Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ № 8

Классы контейнеры-итераторы.

STL-контейнеры

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 А.П. Красько

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

# 1 ПОСТАНОВКА ЗАДАЧИ

Реализовать классы контейнер и итератор для работы с двунаправленным кольцом. Реализовать класс алгоритма, в котором определить методы поиска и сортировки двунаправленного кольца. Выполнить программу.

# 2 ДИАГРАММА КЛАССОВ

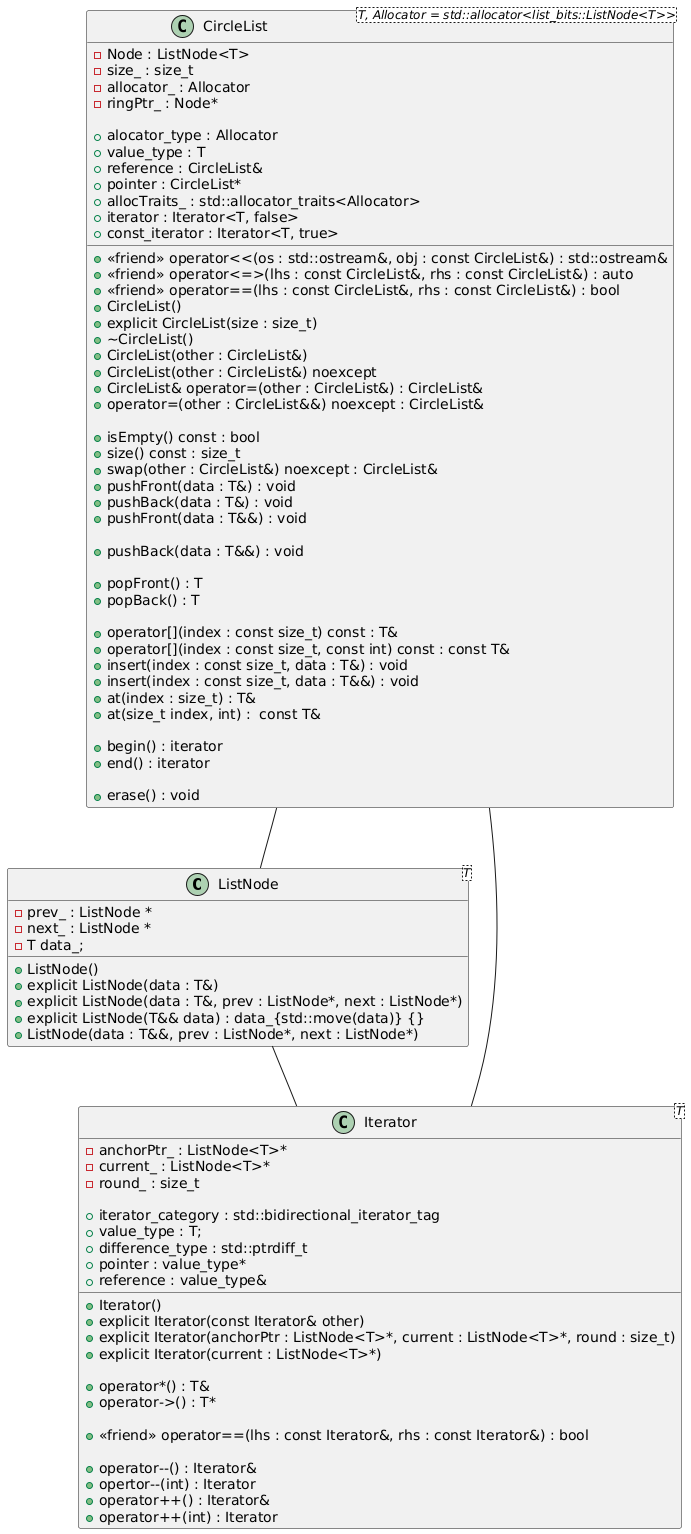


Рисунок 2.1 – Диаграмма классов

# 3 ЛИСТИНГ КОДА

Файл list\_algorithm.hh:

#pragma once  
#include <optional>  
  
#include "list.hh"  
namespace cList {  
template <typename U, std::forward\_iterator Iter>  
Iter find(Iter begin, Iter end, U& data) {  
 for (; begin != end; ++begin) {  
 if (\*begin == data) break;  
 }  
 return begin;  
}  
  
template <std::forward\_iterator ForwardIterator>  
void bubbleSort(ForwardIterator first, ForwardIterator last) {  
 if (first == last) {  
 return;  
 }  
 bool swapped;  
 do {  
 swapped = false;  
 ForwardIterator current = first;  
 ForwardIterator next\_it = std::next(current);  
  
 while (next\_it != last) {  
 if (\*next\_it < \*current) {  
 std::iter\_swap(current, next\_it);  
 swapped = true;  
 }  
 ++current;  
 ++next\_it;  
 }  
 } while (swapped);  
}  
} // namespace cList

Файл list\_bits.hh:

#include <iostream>  
#include <iterator>  
namespace list\_bits {  
template <typename T>  
struct ListNode {  
 public:  
 ListNode \*prev\_ = nullptr;  
 ListNode \*next\_ = nullptr;  
 T data\_;  
 ListNode() : data\_{T()} {}  
 explicit ListNode(T& data) : data\_{data} {}  
 explicit ListNode(T& data, ListNode\* prev, ListNode\* next) : prev\_{prev}, next\_{next}, data\_{data} {}  
 explicit ListNode(T&& data) : data\_{std::move(data)} {}  
 ListNode(T&& data, ListNode\* prev, ListNode\* next) : prev\_{prev}, next\_{next}, data\_{std::move(data)} {}  
};  
template <typename T>  
class Iterator {  
 private:  
 ListNode<T>\*anchorPtr\_;  
 ListNode<T> \*current\_;  
 size\_t round\_ = 0;  
  
 public:  
 using iterator\_category = std::bidirectional\_iterator\_tag;  
 using value\_type = T;  
 using difference\_type = std::ptrdiff\_t;  
 using pointer = value\_type\*;  
 using reference = value\_type&;  
  
 friend Iterator<const T>;  
 Iterator() = default;  
 explicit Iterator(const Iterator& other) = default;  
 explicit Iterator(ListNode<T>\* anchorPtr, ListNode<T>\* current, size\_t round)  
 : anchorPtr\_{anchorPtr}, current\_{current}, round\_{round} {}  
 explicit Iterator(ListNode<T>\* current) : anchorPtr\_{current}, current\_{current} {}  
  
 template <typename U> requires (!std::is\_const\_v<U>)  
 Iterator(const Iterator<U>& rhs) : anchorPtr\_{rhs.anchorPtr\_}, current\_{rhs.current\_} {}  
  
 reference operator\*() const { return this->current\_->data\_; }  
 pointer operator->() { return &(this->current\_->data\_); }  
  
 friend bool operator==(const Iterator& lhs, const Iterator& rhs) { return lhs.current\_ == rhs.current\_ && lhs.round\_ == rhs.round\_; }  
  
 Iterator& operator--() {  
 this->current\_ = this->current\_->prev\_;  
 if (this->current\_ == this->anchorPtr\_) {  
 this->round\_--;  
 }  
 return \*this;  
 }  
 Iterator operator--(int) {  
 Iterator cop{\*this};  
 --(\*this);  
 return cop;  
 }  
 Iterator& operator++() {  
 this->current\_ = this->current\_->next\_;  
 if (this->current\_ == this->anchorPtr\_) {  
 this->round\_++;  
 }  
 return \*this;  
 }  
 Iterator operator++(int) {  
 Iterator cop{\*this};  
 ++(\*this);  
 return cop;  
 }  
   
};  
static\_assert(std::bidirectional\_iterator<Iterator<int>>);  
} // namespace list\_bits

Файл screens.hh:

#pragma once  
#include <l8/include/list.hh>  
#include <memory>  
  
namespace screens {  
void printMainScreen();  
bool inputList(cList::CircleList<double> &list);  
bool addElement(cList::CircleList<double> &list);  
bool clearList(cList::CircleList<double> &list);  
bool sortList(cList::CircleList<double> &list);  
bool findElement(cList::CircleList<double> &list);  
bool printList(const cList::CircleList<double> &list);  
} // namespace screens

Файл list.hh:

#pragma once  
#include <execution>  
#include <iostream>  
#include <memory>  
  
#include "list\_bits.hh"  
namespace cList  
{  
   
template <typename T, typename Allocator = std::allocator<list\_bits::ListNode<T>>>  
class CircleList {  
 private:  
 using Node = list\_bits::ListNode<T>;  
 size\_t size\_;  
 [[no\_unique\_address]]  
 Allocator allocator\_{};  
 Node\* ringPtr\_;  
  
 public:  
 using alocator\_type = Allocator;  
 using value\_type = T;  
 using reference = CircleList&;  
 using pointer = CircleList\*;  
 using allocTraits\_ = std::allocator\_traits<Allocator>;  
 using iterator = list\_bits::Iterator<T>;  
 using const\_iterator = list\_bits::Iterator<const T>;  
  
  
  
 friend std::ostream& operator<<(std::ostream& os, const CircleList& obj) {  
 os << '[';  
 for (size\_t i = 0; i < obj.size\_; i++) {  
 os << obj[i] << " ";  
 }  
 os << ']';  
 return os;  
 }  
  
 friend auto operator<=>(const CircleList& lhs, const CircleList& rhs) { return lhs.size\_ <=> rhs.size\_; }  
 friend bool operator==(const CircleList& lhs, const CircleList& rhs) {  
 if (lhs.size\_ != rhs.size\_) return false;  
 for (size\_t i = 0; i < lhs.size\_; i++) {  
 if (lhs[i] != rhs[i]) return false;  
 }  
 return true;  
 }  
  
 public:  
 CircleList() : size\_{0}, ringPtr\_{nullptr} {}  
   
 explicit CircleList(size\_t size) : size\_{size}, ringPtr\_{nullptr} {  
 if (size\_ == 0) return;  
 ringPtr\_ = allocator\_.allocate(1);  
 allocTraits\_::construct(allocator\_, ringPtr\_);  
 Node\* current = ringPtr\_;  
 for (size\_t i = 1; i < size\_; ++i) {  
 Node\* newNode = allocator\_.allocate(1);  
 allocTraits\_::construct(allocator\_, newNode);  
 current->next\_ = newNode;  
 newNode->prev\_ = current;  
 current = newNode;  
 }  
 }  
 ~CircleList() { erase(); };  
 CircleList(CircleList& other) : size\_{other.size\_}, ringPtr\_{allocator\_.allocate(1)} {  
 allocTraits\_::construct(allocator\_, ringPtr\_, other[0]);  
 if (size\_ == 1) {  
 ringPtr\_->next\_ = ringPtr\_;  
 ringPtr\_->prev\_ = ringPtr\_;  
 }  
 Node\* current = ringPtr\_;  
 for (size\_t i = 1; i < size\_; ++i) {  
 Node\* newNode = allocator\_.allocate(1);  
 allocTraits\_::construct(allocator\_, newNode, other[i]);  
 current->next\_ = newNode;  
 newNode->prev\_ = current;  
 current = newNode;  
 }  
 ringPtr\_->prev\_ = current;  
 current->next\_ = ringPtr\_;  
 }  
 CircleList(CircleList&& other) noexcept : size\_{other.size\_}, ringPtr\_{other.ringPtr\_} { other.ringPtr\_ = nullptr; }  
  
 CircleList& operator=(CircleList& other) {  
 \*this = CircleList(other);  
 return \*this;  
 }  
 CircleList& operator=(CircleList&& other) noexcept {  
 size\_ = other.size\_;  
 allocator\_ = std::move(other.allocator\_);  
 ringPtr\_ = other.ringPtr\_;  
 other.ringPtr\_ = nullptr;  
 return \*this;  
 }  
  
 bool isEmpty() const { return size\_ == 0; }  
 size\_t size() const { return size\_; }  
 CircleList& swap(CircleList& other) noexcept {  
 CircleList tmp{std::move(other)};  
 other = std::move(\*this);  
 \*this = std::move(tmp);  
 return \*this;  
 }  
 void pushFront(T& data) {  
 pushBack(data);  
 if (size\_ != 1) {  
 ringPtr\_ = ringPtr\_->prev\_;  
 }  
 }  
 void pushBack(T& data) {  
 Node\* tmpPtr = allocator\_.allocate(1);  
 allocTraits\_::construct(allocator\_, tmpPtr, data);  
 if (size\_ != 0) {  
 tmpPtr->next\_ = ringPtr\_;  
 tmpPtr->prev\_ = ringPtr\_->prev\_;  
 ringPtr\_->prev\_->next\_ = tmpPtr;  
 ringPtr\_->prev\_ = tmpPtr;  
 } else {  
 ringPtr\_ = tmpPtr;  
 ringPtr\_->next\_ = ringPtr\_;  
 ringPtr\_->prev\_ = ringPtr\_;  
 }  
 size\_++;  
 }  
 void pushFront(T&& data) {  
 pushBack(std::move(data));  
 if (size\_ != 1) {  
 ringPtr\_ = ringPtr\_->prev\_;  
 }  
 }  
  
 void pushBack(T&& data) {  
 Node\* tmpPtr = allocator\_.allocate(1);  
 allocTraits\_::construct(allocator\_, tmpPtr, std::move(data));  
 if (size\_ != 0) {  
 tmpPtr->next\_ = ringPtr\_;  
 tmpPtr->prev\_ = ringPtr\_->prev\_;  
 ringPtr\_->prev\_->next\_ = tmpPtr;  
 ringPtr\_->prev\_ = tmpPtr;  
 } else {  
 ringPtr\_ = tmpPtr;  
 ringPtr\_->next\_ = ringPtr\_;  
 ringPtr\_->prev\_ = ringPtr\_;  
 }  
 size\_++;  
 }  
  
 T popFront() {  
 ringPtr\_ = ringPtr\_->next\_;  
 return popBack();  
 }  
 T popBack() {  
 if (size\_ == 0) throw std::length\_error("Circle list is empty");  
 Node\* tmpPtr = ringPtr\_->prev\_;  
  
 ringPtr\_->prev\_ = tmpPtr->prev\_;  
 tmpPtr->prev\_->next\_ = ringPtr\_;  
  
 T data = tmpPtr->data\_;  
 allocTraits\_::destroy(allocator\_, tmpPtr);  
 return data;  
 }  
  
 T& operator[](const size\_t index) const {  
 Node\* tmp = ringPtr\_;  
 for (size\_t i = 0, count = index % size\_; i < count; i++) {  
 tmp = tmp->next\_;  
 }  
 return tmp->data\_;  
 }  
 const T& operator[](const size\_t index, const int) const {  
 Node\* tmp = ringPtr\_;  
 for (size\_t i = 0, count = index % size\_; i < count; i++) {  
 tmp = tmp->next\_;  
 }  
 return tmp->data\_;  
 }  
 void insert(const size\_t index, T& data) {  
 if (index >= size\_) throw std::invalid\_argument("Index out of range");  
 this->operator[](index) = data;  
 }  
 void insert(const size\_t index, T&& data) {  
 this->operator[](index) = std::move(data);  
 }  
 T& at(size\_t index){  
 if (index >= size\_) throw std::invalid\_argument("Index out of range");  
 return this->operator[](index);  
 }  
 const T& at(size\_t index, int){  
 if (index >= size\_) throw std::invalid\_argument("Index out of range");  
 return this->operator[](index);  
 }  
  
 iterator begin() { return iterator(ringPtr\_); }  
 iterator end() { return iterator(ringPtr\_, ringPtr\_, 1); }  
  
 void erase() {  
 if (!ringPtr\_) return;  
 Node\* current = ringPtr\_;  
 for (size\_t i = 0; i < size\_; i++) {  
 Node\* tmp = current->next\_;  
 allocTraits\_::destroy(allocator\_, current);  
 allocator\_.deallocate(current, 1);  
 current = tmp;  
 }  
 size\_ = 0;  
 };  
};  
} // namespace cList

Файл screens.cc:

#include <consoleUtils.hh>  
#include <l8/include/list.hh>  
#include <l8/include/list\_algorithm.hh>  
#include <print>  
  
using namespace std;  
  
using namespace console\_utils;  
  
namespace screens {  
void printMainScreen() {  
 auto [cols, rows] = getConsoleDimensions();  
 println("{:^{}}", "\x{1B}[48;5;35mLab 8\x{1B}[0m", cols);  
 println("Please select action:\n");  
 println(" 1.Input List");  
 println(" 2.Add element");  
 println(" 3.Clear List");  
 println(" 4.Sort List");  
 println(" 5.Find element");  
 println(" 6.Print list");  
 println(" 7.Exit");  
}  
bool inputList(cList::CircleList<double> &list) {  
 list.erase();  
 size\_t num;  
 readT(num, "Plese enter list length:", [](size\_t number) { return number > 0; }, "Number should be > 0");  
 for (size\_t i = 0; i < num; i++) {  
 double data;  
 readT(data, "Plese enter list element:");  
 list.pushBack(data);  
 }  
 return true;  
}  
bool addElement(cList::CircleList<double> &list) {  
 double data;  
 readT(data, "Plese enter list element:");  
 list.pushBack(data);  
 return true;  
}  
bool clearList(cList::CircleList<double> &list) {  
 list.erase();  
 return true;  
}  
bool sortList(cList::CircleList<double> &list) {  
 cList::bubbleSort(list.begin(), list.end());  
 return true;  
}  
bool findElement(cList::CircleList<double> &list) {  
 double data;  
 readT(data, "Plese enter list element:");  
   
 if (auto res = cList::find(list.begin(), list.end(), data); res == list.end()) {  
 std::cout << "No such element\n";  
 } else {  
 std::cout << "List contains element\n";  
 }  
 return true;  
}  
bool printList(const cList::CircleList<double> &list) {  
 std::cout << list << '\n';  
  
 return true;  
}  
} // namespace screens

Файл main.cc:

#include <consoleUtils.hh>  
#include <iostream>  
#include <l8/include/list.hh>  
#include <l8/include/list\_algorithm.hh>  
#include <l8/include/screens.hh>  
#include <functional>  
  
using namespace std;  
using namespace screens;  
using namespace console\_utils;  
int main(void) {  
  
 cList::CircleList<double> list;  
  
 static array<function<bool()>, 7> actions = {  
 [&list]() { return inputList(list); },  
 [&list]() { return addElement(list); },  
 [&list]() { return clearList(list); },  
 [&list]() { return sortList(list); },  
 [&list]() { return findElement(list); },  
 [&list]() { return printList(list); },  
  
 []() { return false; },  
  
 };  
 unsigned int response;  
 do {  
 printMainScreen();  
 readT(response, ">", [](unsigned int num) { return num > 0 && num <= 7; });  
 cout << "\x{1B}[2J\x{1B}[H\n";  
 } while (actions[response - 1]());  
  
  
 return 0;  
}

Файл consoleUtils.hh:

#pragma once  
#include <functional>  
#include <iostream>  
#include <limits>  
#include <iomanip>  
#include <chrono>  
namespace console\_utils {  
std::pair<int, int> getConsoleDimensions();  
  
  
template <typename T, typename CT>  
void readT(T& data, const std::string& massage, CT validator) {  
 std::cout << massage;  
 while (((std::cin >> data).fail()) || !validator(data)) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T>  
void readT(T& data, const std::string& massage) {  
 std::cout << massage;  
 while ((std::cin >> data).fail()) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T, typename CT>  
void readT(T& data, const std::string& message, CT validator, const std::string& errmess) {  
 std::cout << message;  
 while (((std::cin >> data).fail()) || !validator(data)) {  
 std::cout << errmess;  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << message;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T = std::chrono::sys\_seconds, typename CT = const char \* >  
void readT(std::chrono::sys\_seconds &data, const char \* massage, const char \* format) {  
 std::cout << massage;  
 while ((std::cin >> std::chrono::parse(format, data)).fail()) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
} // namespace console\_utils

Файл consoleUtils.cc:

#include <iostream>  
  
#ifdef \_\_linux\_\_  
#include <sys/ioctl.h>  
#include <unistd.h>  
#endif  
  
#ifdef \_WIN32  
#include <Windows.h>  
#endif  
  
namespace console\_utils {  
std::pair<int, int> getConsoleDimensions() {  
#ifdef \_WIN32  
 CONSOLE\_SCREEN\_BUFFER\_INFO csbi;  
 GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);  
 return std::make\_pair(csbi.srWindow.Right - csbi.srWindow.Left + 1, csbi.srWindow.Bottom - csbi.srWindow.Top + 1);  
#endif  
#ifdef \_\_linux\_\_  
  
 struct winsize w;  
 ioctl(STDOUT\_FILENO, TIOCGWINSZ, &w);  
 return std::make\_pair(w.ws\_col, w.ws\_row);  
  
#endif  
}  
} // namespace console\_utils

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**



Рисунок 4.1 – Главное меню

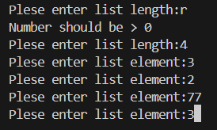


Рисунок 4.2 – Ввод кольца

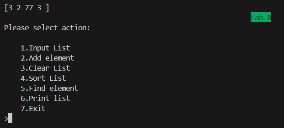


Рисунок 4.3 – Вывод кольца

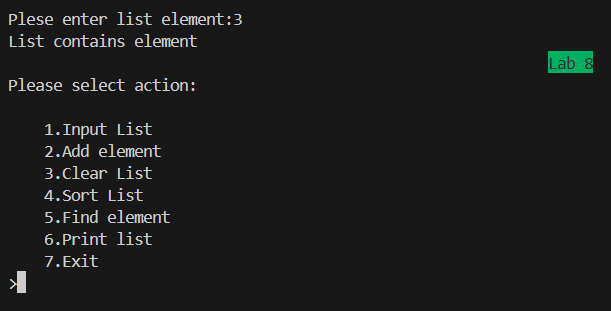


Рисунок 4.4 – Поиск элемента

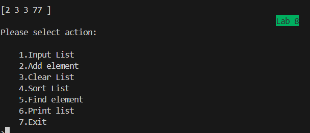


Рисунок 4.4 – Сортировка

**5 ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы был успешно реализован класс контейнера «двунаправленное кольцо» с поддержкой итераторов, а также алгоритмы поиска и сортировки. Реализация соответствует принципам STL: обеспечена корректная работа с итераторами, поддержка семантики перемещения, использование аллокаторов и совместимость с стандартными алгоритмами.