Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

СТЕКИ И ОЧЕРЕДИ

Отчет по лабораторной работе №2

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Выполнил: студент гр. 439-3

_____ Мазовец А. С.

«___»_____ 2020 г.

Проверил: ассистент каф. АСУ

_____ Яблонский Я. В.

«___»_____ 2020 г.

Томск 2020

1 Задание на лабораторную работу

Вариант 3.

Пусть даны две очереди X и Y, содержащие вещественные числа. Из каждой очереди одновременно извлекается по одному числу x и y, соответственно.

Если x < y, то число (x + y) помещается в конец очереди X, иначе число (x – y) помещается в конец очереди Y.

Вычисления заканчиваются, когда одна из очередей становится пустой.

Подсчитайте число шагов, через которое одна из очередей станет пустой.

Для реализации АТД Очередь использовать динамическое распределение памяти. Начальное заполнение очередей X и Y считываются из файла.

2 Алгоритм решения задачи

С помощью метода АТД очередь top, который возвращает значение первого элемента в очереди с начала, сравниваю значения для двух очередей. Далее, в соответствии с заданием, с помощью методов АТД очереди deQueue и enQueue вынимаю значения из очередей и помещаю в конец требуемой очереди.

Для решения задачи выбрал реализацию очереди на динамическом односвязном линейном списке с фиктивным корневым элементом. Следовательно, для выполнения задачи необходимо написать АТД список и АДТ очередь.

3 Листинг программы

Реализация класса АТД линейный список. Вспомогательный класс node.

asmazovec@case insert ./sem3.СиАОДвЭВМ.lab2 > cat list.hpp

```
#ifndef list_HPP_
#define list_HPP_

#include <iostream>
#include <stdexcept>

namespace ads {
    /* class node */
    template <typename T>
    class _node;

    template <typename T>
    std::ostream& operator<< (std::ostream& o, const _node<T> &n);

    template <typename T>
    class _node {
        public:
            T field;
            _node *next;

            _node (T data = 0);
            _node (const _node &) = delete;
            _node (_node &&n);
            _node &operator= (const _node &) = delete;
            _node &operator= (_node &&n);
            _node &operator= (T data);

            friend std::ostream& operator<< <> (std::ostream& o,
                                                const _node<T> &n);
    };

    template <typename T>
    _node<T>::_node (T data): field (data), next(nullptr) {}

    template <typename T>
    _node<T>::_node (_node &&n)
        :field (n.field), next (n.next) {
        std::cout << "move clone node " << std::endl;
        n.next = nullptr;
    }

    template <typename T>
    _node<T> &_node<T>::operator= (_node<T> &&n) {
        std::cout << "move operator= node " << std::endl;
        if (&n == this)
            return *this;
```

```
        field = n.field;
        next = n.next;
        n.next = nullptr;
        return *this;
    }

    template <typename T>
    _node<T> &_node<T>::operator= (T data) {
        field = data;
        return *this;
    }

    template <typename T>
    std::ostream& operator<< (std::ostream& o, const _node<T> &n) {
        o << n.field;
        return o;
    }
}
```

Реализация класса АТД линейный список. Реализация класса list.

asmazovec@case insert ./sem3.СиАОДвЭВМ.lab2 > cat list.hpp

```
namespace ads {
    /* class list */
    template <typename T>
    class list;

    template <typename T>
    std::ostream& operator<< (std::ostream& o, const list<T> &l);

    template <typename T>
    class list {
        private:
            mutable _node<T> node;
            _node<T> &getNode (int index) const;

        public:
            list();
            ~list();
            T &operator[] (int index);
            T operator[] (int index) const;

            list &push (T data, int index);
            list &pushStart (T data);
            list &pushBack (T data);
            T pop (int index);
            int len() const;
            bool isEmpty() const;

            friend std::ostream& operator<< <> (std::ostream& o,
                                                const list<T> &l);
    };

    template <typename T>
    list<T>::list(): node (_node<T>()) {}

    template <typename T>
    list<T>::~list() {
        _node<T> *cur = node.next;
        _node<T> *next;
        while (cur) {
            next = cur->next;
            delete cur;
            cur = next;
        }
    }

    template <typename T>
    _node<T> &list<T>::getNode (int index) const {
        if (++index < 0)
            throw std::invalid_argument ("Invalid list index.");
```

```
    _node<T> *cur = &node;
    for (int i = 0; i < index; i++)
        if (cur->next)
            cur = cur->next;
        else
            throw std::out_of_range ("Index is out of range.");
    return *cur;
}

template <typename T>
T &list<T>::operator[] (int index) {
    if (index < 0)
        throw std::invalid_argument ("Invalid list index.");
    if (isEmpty())
        throw std::out_of_range ("List is empty.");
    _node<T> *n = &getNode (index);
    return n->field;
}

template <typename T>
T list<T>::operator[] (int index) const {
    if (index < 0)
        throw std::invalid_argument ("Invalid list index.");
    if (isEmpty())
        throw std::out_of_range ("List is empty.");
    _node<T> *n = &getNode (index);
    return n->field;
}

template <typename T>
list<T> &list<T>::push (T data, int index) {
    if (index < 0)
        throw std::invalid_argument ("Invalid list index.");
    _node<T> *n = &getNode (index - 1);
    _node<T> *cur = new _node<T> (data);
    cur->next = n->next;
    n->next = cur;
    return *this;
}

template <typename T>
list<T> &list<T>::pushStart (T data) {
    push (data, 0);
    return *this;
}

template <typename T>
list<T> &list<T>::pushBack (T data) {
    push (data, len());
    return *this;
}
```

```
    template <typename T>
    T list<T>::pop (int index) {
        if (index < 0)
            throw std::invalid_argument ("Invalid list index.");
        if (isEmpty())
            throw std::out_of_range ("List is empty.");
        _node<T> *n = &getNode (index - 1);
        int data = n->next->field;
        _node<T> *next = n->next->next;
        n->next->next = nullptr;
        delete n->next;
        n->next = next;
        return data;
    }

    template <typename T>
    int list<T>::len() const {
        _node<T> *cur = &node;
        int len = 0;
        while (cur->next) {
            cur = cur->next;
            len++;
        }
        return len;
    }

    template <typename T>
    bool list<T>::isEmpty() const {
        return !node.next;
    }

    template <typename T>
    std::ostream& operator<< (std::ostream& o, const list<T> &l) {
        _node<T> *cur = &l.node;
        while (cur->next) {
            cur = cur->next;
            o << *cur << " ";
        }
        return o;
    }
}

#endif /* list_HPP_ */
```

asmazovec@case insert ./sem3.СиАОДвЭВМ.lab2 > cat queue.hpp

```
#ifndef QUEUE_HPP_
#define QUEUE_HPP_

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

namespace ads {
    template <typename T>
    class queue {
        private:
            list<T> qlist;

        public:
            queue &enQueue (T data);
            T deQueue();
            T top() const;
            T end() const;
            bool isEmpty() const;
    };

    template <typename T>
    queue<T> &queue<T>::enQueue (T data) {
        qlist.pushBack(data);
        return *this;
    }

    template <typename T>
    T queue<T>::deQueue () {
        return qlist.pop(0);
    }

    template <typename T>
    T queue<T>::top() const {
        return qlist[0];
    }

    template <typename T>
    T queue<T>::end() const {
        return qlist[qlist.len() - 1];
    }

    template <typename T>
    bool queue<T>::isEmpty() const {
        return qlist.isEmpty();
    }
}

#endif /* QUEUE_HPP_ */
```

asmazovec@case insert ./sem3.СиАОДвЭВМ.lab2 > cat main.cpp

```
#include "queue.hpp"
#include <iostream>
#include <fstream>
#include <stdexcept>

using namespace ads;

int task() {
    queue<float> X;
    queue<float> Y;

    /* чтение очередей из файлов */
    float x, y;
    std::fstream file;

    file.open("X");
    while (file >> x)
        X.enQueue (x);
    file.close();

    file.open("Y");
    while (file >> y)
        Y.enQueue (y);
    file.close();

    int iter = 0;
    /* вычисление */
    while (!X.isEmpty() && !Y.isEmpty()) {
        if (X.top() < Y.top())
            X.enQueue (X.deQueue() + Y.deQueue());
        else
            Y.enQueue (X.deQueue() - Y.deQueue());
        iter++;
    }
    return iter;
}

int main() {
    try {
        std::cout << task() << std::endl;
    } catch (std::exception &exception) {
        std::cerr << "Standard exception: " << exception.what()
                                            << std::endl;
    } catch (...) {
        std::cerr << "Undetermined exception" << std::endl;
    }
    return 0;
}
```

4 Пример решения

Тест 1. Входные данные:

X = {0, 1, 2, 3},

Y = {0, 1, 2, 3}

Ожидаемый результат работы программы — 4:

(0) X = {0, 1, 2, 3}, Y = {0, 1, 2, 3} => Y.enQueue (0 - 0),

(1) X = {1, 2, 3}, Y = {1, 2, 3, 0} => Y.enQueue (1 - 1),

(2) X = {2, 3}, Y = {2, 3, 0, 0} => Y.enQueue (2 - 2),

(3) X = {3}, Y = {3, 0, 0, 0} => Y.enQueue (3 - 3),

(4) X = {}, Y = {0, 0, 0, 0}

Пример работы программы (рисунок 4.1 — Тест 1):

```
asmazovec@case insert ~/dev/sem3.СиАОДвЭВМ.lab2 > ./a.out
4
```

Рисунок 4.1 — Тест 1.

Тест 2. Входные данные:

X = {0, 1, 2, 3},

Y = {3, 2, 1, 0}

Ожидаемый результат работы программы — 6:

(0) X = {0, 1, 2, 3}, Y = {3, 2, 1, 0} => X.enQueue (3 + 0),

(1) X = {1, 2, 3, 3}, Y = {2, 1, 0} => X.enQueue (1 + 2),

(2) X = {2, 3, 3, 3}, Y = {1, 0} => Y.enQueue (1 - 0),

(3) X = {3, 3, 3}, Y = {0, 1} => Y.enQueue (3 - 0),

(4) X = {3, 3}, Y = {1, 3} => Y.enQueue (3 - 1),

(5) X = {3}, Y = {3, 2} => Y.enQueue (3 - 3),

(6) X = {}, Y = {2, 0}

Пример работы программы (рисунок 4.2 — Тест 2):

```
asmazovec@case insert ~/dev/sem3.СиАОДвЭВМ.lab2 > ./a.out
6
```

5 Вывод

Были изучены АТД линейный список и АТД очередь с динамически выделяемой памятью. Были реализованы классы list и queue, реализующие функционал и структуры данных изученных АТД. Решена поставленная задача.