

Министерство образования и науки РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**ДИНАМИЧЕСКОЕ СОЗДАНИЕ ЭКЗЕМПЛЯРОВ.  
ИСПОЛЬЗОВАНИЕ ОПЕРАЦИЙ NEW, DELETE.**

Лабораторная работа № 3  
по курсу «Объектно-ориентированное программирование»

Вариант № 11

|           |                              |           |              |
|-----------|------------------------------|-----------|--------------|
| Выполнил: | студент группы 220601        | _____     | Белым А.А.   |
|           |                              | (подпись) |              |
| Проверил: | к. ф.-м. н., доцент каф. АТМ | _____     | Середин О.С. |
|           |                              | (подпись) |              |

Тула 2013

## Цель работы

Научиться использовать в программах операции new, delete. Написать код, использующий эти операции в объектно-ориентированной программе.

## Задание

а) Выполнить лабораторную работу №2, задание которой приведено ниже, используя операторы `new` и `delete` для создания и уничтожения экземпляров классов.

Используя списки, преобразовать введенный текст следующим образом. Если строка начинается с цифры, то перед ней поставить левую квадратную скобку, а в конец строки – правую квадратную скобку.

б) используя динамические массивы, выполнить задание, приведенное ниже.

Дано  $N$  действительных чисел. Преобразовать эту последовательность, расположив вначале отрицательные члены, а затем неотрицательные, с сохранением порядка чисел.

## Теоретическая справка

[illegible]

[illegible]

# 1. Задание лабораторной №2

## 1.1. Реализация класса

Ниже представлено определение класса, файл List.h:

```
#ifndef LIST_H
#define LIST_H
/*!Класс "односвязный список".*/
class List {
private:
    /*!Тип "элемент списка".*/
    struct Elem{
        char c; /*!<значение элемента списка
        Elem* next;/*!<указатель на следующий элемент
    };
    Elem *first,/*!<указатель на первый элемент списка
        *last;/*!<указатель на последний элемент списка
    mutable Elem *current;/*!<указатель на текущий элемент списка
public:
    /*!Конструктор по умолчанию.*/
    List();
    /*!Конструктор по ASCIIZ строке.
    * \param source исходная строка
    */
    List(const char* source);
    /*!Конструктор копирования.
    * \param other копируемый список
    */
    List(const List& other);
    /*!Деструктор класса.*/
    ~List();
    /*!Установка значения по ASCIIZ строке.
    * \param source исходная строка
    */
    void FromChar(const char* source);
    /*!Метод добавления символа в конец списка.
    * \param new_char добавляемый символ
    */
    void Append(char new_char);
    /*!Метод добавления символа в начало списка.
    * \param new_char добавляемый символ
    */
    void Prepend(char new_char);
    /*!Вывод списка на экран.
    */
    void Print() const;
    /*!Изменяет позицию в списке с текущего элемента
    * на следующий за ним.
    */
    void Next() const;
    /*!Изменяет позицию в списке с текущего элемента
    * на начальный элемент.
    */
    void Reset() const;
    /*!Метод получения значения текущего элемента.
    * \return значение текущего элемента.
    */
    char Value() const;
    /*!Проверяет, достигнут ли конец списка.
    */
    bool isEnd() const;
    /*!Удаляет текущий элемент.
    * Текущим становится следующий после удаляемого
    * элемент.
    */
    */
};
```

```

void Delete();
/*!Удаляет следующий после текущего элемент.*/
void DeleteNext();
/*!Вставляет новый элемент на место текущего.
 * Текущий элемент становится следующим после вставленного.
 * \param с добавляемый символ
 */
void Insert(char c);
/*!Вставляет новый элемент после текущего.
 * \param с добавляемый символ
 */
void InsertNext(char c);
/*!Очистка списка, удаление всех содержащихся в списке элементов.
 */
void Clear();
/*!Копирование другого списка.
 * \param other копируемый список
 */
void Copy(const List&);
/*!Перегрузка оператора присваивания.
 * \param other присваиваемый список
 * \return данный список
 */
List& operator=(const List& other);
};

#endif //LIST_H

```

Ниже представлена реализация класса, файл List.cpp:

```

#include "List.h"
#include "cstdlib"
#include <iostream>
using namespace std;
List::List():first(NULL),
              last(NULL),
              current(NULL)
{
}

List::List(const char* source):first(NULL),
                              last(NULL),
                              current(NULL)
{
    FromChar(source);
}

void List::Next() const{
    if(current&&current->next)
        current=current->next;
}

char List::Value() const{
    if(current)
        return current->c;
    else
        return '\\0';
}

void List::Reset() const{
    current=first;
}

bool List::isEnd() const{
    return current==last||current->next==NULL||current==NULL;
}

```

```

void List::Append(char new_char){
    Elem* new_elem=new (nothrow) Elem;
    if(new_elem==NULL){
        cerr<<"Error of memory allocation for new element!"<<endl;
        return;
    }
    if(last!=NULL)
        last->next=new_elem;
    else {
        first=new_elem;
        current=new_elem;
    }
    last=new_elem;;
    last->next=NULL;
    last->c=new_char;
}

void List::Prepend(char new_char){
    Elem* new_elem=new (nothrow) Elem;
    if(new_elem==NULL){
        cerr<<"Error of memory allocation for new element!"<<endl;
        return;
    }
    if(first==NULL){
        last=new_elem;
        current=new_elem;
    };
    new_elem->next=first;
    first=new_elem;
    first->c=new_char;
}

void List::FromChar(const char* source){
    const char* ptr=source;
    for(ptr;*ptr;ptr++){
        Append(*ptr);
    }
}

void List::Clear(){
    current=first;
    if(current==NULL);
        return;
    Elem *prev=current;
    while(!isEnd()){
        Next();
        delete prev;
        prev=current;
    }
    delete prev;
    current=first=last=NULL;
}

List::~~List(){
    Clear();
}

void List::Print() const{
    Elem* backup=current;
    Reset();
    while(!isEnd()){
        std::cout<<Value();
        Next();
    }
    std::cout<<Value()<<std::endl;
    current=backup;
}

```

```

void List::Delete() {
    if(!current)
        return;
    Elem* tmp_next=current->next;
    if(tmp_next){
        current->c=current->next->c;
        current->next=current->next->next;
        delete tmp_next;
    } else {
        if(current==first) {
            first=NULL;
            last=NULL;
            current=NULL;
            delete tmp_next;
        } else {
            Reset();
            while(current->next&&current->next->next)
                Next();
            current->next=NULL;
            last=current;
            delete tmp_next;
        }
    }
}

void List::DeleteNext() {
    if(!current)
        return;
    Elem* tmp_next=current->next;
    if(tmp_next){
        current->next=current->next->next;
        delete tmp_next;
    } else
        current->next=NULL;
}

void List::Insert(char c) {
    if(current==NULL)
        Prepend(c);
    Elem* new_elem=new (nothrow) Elem;
    if(new_elem==NULL)
        std::cout<<"malloc error"<<std::endl;
    new_elem->c=current->c;
    new_elem->next=current->next;
    current->c=c;
    current->next=new_elem;
    if(current==last)
        last=new_elem;
}

void List::InsertNext(char c) {
    if(current==NULL||current==last)
        Append(c);
    Elem* new_elem=new (nothrow) Elem;
    if(new_elem==NULL)
        return;
    new_elem->c=c;
    new_elem->next=current->next;
    current->next=new_elem;
}

void List::Copy(const List& other) {
    if(this==&other)
        return;
    Clear();
    Elem* backup=other.current;

```

```

        other.Reset();
        while(!other.isEnd()){
            Append(other.Value());
            other.Next();
        }
        Append(other.Value());
        other.current=backup;
    };

List::List(const List& other):first(NULL),
                                last(NULL),
                                current(NULL)
{
    Copy(other);
}

List& List::operator=(const List& other){
    Copy(other);
    return *this;
}

```

## 1.2. Демонстрационная программа

Далее приводится демонстрационная программа, файл lab31.cpp:

```

#include "List.h"
#include <iostream>
#include <string>
#include <cctype>

int main(){
    std::string str;
    std::cout<<"Введите любую строку\n";
    std::cin>>str;

    List *a=new List(str.c_str());

    if(isdigit(a->Value())){
        a->Prepend('[');
        a->Append(']');
    };
    a->Print();
    delete a;
    return 0;
}

```

## 1.3. Инструкция программисту

Далее приводится описание функций, методов, типов данных и классов.

### **Структура List::Elem**

Тип "элемент списка".

#### **Открытые атрибуты**

- **char c**  
*значение элемента списка*
- **Elem \* next**  
*указатель на следующий элемент*



## Класс List

Класс "односвязный список".

```
#include <List.h>
```

### Классы

- struct **Elem**

### Открытые члены

- **List** ()
- **List** (const char \*source)
- **List** (const **List** &other)
- **~List** ()
- void **FromChar** (const char \*source)
- void **Append** (char new\_char)
- void **Prepend** (char new\_char)
- void **Print** () const
- void **Next** () const
- void **Reset** () const
- char **Value** () const
- bool **isEnd** () const
- void **Delete** ()
- void **DeleteNext** ()
- void **Insert** (char c)
- void **InsertNext** (char c)
- void **Clear** ()
- void **Copy** (const **List** &)
- **List** & **operator=** (const **List** &other)

### Закрытые данные

- **Elem \* first**  
*указатель на первый элемент списка*
- **Elem \* last**  
*указатель на последний элемент списка*
- **Elem \* current**  
*указатель на текущий элемент списка*

### Конструкторы

#### **List::List** ()

Конструктор по умолчанию.

### **List::List (const char \* *source*)**

Конструктор по ASCIIZ строке.

Аргументы конструктора представлены в таблице 1.1.

#### **Аргументы:**

Таблица 1.1 - Аргументы конструктора char\*

|               |                 |
|---------------|-----------------|
| <i>source</i> | исходная строка |
|---------------|-----------------|

### **List::List (const List & *other*)**

Конструктор копирования.

Аргументы конструктора представлены в таблице 1.2.

#### **Аргументы:**

Таблица 1.2 - Аргументы конструктора копирования

|              |                   |
|--------------|-------------------|
| <i>other</i> | копируемый список |
|--------------|-------------------|

### **List::~~List ()**

Деструктор класса.

## **Методы**

### **void List::Append (char *new\_char*)**

Метод добавления символа в конец списка.

Аргументы метода представлены в таблице 1.3.

#### **Аргументы:**

Таблица 1.3 – Аргументы метода добавления в конец списка

|                 |                    |
|-----------------|--------------------|
| <i>new_char</i> | добавляемый символ |
|-----------------|--------------------|

### **void List::Clear ()**

Очистка списка, удаление всех содержащихся в списке элементов.

### **void List::Copy (const List & *other*)**

Копирование другого списка.

Аргументы метода представлены в таблице 1.4.

#### **Аргументы:**

Таблица 1.4 – Аргументы метода копирования списка

|              |                   |
|--------------|-------------------|
| <i>other</i> | копируемый список |
|--------------|-------------------|

### **void List::Delete ()**

Удаляет текущий элемент. Текущим становится следующий после удаляемого элемент.

### **void List::DeleteNext ()**

Удаляет следующий после текущего элемент.

**void List::FromChar (const char \* *source*)**

Установка значения по ASCIIZ строке.

Аргументы метода представлены в таблице 1.5.

**Аргументы:**

Таблица 1.5 - Аргументы метода установки значения

|               |                 |
|---------------|-----------------|
| <i>source</i> | исходная строка |
|---------------|-----------------|

**void List::Insert (char *c*)**

Вставляет новый элемент на место текущего. Текущий элемент становится следующим после вставленного.

Аргументы метода представлены в таблице 1.6.

**Аргументы:**

Таблица 1.6 - Аргументы метода вставки

|          |                    |
|----------|--------------------|
| <i>c</i> | вставляемый символ |
|----------|--------------------|

**void List::InsertNext (char *c*)**

Вставляет новый элемент после текущего.

Аргументы метода представлены в таблице 1.7.

**Аргументы:**

Таблица 1.7 - Аргументы метода вставки после текущего

|          |                    |
|----------|--------------------|
| <i>c</i> | вставляемый символ |
|----------|--------------------|

**bool List::isEnd ()**

Проверяет, достигнут ли конец списка.

**void List::Next ()**

Изменяет позицию в списке с текущего элемента на следующий за ним.

**void List::Prepend (char *new\_char*)**

Метод добавления символа в начало списка.

Аргументы метода представлены в таблице 1.8.

**Аргументы:**

Таблица 1.8 - Аргументы метода добавления в начало списка

|                 |                    |
|-----------------|--------------------|
| <i>new_char</i> | добавляемый символ |
|-----------------|--------------------|

**void List::Print ()**

Вывод списка на экран.

### **void List::Reset ()**

Изменяет позицию в списке с текущего элемента на начальный элемент.

### **char List::Value ()**

Метод получения значения текущего элемента.

#### ***Возвращает:***

значение текущего элемента.

### **List & List::operator= (const List & other)**

Перегрузка оператора присваивания.

Аргументы оператора представлены в таблице 1.9.

#### ***Аргументы:***

Таблица 1.9 - Аргументы оператора присваивания

|              |                      |
|--------------|----------------------|
| <i>other</i> | присваиваемый список |
|--------------|----------------------|

#### ***Возвращает:***

данный список.

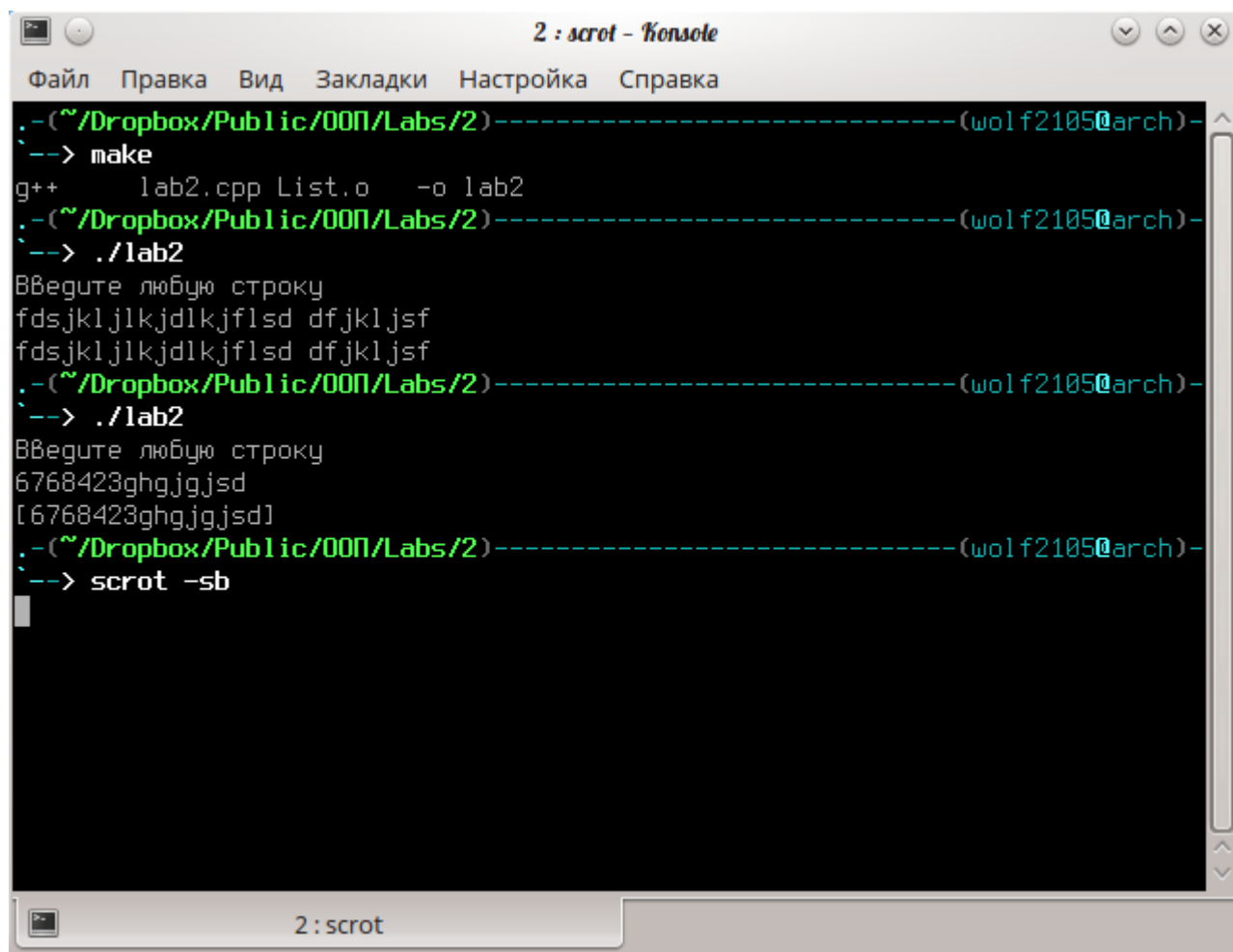
## **1.4. Инструкция пользователю**

Данная программа заключает строку в квадратные скобки, если она начинается с цифры.

Для работы программы введите произвольную строку. Если строка начинается с цифры, то она выводится заключенной в квадратные скобки; иначе она будет выведена без изменений.

## **1.5. Контрольный пример**

На рисунке 1.1 представлен пример работы программы, работающей со списком.



```
2 : scrot - Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка

~/.Dropbox/Public/00П/Labs/2)------(wolf2105@arch)-
--> make
g++    lab2.cpp List.o    -o lab2
~/.Dropbox/Public/00П/Labs/2)------(wolf2105@arch)-
--> ./lab2
Введите любую строку
fdsjkljlkjdlkjflsd dfjkljsf
fdsjkljlkjdlkjflsd dfjkljsf
~/.Dropbox/Public/00П/Labs/2)------(wolf2105@arch)-
--> ./lab2
Введите любую строку
6768423ghgjgjsd
[6768423ghgjgjsd]
~/.Dropbox/Public/00П/Labs/2)------(wolf2105@arch)-
--> scrot -sb
```

Рисунок 1.1— Пример работы программы, работающей со списком

## 2. Задание с динамическим массивом

### 2.1. Реализация класса

Ниже представлено определение класса, файл Array.h:

```
#ifndef ARRAY_H
#define ARRAY_H
/*!Класс "динамический массив".*/
class Array {
private:
    double *data;/*!<указатель-массив данных
    int size;/*!<размер массива
public:
    /*!Конструктор по умолчанию.*/
    Array();
    /*!Конструктор, выделяющий память под массив указанного размера.
    * \param len размер массива
    */
    Array(const int len);
    /*!Конструктор, создающий объект, используя указатель на массив
    * и размер массива.
    * \param new_data указатель на копируемые данные
    * \param len размер массива
    */
    Array(const double *new_data,const int len);
    /*!Конструктор копирования.
    * \param other копируемый массив
    */
    Array(const Array& other);
    /*!Деструктор класса.*/
```

```

~Array();
/*!Установка размера массива.
 * При изменении размера освобождается выделенная память,
 * и захватывается новая, что может привести к потере данных.
 * \param len размер массива
 */
void setLen(const int len);
/*!Получение текущего размера массива.
 * \return размер массива.
 */
int len();
/*!Получение ссылки на элемент массива.
 * \param n индекс элемента
 * \return ссылка на значение элемента
 */
double& valueAt(int n);
/*!Получение ссылки на элемент массива (версия для объекта-константы).
 * \param n индекс элемента
 * \return ссылка на значение элемента
 */
const double& valueAt(int n) const;
/*!Установка значения элемента массива.
 * \param n индекс элемента
 * \param value ссылка на значение элемента
 */
void setValueAt(const int n, double value);
/*!Сортировка массива, после которой следуют сначала все отрицательные
 * элементы, затем положительные.
 * Относительное положение среди положительных и отрицательных
 * элементов сохраняется такое же, как и в исходном массиве.*/
void strangeSort();
/*!Вывод массива на экран.*/
void Print() const;
/*!Копирование элементов другого массива по ссылке.
 * \param other копируемый массив
 */
void Copy(const Array& other);
/*!Перегруженный копирующий оператор присваивания.
 * \param other присваиваемый массив
 * \return данный список
 */
Array& operator=(const Array& other);
/*!Перегруженный оператор индексации - получение значения элемента
 * массива.
 * \param n индекс элемента
 * \return ссылка на значение элемента
 */
double& operator[](const int n);
/*!Перегруженный оператор индексации - получение значения элемента
 * массива (версия для объекта-константы).
 * \param n индекс элемента
 * \return ссылка на значение элемента
 */
const double& operator[](const int n) const;
};
#endif //ARRAY_H

```

Ниже представлена реализация класса, файл Array.cpp:

```

#include "Array.h"
#include <iostream>
#include <cstdlib>
using namespace std;
Array::Array():data(NULL),
               size(0)
{

```

```

}

Array::Array(const int len){
    if(len>0){
        size=len;
        try{
            data=new double[size];
        } catch (bad_alloc) {
            cerr<<"Memory allocation error!"<<endl;
            size=0;
            data=NULL;
        }
    } else {
        size=0;
        data=NULL;
    }
}

Array::Array(const double *new_data,const int len){
    if(len>0){
        size=len;
        try{
            data=new double[size];
        } catch (bad_alloc) {
            cerr<<"Memory allocation error!"<<endl;
            size=0;
            data=NULL;
        }
        for(int i=0;i<size;i++){
            data[i]=new_data[i];
        }
    } else {
        size=0;
        data=NULL;
    }
}

Array::~Array(){
    delete[] data;
    size=0;
}

void Array::setLen(const int len){
    if(len>0){
        delete[] data;
        try{
            data=new double[size];
        } catch (bad_alloc) {
            cerr<<"Memory allocation error!"<<endl;
            size=0;
            data=NULL;
        }
        size=len;
    }
}

int Array::len(){
    return size;
}

double& Array::valueAt(int n){
    if(data&& n>=0&& n<size)
        return data[n];
    else
        cerr<<"Index out of range!"<<endl;
    //else
    //    return 0;
}

```

```

const double& Array::valueAt(int n) const{
    if(data&&n>=0&&n<size)
        return data[n];
    else
        cerr<<"Index out of range!"<<endl;
    //else
    //    return 0;
}

void Array::setValueAt(const int n, double value){
    if(data&&n>=0&&n<size)
        data[n]=value;
    else
        cerr<<"Index out of range!"<<endl;
}

void Array::strangeSort(){
    for(int i=size-1;i>0;--i){
        for(int j=0;j<i;++j){
            if(data[j]>=0&&data[j+1]<0){
                double temp=data[j];
                data[j]=data[j+1];
                data[j+1]=temp;
            }
        }
    }
}

void Array::Print() const{
    for(int i=0;i<size-1;++i){
        cout<<data[i]<<" ";
    };
    cout<<data[size-1]<<endl;
}

void Array::Copy(const Array& other){
    if(this==&other)
        return;
    delete[] data;
    size=other.size;
    try{
        data=new double[size];
    } catch (bad_alloc) {
        cerr<<"Memory allocation error!"<<endl;
        size=0;
        data=NULL;
        return;
    }
    for(int i=0;i<size;i++)
        data[i]=other.data[i];
}

Array& Array::operator=(const Array& other){
    Copy(other);
    return *this;
}

double& Array::operator[] (int n){
    valueAt(n);
}

const double& Array::operator[] (int n) const{
    valueAt(n);
}

```



## 2.2. Демонстрационная программа

Далее приводится демонстрационная программа, файл lab32.cpp:

```
#include "Array.h"
#include <iostream>
using namespace std;
int main() {
    int len;
    cout<<"Введите размер массива."<<endl;
    cin>>len;
    Array a(len);
    if(len<0)
        return 1;

    for(int i=0;i<len;++i){
        cout<<"Введите число номер "<<i<<". "<<endl;
        cin>>a[i];
    }

    a.strangeSort();
    a.Print();
    return 0;
}
```

## 2.3. Инструкция программисту

Далее приводится описание функций, методов, типов данных и классов.

### *Класс Array*

Класс "динамический массив".

```
#include <Array.h>
```

### Открытые члены

- **Array ()**
- **Array (const int len)**
- **Array (const double \*new\_data, const int len)**
- **Array (const Array &other)**
- **~Array ()**
- **void setLen (const int len)**
- **int len ()**
- **double & valueAt (int n)**
- **const double & valueAt (int n) const**
- **void setValueAt (const int n, double value)**
- **void strangeSort ()**
- **void Print () const**
- **void Copy (const Array &other)**
- **Array & operator= (const Array &other)**
- **double & operator[] (const int n)**
- **const double & operator[] (const int n) const**

## Закрытые данные

- `double * data`  
*указатель-массив данных*
- `int size`  
*размер массива*

## Конструктор(ы)

### **Array::Array ()**

Конструктор по умолчанию.

### **Array::Array (const int len)**

Конструктор, выделяющий память под массив указанного размера.

Аргументы конструктора представлены в таблице 2.1.

#### **Аргументы:**

Таблица 2.1 - Аргументы конструктора `const int`

|            |                |
|------------|----------------|
| <i>len</i> | размер массива |
|------------|----------------|

### **Array::Array (const double \* new\_data, const int len)**

Конструктор, создающий объект, используя указатель на массив и размер массива.

Аргументы конструктора представлены в таблице 2.2.

#### **Аргументы:**

Таблица 2.2 - Аргументы конструктора (`const double*`, `const int`)

|                 |                                |
|-----------------|--------------------------------|
| <i>new_data</i> | указатель на копируемые данные |
| <i>len</i>      | размер массива                 |

### **Array::Array (const Array & other)**

Конструктор копирования.

Аргументы конструктора представлены в таблице 2.3.

#### **Аргументы:**

Таблица 2.3 - Аргументы конструктора копирования

|              |                   |
|--------------|-------------------|
| <i>other</i> | копируемый массив |
|--------------|-------------------|

### **Array::~Array ()**

Деструктор класса.

---

## Методы

### **void Array::Copy (const Array & other)**

Копирование элементов другого массива по ссылке.

Аргументы метода представлены в таблице 2.4.

**Аргументы:**

Таблица 2.4 - Аргументы метода копирования

|              |                   |
|--------------|-------------------|
| <i>other</i> | копируемый массив |
|--------------|-------------------|

**int Array::len ()**

Получение текущего размера массива.

**Возвращает:**

размер массива.

**Array & Array::operator= (const Array & other)**

Перегруженный копирующий оператор присваивания.

Аргументы оператора представлены в таблице 2.5.

**Аргументы:**

Таблица 2.5 - Аргументы оператора присваивания

|              |                      |
|--------------|----------------------|
| <i>other</i> | присваиваемый массив |
|--------------|----------------------|

**Возвращает:**

данный список

**double & Array::operator[] (const int n)**

Перегруженный оператор индексации - получение значения элемента массива.

Аргументы оператора представлены в таблице 2.6.

**Аргументы:**

Таблица 2.6 - Аргументы оператора индексации

|          |                 |
|----------|-----------------|
| <i>n</i> | индекс элемента |
|----------|-----------------|

**Возвращает:**

ссылка на значение элемента

**const double & Array::operator[] (const int n) const**

Перегруженный оператор индексации - получение значения элемента массива(версия для объекта-константы).

Аргументы оператора представлены в таблице 2.7.

**Аргументы:**

Таблица 2.7 - Аргументы оператора индексации для констант

|          |                 |
|----------|-----------------|
| <i>n</i> | индекс элемента |
|----------|-----------------|

**Возвращает:**

ссылка на значение элемента

**void Array::Print () const**

Вывод массива на экран.

### **void Array::setLen (const int *len*)**

Установка размера массива. При изменении размера освобождается выделенная память, и захватывается новая, что может привести к потере данных.

Аргументы метода представлены в таблице 2.8.

#### **Аргументы:**

Таблица 2.8 - Аргументы метода установки длины массива

|            |                |
|------------|----------------|
| <i>len</i> | размер массива |
|------------|----------------|

### **void Array::setValueAt (const int *n*, double *value*)**

Установка значения элемента массива.

Аргументы метода представлены в таблице 2.9.

#### **Аргументы:**

Таблица 2.9 - Аргументы метода установки значения массива

|              |                             |
|--------------|-----------------------------|
| <i>n</i>     | индекс элемента             |
| <i>value</i> | ссылка на значение элемента |

### **void Array::strangeSort ()**

Сортировка массива, после которой следуют сначала все отрицательные элементы, затем положительные. Относительное положение среди положительных и отрицательных элементов сохраняется такое же, как и в исходном массиве.

### **double & Array::valueAt (int *n*)**

Получение ссылки на элемент массива.

Аргументы метода представлены в таблице 2.10.

#### **Аргументы:**

Таблица 2.10 - Аргументы метода получения значения

|          |                 |
|----------|-----------------|
| <i>n</i> | индекс элемента |
|----------|-----------------|

#### **Возвращает:**

ссылку на значение элемента.

### **const double & Array::valueAt (int *n*) const**

Получение ссылки на элемент массива(версия для объекта-константы).

Аргументы метода представлены в таблице 2.11.

#### **Аргументы:**

Таблица 2.11 - Аргументы метода получения значения для константы

|          |                 |
|----------|-----------------|
| <i>n</i> | индекс элемента |
|----------|-----------------|

#### **Возвращает:**

ссылку на значение элемента.

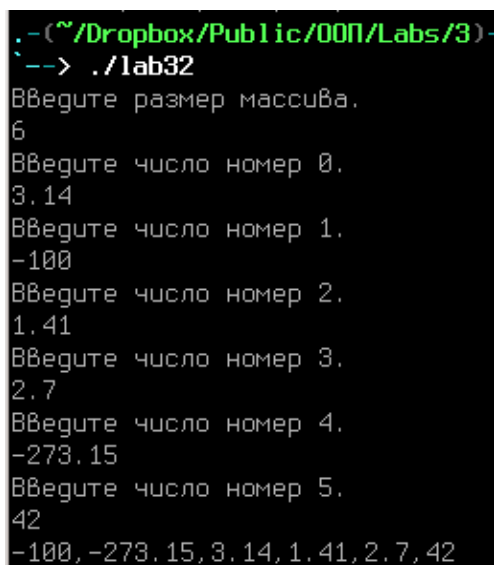
## 2.4. Инструкция пользователю

Данная программа позволяет отсортировать массив следующим образом: вначале располагаются отрицательные члены, а затем неотрицательные, с сохранением порядка чисел относительно исходного массива.

Для работы введите размер исходного массива. После введите столько же элементов, сколько было указано при вводе размера массива. После ввода последнего числа программа отсортирует массив и выведет его на экран.

## 2.5. Контрольный пример

На рисунке 2.1 представлен пример работы программы, работающей с динамическим массивом.



```
.- (~ /Dropbox/Public/ООП/Labs/3) -  
--> ./lab32  
Введите размер массива.  
6  
Введите число номер 0.  
3.14  
Введите число номер 1.  
-100  
Введите число номер 2.  
1.41  
Введите число номер 3.  
2.7  
Введите число номер 4.  
-273.15  
Введите число номер 5.  
42  
-100, -273.15, 3.14, 1.41, 2.7, 42
```

Рисунок 2.1— Пример работы программы, работающей с динамическим массивом

## Вывод

В данной лабораторной работе я изучил операторы `new`, `delete` и `delete[]` языка C++, а также написал программы, которые используют эти операции для создания и уничтожения одиночных объектов и массивов.