ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ 5

за курсом «Програмування»

студента групи ПЗ-18-2

Чакова В. В.

кафедра математичного забезпечення ЕОМ, ДНУ

2019/2020 навч.р.

1. Постановка задачі

Розробити програму – 32-разрядний додаток для ОС Windows, що демонструє

можливості мови C++, а також його стандартної бібліотеки по роботі з потоками,

контейнерами та ітераторами.

Використання всіх вищезгаданих можливостей є обов’язковим в кожному з варіантів

індивідуальних завдань.

В кожному з нижченаведених варіантів завдань Вами повинні бути реалізовані основні

можливості бібліотеки стандартних шаблонів: ітератор(и), пошук, упорядкування, сума або

щось аналогічне зі згоди викладача.

Вариант 18:

Напишіть програму, в якій би клас „Відображення (асоціативні масиви)”

використовувався як стандартний контейнер.

2. Опис розв’язку

Для начала был описан темплейт с итераторами, затем остальные доп функции.

template < typename Key, typename T, typename Compare = std::less<Key>, typename Allocator = std::allocator< std::pair<Key, T> > >

class assoc\_array

Итераторы:

class miterator

{

public:

typedef miterator self\_type;

typedef std::pair<Key, T> value\_type;

typedef std::pair<Key, T>& reference;

typedef std::pair<Key, T>\* pointer;

typedef std::bidirectional\_iterator\_tag iterator\_category;

typedef std::ptrdiff\_t difference\_type;

miterator(pointer ptr) : ptr\_(ptr) { };

miterator() {}

self\_type operator=(const self\_type& other) { ptr\_ = other.ptr\_; return \*this; }

self\_type operator++() { ptr\_++; return \*this; }

self\_type operator--() { ptr\_--; return \*this; }

self\_type operator++(int junk) { self\_type i = \*this; ptr\_++; return i; }

self\_type operator--(int junk) { self\_type i = \*this; ptr\_--; return i; }

reference operator\*() { return \*ptr\_; }

pointer operator->() { return ptr\_; }

pointer operator&() { return ptr\_; }

bool operator==(const self\_type& rhs) { return ptr\_ == rhs.ptr\_; }

bool operator!=(const self\_type& rhs) { return ptr\_ != rhs.ptr\_; }

private:

pointer ptr\_;

};

class mconst\_iterator

{

public:

typedef mconst\_iterator self\_type;

typedef std::pair<Key, T> value\_type;

typedef std::pair<Key, T>& reference;

typedef std::pair< Key, T>\* pointer;

typedef std::ptrdiff\_t difference\_type;

typedef std::bidirectional\_iterator\_tag iterator\_category;

mconst\_iterator(pointer ptr) : ptr\_(ptr) { };

mconst\_iterator() {}

self\_type operator=(const self\_type& other) { ptr\_ = other.ptr\_; return \*this; }

self\_type operator++() { self\_type i = \*this; ptr\_++; return i; }

self\_type operator--() { self\_type i = \*this; ptr\_--; return i; }

self\_type operator++(int junk) { ptr\_++; return \*this; }

self\_type operator--(int junk) { ptr\_--; return \*this; }

reference operator\*() { return \*ptr\_; }

pointer operator->() { return ptr\_; }

pointer operator&() { return ptr\_; }

bool operator==(const self\_type& rhs) { return ptr\_ == rhs.ptr\_; }

bool operator!=(const self\_type& rhs) { return ptr\_ != rhs.ptr\_; }

pointer ptr\_;

};

Обычный и константный bidirectional итераторы.

find, то есть поиск по ключу:

iterator find(const key\_type& Key)

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == Key)

{

return i;

}

} iterator res = end(); return res;}

Добавление элемента производится через аллокатор и прямое выделение памяти. Так же возможно добавление с помощью оператора [].

typedef assoc\_array<Key, T, Compare, Allocator> mymap;

mymap& insert(const value\_type& pair)

{

if (!is\_present(pair))

{

if (++size >= capacity)

{

reserve(capacity \* 2);

}

Allocator().construct(&data[size - 1], pair);

return \*this;

}

}

mapped\_type& operator[](const key\_type& \_Key)

{

if (has\_key(\_Key))

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == \_Key)

{

return i->second;

}

}

}

size\_type op = size;

insert(value\_type(\_Key, mapped\_type()));

return data[op].second;

}

Так же память выделяется при вызове конструкторов.

assoc\_array()

: size(0), capacity(20), data(Allocator().allocate(20))

{

}

assoc\_array(const mymap& \_Rhs)

: size(\_Rhs.size), capacity(\_Rhs.size + 20), data(Allocator().allocate(\_Rhs.size))

{

int count = 0;

for (iterator i = &\_Rhs.data[0]; i != &\_Rhs.data[\_Rhs.size]; ++i, ++count)

{

Allocator().construct(&data[count], \*i);

}

}

Удаление элемента по ключу:

void erace(const key\_type& Key)

{

int n = 0;

for (iterator i = begin(); i != end(); ++i, n++)

{

if (i->first == Key)

{

std::move(++i, end(), i);

--size;

data[size].value\_type::~value\_type();

return;

}

}

}

3. Вихідний текст програми розв’язку задачі

Ассоциативный массив:

#include <iostream>

#include <string>

template < typename Key, typename T, typename Compare = std::less<Key>, typename Allocator = std::allocator< std::pair<Key, T> > >

class assoc\_array

{

public:

class miterator

{

public:

typedef miterator self\_type;

typedef std::pair<Key, T> value\_type;

typedef std::pair<Key, T>& reference;

typedef std::pair<Key, T>\* pointer;

typedef std::bidirectional\_iterator\_tag iterator\_category;

typedef std::ptrdiff\_t difference\_type;

miterator(pointer ptr) : ptr\_(ptr) { };

miterator() {}

self\_type operator=(const self\_type& other) { ptr\_ = other.ptr\_; return \*this; }

self\_type operator++() { ptr\_++; return \*this; }

self\_type operator--() { ptr\_--; return \*this; }

self\_type operator++(int junk) { self\_type i = \*this; ptr\_++; return i; }

self\_type operator--(int junk) { self\_type i = \*this; ptr\_--; return i; }

reference operator\*() { return \*ptr\_; }

pointer operator->() { return ptr\_; }

pointer operator&() { return ptr\_; }

bool operator==(const self\_type& rhs) { return ptr\_ == rhs.ptr\_; }

bool operator!=(const self\_type& rhs) { return ptr\_ != rhs.ptr\_; }

private:

pointer ptr\_;

};

class mconst\_iterator

{

public:

typedef mconst\_iterator self\_type;

typedef std::pair<Key, T> value\_type;

typedef std::pair<Key, T>& reference;

typedef std::pair< Key, T>\* pointer;

typedef std::ptrdiff\_t difference\_type;

typedef std::bidirectional\_iterator\_tag iterator\_category;

mconst\_iterator(pointer ptr) : ptr\_(ptr) { };

mconst\_iterator() {}

self\_type operator=(const self\_type& other) { ptr\_ = other.ptr\_; return \*this; }

self\_type operator++() { self\_type i = \*this; ptr\_++; return i; }

self\_type operator--() { self\_type i = \*this; ptr\_--; return i; }

self\_type operator++(int junk) { ptr\_++; return \*this; }

self\_type operator--(int junk) { ptr\_--; return \*this; }

reference operator\*() { return \*ptr\_; }

pointer operator->() { return ptr\_; }

pointer operator&() { return ptr\_; }

bool operator==(const self\_type& rhs) { return ptr\_ == rhs.ptr\_; }

bool operator!=(const self\_type& rhs) { return ptr\_ != rhs.ptr\_; }

pointer ptr\_;

};

typedef assoc\_array<Key, T, Compare, Allocator> mymap;

typedef Key key\_type;

typedef T mapped\_type;

typedef std::pair<Key, T> value\_type;

typedef std::size\_t size\_type;

typedef std::ptrdiff\_t difference\_type;

typedef Compare key\_compare;

typedef Allocator allocator\_type;

typedef value\_type& reference;

typedef const value\_type& const\_reference;

typedef value\_type\* pointer;

typedef const value\_type\* const\_pointer;

typedef miterator iterator;

typedef mconst\_iterator const\_iterator;

typedef std::reverse\_iterator<miterator> reverse\_iterator;

typedef std::reverse\_iterator<mconst\_iterator> const\_reverse\_iterator;

assoc\_array()

: size(0), capacity(20), data(Allocator().allocate(20))

{

}

assoc\_array(const mymap& \_Rhs)

: size(\_Rhs.size), capacity(\_Rhs.size + 20), data(Allocator().allocate(\_Rhs.size))

{

int count = 0;

for (iterator i = &\_Rhs.data[0]; i != &\_Rhs.data[\_Rhs.size]; ++i, ++count)

{

Allocator().construct(&data[count], \*i);

}

}

~assoc\_array()

{

if (!empty())

{

Allocator().deallocate(data, capacity);

}

}

mymap& insert(const value\_type& pair)

{

if (!is\_present(pair))

{

if (++size >= capacity)

{

reserve(capacity \* 2);

}

Allocator().construct(&data[size - 1], pair);

return \*this;

}

}

bool is\_present(const value\_type& pair)

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == pair.first)

{

return true;

}

return false;

}

}

bool has\_key(const key\_type& \_Key)

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == \_Key)

{

return true;

}

}

return false;

}

mapped\_type& operator[](const key\_type& \_Key)

{

if (has\_key(\_Key))

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == \_Key)

{

return i->second;

}

}

}

size\_type op = size;

insert(value\_type(\_Key, mapped\_type()));

return data[op].second;

}

mymap& reserve(size\_type \_Capacity)

{

int count = 0;

if (\_Capacity < capacity)

{

return \*this;

}

pointer buf = Allocator().allocate(\_Capacity);

for (iterator i = begin(); i != end(); ++i, ++count)

{

Allocator().construct(&buf[count], \*i);

}

std::swap(data, buf);

Allocator().deallocate(buf, capacity);

capacity = \_Capacity;

return \*this;

}

void erace(const key\_type& Key)

{

int n = 0;

for (iterator i = begin(); i != end(); ++i, n++)

{

if (i->first == Key)

{

std::move(++i, end(), i);

--size;

data[size].value\_type::~value\_type();

return;

}

}

}

size\_type get\_size() const { return get\_size; }

bool empty() const

{

return size == 0;

}

iterator begin()

{

return &data[0];

}

iterator end()

{

return &data[size];

}

reverse\_iterator rbegin()

{

return &data[0];

}

reverse\_iterator rend()

{

return &data[size];

}

const\_iterator cbegin() const

{

return &data[0];

}

const\_iterator cend() const

{

return &data[size];

}

const\_reverse\_iterator rbegin() const

{

return &data[0];

}

const\_reverse\_iterator rend() const

{

return &data[size];

}

iterator find(const key\_type& Key)

{

for (iterator i = begin(); i != end(); ++i)

{

if (i->first == Key)

{

return i;

}

}

iterator res = end();

return res;

}

private:

pointer data;

size\_type size, capacity;

};

4. Опис іинтерфейсу (керівництво користувача)

Интерфейс как таковой отсутствует, так как создавался ассоциативный массив для работы с ним.

5. Опис тестових прикладів

Создадим массив и добавим в него несколько элементов. Сначала с помощью функции insert а затем с помощью оператора [].

assoc\_array<std::string, int> map1;

map1.insert({ "a", 30 });

map1["b"] = 5;

map1["c"] = 7;

map1["aa"] = 22;

map1["ba"] = 55;

map1["ca"] = 77;

Проверим еще одну функцию работы оператора [], попробуем достать элемент по ключу.

cout << "Element <b> = " << map[“b”] << endl;

Удалим элемент и посмотрим удалился ли он.

for (assoc\_array<std::string, int>::miterator i = map1.begin(); i != map1.end(); i++)

{

std::cout << "[" << i->first << "]" << "-->" << i->second << " ";

}

std::cout << std::endl;

map1.erace("b");

std::cout << "\nAfter arasing <b> pair\n" << std::endl;

for (assoc\_array<std::string, int>::miterator i = map1.begin(); i != map1.end(); i++)

{

std::cout << "[" << i->first << "]" << "-->" << i->second << " ";

}

std::cout << std::endl;

Так же проверим функцию find.

auto result = map1.find("aa");

if (result != map1.end())

{

std::cout<<"Found elem: " << "[" << result->first << "]" << "-->" << result->second << std::endl;

}

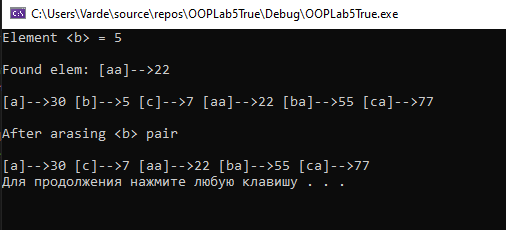
else

{

std::cout << "\nCannot find elem\n\n";

}

Вывод:



Как можно убедиться, все функции сработали как и планировалось.

6. Аналіз помилок (опис усунення зауважень)