,,Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный технологический университет»

ФИТ

Кафедра Информационных систем и технологий 1курс

Группа 2-2

**Структуры данных**

Тема:

**Индексирование записей. Простой/сложный индекс**

Лабораторная работа 6, 7

**Выполнил:**

Волковец Сергей Николаевич

**Проверил:**

Белодед Николай Иванович

Оглавление

[Теоретические сведения 3](#_Toc196859008)

[Простой индекс 5](#_Toc196859009)

[Сложный индекс 6](#_Toc196859010)

[Основные операции, применимые к struct: 6](#_Toc196859011)

# Теоретические сведения

**Struct** (структура) - это тип данных в С++, который позволяет объединять несколько переменных разных типов в одну единицу данных. В структуре можно объединить переменные разных типов, такие как целые числа, числа с плавающей точкой, символьные строки, указатели и другие структуры.

Структуры объявляются с помощью ключевого слова "struct", за которым следует имя структуры и фигурные скобки, в которых указываются переменные, которые должны быть включены в структуру. Например, вот как объявляется простая структура с двумя полями:

struct Point {

int x;

int y;

};

В этом примере мы объявили структуру "Point", которая состоит из двух целочисленных полей "x" и "y". После объявления структуры можно создать переменную этого типа и обращаться к полям структуры, как показано ниже:

Point p1 = { 1, 2 };

p1.x = 3;

p1.y = 4;

В этом примере мы создали переменную "p1" типа "Point" и задали значения ее полей "x" и "y". Затем мы обратились к полям структуры отдельно, чтобы изменить их значения.

Структуры могут быть использованы для создания сложных типов данных, таких как списки, деревья, графы и другие структуры данных. Они также могут быть использованы для передачи нескольких значений в функцию вместо передачи каждой переменной отдельно.

**Массив структур** - это массив, который содержит элементы структурного типа данных. Такой массив можно использовать для хранения набора структур, каждая из которых содержит набор переменных различных типов.

Чтобы объявить массив структур в С++, нужно сначала определить структуру, которая будет использоваться для создания элементов массива. Например, следующий код определяет структуру "Person", которая содержит два поля:

struct Person {

std::string name;

int age;

};

Затем можно создать массив структур типа "Person", как показано ниже:

Person people[3];

К структурам данных в С++ применимы следующие операции:

**Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.**

**Индексирование записей** - это способ доступа к отдельным элементам записи (struct) в С++. Запись является типом данных, в котором связанные переменные объединяются в один объект.

Для доступа к отдельным полям внутри записи, можно использовать оператор доступа к членам (member access operator) - точку ".". Индексирование записей может использоваться для доступа к элементам в массиве записей или для доступа к элементам внутри одной записи.

Для индексирования записей в С++ используется оператор квадратных скобок "[]". Например, если у вас есть массив записей, вы можете обратиться к отдельным элементам массива, используя индексацию, как показано в следующем примере:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

struct Person {

string name;

int age;

};

Person people[3];

people[0].name = "Alice";

people[0].age = 25;

people[1].name = "Bob";

people[1].age = 30;

people[2].name = "Charlie";

people[2].age = 35;

cout << people[1].name << " is " << people[1].age << " years old." << endl;

return 0;

}



В этом примере мы создали структуру "Person" с полями "name" и "age". Затем мы объявили массив "people" из трех элементов, каждый из которых является записью типа "Person". Затем мы задали значения для полей каждого элемента массива и вывели на экран значения полей второго элемента массива.

Для структур данных, объявленных в программе, память выделяется автоматиче-ски средствами системы программирования либо на этапе компиляции, либо при активизации процедурного блока, в котором объявляются соответствующие переменные.

Главное заключается в том, что независимо от используемого языка программирования, имеющиеся в программе структуры данных не появляются "из ничего", а явно или неявно объявляются операторами создания структур. В результате этого всем структурам программы выделяется память для их размещения.

Простой индекс и сложный индекс - это две разные концепции в С++, связанные с обращением к элементам массива.

Простой индекс - это индекс, который используется для обращения к элементу массива, который находится на определенной позиции в массиве.

Изображение выглядит как коробка, Бумажное изделие, бумага, оригами

Контент, сгенерированный ИИ, может содержать ошибки.

Можно дать так же второе определение: Простой индекс - это структура данных, которая содержит список значений ключа и ссылок на соответствующие записи в файле данных. Простой индекс может быть создан для любого поля в файле данных, которое может быть использовано для быстрого поиска. Простой индекс может быть создан для единственного поля или для нескольких полей в файле данных.

Например, если у вас есть массив чисел, то простой индекс используется для обращения к определенному элементу массива, как в следующем примере:

int arr[5] = { 1, 2, 3, 4, 5 };

cout << arr[2]; // выведет 3, так как 3 находится на позиции 2 в массиве

В этом примере мы объявили массив "arr" из 5 элементов и вывели на экран значение элемента, который находится на позиции 2 (индекс 2) в массиве.

Сложный индекс - это индекс, который используется для обращения к элементу в многомерном массиве.

Изображение выглядит как Легкая закуска, еда

Контент, сгенерированный ИИ, может содержать ошибки.

Можно дать так же второе определение: Сложный индекс - это структура данных, которая содержит несколько полей и ссылок на соответствующие записи в файле данных. Сложный индекс может быть создан для любой комбинации полей в файле данных, которая может быть использована для быстрого поиска. Сложный индекс может быть создан для нескольких полей, которые используются вместе для поиска.

Если у вас есть, например, двумерный массив, то для доступа к его элементам вам нужно использовать два индекса: первый индекс обозначает номер строки, а второй индекс - номер столбца.

Например:

int matrix[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

cout << matrix[1][2]; // выведет 6, так как 6 находится во второй строке и третьем столбце

В этом примере мы объявили двумерный массив "matrix" из 3 строк и 3 столбцов и вывели на экран значение элемента, который находится во второй строке и третьем столбце.

# Основные операции, применимые к struct:

1. Присваивание (=):

Структуры могут быть присвоены друг другу с помощью оператора присваивания (=). Например:

struct Person {

std::string name;

int age;

};

Person p1 = { "Alice", 25 };

Person p2 = p1; // присваивание p2 значений p1

1. Сравнение (==, !=, <, >, <=, >=):

Структуры могут быть сравниваемы между собой, если они содержат сравнимые поля. Например:

Person p1 = { "Alice", 25 };

Person p2 = { "Bob", 30 };

if (p1.age < p2.age) {

std::cout << p1.name << " is younger than " << p2.name << std::endl;

}

1. Доступ к полям ("."):

Поля структуры можно получить, используя оператор доступа к полям (".") с именем структуры и именем поля. Например:

Person p = { "Alice", 25 };

std::cout << p.name << " is " << p.age << " years old." << std::endl;

1. Инициализация:

Структуры могут быть инициализированы при объявлении или во время выполнения программы. Например:

Person p1 = { "Alice", 25 };

Person p2;

p2.name = "Bob";

p2.age = 30;

1. Передача аргументов в функции:

Структуры могут быть переданы в функцию как параметры и возвращены из функции как результат. Например:

// Определение структуры Person с полями name и age

…

void printPerson(Person p);

// Функция createPerson, которая создает новый объект Person с заданным именем и возрастом

Person createPerson(string name, int age) {

Person p;

p.name = name;

p.age = age;

return p;

}

// Функция printPerson, которая выводит информацию о Person

void printPerson(Person p) {

cout << p.name << " is " << p.age << " years old." << endl;

}

int main() {

Person p1 = createPerson("Alice", 25);

Person p2 = createPerson("Bob", 30);

printPerson(p1);

printPerson(p2);

}

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

# Задачи

1. Основываясь на вашей схеме базы данных из лабораторной работы №2, реализовать структуры, создать индексы для каждой структуры, а так же реализовать вывод структур, линейный поиск данных по индексу (при вводе индекса, выводило/удаляло всю структуру).

Код:

// 6-7(1).cpp : Основываясь на вашей схеме базы данных из лаборат

// орной работы №2, реа-лизовать структуры,

// создать индексы для каждой структуры, а так же реали-зовать вывод структур,

// линейный поиск данных по индексу (при вводе ин-декса, выводило/удаляло всю структуру).

#include <iostream>

#include <string>

#include <vector>

#include <fstream> // Добавляем для работы с файлами

using namespace std;

struct Person {

int id;

string name;

};

void add(string s, vector<Person>& arr) {

Person temp;

temp.id = arr.size() + 1;

temp.name = s;

arr.push\_back(temp);

}

void out(vector<Person>& arr) {

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

cout << arr[i].id << " " << arr[i].name << endl;

}

}

void search(vector<Person>& arr) {

cout << "Введите номер нужной структуры ";

int n;

cin >> n;

cout << "Если нужно вывести введите 1, если удалить 2";

if (n == 1)

cout << arr[n].id << " " << arr[n].name << endl;

else {

arr.erase(arr.begin() + n);

}

}

void saveToFile(const vector<Person>& arr) {

ofstream outFile("in.txt");

if (!outFile) {

cout << "Ошибка открытия файла для записи!" << endl;

return;

}

for (const auto& p : arr) {

outFile << p.id << " " << p.name << "\n";

}

outFile.close();

cout << "Данные успешно записаны в файл " << "in.txt" << endl;

}

void readFromFile(vector<Person>& arr) {

ifstream inFile("out.txt");

if (!inFile) {

cout << "Ошибка открытия файла для чтения!" << endl;

return;

}

arr.clear();

Person temp;

while (inFile >> temp.id >> temp.name) {

arr.push\_back(temp);

}

inFile.close();

cout << "Данные успешно прочитаны из файла " << endl;

}

int main() {

setlocale(LC\_ALL, "ru");

vector<Person> arr;

int choice;

string name;

string filename;

do {

cout << "\nМеню:\n";

cout << "1. Добавить запись\n";

cout << "2. Вывести список записей\n";

cout << "3. Поиск/Удаление записи\n";

cout << "4. Записать данные в файл\n";

cout << "5. Прочитать данные из файла\n";

cout << "6. Выход\n";

cout << "Выберите действие: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Введите имя: ";

cin >> name;

add(name, arr);

break;

case 2:

out(arr);

break;

case 3:

search(arr);

break;

case 4:

saveToFile(arr);

break;

case 5:

readFromFile(arr);

break;

case 6:

cout << "Выход из программы..." << endl;

break;

default:

cout << "Неверный выбор!" << endl;

}

} while (choice != 6);

return 0;

}