main.cpp

#include <iostream>

#include <Windows.h>

#include <limits>

#include "Handler.h"

using namespace std;

int main(){

system("mode con cols=110 lines=35");

system("color E0");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

startin();

}

Question.h

#pragma once

#include <string>

#include "List.h"

#include "TextObject.h"

using namespace std;

class Question: public TextObject

{

public:

Question(string question, List<string>& answer, List<int>& weight) {

\_question = question;

\_answer.copy(answer);

\_weight.copy(weight);

}

Question() { }

void toString() override;

void parseStringToQuestion(string question, string answer);

string answersToString();

string getQuestion() { return \_question; }

int getSize() { return \_answer.GetSize(); }

int getWeightAt(int index) {

return \_weight[index];

}

void edit();

static int runTest(List<Question>& qns);

static int maxWeight(List<Question>& qns);

void copy(Question& qns) {

\_weight.copy(qns.\_weight);

\_answer.copy(qns.\_answer);

\_question = qns.\_question;

}

private:

List<int> \_weight;

List<string> \_answer;

string \_question;

};

Question.cpp

#include "Question.h"

#include "InputException.h"

void Question::toString() {

cout << "Вопрос: " << \_question << endl;

for (int i = 0; i < \_answer.GetSize(); i++) {

cout << to\_string(i+1) << ": " << \_answer[i] << endl;

}

}

void Question::parseStringToQuestion(string question, string answer) {

List<int> weight;

List<string> answers;

string integer;

char\* piece;

char\* cstr = new char[answer.length() + 1];

strcpy(cstr, answer.c\_str());

piece = strtok(cstr, "#");

while (piece != NULL) {

integer = piece;

string in, wo;

bool flag = false;

for (int i = 0; i < integer.size(); i++) {

if (!flag) {

if (integer[i] == ' ') {

flag = true;

continue;

}

}

if (!flag)

in += integer[i];

else

wo += integer[i];

}

weight.push\_back(stoi(in));

answers.push\_back(wo);

piece = strtok(NULL, "#");

}

\_question = question;

\_weight.copy(weight);

\_answer.copy(answers);

}

string Question::answersToString() {

string str = "";

for (int i = 0; i < \_answer.GetSize(); i++)

str += "#" + to\_string(\_weight[i]) + " " + \_answer[i];

return str;

}

void Question::edit() {

List<string> answers;

List<int> weight;

string question, temp;

string w;

cout << "Введите вопрос: \n";

while (true) {

try {

getline(cin, question);

if (question == "")

throw(InputException(2));

else

break;

}

catch (InputException e) {

e.error();

cout << "Повторите ввод:\n";

}

}

string flag = "1";

bool tempFlag = false;

while (stoi(flag)) {

cout << "Введите вариант ответа: ";

while (true) {

try {

getline(cin, temp);

if (temp == "")

throw(InputException(2));

else

break;

}

catch (InputException e) {

e.error();

cout << "Повторите ввод:\n";

}

}

cout << "Введите вес ответа: \n";

while (true) {

try {

getline(cin, w);

if (w == "")

throw(InputException(2));

else

break;

}

catch (InputException e) {

e.error();

cout << "Повторите ввод:\n";

}

}

weight.push\_back(stoi(w));

answers.push\_back(temp);

cout << "Ещё вводим? 1 - да, 0 - нет: \n";

getline(cin, flag);

}

\_question = question;

\_answer.clear();

\_answer.copy(answers);

\_weight.clear();

\_weight.copy(weight);

}

int Question::runTest(List<Question>& qns){

int counter = 0, choise;

for (int i = 0; i < qns.GetSize(); i++) {

system("cls");

qns[i].toString();

cin.ignore();

while (true) {

try {

cin >> choise;

if (choise > 0 && choise <= qns[i].getSize()) {

choise--;

counter += qns[i].getWeightAt(choise);

break;

}

else

throw(InputException(3));

}

catch (InputException e) {

e.error();

cout << "Повторите ввод:\n";

}

}

}

return counter;

}

int Question::maxWeight(List<Question>& qns){

int counter = 0;

int max = -1000;

for (int i = 0; i < qns.GetSize(); i++) {

for (int j = 0; j < qns[i].getSize(); j++) {

if (max < qns[i].getWeightAt(j)) {

max = qns[i].getWeightAt(j);

}

}

counter += max;

max = -1000;

}

return counter;

}

TextObject.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

class TextObject

{

string text;

virtual void toString();

public:

void print(string str);

void printHello();

TextObject(string str) {

text = str;

}

TextObject() {}

void println(string str);

void cls() { system("cls"); }

};

TextObject.cpp

#include "TextObject.h"

void TextObject::toString() {

cout << text << endl;

}

void TextObject::print(string str) {

cout << str;

}

void TextObject::println(string str){

cout << str << endl;

}

Result.h

#pragma once

#include "Question.h"

class Result: public Question

{

public:

Result(List<string> res) {

result.copy(res);

}

Result() {}

int getSize() { return result.GetSize(); }

void parse(string str) {

result.push\_back(str);

}

string get(int index) { return result[index]; }

private:

List<string> result;

};

List.h

#include <iostream>

using namespace std;

template<typename T>

class List{

public:

List();

~List();

void pop\_front(); //удаление первого элемента в списке

void push\_back(T data); //добавление элемента в конец списка

void clear(); // очистить список

int GetSize() { return Size; } // получить количество элементов в списке

T& operator[](const int index);

void copy(List<T>& source);

void push\_front(T data); //добавление элемента в начало списка

void insert(T data, int index); //добавление элемента в список по указанному индексу

void removeAt(int index); //удаление элемента в списке по указанному индексу

void pop\_back(); //удаление последнего элемента в списке

private:

template<typename T>

class Node{

public:

Node\* pNext;

T data;

Node(T data = T(), Node\* pNext = nullptr){

this->data = data;

this->pNext = pNext;

}

};

int Size;

Node<T>\* head;

};

template<typename T>

List<T>::List()

{

Size = 0;

head = nullptr;

}

template<typename T>

List<T>::~List(){

clear();

}

template<typename T>

void List<T>::output() {

if (head == nullptr) {

cout << "Стек пуст." << endl;

return;

}

else {

for (Node<T>\* node = head; node != nullptr; node = node->pNext) {

cout << node->data;

}

}

}

template<typename T>

void List<T>::copy(List<T>& source)

{

this->clear();

for (int i = 0; i < source.GetSize(); i++)

this->push\_back(source[i]);

}

template<typename T>

void List<T>::pop\_front()

{

Node<T>\* temp = head;

head = head->pNext;

delete temp;

Size--;

}

template<typename T>

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

{

if (head == nullptr){

head = new Node<T>(data);

}

else{

Node<T>\* current = this->head;

while (current->pNext != nullptr){

current = current->pNext;

}

current->pNext = new Node<T>(data);

}

Size++;

}

template<typename T>

void List<T>::clear()

{

while (Size){

pop\_front();

}

}

template<typename T>

T& List<T>::operator[](const int index)

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr){

if (counter == index){

return current->data;

}

current = current->pNext;

counter++;

}

}

template<typename T>

void List<T>::push\_front(T data)

{

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

Size++;

}

template<typename T>

void List<T>::insert(T data, int index)

{

if (index == 0) {

push\_front(data);

}

else {

Node<T>\* previous = this->head;

for (int i = 0; i < index - 1; i++){

previous = previous->pNext;

}

Node<T>\* newNode = new Node<T>(data, previous->pNext);

previous->pNext = newNode;

Size++;

}

}

template<typename T>

void List<T>::removeAt(int index){

if (index == 0){

pop\_front();

}

else{

Node<T>\* previous = this->head;

for (int i = 0; i < index - 1; i++){

previous = previous->pNext;

}

Node<T>\* toDelete = previous->pNext;

previous->pNext = toDelete->pNext;

delete toDelete;

Size--;

}

}

template<typename T>

void List<T>::pop\_back()

{

removeAt(Size - 1);

}

Exception.h

#pragma once

class Exception

{

protected:

int error;

public:

Exception() { error = 0;}

Exception(int n) { error = n; }

};

FileException.h

#pragma once

#include "exception.h"

class FileException : public Exception

{

public:

FileException() :Exception() {};

FileException(int n) :Exception(n) {};

void error();

};

void FileException::error()

{

switch (Exception::error) {

case 1:

cout << "Файл поврежден или изменён" << endl;

cout << "Не удалось открыть файл" << endl;

break;

}

}

ListException.h

#pragma once

#include "exception.h"

#include <iostream>

using namespace std;

class ListException : public Exception

{

public:

ListException() : Exception() { };

ListException(int n) : Exception(n) { };

void error();

};

void ListException::error()

{

switch (Exception::error) {

case 1:

cout << "Код ошибки #2.1" << endl;

cout << "Список пуст";

break;

case 2:

cout << "Код ошибки #2.2" << endl;

cout << "Нарушение границ cписка";

break;

default:

cout << "Код ошибки #2.Err" << endl;

cout << "Ошибка работы со списком" << endl;

break;

}

}

InputException.h

#pragma once

#include "exception.h"

#include <iostream>

using namespace std;

class InputException: public Exception

{

public:

InputException() :Exception() { };

InputException(int n) :Exception(n) { };

void error();

};

void InputException::error()

{

switch (Exception::error) {

case 1:

cout << "Код ошибки #1.2" << endl;

cout << "Введён недопустимый символ" << endl;

cout << "Разрешены только цифры";

break;

case 2:

cout << "Код ошикбки #1.3" << endl;

cout << "Пустое место запрещено для ввода" << endl;

break;

case 3:

cout << "Код ошибки #1.4" << endl;

cout << "Невозможно зафиксировать ответ" << endl;

cout << "Ответ превышает максимальное кол-во ответов";

break;

case 0:

cout << "Код ошибки #1.1" << endl;

cout << "Данное имя User-a не зарегестрировано администратором" << endl;

cout << "Проверьте существующую базу и повторите ввод" << endl;

break;

}

}

File.h

#pragma once

#include <fstream>

#include "Question.h"

#include "FileException.h"

using namespace std;

template <class X>

class File

{

public:

File(string filename) { this->filename = filename; }

void getData(List<string>& data) {

ifstream dataHolder(filename);

try {

if (!dataHolder.is\_open())

throw (FileException(1));

}

catch (FileException e) {

e.error();

exit(1);

}

string temp;

while (dataHolder) {

getline(dataHolder, temp);

if (dataHolder)

data.push\_back(temp);

}

dataHolder.close();

}

void setData(List<string>& data) {

ofstream dataHolder(filename);

try {

if (!dataHolder.is\_open())

throw (FileException(1));

}

catch (FileException e) {

e.error();

exit(1);

}

for (int i = 0; i < data.GetSize(); i++)

dataHolder << data[i] << "\n";

dataHolder.close();

}

private:

string filename = "";

};

template<>

class File<string>

{

private:

string filename = "";

public:

File(string filename) { this->filename = filename; }

void getData(List<string>& str)

{

str.clear();

ifstream file(filename);

string temp;

while (file) {

if (file) {

getline(file, temp);

str.push\_back(temp);

}

}

file.close();

}

void setData(List<string>& str)

{

ofstream file(filename);

for (int i = 0; i < str.GetSize(); i++) {

file << str[i] << "\n";

}

file.close();

}

void addData(string str)

{

ofstream file(filename, ios::app);

file << str<< "\n";

file.close();

}

void changeFilename(string str) {

filename = str;

}

};

template<>

class File<Question>

{

private:

string filename = "";

public:

File(string filename) { this->filename = filename; }

void changeFilename(string str)

{

filename = str;

}

void getData(List<Question>& qns)

{

string ans, que;

qns.clear();

ifstream questions((filename + "Question.bin"));

ifstream answer((filename + "Answer.bin"), ios::binary);

try {

if (!answer.is\_open())

throw (FileException(1));

}

catch (FileException e)

{

e.error();

exit(1);

}

try {

if (!questions.is\_open())

throw (FileException(1));

}

catch (FileException e) {

e.error();

exit(1);

}

while (answer) {

getline(answer, ans);

if (ans != "") {

getline(questions, que);

qns.push\_back(Question());

qns[qns.GetSize() - 1].parseStringToQuestion(que, ans);

}

}

questions.close();

answer.close();

}

void setData(List<Question>& qns)

{

ofstream questions((filename + "Question.bin"));

ofstream answer((filename + "Answer.bin"), ios::binary);

try {

if (!answer.is\_open())

throw (FileException(1));

}

catch (FileException e) {

e.error();

exit(1);

}

try {

if (!questions.is\_open())

throw (FileException(1));

}

catch (FileException e) {

e.error();

exit(1);

}

for (int i = 0; i < qns.GetSize(); i++) {

questions << qns[i].getQuestion() << "\n";

answer << qns[i].answersToString() << "\n";

}

questions.close();

answer.close();

}

};

Handler.h

#pragma once

#include "Question.h"

#include "InputException.h"

#include <fstream>

#include "File.h"

#include "Result.h"

#include <string>

void startin();

void Run(bool isAdmin, string filename);

bool getUser();

void searchAndPrint();

Handler.cpp

#include <iostream>

#include "Handler.h"

using namespace std;

string superLogin;

void Run(bool isAdmin, string filename){

if (isAdmin) {

TextObject out;

char clearStats;

File<Question> file(filename);

List<Question> qns;

List<string> stats;

List<string> resForInput;

file.getData(qns);

File<string> result(filename + "Result.bin");

File<string> stat("stats.txt");

List<string> res;

char choise;

double temp;

string flag = "1";

string tempStr;

string str, testName;

bool hasPrev = false;

bool isAdm = false;

ofstream f;

ofstream af;

ofstream ff;

ofstream fff;

List<Question> prev; Question q;

while (true) {

system("cls");

out.printHeadline();

cout << "Добро пожаловать в редактирование." << endl << endl;

cout << "Выберите действие:" << endl;

cout << "[1] - добавить пользователя" << endl;

cout << "[2] - создать новый тест - файлы тестов" << endl << endl;

cout << "Выбранный тест - " << filename << ".test" << endl;

cout << "[3] - установить новые результаты." << endl;

cout << "[4] - просмотр теста." << endl;

cout << "[5] - редактировать вопрос." << endl;

cout << "[6] - удалить вопрос." << endl;

cout << "[7] - добавить вопрос." << endl;

cout << "[8] - отмена последнего действия." << endl;

cout << "[9] - просмотреть статистику всех пользователей" << endl << endl;

cout << "[press any key] - сохранить изменения и вернуться в меню." << endl;

cout << endl << ">> ";

cin >> choise;

switch (choise) {

case '4':

if (qns.GetSize() == 0) {

cout << "\t\t\t\tВопросы:" << endl;

cout << "В данном тесте пока нету вопросов." << endl << endl;

if (res.GetSize() == 0) {

cout << "\t\t\t\tВозможные результаты:" << endl;

cout << "В данном тесте пока нету результатов." << endl << endl;

}

}

else {

system("CLS");

cout << "\t\t\t\tВопросы:" << endl;

for (int i = 0; i < qns.GetSize(); i++) {

cout << "Вопрос №" << i + 1 << ": "; qns[i].toString();

}

cout << "\t\t\t\tВозможные результаты:" << endl;

result.getData(res);

if (res.GetSize() == 0) {

cout << "В данном тесте пока нету результатов." << endl << endl;

}

for (int i = 0; i < res.GetSize() - 1; i++) {

cout << i+1 << ". " << res[i] << endl;

}

}

system("PAUSE");

break;

case '5':

if (qns.GetSize() == 0) {

cout << "В данном тесте пока нету вопросов.Редактирование невозможно";

system("PAUSE");

break;

}

for (int i = 0; i < qns.GetSize(); i++) {

cout << "Вопрос №" << i + 1 << ": "; qns[i].toString();

}

while (true) {

cout << "Какой номер вопроса редактировать?: ";

try {

int choi;

cin >> choi;

if (cin.fail() || cin.get() != '\n') {

throw(InputException(1));

}

if (choi < 1 || choi >= (qns.GetSize() + 1))

throw (InputException(3));

else {

cout << "Введите вопрос: \n";

choi--;

prev.clear();

for (int i = 0; i < qns.GetSize(); i++) {

prev.push\_back(Question());

prev[prev.GetSize() - 1].copy(qns[i]);

}

hasPrev = true;

cin.ignore();

qns[choi].edit();

system("pause");

break;

}

}

catch (InputException e) {

cin.clear();

rewind(stdin);

e.error();

}

}

break;

case '6':

if (qns.GetSize() == 0) {

cout << "В данном тесте пока нету вопросов. Удаление невозможно.";

system("PAUSE");

break;

}

for (int i = 0; i < qns.GetSize(); i++) {

cout << "Вопрос №" << i + 1 << ": "; qns[i].toString();

}

cout << "Какой номер вопроса вы хотите удалить? " << endl;

while (true)

try {

int choi;

cin >> choi;

if (choi < 1 || choi >= (qns.GetSize() + 1))

throw (InputException(3));

else {

choi--;

prev.clear();

for (int i = 0; i < qns.GetSize(); i++) {

prev.push\_back(Question());

prev[prev.GetSize() - 1].copy(qns[i]);

}

hasPrev = true;

qns.removeAt(choi);

cout << "Вопрос удалён." << endl;

system("pause");

break;

}

}

catch (InputException e) {

e.error();

}

break;

case '7':

cin.ignore();

q.edit();

prev.clear();

for (int i = 0; i < qns.GetSize(); i++) {

prev.push\_back(Question());

prev[prev.GetSize() - 1].copy(qns[i]);

}

hasPrev = true;

qns.push\_back(Question());

qns[qns.GetSize() - 1].copy(q);

system("pause");

break;

case '8':

if (hasPrev) {

qns.clear();

for (int i = 0; i < prev.GetSize(); i++) {

qns.push\_back(Question());

qns[qns.GetSize() - 1].copy(prev[i]);

}

hasPrev = false;

}

break;

case 's':

file.setData(qns);

break;

case '1':

f.open("users.bin", ios::binary | ios::app);

if (f.is\_open()) {

cout << "Добавление пользователя" << endl;

cout << "Ввод нового логина пользоавтеля" << endl;

cout << endl << ">> ";

cin >> str;

cout << "Права пользователя." << endl;

cout << "[1] - Admin" << endl;

cout << "[0] - User " << endl;

cout << endl << ">> ";

cin >> isAdm;

string user = str + " " + to\_string(isAdm);

cout << user;

f << "\n" << user;

f.close();

cout << "Пользователь успешно добавлен" << endl;

system("pause");

}

else {

throw FileException(1);

abort();

}

break;

case '2':

cout << "Создание нового теста" << endl;

cout << "Ввод названия нового теста " << endl;

cout << endl << ">> ";

cin >> testName;

if (testName == "exit") {

break;

}

af.open(testName + "Answer.bin");

ff.open(testName + "Question.bin");

fff.open(testName + "Result.bin");

af.close();

ff.close();

fff.close();

af.open("tests.txt", ios::app);

af << "\n" << testName;

af.close();

file.changeFilename(testName);

result.changeFilename(testName + "Result.bin");

file.getData(qns);

cout << endl << "Новый файл с тестом - " << testName << " успешно создан.";

cout << "Связь установлена" << endl;

system("pause");

break;

case '3':

cout << "Внимание! Вы устанавливаете возможные результаты." << endl;

cin.ignore(37260, '\n');

while (stoi(flag)) {

cout << endl << ">> ";

getline(cin, tempStr);

resForInput.push\_back(tempStr);

cout << endl << "Элемент успешно добавлен" << endl;

cout << "[1] - добавить" << endl;

cout << "[0] - выход " << endl;

cout << endl << ">> ";

getline(cin, flag);

system("CLS");

}

result.setData(resForInput);

break;

case '9':

system("cls");

cout << "Статистика всех прохождений теста" << endl;

stat.getData(stats);

if (stats.GetSize() == 0)

cout << "Данных пока нет!";

else

for (int i = 0; i < stats.GetSize(); i++)

cout << stats[i] << endl;

cout << endl << endl;

cout << "[1] - очистить статистику" << endl;

rewind(stdin);

clearStats = getchar();

if (clearStats == '1') {

fstream clear\_file("stats.txt", ios::out);

clear\_file.close();

}

system("pause");

break;

default:

file.setData(qns);

system("cls");

startin();

break;

}

}

}

else {

TextObject out;

double temp;

File<string> stat("stats.txt");

File<Question> q(filename);

List<Question> qns;

q.getData(qns);

Result ress;

string tmp = " ";

List<string> str;

List<string> stats;

File<Result> res(filename + "Result.bin");

while (true) {

system("cls");

out.printHeadline();

cout << "Выбранный тест - " << filename << endl << endl;

cout << "Выберите действие:\n[1] - пройти тест\

cout << " [2] - просмотреть свой результат\n[0] - выйти обратно в меню";

int choise;

cout << endl << ">> ";

cin >> choise;

switch (choise) {

case 1:

if (qns.GetSize() == 0) {

cout << "В тесте пока нету вопросов. Обратитесь к Администратору";

system("PAUSE");

break;

}

temp = Question::runTest(qns);

cout << "Результат: ";

res.getData(str);

for (int i = 0; i < str.GetSize(); i++)

ress.parse(str[i]);

if (ress.getSize() != 0) {

tmp = ress.get(((int)(temp / Question::maxWeight(qns) \* (ress.getSize() - 1))) % ress.getSize());

cout << tmp << endl;

stat.addData(superLogin + " " + tmp);

}

else {

cout << "\nРезультаты пока не занесены в базу.\n";

}

system("pause");

break;

case 2:

if (tmp == " ") {

cout << "Результата пока нет. Необходимо пройти тест." << endl;

system("pause");

break;

}

cout << (superLogin + ' ' + tmp) << endl;

/\*if (ress.getSize() != 0) {

}

case 0:

system("CLS");

startin();

break;

default:

cout << endl << "Такого действия не существует. Повторите ввод" << endl;

system("PAUSE");

break;

}

}

}

}

bool getUser(){

ifstream f("users.bin", ios::binary);

string login; string str, temp = ""; string val = ""; bool flag = true;

cout << "\tВведите ваше имя-Login: ";

cout << endl << "\t>> ";

cin >> login;

while (f) {

getline(f, str);

for (int i = 0; i < str.size(); i++) {

if (str[i] != ' ') {

if (flag)

temp += str[i];

else {

val += str[i];

}

}

else flag = false;

}

if (login == temp) {

superLogin = login;

return (bool)stoi(val);

}

flag = true;

temp = "";

val = "";

}

throw (InputException(0));

}

void startin() {

List<Question> q;

List<string> tests;

TextObject out;

File<string> testData("tests.txt");

testData.getData(tests);

string filename;

out.printHeadline();

out.printHello();

out.println("\t\t\tСписок доступных тестов: ");

for (int i = 0; i < tests.GetSize() - 1; i++) {

out.println("\t\t" + to\_string(i + 1) + ": " + tests[i]);

}

int choise;

out.print("\tДля продолжения необходимо выбрать тест:");

cout << endl << "\t>> ";

while (true) {

try {

cin.clear();

rewind(stdin);

cin >> choise;

if (choise == -1) {

exit(0);

}

if (cin.fail() || cin.get() != '\n') {

throw(InputException(1));

}

if ((choise <= (tests.GetSize() - 1)) && choise > 0) {

filename = tests[--choise];

break;

}

else {

throw(InputException(3));

}

}

catch (InputException ex) {

cin.clear();

rewind(stdin);

ex.error();

}

}

File<Question> qs(filename);

qs.getData(q);

while (true) {

try {

Run(getUser(), filename);

break;

}

catch (InputException e) {

e.error();

}

}

}