Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

**Отчет по программированию**

Отчет по программированию выполнил:

студент 1 курса 141 группы

по направлению «Математическое обеспечение и администрирование информационных систем»

факультета «Компьютерных наук и информационных технологий»

Черногоров Владислав Максимович

Проверил(а): Казачкова А. А.

**СОДЕРЖАНИЕ**

1. Рекурсивные функции, перегрузки, шаблонные функции 3
2. Сортировки 12
3. Классы и списки 19
4. Наследование 38
5. Вектора 45

**Рекурсивные функции**

**Упражнение I, задание 9**

Разработать функцию, которая для заданного натурального числа N возвращает сумму его делителей. С помощью данной функции: для каждого целого числа на отрезке [а, b] вывести на экран сумму его делителей**.**

**Код программы**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <fstream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

#define ini(type, n) \

type n; \

cin >> n;

// Функция возвращает сумму делителей числа.

long long devSum (int c) {

long long devCount;

if (c == 0)

return 0;

else

devCount = !!(abs(c)-1)\*(c/abs(c))+c;

for (int i = 2; i <= c/2; i++) {

if (c%i == 0) devCount+=i\*(c/abs(c));

}

return devCount;

}

int main()

{

ini(int, a);

ini(int, b);

for (int i = min(a, b); i <= max(a ,b); i++) {

long long c = devSum(i);

cout << i << ": ";

if (c == 0) cout << "INF";

else cout << c;

cout << endl;

}

}

**Рекурсивные функции**

**Пример(ы)**

|  |  |
| --- | --- |
| Ввод | Вывод |
| 0 10 | 0: INF  1: 1  2: 3  3: 4  4: 7  5: 6  6: 12  7: 8  8: 15  9: 13  10: 18 |
| 30 20 | 20: 42  21: 32  22: 36  23: 24  24: 60  25: 31  26: 42  27: 40  28: 56  29: 30  30: 72 |

**Рекурсивные функции**

**Упражнение I, задача 16**

Разработать функцию, которая для заданного натурального числа N возвращает сумму его цифр. С помощью данной функции для заданного числа А вывести на экран предшествующее по отношению к нему число, сумма цифр которого равна сумме цифр числа А.

**Код программы**

#include <iostream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

// Функция возвращает сумму цифр числа

int numSum (int c) {

int sumCount = 0;

while (c != 0) {

sumCount += abs(c%10);

c /= 10;

}

return sumCount;

}

int main() {

int a;

cin >> a;

if (a == 0) {

cout << "ERROR\n";

return 0;

}

// Сумма цифр числа А

int aSum = numSum(a);

a--;

// Находим число, предшествующее числу aSum,

// сумма цифр которого равна сумме цифр А

while (aSum != numSum(a)) a--;

cout << a << endl;

}

**Пример(ы)**

|  |  |
| --- | --- |
| Ввод | Вывод |
| 1253 | 1244 |
| -27285 | -27294 |

**Рекурсивные функции**

**Упражнение II, задача 11**

Для вычисления цепной дроби: x/(1+x/(2+x/(3+...+x/(n+x)))). Найти значение данной дроби при заданном натуральном n.

**Код программы**

#include <iostream>

#include <iomanip>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

#define ini(type, n) \

type n; \

cin >> n;

int N;

// Функция заходит в рекурсию с неизмененным значением х N раз.

// Затем, выходя из рекурсии, возвращает вычесления выражения

// и подставляет под х, и так до тех пор, пока функция полностью

// не выйдет из рекурсии.

double calc (int n, double x) {

if (n != N) {

return x/(calc(n+1, x) + n);

}

else return x/(x + n);

}

int main()

{

cin >> N;

ini(double, b);

cout << fixed << setprecision(10) << b/calc(0, b) << endl;

}

**Пример(ы)**

|  |  |
| --- | --- |
| Ввод | Вывод |
| 2 1 | 0.7500000000 |
| 42 12.2324 | 3.2368381558 |

**Рекурсивные функции**

**Упражнение III, задача 12**

Разработать рекурсивную функцию для вывода на экран следующей картинки:

\* \* (n пробелов между звездочками)

\*\* \*\* (n-2 пробела)

\*\*\* \*\*\* (n-4 пробела)

...

\*\*\*\* \*\*\*\* (2 пробела)

\*\*\*\*\*\*\*\*\*\* (0 пробелов)

\*\*\*\* \*\*\*\* (2 пробела)

...

\*\*\* \*\*\* (n-4 пробела)

\*\* \*\* (n-2 пробела)

\* \* (n пробелов между звездочками)

**Код программы**

#include <iostream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

#define ini(type, n) \

type n; \

cin >> n;

void stars(int n, int temp) {

// Выполняем эти действия перед заходом в рекурсию

if (n != 0) {

forn(i, temp)

cout << '\*';

forn(i, n)

cout << ' ';

forn(i, temp)

cout << '\*';

cout << endl;

stars(n - 2, temp + 1);

}

// Эти действия выполняются перед выходом из рекурсии

forn(i, temp)

cout << '\*';

forn(i, n)

cout << ' ';

forn(i, temp)

cout << '\*';

cout << endl;

}

int main()

{

ini(int, n);

stars(n, 1);

}

**Рекурсивные функции**

cout << '\*';

forn(i, n)

cout << ' ';

forn(i, temp)

cout << '\*';

cout << endl;

}

int main()

{

ini(int, n);

stars(n, 1);

}

**Пример(ы)**

|  |  |
| --- | --- |
| Ввод | Вывод |
| 10 | \* \*  \*\* \*\*  \*\*\* \*\*\*  \*\*\*\* \*\*\*\*  \*\*\*\*\* \*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\* \*\*\*\*\*  \*\*\*\* \*\*\*\*  \*\*\* \*\*\*  \*\* \*\*  \* \* |

**Перегрузка функции**

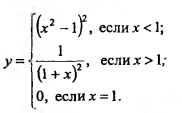
**Упражнение IV, задание 9**

Используя механизм перегрузки функций, разработайте две версии функции F, заголовки которых выглядят следующим образом:

- float F(float х);

- void F (float x, float &y);

Продемонстрируйте работу данных функций на примерах.



**Код программы**

#include <iostream>

using namespace std;

float F(float x) {

return 1/((1+x)\*(1+x));

}

void F(float x, float &y) {

y = (x\*x-1)\*(x\*x-1);

}

int main()

{

float x, y;

cin >> x;

if (x > 1) y=F(x);

else if (x < 1) F(x, y);

else y = 0;

printf("%.5f\n", y);

}

**Пример(ы)**

|  |  |
| --- | --- |
| Ввод | Вывод |
| 4 | 0.04000 |
| 0.2423 | 0.88603 |

**Шаблонные функции**

**Упражнение V, задание 9**

Использование функций-шаблонов: для работы с двумерными массивами арифметических типов данных разработать шаблоны ввода и вывода массива, а также шаблон для решения основной задачи: Подсчитать среднее арифметическое элементов, расположенных под побочной диагональю.

**Код программы**

#include <iostream>

#include <vector>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

// Функция выводит матрицу в стандартный поток

template <typename x>

void printMatrix(vector<vector<x>> vec) {

forn(i, vec.size()) {

cout << endl;

forn(j, vec[i].size())

cout << vec[i][j] << ' ';

}

}

// Главная шаблонная функция запрашивает пользователя

// ввести матрицу после чего выполняет над ней требуемые в задаче

// действия и выводит результат в стандартный поток

template <typename x>

void mainFunction(vector<vector<x>> vec) {

cout << "Enter the size: ";

ini(int, n);

ini(int, m);

vec.resize(n);

cout << "Enter the matrix: " << endl;

forn(i, n)

vec[i].resize(m);

forn(i, n)

forn(j, m)

cin >> vec[i][j];

problemSolution(vec, n, m);

printMatrix(vec);

}

// Функция считает среднее арифметическое элементов, расположенных

// под побочной диагональю матрицы (все элементы под побочной

// диагональю и на ней заменяются нулями)

template <typename x>

void problemSolution(vector<vector<x>> &vec, int n, int m) {

double middleSum = 0;

int cnt = 0;

int nm = min(n, m);

for (int i = (n - nm) / 2; i < n; i++) {

for (int j = max((m - nm) / 2 + nm - (i - ((n - nm) / 2)) - 1, 0); j < m; j++) {

middleSum += vec[i][j];

vec[i][j] = 0;

cnt++;

}

}

cout << "\nSolution: " << ((n\*m==max(n,m)) ? 0 : middleSum/cnt) << "\n";

}

// Функция запрашивает пользователя ввести тип значения,

// который он будет использовать, а затем запускает шаблонную функцию

bool chooseType() {

vector<vector<ll>> vec1;

vector<vector<double>> vec2;

cout << "Type of matrix:\n[1] long long\n[2] double\n";

ini(char, answer);

switch (answer) {

case '1': mainFunction(vec1); return true;

case '2': mainFunction(vec2); return true;

}

return false;

}

int main() {

while (!chooseType());

return 0;

}

**Шаблонные функции**

**Пример(ы)**

template <typename x>

void problemSolution(vector<vector<x>> &vec, int n, int m) {

double middleSum = 0;

int cnt = 0, nm = min(n, m);

for (int i = (n - nm) / 2; i < n; i++) {

for (int j = max((m - nm) / 2 + nm - (i - ((n - nm) / 2)) - 1, 0); j < m; j++) {

middleSum += vec[i][j];

vec[i][j] = 0;

cnt++;

}

}

cout << "\nSolution: " << ((n\*m==max(n,m)) ? 0 : middleSum/cnt) << "\n";

}

// Функция запрашивает пользователя ввести тип значения,

// который он будет использовать, а затем запускает шаблонную функцию

bool chooseType() {

vector<vector<ll>> vec1;

vector<vector<double>> vec2;

cout << "Type of matrix:\n[1] long long\n[2] double\n";

char answer;

cin >> answer;

switch (answer) {

case '1': mainFunction(vec1); return true;

case '2': mainFunction(vec2); return true;

}

return false;

}

int main() {

while (!chooseType());

return 0;

}

|  |  |
| --- | --- |
| Ввод | Вывод |
| Type of matrix:  [1] long long  [2] double  1  Enter the size: 3 6  Enter the matrix:  3 6 7 0 2 1  4 2 1 1 1 3  9 9 3 7 6 2 | Solution: 3  3 6 7 0 0 0  4 2 0 0 0 0  9 0 0 0 0 0 |

**Сортировки**

**Упражнение I, задание 18**

В файле input.txt содержатся сведения о группе студентов в формате:

- номер группы;

- запись о каждом студенте группы содержит следующие сведения: фамилия, имя, отчество, год рождения, оценки по пяти предметам.

Переписать данные файла input.txt в файл output.txt, отсортировав их в алфавитном порядке по фамилии, имени, отчеству методом выбора.

**Код программы**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

#define ini(type, n) \

type n; \

in >> n;

ofstream out("output.txt");

ifstream in("input.txt");

// Информационное поле студента

struct inf {

string secondName;

string firstName;

string thirdName;

int year;

int marks[5];

void scan();

void print();

};

// Функция структуры выводит все данные о студенте

void inf::print(){

out << secondName << ' ' << firstName << ' ' << thirdName << ' ' << year <<' ';

forn(j, 5) out << marks[j] << ' ';

out << endl;

}

// Функция структуры считывает все данные о студенте

void inf::scan(){

in >> secondName >> firstName >> thirdName >> year;

forn(j,5) in >> marks[j];

}

// Функция производит сортировку данных группы по

// фамилиям, именам, отчествам студентов методом выбора

void f(vector<inf> &a, int n) {

forn(i, n-1) {

int num = i;

// Начиная с i-ого элемента ищем минимальный элемент

// из всех элементов впереди i-ого и меняем местами найденный

// элемент и i-ый.

for (int j = i + 1; j < a.size(); j++) {

if (a[j].secondName < a[num].secondName) num = j;

else if (a[j].firstName < a[num].firstName) num = j;

else if (a[j].thirdName < a[num].thirdName) num = j;

}

swap(a[i], a[num]);

}

}

int main() {

ini(int, n);

vector<inf> vec(n);

forn(i, n) vec[i].scan();

f(vec, n);

forn(i, n) vec[i].print();

fclose(stdout);

in.close();

out.close();

return 0;

}

**Сортировки**

// Функция структуры считывает все данные о студенте

void inf::scan(){

in >> secondName >> firstName >> thirdName >> year;

forn(j,5) in >> marks[j];

}

// Функция производит сортировку данных группы по

// фамилиям, именам, отчествам студентов методом выбора

void f(vector<inf> &a, int n) {

forn(i, n-1) {

int num = i;

// Начиная с i-ого элемента ищем минимальный элемент

// из всех элементов впереди и меняем местами найденный

// элемент и i-ый.

for (int j = i + 1; j < a.size(); j++) {

if (a[j].secondName < a[num].secondName) num = j;

else if (a[j].secondName == a[num].secondName && a[j].firstName < a[num].firstName) num = j;

else if (a[j].firstName == a[num].firstName && a[j].thirdName < a[num].thirdName) num = j;

}

swap(a[i], a[num]);

}

}

int main() {

ini(int, n);

vector<inf> vec(n);

forn(i, n) vec[i].scan();

f(vec, n);

forn(i, n) vec[i].print();

in.close();

out.close();

return 0;

}

**Пример(ы)**

|  |
| --- |
| input.txt |
| 6  Павлов Витор Геогригевич 1994 4 3 2 3 4  Рапутян Лоар Виарович 1990 3 4 5 5 4  Павлов Витор Геогригевич 1500 4 5 5 5 5  Рапутян Лоа Ювирович 1990 3 4 4 3 3  Парус Гаргуро Тирнаолович 2000 3 4 2 2 1  Аава Аанг Альбрусович 1000 1 1 1 1 1 |

**Сортировки**

|  |
| --- |
| output.txt |
| Аава Аанг Альбрусович 1000 1 1 1 1 1  Павлов Витор Геогригевич 1500 4 5 5 5 5  Павлов Витор Геогригевич 1994 4 3 2 3 4  Парус Гаргуро Тирнаолович 2000 3 4 2 2 1  Рапутян Лоа Ювирович 1990 3 4 4 3 3  Рапутян Лоар Виарович 1990 3 4 5 5 4 |

**Сортировки**

**Упражнение I, задание 4**

В файле input.txt содержатся сведения о группе студентов в формате:

- номер группы;

- запись о каждом студенте группы содержит следующие сведения: фамилия, имя, отчество, год рождения, оценки по пяти предметам.

Переписать данные файла input.txt в файл output.txt, отсортировав их по убыванию суммы оценок методом «пузырька».

**Код программы**

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

ofstream out("output.txt");

ifstream in("input.txt");

struct inf {

string secondName, firstName, thirdName;

int year, marks;

void scan();

void print();

};

void inf::print(){

out << secondName << ' ' << firstName << ' ' << thirdName << ' ' << year << ' ' << marks;

out << endl;

}

void inf::scan(){

int a;

marks = 0;

in >> secondName >> firstName >> thirdName >> year;

forn(j,5) {

in >> a;

marks+= a;

}

}

// Функция сортирует данные студентов по убыванию

// суммы оценок методом пузырька

void f(vector<inf> &a, int n) {

for (int i = a.size() - 1; i >= 1; i--) {

for (int j = 1; j <= i; j++) {

if (a[j].marks < a[j-1].marks)

swap(a[j], a[j-1]);

}

}

}

int main() {

vector<inf> vec;

// Считывание данных при неизвестном их количестве

forn(i, vec.size()+1) {

if (!in.eof()) vec.resize(vec.size()+1);

else break;

vec[i].scan();

}

// Сортировка и вывод в файл

f(vec, vec.size());

forn(i, vec.size())

vec[i].print();

in.close();

out.close();

return 0;

}

**Сортировки**

void f(vector<inf> &a, int n) {

for (int i = a.size() - 1; i >= 1; i--) {

for (int j = 1; j <= i; j++) {

if (a[j].marks < a[j-1].marks)

swap(a[j], a[j-1]);

}

}

}

int main() {

vector<inf> vec;

// Считывание данных при неизвестном их количестве

forn(i, vec.size()+1) {

if (!in.eof()) vec.resize(vec.size()+1);

else break;

vec[i].scan();

}

// Сортировка и вывод в файл

f(vec, vec.size());

forn(i, vec.size())

vec[i].print();

in.close();

out.close();

return 0;

}

**Пример(ы)**

|  |
| --- |
| input.txt |
| Павлов Витор Геогригевич 1994 4 3 2 3 4  Рапутян Лоар Виарович 1990 3 4 5 5 4  Павлов Витор Геогригевич 1500 4 5 5 5 5  Рапутян Лоа Ювирович 1990 3 4 4 3 3  Парус Гаргуро Тирнаолович 2000 3 4 2 2 1  Аава Аанг Альбрусович 1000 1 1 1 1 1 |
| output.txt |
| Аава Аанг Альбрусович 1000 5  Парус Гаргуро Тирнаолович 2000 12  Павлов Витор Геогригевич 1994 16  Рапутян Лоа Ювирович 1990 17  Рапутян Лоар Виарович 1990 21  Павлов Витор Геогригевич 1500 24 |

**Сортировки**

**Упражнение II, задача 20**

Дана матрица размерностью n х n, содержащая целые числа. Отсортировать диагонали матрицы, расположенные выше побочной, по возрастанию элементов, а диагонали матрицы, расположенные ниже побочной, по убыванию элементов методом вставки.

**Код программы**

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

#define forn(i,n) for(int i = 0; i < n; i++)

#define ll long long

#define ini(type, n) \

type n; \

in >> n;

ofstream out("output.txt");

ifstream in("input.txt");

// Функция сортировки массива методом вставок

void sorting(vector<int> &a, int l, int r) {

int i, j, temp;

for (i = l+1; i <= r; i++) {

temp = a[i];

for (j = i - 1; j >= l; j--) {

// В зависимости от расположения сортируемого массива,

// будут разные условия для выхода из цикла (первое -

// для элементов выше побочной диагонали, второе -

// для элементов ниже побочной диагонали)

if (l == 0 && a[j] > temp)

break;

else if (r == a.size()-1 && a[j] < temp)

break;

a[j + 1] = a[j];

a[j] = temp;

}

}

}

int main() {

ini(int, n);

vector<vector<int>> vec(n, vector<int>(n));

forn(i, n)

forn(j, n)

in >> vec[i][j];

//Сортируем элементы выше побочной диагонали

for (int i = 0; i < vec.size()-1; i++) {

sorting(vec[i], 0, n-i-2);

}

//Сортируем элементы ниже побочной диагонали

for (int i = 1; i < vec.size(); i++) {

sorting(vec[i], n - i, n-1);

}

forn (i, n) {

forn(j, n-i-1)

out << vec[i][j] << ' ';

out << "X ";

for (int j = n-i; j < n; j++)

out << vec[i][j] << ' ';

out << endl;

}

in.close();

out.close();

return 0;

}

**Сортировки**

forn(i, n)

forn(j, n)

in >> vec[i][j];

//Сортируем элементы выше побочной диагонали

for (int i = 0; i < vec.size()-1; i++) {

sorting(vec[i], 0, n-i-2);

}

//Сортируем элементы ниже побочной диагонали

for (int i = 1; i < vec.size(); i++) {

sorting(vec[i], n - i, n-1);

}

forn (i, n) {

forn(j, n-i-1)

out << vec[i][j] << ' ';

out << "X ";

for (int j = n-i; j < n; j++)

out << vec[i][j] << ' ';

out << endl;

}

in.close();

out.close();

return 0;

}

**Пример(ы)**

|  |  |
| --- | --- |
| input.txt | output.txt |
| 8  3 0 1 5 3 4 2 1  4 3 2 9 5 2 0 0  3 8 1 1 1 4 6 7  2 3 8 1 2 5 2 1  9 5 2 0 0 5 2 1  3 2 9 5 2 0 8 1  3 8 1 2 5 2 2 0  3 8 1 1 1 4 6 7 | 5 4 3 3 2 1 0 X  9 5 4 3 2 2 X 0  8 3 1 1 1 X 6 7  8 3 2 1 X 1 2 5  9 5 2 X 0 1 2 5  3 2 X 0 1 2 5 8  3 X 0 1 2 2 2 5  X 1 1 1 4 6 7 8 |

**Классы**

**Упражнение I, задача 11**

Создать класс Time, с полями hours, minutes, seconds, представляющими собой часы, минуты и секунды. Данный класс должен позволять выводить информацию о текущем времени, вычислять время через заданное количество часов, минут и секунд, а также вычислять время, прошедшее между двумя заданными моментами.

Детальную структуру класса продумайте самостоятельно.

**Замечание:** В конце файла вывести максимальное и минимальное время.

**Код программы**

**main.cpp**

#include "classtime.h"

#define forn(i, n) for (int i = 0; i < n; i++)

ifstream in("input.txt");

int main()

{

// Создание экземпляров класса classTime

classTime basicTime, compareTime, compareTime2;

while (!in.eof()) {

// Считываем первый символ строки

// Если считываемый символ является \*, то вся

// информация после этого символа - это комментарий,

// и считывать ее не надо

char typingCommand;

if (in.peek() == '#') {

in.ignore(256, '\n');

continue;

}

else

in >> typingCommand;

// В зависимости от введенной команды будут

// выполняться определенные действия.

// Если команда не находится в пределах 1 и 5,

// то она введена не верно и выведется сообщение

// об ошибке

switch(typingCommand) {

case '1': basicTime.setTheTime(in);

break;

case '2': basicTime.showTheTimeFormed();

break;

case '3': compareTime.setTheTime(in);

basicTime.afterTheTime(compareTime);

break;

case '4': compareTime.setTheTime(in);

compareTime2.setTheTime(in);

basicTime.betweenTheTime(compareTime, compareTime2);

break;

case '5': classTime::showCurrentMaxMin();

return 0;

default: cout << "string [" << classTime::stringNumber << "] unknown command: " << typingCommand << endl;

in.ignore(256, '\n');

}

classTime::stringNumber++;

}

}

**Классы**

**main.cpp**

case '3': compareTime.setTheTime(in);

basicTime.afterTheTime(compareTime);

break;

case '4': compareTime.setTheTime(in);

compareTime2.setTheTime(in);

basicTime.betweenTheTime(compareTime, compareTime2);

break;

case '5': classTime::showCurrentMaxMin();

return 0;

default: cout << "string [" << classTime::stringNumber << "] unknown command: " << typingCommand << endl;

in.ignore(256, '\n');

}

classTime::stringNumber++;

}

}

**classtime.h**

// Подключение заголовочного файла единожды

#ifndef CLASSTIME\_H

#define CLASSTIME\_H

#include <iostream>

#include <vector>

#include <ctime>

#include <fstream>

#include <string>

using namespace std;

class classTime

{

public:

// Конструкторы класса

classTime() : hours(0), minutes(0), seconds(0) {}

classTime(int h, int m, int s) : hours(h), minutes(m), seconds(s) {}

// Методы класса

void showTheTime();

void showTheTimeFormed();

void setTheTime(ifstream &in);

void afterTheTime(classTime varTime1);

void betweenTheTime(classTime varTime1, classTime varTime2);

static void showCurrentMaxMin();

// Перегруженные операторы

classTime operator +(classTime varTime1);

classTime operator -(classTime varTime1);

bool operator <(classTime varTime1);

bool operator >(classTime varTime1);

// Статические поля класса

static classTime maxTime;

static classTime minTime;

static int stringNumber;

private:

// Поля класса

int hours;

int minutes;

int seconds;

};

#endif

**Классы**

**classtime.h**

static void showCurrentMaxMin();

// Перегруженные операторы

classTime operator +(classTime varTime1);

classTime operator -(classTime varTime1);

bool operator <(classTime varTime1);

bool operator >(classTime varTime1);

// Статические поля класса

static classTime maxTime;

static classTime minTime;

static int stringNumber;

private:

// Поля класса

int hours;

int minutes;

int seconds;

};

#endif

**classtime.cpp** (Подкючение хедера, дефайнов, потока и инциализация статических переменных)

#include "classtime.h"

#define forn(i, n) for (int i = 0; i < n; i++)

#define ll long long

ofstream out("output.txt");

// Инициализация статических полей класса

int classTime::stringNumber = 1;

classTime classTime::maxTime;

classTime classTime::minTime(24, 60, 60);

// Статическая функция выводит в поток макс. и мин.

// время, которое было найдено (запускается в конце

// программы)

void classTime::showCurrentMaxMin() {

out << "max time: ";

maxTime.showTheTime();

out << endl;

out << "min time: ";

minTime.showTheTime();

out << endl;

}

// Функция устанавливает время объекта

// согласно данным, которые ввел пользователь

void classTime::setTheTime(ifstream &in)

{

string strTime1;

int intTime2 = 0;

in >> strTime1;

forn(i, strTime1.size()) {

// Во время считывания времени, все должно

// быть по образцу ЧЧ:ММ:СС. Если считанное

// время не удовлетворяет образцу, то будет

// выведенно сообщение об ошибке и запись не произойдет

if (i < 8) {

if (!(isdigit(strTime1[i]) && ((i + 1) % 3 != 0))) {

if (!((strTime1[i] == ':') && ((i + 1) % 3 == 0))) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

else

continue;

}

else

intTime2 = intTime2\*10 + (strTime1[i] - '0');

}

else {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

}

// Если данные введены некорректно

// (минут или секунд больше 60), то

// присваивания не произойдет и программа

// перейдет к следующему действию

if ((intTime2 / 10000 >= 24) || ((intTime2 / 100) % 100 >= 60) || (intTime2 % 100 >= 60)) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

// Устанавливаем считанное время полям

// текущего объекта и заодно обновляем

// макс. и мин. время

this->hours = intTime2 / 10000;

this->minutes = (intTime2 / 100) % 100;

this->seconds = intTime2 % 100;

classTime compareTime(hours, minutes, seconds);

if (compareTime > maxTime)

maxTime = compareTime;

if (compareTime < minTime)

minTime = compareTime;

}

// Функция выводит поля текущего экземпляра класса

void classTime::showTheTime()

{

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод showCurrentMaxMin)

// Статическая функция выводит в поток макс. и мин.

// время, которое было найдено (запускается в конце

// программы)

void classTime::showCurrentMaxMin() {

out << "max time: ";

maxTime.showTheTime();

out << endl;

out << "min time: ";

minTime.showTheTime();

out << endl;

}

// Функция устанавливает время объекта

// согласно данным, которые ввел пользователь

void classTime::setTheTime(ifstream &in)

{

string strTime1;

int intTime2 = 0;

in >> strTime1;

forn(i, strTime1.size()) {

// Во время считывания времени, все должно

// быть по образцу ЧЧ:ММ:СС. Если считанное

// время не удовлетворяет образцу, то будет

// выведенно сообщение об ошибке и запись не произойдет

if (i < 8) {

if (!(isdigit(strTime1[i]) && ((i + 1) % 3 != 0))) {

if (!((strTime1[i] == ':') && ((i + 1) % 3 == 0))) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

else

continue;

}

else

intTime2 = intTime2\*10 + (strTime1[i] - '0');

}

else {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

}

// Если данные введены некорректно

// (минут или секунд больше 60), то

// присваивания не произойдет и программа

// перейдет к следующему действию

if ((intTime2 / 10000 >= 24) || ((intTime2 / 100) % 100 >= 60) || (intTime2 % 100 >= 60)) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

// Устанавливаем считанное время полям

// текущего объекта и заодно обновляем

// макс. и мин. время

this->hours = intTime2 / 10000;

this->minutes = (intTime2 / 100) % 100;

this->seconds = intTime2 % 100;

classTime compareTime(hours, minutes, seconds);

if (compareTime > maxTime)

maxTime = compareTime;

if (compareTime < minTime)

minTime = compareTime;

}

// Функция выводит поля текущего экземпляра класса

void classTime::showTheTime()

{

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**Классы**

out << "min time: ";

minTime.showTheTime();

out << endl;

}

// Функция устанавливает время объекта

// согласно данным, которые ввел пользователь

void classTime::setTheTime(ifstream &in)

{

string strTime1;

int intTime2 = 0;

in >> strTime1;

forn(i, strTime1.size()) {

// Во время считывания времени, все должно

// быть по образцу ЧЧ:ММ:СС. Если считанное

// время не удовлетворяет образцу, то будет

// выведенно сообщение об ошибке и запись не произойдет

if (i < 8) {

if (!(isdigit(strTime1[i]) && ((i + 1) % 3 != 0))) {

if (!((strTime1[i] == ':') && ((i + 1) % 3 == 0))) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

else

continue;

}

else

intTime2 = intTime2\*10 + (strTime1[i] - '0');

}

else {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

}

// Если данные введены некорректно

// (минут или секунд больше 60), то

// присваивания не произойдет и программа

// перейдет к следующему действию

if ((intTime2 / 10000 >= 24) || ((intTime2 / 100) % 100 >= 60) || (intTime2 % 100 >= 60)) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

// Устанавливаем считанное время полям

// текущего объекта и заодно обновляем

// макс. и мин. время

this->hours = intTime2 / 10000;

this->minutes = (intTime2 / 100) % 100;

this->seconds = intTime2 % 100;

classTime compareTime(hours, minutes, seconds);

if (compareTime > maxTime)

maxTime = compareTime;

if (compareTime < minTime)

minTime = compareTime;

}

// Функция выводит поля текущего экземпляра класса

void classTime::showTheTime()

{

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод setTheTime)

// Метод устанавливает время объекта

// согласно данным, которые ввел пользователь

void classTime::setTheTime(ifstream &in)

{

string strTime1;

int intTime2 = 0;

in >> strTime1;

forn(i, strTime1.size()) {

// Во время считывания времени, все должно

// быть по образцу ЧЧ:ММ:СС. Если считанное

// время не удовлетворяет образцу, то будет

// выведенно сообщение об ошибке и запись не произойдет

if (i < 8) {

if (!(isdigit(strTime1[i]) && ((i + 1) % 3 != 0))) {

if (!((strTime1[i] == ':') && ((i + 1) % 3 == 0))) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

else

continue;

}

else

intTime2 = intTime2\*10 + (strTime1[i] - '0');

}

else {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

}

// Если данные введены некорректно

// (минут или секунд больше 60), то

// присваивания не произойдет и программа

// перейдет к следующему действию

if ((intTime2 / 10000 >= 24) || ((intTime2 / 100) % 100 >= 60) || (intTime2 % 100 >= 60)) {

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

// Устанавливаем считанное время полям

// текущего объекта и заодно обновляем

// макс. и мин. время

this->hours = intTime2 / 10000;

this->minutes = (intTime2 / 100) % 100;

this->seconds = intTime2 % 100;

classTime compareTime(hours, minutes, seconds);

if (compareTime > maxTime)

maxTime = compareTime;

if (compareTime < minTime)

minTime = compareTime;

}

// Функция выводит поля текущего экземпляра класса

void classTime::showTheTime()

{

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**Классы**

out << stringNumber << "> " << "wrong parameters: " << strTime1 << endl;

return;

}

// Устанавливаем считанное время полям

// текущего объекта и заодно обновляем

// макс. и мин. время

this->hours = intTime2 / 10000;

this->minutes = (intTime2 / 100) % 100;

this->seconds = intTime2 % 100;

classTime compareTime(hours, minutes, seconds);

if (compareTime > maxTime)

maxTime = compareTime;

if (compareTime < minTime)

minTime = compareTime;

}

// Функция выводит поля текущего экземпляра класса

void classTime::showTheTime()

{

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод showTheTime)

// Метод выводит поля текущего экземпляра класса

void classTime::showTheTime() {

out << hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10;

}

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed()

{

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод showTheTimeFormed)

// Функция выводит поля текущего экземпляра класса

// с нужным форматированием

void classTime::showTheTimeFormed() {

out << stringNumber << "> " << "current time is "

<< hours / 10 << hours % 10 << ':'

<< minutes / 10 << minutes % 10 << ':'

<< seconds / 10 << seconds % 10 << endl;

}

// Функция вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает функцию вывода результата

void classTime::afterTheTime(classTime varTime1)

{

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод afterTheTime)

// Метод вычисляет сколько будет времени

// после заданного кол-ва часов, минут, секунд

// и вызывает метод вывода результата

void classTime::afterTheTime(classTime varTime1) {

classTime varTime2 = \*this;

varTime2 = varTime1 + varTime2;

out << stringNumber << "> " << "the time after ";

varTime1.showTheTime();

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**Классы**

out << " will be ";

varTime2.showTheTime();

out << endl;

}

// Функция вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает функцию вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2)

{

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Метод betweenTheTime)

// Метод вычисляет сколько должно пройти

// времени в указанном пользователем промежутке

// и вызывает метод вывода результата

void classTime::betweenTheTime(classTime varTime1, classTime varTime2) {

out << stringNumber << "> " << "the time between ";

varTime1.showTheTime();

out << " and ";

varTime2.showTheTime();

out << " is ";

if (varTime2 < varTime1)

varTime2 = varTime1 - varTime2;

else

varTime2 = varTime2 - varTime1;

varTime2.showTheTime();

out << endl;

}

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1)

{

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1)

{

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**classtime.cpp** (Перегрузки операторов)

// Перегрузка оператора <

bool classTime::operator <(classTime varTime1) {

return (hours\*10000 + minutes\*100 + seconds < varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора >

bool classTime::operator >(classTime varTime1) {

return (hours\*10000 + minutes\*100 + seconds > varTime1.hours\*10000 + varTime1.minutes\*100 + varTime1.seconds);

}

// Перегрузка оператора + c учетом ограничений

classTime classTime::operator +(classTime varTime1) {

varTime1.hours = ((hours + varTime1.hours) % 24 + (minutes + varTime1.minutes + (seconds + varTime1.seconds) / 60) / 60) % 24;

varTime1.minutes = ((minutes + varTime1.minutes) % 60 + (seconds + varTime1.seconds) / 60) % 60;

varTime1.seconds = (seconds + varTime1.seconds) % 60;

return varTime1;

}

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1)

{

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**Классы**

// Перегрузка оператора - c учетом ограничений

classTime classTime::operator -(classTime varTime1) {

if (seconds - varTime1.seconds >= 0)

varTime1.seconds = seconds - varTime1.seconds;

else {

varTime1.seconds = 60 + seconds - varTime1.seconds;

varTime1.minutes++;

}

if (minutes - varTime1.minutes >= 0)

varTime1.minutes = minutes - varTime1.minutes;

else {

varTime1.minutes = 60 + minutes - varTime1.minutes;

varTime1.hours++;

}

varTime1.hours = abs(hours - varTime1.hours)%24;

return varTime1;

}

**Пример(ы)**

|  |  |
| --- | --- |
| input.txt | output.txt |
| # 1 - set the time  # 2 - show the time  # 3 - the time after settled  # 4 - the time between settled  # 5 - exit  1 12:00:20  2  3 02:20:40  Error message  2  1 24:00:00  1 23:8942:1245  1 01:50:20  4 20:30:10 15:10:50  2  9  5 | 2> current time is 12:00:20  3> the time after 02:20:40 will be 14:21:00  5> current time is 12:00:20  6> wrong parameters: 24:00:00  7> wrong parameters: 23:8942:1245  9> the time between 20:30:10 and 15:10:50 is 05:19:20  10> current time is 01:50:20  max time: 20:30:10  min time: 01:50:20 |
| stdout | |
| string [4] unknown command: E  string [11] unknown command: 9 | |

**Списки**

**Упражнение I, задача 12**

Создать список из слов. Подсчитать количество слов, совпадающих с последним словом. Удалить все такие слова из списка, оставив одно последнее.

**Код программы**

**main.cpp**

#include "pstack.h"

ifstream in("input.txt");

int main() {

PStack custom2, custom1;

string s;

// Заполнение стека считанной строкой

while (!in.eof()) {

in >> s;

custom2.push(s);

}

int num = 0;

// Сохраняем верхний элемент стека и удаляем его из

// второго стека и добавляем его в первый, так как

// нам нужно сохранить его, а его совпадения удалить

string correctTop = custom2.pop();

custom1.push(correctTop);

// Пока второй стек не будет пуст, будем брать верхний

// элемент и сравнивать с correctTop, и если они будут

// различны, то добавим элемент из custom2 в custom3

// Таким образом, все совпадающие элементы с correctTop

// не войдут в новый стек и получится такойже стек custom3,

// но уже без элементов, совпадающих с верхним

while (!custom2.empty()) {

string compare = custom2.pop();

if (correctTop == compare) {

num++;

}

else {

custom1.push(compare);

}

}

while (!custom1.empty()) {

cout << custom1.pop() << ' ';

}

cout << endl << num - 1 << endl;

return 0;

}

**Списки**

**pstack.h**

#ifndef PSTACK

#define PSTACK

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class PStack

{

private:

// Информационное поле элемента стека, которое содержит

// значение и указатель на следующий элемент

struct Element{

string value;

Element\* next;

// Конструктор информационного поля элемента

Element(string value, Element\* next) {

this->value = value;

this->next = next;

}

};

// Каждый экземпляр класса хранит указатель на последний

// добавленный элемент и размер стека

Element\* head;

int sizeOfStack;

public:

// Конструктор, деструктор

PStack() : head(nullptr), sizeOfStack(0){}

~PStack(){}

// Метод проверки на пустоту

bool empty(){

return head == nullptr;

}

// Метод добавления элемента в стек

void push(string u){

// Увеличиваем размер стека на 1 и смещаем

// указатель head на новый элемент

sizeOfStack++;

head = new Element(u, head);

}

// Метод взятия элемента из стека

string pop() {

// Если стек не пуст, то уменьшаем размер стека

// и переприсваиваем указатель head предыдущему после

// текущего верхнего элемента и возвращаем его значение

if (empty()) return "";

sizeOfStack--;

Element \*r = head;

string i = r->value;

head = r->next;

delete r;

return i;

}

// Метод возвращает верхний элемент стека без удаления

string top(){

if (empty()) return "";

return head->value;

}

};

#endif

**Списки**

**pstack.h**

// и переприсваиваем указатель head предыдущему после

// текущего верхнего элемента и возвращаем его значение

if (empty()) return "";

sizeOfStack--;

Element \*r = head;

string i = r->value;

head = r->next;

delete r;

return i;

}

// Метод возвращает верхний элемент стека без удаления

string top(){

if (empty()) return "";

return head->value;

}

};

#endif

**Пример(ы)**

|  |  |
| --- | --- |
| input.txt | stdout |
| aaa bb aaa aa bbb a bb b aaa aa bb | aaa aaa aa bbb a b aaa aa bb  2 |
| . . . . S . . T . A . C . . K . . . | S T A C K .  12 |

**Списки**

**Упражнение I, задача 12**

Создать очередь из слов. Подсчитать количество слов, совпадающих с последним словом. Удалить все такие слова из списка, оставив одно последнее.

**Код программы**

**main.cpp**

#include "pqueue.h"

ifstream in("input.txt");

int main() {

PQueue custom1, custom2;

string topElement, compareElement;

// Каждый новый считываемый элемент будет последним

while (!in.eof()) {

in >> topElement;

custom1.PPut(topElement);

}

int num = 0;

// Производим те же действие, что и со стеком

while (!custom1.PEmpty()) {

compareElement = custom1.PGet();

if (topElement == compareElement) {

num++;

}

else {

custom2.PPut(compareElement);

}

}

// Кладем в очередь последний считанный элемент,

// так как нам нужно его сохранить и вычитаем из

// num 1, потому что программа посчитала все совпадения

custom2.PPut(topElement);

cout << num - 1 << endl;

while (!custom2.PEmpty()) {

cout << custom2.PGet() << ' ';

}

return 0;

}

**Списки**

**pqueue.h**

#ifndef PQUEUE

#define PQUEUE

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class PQueue

{

private:

// Данные, которые содержит элемент очереди

struct Element{

string value;

Element\* next;

Element(string value) {

this->value = value;

this->next = nullptr;

}

};

// Указатели на начало и конец очереди

Element\* head;

Element\* tail;

public:

PQueue() : head(nullptr), tail(nullptr){}

~PQueue(){}

// Метод проверки на пустоту очереди

bool PEmpty(){

return head == nullptr;

}

// Метод взятия элемента из очереди

string PGet() {

if (PEmpty()) return "";

// Принцип работы метода такойже, как у стека,

// но в очереди нужно еще обнулить указатель tail

Element \*t = head;

string i = t->value;

head = t->next;

if (head == NULL) tail = NULL;

delete t;

return i;

}

// Метод добавления элемента в очередь

void PPut(string value){

// Добавление происходит в конец очереди, так как

// в начале находится первый добавленный элемент

// Для того, чтобы добавить элемент, нужно переместить

// указатель tail на новый, и если это не первый добавленный

// элемент, то указатель next будет указывать на следующий

// в очереди элемент, иначе next будет инициализирован

// nullptr, а head будет указывать на этот единственный элемент

Element \*t = tail;

tail = new Element(value);

if (!head) head = tail;

else t->next = tail;

}

};

#endif

**Списки**

// Метод добавления элемента в очередь

void PPut(string value){

// Добавление происходит в конец очереди, так как

// в начале находится первый добавленный элемент

// Для того, чтобы добавить элемент, нужно переместить

// указатель tail на новый, и если это не первый добавленный

// элемент, то указатель next будет указывать на следующий

// в очереди элемент, иначе next будет инициализирован

// nullptr, а head будет указывать на этот единственный элемент

Element \*t = tail;

tail = new Element(value);

if (!head) head = tail;

else t->next = tail;

}

};

#endif

**Пример(ы)**

|  |  |
| --- | --- |
| input.txt | stdout |
| aaa bb aaa aa bbb a bb b aaa aa bb | aaa aaa aa bbb a b aaa aa bb  2 |
| . . . . S . . T . A . C . . K . . . | S T A C K .  12 |

**Списки**

**Упражнение I, задача 12**

Создать однонаправленный список из слов. Подсчитать количество слов, совпадающих с последним словом. Удалить все такие слова из списка, оставив одно последнее.

**Код программы**

**main.cpp**

#include "psinglelist.h"

int main() {

int num = 0;

string s, topString;

// Заводим список типа string

PSingleList<string> custom;

// Считываем все элементы в строке

custom.PRowInsert();

// Удаление всех элементов, равных topString

cout << custom.PRowRemove() << endl;

custom.PPrint();

}

**psinglelist.h** (Класс исключения)

**psinglelist.h** (Структура класса)

// Класс исключения, который выдает сообщение

// об ошибке при вызове

class ListException: public exception

{

public:

ListException(const string & message = ""): exception(message.c\_str()) {}

};

// Класс является шаблонным

template <class item>

class PSingleList

{

protected:

// Структура элемента списка описывается

// значением элемента и указателем на следующий

struct Element{

item value;

Element\* next;

Element(item x) : value(x), next(0) {}

};

// Остальные данные класса хранят в себе указатель на

// начальный элемент, размер контейнера, значение верхнего

// элемента списка и метод, производящий поиск по

// всем содержащимся элементам

Element \*head;

int sizeOfSL;

item topElement;

// Метод возвращает элемент списка под нужным индексом

Element \*PFind(int index) {

if ((index < 1) || (index > sizeOfSL)) {

return NULL;

}

// В начале указатель устанавливается на head,

// затем смещаем указатель до нужного индекса

Element \*cur = head;

for (int i = 1; i < index; i++) {

cur = cur->next;

}

return cur;

}

public:

// Конструктор и деструктор

PSingleList() : head(0), sizeOfSL(0){}

~PSingleList() {

while(!PEmpty()) {

PRemove(1);

}

}

// Функция проверяет список на пустоту

bool PEmpty() {

return head == 0;

}

// Функция возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Функция возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Функция добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Функция добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Функция удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Функция печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

// Класс является шаблонным

template <class item>

class PSingleList

{

protected:

// Структура элемента списка описывается

// значением элемента и указателем на следующий

struct Element{

item value;

Element\* next;

Element(item x) : value(x), next(0) {}

};

// Остальные данные класса хранят в себе указатель на

// начальный элемент, размер контейнера, значение верхнего

// элемента списка и метод, производящий поиск по

// всем содержащимся элементам

Element \*head;

int sizeOfSL;

item topElement;

// Метод возвращает элемент списка под нужным индексом

Element \*PFind(int index) {

if ((index < 1) || (index > sizeOfSL)) {

return NULL;

}

// В начале указатель устанавливается на head,

// затем смещаем указатель до нужного индекса

Element \*cur = head;

for (int i = 1; i < index; i++) {

cur = cur->next;

}

return cur;

}

public:

// Конструктор и деструктор

PSingleList() : head(0), sizeOfSL(0){}

~PSingleList() {

while(!PEmpty()) {

PRemove(1);

}

}

// Функция проверяет список на пустоту

bool PEmpty() {

return head == 0;

}

// Функция возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Функция возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Функция добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Функция добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Функция удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Функция печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**Списки**

};

// Остальные данные класса хранят в себе указатель на

// начальный элемент, размер контейнера, значение верхнего

// элемента списка и метод, производящий поиск по

// всем содержащимся элементам

Element \*head;

int sizeOfSL;

item topElement;

// Метод возвращает элемент списка под нужным индексом

Element \*PFind(int index) {

if ((index < 1) || (index > sizeOfSL)) {

return NULL;

}

// В начале указатель устанавливается на head,

// затем смещаем указатель до нужного индекса

Element \*cur = head;

for (int i = 1; i < index; i++) {

cur = cur->next;

}

return cur;

}

public:

// Конструктор и деструктор

PSingleList() : head(0), sizeOfSL(0){}

~PSingleList() {

while(!PEmpty()) {

PRemove(1);

}

}

// Функция проверяет список на пустоту

bool PEmpty() {

return head == 0;

}

// Функция возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Функция возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Функция добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Функция добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Функция удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Функция печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Конструктор и деструктор класса)

PSingleList() : head(0), sizeOfSL(0){}

~PSingleList() {

while(!PEmpty()) {

PRemove(1);

}

}

// Функция проверяет список на пустоту

bool PEmpty() {

return head == 0;

}

// Функция возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Функция возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Функция добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Функция добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Функция удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Функция печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PEmpty)

// Метод проверяет список на пустоту

bool PEmpty() {

return head == 0;

}

// Метод возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Метод возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Метод добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Метод добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PSize)

// Метод возвращает размер списка

int PSize() {

return sizeOfSL;

}

// Метод возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Метод добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Метод добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**Списки**

**psinglelist.h** (Метод PSize)

// Метод возвращает элемент с нужным индексом

item PGet(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

throw ListException("PGet error");

}

// Для нахождения элемент с нужным индексом

// запускается метод PFind

Element \*r = PFind(index);

item i = r-> value;

return i;

}

// Метод добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Метод добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PInsert)

// Метод добавляет элемент в список

void PInsert(int index, item value){

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// Обновление верхнего элемента

if (index == sizeOfSL + 1)

topElement = value;

Element \*newPtr = new Element(value);

sizeOfSL = PSize() + 1;

// Если список пуст, то указатель добавляемого

// элемента будет указаывать на nullptr, так как

// head инициализирован nullptr, а указателю head

// присваивается этот новый элемент

if (index == 1) {

newPtr->next = head;

head = newPtr;

}

// Если же список не пуст, то мы должны найти

// элемент, после которого стоит вставляемый элемент

// и скопировать его указатель нашему новому

// элементу, в то время как его указатель установить

// на наш новый элемент

else {

Element \*prev = PFind(index - 1);

newPtr->next = prev->next;

prev->next = newPtr;

}

}

// Метод добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**Списки**

**psinglelist.h** (Метод PRowInsert)

// Метод добавляет все элементы из файла в список

void PRowInsert(){

item s;

Element \*current = head;

Element \*previous;

// Проводим те же действия, что и с методом PInsert,

// только считываем мы подряд, поэтому нам не нужно

// использовать поиск по индексу, так как мы

// будем сохрнанять значение предыдущего элемента

while (!in.eof()) {

in >> s;

topElement = s;

sizeOfSL = PSize() + 1;

current = new Element(s);

if (sizeOfSL == 1) {

current->next = head;

head = current;

previous = current;

}

else {

previous->next = current;

previous = previous->next;

}

}

}

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PRemove)

// Метод удаляет элемент из списка

void PRemove(int index) {

if ((index < 1) || (index > sizeOfSL + 1)) {

return;

}

// При удалении элемента мы переприсваиваем указатель

// предыдущего элемента на следующий за удаленным

Element \*cur;

sizeOfSL--;

if (index == 1) {

cur = head;

head = head->next;

}

else {

Element \*prev = PFind(index - 1);

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**Списки**

cur = prev->next;

prev->next = cur->next;

}

cur->next = NULL;

delete cur;

}

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PPrint)

// Метод печатает весь список с начала

void PPrint() {

for (Element \*cur = head; cur != NULL; cur = cur-> next)

cout << cur->value << ' ';

cout << endl;

}

// Решение задачи

int problemSolver (PSingleList &custom) {

int num = 0;

for (int i = 1; i < custom.PSize(); i++) {

if (topElement == custom.PGet(i)) {

num++;

custom.PRemove(i);

}

}

return num;

}

};

#endif

**psinglelist.h** (Метод PRowRemove)

// Функция удаляет все элементы списка, которые равны topElement

int PRowRemove() {

// Для двух исключений кол-во удаленных элементов

// будет равно 0, и ничего не нужно удалять

if(sizeOfSL == 0 || sizeOfSL == 1) return 0;

int deleted = 0;

Element \*cur = head;

// Запускаем цикл, который будет проверять второй

// следующий элемента от текущего, и если он будет

// равен topElement, то мы его удалим, иначе

// простой перейдем к следующему

while (cur->next->next != nullptr) {

if (cur->next->value == topElement) {

deleted++;

sizeOfSL--;

Element \*tmp = cur->next;

cur->next = cur->next->next;

delete tmp;

}

else cur = cur->next;

}

// В цикле мы не проверяем первый элемент, так как

// идем от него, следовательно нужно его проверить

// является ли он совпадающим с topElement и удалить

// его при необходимости

if (head->value == topElement) {

deleted++;

sizeOfSL--;

Element \*tmp = head;

head = head->next;

delete tmp;

}

return deleted;

}

**Списки**

Element \*tmp = head;

head = head->next;

delete tmp;

}

return deleted;

}

**Пример(ы)**

|  |  |
| --- | --- |
| input.txt | stdout |
| aaa bb aaa aa bbb a bb b aaa aa bb | aaa aaa aa bbb a b aaa aa bb  2 |
| . . . . S . . T . A . C . . K . . . | S T A C K .  12 |

**Наследование**

**Упражнение I, задача 8**

1. Создать абстрактный класс TelephoneDirectory с функциями, позволяющими вывести на экран информацию о записях в телефонном справочнике, а также определить соответствие записи критерию поиска.

2. Создать производные классы: Persona (фамилия, адрес, номер телефона), Organization (название, адрес, телефон, факс, контактное лицо), Friend (фамилия, адрес, номер телефона, дата рождения).

3. Создать базу (массив) из n записей, вывести полную информацию из базы на экран, а также организовать поиск в базе по фамилии.

**Код программы**

**main.cpp**

#include "inheritance.h"

#include "persona.h"

#include "organization.h"

#include "friend.h"

using namespace std;

const int INDEX = 100000;

const int INDEX\_ORG = 1000;

ifstream in ("input.txt");

int main()

{

setlocale( LC\_ALL,"Russian" );

// Создаем массив экземпляров класса

TelephoneDirectory \*customSpreadsheet[1000];

int i = 0;

string A1, A2, A3, A4, A5, A6, A7, A8;

// Считываем данные для Persona

while (!in.eof()) {

in >> A1;

if (A1 == ">Organization")

break;

in >> A2 >> A3;

customSpreadsheet[i++] = new Persona(i + INDEX, A1, A2, A3);

}

// Считываем данные для Organization

while (!in.eof()) {

in >> A1;

if (A1 == ">Friend")

break;

in >> A2 >> A3 >> A4 >> A5 >> A6 >> A7;

customSpreadsheet[i++] = new Organization(i + INDEX\_ORG, i + INDEX, A1, A2, A3, A4, A5, A6, A7);

}

// Считываем данные для Friend

while (!in.eof()) {

in >> A1 >> A2 >> A3 >> A4;

customSpreadsheet[i++] = new Friend(i + INDEX\_ORG, i + INDEX, A1, A2, A3, A4);

}

customSpreadsheet[i] = nullptr;

i = 0;

// Выводим все считанные данные в стандартный поток

while (customSpreadsheet[i] != nullptr) {

customSpreadsheet[i++]->PShow();

}

string searchingElement;

// Выводим данные, удовлетворяющие поиску в стандартный поток

while(true) {

cout << "\n\nВведите фамилию для поиска (exit - для выхода): ";

cin >> searchingElement;

if (searchingElement == "exit") break;

i = 0;

while (customSpreadsheet[i] != nullptr) {

customSpreadsheet[i++]->PCheck(searchingElement);

}

}

return 0;

}

**Наследование**

in >> A2 >> A3 >> A4 >> A5 >> A6 >> A7;

customSpreadsheet[i++] = new Organization(i + INDEX\_ORG, i + INDEX, A1, A2, A3, A4, A5, A6, A7);

}

// Считываем данные для Friend

while (!in.eof()) {

in >> A1 >> A2 >> A3 >> A4;

customSpreadsheet[i++] = new Friend(i + INDEX\_ORG, i + INDEX, A1, A2, A3, A4);

}

customSpreadsheet[i] = nullptr;

i = 0;

// Выводим все считанные данные в стандартный поток

while (customSpreadsheet[i] != nullptr) {

customSpreadsheet[i++]->PShow();

}

string searchingElement;

// Выводим данные, удовлетворяющие поиску в стандартный поток

while(true) {

cout << "\n\nВведите фамилию для поиска (exit - для выхода): ";

cin >> searchingElement;

if (searchingElement == "exit") break;

i = 0;

while (customSpreadsheet[i] != nullptr) {

customSpreadsheet[i++]->PCheck(searchingElement);

}

}

return 0;

}

**inheritance.h**

#ifndef INHERITANCE\_H

#define INHERITANCE\_H

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

using namespace std;

//Абстрактный класс

class TelephoneDirectory {

public:

virtual void PShow() = 0;

virtual void PCheck(string s) = 0;

};

#endif // INHERITANCE\_H

**Наследование**

virtual void PCheck(string s) = 0;

};

#endif // INHERITANCE\_H

**persona.h**

#include "inheritance.h"

// Наследование от абстрактного класса

class Persona: public TelephoneDirectory {

protected:

int ID;

string firstName;

string secondName;

string telephoneNumber;

public:

Persona():

ID(999999),

firstName("UNKNOWN"),

secondName("UNKNOWN"),

telephoneNumber("UNKNOWN") {}

Persona(int A1, string A2, string A3, string A4):

ID(A1),

firstName(A3),

secondName(A2),

telephoneNumber(A4) {}

~Persona() {}

// Метод выводит данные класса в стандартный поток

void PShow() {

cout << "[" << ID << "] "

<< firstName << ' '

<< secondName << ' '

<< telephoneNumber << endl;

}

// Метод возвращает значение поля secondName

string PSecondName() {

return secondName;

}

// Метод проверяет входной аргумент и secondName

// и выводит данные на экран при совпадении

void PCheck(string compareElement) {

if (compareElement == secondName)

PShow();

}

};

**Наследование**

**organization.h**

#ifndef ORGANIZATION\_H

#define ORGANIZATION\_H

#include "persona.h"

class Organization : public TelephoneDirectory {

protected:

int ID;

Persona custom;

string name;

string adress;

string telephoneNumber;

string fax;

public:

Organization():

ID(999999),

name("UNKNOWN"),

adress("UNKNOWN"),

telephoneNumber("UNKNOWN"),

fax("UNKNOWN") {

Persona();

}

Organization(int A0, int A1, string A2, string A3, string A4, string A5, string A6, string A7, string A8):

ID(A0),

name(A2),

adress(A3),

telephoneNumber(A4),

fax(A5) {

custom = Persona(A1, A6, A7, A8);

}

Organization(int A0, Persona A1, string A2, string A3, string A4, string A5):

ID(A0),

custom(A1),

name(A2),

adress(A3),

telephoneNumber(A4),

fax(A5) {

}

~Organization() {}

// Метод выводит данные текущего экземпляра класса и

// экземпляра custom в стандартный поток

void PShow() {

cout << "[" << ID << "] "

<< name << ' '

<< adress << ' '

<< telephoneNumber << ' '

<< fax << " Contact: ";

custom.PShow();

}

// Метод производит поиск по фамилии в поле

// экземпляра класса custom

void PCheck(string compareElement) {

if (compareElement == custom.PSecondName())

PShow();

}

};

#endif // ORGANIZATION\_H

**Наследование**

void PShow() {

cout << "[" << ID << "] "

<< name << ' '

<< adress << ' '

<< telephoneNumber << ' '

<< fax << " Contact: ";

custom.PShow();

}

// Метод производит поиск по фамилии в поле

// экземпляра класса custom

void PCheck(string compareElement) {

if (compareElement == custom.PSecondName())

PShow();

}

};

#endif // ORGANIZATION\_H

**friend.h**

#ifndef FRIEND\_H

#define FRIEND\_H

#include "persona.h"

class Friend : public TelephoneDirectory {

protected:

int ID;

Persona custom;

string birthdayDate;

public:

Friend() :

ID(999999),

birthdayDate("UNKNOWN") {

Persona();

}

Friend(int A0, int A1, string A2, string A3, string A4, string A5) :

ID(A0),

birthdayDate(A5) {

custom = Persona(A1, A2, A3, A4);

}

Friend(int A0, Persona A1, string A2) :

ID(A0),

custom(A1),

birthdayDate(A2) {}

~Friend() {}

void PShow() {

cout << "[" << ID << "] "

<< birthdayDate

<< " Friend: ";

custom.PShow();

}

void PCheck(string compareElement) {

if (compareElement == custom.PSecondName())

PShow();

}

};

#endif // FRIEND\_H

**Наследование**

~Friend() {}

void PShow() {

cout << "[" << ID << "] "

<< birthdayDate

<< " Friend: ";

custom.PShow();

}

void PCheck(string compareElement) {

if (compareElement == custom.PSecondName())

PShow();

}

};

#endif // FRIEND\_H

**Пример(ы)**

|  |
| --- |
| input.txt |
| Иванов Генадий +71111111111  Смирнов Валерий +72222222222  Кузнецов Владимир +73333333333  Gagov Tyty +74444444444  >Organization  Унистрим Россия 87198 100 Alliet Jarom +7352352626  КачергаINC США 41904 200 Энди Моно +7253834863  Вершески Жажба 24982 999 Пити Тронт +7346323456  >Friend  Морозов Чарли +32958235 01.23.2001  Afafa Olyt +245138978 04.04.1990  Поповски Поп +918375421 01.01.1970 |
| stdout |
| [100000] Генадий Иванов +71111111111  [100001] Валерий Смирнов +72222222222  [100002] Владимир Кузнецов +73333333333  [100003] Tyty Gagov +74444444444  [1004] Унистрим Россия 87198 100 Contact: [100004] Jarom Alliet +7352352626 |

**Наследование**

|  |
| --- |
| [1005] КачергаINC США 41904 200 Contact: [100005] Моно Энди +7253834863  [1006] Вершески Жажба 24982 999 Contact: [100006] Тронт Пити +7346323456  [1007] 01.23.2001 Friend: [100007] Чарли Морозов +32958235  [1008] 04.04.1990 Friend: [100008] Olyt Afafa +245138978  [1009] 01.01.1970 Friend: [100009] Поп Поповски +918375421  Введите фамилию для поиска (exit - для выхода): Alliet  [1004] Унистрим Россия 87198 100 Contact: [100004] Jarom Alliet +7352352626  Введите фамилию для поиска (exit - для выхода): exit |

**Вектора**

**Упражнение I, задача 2**

Заменить все четные элементы на x.

**Код программы**

**Пример(ы)**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

// Предикат для функции replace\_if, который

// проверяет аргумент на четность

bool f(int i) {

return i & 1;

}

// Предикат для функции for\_each

void fcin(int &i) {

cin >> i;

}

int main() {

int n, tochange;

cout << ">number of elements: ";

cin >> n;

cout << ">enter x: ";

cin >> tochange;

cout << ">elements: ";

vector<int> a(n);

// Считывание функцией for\_each

for\_each(a.begin(), a.end(), fcin);

// Функция заменяет все нечетные числа на введеное пользователем

replace\_if(a.begin(), a.end(), f, tochange);

cout << ">result: ";

// Вывод элементов вектора через итераторы

for (vector<int>::iterator i = a.begin(); i != a.end(); i++) {

cout << \*i << ' ';

}

return 0;

}

|  |  |
| --- | --- |
| stdin | stdout |
| >number of elements: 5  >enter x: 0  >elements: 54 23 -24 0 3 | >result: 0 23 0 0 3 |

**Вектора**

**Упражнение II, задача 11**

Вставить новый элемент после всех элементов, которые заканчиваются на заданную цифру.

**Код программы**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int x;

// Предикат функции find\_if проверяет аргумент

// на равенство первой цифры числа и х

bool f(int i) {

return abs(i % 10) == x;

}

void fcin(int &i) {

cin >> i;

}

void fcout(int i) {

cout << i << ' ';

}

int main() {

int n, c;

cout << ">number of elements: ";

cin >> n;

cout << ">enter x: ";

cin >> x;

cout << ">enter inserting element: ";

cin >> c;

cout << ">elements: ";

vector<int> a(n);

for\_each(a.begin(), a.end(), fcin);

// Итерационный цикл вставляет элемент c после всех

// элементов, последняя цифра которых равна x

for (vector<int>::iterator it = a.begin(); find\_if(it, a.end(), f) != a.end();) {

// Пока в векторе от итератора it до конца вектора

// есть элементы, удовлетворяющие условию из предиката f

// будем вставлять значение c после таких элементов

it = a.insert(find\_if(it, a.end(), f) +1, c) + 1;

}

cout << ">result: ";

for\_each(a.begin(), a.end(), fcout);

}

**Вектора**

**Пример(ы)**

|  |  |
| --- | --- |
| stdin | stdout |
| >number of elements: 10  >enter x: 4  >enter inserting element: 100  >elements: 424 8932 4 0 100 20023 24 44441 40 1 | >result: 424 100 8932 4 100 0 100 20023 24 100 44441 40 1 |