ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. О.ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ ТА

МАТЕМАТИЧНОЇ КІБЕРНЕТИКИ

**Звіт з лабораторної роботи №5  
за дисципліною "Об'єктно-орієнтоване програмування"**

Виконала: Мовсісян Лаура

студентка групи ПА-20-1з

Дніпро, 2022

**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.

Варіант

1. (18.10) Напишіть зв’язувач binder3() [1, 18.4.4], який би зв’язував другий та третій аргументи трьохаргументної функції для отримання унарного предиката. Наведіть приклад, де функція binder3() є корисною функцією.

## Опис розв’язку

### Головна програма (main.cpp)

Головна програма має такі класи та структури:

1. struct ternary\_function - задає типи для тернарної функції.
2. struct in\_range - структура-функтор, яка перевіряє знаходиться число у числовому діапазоні.
3. struct linear\_function - структура-функтор, яка виводить значення функції a \* x + b, x - значення функції, а a та b - константи лінійної функції.
4. class binder3 - зв'язує другий та третій аргументи для утворення унарної функції.
5. struct filter\_compare - структура-функтор, яка перевіряє чи знаходяться два числа у заданому діапазоні і перше число менше другого.

Також головна програма має такі функції:

1. binder3<Operation> bind3 - виводить binder3
2. int main() - запускає головне меню.

Структура має такі внутрішні типи:

1. typedef Arg1 first\_argument\_type; - тип першого аргументу тернарної функції.
2. typedef Arg2 second\_argument\_type; - тип другого аргументу тернарної функції.
3. typedef Arg3 third\_argument\_type; - тип третього аргументу тернарної функції.
4. typedef Result result\_type; - тип результату тернарної функції.

#### Cтруктура in\_range

Структура має єдину функцію - оператор функтор:

bool operator() (const T& x, const T& y, const T& z) const - перевіряє чи знаходиться x у проміжку між y та z;

#### Cтруктура linear\_function

Структура має єдину функцію - оператор функтор:

T operator() (const T& x, const T& a, const T& b) const - виводить результат функції a \* x + b.

Клас binder3 має єдине поле op, яке повинно бути класом з тернарним оператором-функтором і задає два внутрішніх типи:

1. typename Operation::second\_argument\_type second; - тип першого аргументу.
2. typename Operation::third\_argument\_type third; - тип другого аргументу.

Також клас має дві функції:

1. binder3(const Operation& x, const second& y, const third& z); - конструктор з параметрами, параметр x - клас з тернарним оператором-функтором і прив'язані другий та третій аргументи y та z.
2. typename Operation::result\_type operator() (const typename Operation::first\_argument\_type& x) - оператор-функтор, який перетворює клас у унарну функцію.

#### Cтруктура filter\_compare

Структура має поле унарного функтора f, та функції конструктор та оператор-функтор:

bool operator() (const T& x, const T& y) const - перевіряє чи функтор f виводить true для x та y і перевіряє чи x менше за y.

## Опис іинтерфейсу

Проект має єдину програму з таким меню:

Choose option:

0. Exit.

1. Input array.

2. Output array.

3. Count elements that in value range.

4. Smart sort vector.

5. Transform by linear function.

Після того як програма вивела меню, вона циклічно запитує команди з цього меню.

Опис пунктів меню:

1. 0. Exit - Вихід із програми
2. 1. Input array. - ввести масив з клавіатури.
3. 2. Output array. - вивести масив на екран.
4. 3. Count elements that in value range. - запитує деякий діапазон чисел та рахує кількість елементів які задовольняють цьому діапазону.
5. 4. Smart sort vector. - запитує деякий діапазон чисел та сортує вибірково за заданим діапазоном.
6. 5. Transform by linear function. - запитує коефіцієнти a та b і перетворює кожний елемент за формулою a \* x + b, де x - значення елемента.

## Опис тестових прикладів

Choose option:

0. Exit.

1. Input array.

2. Output array.

3. Count elements that in value range.

4. Smart sort vector.

5. Transform by linear function.

>>> 1

Type new size of array:

>>> 9

Type elements:

9 8 7 6 5 4 3 2 1

>>> 2

Array has 9 elements:

9 8 7 6 5 4 3 2 1

>>> 3

Type value interval:

>>> 4 7

Array has 4 elements between this interval

>>> 4

Type value interval:

>>> 3 6

>>> 2

Array has 9 elements:

9 8 7 3 4 5 6 2 1

>>> 5

Type cofficients a and b for linear function a \* x + b:

>>> 2 3

>>> 2

Array has 9 elements:

21 19 17 9 11 13 15 7 5

>>> 0

## Вихідний текст програми розв’язку задачі

### Файл main cpp.

#include <cstdlib>

#ifndef NOSTL

#include <iostream>

#include <vector>

#include <algorithm>

#include <iomanip>

#include <functional>

using namespace std;

#else

#include "gaura.h"

using namespace gaura;

#endif

typedef unsigned int ui;

template <class Arg1, class Arg2, class Arg3, class Result>

struct ternary\_function {

typedef Arg1 first\_argument\_type;

typedef Arg2 second\_argument\_type;

typedef Arg3 third\_argument\_type;

typedef Result result\_type;

};

template <class T> struct in\_range : ternary\_function <T,T,T,bool> {

bool operator() (const T& x, const T& y, const T& z) const {

return (y <= z) ? y <= x && x <= z : z <= x && x <= y;

}

};

template <class T> struct linear\_function : ternary\_function <T,T,T,T> {

T operator() (const T& x, const T& a, const T& b) const {

return a \* x + b;

}

};

template <class Operation>

class binder3

: public unary\_function <typename Operation::first\_argument\_type,

typename Operation::result\_type> {

protected:

Operation op;

typename Operation::second\_argument\_type second;

typename Operation::third\_argument\_type third;

public:

binder3 (const Operation& x,

const typename Operation::second\_argument\_type& y,

const typename Operation::third\_argument\_type& z) :

op (x), second(y), third(z) {}

typename Operation::result\_type

operator() (const typename Operation::first\_argument\_type& x) const {

return op(x, second, third);

}

};

template <class Operation, class T>

binder3<Operation> bind3 (const Operation& op, const T& x, const T& y) {

typedef typename Operation::second\_argument\_type s;

typedef typename Operation::third\_argument\_type t;

return binder3<Operation>(op, s(x), t(y));

}

template <class F, class T>

struct filter\_compare : binary\_function <T,T,bool> {

filter\_compare(const F& filter) : f(filter) {}

bool operator() (const T& x, const T& y) const {

return f(x) and f(y) and x < y;

}

F f;

};

int main() {

vector<ui> a(0, 0);

int answer = -1;

streamsize mstrm = numeric\_limits<streamsize>::max();

cout << "Choose option:\n"

<< " 0. Exit.\n"

<< " 1. Input array.\n"

<< " 2. Output array.\n"

<< " 3. Count elements that in value range.\n"

<< " 4. Smart sort vector.\n"

<< " 5. Transform by linear function.\n";

while (answer != 0) {

cout << ">>> ";

cin >> answer;

cin.ignore(mstrm,'\n');

switch (answer) {

case 0:

return 0;

case 1: {

cout << "Type new size of array:\n>>> ";

ui size;

cin >> size;

a.resize(size, 0);

cout << "Type elements:\n";

vector<ui>::iterator i;

for (i = a.begin(); i != a.end(); i++) {

cin >> (\*i);

}

break;

}

case 2: {

cout << "Array has " << a.size() << " elements:\n";

vector<ui>::iterator i;

for (i = a.begin(); i != a.end(); i++) {

cout << \*i;

if (i + 1 != a.end()) {

cout << " ";

} else {

cout << "\n";

}

}

break;

}

case 3: {

cout << "Type value interval:\n>>> ";

ui b, e;

cin >> b >> e;

cin.ignore(mstrm,'\n');

cout << "Array has "

<< count\_if(a.begin(), a.end(), bind3(in\_range<ui>(), b, e))

<< " elements between this interval\n";

break;

}

case 4: {

cout << "Type value interval:\n>>> ";

ui b, e;

cin >> b >> e;

sort(a.begin(), a.end(), filter\_compare<binder3<in\_range<ui>>, ui>(bind3(in\_range<ui>(), b, e)));

continue;

}

case 5: {

cout << "Type cofficients a and b for linear function a \* x + b:\n>>> ";

ui ca, cb;

cin >> ca >> cb;

transform(a.begin(), a.end(), a.begin(), bind3(linear\_function<ui>(), ca, cb));

break;

}

default :

cout << "You typed wrong option, try again\n";

}

}

return 0;

}

### Файл gaura.h

#ifndef GAURA\_H

#define GAURA\_H

#include <stddef.h>

#include <stdio.h>

#include <limits.h>

namespace gaura {

typedef unsigned long long streamsize;

struct MyException {

MyException(const char \* string) : string\_(string) {}

const char \* what () const {

return string\_;

}

const char \* string\_;

};

class ostream {

public:

ostream(FILE\* file) : file\_(file) {}

ostream& operator<<(const char \* value) {

fprintf(file\_, "%s", value);

return \*this;

}

ostream& operator<<(unsigned int value) {

fprintf(file\_, "%u", value);

return \*this;

}

ostream& operator<<(unsigned long value) {

fprintf(file\_, "%lu", value);

return \*this;

}

private:

FILE\* file\_;

};

static ostream cout(stdout);

class istream {

public:

istream(FILE\* file) : file\_(file) {}

istream& operator>>(unsigned int& value) {

fscanf(file\_, "%u", &value);

return \*this;

}

istream& operator>>(int& value) {

fscanf(file\_, "%i", &value);

return \*this;

}

void ignore(streamsize n = 1, int delim = EOF) {

int c = 0;

for (streamsize i = 0; i < n && c != delim; i++) {

c = getc(file\_);

}

}

private:

FILE\* file\_;

};

static istream cin(stdin);

template<class T>

class vector;

typedef unsigned long size\_type;

template<class InputIterator, class OutputIterator>

OutputIterator copy (InputIterator first, InputIterator last,

OutputIterator result) {

while (first != last) {

\*result = \*first;

++result; ++first;

}

return result;

}

template <class T> const T& min (const T& a, const T& b) {

return !(b < a) ? a : b;

}

template<class T>

class VectorIterator {

public:

typedef vector<T> vector\_type;

typedef VectorIterator<T> iterator;

typedef size\_type difference\_type;

typedef T value\_type;

typedef value\_type& reference;

typedef value\_type\* pointer;

explicit VectorIterator(vector\_type\* vector = nullptr,

size\_type index = 0) :

vector\_(vector), index\_(index) {}

iterator& operator++() {

return \*this += 1;

}

iterator operator++(int) {

iterator out = \*this;

\*this += 1;

return out;

}

iterator& operator--() {

return \*this -= 1;

}

iterator operator--(int) {

iterator out = \*this;

\*this -= 1;

return out;

}

iterator& operator+=(size\_type index) {

return \*this = \*this + index;

}

friend iterator operator+(const iterator& lhs, size\_type rhs) {

return iterator(lhs.vector\_, lhs.index\_ + rhs);

}

friend iterator operator+(size\_type lhs, const iterator& rhs) {

return rhs + lhs;

}

iterator& operator-=(size\_type value) {

return \*this = \*this - value;

}

friend iterator operator-(const iterator& lhs, size\_type rhs) {

return iterator(lhs.vector\_, lhs.index\_ - rhs);

}

friend difference\_type operator-(const iterator& lhs,

const iterator& rhs) {

if (lhs.vector\_ != rhs.vector\_)

throw MyException("Iterators from different vectors");

return lhs.index\_ - rhs.index\_;

}

reference operator\*(){

return (\*vector\_)[index\_];

}

pointer operator->() const {

return &(\*vector\_)[index\_];

}

friend bool operator==(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ == rhs.index\_;

return s && i;

}

friend bool operator!=(const iterator& lhs, const iterator& rhs) {

return !(lhs == rhs);

}

friend bool operator<(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ < rhs.index\_;

return s && i;

}

friend bool operator>(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ > rhs.index\_;

return s && i;

}

friend bool operator<=(const iterator& lhs, const iterator& rhs) {

return lhs < rhs || lhs == rhs;

}

friend bool operator>=(const iterator& lhs, const iterator& rhs) {

return lhs > rhs || lhs == rhs;

}

private:

vector\_type\* vector\_;

size\_type index\_;

};

template<class T>

class ConstVectorIterator {

public:

typedef vector<T> vector\_type;

typedef ConstVectorIterator<T> iterator;

typedef size\_type difference\_type;

typedef T value\_type;

typedef value\_type& reference;

typedef value\_type\* pointer;

explicit ConstVectorIterator(const vector\_type\* vector = nullptr,

size\_type index = 0) :

vector\_(vector), index\_(index) {}

iterator& operator++() {

return \*this += 1;

}

iterator operator++(int) {

iterator out = \*this;

\*this += 1;

return out;

}

iterator& operator--() {

return \*this -= 1;

}

iterator operator--(int) {

iterator out = \*this;

\*this -= 1;

return out;

}

iterator& operator+=(size\_type index) {

return \*this = \*this + index;

}

friend iterator operator+(const iterator& lhs, size\_type rhs) {

return iterator(lhs.vector\_, lhs.index\_ + rhs);

}

friend iterator operator+(size\_type lhs, const iterator& rhs) {

return rhs + lhs;

}

iterator& operator-=(size\_type value) {

return \*this = \*this - value;

}

friend iterator operator-(const iterator& lhs, size\_type rhs) {

return iterator(lhs.vector\_, lhs.index\_ - rhs);

}

friend difference\_type operator-(const iterator& lhs,

const iterator& rhs) {

if (lhs.vector\_ != rhs.vector\_)

throw MyException("Iterators from different vectors");

return lhs.index\_ - rhs.index\_;

}

value\_type operator\*(){

return (\*vector\_)[index\_];

}

pointer operator->() const {

return &(\*vector\_)[index\_];

}

friend bool operator==(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ == rhs.index\_;

return s && i;

}

friend bool operator!=(const iterator& lhs, const iterator& rhs) {

return !(lhs == rhs);

}

friend bool operator<(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ < rhs.index\_;

return s && i;

}

friend bool operator>(const iterator& lhs, const iterator& rhs) {

bool s = lhs.vector\_ == rhs.vector\_;

bool i = lhs.index\_ > rhs.index\_;

return s && i;

}

friend bool operator<=(const iterator& lhs, const iterator& rhs) {

return lhs < rhs || lhs == rhs;

}

friend bool operator>=(const iterator& lhs, const iterator& rhs) {

return lhs > rhs || lhs == rhs;

}

private:

const vector\_type\* vector\_;

size\_type index\_;

};

template<class T>

class vector {

public:

typedef T value\_type;

typedef VectorIterator<T> iterator;

typedef ConstVectorIterator<T> const\_iterator;

explicit vector(size\_type size = 0, value\_type value = value\_type()) :

size\_(size), array\_(nullptr) {

array\_ = new value\_type[size];

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

\*i = value;

}

}

vector(const vector& vector\_) : size\_(vector\_.size\_), array\_(nullptr) {

array\_ = new value\_type[size\_];

copy(vector\_.begin(), vector\_.end(), begin());

}

vector& operator=(const vector& vector\_) {

delete [] array\_;

size\_ = vector\_.size\_;

array\_ = new value\_type[size\_];

copy(vector\_.begin(), vector\_.end(), begin());

return \*this;

}

~vector() {

delete [] array\_;

array\_ = nullptr;

}

void resize(size\_type size, value\_type value = value\_type()) {

vector vector\_(size, value);

copy(begin(), begin() + min(size, size\_), vector\_.begin());

\*this = vector\_;

}

size\_type size() const { return size\_; }

iterator begin() { return iterator(this, 0); }

const\_iterator begin() const { return const\_iterator(this, 0); }

iterator end() { return iterator(this, size\_); }

const\_iterator end() const { return const\_iterator(this, size\_); }

value\_type& operator[](size\_type index) {

return array\_[index];

}

value\_type operator[](size\_type index) const {

return array\_[index];

}

private:

size\_type size\_;

T\* array\_;

};

template <class Arg, class Result>

struct unary\_function {

typedef Arg argument\_type;

typedef Result result\_type;

};

template <class Arg1, class Arg2, class Result>

struct binary\_function {

typedef Arg1 first\_argument\_type;

typedef Arg2 second\_argument\_type;

typedef Result result\_type;

};

template <class T>

struct numeric\_limits;

template <>

struct numeric\_limits<unsigned long long> {

static unsigned long long max() { return ULLONG\_MAX; }

};

template <class Iterator> struct iterator\_traits {

typedef typename Iterator::difference\_type difference\_type;

typedef typename Iterator::value\_type value\_type;

typedef typename Iterator::pointer pointer;

typedef typename Iterator::reference reference;

};

template <class T> class iterator\_traits<T\*> {

typedef ptrdiff\_t difference\_type;

typedef T value\_type;

typedef value\_type\* pointer;

typedef value\_type& reference;

};

template <class T> class iterator\_traits<const T\*> {

typedef ptrdiff\_t difference\_type;

typedef T value\_type;

typedef const value\_type\* pointer;

typedef const value\_type& reference;

};

template <class InputIterator, class UnaryPredicate>

typename iterator\_traits<InputIterator>::difference\_type

count\_if (InputIterator first, InputIterator last, UnaryPredicate pred) {

typename iterator\_traits<InputIterator>::difference\_type ret = 0;

while (first!=last) {

if (pred(\*first)) ++ret;

++first;

}

return ret;

}

template <class RandomAccessIterator, class Compare>

void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp) {

for (RandomAccessIterator i = first; i != last; i++) {

for (RandomAccessIterator j = i; j != last; j++) {

if (comp(\*j, \*i)) {

typename iterator\_traits<RandomAccessIterator>::value\_type tmp = \*i;

\*i = \*j;

\*j = tmp;

}

}

}

}

template <class InputIterator, class OutputIterator, class UnaryOperator>

OutputIterator transform (InputIterator first1, InputIterator last1,

OutputIterator result, UnaryOperator op)

{

while (first1 != last1) {

\*result = op(\*first1);

++result; ++first1;

}

return result;

}

}

#endif