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

Кафедра И5

«Информационные системы и программная инженерия»

Практическое задание №1

по дисциплине «Структуры данных»

На тему **«Линейные структуры данных»**

**Выполнил:**

Студент : Дубровский В.И.

Группа: И582

**Преподаватель:**

Палехова О. А.

Санкт-Петербург

2019 г.

**Вариант 8**

*Задание 1:*

Данные хранятся в бинарном файле записей, а для обработки считываются в связанный список. При выходе из программы обработанные данные сохраняются в том же файле. Для организации интерфейса должно использоваться меню.

(Средний уровень сложности.) Имя файла с данными должно передаваться программе при ее запуске (через параметры функции main()). Если параметры пользователем при запуске программы не заданы, имя файла задается константой. Список линейный односвязный. Элементами данных должны являться записи сложной структуры**.** Обязательные операции: добавление элемента в произвольное место списка, просмотр списка, удаление произвольного элемента списка. Вывод данных осуществлять в табличном виде с графлением визуально подходящими символами.

Поля данных: название, символическое обозначение, массу атома, заряд ядра. Вывести сведения о химическом элементе по его символическому названию. Найти элемент с самой большой массой

*Описание структур:*

Структура table: //описывает химический элемент

Поля структуры:

char name[20]; //название хим. элемента

char symbol[4]; //символическое обозначение хим.элемента

double weight; //масса

char charge; //заряд

Структура list: //описывает элементы односвязного списка

Поля структуры:

DataType data; //элемент структурного типа DataType

struct list \*next; //указатель на следующий объект структурного //типа List

*Описание функций:*

DataType input\_el (void); //Ввод данных об игрушке

Входные данные: нет

Возвращаемые значения: table – тип DataType (заполненные сведения об элементе)

list add\_begin (list begin, DataType table); // Добавления в начало списка

Входные данные: 1ый аргумент - указатель на начало списка , 2-ой - заполненная структура(строка из таблицы) возвращает temp - указатель на начало списка

list add\_end(list begin, DataType table); // Добавления в конец списка

Входные данные: 1ый аргумент - указатель на начало списка , 2-ой - заполненная структура(строка из таблицы)

Возвращаемые значения: begin – указатель на начало списка

list add(list begin); // Добавления после существующего элемента структуры

Новый элемент заполняется пользователем с клавиатуры

Входные данные: begin – указатель на начало списка

Возвращаемые значения: begin – указатель на начало списка

list read\_file (FILE \*file\_ptr, list begin); //Считывание данных из файла

Входные данные: file\_ptr - указатель на структуру FILE, begin – указатель на первый элемент типа list

Возвращаемые значения: list – указатель на первый элемент;

list del\_element(list begin); //Удаление элемента из списка

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: begin – указатель на первый элемент типа list

void del\_list(list begin); //Удаление списка

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: ничего не возвращает

list search(list begin); //Поиск элемента по символическому обозначению

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: temp – указатель на найденный элемент типа list

void biggest(list begin); //Вывод элемента с наибольшей массой

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: ничего не возвращает

void resave(list begin, char \*name) //Перезапись файла

Входные данные: begin – указатель на первый элемент типа list, name – строка содержащая имя файла

Возвращаемые значения: ничего не возвращает

list reduct(list begin); //Редактирование информации об элементе

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: begin – указатель на первый элемент типа list

void output\_table(list begin) ; //Вывод таблицы элементов

Входные данные: begin – указатель на первый элемент типа list

Возвращаемые значения: ничего не возвращает

void menu\_return(void) //Просит нажать enter и очищает экран

Входные данные: нет

Возвращаемы е значения: ничего не возвращает

void alloc\_struct(list \*begin); //Проверка выделения памяти

Входные данные: \*begin – указатель на указатель элемента типа list

Возвращаемые значения: ничего не возвращает

int delhelp(list temp); //Выводит вспомогательное контекстное меню

Входные данные: temp – указатель на найденный элемент

Возвращаемые значения: флаг типа int

На рисунках 1, 2, 3 представлен результат работы программы.

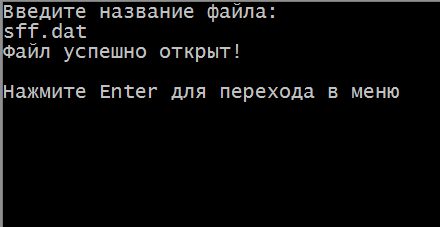
**

Рисунок 1.

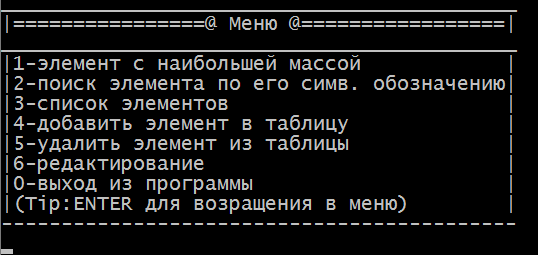


Рисунок 2*.*

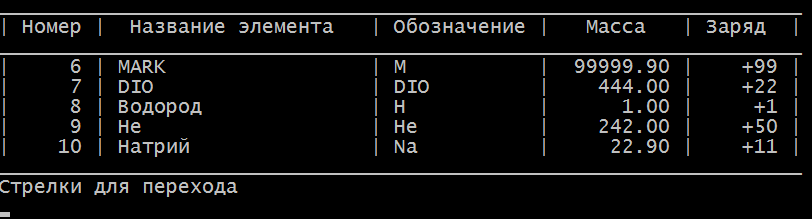
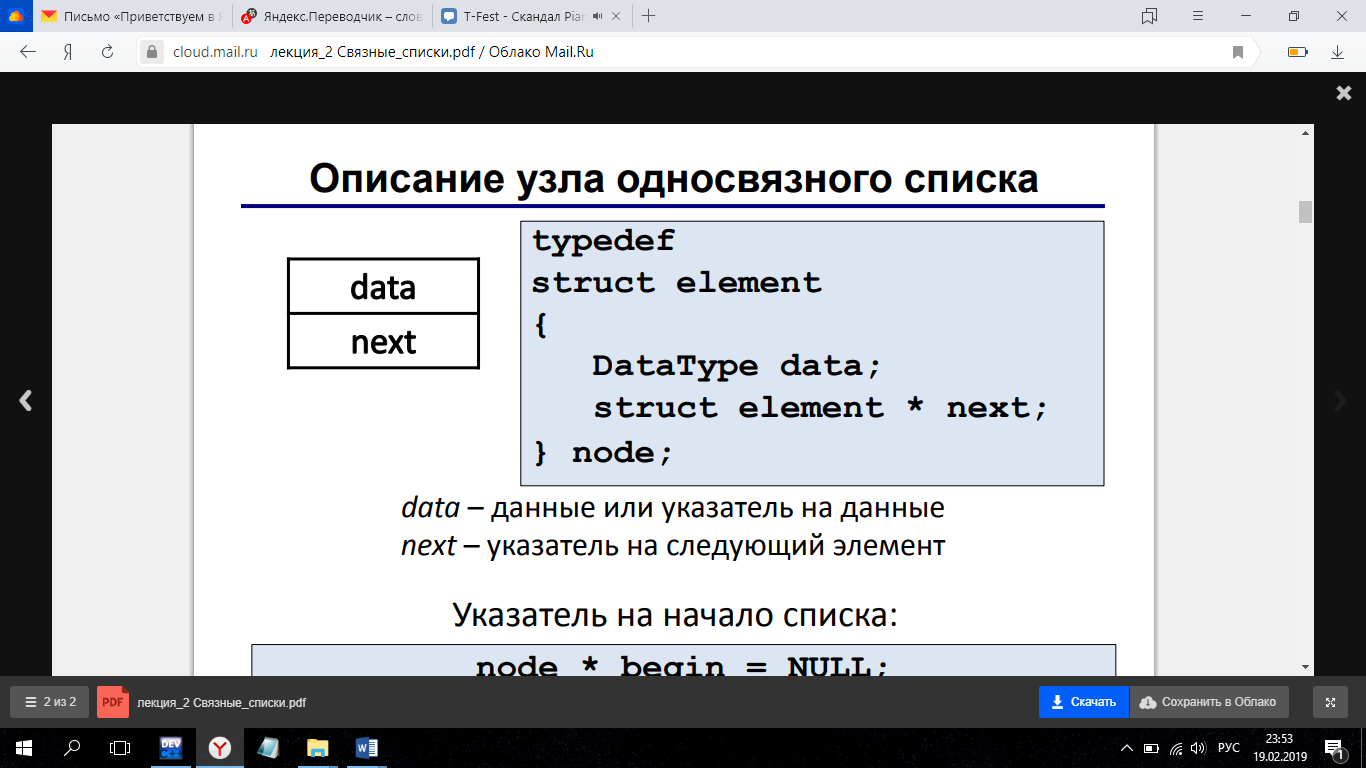
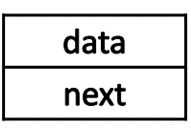
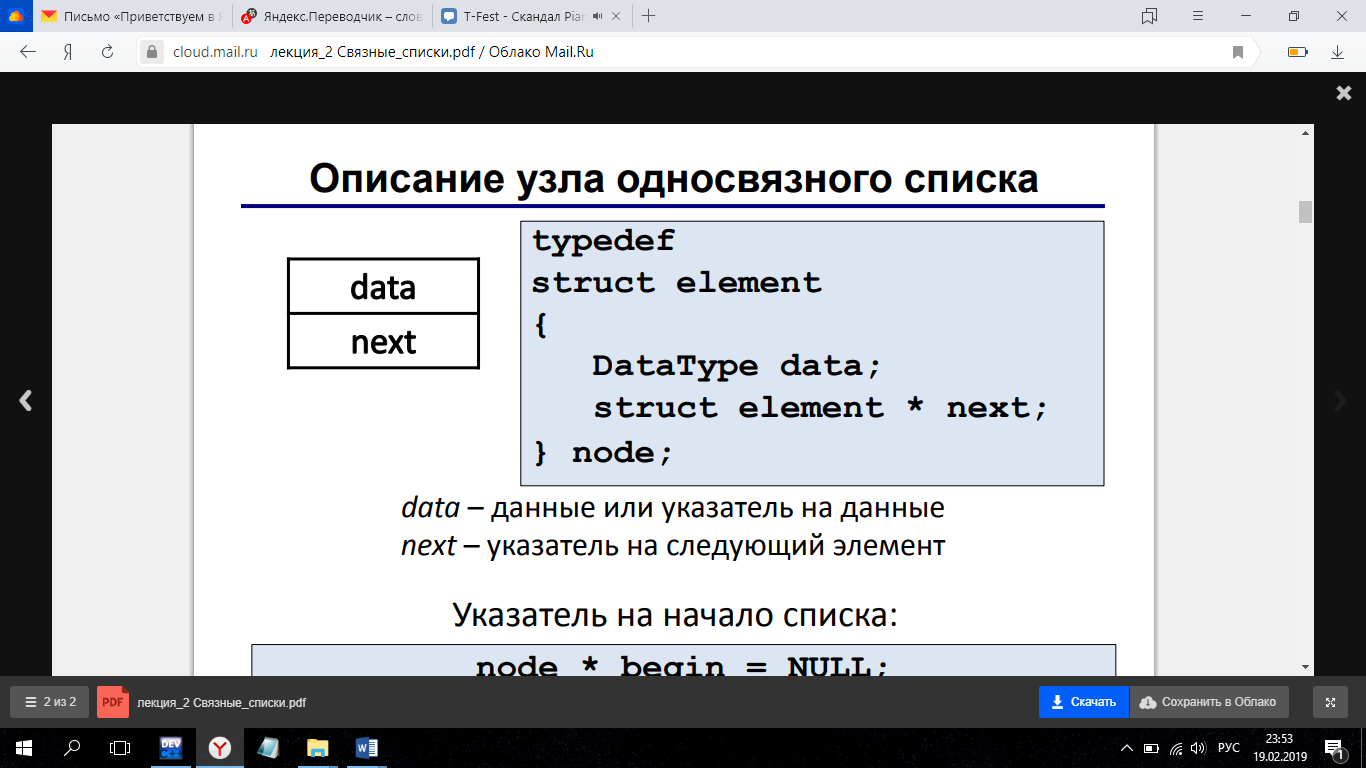
**

Рисунок 3.

*Структура данных – односвязный список:*

begin

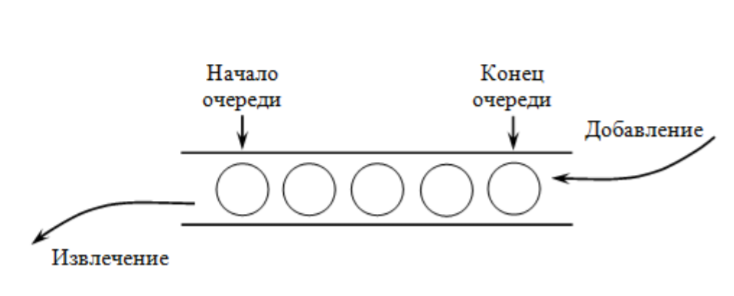
  

NULL

*Задание 2:*

(Уровень сложности – повышенный.) Многочлены вида , где *e*1>*e*2>…>*e*n≥0, можно представить в виде очереди, где каждый элемент имеет два поля: одно – для коэффициента *ci*, второе – для показателя степени *ei*. Для описанного представления многочленов напишите программу их дифференцирования.

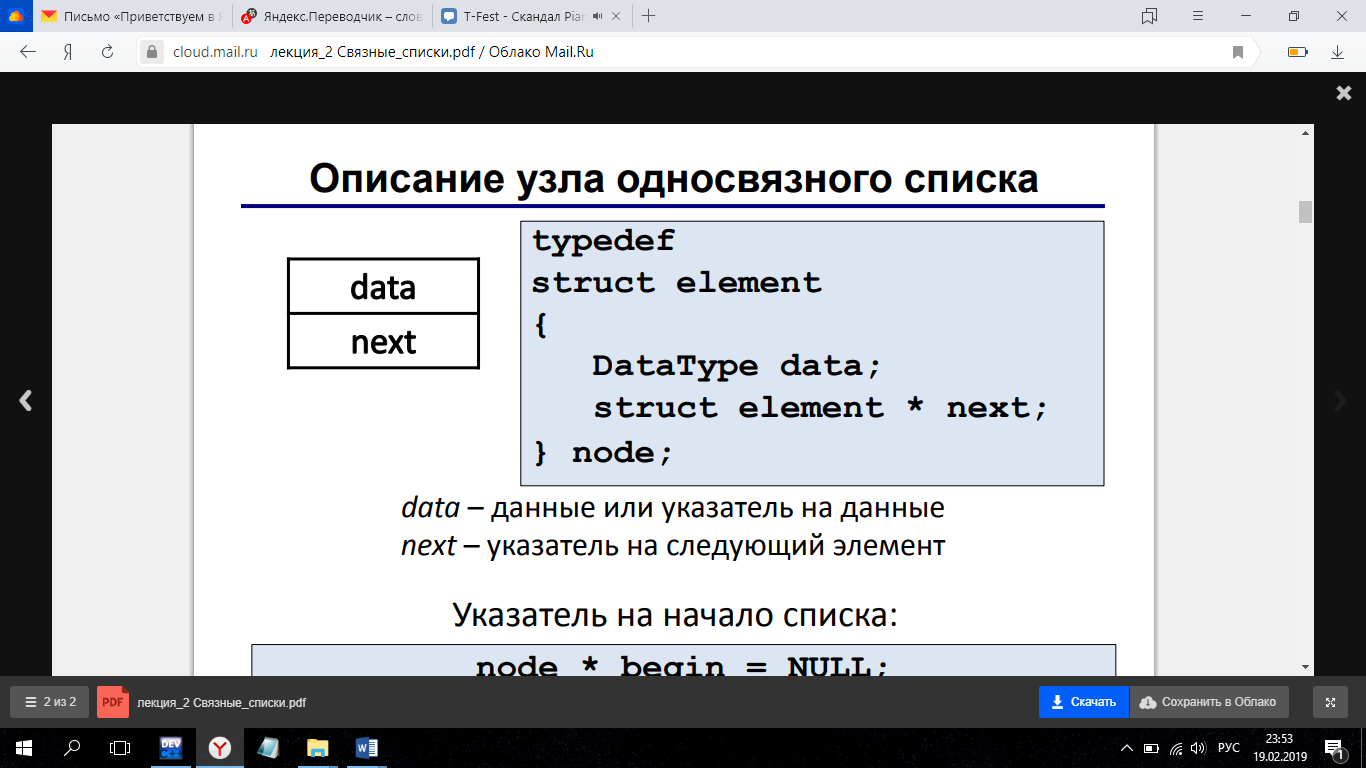
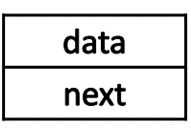
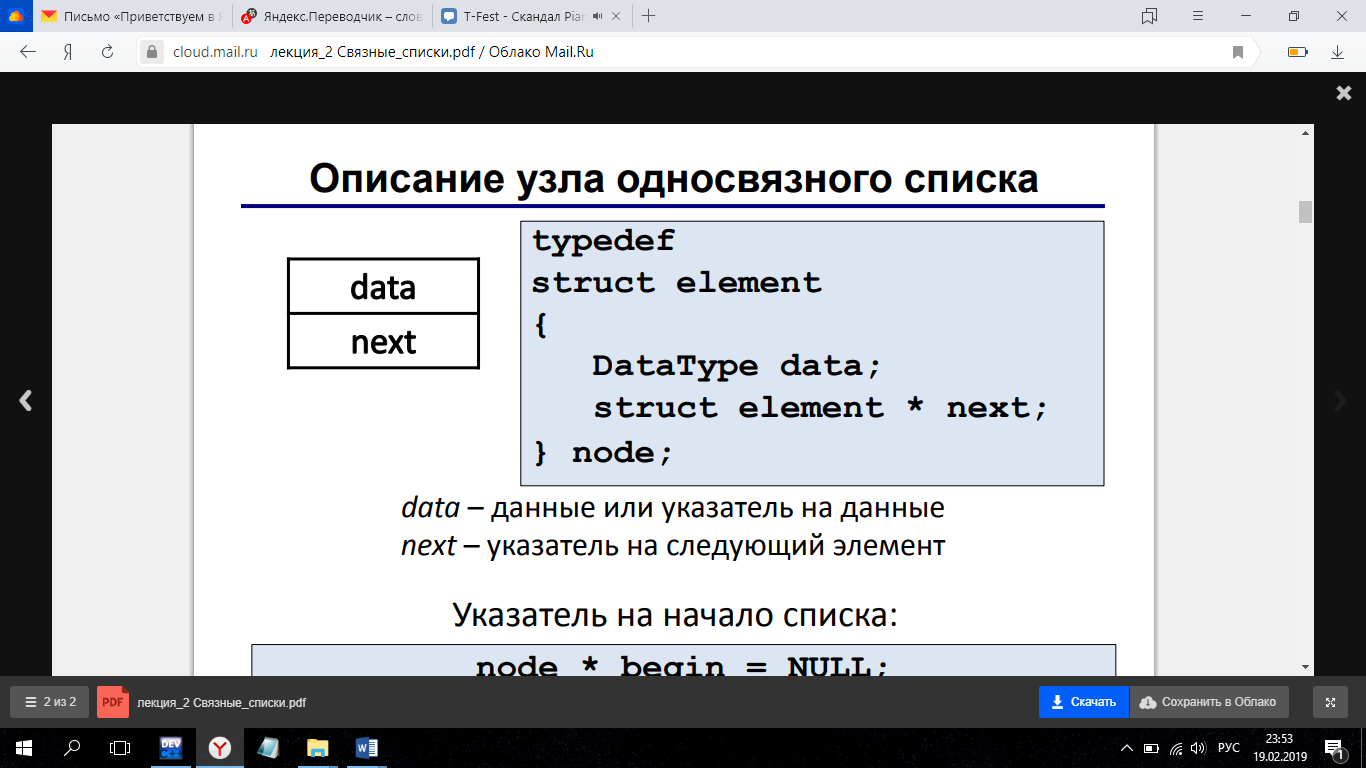
**Структура данных - очередь**



*Структура данных – очередь, структура хранения – списковая.*

end

begin

NULL

*Описание класса Queue:*  
Поля класса:

element \*begin; //указатель на первый элемент очереди

element \*end; //указатель на последний элемент очереди

Методы класса:

Queue(){ // обнуление указателей (конструктор класса);

begin = end = nullptr;

}

~Queue(){ // освобождение выделенной памяти (деструктор класса);

element \*temp;

while (begin)

{

temp = begin;

begin = begin->next;

delete temp;

}

}

void Queue::InQueue(DataType nums){ //Добавление элемента в конец очереди

element \*temp = new (nothrow) element;

if (temp == nullptr) throw Myexception("Ошибка выделения памяти", 2);

temp->num = nums;

temp->next = nullptr;

if(Empty())

begin = end = temp;

else

{

end->next = temp;

end = end->next;

}

}

DataType Queue::OutQueue(void){ //Извлечение элемента с начала очереди

DataType temp = begin->num;

element \*temp1 = begin;

begin = begin->next;

delete temp1;

return temp;

}

DataType Queue::read(void) //Неразрушающие чтение с головы

{

return begin->num;

}

int Queue::Full(void) //Проверка на полноту

{

element \*temp;

temp = new (nothrow) element;

if (temp == nullptr) return 1;

delete temp;

return 0;

}

int Queue::Empty(void) //Проверка на пустоту

{

return begin == nullptr; // по сути вставляем bool выражение

}

*Структура данных – очередь, структура хранения – векторная.*

Заполненная очередь

data

data

data

data

data

rear

front

data

data

data

front

rear

Пустая очередь.

front

rear

*Описание класса Queue:*

Поля класса:

int maxlenght; //константа, максимальное количество элементов

DataType \*data //указатель на начало массива

int front; //индекс начала массива

int rear; //индекс конца массива

Методы класса:

Queue(int i = 10) : maxlenght(i) //Определение индекса начала и конца массива,

{ выделение памяти под массив(конструктор класса)

data = new DataType[maxlenght];

front = 0;

rear = maxlenght - 1;

};

~Queue() { delete data; } // освобождение выделенной памяти (деструктор класса)

bool Queue::Empty(void) //Проверка на пустоту

{

return (rear + 1) % maxlenght == front;

}

bool Queue::Full(void) //Проверка на полноту

{

return (rear + 2) % maxlenght == front;

}

int Queue::InQueue(DataType x) //Добавление элемента в конец очереди

{

if(Full()) return 0;

rear = (rear + 1) % maxlenght;

data[rear] = x;

return 1;

}

DataType Queue::OutQueue(void) //Извлечение элемента с начала очереди

{

int temp = front;

front = (front + 1) % maxlenght;

return data[temp];

}

DataType Queue::read(void) //Неразрушающие чтение с головы

{

return data[front];

}

main.cpp

#include "queue\_array.h"

#include <iostream>

#include <locale.h>

#include <conio.h>

using namespace std;

int main()

{

try {

setlocale(LC\_ALL, "rus");

char c=0;

cout << "Init/ESC to exit";

Queue obj;

do

{

c = \_getch();

if (c == 27) break;

int n;

double flag;

cout.unsetf(cout.showpos);

cout << "\nВведите количество членов многочленна: ";

if (!(cin >> n)) throw Myexception("Ошибка ввода", 1);

if (n <= 0) continue;

if (n >= 10)

{

cout << "mNoGo" << endl;

n = 9;

}

for (int i = 1; i <= n; i++)

{

DataType temp;

cout << "x" << i<< ":" << "c" << i << " = ";

if (!(cin >> temp.c)) throw Myexception("Ошибка ввода", 1);

cout << " e" << i << " = ";

if (!(cin >> temp.e)) throw Myexception("Ошибка ввода", 1);

if (obj.Empty()) {

while (temp.e < 0)

{

cout << "Степень не соотвествует условию e1>e2>...en>=0" << endl;

cout << " e" << i << " = ";

if (!(cin >> temp.e)) throw Myexception("Ошибка ввода", 1);

}

}

else {

while (temp.e < 0 || temp.e >= flag)

{

cout << "Степень не соотвествует условию e1>e2>...en>=0" << endl;

cout << " e" << i << " = ";

if (!(cin >> temp.e)) throw Myexception("Ошибка ввода", 1);

obj.push(t

}

}

cout << "P(x)' = ";

bool flag1=true;emp);

flag = temp.e;

if (temp.e == 0) break;

}

for (int i = 1; !obj.Empty(); i++)

{

DataType temp = obj.pop();

if (temp.e == 0 || temp.c == 0);

else if (temp.e == 1)

{

cout << temp.c\*temp.e;

}

else

{

if (temp.e == 2) cout << temp.c\*temp.e << "\*x";

else {

cout << temp.c\*temp.e << "\*x" << "^";

cout.unsetf(cout.showpos);

cout << temp.e - 1;

cout.setf(cout.showpos);

}

}

if (i == 1) cout.setf(cout.showpos);

}

} while (true);

cout << "\nВышли";

\_getch();

return 0;

}

catch (Myexception &e) {

cout << "Error: " << e.GetErr() << endl;

\_getch();

return e.GetValue();

}

}

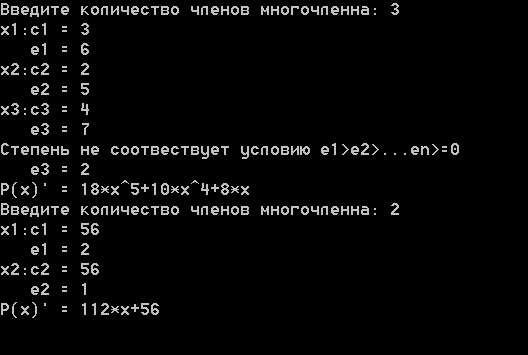


Рисунок 4 - результат работы программы.