Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

**ОТЧЁТ**

Тема: «Шаблоны классов»

Выполнил

Студент РИС-22-2б

Зубов Р.А.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

**Постановка задачи**

1. Создать пользовательский класс с минимальной функциональностью.

2. Написать функцию для создания объектов пользовательского класса (ввод исходной информации с клавиатуры) и сохранения их в потоке (файле).

3. Написать функцию для чтения и просмотра объектов из потока.

4. Написать функцию для удаления объектов из потока в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

5. Написать функцию для добавления объектов в поток в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

6. Написать функцию для изменения объектов в потоке в соответствии с заданием варианта. Для выполнения задания выполнить перегрузку необходимых операций.

7. Для вызова функций в основной программе предусмотреть меню.

Удалить все записи меньшие заданного значения.

Увеличить все записи с заданным значением на число L.

Добавить K записей после элемента с заданным номером.

Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

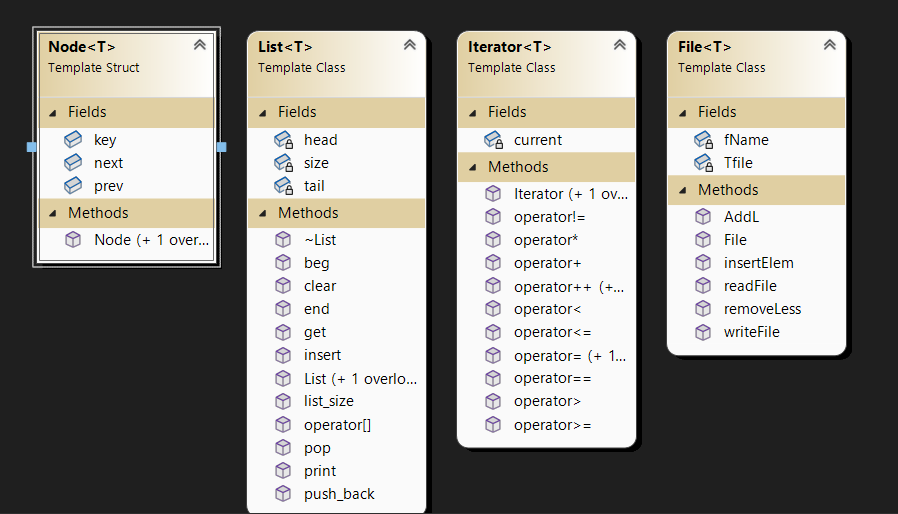
[] – доступа по индексу;

int() – определение размера списка;

\* вектор – умножение элементов списков a[i]\*b[i];

Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

**UML**

****

**Код программы**

**Pair.h:** #pragma once

#include <iostream>

class Pair

{

public:

int fNum = 0;

double sNum = 0;

Pair() {}

Pair(int first, double second) {

fNum = first;

sNum = second;

}

Pair& operator=(const Pair& t);

Pair& operator=(const int& t);

Pair operator\*(Pair& t);

friend std::ostream& operator<<(std::ostream& out, const Pair& t);

friend std::istream& operator>>(std::istream& in, const Pair& t);

};

**Pair.cpp:**

#include "Pair.h"

Pair& Pair::operator=(const Pair& pair) {

this->fNum = pair.fNum;

this->sNum = pair.sNum;

return \*this;

}

Pair& Pair::operator=(const int& t) {

this->fNum = t;

this->sNum = t;

return \*this;

}

Pair Pair::operator\*(Pair& pair) {

Pair new\_pair;

new\_pair.fNum = this->fNum \* pair.fNum;

new\_pair.sNum = this->sNum \* pair.sNum;

return new\_pair;

}

std::ostream& operator<<(std::ostream& out, const Pair& pair) {

out << pair.fNum << " : " << pair.sNum;

return out;

}

std::istream& operator>>(std::istream& in, Pair& pair) {

in >> pair.fNum >> pair.sNum;

return in;

}

**List.h:** #pragma once

#include <cstddef>

#include <iostream>

template <typename T>

struct Node {

T key;

Node<T>\* next = NULL;

Node<T>\* prev = NULL;

Node();

Node(T k, Node<T>\* n, Node<T>\* p);

};

template<typename T>

class Iterator {

template<typename T>

friend class List;

private:

Node<T>\* current = NULL;

public:

Iterator() { current = NULL; }

Iterator(Node<T>\* node) : current(node) {}

bool operator==(const Iterator<T>& other) const { //ñðàâíèâàþò òåêóùèé óçåë

return current == other.current; // è óçåë äðóãîãî èòåðàòîðà

} //

bool operator!=(const Iterator<T>& other) const { // íà ðàâåíñòâî è íåðàâåíñòâî.

return current != other.current;

}

T& operator\*() const { // îïåðàöèÿ ðàçûìåíîâàíèÿ èòåðàòîðà

return current->key;

}

Iterator& operator++() { //++i

current = current->next;

return \*this;

}

Iterator operator++(int) { // i++

Iterator<T> old = \*this;

current = current->next;

return old;

}

Iterator operator+(int n) const { // ïåðåõîä âïðàâî ê n ýëåìåíòó

Iterator<T> it(\*this);

while (n > 0 && it.current != NULL) {

it.current = it.current->next;

n--;

}

return it;

}

};

template<typename T>

class List {

Node<T>\* head;

Node<T>\* tail;

int size;

public:

List() {

head = NULL;

tail = NULL;

size = 0;

}

~List() {

clear();

}

T& get(int index);

T& operator[](int index) {

return get(index); // âîçâðàùàåì ññûëêó íà ýëåìåíò ïî èíäåêñó

}

// Ìåòîä äîáàâëåíèÿ ýëåìåíòà â êîíåö ñïèñêà

void push\_back(T k);

void pop(T key);

int list\_size() {

return size;

}

void clear();

void print() const;

friend std::ostream& operator<<(std::ostream& out, const List<T>& list)

{

list.print();

return out;

}

friend std::istream& operator>>(std::istream& in, List<T>& list)

{

int k;

in >> k;

list.push\_back(k);

return in;

}

Iterator<T> beg() {

return Iterator<T>(head); // âîçâðàùàåì èòåðàòîð íà ïåðâûé ýëåìåíò

}

Iterator<T> end() {

return Iterator<T>(tail->next); // âîçâðàùàåì èòåðàòîð íà ïîñëåäíèé ýëåìåíò

}

friend List<T> operator\*(List<T>& left, List<T>& right) {

List<T> result; // ñîçäàåì íîâûé ñïèñîê

Iterator<T> it1 = left.beg(); // èòåðàòîð ïî ïåðâîìó ñïèñêó

Iterator<T> it2 = right.beg(); // èòåðàòîð ïî âòîðîìó ñïèñêó

while (it1 != left.end() && it2 != right.end()) {

T res = (\*it1) \* (\*it2);

result.push\_back(res); // äîáàâëÿåì ïðîèçâåäåíèå ýëåìåíòîâ â íîâûé ñïèñîê

++it1; // ïåðåõîäèì ê ñëåäóþùåìó ýëåìåíòó ïåðâîãî ñïèñêà

++it2; // ïåðåõîäèì ê ñëåäóþùåìó ýëåìåíòó âòîðîãî ñïèñêà

}

return result; // âîçâðàùàåì íîâûé ñïèñîê

}

};

template<typename T>

Node<T>::Node(T k, Node<T>\* n, Node<T>\* p)

{

key = k;

next = n;

prev = p;

}

template<typename T>

T& List<T>::get(int index)

{

Node<T>\* current = head; // íà÷èíàåì ñ ãîëîâû ñïèñêà

int i = 0;

while (i < index && current->next != NULL) {

current = current->next; // ïåðåõîäèì ê ñëåäóþùåìó óçëó

}

return current->key; // âîçâðàùàåì ññûëêó íà äàííûå óçëà

}

template<typename T>

void List<T>::push\_back(T k)

{

Node<T>\* newNode = new Node<T>(k, NULL, tail); // ñîçäàåì íîâûé óçåë ñ êëþ÷îì key, óêàçàòåëåì íà ñëåäóþùèé óçåë ðàâíûì NULL è óêàçàòåëåì íà ïðåäûäóùèé óçåë ðàâíûì tail

if (tail != NULL) {

tail->next = newNode; // åñëè tail íå ðàâåí NULL, òî óñòàíàâëèâàåì óêàçàòåëü íà ñëåäóþùèé óçåë ó tail íà íîâûé óçåë

}

tail = newNode; // óñòàíàâëèâàåì tail íà íîâûé óçåë

if (head == NULL) {

head = newNode; // åñëè ñïèñîê áûë ïóñòîé, òî óñòàíàâëèâàåì head íà íîâûé óçåë

}

size++;

}

template<typename T>

void List<T>::pop(T key) {

Node<T>\* p = head;

while (p != NULL && p->key != key) { // Ïðîõîäèì ïî ñïèñêó äî íóæíîãî êëþ÷à

p = p->next;

}

if (p != NULL) { // Åñëè íàøëè óçåë ïî ïîçèöèè

if (p->prev != NULL) { // Åñëè óçåë íå ÿâëÿåòñÿ ãîëîâíûì

p->prev->next = p->next;

}

else { // Åñëè óçåë ÿâëÿåòñÿ ãîëîâíûì

head = p->next;

}

if (p->next != NULL) { // Åñëè óçåë íå ÿâëÿåòñÿ õâîñòîâûì

p->next->prev = p->prev;

}

else { // Åñëè óçåë ÿâëÿåòñÿ õâîñòîâûì

tail = p->prev;

}

delete p; // Óäàëÿåì òåêóùèé óçåë èç ïàìÿòè

size--;

}

}

template<typename T>

void List<T>::clear()

{

Node<T>\* p = head;

while (p != NULL) {

Node<T>\* q = p;

p = p->next;

delete q; // Óäàëÿåì òåêóùèé óçåë èç ïàìÿòè

}

head = NULL;

tail = NULL;

size = 0;

}

template<typename T>

void List<T>::print() const

{

Node<T>\* node = head;

while (node != NULL) {

std::cout << node->key << "->";

node = node->next;

}

std::cout << std::endl;

}

**File.h:** #pragma once

#include "List.h"

#include "Pair.h"

#include <iostream>

#include <string>

#include <fstream>

template <typename T>

class File

{

public:

File(std::string fName) : fName(fName) {}

int writeFile(List<T>& L);

int readFile(List<T>& L);

void insertElem(List<T>& L, int pos, int K);

void removeLess(const T& elem);

void AddL(T L);

private:

std::string fName; // main file

std::string Tfile = "temp.txt"; // temp file

};

template<typename T>

int File<T>::writeFile(List<T>& L)

{

std::fstream stream(fName, std::ios::out | std::ios::trunc /\*| std::ios::app\*/);

if (!stream) return -1;

int n = 0;

for (auto it = L.beg(); it != L.end(); it++)

{

stream << \*it << "\n";

n++;

}

stream.close();

return n;

}

template<typename T>

int File<T>::readFile(List<T>& L)

{

std::fstream stream(fName, std::ios::in);

if (!stream) return -1;

int n = 0;

L.clear();

T value;

while (stream >> value)

{

L.push\_back(value);

n++;

}

stream.close();

return n;

}

template<typename T>

void File<T>::insertElem(List<T>& L,int pos, int K)

{

std::ifstream file(fName);

T value;

for (int i = 0; i < K; i++) {

std::cin >> value;

L.insert(pos, value);

}

writeFile(L);

}

template<typename T>

void File<T>::removeLess(const T& elem)

{

std::ifstream inFile(fName);

std::ofstream outFile(Tfile);

T value;

while (inFile >> value)

{

if (value >= elem)

outFile << value << "\n";

}

inFile.close();

outFile.close();

remove(fName.c\_str());

rename(Tfile.c\_str(), fName.c\_str());

}

template<typename T>

void File<T>::AddL(T L)

{

std::ifstream inFile(fName);

std::ofstream outFile(Tfile);

T k;

while (inFile >> k)

{

outFile << k + L << "\n";

}

inFile.close();

outFile.close();

remove(fName.c\_str());

rename(Tfile.c\_str(), fName.c\_str());

}

**Source.cpp:**

#include "List.h"

#include "File.h"

#include "Pair.h"

#include <iostream>

#include <string>

using namespace std;

string FILE\_NAME = "List.txt";

int main()

{

List<Pair> list(3, Pair(0, 0));

File<Pair> file(FILE\_NAME);

list.push\_back(Pair(1, 1.1));

list.push\_back(Pair(2, 2.2));

list.push\_back(Pair(3, 3.3));

cout << "\n" << FILE\_NAME << " : " << endl;

cout << list << endl;

file.writeFile(list);

Pair K(2, 0.1);

cout << "\n" << FILE\_NAME << " Pair(2,0.1) " << endl;

file.AddL(K);

cout << list << endl;

cout << "\n" << FILE\_NAME << " " << endl;

file.removeLess(Pair(0, 1.1));

cout << list << endl;

int n, k;

cout << "\n" << FILE\_NAME << " " << endl;

cin >> n >> k;

cout << list << endl;

return 0;

}

**Контрольные вопросы**

1. Поток - это последовательный поток данных, который передает данные между программой и устройством ввода-вывода, например, между программой и консолью или файлом.
2. Существуют три типа потоков: ввода, вывода и ввода-вывода.
3. Для использования стандартных потоков не требуется подключать какие-либо дополнительные библиотеки, так как они уже встроены в стандартную библиотеку языка C++.
4. Для использования файловых потоков необходимо подключить библиотеку <fstream>.
5. Для использования строковых потоков необходимо подключить библиотеку <sstream>.
6. Для вывода в форматированный поток используется оператор вставки <<.
7. Для ввода из форматированных потоков используется оператор извлечения >>.
8. При выводе в форматированный поток используются методы, такие как width(), precision(), и fill().
9. При вводе из форматированного потока используются методы, такие как getline(), ignore() и peek().
10. Существуют различные режимы открытия файловых потоков, такие как ios::in, ios::out, ios::app, ios::trunc и ios::binary.
11. Режим ios::app используется для добавления записей в конец файла без удаления предыдущего содержимого.
12. Режим ios::in используется для чтения данных из файла, по умолчанию открытие файла происходит в режиме ios::in.
13. Режим ios::in | ios::out используется для одновременного чтения и записи в файл.
14. Режим ios::out используется для записи данных в файл, при этом предыдущее содержимое файла удаляется.
15. Для открытия потока в режиме ios::out | ios::app используется следующая команда:

std::fstream stream(fName, std::ios::out /\*| std::ios::app\*/);

1. Для открытия потока в режиме ios::out | ios::trunc используется следующая команда:

std::fstream stream(fName, std::ios::out | std::ios::trunc /\*| std::ios::app\*/);

1. Для открытия потока в режиме ios::out | ios::in | ios::trunc используется следующая команда:

std::fstream stream(fName, std::ios::out | std::ios::in | std::ios::trunc /\*| std::ios::app\*/);

1. Для открытия файла для чтения используется команда:

std::ifstream inFile(fName);

1. Для открытия файла для записи используется команда:

std::ofstream outFile(Tfile);

1. Примеры открытия файловых потоков в различных режимах:

#include <fstream>

std::ifstream input\_file("example.txt"); // открытие файла для чтения

if (input\_file.is\_open()) { // проверка, открыт ли файл

// ... чтение файла

input\_file.close(); // закрытие файла

}

std::ofstream output\_file("example.txt"); // открытие файла для записи

if (output\_file.is\_open()) { // проверка, открыт ли файл

// ... запись в файл

output\_file.close(); // закрытие файла

}

std::fstream inout\_file("example.txt", std::ios::in | std::ios::out); // открытие файла для чтения и записи

if (inout\_file.is\_open()) { // проверка, открыт ли файл

// ... чтение и запись в файл

inout\_file.close(); // закрытие файла

}

std::fstream binary\_file("example.bin", std::ios::binary | std::ios::in | std::ios::out); // открытие файла в двоичном режиме

if (binary\_file.is\_open()) { // проверка, открыт ли файл

// ... чтение и запись в файл в двоичном режиме

binary\_file.close(); // закрытие файла

}

1. Примеры чтения объектов из потока:

#include <iostream>

#include <fstream>

class Person {

public:

std::string name;

int age;

};

int main() {

std::ifstream input\_file("example.bin", std::ios::binary); // открытие файла в двоичном режиме для чтения\n if (input\_file.is\_open()) { // проверка, открыт ли файл

Person person;

input\_file.read((char\*)&person, sizeof(Person)); // чтение объекта из потока

std::cout << "Name: " << person.name << \", Age: " << person.age << std::endl;

input\_file.close(); // закрытие файла

}

return 0;

}

#include <iostream>

#include <fstream>\n\nclass Person {

public:

std::string name;

int age;

};

int main() {

std::ifstream input\_file("example.bin", std::ios::binary); // открытие файла в двоичном режиме для чтения

if (input\_file.is\_open()) { // проверка, открыт ли файл

Person person;

while (input\_file.read((char\*)&person, sizeof(Person))) { // чтение объектов из потока в цикле

std::cout << \"Name: " << person.name << \", Age: " << person.age << std::endl;\n }

input\_file.close(); // закрытие файла

}

return 0;

}

1. Примеры записи объектов в поток:

#include <fstream>

class Person {

public:

std::string name;

int age;

};

int main() {

std::ofstream output\_file("example.bin", std::ios::binary); // открытие файла в двоичном режиме для записи

if (output\_file.is\_open()) { // проверка, открыт ли файл

Person person = {"John", 30};

output\_file.write((char\*)&person, sizeof(Person)); // запись объекта в поток

output\_file.close(); // закрытие файла

}

return 0;

}

#include <iostream>\n#include <fstream>

class Person {

public:

std::string name;

int age;

};

int main() {

std::ofstream output\_file("example.bin", std::ios::binary); // открытие файла в двоичном режиме для записи

if (output\_file.is\_open()) { // проверка, открыт ли файл

Person people[] = {{"John", 30}, {"Jane", 25}, {"Bob", 40}};

for (const auto& person : people) { // запись объектов в поток в цикле

output\_file.write((char\*)&person, sizeof(Person));

}

output\_file.close(); // закрытие файла

}

return 0;

}

1. Алгоритм удаления записей из файла:

Открыть файл в режиме чтения и записи.

Создать временный файл.

Считать записи из исходного файла и записать их во временный файл, пропуская удаляемые записи.

Закрыть исходный файл и временный файл.

Удалить исходный файл.

Переименовать временный файл в имя исходного файла.

1. Алгоритм добавления записей в файл:

Открыть файл в режиме добавления.

Записать новые записи в конец файла.

Закрыть файл.

1. Алгоритм изменения записей в файле:

Открыть файл в режиме чтения и записи.

Создать временный файл.

Считать записи из исходного файла, изменять необходимые записи и записывать их во временный файл.

Закрыть исходный файл и временный файл.

Удалить исходный файл.

Переименовать временный файл в имя исходного файла.