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»
часть 2 «Стеки, очереди, деки»

вариант 5

Выполнил:
Студент Вяткин Н.А.
Группа О722Б

Преподаватель:
Гладевич А. А

Санкт-Петербург
2023 г.

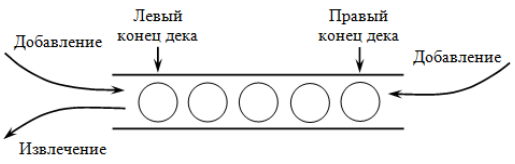
Структура данных – дек с ограниченным выходом.

Уровень сложности – повышенный.

Написать на языке C++ программу, решающую поставленную в вариативной части задачу, используя указанную там же структуру данных. Выполнение задания осуществляется в два этапа. На первом этапе требуется реализовать необходимую 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. Методы класса должны выполнять только основные и дополнительные операции, допустимые над элементами описываемой СД (перечислены ниже), никаких других операций в классе быть не должно. На втором этапе требуется использовать объекты созданных классов при решении поставленной задачи. Выбор структуры хранения предоставить пользователю, объекты создавать динамически, используя указатель на базовый класс.

Структура данных – дек с ограниченным выходом. Некая программа может получать входные данные от пользователя и изредка от другой программы, причем данные, предоставляемые другой программой, имеют более высокий приоритет в сравнении с пользователем. Напишите программу для моделирования потоков данных в описанном процессе. Пояснения к заданию. С клавиатуры никакие данные не вводятся, все «данные» в моделировании генерируются программой. Работа программы осуществляется в «бесконечном» цикле (лучше всего – до нажатия клавиши). На каждом шаге цикла с разными вероятностями (например, вероятность появления сообщения от пользователя 95%, вероятность появления сообщения от другой программы 10%) генерируются случайные значения. Если есть сообщение от программы, оно добавляется в дек с одного конца (с которого возможно извлечение), если есть сообщение от пользователя, оно добавляется с другого конца. Если дек не пуст, одно сообщение из дека извлекается и «передается обрабатывающей программе», после чего происходит возврат к началу цикла. Каждое действие должно сопровождаться выводом на экран (от программы поступило сообщение такое-то; от пользователя поступило сообщение такое-то; такое-то сообщение передано обрабатывающей программе).

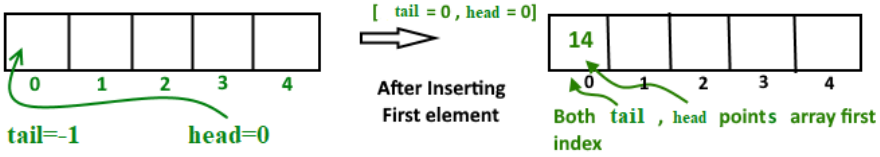
Схематичное изображение структуры данных:



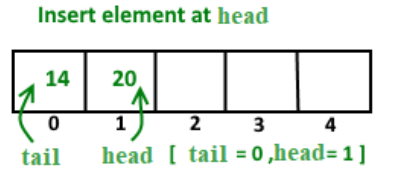
Схематичное изображение структуры хранения, использованной в программе:

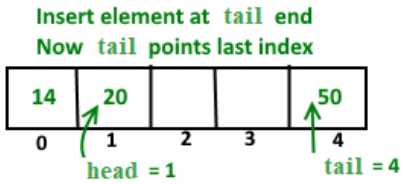
Векторная структура хранения

Пустая и заполнена одним элементом

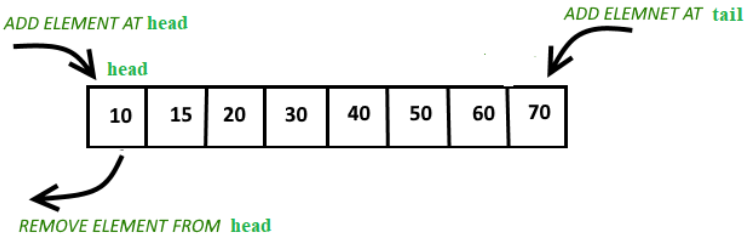


Заполнена несколькими элементами, при разных добавлениях

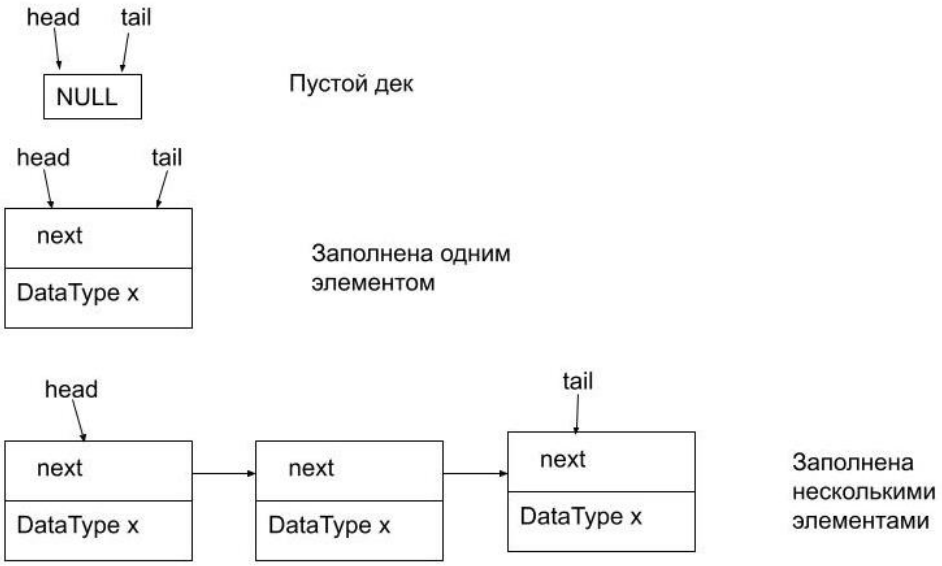




Заполненная



Связная структура хранения



Текст программы:

main.cpp

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

using namespace std;

int main(void)
{
    setlocale(LC_ALL, "Rus");
    BaseDeqOut *Deq;
    srand(time(0)); // подключаем рандом
    char c;
    int i, k, n, l, x;
    // просим пользователя выбрать структуру хранения
    cout << "Выберите структру хранения для дека" << endl
         << "1. Векторная структура хранения" << endl
         << "2. Связная структура хранения" << endl;
    do
    {
        cin >> i;
        if (i < 1 || i > 2) // если введено что-то некоректное
        {
            cout << "Введите корректное число, 1 или 2";
            cin.clear(); // очищаем поле
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    } while (i != 1 && i != 2);
    if (i == 1) // если векторная
        Deq = new DeqOutVector(100);
    if (i == 2) // если связная
        Deq = new DeqOutList();
    system("cls");
    while (1)
    {
        n = rand() % 5; // генерируем случайное количество сообщения
        if (n == 0)
        {
            cout << "Сообщения не были получены" << endl
                 << endl;
        }
        for (i = 0; i < n; i++) // генерируем случайные сообщения
        {
            k = -5 + rand() % 20;
            if (k >= 0) // сообщение от пользователя если больше или равно
нулю
            {
                cout << "-----------------------------------" << endl;
                cout << "|Сообщение получено от пользователя|" << endl;
                cout << "-----------------------------------" << endl;
                cout << "|Содержимое сообщения: " << setw(12) << left <<
k << "|" << endl;
                cout << "-----------------------------------" << endl;
```

```
                    Deq->addBack(k); // добавляем запись в конец с которого
нет извлечения
                }
                if (k < 0) // если меньше нуля, то получено от программы
                {
                    cout << "---------------------------------" << endl;
                    cout << "|Сообщение получено от программы|" << endl;
                    cout << "---------------------------------" << endl;
                    cout << "|Содержимое сообщения: " << setw(9) << left << k
<< "|" << endl
                         << "---------------------------------" << endl
                         << endl;
                    Deq->addFront(k); // добавляем в открытый конец дека
                }
            }
            l = Deq->outHead();
            if (l < 0 && l > -100) // если сообщение меньше нуля то это от
программы b дек не пуст
            {
                cout << endl
                     <<  "Сообщение  от  программы  передано  обрабатывающей
программе" << endl;
                cout << "Содержимое сообщения: " << l << endl
                     << endl;
            }
            if  (l  >=  0)  //  иначе  сообщение  от  пользователя  передано
обрабатывающей программе
            {
                cout << endl
                     << "Сообщение от пользователя передано обрабатывающей
программе" << endl;
                cout << "Содержимое сообщения: " << l << endl
                     << endl;
            }
            if (l == -100) // если дек пуст l будет =-100
                cout << endl
                     << "Дек пуст" << endl;
            // спрашиваем пользователя не хочет ли он закончить
            cout << "Для выхода нажмите escape" << endl;
            cout << "Для продолжений нажмите любую кнопку" << endl
                 << endl;
            c = getch();
            if (c == 27)
            {
                break;
            }
        }
        system("pause");
        delete Deq;
        return 0;
    }
```

BaseClass.h

```
#ifndef DeqOut_H
#define DeqOut_H
#include <iostream>

typedef int DataType;

class BaseDeqOut
{
```

```
    public:
        virtual ~BaseDeqOut(){};              // виртуальный деструктор
        virtual void addFront(DataType) = 0; // добавление сообщения в начало
дека
        virtual void addBack(DataType) = 0;   // доавление сообщения 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 сообщения с открытого
конца
    };

    #endif
```

DeqOutList.h

```
    #ifndef DeqOutList_H
    #define DeqOutList_H
    #include "BaseClass.h"
    #include <iostream>

    using namespace std;

    typedef int DataType;

    // запись односвязного списка
    class Node
    {
    public:
        DataType data;
        Node *next;

    public:
        Node(DataType x) // конструктор записи односвязного списка
        {
            this->data = x;
            this->next = NULL;
        }
    };

    class DeqOutList : public BaseDeqOut
    {
        Node *head, *tail; // адреса начала и конца дека
        int size;          // размер дека

    public:
        DeqOutList();                                        // конструктор без
параметро 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 открытого конца дека
          void deleteHead();                                  // удаление сообщения
с открытого конца
          DataType outHead();                                // извлечение сообщения
с открытого конца
      };

      #endif
```

DeqOutList.cpp

```
      #include "DeqOutList.h"
      // конструктор без параметро дека
      DeqOutList::DeqOutList()
      {
          this->head = this->tail = NULL;
      }
      // конструктор копирования
      DeqOutList::DeqOutList(const DeqOutList &deq)
      {
          head = deq.head;
          Node *temp = head;
          while (temp != NULL)
          {
              addBack(temp->data);
              temp = temp->next;
          }
      }
      // перегрузка оператора присваивания
      const DeqOutList &DeqOutList::operator=(const DeqOutList &deq)
      {
          if (this != &deq)
          {
              head = deq.head;
              Node *temp = head;
              while (temp != NULL)
              {
                  addBack(temp->data);
                  temp = temp->next;
              }
          }
          return *this;
      }
      // деструктор дека
      DeqOutList::~DeqOutList()
      {
          while (head != NULL)
          {
              deleteHead();
          }
      }
      // удаление сообщения с открытого конца
      void DeqOutList::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
void DeqOutList::addFront(DataType x)
{
    if (!isFull())
    {
        Node *newNode = new (nothrow) Node(x);
        newNode->next = head;
        head = newNode;
        if (tail == NULL)
            tail = newNode;
    }
}
// добавление сообщения в конец дека
void DeqOutList::addBack(DataType x)
{
    if (!isFull())
    {
        Node *newNode = new (nothrow) Node(x);
        if (isEmpty())
            head = newNode;
        if (tail != NULL)
            tail->next = newNode;
        tail = newNode;
    }
}
// проверка на пустоту
bool DeqOutList::isEmpty()
{
    return (!head);
}
// проверка на заполненность
bool DeqOutList::isFull()
{

    Node *newNode = new (nothrow) Node(0);
    newNode->next = head;
    head = newNode;
    if (tail == NULL)
        tail = newNode;
    if (!newNode)
    {
        cout << "Ошибка выделения памяти" << endl;
        cout << "Дек заполнен" << endl;
        cout << "Сообщение не было добавлено" << endl
             << endl;
        deleteHead();
        return true;
    }
    else
    {
        deleteHead();
        return false;
    }
}
// неразрушающее чтения
DataType DeqOutList::getHead()
```

```
{
    if (isEmpty())
        return -100;
    else
        return head->data;
}
// извлечение сообщения с открытого конца
DataType DeqOutList::outHead()
{
    if (isEmpty())
        return -100;
    DataType x = head->data;
    deleteHead();
    return x;
}
```

DeqOutVector.h

```
#ifndef DeqOutVector_H
#define DeqOutVector_H
#include "BaseClass.h"
#include <iostream>

using namespace std;

typedef int DataType;

class DeqOutVector : public BaseDeqOut
{
    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
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
    int getSize() { return size; };                        // получение
размера дека
    void deleteHead();                                     // удаление
сообщения с открытого конца
    DataType outHead();                                   // извлечение
сообщения с открытого конца
};

#endif
```

DeqOutVector.cpp

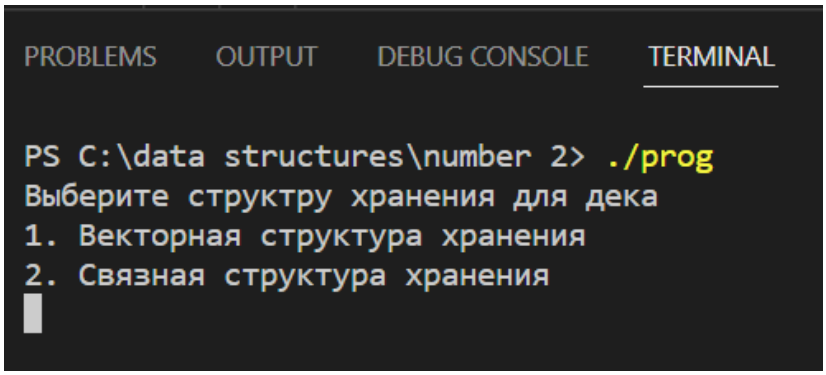
```
#include "DeqOutVector.h"
// конструктор дека с парметром
DeqOutVector::DeqOutVector(int n) : size(n)
{
    data = new (nothrow) DataType[size];
    if (!data)
    {
        cout << "Ошибка выделения памяти" << endl;
        return;
    }
    head = 0;
    tail = -1;
    this->size = size;
}
// конструктор копирования
DeqOutVector::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
const DeqOutVector &DeqOutVector::operator=(const DeqOutVector &deq)
{
    if (this != &deq)
    {
        delete[] data;
        data = new (nothrow) DataType[deq.size];
        if (!data)
        {
            cout << "Ошибка выдеения 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
DeqOutVector::~DeqOutVector()
{
    delete[] data;
}
// проверка на пустоту
bool DeqOutVector::isEmpty()
{
    return (tail == -1);
}
// проверка на заполненность
bool DeqOutVector::isFull()
```

```
{
    return (tail == 0 && head == size - 1 || tail == head + 1);
}
// неразрушающее чтения
DataType DeqOutVector::getHead()
{
    if (isEmpty())
        return -100;
    return data[head];
}
// добавление сообщения в начало дека
void DeqOutVector::addFront(DataType x)
{
    if (isFull())
    {
        cout << "Дек заполнен" << endl;
        cout << "Сообщение не было добавлено" << endl
             << 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
void DeqOutVector::addBack(DataType x)
{
    if (isFull())
    {
        cout << "Дек заполнен" << endl;
        cout << "Сообщение не было добавлено" << endl
             << 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
void DeqOutVector::deleteHead()
{
    if (isEmpty())
        return;
```

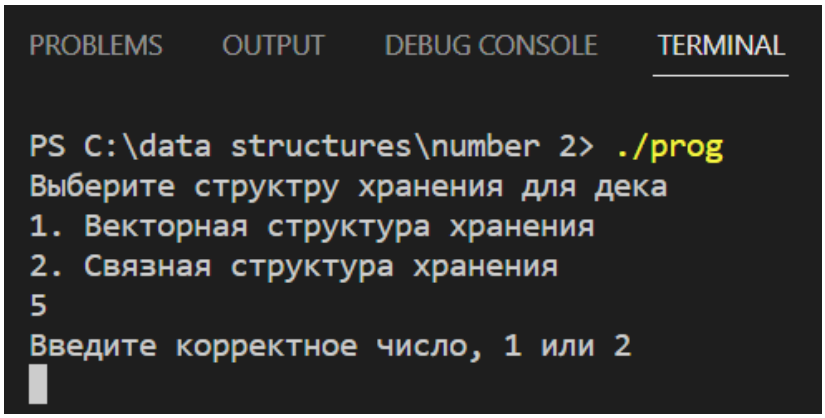
```
    if (tail == head)
    {
        head = -1;
        tail = -1;
    }
    else if (head == 0)
        head = size - 1;
    else
        head = head - 1;
}
// извлечение сообщения с открытого конца
DataType DeqOutVector::outHead()
{
    if (isEmpty())
        return -100;
    int temp = data[head];
    deleteHead();
    return temp;
}
```

Результаты работы программы:

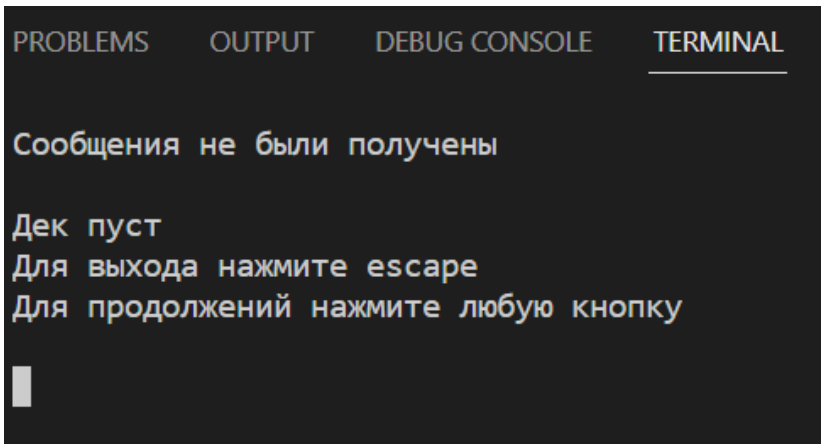
Меню выбора структуры хранения.



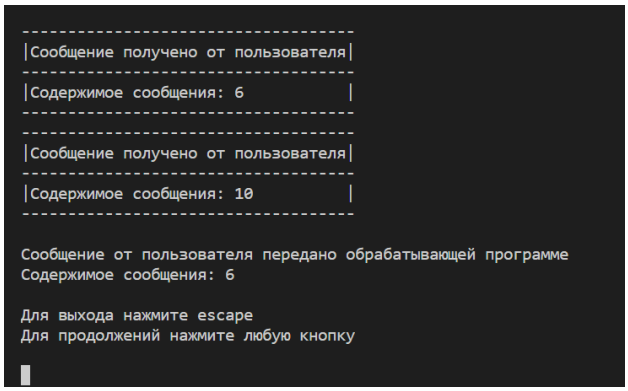
Если пользователь ввел что-то не то.



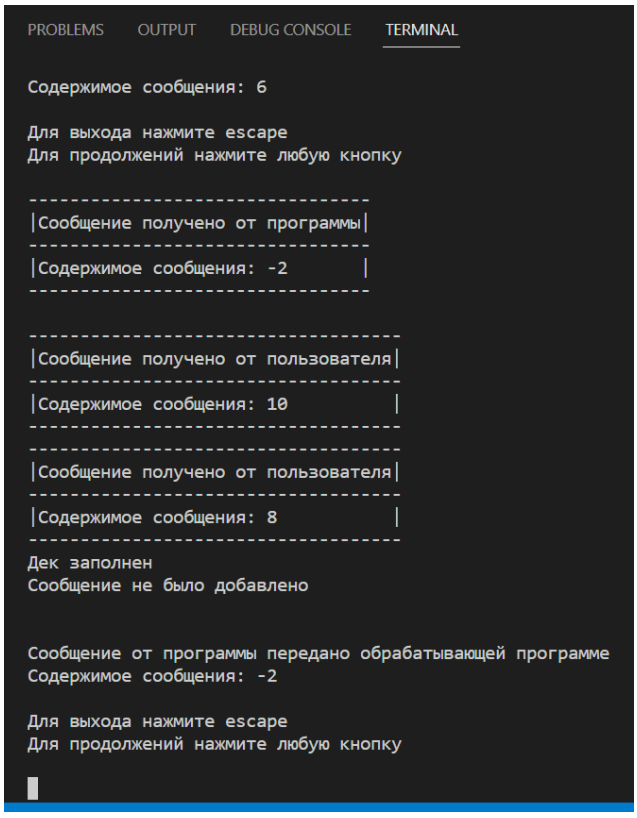
Если никаких сообщений не поступило и дек пуст для передачи сообщения для обработки.



Получено несколько сообщений от пользователя (положительные числа от 0 включительно) и передано обрабатывающей программе первое полученное сообщение от пользователя, так как первое сообщение было записано в правый конец дека (закрытый для извлечения), следующее полученное сообщение было записано также в конец дека, поэтому первое сообщение попало в начало дека (голову дека, открытую для извлечения).



При векторной структуре хранения. Получено несколько сообщений, одно от программы, два от пользователя, сообщение от программы (отрицательное число) записывается в открытый конец дека (слева, где возможно извлечение), а сообщения от пользователя записывают в конец дека (справа, где нет извлечения), поэтому обрабатывающей программе передано сообщение от программы, также сообщение от пользователя (8) не было добавлено, так как дек был заполнен.



Сообщения не были получены, поэтому обрабатывающей программе передается то сообщение, которое находится с открытого конца (в голове дека слева).

```
Для выхода нажмите escape
Для продолжений нажмите любую кнопку

Сообщения не были получены

Сообщение от пользователя передано обрабатывающей программе
Содержимое сообщения: 10

Для выхода нажмите escape
Для продолжений нажмите любую кнопку
```

Получено несколько сообщений, два от программы (отрицательные числа) и два от пользователя (положительные числа от 0), сообщение от программы записывается в начало дека (открытый для извлечения конец дека), следующее тоже от программы и записывается в начало дека, следующие сообщения от пользователя записываются в конец дека(где нет извлечения), поэтому обрабатывающей программе передано второе сообщение от программы, которое находится в начале дека, то есть выводит -4.

```
--------------------------------
|Сообщение получено от программы|
--------------------------------
|Содержимое сообщения: -5       |
--------------------------------

--------------------------------
|Сообщение получено от программы|
--------------------------------
|Содержимое сообщения: -4       |
--------------------------------

-----------------------------------
|Сообщение получено от пользователя|
-----------------------------------
|Содержимое сообщения: 11          |
-----------------------------------
-----------------------------------
|Сообщение получено от пользователя|
-----------------------------------
|Содержимое сообщения: 5           |
-----------------------------------

Сообщение от программы передано обрабатывающей программе
Содержимое сообщения: -4

Для выхода нажмите escape
Для продолжений нажмите любую кнопку
```

Получено несколько сообщений, все от пользователя (положительные число от 0) добавились в конец дека, где нет извлечения, сообщений от программы не получено, поэтому в голову дека попало сообщение от пользователя 8, оно было передано обрабатывающей программе, получено еще несколько сообщений от пользователя, все они записаны были в конец дека, где нет извлечения, поэтому было передано программе сообщение от пользователя с содержим 2, которое находится в голове дека после извлечения предыдущего сообщения.