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

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

ШАБЛОНЫ КЛАССОВ И ФУНКЦИЙ

Лабораторная работа № 6
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

Изучить понятие шаблонов функций и классов. Написать программу на C++ с использованием этого принципа.

Задание

Создать класс для работы с целыми числами. Используя шаблоны, выполнить то же задание для вещественных чисел.

Из последовательности 2*n целых чисел получить число min{a(1)+a(n+1),a(2)+a(n+2),...,a(n)+a(2n)}.

Теоретическая справка

Реализация класса

Ниже представлено определение класса, файл Array.h:

```
#ifndef ARRAY_H
#define ARRAY_H
/*!Класс "динамический массив".*/
template <typename T>
class Array {
    private:
        T *data;//!<указатель-массив данных
        int size;//!<размер массива
    public:
        /*!Конструктор по умолчанию.*/
        Array();
        /*!Конструктор, выделяющий память под массив указанного размера.
         * \param len размер массива
         */
        explicit 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 T *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
         */
        T& valueAt(int n);
        /*!Получение ссылки на элемент массива(версия для объекта-константы).
         * \param n индекс элемента
         * \return ссылка на значение элемента
         */
        const T& valueAt(int n) const;
        /*!Установка значения элемента массива.
         * \param n индекс элемента
         * \param value ссылка на значение элемента
         */
        void setValueAt(const int n, T value);
        /*! Получение из первых 2*n чисел (n - размер массива, целочисленно
           деленый пополам) данного массива (a) значения, равного
           min{a(1)+a(n+1),a(2)+a(n+2),...,a(n)+a(2n)} .
         * \return вычисленное значение
         */
        T findMin();
        /*!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);
        /*!Перегруженый оператор индексации - получение значения элемента
         * массива.
         * \param n индекс элемента
         * \return ссылка на значение элемента
         */
        T& operator[](const int n);
        /*!Перегруженый оператор индексации - получение значения элемента
         * массива(версия для объекта-константы).
         * \param n индекс элемента
         * \return ссылка на значение элемента
         */
        const T& operator[](const int n) const;
};
#include "Array.cpp"
#endif //ARRAY_H
```

Ниже представлена реализация класса, файл Array.cpp:

```
#include <iostream>
#include <cstdlib>
//#include "Array.h"
using namespace std;
template <typename T>
Array<T>::Array():data(NULL),
                  size(0)
{
}

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

template <typename T>
Array<T>::Array(const T *new_data,const int len){
    if(len>0){
        size=len;
        try{
            data=new T[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;
    }
}

template <typename T>
Array<T>::~Array(){
    delete[] data;
    size=0;
}

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

template <typename T>
int Array<T>::len(){
    return size;
}

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

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

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

template <typename T>
T Array<T>::findMin(){
    int n=size/2;
    T min=data[0]+data[n],s;
    for(int i=1;i<n;++i){
        s=data[i]+data[i+n];
        if(s<min)
```

```
            min=s;
    }
    return min;
}

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

template <typename T>
void Array<T>::Copy(const Array& other){
    if(this==&other)
        return;
    delete[] data;
    size=other.size;
    try{
            data=new T[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];
}

template <typename T>
Array<T>& Array<T>::operator=(const Array& other){
    Copy(other);
    return *this;
}

template <typename T>
T& Array<T>::operator[] (int n){
    valueAt(n);
}

template <typename T>
const T& Array<T>::operator[] (int n) const{
    valueAt(n);
}
```

Демонстрационная программа

Далее приводится демонстрационная программа, файл lab6.cpp:

```
#include "Array.h"
#include <iostream>
template <typename T>
void testTemplate(const char *name,const char* type){
    int len;
    cout<<"Введите размер массива "<<name<<"."<<endl;
    cout<<"(Тип элемента - "<<type<<" число)"<<endl;
    cin>>len;
    Array<T> a(len);
    if(len<0)
        return;

    for(int i=0;i<len;++i){
```

```
            cout<<"Введите "
                <<type
                <<" число номер "
                <<i
                <<" в массиве "
                <<name<<"."<<endl;
            cin>>a[i];
        }
        std::cout<<name<<".findMin()=";
        std::cout<<a.findMin()<<endl;
}

int main(){
        testTemplate<int>("a","целое");
        testTemplate<double>("b","вещественное");
        return 0;
}
```

Инструкция программисту

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

Шаблон класса Array< T >

template<typename T>class Array< T >

Класс "динамический массив".

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

Открытые члены

- **Array** ()
- **Array** (const int **len**)
- **Array** (const T *new_data, const int **len**)
- **Array** (const **Array** &other)
- **~Array** ()
- void **setLen** (const int **len**)
- int **len** ()
- T & **valueAt** (int n)
- const T & **valueAt** (int n) const
- void **setValueAt** (const int n, T value)
- T **findMin** ()
- void **Print** () const
- void **Copy** (const **Array** &other)
- **Array** & **operator=** (const **Array** &other)
- T & **operator[]** (const int n)
- const T & **operator[]** (const int n) const

Закрытые данные

- T * **data**

указатель-массив данных

- int **size**

размер массива

Конструктор(ы)

template<typename T > Array< T >::Array ()

Конструктор по умолчанию.

template<typename T > Array< T >::Array (const int *len*) [explicit]

Конструктор, выделяющий память под массив указанного размера.

Аргументы конструктора представлены в таблице 1.

Аргументы:

Таблица 1 - Аргументы конструктора const int

len	размер массива

template<typename T > Array< T >::Array (const T * *new_data*, const int *len*)

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

Аргументы конструктора представлены в таблице 2.

Аргументы:

Таблица 2 - Аргументы конструктора (const T*,cons t int)

new_data	указатель на копируемые данные
len	размер массива

template<typename T > Array< T >::Array (const Array< T > & *other*)

Конструктор копирования.

Аргументы конструктора представлены в таблице 3.

Аргументы:

Таблица 3 - Аргументы конструктора копирования

other	копируемый массив

template<typename T > Array< T >::~Array ()

Деструктор класса.

Методы

template<typename T > void Array< T >::Copy (const Array< T > & *other*)

Копирование элементов другого массива по ссылке.

Аргументы метода представлены в таблице 4.

Аргументы:

Таблица 4 - Аргументы метода копирования

other	копируемый массив

template<typename T > T Array< T >::findMin ()

Получение из первых 2*n чисел (n - размер массива, целочисленно деленый пополам) данного массива (a) значения, равного min{a(1)+a(n+1),a(2)+a(n+2),...,a(n)+a(2n)} .

Возвращает:

вычисленное значение

template<typename T > int Array< T >::len ()

Получение текущего размера массива.

Возвращает:

размер массива.

template<typename T > Array< T > & Array< T >::operator= (const Array< T > & *other*)

Перегруженный копирующий оператор присваивания.

Аргументы оператора представлены в таблице 5.

Аргументы:

Таблица 5 - Аргументы оператора присваивания

other	присваиваемый массив

Возвращает:

данный список

template<typename T > T & Array< T >::operator[] (const int *n*)

Перегруженный оператор индексации - получение значения элемента массива.

Аргументы оператора представлены в таблице 6.

Аргументы:

Таблица 6 - Аргументы оператора индексации

n	индекс элемента

Возвращает:

ссылка на значение элемента

template<typename T > const T & Array< T >::operator[] (const int *n*) const

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

Аргументы оператора представлены в таблице 7.

Аргументы:

Таблица 7 - Аргументы оператора индексации для констант

n	индекс элемента

Возвращает:

ссылка на значение элемента

template<typename T > void Array< T >::Print () const

Вывод массива на экран.

template<typename T > void Array< T >::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 8.

Аргументы:

Таблица 8 - Аргументы метода установки длины массива

len	размер массива

template<typename T > void Array< T >::setValueAt (const int *n*, T *value*)

Установка значения элемента массива.

Аргументы метода представлены в таблице 9.

Аргументы:

Таблица 9 - Аргументы метода установки значения массива

n	индекс элемента
value	ссылка на значение элемента

template<typename T > T & Array< T >::valueAt (int *n*)

Получение ссылки на элемент массива.

Аргументы метода представлены в таблице 10.

Аргументы:

Таблица 10 - Аргументы метода получения значения

n	индекс элемента

Возвращает:

ссылку на значение элемента.

template<typename T > const T & Array< T >::valueAt (int *n*) const

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

Аргументы метода представлены в таблице 11.

Аргументы:

Таблица 11 - Аргументы метода получения значения для константы

n	индекс элемента

Возвращает:

ссылку на значение элемента.

Инструкция пользователю

Данная программа позволяет из последовательности 2*n целых чисел получить число min{a(1)+a(n+1),a(2)+a(n+2),...,a(n)+a(2n)}.

Для работы введите размер массива целых чисел. После введите столько же элементов, сколько было указано при вводе размера массива. После ввода последнего числа программа найдет значение по указанной формуле. Далее данная процедура повторяется для вещественного массива.

Контрольный пример

На рисунке 1 представлен пример работы программы, работающей с динамическими массивами на основе шаблона.

```
ВВедите размер массива a.
(Тип элемента - целое число)
5
ВВедите целое число номер 0 в массиве a.
1
ВВедите целое число номер 1 в массиве a.
2
ВВедите целое число номер 2 в массиве a.
3
ВВедите целое число номер 3 в массиве a.
4
ВВедите целое число номер 4 в массиве a.
5
a.findMin()=4
ВВедите размер массива b.
(Тип элемента - Вещественное число)
4
ВВедите Вещественное число номер 0 в массиве b.
1.1
ВВедите Вещественное число номер 1 в массиве b.
2.2
ВВедите Вещественное число номер 2 в массиве b.
3.3
ВВедите Вещественное число номер 3 в массиве b.
4.4
b.findMin()=4.4
```

Рисунок 1— Пример работы программы, работающей с динамическими массивами

Вывод

В данной лабораторной работе я изучил шаблоны языка Си++. Была написана программа, реализующая на основе шаблона класс «динамический массив», и применяющая его для хранения целых и вещественных чисел.