Отчёт

Скворцова Анастасия Дмитриевна

#include <iostream>

#include <string>

Эти строки подключают библиотеки. #include <iostream> позволяет использовать стандартный ввод и вывод, а #include <string> позволяет работать со строками.

class Student {

public:

Student(int id, double avg, const std::string& name)

: id(id), AvgPoints(avg), name(name), next(nullptr) {

count++;

}

Объявляем класс Student, который представляет студента.

Здесь определен конструктор, который принимает три параметра: id (идентификатор студента), avg (средний балл) и name (имя студента).

В списке инициализации : id(id), AvgPoints(avg), name(name), next(nullptr) мы инициализируем переменные объекта, а также устанавливаем next в nullptr, чтобы указать, что у этого студента нет следующего элемента.

count++ увеличивает общее количество студентов на 1, так как мы создали нового студента.

~Student() { count--; }

Это деструктор класса Student, который вызывается, когда объект уничтожается.

Он уменьшает общее количество студентов (count) на 1, когда студент удаляется.

Student\* next;

int id;

double AvgPoints;

std::string name;

static int count;

};

Здесь мы объявляем переменные-члены класса Student:

Student\* next — указатель на следующего студента в списке.

int id — идентификатор студента.

double AvgPoints — средний балл студента.

std::string name — имя студента.

static int count — статическая переменная, которая хранит общее количество студентов.

int Student::count = 0;

Здесь мы инициализируем статическую переменную count значением 0. Это значит, что изначально нет студентов.

class StudentList {

private:

Student\* head = nullptr;

Мы объявляем класс StudentList, который представляет собой список студентов.

Student\* head — указатель на первого студента в списке. Изначально он установлен в nullptr, что означает, что список пуст.

public:

void addToBeginning(int id, double avg, const std::string& name) {

Объявляем метод addToBeginning, который добавляет нового студента в начало списка. Метод принимает идентификатор, средний балл и имя студента.

Student\* newStudent = new Student(id, avg, name);

Создаем нового студента, используя динамическое выделение памяти с помощью new.

newStudent->next = head;

head = newStudent;

Указываем, что следующий элемент для нового студента — это текущий первый студент (т.е. head).

Затем обновляем head, чтобы он указывал на нового студента, который теперь является первым в списке.

void addToEnd(int id, double avg, const std::string& name) {

Объявляем метод addToEnd, который добавляет нового студента в конец списка.

Student\* newStudent = new Student(id, avg, name);

Создаем нового студента.

if (!head) {

head = newStudent;

return;

}

Проверяем, пуст ли список (if (!head)). Если пуст, то новый студент становится первым элементом (head указывает на него).

Student\* temp = head;

while (temp->next) temp = temp->next;

Если список не пуст, проходим по нему до последнего студента, используя временный указатель temp.

temp->next = newStudent;

}

Указываем, что следующий элемент последнего студента — это новый студент.

void addAtIndex(int index, int id, double avg, const std::string& name) {

Объявляем метод addAtIndex, который добавляет студента по указанному индексу.

if (index == 0) {

addToBeginning(id, avg, name);

return;

}

Если индекс равен 0, вызываем метод addToBeginning, чтобы добавить студента в начало списка.

Student\* newStudent = new Student(id, avg, name);

Создаем нового студента.

Student\* temp = head;

for (int i = 0; i < index - 1 && temp; i++) temp = temp->next;

Проходим по списку, чтобы найти студента перед тем, кого нужно добавить.

if (!temp) {

std::cout << "Некорректный индекс" << std::endl;

delete newStudent;

return;

}

Если индекс некорректен (т.е. temp стал nullptr), выводим сообщение об ошибке и освобождаем память для студента.

newStudent->next = temp->next;

temp->next = newStudent;

}

Вставляем нового студента в список, указывая, что его следующий элемент — это тот, который был после temp, и обновляем next у temp.

void removeAtBeginning() {

if (!head) return;

Метод removeAtBeginning удаляет первого студента. Если список пуст, просто выходим из метода.

Student\* temp = head;

head = head->next;

delete temp;

}

Сохраняем указатель на первого студента в temp, обновляем head, чтобы указывать на второго студента, и освобождаем память для первого студента.

void removeAtIndex(int index) {

if (index == 0) {

removeAtBeginning();

return;

}

Метод removeAtIndex удаляет студента по указанному индексу. Если индекс 0, просто вызываем removeAtBeginning.

Student\* temp = head;

for (int i = 0; i < index - 1 && temp; i++) temp = temp->next;

Проходим по списку, чтобы найти студента перед тем, кого нужно удалить.

if (!temp || !temp->next) {

std::cout << "Некорректный индекс" << std::endl;

return;

}

Если индекс некорректен (т.е. temp стал nullptr или temp->next равен nullptr), выводим сообщение об ошибке.

Student\* toDelete = temp->next;

temp->next = toDelete->next;

delete toDelete;

}

Отмечаем студента для удаления, перенаправляем next у temp, чтобы обойти удаляемого студента, и освобождаем память для него.

void displayAll() const {

Метод displayAll выводит всех студентов в списке.

for (Student\* temp = head; temp; temp = temp->next) {

Используем цикл для прохождения по всем студентам в списке, начиная с head.

std::cout << "ID: " << temp->id

<< ", Средний балл: " << temp->AvgPoints

<< ", Имя: " << temp->name << std::endl;

}

}

};

Для каждого студента выводим его идентификатор, средний балл и имя.

int main() {

StudentList list;

Точка входа в программу. Создаем объект списка студентов.

list.addToEnd(1, 12.5, "Vasya");

list.addToEnd(2, 9.0, "Petya");

list.addToBeginning(3, 15.0, "Misha");

list.addAtIndex(1, 4, 10.0, "Sasha");

Добавляем студентов в список: два в конец, один в начало и один по индексу.

std::cout << "Список студентов:" << std::endl;

list.displayAll();

Выводим заголовок и отображаем всех студентов.

list.removeAtIndex(2);

Удаляем студента по индексу 2.

std::cout << "\nСписок студентов после удаления:" << std::endl;

list.displayAll();

Выводим заголовок для списка после удаления и снова отображаем студентов.

return 0;

}

Завершаем выполнение программы и возвращаем 0, что означает успешное завершение.