



З майго кода тут толькі гэтая функцыія

#include <iostream>

#include <Windows.h>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

// Структура для прадстаўлення адраса

struct Address

{

char name[NAME\_SIZE];

char city[CITY\_SIZE];

Address\* next;

Address\* prev;

};

// Функцыя для вываду меню

int menu(void)

{

char s[80];

int c;

cout << endl;

cout << "1. Увод імя" << endl;

cout << "2. Выдаленне імя" << endl;

cout << "3. Выснова на экран" << endl;

cout << "4. Пошук" << endl;

cout << "5. Выдаленне кожнага M-га элемента" << endl;

cout << "0. Выхад" << endl;

cout << endl;

do

{

cout << "Ваш выбар: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

} while (c < 0 || c > 5);

return c;

}

// Функцыя для дадання элемента ў канец спісу

void insert(Address\* e, Address\*\* phead, Address\*\* plast)

{

Address\* p = \*plast;

if (\*plast == NULL)

{

// Калі спіс пусты, усталёўваны паказальнікі на новы элемент

e->next = NULL;

e->prev = NULL;

\*plast = e;

\*phead = e;

return;

}

else

{

// Дадаем элемент у канец спісу

p->next = e;

e->next = NULL;

e->prev = p;

\*plast = e;

}

}

// Функцыя для стварэння элемента і ўводу яго значэнняў з клавіятуры

Address\* setElement()

{

Address\* temp = new Address();

if (!temp)

{

cerr << "Памылка вылучэння памяці памяці";

return NULL;

}

cout << "Увядзіце імя: ";

cin.getline(temp->name, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

cout << "Увядзіце горад: ";

cin.getline(temp->city, CITY\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

temp->next = NULL;

temp->prev = NULL;

return temp;

}

// Функцыя для вываду спісу на экран

void outputList(Address\*\* phead, Address\*\* plast)

{

Address\* t = \*phead;

while (t)

{

// Выводзім імя і горад на экран

cout << t->name << ' ' << t->city << endl;

t = t->next;

}

cout << "" << endl;

}

// Функцыя для пошуку імя ў спісе

void find(char name[NAME\_SIZE], Address\*\* phead)

{

Address\* t = \*phead;

while (t)

{

// Шукаем у спісе уведзенае імя

if (!strcmp(name, t->name))

break;

t = t->next;

}

if (!t)

cerr << "Імя не знойдзена" << endl;

else

cout << t->name << ' ' << t->city << endl;

}

// Функцыя для выдалення імя са спісу

void delet(char name[NAME\_SIZE], Address\*\* phead, Address\*\* plast)

{

struct Address\* t = \*phead;

while (t)

{

// Шукаем у спісе уведзенае імя

if (!strcmp(name, t->name))

break;

t = t->next;

}

if (!t)

cerr << "Імя не знойдзена" << endl;

else

{

if (\*phead == t)

{

// Калі элемент, які выдаляецца, - першы, то абнаўляем паказальнік на пачатак спісу

\*phead = t->next;

if (\*phead)

(\*phead)->prev = NULL;

else

\*plast = NULL;

}

else

{

// Абнаўляем паказальнікі пры выдаленні элемента з сярэдзіны ці канца спісу

t->prev->next = t->next;

if (t != \*plast)

t->next->prev = t->prev;

else

\*plast = t->prev;

}

// Вызваляем памяць і паведамляем аб выдаленні

delete t;

cout << "Элемент выдалены" << endl;

}

}

// Функцыя для выдалення кожнага M-га элемента з спісу

void deleteEveryM(int m, Address\*\* phead, Address\*\* plast)

{

if (m <= 0)

{

cerr << "Invalid value of M" << endl;

return;

}

Address\* current = \*phead;

Address\* prev = nullptr;

int count = 1;

while (current)

{

if (count % m == 0)

{

// Выдаляем M-ы элемент

if (current == \*phead)

{

\*phead = current->next;

if (\*phead)

(\*phead)->prev = nullptr;

}

else

{

// Абнаўляем паказальнікі пры выдаленні элемента з сярэдзіны ці канца спісу

prev->next = current->next;

if (current == \*plast)

\*plast = prev;

else

current->next->prev = prev;

}

// Вызваляем памяць і паведамляем аб выдаленні

Address\* temp = current;

current = current->next;

delete temp;

cout << "Element deleted at position " << count << endl;

}

else

{

// Пераходзім да наступнага элемента

prev = current;

current = current->next;

}

count++;

}

}

// Асноўная функцыя праграмы

int main(void)

{

Address\* head = NULL;

Address\* last = NULL;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

while (true)

{

switch (menu())

{

case 1:

insert(setElement(), &head, &last);

break;

case 2:

{

char dname[NAME\_SIZE];

cout << "Увядзіце імя: ";

cin.getline(dname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

delet(dname, &head, &last);

}

break;

case 3:

outputList(&head, &last);

break;

case 4:

{

char fname[NAME\_SIZE];

cout << "Увядзіце імя: ";

cin.getline(fname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

find(fname, &head);

}

break;

case 5:

{

int m;

cout << "Увядзіце M: ";

cin >> m;

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

deleteEveryM(m, &head, &last);

}

break;

default:

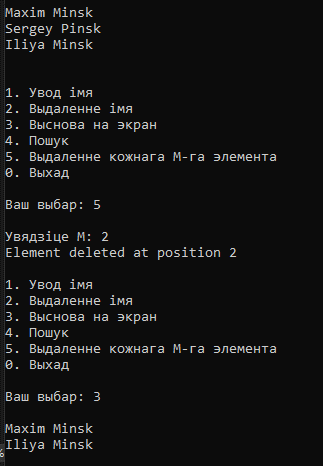
exit(1);

}

}

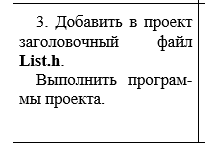
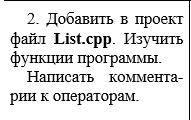
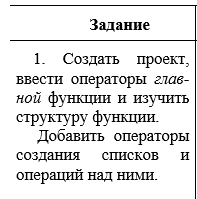
return 0;

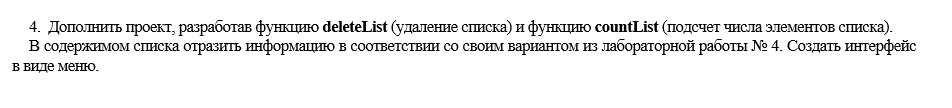
}



Дап. Заданні







Main.cpp

#include "List.h"

#include <iostream>

#include <Windows.h>

using namespace std;

void print(void\* b) //Функцыя выкарыстоўваецца пры вывадзе

{

Person\* a = (Person\*)b;

cout << a->name << " " << a->age << endl;

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

Object L1 = Create();

int choice;

do

{

cout << "\nMenu:\n";

cout << "1. Уставіць элемент\n";

cout << "2. Вывесці спіс\n";

cout << "3. Знайсці элемент\n";

cout << "4. Выдаліць элемент\n";

cout << "5. Выдаліць увесь спіс\n";

cout << "6. Палічыць элементы ў спісе\n";

cout << "0. Выйсці\n";

cout << "Увядзіце ваш выбар: ";

cin >> choice;

switch (choice)

{

case 1:

{

Person newPerson;

cout << "Увядзіце імя: ";

cin >> newPerson.name;

cout << "Увядзіце ўзрост: ";

cin >> newPerson.age;

L1.Insert(&newPerson);

cout << "Элемент устаўлены ў спіс.\n";

}

break;

case 2:

cout << "Бягучы спіс:\n";

L1.PrintList(print);

break;

case 3:

// Паклькі дадзеныя функцыі Search() Delete() прымаюць паказальнікі на аб'екты, якія трэба выдаліць / знайсці, то нельга проста перадаць значэнне і знайсці па яму, а трэба перадаць адрас пэўнага аб'екта для выдалення (спасылку на яго), з-за чаго прыйшлося б перарабляць палову кода, можна не трэба

break;

case 4:

break;

case 5:

if (L1.DeleteList())

cout << "Спіс выдалены паспяхова.\n";

break;

case 6:

cout << "Колькасць элементаў у спісе: " << L1.CountList() << endl;

break;

case 0:

cout << "Выхад з праграмы.\n";

break;

default:

cout << "Няправільны выбар\n";

break;

}

} while (choice != 0);

return 0;

}

List.cpp

#include "List.h"

bool Object::Insert(void\* data)

{

bool rc = false;

if (Head == NULL)

{

// Вылучаем памяць і ствараем копію дадзеных для першага элемента

Head = new Element(NULL, new Person(\*((Person\*)data)), Head);

rc = true;

}

else

{

// Вылучаем памяць і ствараем копію дадзеных для наступных элементаў

Head = (Head->Prev = new Element(NULL, new Person(\*(static\_cast<Person\*>(data))), Head));

rc = true;

}

return rc;

}

//------------------------------------------------ -----------

Element\* Object::Search(void\* data) // Знайсці зададзены элемент

{

Element\* rc = Head;

while ((rc != NULL) && (rc->Data != data))

rc = rc->Next;

return rc;

}

//------------------------------------------------ -----------

void Object::PrintList(void(\*f)(void\*)) // Выснова

{

Element\* e = Head;

while (e != NULL)

{

f(e->Data); // Выклікаем функцыю для вываду дадзеных

e = e->GetNext();

};

}

//------------------------------------------------- ----------

void Object::PrintList(void(\*f)(void\*), Element\* e)

{

f(e->Data); // Выклікаем функцыю для вываду дадзеных

}

//------------------------------------------------ -----------

bool Object::Delete(Element\* e) // Выдаліць па спасылцы

{

bool rc = 0;

if (rc = (e != NULL))

{

if (e->Next != NULL)

e->Next->Prev = e->Prev;

if (e->Prev != NULL)

e->Prev->Next = e->Next;

else

Head = e->Next;

delete e; // Вызваляем памяць, якую займае элемент, які выдаляецца.

}

return rc;

}

//------------------------------------------------ -----------

bool Object::Delete(void\* data) // Выдаліць па значэнні

{

return Delete(Search(data));

}

//------------------------------------------------ -----------

Element\* Object::GetLast()

{

Element\* e = Head, \* rc = e;

while (e != NULL)

{

rc = e; // Абнаўляем паказальнік на апошні элемент

e = e->GetNext();

}

return rc;

}

//------------------------------------------------ -----------

Object Create()

{

return \*(new Object());

}

bool Object::DeleteList()

{

while (Head != nullptr)

{

Element\* temp = Head;

Head = Head->Next;

delete temp; // Вызваляем памяць, якую займае элемент, які выдаляецца.

}

return true;

}

int Object::CountList()

{

int count = 0;

Element\* current = Head;

while (current != nullptr)

{

count++;

current = current->Next;

}

return count;

}

List.h

#pragma once

#include <iostream>

struct Person

{

char name[20];

int age;

Person\* next;

};

struct Element // Элемент спісу

{

Element\* Prev; // паказальнік на папярэдні элемент

Element\* Next; // паказальнік на наступны элемент

void\* Data; // дадзеныя

Element(Element\* prev, void\* data, Element\* next)

{

Prev = prev;

Data = data;

Next = next;

}

Element\* GetNext() // атрымаць наступны

{

return Next;

};

Element\* GetPrev() // атрымаць папярэдні

{

return Prev;

};

};

//------------------------------------------------ -----------

struct Object // Блок кіравання спісам

{

Element\* Head; // паказальнік на пачатак спісу

Object()

{

Head = NULL;

};

Element\* GetFirst() // атрымаць першы элемент спісу

{

return Head;

};

Element\* GetLast(); // атрымаць апошні элемент спісу

Element\* Search(void\* data); // знайсці першы элемент па дадзеных

bool Insert(void\* data); // дадаць элемент у пачатак

bool InsertEnd(void\* data); // дадаць у канец

bool Delete(Element\* e); // выдаліць па адрасе элемента

bool Delete(void\* data); // выдаліць першы па дадзеных

bool DeleteList(); // ачысціць спіс

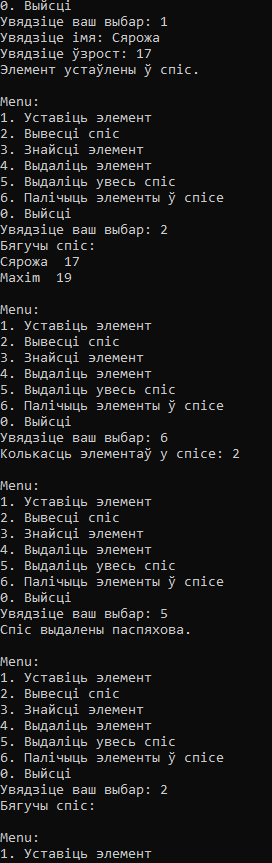
void PrintList(void(\*f)(void\*)); // Вывесці спіс

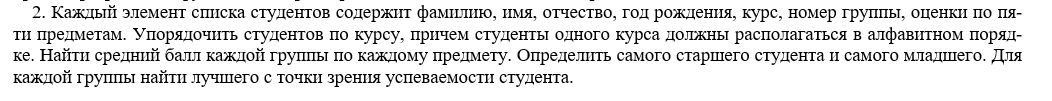
void PrintList(void(\*f)(void\*), Element\*); // Вывесці спіс

int CountList(); // Палічыць элементы ў спісе

};

Object Create();





#include <Windows.h>

#include <iostream>

#include <string>

using namespace std;

struct LinkedList

{

LinkedList\* prev; // Паказальнік на папярэдні элемент у спісе

LinkedList\* next; // Паказальнік на наступны элемент у спісе

std::string lastName; // Прозвішча студэнта

std::string firstName; // Імя студэнта

std::string secondName; // Імя па бацьку студэнта

int bdayYear; // Год нараджэння студэнта

int course; // Курс, на якім вучыцца студэнт

int group; // Нумар групы студэнта

int\* grades; // Паказальнік на масіў адзнак

// Канструктар класа

LinkedList()

{

prev = next = nullptr; // Ініцыялізацыя паказальнікаў на папярэдні і наступны элементы

grades = new int[5]; // Вылучэнне памяці пад масіў адзнак

}

// Дэструктар класа

~LinkedList()

{

delete[] grades; // Вызваленне памяці, выдзеленай пад масіў адзнак

}

};

// Функцыя сартавання звязанага спісу па зададзеным крытэрыі

void sortList(LinkedList\*& p);

// Функцыя дадання студэнта ў звязаны спіс

void add(LinkedList\*& p)

{

// Ствараем новы аб'ект студэнта

LinkedList\* student = new LinkedList;

// Запытваем у карыстальніка інфармацыю пра студэнта

cout << "Увядзіце прозвішча: "; getline(cin, student->lastName);

cout << "Увядзіце імя: "; getline(cin, student->firstName);

cout << "Увядзіце імя па бацьку: "; getline(cin, student->secondName);

cout << "Увядзіце год нараджэння: "; cin >> student->bdayYear;

cout << "Увядзіце курс: "; cin >> student->course;

cout << "Увядзіце групу: "; cin >> student->group;

cout << "Увядзіце 5 адзнак праз прабел: "; cin >> student->grades[0] >> student->grades[1] >> student->grades[2] >> student->grades[3] >> student->grades[4];

cout << "\n";

cin.ignore(); // Чысцім буфер уводу

// Калі спіс пусты, прысвойваем яму першага студэнта

if (p == nullptr)

{

p = student;

}

else

{

// Калі спіс не пусты, дадаем студэнта ў пачатак спісу

p->prev = student;

student->next = p;

student->prev = nullptr;

p = student;

}

}

// Функцыя вываду інфармацыі пра студэнта на экран

void printStudent(const LinkedList\* ptr)

{

// Выводзім інфармацыю пра студэнта

cout << "\n-------------------------------------------\n";

cout << "Прозвішча: " << ptr->lastName << '\n';

cout << "Імя: " << ptr->firstName << '\n';

cout << "Імя па бацьку: " << ptr->secondName << '\n';

cout << "Год нараджэння: " << ptr->bdayYear << '\n';

cout << "Курс: " << ptr->course << '\n';

cout << "Група: " << ptr->group << '\n';

cout << "5 адзнак: " << ptr->grades[0] << ' ' << ptr->grades[1] << ' ' << ptr->grades[2] << ' ' << ptr->grades[3] << ' ' << ptr->grades[4] << ' ' << '\n';

cout << "-------------------------------------------\n";

}

// Функцыя вываду ўсяго звязанага спісу на экран

void print(LinkedList\* p)

{

LinkedList\* ptr = p;

// Праходзім па спісе і выводзім інфармацыю пра кожнага студэнта

while (ptr)

{

printStudent(ptr);

ptr = ptr->next;

}

}

// Функцыя разліку сярэдняга бала па кожнай групе

void calculateAverage(LinkedList\* p)

{

// Праходзім па кожнай групе (мяркуецца, што групы нумаруюцца ад 1 да 4)

for (int i = 1; i <= 4; i++)

{

LinkedList\* ptr = p;

int count = 0;

int\* allGrades = new int[5];

// Ініцыялізуем масіў сум адзнак для кожнага студэнта ў групе

for (size\_t i = 0; i < 5; i++)

{

allGrades[i] = 0;

}

// Праходзім па звязаным спісе студэнтаў

while (ptr)

{

// Калі студэнт належыць бягучай групе, абнаўляем сумы адзнак і павялічваем лічыльнік

if (ptr->group == i)

{

for (int j = 0; j < 5; ++j)

{

allGrades[j] += ptr->grades[j];

}

count++;

}

ptr = ptr->next;

}

// Калі ў групе ёсць студэнты, выводзім сярэдні бал па кожным прадмеце

if (count > 0)

{

printf\_s("Група %d: ", i);

for (size\_t j = 0; j < 5; j++)

{

// Разлічваем сярэдні бал і выводзім яго з адной лічбай пасля коскі

double grade = (double)allGrades[j] / count;

printf\_s("%.1f ", grade);

}

cout << '\n';

}

// Вызваляем выдзеленую памяць пад масіў сум адзнак

delete[] allGrades;

}

}

// Функцыя разліку сярэдняга бала для аднаго студэнта

double calculateAverageGrade(const LinkedList\* student)

{

double sum = 0.0;

// Праходзім па кожнай адзнацы студэнта і сумуем іх

for (int i = 0; i < 5; ++i)

{

sum += student->grades[i];

}

// Разлічваем сярэдні бал і вяртаем яго

return sum / 5.0;

}

// Функцыя пошуку студэнта з самым раннім годам нараджэння ў звязаным спісе

LinkedList\* findOldestStudent(LinkedList\* p)

{

// Ініцыялізуем паказальнік на старэйшага студэнта першым элементам спісу

LinkedList\* oldestStudent = p;

// Ініцыялізуем паказальнік для ітэрацыі па спісе, пачынаючы з другога элемента

LinkedList\* ptr = p->next;

// Праходзім па астатніх элементах спісу

while (ptr)

{

// Параўноўваем гады нараджэння бягучага студэнта з годам нараджэння старэйшага студэнта

if (ptr->bdayYear < oldestStudent->bdayYear)

{

// Калі бягучы студэнт старэйшы, абнаўляем паказальнік на старэйшага студэнта

oldestStudent = ptr;

}

// Пераходзім да наступнага студэнту ў спісе

ptr = ptr->next;

}

// Вяртаем паказальнік на старэйшага студэнта

return oldestStudent;

}

// Функцыя пошуку студэнта з самым познім годам нараджэння ў звязаным спісе

LinkedList\* findYoungestStudent(LinkedList\* p)

{

// Ініцыялізуем паказальнік на малодшага студэнта першым элементам спісу

LinkedList\* youngestStudent = p;

// Ініцыялізуем паказальнік для ітэрацыі па спісе, пачынаючы з другога элемента

LinkedList\* ptr = p->next;

// Праходзім па астатніх элементах спісу

while (ptr)

{

// Параўноўваем гады нараджэння бягучага студэнта з годам нараджэння малодшага студэнта

if (ptr->bdayYear > youngestStudent->bdayYear)

{

// Калі бягучы студэнт маладзейшы, абнаўляем паказальнік на малодшага студэнта

youngestStudent = ptr;

}

// Пераходзім да наступнага студэнту ў спісе

ptr = ptr->next;

}

// Вяртаем паказальнік на малодшага студэнта

return youngestStudent;

}

// Функцыя вываду інфармацыі пра лепшага студэнта ў зададзенай групе

void printBestStudentInGroup(LinkedList\* p, int group)

{

// Ініцыялізуем паказальнік на лепшага студэнта і зменную для лепшага сярэдняга бала

LinkedList\* bestStudent = nullptr;

double bestAverage = 0.0;

// Ініцыялізуем паказальнік для ітэрацыі па звязаным спісе

LinkedList\* ptr = p;

// Праходзім па кожным студэнту ў спісе

while (ptr)

{

// Правяраем, ці належыць бягучы студэнт да зададзенай групы

if (ptr->group == group)

{

// Разлічваем сярэдні бал бягучага студэнта

double average = calculateAverageGrade(ptr);

// Правяраем, ці з'яўляецца бягучы студэнт лепшым

if (bestStudent == nullptr || average > bestAverage)

{

bestStudent = ptr;

bestAverage = average;

}

}

// Пераходзім да наступнага студэнту ў спісе

ptr = ptr->next;

}

// Выводзім інфармацыю пра лепшага студэнта ў групе

if (bestStudent != nullptr)

{

cout << "\nЛепшы студэнт у групе " << group << ":\n";

printStudent(bestStudent);

}

else

{

cout << "\nУ групе " << group << " студэнтаў няма.\n";

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Ініцыялізуем паказальнік на звязаны спіс

LinkedList\* l = nullptr;

// Дадаем некалькі студэнтаў у спіс

add(l);

add(l);

add(l);

add(l);

cout << "Зыходны спіс: \n";

print(l);

// Сартуем спіс

sortList(l);

cout << "Адсартаваны спіс:\n";

// Выводзім адсартаваны спіс

print(l);

cout << "Сярэдні бал па 5 прадметах для кожнай групы:\n";

// Разлічваем сярэдні бал для кожнай групы і выводзім вынік

calculateAverage(l);

// Знаходзім самага старэйшага і самага малодшага студэнта і выводзім інфармацыю пра іх

LinkedList\* oldest = findOldestStudent(l);

LinkedList\* youngest = findYoungestStudent(l);

cout << "\nСамы старэйшы студэнт:\n";

printStudent(oldest);

cout << "\nСамы малодшы студэнт:\n";

printStudent(youngest);

// Знаходзім лепшага студэнта ў кожнай групе і выводзім інфармацыю пра іх

for (int i = 1; i <= 4; ++i)

{

printBestStudentInGroup(l, i);

}

return 0;

}

// Функцыя для параўнання двух студэнтаў па курсе і прозвішчы па алфавітным парадку

bool sortingKey(const LinkedList\* firstStudent, const LinkedList\* secondStudent)

{

// Параўноўваем курсы студэнтаў

if (firstStudent->course == secondStudent->course)

{

// Калі курсы аднолькавыя, параўноўваем прозвішчы студэнтаў лексікаграфічна

return firstStudent->lastName < secondStudent->lastName;

}

// Калі курсы розныя, параўноўваем іх

return firstStudent->course < secondStudent->course;

}

// Функцыя для сартавання спісу

void sortList(LinkedList\*& p)

{

// Ствараем копію арыгінальнага спісу, каб не змяняць зыходныя дадзеныя

LinkedList\* originListCopy = p;

// Ініцыялізуем паказальнік на адсартаваны спіс

LinkedList\* sortedList = nullptr;

// Праходзім па арыгінальным спісе

while (p != nullptr)

{

// Захоўваем бягучага студэнта і перамяшчаем паказальнік на наступнага

LinkedList\* currentStudent = p;

p = p->next;

// Устаўляемы студэнта ва ўпарадкаваны спіс

// Калі спарадкаваны спіс пусты або бягучы студэнт "менш" за першы ва ўпарадкаваным спісе

if (sortedList == nullptr || sortingKey(currentStudent, sortedList))

{

// Устаўляем бягучага студэнта ў пачатак спарадкаванага спісу

currentStudent->next = sortedList;

currentStudent->prev = nullptr;

if (sortedList != nullptr)

{

sortedList->prev = currentStudent;

}

sortedList = currentStudent;

}

else

{

// Шукаем месца ва ўпарадкаваным спісе для ўстаўкі бягучага студэнта

// Пачынаем пошук з пачатку спарадкаванага спісу

LinkedList\* temp = sortedList;

// Ідзем па спарадкаваным спісе, пакуль не знойдзем элемент, большы за бягучага студэнта

while (temp->next != nullptr && !sortingKey(currentStudent, temp->next))

{

temp = temp->next;

}

// Устаўляем бягучага студэнта паміж вузламі

// Звязваем бягучага студэнта з папярэднім вузлом

currentStudent->prev = temp;

// Звязваем бягучага студэнта з наступным вузлом

currentStudent->next = temp->next;

// Калі наступны вузел існуе, звязваем яго з бягучым студэнтам

if (temp->next != nullptr)

{

temp->next->prev = currentStudent;

}

// Звязваем папярэдні вузел з бягучым студэнтам

temp->next = currentStudent;

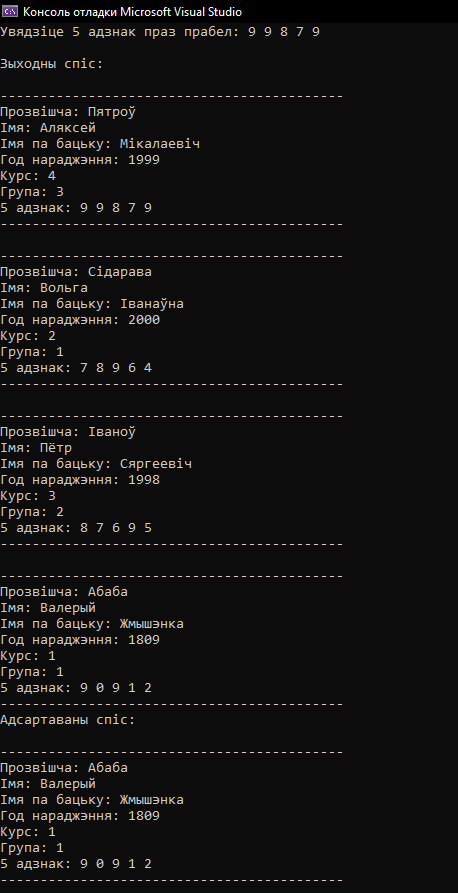
}

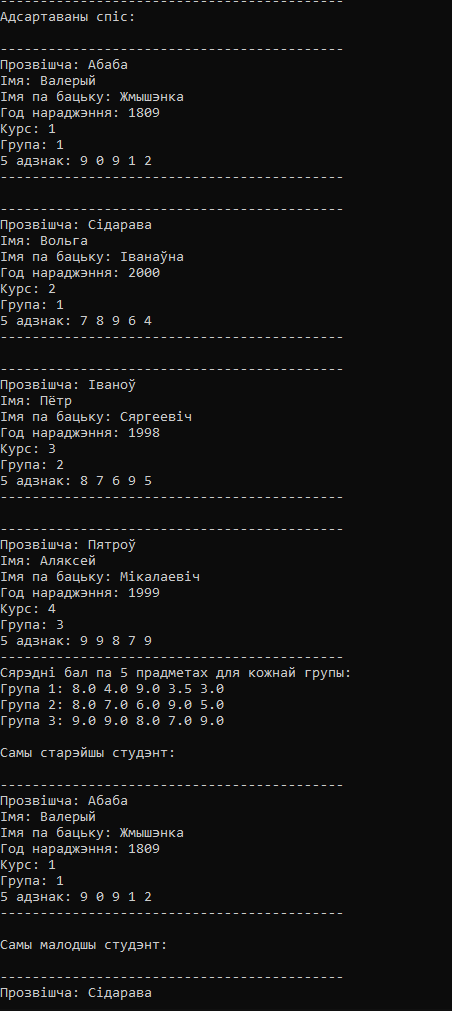
}

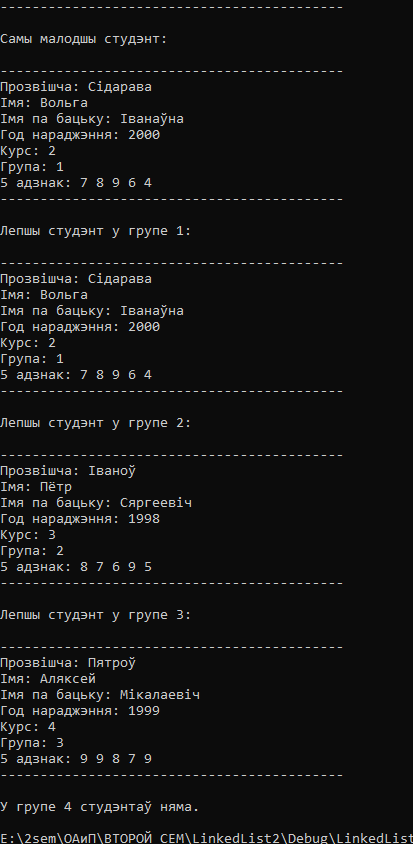
// Абнаўляем паказальнік на пачатак адсартаванага спісу

p = sortedList;

}







У метадычцы толькі 2 допа, ўсе роўна 15 балаў жа?

