



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Факультет

О

Естественнонаучный

шифр

наименование

Кафедра

О7

Информационные системы и программная инженерия

шифр

наименование

Дисциплина

Информационные технологии и программирование

ПРАКТИЧЕСКАЯ РАБОТА №3

Шаблоны

Вариант №6

Выполнил студент группы

О722Б

Вяткин Н.А.

Фамилия И.О.

ПРЕПОДАВАТЕЛЬ

Гладевич А.А.

Фамилия И.О.

Подпись

« 30 »

марта

2023 г.

Цель работы

Научиться создавать шаблоны функций и шаблоны классов для работы с любыми типами данных без переписывания кода программы.

Постановка задачи

Написать шаблон функции, выполняющей указанные в вариативной части задания действия. Написать программу тестирования шаблонных функций, созданных на основе этого шаблона, с аргументами указанных типов. Разработать шаблон класса, описывающий указанный в вариативной части задания абстрактный тип данных, и написать программу тестирования объектов двух шаблонных классов. Выбор тестируемого метода должен осуществляться с помощью меню. Это задание может быть выполнено на трех уровнях сложности:

- 1) низкий: указанный АТД можно реализовать любым удобным способом;
- 2) средний: заданный АТД реализовать с помощью указанной структуры хранения;
- 3) повышенный – создать требуемый АТД с помощью двух структур хранения: векторной и списковой, реализацию оформить в виде шаблонов классов с единым интерфейсом.

Вариант 6

Типы аргументов: unsigned char, float.

Вычисление суммы положительных элементов массива.

АТД – дек с ограниченным выходом.

Структура хранения – связанный список.

Текст программы

Файл main.cpp:

```
#include "DeqOutList.hpp"
#include "DeqOutVector.hpp"
#include <iostream>
#include <conio.h>
#include <time.h>
#include <limits>
#include <iomanip>
#include <locale>

using namespace std;
// случайное заполнение массива
template <class DataType>
DataType *randomFill(DataType *a, int n)
{
    srand(time(0)); // подключаем рандом
    a = new (nothrow) DataType[n];
    if (!a)
    {
        cout << "Ошибка выделения памяти" << endl;
        n = 0;
        return NULL;
    }
    if (typeid(DataType) == typeid(unsigned char)) // если unsigned
char
        for (int i = 0; i < n; i++)
            a[i] = rand() % 256;
    if (typeid(DataType) == typeid(float)) // если float
        for (int i = 0; i < n; i++)
            a[i] = -100 + static_cast<float>(rand()) /
static_cast<float>(RAND_MAX / (200));
    return a;
}
// вывод элементов массива
template <class DataType>
void showArray(DataType *a, int n)
{
    system("cls");
    if (a == NULL || n == 0)
    {
        cout << "Массив пуст" << endl;
    }
    else // если массив не пуст
    {
        cout << "Вывод элементов массива" << endl;
        for (int i = 0; i < n; i++)
            cout << +a[i] << " ";
        cout << endl;
    }
}
// узнать сумму элементов массива
template <class DataType>
float getSum(DataType *a, int n)
{
    system("cls");
```

```

float sum = 0;
if (a == NULL || n == 0) // если пуст
{
    cout << "Массив пуст" << endl;
    return 0;
}
else // если не пуст
{
    cout << "Вычисление суммы положительных элементов массива"
<< endl;
    {
        for (int i = 0; i < n; i++)
            if (a[i] > 0)
                sum += a[i];
        return sum;
    }
}
// ввод размерности массива
int inputSizeArray()
{
    int n;
    system("cls");
    cout << "Введите количество элементов массива для создания" <<
endl;
    while (1)
    {
        cin >> n;
        if (n > 0)
            return n;
        else // если пользователь ввел что то не то
        {
            cout << "Введите корректное количество элементов
массива" << endl;
            cin.clear(); // очищаем поле
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}
// меню тестирования массива
template <class DataType>
void TestArray(DataType *arr, int n)
{
    char c;
    float sum;
    int variant;
    arr = randomFill(arr, n);
    do
    {
        system("cls");
        cout << "Тестирование функции массива" << endl;
        cout << "1. Вывод элементов массива" << endl
            << "2. Узнать сумму положительных элементов массива"
<< endl
            << "3. Обратно в меню" << endl;
        cin >> variant;
        if (variant < 1 || variant > 3)

```

```

        {
            cin.clear(); // очищаем поле
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        switch (variant)
        {
        case 1:
        {
            showArray(arr, n);
            cout << "Для возврата в меню нажмите escape" << endl;
            do
            {
                c = getch();
            } while (c != 27);
            break;
        }
        case 2:
        {
            sum = getSum(arr, n);
            if (+sum == 0)
            {
                cout << "Массив пуст" << endl;
            }
            cout << "Сумма положительных элементов массива: " <<
+sum << endl;

            cout << "Для возврата в меню нажмите escape" << endl;
            do
            {
                c = getch();
            } while (c != 27);
            break;
        }
        }

        } while (variant != 3);
        delete[] arr;
    }
    // меню выбора массива для тестирования
    void menuArray()
    {
        char c;
        int n;
        int menu;
        do
        {
            system("cls");
            cout << "1. Создать массив с элементами типа unsigned char"
<< endl
            << "2. Создать массив с элементами типа float " <<
endl
            << "3. Обратно в меню" << endl;
            cin >> menu;
            if (menu < 1 || menu > 3)
            {
                cin.clear(); // очищаем поле
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            }
        }
    }

```

```

        switch (menu)
        {
        case 1:
        {
            n = inputSizeArray();
            unsigned char *arr = NULL;
            TestArray(arr, n);
            break;
        }
        case 2:
        {
            n = inputSizeArray();
            float *arr = NULL;
            TestArray(arr, n);
            break;
        }
        }
    } while (menu != 3);
}
// меню тестирования дека с ограниченным деком
template <class DataType>
void TestDeq(BaseDeqOut<DataType> *Deq)
{
    DataType x;
    DataType m;
    int menu;
    char c;
    do
    {
        system("cls");
        cout << "Выберите действие" << endl
            << "1. Добавить элемент в открытый конец дека" << endl
            << "2. Добавить элемент в закрытый конец дека" << endl
            << "3. Извлечь элемент с открытого конца" << endl
            << "4. Обратно в меню " << endl;
        cin >> menu;
        if (menu < 1 || menu > 4)
        {
            cin.clear(); // очищаем поле
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        switch (menu)
        {
        case 1:
        {
            system("cls");
            cout << "Добавление элемента в открытый конец дека(в
начало)" << endl;
            cout << "Введите элемент для добавления" << endl;
            cin >> x;
            Deq->addFront(x);
            cout << "Для возврата в меню нажмите escape" << endl;
            do
            {
                c = getch();
            } while (c != 27);
            break;
        }
        case 2:
        {
            system("cls");
            cout << "Добавление элемента в закрытый конец дека(в
конец)" << endl;
            cout << "Введите элемент для добавления" << endl;
            cin >> x;
            Deq->addBack(x);
            cout << "Для возврата в меню нажмите escape" << endl;
            do
            {
                c = getch();
            } while (c != 27);
            break;
        }
        case 3:
        {
            system("cls");
            cout << "Извлечение элемента с открытого конца" << endl;
            Deq->popFront();
            cout << "Для возврата в меню нажмите escape" << endl;
            do
            {
                c = getch();
            } while (c != 27);
            break;
        }
        case 4:
        {
            system("cls");
            cout << "Обратно в меню" << endl;
            break;
        }
        }
    } while (menu != 4);
}

```

```

    }
    case 2:
    {
        system("cls");
        cout << "Добавление элемента в закрытый конец дека" <<
endl;

        cout << "Введите элемент для добавления" << endl;
        cin >> x;
        Deq->addBack(x);
        cout << "Для возврата в меню нажмите escape" << endl;
        do
        {
            c = getch();
        } while (c != 27);
        break;
    }
    case 3:
    {
        system("cls");
        if (!Deq->isEmpty())
        {
            cout << "Извлечение элемента с открытого конца
дека" << endl;
            cout << "Извлечен элемент с открытого конца: " <<
Deq->outHead() << endl;
        }
        else
            cout << "Дек пуст" << endl;
        cout << "Для возврата в меню нажмите escape" << endl;
        do
        {
            c = getch();
        } while (c != 27);
        break;
    }
    }
    } while (menu != 4);
    delete Deq;
}
// меню дека
void menuDeq()
{
    char c;
    int i;
    int menu;
    do
    {
        system("cls");
        cout << "1. Протестировать дек с элементами типа unsigned
char" << endl
        << "2. Протестировать дек с элементами типа float" <<
endl
        << "3. Обратно в меню" << endl;
        cin >> menu;
        if (menu < 1 || menu > 3)
        {
            cin.clear(); // очищаем поле

```

```

        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    switch (menu)
    {
    case 1: // unsigned char
    {
        BaseDeqOut<unsigned char> *Deq;
        system("cls");
        cout << "Выберите структуру хранения для дека" << endl
              << "1. Векторная структура хранения" << endl
              << "2. Связная структура хранения" << endl;
        do
        {
            cin >> i;
            if (i < 1 || i > 3) // если введено что-то
                {
                    cout << "Введите корректное число, 1 или 2" <<
endl;

                    cin.clear(); // очищаем поле
                    cin.ignore(numeric_limits<streamsize>::max(),
'\n');

                }
            } while (i != 1 && i != 2);
            if (i == 1) // если векторная
            {
                Deq = new DeqOutVector<unsigned char>(3);
            }
            if (i == 2) // если связная
            {
                Deq = new DeqOutList<unsigned char>();
            }
            TestDeq(Deq);
            break;
        }
    case 2: // float
    {
        BaseDeqOut<float> *Deq;
        system("cls");
        cout << "Выберите структуру хранения для дека" << endl
              << "1. Векторная структура хранения" << endl
              << "2. Связная структура хранения" << endl;
        do
        {
            cin >> i;
            if (i < 1 || i > 3) // если введено что-то
                {
                    cout << "Введите корректное число, 1 или 2" <<
endl;

                    cin.clear(); // очищаем поле
                    cin.ignore(numeric_limits<streamsize>::max(),
'\n');

                }
            } while (i != 1 && i != 2);
            if (i == 1) // если векторная
            {

```



```

        Deq = new DeqOutVector<float>(3);
    }
    if (i == 2) // если связная
    {
        Deq = new DeqOutList<float>();
    }
    TestDeq(Deq);
    break;
}
}
} while (menu != 3);
}

int main(void)
{
    setlocale(LC_ALL, "Rus");
    int menu, variant;
    do
    {
        system("cls");
        cout << "1. Протестировать работу функции с массивом" <<
endl
        << "2. Протестировать дек с ограниченным выходом" <<
endl
        << "3. Exit" << endl;
        cin >> menu;
        if (menu < 1 || menu > 3)
        {
            cin.clear(); // очищаем поле
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        switch (menu)
        {
            case 1:
            {
                menuArray();
                break;
            }
            case 2:
            {
                menuDeq();
                break;
            }
        }
    } while (menu != 3);
    return 0;
}

```

Файл BaseDeq.h:

```

#ifndef BaseDeqOut_H
#define BaseDeqOut_H
#include <iostream>

template <class DataType>
class BaseDeqOut
{

```

```

public:
    virtual ~BaseDeqOut(){}; // виртуальный деструктор
    virtual void addFront(DataType) = 0; // добавление элемента в
начало дека
    virtual void addBack(DataType) = 0; // доавление элемента в
конец дека
    virtual bool isEmpty() = 0; // проверка на пустоту
    virtual bool isFull() = 0; // проверка на заполненность
    virtual DataType getHead() = 0; // неразрушающее чтение
    virtual DataType outHead() = 0; // извлечение элемента с
открытого конца
};

```

```
#endif
```

Файл DeqOutList.hpp:

```

#ifndef DeqOutList_HPP
#define DeqOutList_HPP
#include "BaseDeq.h"
#include <iostream>

using namespace std;

template <class DataType>
class DeqOutList : public BaseDeqOut<DataType>
{
    class Node
    {
    public:
        DataType data;
        Node *next;

    public:
        Node(DataType x) // конструктор записи односвязного
списка
        {
            this->data = x;
            this->next = NULL;
        }
    };
    Node *head, *tail; // адреса начала и конца дека

public:
    DeqOutList(); //
конструктор без параметра дека
    ~DeqOutList(); //
деструктор дека
    DeqOutList(const DeqOutList &); //
конструктор копирования
    const DeqOutList &operator=(const DeqOutList &); //
перегрузка оператора присваивания
    void addFront(DataType); //
добавление сообщения в начало дека
    void addBack(DataType); //
добавление сообщения в конец дека
    bool isEmpty(); // проверка
на пустоту

```

```

        bool isFull(); // проверка
на заполненность
        DataType getHead(); //
неразрушающее чтения
        void deleteHead(); // удаление
сообщения с открытого конца
        DataType outHead(); //
извлечение сообщения с открытого конца
    };

    // конструктор без параметра дека
    template <class DataType>
    DeqOutList<DataType>::DeqOutList()
    {
        this->head = this->tail = NULL;
    }
    // конструктор копирования
    template <class DataType>
    DeqOutList<DataType>::DeqOutList(const DeqOutList &deq)
    {
        head = deq.head;
        Node *temp = head;
        while (temp != NULL)
        {
            addBack(temp->data);
            temp = temp->next;
            if (isFull())
            {
                cout << "Дек не может быть полностью скопирован" <<
endl;

                delete this;
                break;
            }
        }
    }
    // перегрузка оператора присваивания
    template <class DataType>
    const DeqOutList<DataType> &DeqOutList<DataType>::operator=(const
DeqOutList &deq)
    {
        if (this != &deq)
        {
            head = deq.head;
            Node *temp = head;
            while (temp != NULL)
            {
                addBack(temp->data);
                temp = temp->next;
                if (isFull())
                {
                    cout << "Дек не может быть полностью скопирован"
<< endl;

                    delete this;
                    break;
                }
            }
        }
    }
}

```

```

        return *this;
    }
    // деструктор дека
    template <class DataType>
    DeqOutList<DataType>::~DeqOutList()
    {
        while (head != NULL)
        {
            deleteHead();
        }
    }
    // удаление сообщения с открытого конца
    template <class DataType>
    void DeqOutList<DataType>::deleteHead()
    {
        if (isEmpty())
            return;
        if (head == tail)
        {
            delete tail;
            head = tail = NULL;
            return;
        }

        Node *temp = head;
        head = temp->next;
        delete temp;
    }
    // добавление сообщения в начало дека
    template <class DataType>
    void DeqOutList<DataType>::addFront(DataType x)
    {
        if (!isFull())
        {
            Node *newNode = new (nothrow) Node(x);
            newNode->next = head;
            head = newNode;
            if (tail == NULL)
                tail = newNode;
        }
        else
            cout << "Данный элемент не добавлен" << endl;
    }
    // добавление сообщения в конец дека
    template <class DataType>
    void DeqOutList<DataType>::addBack(DataType x)
    {
        if (!isFull())
        {
            Node *newNode = new (nothrow) Node(x);
            if (isEmpty())
                head = newNode;
            if (tail != NULL)
                tail->next = newNode;
            tail = newNode;
        }
        else

```

```

        cout << "Данный элемент не добавлен" << endl;
    }
    // проверка на пустоту
    template <class DataType>
    bool DeqOutList<DataType>::isEmpty()
    {
        return (!head);
    }
    // проверка на заполненность
    template <class DataType>
    bool DeqOutList<DataType>::isFull()
    {
        Node *newNode = new (nothrow) Node(0);
        if (!newNode)
        {
            cout << "Ошибка выделения памяти" << endl;
            cout << "Дек заполнен" << endl;
            delete newNode;
            return true;
        }
        else
        {
            return false;
        }
    }
    // неразрушающее чтение
    template <class DataType>
    DataType DeqOutList<DataType>::getHead()
    {
        if (isEmpty())
            return -100;
        else
            return head->data;
    }
    // извлечение сообщения с открытого конца
    template <class DataType>
    DataType DeqOutList<DataType>::outHead()
    {
        if (isEmpty())
            return -100;
        DataType x = head->data;
        deleteHead();
        return x;
    }
#endif

```

Файл DeqOutVector.hpp:

```

#ifndef DeqOutVector_HPP
#define DeqOutVector_HPP
#include "BaseDeq.h"
#include <iostream>

using namespace std;

template <class DataType>

```

```

class DeqOutVector : public BaseDeqOut<DataType>
{
    int head, tail; // индекс первого и конечного элемента
    int size;       // размер массива
    DataType *data; // массив сообщений

public:
    DeqOutVector(int); //
конструктор дека с парметром
    ~DeqOutVector(); //
деструктор дека
    DeqOutVector(const DeqOutVector &); //
конструктор копирования
    const DeqOutVector &operator=(const DeqOutVector &); //
перегрузка оператора присваивания
    void addFront(DataType); //
добавление элемента в начало дека
    void addBack(DataType); //
добавление элемента в конец дека
    bool isEmpty(); // проверка
на пустоту
    bool isFull(); // проверка
на заполненность
    DataType getHead(); //
неразрушающее чтения
    int getSize() { return size; }; //
получение размера дека
    void deleteHead(); // удаление
сообщения с открытого конца
    DataType outHead(); //
извлечение сообщения с открытого конца
};

// конструктор дека с парметром
template <class DataType>
DeqOutVector<DataType>::DeqOutVector(int n) : size(n)
{
    data = new (nothrow) DataType[size];
    if (!data)
    {
        cout << "Ошибка выделения памяти" << endl;
        return;
    }
    head = 0;
    tail = -1;
    this->size = size;
}

// конструктор копирования
template <class DataType>
DeqOutVector<DataType>::DeqOutVector(const DeqOutVector &deq)
{
    data = new (nothrow) DataType[deq.size];
    if (!data)
    {
        cout << "Ошибка выделения памяти" << endl;
        return;
    }
}

```

```

    }
    for (int i = 0; i < deq.size; i++)
        data[i] = deq.data[i];
    size = deq.size;
    head = deq.head;
    tail = deq.tail;
}
// перегрузка оператора присваивания
template <class DataType>
const                               DeqOutVector<DataType>
&DeqOutVector<DataType>::operator=(const DeqOutVector &deq)
{
    if (this != &deq)
    {
        delete[] data;
        data = new (nothrow) DataType[deq.size];
        if (!data)
        {
            cout << "Ошибка выделения памяти" << endl;
            return NULL;
        }
        for (int i = 0; i < deq.size; i++)
            data[i] = deq.data[i];
        size = deq.size;
        head = deq.head;
        tail = deq.tail;
    }
    return *this;
}
// деструктор дека
template <class DataType>
DeqOutVector<DataType>::~DeqOutVector()
{
    delete[] data;
}
// проверка на пустоту
template <class DataType>
bool DeqOutVector<DataType>::isEmpty()
{
    return (tail == -1);
}
// проверка на заполненность
template <class DataType>
bool DeqOutVector<DataType>::isFull()
{
    return (tail == 0 && head == size - 1 || tail == head + 1);
}
// неразрушающее чтения
template <class DataType>
DataType DeqOutVector<DataType>::getHead()
{
    if (isEmpty())
        return -100;
    return data[head];
}
// добавление сообщения в начало дека
template <class DataType>

```

```

void DeqOutVector<DataType>::addFront(DataType x)
{
    if (isFull())
    {
        cout << "Дек заполнен" << endl;
        cout << "Данный элемент не добавлен" << endl;
    }
    else
    {
        if (isEmpty())
        {
            head = 0;
            tail = 0;
        }
        else if (head == size - 1)
            head = 0;
        else
            head += 1;
        data[head] = x;
    }
}
// добавление сообщения в конец дека
template <class DataType>
void DeqOutVector<DataType>::addBack(DataType x)
{
    if (isFull())
    {
        cout << "Дек заполнен" << endl;
        cout << "Данный элемент не добавлен" << endl;
    }
    else
    {
        if (isEmpty())
        {
            head = 0;
            tail = 0;
        }
        else if (tail == 0)
            tail = size - 1;
        else
            tail = tail - 1;
        data[tail] = x;
    }
}
// удаление элемента с открытого конца
template <class DataType>
void DeqOutVector<DataType>::deleteHead()
{
    if (isEmpty())
        return;
    if (tail == head)
    {
        head = -1;
        tail = -1;
    }
    else if (head == 0)
        head = size - 1;
}

```



```

        else
            head = head - 1;
    }
    // извлечение сообщения с открытого конца
    template <class DataType>
    DataType DeqOutVector<DataType>::outHead()
    {
        if (isEmpty())
            return -100;
        DataType temp = data[head];
        Delet eHead();
        return temp;
    }
#endif

```

Скриншоты

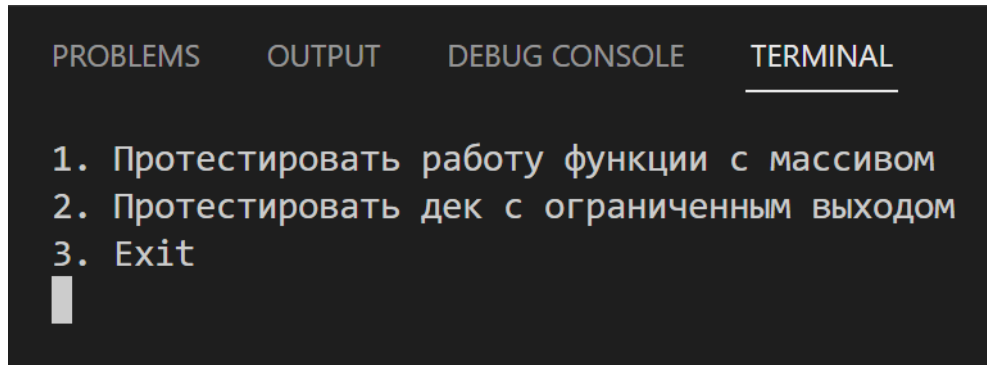


Рисунок 1 – Основное меню

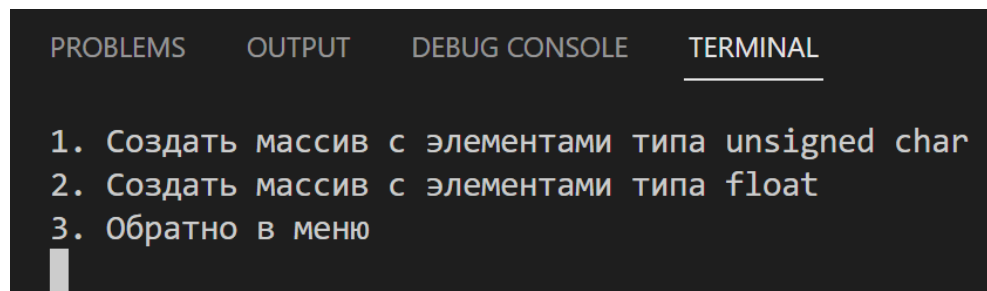


Рисунок 2 – Меню выбора типа элементов массива

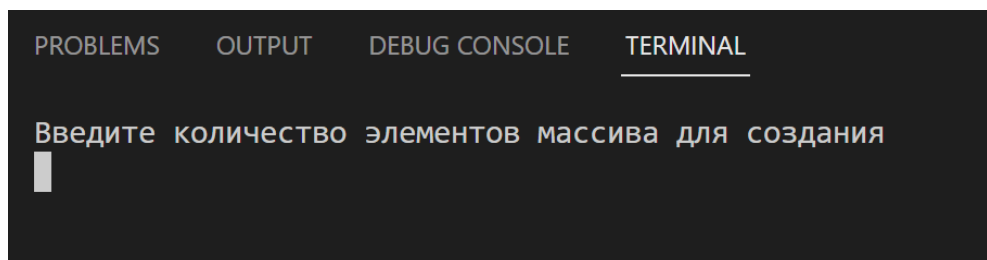


Рисунок 3 – Ввод количества элементов массива

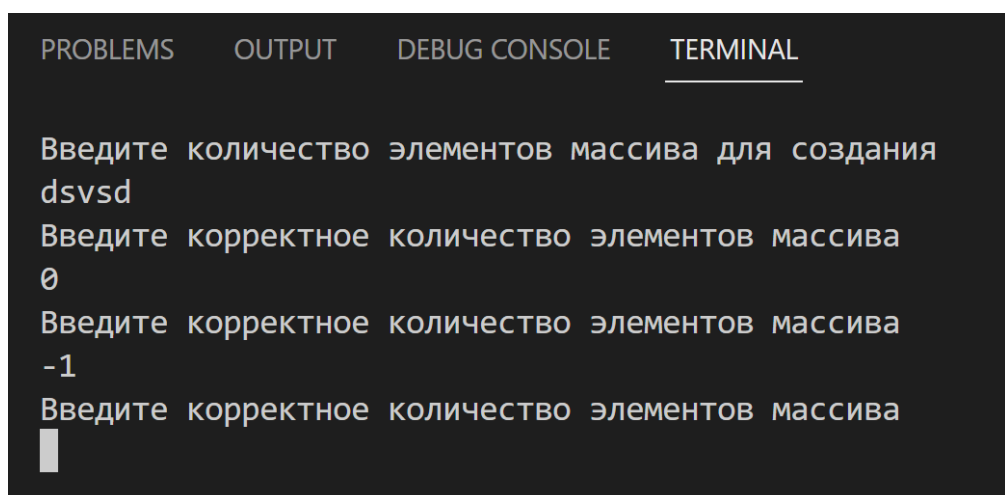


Рисунок 4 – Введено некорректное количество элементов массива

```
Тестирование функции массива
1. Вывод элементов массива
2. Узнать сумму положительных элементов массива
3. Обрато в меню
█
```

Рисунок 5 – Меню тестирования функций массива

```
Вывод элементов массива
211 42 66 80 21
Для возврата в меню нажмите escape
█
```

Рисунок 6 – Вывод элементов массива при типе элементов unsigned char

```
Вычисление суммы положительных элементов массива
Сумма положительных элементов массива: 420
Для возврата в меню нажмите escape
█
```

Рисунок 7 – Вывод суммы положительных элементов массива

```
Вывод элементов массива
46.411 28.8221 50.411 -79.9632 80.6749
Для возврата в меню нажмите escape
█
```

Рисунок 8 – Вывод элементов массива при типе элементов float

```
Вычисление суммы положительных элементов массива
Сумма положительных элементов массива: 206.319
Для возврата в меню нажмите escape
█
```

Рисунок 9 – Вывод суммы положительных элементов массива


1. Протестировать дек с элементами типа unsigned char
 2. Протестировать дек с элементами типа float
 3. Обратно в меню
- 

Рисунок 10 – Меню выбора типа элементов для дека

Выберите структуру хранения для дека


1. Векторная структура хранения
 2. Связная структура хранения
- 

Рисунок 11 – Меню выбора структуры хранения для дека

Выберите структуру хранения для дека

1. Векторная структура хранения
2. Связная структура хранения

4

Введите корректное число, 1 или 2
выа

Введите корректное число, 1 или 2
fsdf

Введите корректное число, 1 или 2




Рисунок 12 – Если пользователь делает некорректный выбор

Выберите действие


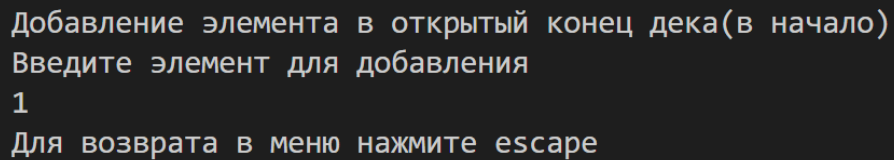
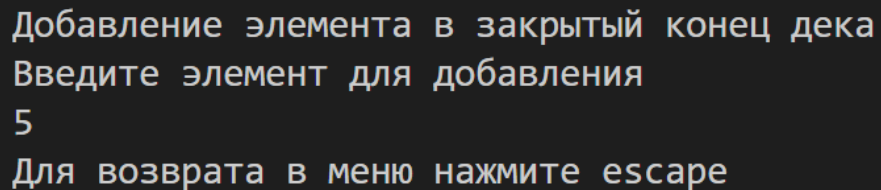
1. Добавить элемент в открытый конец дека
 2. Добавить элемент в закрытый конец дека
 3. Извлечь элемент с открытого конца
 4. Обратно в меню
- 

Рисунок 13 – Меню тестирования дека



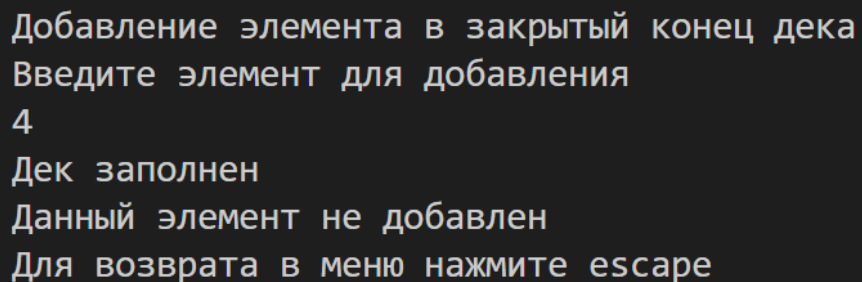
Добавление элемента в открытый конец дека(в начало)
Введите элемент для добавления
1
Для возврата в меню нажмите escape

Рисунок 14– Добавление элемента в начало дека, где есть извлечение



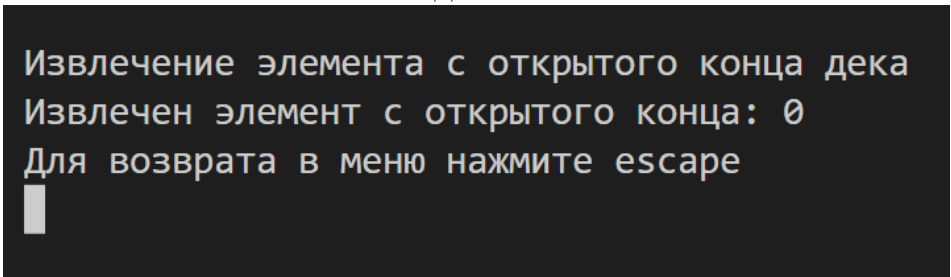
Добавление элемента в закрытый конец дека
Введите элемент для добавления
5
Для возврата в меню нажмите escape

Рисунок 15 – Добавление элемента в конец дека



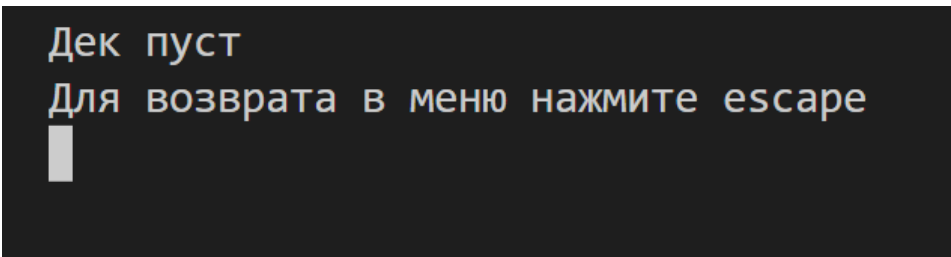
Добавление элемента в закрытый конец дека
Введите элемент для добавления
4
Дек заполнен
Данный элемент не добавлен
Для возврата в меню нажмите escape

Рисунок 16– При векторной структуре хранения элемент не был добавлен,
так как дек заполнен



Извлечение элемента с открытого конца дека
Извлечен элемент с открытого конца: 0
Для возврата в меню нажмите escape

Рисунок 17 – Извлечение элемента с открытого конца дека



Дек пуст
Для возврата в меню нажмите escape

Рисунок 18 – Извлечение с открытого конца, если дек пуст