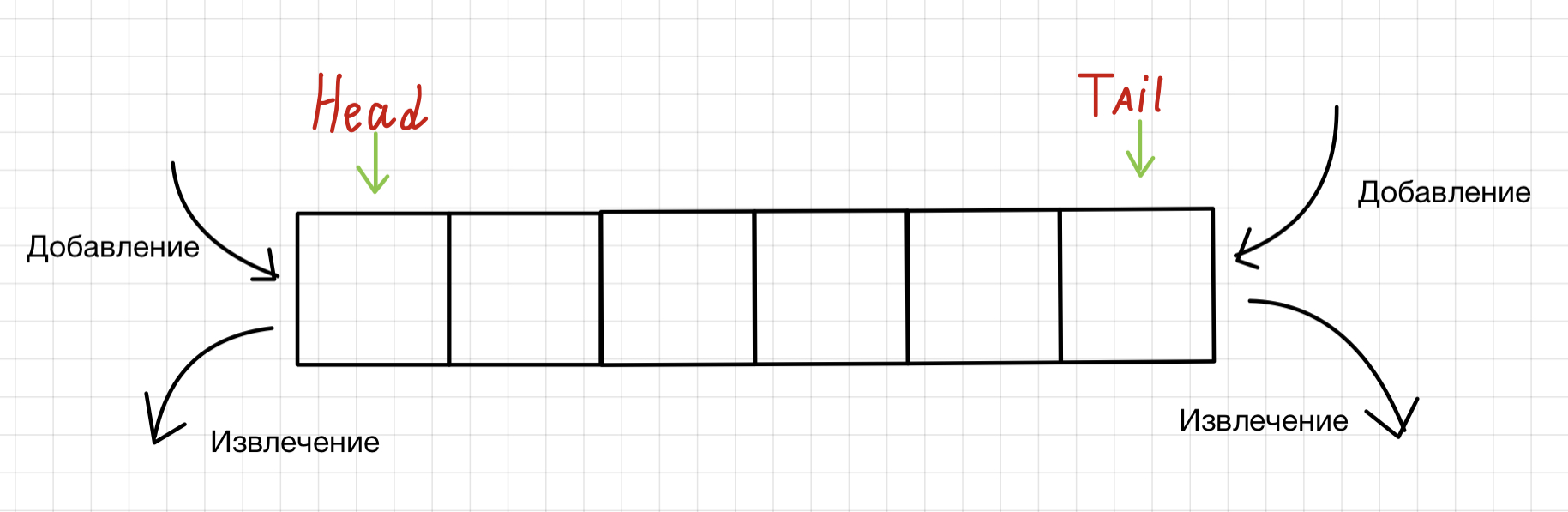
Уровень сложности – **повышенный**. Реализовать необходимую для решения задачи структуру данных с помощью двух структур хранения: векторной и связной,– реализацию оформить в виде иерархии классов. Абстрактный базовый класс должен задавать интерфейс, а производные – реализацию структуры данных. В программе, решающей поставленную задачу, выбор структуры хранения предоставить пользователю.

Структура данных – дек. Элементами дека являются натуральные числа. Удалите из дека элементы, являющиеся простыми числами. Расположите оставшиеся элементы в порядке убывания. В качестве вспомогательной структуры данных можно и нужно использовать второй дек. Сортировку производить, использую только стандартные операции работы с деком.

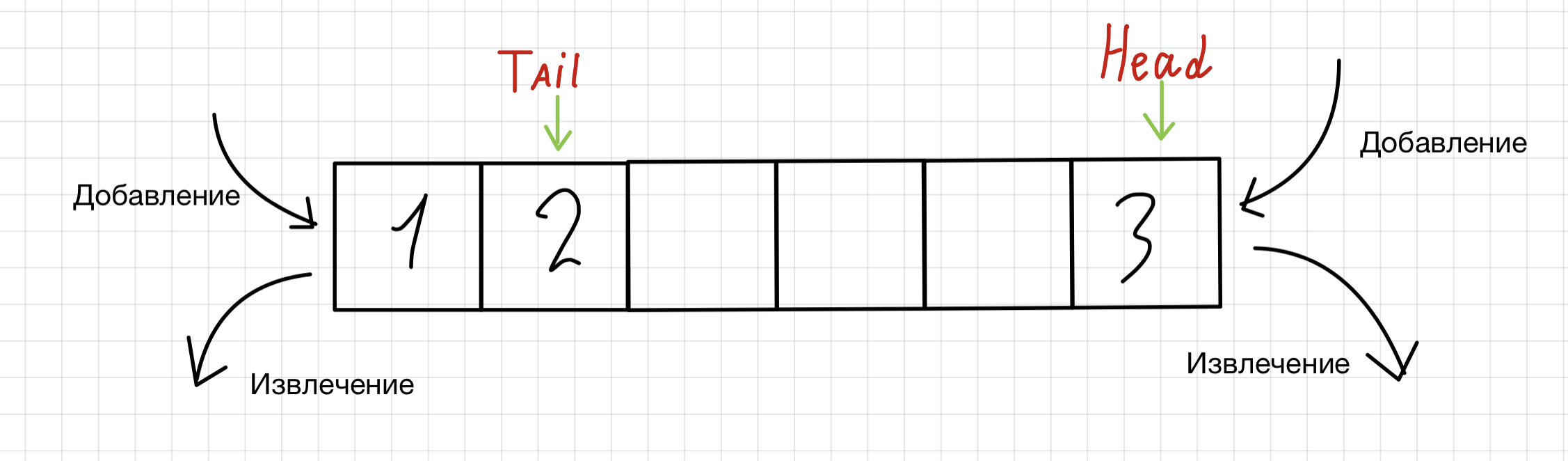
Схематичное изображение структуры данных:

Схематичное изображение структуры хранения, использованной в программе:

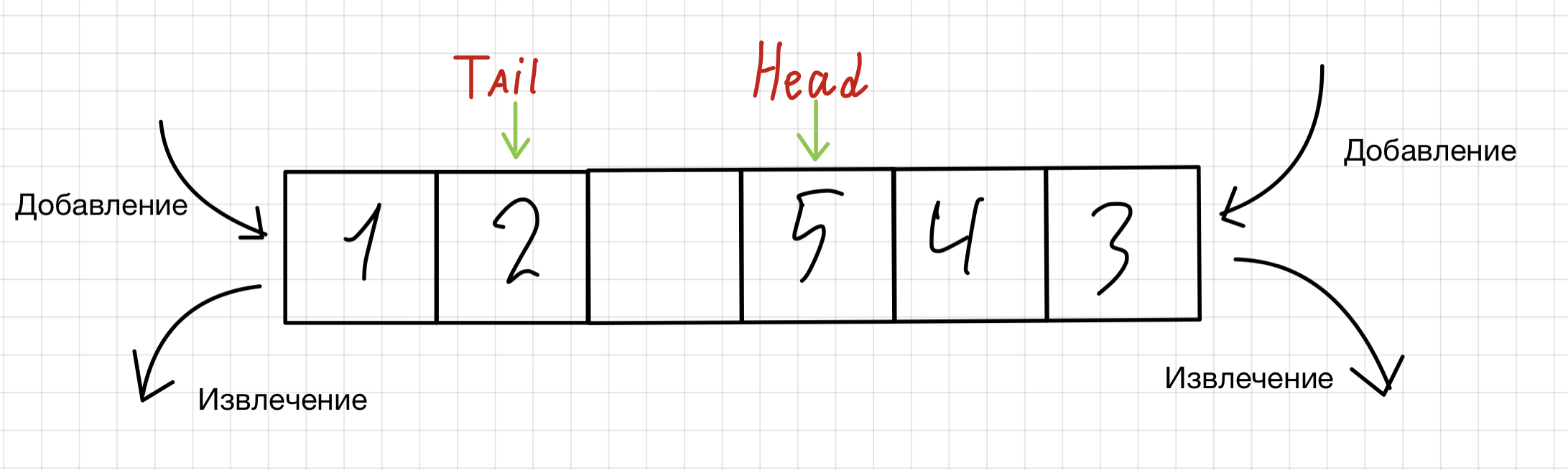
Векторная пустая:



Векторная частично заполнена:



Векторная заполнена:



Текст программы:

// файл main.h

#include "iostream"

#include "stdlib.h"

#include "time.h"

#include "cstdlib"

#include "..\..\..\helpFunctions\functions.cpp"

using namespace std;

typedef int DataType;

// deque(дек) - double ended queue

class Deque {

public:

int size;

int maxlength;

virtual int Empty (void) = 0; //проверка на пустоту

virtual int Full (void) = 0; //проверка на полноту заполнения

virtual void pushBack(DataType) = 0; // добавление элемента в начало дека

virtual void pushFront(DataType) = 0; // добавление элемента в конец дека

virtual DataType getFront (void) = 0; // неразрушающее чтение элемента c начала дека

virtual DataType getBack (void) = 0; // неразрушающее чтение элемента c конца дека

virtual DataType popBack (void) = 0; // удаление эоемента с конца дека

virtual DataType popFront (void) = 0; // удаление эоемента с начала дека

virtual ~Deque();

};

class List: public Deque

{

struct Node {

Node(DataType data = DataType(), Node \*pPrev = nullptr, Node \*pNext = nullptr) {

this->data = data;

this->pNext = pNext;

this->pPrev = pPrev;

}

DataType data;

Node \* pNext, \* pPrev;

} \*head, \*tail; //индексы головы и хвоста

public:

List (int length = 10);

~List ();

int Empty (void) override;

int Full (void) override;

void pushBack(DataType) override;

void pushFront(DataType) override;

DataType getFront (void) override;

DataType getBack (void) override;

DataType popBack (void) override;

DataType popFront (void) override;

};

class Vector: public Deque

{

int head, tail; //индексы головы и хвоста

DataType \*data; //массив для хранения элементов

public:

Vector(int maxlength=10);

~Vector();

int Empty (void) override;

int Full (void) override;

void pushBack(DataType) override;

void pushFront(DataType) override;

DataType getFront (void) override;

DataType getBack (void) override;

DataType popBack (void) override;

DataType popFront (void) override;

};

// Конструкторы

List::List(int length) {

this->head = tail = NULL;

this->maxlength = length;

this->size = 0;

}

Vector::Vector(int length) {

this->maxlength = length;

this->data = new DataType[this->maxlength];

this->head = 0;

this->tail = maxlength - 1;

this->size = 0;

}

// Деструкторы

Deque::~Deque(){}

List::~List () {

Node \* temp = head;

while (head) {

temp = head;

head = head->pNext;

delete temp;

}

}

Vector::~Vector()

{

delete[] this->data;

}

// Проверка на пустоту

int List::Empty() {

return this->head == NULL;

}

int Vector::Empty() {

return (this->tail + 1 ) % maxlength == this->head;

}

// Проверка на полноту заполнения

int List::Full() {

return this->size == this->maxlength;

}

int Vector::Full() {

return ( this->tail + 2 ) % maxlength == this->head;

}

// Добавление элемента в начало дека

void List::pushFront(DataType item) {

if (Full()) {

cout << "deque is full :(" << endl;

return;

}

if (Empty()) {

// Создаём первый элемент

head = tail = new Node(item);

}

else {

// Создаём новый элемент

Node \*pNew = new Node(item, nullptr, head);

head->pPrev = pNew;

// Перемещаем указатель на новый элемент

head = pNew;

}

this->size++;

}

void Vector:: pushFront(DataType item) {

if (Full()) {

cout << "deque is full :(" << endl;

return;

} else {

int value;

this->head = this->head ? this->head - 1 : maxlength - 1;

this->data[this->head] = item;

}

this->size++;

}

// Добавление элемента в конец дека

void List::pushBack(DataType data) {

if (Full()) {

cout << "deque is full :(" << endl;

return;

}

if (Empty()) {

head = tail = new Node(data);

} else {

tail->pNext = new Node(data, tail, nullptr);

tail = tail->pNext;

}

this->size++;

}

void Vector::pushBack(DataType item) {

if (Full()) {

cout << "deque is full :(" << endl;

return;

}

this->tail = ( this->tail + 1 ) % maxlength;

this->data[this-> tail] = item;

this->size++;

}

// Неразрушающее чтение элемента c начала дека

DataType List::getFront() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

return head->data;

}

DataType Vector::getFront() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

return this->data[this->head];

}

// Неразрушающее чтение элемента c конца дека

DataType List::getBack() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

return tail->data;

}

DataType Vector::getBack() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

return this->data[this->tail];

}

// Удаление эоемента с конца дека

DataType List::popBack() {

Node \*temp = tail;

DataType tempData;

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

} else if (size == 1) {

tempData = tail->data;

head = head->pNext;

tail = nullptr;

} else {

tempData = tail->data;

tail = tail->pPrev;

tail->pNext = nullptr;

}

delete temp;

size--;

return tempData;

}

DataType Vector::popBack() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

DataType temp = tail;

this->tail = this->tail ? this->tail - 1 : maxlength - 1;

size--;

return this->data[temp];

}

// Удаление элемента с начала дека

DataType List::popFront() {

Node \*temp= head;

DataType tempData;

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

} else if (size == 1) {

tempData = head->data;

head = head->pNext;

tail = nullptr;

} else {

tempData = head->data;

head = head->pNext;

head->pPrev = nullptr;

}

delete temp;

size--;

return tempData;

}

DataType Vector::popFront() {

if (Empty()) {

cout << "deque is empty :(" << endl;

return 0;

}

DataType temp = this->head;

this->head = (this->head + 1) % maxlength;

size--;

return this->data[temp];

}

// файл main.cpp

#include "main.h"

template <class T>

void testDeque(T \*);

void printDeque(Deque \*);

int chooseSize();

int isPrimeNumber(DataType);

int main() {

int variant = 0;

do {

system("cls");

string menu[] = {

"Select storage structure!",

"1. Vector inplementation",

"2. List inplementation"};

printMenu(menu, 3);

exitMenu(3);

variant = getVariant(3);

system("cls");

switch (variant) {

case 1: {

int size = chooseSize();

Vector \*deque = new Vector(size + 1);

testDeque(deque);

break;

}

case 2: {

int size = chooseSize();

List \*deque = new List(size);

testDeque(deque);

break;

}

}

} while (variant != 3);

return 0;

}

// функцию тестирования дека

template <class T>

void testDeque(T \*deque) {

// заполнение дека элементами

{

srand(time(NULL));

while (!deque->Full()) {

int randNumber = 1 + rand();

deque->pushFront(randNumber);

}

}

// удаление простых чисел из дека

int sizeDeque = deque->size;

T \*dequeHelp = new T(sizeDeque);

for (int i = 0; i < sizeDeque; i++) {

DataType temp = deque->getFront();

if (!isPrimeNumber(temp)) {

dequeHelp->pushBack(temp);

}

deque->popFront();

deque->pushBack(temp);

}

T \*dequeSorted = new T(dequeHelp->size);

{

// сортировка дэка в порядке убывания с помощью другого дека

DataType temp;

DataType current;

int sizeDeq = dequeHelp->size;

dequeSorted->pushBack(dequeHelp->getFront());

dequeHelp->popFront();

while(!dequeHelp->Empty()) {

current = dequeHelp->getFront();

if (current >= dequeSorted->getFront()) {

dequeSorted->pushFront(current);

dequeHelp->popFront();

} else if(current <= dequeSorted->getBack()){

dequeSorted->pushBack(current);

dequeHelp->popFront();

} else {

dequeHelp->popFront();

while(current > dequeSorted->getBack()) {

dequeHelp->pushFront(dequeSorted->getBack());

dequeSorted->popBack();

}

dequeSorted->pushBack(current);

}

}

}

int variant;

do {

system("cls");

string menu[] = {

"What you want to do with deque?",

"1. Print start deque",

"2. Print sorted deque"};

printMenu(menu, 3); // вывод меню

exitMenu(3);

variant = getVariant(3);

system("cls");

switch (variant) {

case 1: {

printDeque(deque);

break;

}

case 2: {

printDeque(dequeSorted);

break;

}

}

} while (variant != 3);

}

// выбор размера дека

int chooseSize() {

cout << "Enter deck size(from 1 to 100)" << endl << "> ";

int size = getVariant(100);

return size;

}

// вывод дека

void printDeque(Deque \*deque) {

system("cls"); // очищаем экран

int sizeDeque = deque->size;

DataType array[sizeDeque];

for (int i = 0; i < sizeDeque; i++) {

array[i] = deque->getFront();

cout << "Node# " << i + 1 << ": " << array[i] << endl;

deque->popFront();

}

for (int i = 0; i < sizeDeque; i++) {

deque->pushBack(array[i]);

}

system("pause");

}

// Данная функция проверяет является ли число простым

DataType isPrimeNumber(DataType number) {

bool isPrime = true;

for(int i = 2; i <= number / 2; ++i) {

if(number % i == 0) { isPrime = false; break; }

}

return isPrime;

}

Результаты работы программы:

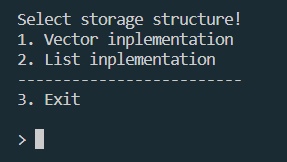


Рисунок 1 – основное меню программы, выбор структуры хранения

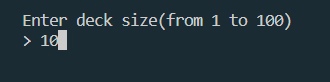


Рисунок 2 – выбор размера дека

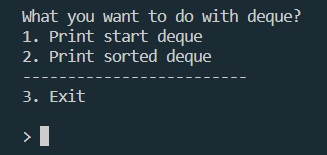


Рисунок 3 – меню для выбора типа выводимого дека

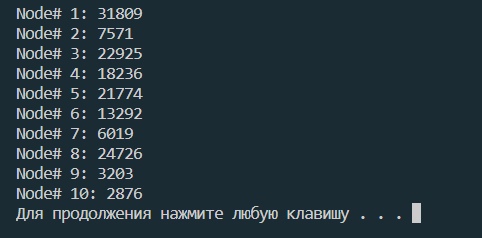


Рисунок 4 – вывод изначального дека(до сортировки)

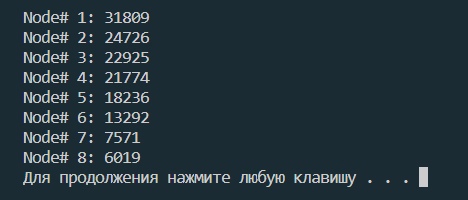


Рисунок 5 – вывод дека после сортировки