Отчет по лабораторной работе № 26 по курсу

«Практикум программирования»

Студент группы М8О-109Б-22 Концебалов Олег Сергеевич

Контакты: telegram @baronpipistron

Работа выполнена: 3.06.2023

Преподаватель: каф.806 Сысоев Максим Алексеевич

Отчет сдан «25» июня 2023 г., итоговая оценка ____

Подпись преподавателя _____

1. Тема: Абстрактные типы данных. Рекурсия. Модульное программирование на C++

2. Цель работы: Реализовать свою структуру данных и написать сортировку для нее

3. Задание (вариант № 3, 1): Структура данных Дек. Сортировка линейным выбором. Процедура поиск и удаление максимального элемента

4. Оборудование (студента):

Процессор AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz, ОП 16,0 Гб, SSD 512 Гб. Монитор 1920x1080 144 Hz

5. Программное обеспечение (студента):

Операционная система семейства Linux, наименование Ubuntu, версия 18.10

Интерпретатор команд: bash, версия 4.4.19

Система программирования – версия --, редактор текстов Emacs, версия 25.2.2

Утилиты операционной системы –

Прикладные системы и программы –

Местонахождение и имена файлов программ и данных на домашнем компьютере –

6. Идея, метод, алгоритм решения задачи *(в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)*

Все просто. Пишу дек и сортировку для него

7. Сценарий выполнения работы *(план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты, либо соображения по тестированию)*

1. Пишу свой дек
2. Пишу сортировку
3. Тестирую

8. Распечатка протокола *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)*

Node.hpp

```
#ifndef NODE_HPP
#define NODE_HPP

#include <iostream>

template <typename T>
class Iterator;

template <typename T>
class Deque;

template <typename T>
class Node{
    friend class Iterator<T>;
    friend class Deque<T>;

private:
    T data;
    Node<T>* next;
    Node<T>* previous;

public:
    Node();
    Node(const T& value);
    Node(const T& value, Node<T>* previous, Node<T>* current);
};

#include "Node.cpp"

#endif
```

Node.cpp

```
#include "Node.hpp"

template <typename T>
Node<T>::Node(): data(0){}
```

```cpp
template <typename T>
Node<T>::Node(const T& value): data(value){}

template <typename T>
Node<T>::Node(const T& value, Node<T>* previous, Node<T>*
current): data(value), next(current), previous(previous){
    if (previous != nullptr){
        previous->next = this;
    }

    if (current != nullptr){
        current->previous = this;
    }
}
```

Iterator.hpp

```cpp
#ifndef ITERATOR_HPP
#define ITERATOR_HPP

#include "Node.hpp"
#include <iostream>

template <typename T>
class Deque;

template <typename T>
class Iterator{
    friend class Deque<T>;

private:
    Node<T>* node;

public:
    Iterator() = default;
    Iterator(Node<T>* node);
    ~Iterator() = default;

    Node<T>* get_node();
```

```
    T& operator*();
    const T& operator*() const;

    Iterator<T> operator++();
    Iterator<T> operator--();
    Iterator<T> operator+(int64_t value);
    Iterator<T> operator-(int64_t value);

    bool operator==(const Iterator<T>& other);
    bool operator!=(const Iterator<T>& other);
};

#include "Iterator.cpp"

#endif
```

Iterator.cpp

```
#include "Iterator.hpp"

template <typename T>
Iterator<T>::Iterator(Node<T>* node): node(node){}

template <typename T>
Node<T>* Iterator<T>::get_node(){
    return node;
}

template <typename T>
T& Iterator<T>::operator*(){
    return node->data;
}

template <typename T>
const T& Iterator<T>::operator*() const{
    return node->data;
}

template <typename T>
```

```cpp
Iterator<T> Iterator<T>::operator++(){
    if (node != nullptr) node = node->next;

    return *this;
}

template <typename T>
Iterator<T> Iterator<T>::operator--(){
    if (node != nullptr) node = node->previous;

    return *this;
}

template <typename T>
Iterator<T> Iterator<T>::operator+(int64_t value){
    if (node == nullptr) return *this;

    if (value >= 0){
        while (node != nullptr && value != 0){
            --value;
            node = node->next;
        }
    } else{
        while (node != nullptr && value != 0){
            ++value;
            node = node->previous;
        }
    }

    return *this;
}

template <typename T>
Iterator<T> Iterator<T>::operator-(int64_t value){
    return *this + (-value);
}

template <typename T>
bool Iterator<T>::operator==(const Iterator<T>& other){
    return this->node == other.node;
```

```
}

template <typename T>
bool Iterator<T>::operator!=(const Iterator<T>& other){
    return !(this->node == other.node);
}
```

Deque.hpp

```
#ifndef DEQUE_HPP
#define DEQUE_HPP

#include "Node.hpp"
#include "Iterator.hpp"
#include <iostream>

template <typename t>
class Iterator;

template <typename T>
class Deque{
    friend class Iterator<T>;

private:
    Node<T>* head;
    Node<T>* tail;
    size_t sz;

public:
    Deque();
    ~Deque();

    Iterator<T> begin() const;
    Iterator<T> end() const;

    void push_front(const T& value);
    void push_back(const T& value);

    void pop_front();
    void pop_back();
```

```
    size_t size() const;
    bool empty() const;
    void clear();

    void sort();

    void print() const;
};

#include "Deque.cpp"

#endif
```

Deque.cpp

```
#include "Deque.hpp"
#include "Iterator.hpp"
#include "Node.hpp"
#include <iostream>

template <typename T>
Deque<T>::Deque(): head(nullptr), tail(nullptr), sz(0){}

template <typename T>
Deque<T>::~Deque(){
    if (begin().node == nullptr) return;

    Iterator<T> cur = begin();
    Iterator<T> to_delet = cur;

    while (cur.node != nullptr){
        ++cur;
        delete to_delet.node;
        to_delet = cur;
    }

    delete cur.node;
}
```

```cpp
template <typename T>
Iterator<T> Deque<T>::begin() const{
    Iterator<T> tmp(head);
    return tmp;
}

template <typename T>
Iterator<T> Deque<T>::end() const{
    Iterator<T> tmp(tail);
    return tmp;
}

template <typename T>
void Deque<T>::push_front(const T& value){
    if (begin().node == nullptr){
        head = new Node<T>(value);
        tail = head;
        ++sz;
        return;
    }

    head = new Node<T>(value, nullptr, head);
    ++sz;

    return;
}

template <typename T>
void Deque<T>::push_back(const T& value){
    if (begin().node == nullptr){
        head = new Node<T>(value);
        tail = head;
        ++sz;
        return;
    }

    tail = new Node<T>(value, tail, nullptr);
    ++sz;
    return;
}
```

```cpp
template <typename T>
void Deque<T>::pop_front(){
    if (sz == 0) return;

    Node<T>* tmp = head;
    head = head->next;
    delete tmp;
    --sz;
}

template <typename T>
void Deque<T>::pop_back(){
    if (sz == 0) return;

    Node<T>* tmp = tail;
    tail = tail->previous;
    delete tmp;
    --sz;
}

template <typename T>
size_t Deque<T>::size() const{
    return sz;
}

template <typename T>
bool Deque<T>::empty() const{
    return sz == 0;
}

template <typename T>
void Deque<T>::clear(){
    Iterator<T> cur = begin();
    Iterator<T> to_delet = cur;

    while (cur.node != nullptr){
        ++cur;
        delete to_delet.node;
        to_delet = cur;
```

```
    }

    delete cur.node;
}

template <typename T>
void Deque<T>::sort(){
    if (sz == 0) return;

    Node<T>* max;
    for (Node<T>* i = begin().node; i != end().node->next; i =
i->next){
        max = i;
        for (Node<T>* j = i->next; j != end().node->next; j =
j->next){
            if (max->data < j->data){
                max = j;
            }
        }

        head = new Node<T>(max->data, nullptr, head);

        if (max == tail){
            tail = tail->previous;
            tail->next = nullptr;
        } else{
            max->previous->next = max->next;
            max->next->previous = max->previous;
        }

        if (i != max) i = i->previous;
    }

    delete max;
}

template <typename T>
void Deque<T>::print() const{
    if (sz == 0){
        std::cout << "Deque is empty\n";
```

```
        return;
    }

    std::cout << "Deque = { ";

    for (Iterator<T> i = begin(); i != end() + 1; ++i){
        std::cout << i.node->data << " ";
    }

    std::cout << "}\n";
}
```

Benchmark.cpp

```
#include "Deque.hpp"
#include <chrono>
#include <deque>
#include <iostream>

void bencmark(){
    std::cout << "Comparing my \"Deque\" and \"deque\" from
STL\n\n";

    Deque<int> my_deque;
    std::deque<int> original_deque;

    std::cout << "10^4" << "\n\n";

    std::cout << "push_front\n";
    std::chrono::steady_clock::time_point start_time =
std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        my_deque.push_front(i);
    }
    std::chrono::steady_clock::time_point end_time =
std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";
```

```
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        original_deque.push_front(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_front\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        my_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        original_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npush_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        my_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
```

```
        original_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        my_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 10000; ++i){
        original_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "10^5" << "\n\n";

    std::cout << "push_front\n";
    std::chrono::steady_clock::time_point start_time =
std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        my_deque.push_front(i);
    }
    std::chrono::steady_clock::time_point end_time =
std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";
```

```
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        original_deque.push_front(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_front\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        my_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        original_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npush_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        my_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
```

```
        original_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        my_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 100000; ++i){
        original_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "10^6" << "\n\n";

    std::cout << "push_front\n";
    std::chrono::steady_clock::time_point start_time =
std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        my_deque.push_front(i);
    }
    std::chrono::steady_clock::time_point end_time =
std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";
```

```
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        original_deque.push_front(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_front\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        my_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        original_deque.pop_front();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npush_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        my_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
```

```
        original_deque.push_back(i);
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    std::cout << "\npop_back\n";
    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        my_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work my Deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    for (int i = 0; i != 1000000; ++i){
        original_deque.pop_back();
    }
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original deque: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    return;
}
```

Run.cpp

```
#include "Deque.hpp"
#include "benchmark.cpp"
#include <iostream>

int main(){
    Deque<int> deq;

    deq.push_back(21);
    deq.push_back(1);
```

```
    deq.push_back(10);
    deq.push_back(3);
    deq.push_back(9);
    deq.push_back(7);
    deq.push_back(2);
    deq.push_back(15);
    deq.push_back(19);
    deq.push_back(17);
    deq.push_back(16);
    deq.push_back(20);

    std::cout << "Before sorting: ";
    deq.print();
    deq.sort();

    std::cout << "\nAfter sorting: ";
    deq.print();

    bencmark();
}
```

9. Дневник отладки *(дата и время сеансов отладки и основные события [ошибки в сценарии и программе, нестандартные ситуации] и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы)*

№	*Лаб. или дом*	*Дата*	*Время*	*Событие*	*Действие по исправлению*	*Примечания*

Особых проблем при выполнении лабы не возникло

10. Замечания автора *(по существу работы)*

Замечания отсутствуют

Тесты производительности

Функция push_back

	10^4	10^5	10^6
myDeque	0[ms]	5[ms]	48[ms]
Std::deque	1[ms]	1[ms]	11[ms]

Функция pop_back

	10^4	10^5	10^6
myDeque	0[ms]	2[ms]	23[ms]
Std::deque	0[ms]	0[ms]	6[ms]

Функция push_front

	10^4	10^5	10^6
myDeque	1[ms]	5[ms]	49[ms]
Std::deque	0[ms]	0[ms]	11[ms]

Функция pop_front

	10^4	10^5	10^6
myDeque	0[ms]	2[ms]	23[ms]
Std::deque	0[ms]	0[ms]	5[ms]

Как видно из результатов тестирования, скорость работы стандартного дека из STL превосходит мой кастомный примерно в 2,5-3,5 раза — варьируется от объема данных. Но стоить отметить, что на малом количестве данных мой дек ничем не уступает стандартному, а в push_back даже обогнал его (мб погрешность). Я думаю, что такие различия в скорости связаны с тем, что дек из STL реализован с помощью чанков на векторах/массивах, а мой реализован на двусвязном списке + под капотом стандартного дека могут быть различные оптимизации, которые так же ускоряют его работу

11. Вывод

Выполняя данную лабу, познакомился со структурой данных дек и написал свою сортировку линейным выбором. Правда не совсем понятна роль процедуры в этой затее, так как она, на мой взгляд, только замедляет работу программы и отсортировать дек можно более эффективным способом. В целом получился прикольный опыт, написал свои итераторы для дека, что понадобится в будущем и поможет другим людям проще ориентироваться в моем коде.

Подпись студента _____