Занятие 9

5.04.2021

**Задание 1**

Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.

Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368c.

Изучить паттерн проектирования Итератор (стр. 249)

**Задание 2**

**Main.cpp**

#include "ArrayReverseIterator.h"  
  
int main() {  
  
 Array<char> arr;  
 arr.Add('c');  
 arr.Add('d');  
 arr.InsertAt('a',0);  
 arr.InsertAt('b',1);  
  
  
 Array<char>::ArrayIterator iterator(&arr);  
 arr.PrintAll(iterator);  
 std::cout<<std::endl;  
 ArrayReverseIterator<char> reverse\_iterator(&arr);  
 for(reverse\_iterator.First(); !reverse\_iterator.IsDone(); reverse\_iterator.Next()) {  
 std::cout<<reverse\_iterator.CurrentItem();  
 }  
  
 return 0;  
}

Iterator.h

template<typename T>  
class Iterator {  
 virtual void First() = 0;  
 virtual void Next() = 0;  
 virtual bool IsDone() const = 0;  
 virtual T CurrentItem() const=0;  
*// protected:  
// Iterator();*};

Разработать класс-контейнер *Одномерный массив* на базе динамического Си-массива. Обращение изменение кол-во элементов

Добавить Итератор для типа данных *Одномерный массив.*

Итератор реализовать в двух вариантах:

1. вложенный класс (member type);

array.h

#include<iostream>  
#include "Iterator.h"  
  
  
template<typename T>  
class Array {  
  
 public:  
 Array() :size\_(0), data\_(nullptr) {};  
 ~Array() { delete[] data\_; }  
 Array(const Array &a);  
 Array(Array&&a);  
 Array& operator=(const Array &a);  
 Array& operator=(Array &&a);  
  
  
 T& operator[](size\_t index);  
 T Get (int index) const;  
 size\_t GetSize() const { return this->size\_; }  
 void Add(T elem);  
 void InsertAt(T elem,int index);  
 void RemoveAt(int index);  
  
  
  
 class ArrayIterator:Iterator<T> {  
 public:  
 explicit ArrayIterator(const Array<T>\* arr):array\_(arr),current\_(0){};  
 virtual void First(){  
 current\_=0;  
 }  
 void Next()override{  
 current\_++;  
 }  
 bool IsDone() const override{  
 return (current\_>=array\_->size\_);  
 }  
 T CurrentItem() const override{  
 if (IsDone()) {  
 return '\0';  
 }  
 return array\_->data\_[current\_];  
 }  
  
 private:  
 const Array<T>\* array\_;  
 size\_t current\_;  
 };  
  
 void PrintAll(ArrayIterator& start);  
  
 */\* ArrayIterator\* CreateIterator() const {  
 return new ArrayIterator(this);  
 }\*/* private:  
 size\_t size\_;  
 T\* data\_;  
};

template<typename T>  
Array<T>::Array(const Array<T> &a) {  
 for (size\_t i = 0; i < a.size\_; i++)  
 {  
 data\_[i] = a.data\_[i];  
 }  
}  
  
  
template<typename T>  
Array<T>::Array(Array<T> &&a):size\_(a.size\_),data\_(a.data\_) {  
 a.size\_ = 0;  
 a.data\_ = nullptr;  
}  
  
template<typename T>  
Array<T> & Array<T>::operator=(const Array<T> &a) {  
 if (this != &a) {  
 delete[]data\_;  
 size\_ = a.size\_;  
 data\_ = new int[size\_];  
 for (size\_t i = 0; i < a.size\_; i++)  
 {  
 data\_[i] = a.data\_[i];  
 }  
 }  
 return \*this;  
}  
  
template<typename T>  
Array<T> & Array<T>::operator=(Array<T> &&a) {  
 if (this != &a) {  
 delete[]data\_;  
 size\_ = a.size\_;  
 data\_ = a.data;  
 a.Size = 0;  
 a.data = nullptr;  
 }  
 return \*this;  
}  
  
  
template<typename T>  
T& Array<T>::operator[](size\_t index) {  
 if(index>=this->size\_)  
 {  
 std::cout << "error index\n";  
 exit(1);  
 }  
 return this->data\_[index];  
}  
  
template<typename T>  
T Array<T>::Get(int index) const {  
 return this->data\_[index];  
}  
  
template<typename T>  
void Array<T>::Add(T elem) {  
  
 int new\_size=size\_+1;  
 T\* new\_arr=new T[new\_size];  
 for(size\_t i=0;i<size\_;i++){  
 new\_arr[i]=data\_[i];  
 }  
 new\_arr[size\_]=elem;  
 delete[] data\_;  
 data\_=new\_arr;  
 size\_=new\_size;  
}  
  
template<typename T>  
void Array<T>::InsertAt(T elem, int index) {  
 int new\_size=size\_+1;  
 T\* new\_arr=new T[new\_size];  
 for(int i=0;i<index;i++){  
 new\_arr[i]=data\_[i];  
 }  
 new\_arr[index]=elem;  
 for (int i = index+1; i < new\_size; ++i) {  
 new\_arr[i]=data\_[i-1];  
 }  
 delete [] data\_;  
 data\_=new\_arr;  
 size\_=new\_size;  
}  
  
template<typename T>  
void Array<T>::RemoveAt(int index) {  
  
 int new\_size=size\_-1;  
 T \*new\_arr=new T[new\_size];  
 for(int i=0;i<index;i++){  
 new\_arr[i]=data\_[i];  
 }  
 for (int i = index+1; i < new\_size; ++i) {  
 new\_arr[i-1]=data\_[i];  
 }  
 delete[] data\_;  
 data\_=new\_arr;  
}  
  
template<typename T>  
void Array<T>::PrintAll(ArrayIterator& start) {  
 for(start.First(); !start.IsDone(); start.Next()) {  
 std::cout<<start.CurrentItem();  
 }  
}

1. внешний класс.

ArrayReverseIterator.h

#include "Iterator.h"  
#include "Array.h"  
  
template<typename T>  
class ArrayReverseIterator:public Iterator<T> {  
 public:  
  
 explicit ArrayReverseIterator(const Array<T>\* arr);  
 void First() override;  
 void Next() override;  
 bool IsDone() const override;  
 T CurrentItem() const override;  
 private:  
 const Array<T>\* array\_;  
 size\_t current\_;  
};

template<typename T>  
ArrayReverseIterator<T>::ArrayReverseIterator(const Array<T>\* arr):array\_(arr),current\_(0){}  
  
template<typename T>  
void ArrayReverseIterator<T>::First() {  
 current\_=array\_->GetSize()-1;  
}  
  
template<typename T>  
void ArrayReverseIterator<T>::Next() {  
 current\_--;  
}  
  
template<typename T>  
bool ArrayReverseIterator<T>::IsDone() const {  
 return (current\_<0 || current\_>=array\_->GetSize());  
}  
  
template<typename T>  
T ArrayReverseIterator<T>::CurrentItem() const {  
 if (IsDone()) {  
 throw std::invalid\_argument("Error");  
 }  
 return array\_->Get(current\_);  
}