Министерство Образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

**СТРУКТУРЫ ДАННЫХ**

**Индексирование записей. Простой/сложный индекс**

**Лабораторная работа № 6, 7.**

Студент:

1курса 2 группы 1 подгруппы

Марушко Тимофей Фёдорович

Проверяющий:

Белодед Николай Иванович

**ОГЛАВЛЕНИЕ**

[**Теоритические сведения 3**](#_Toc193284364)

[**1. Struct (Структура) 3**](#_Toc193284365)

[**2. Массив структур 3**](#_Toc193284366)

[**3. Индексирование записей 3**](#_Toc193284367)

[**4. Простой индекс 5**](#_Toc193284368)

[**5. Основные операции, применимые к struct 7**](#_Toc193284369)

[**6. Указатель на структуру и массив структур 10**](#_Toc193284370)

[**Решение ОСНОВНЫХ ЗАДАЧ 13**](#_Toc193284371)

[**Решение Дополнительных задач 16**](#_Toc193284372)

Теоритические сведения

1. Struct (Структура)

**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:

1. Массив структур

**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:

1. Индексирование записей

**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.

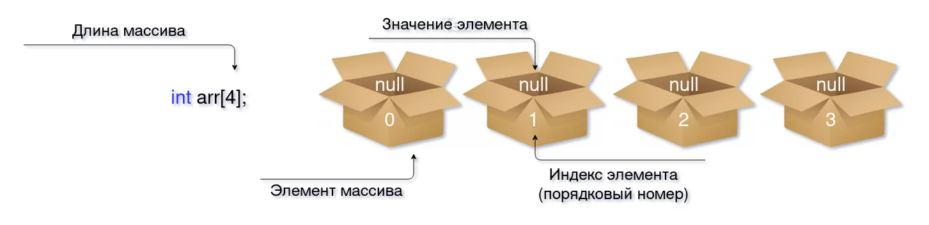
Для структур данных, объявленных в программе, память выделяется автоматически средствами системы программирования либо на этапе компиляции, либо при активизации процедурного блока, в котором объявляются соответствующие переменные.

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.

1. Простой индекс

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.



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.

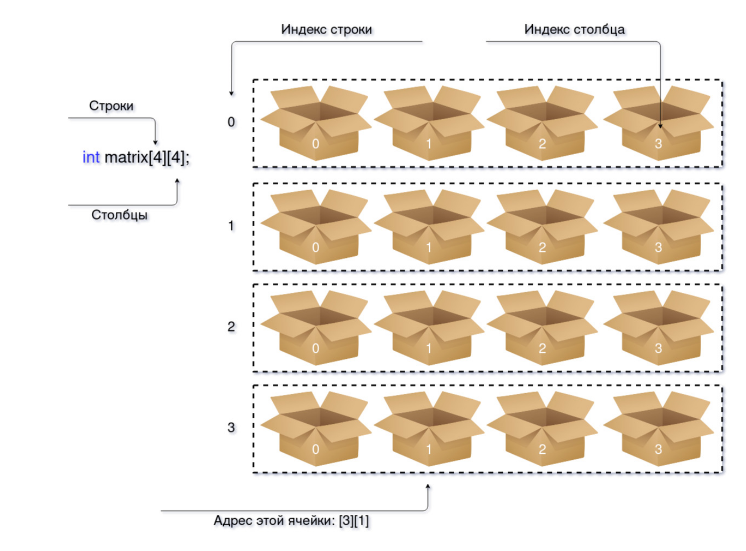
Например, если у вас есть массив чисел, то простой индекс используется для обращения к определенному элементу массива, как в следующем примере:

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.



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.

1. Основные операции, применимые к struct
2. 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.

**Структурность данных и технология программирования**:

При структурировании больших программных изделий возможно применение подхода, основанного на структуризации алгоритмов и известного как "нисходящее" проектирование или "программирование сверху вниз", или подхода, основанного на структуризации данных и известного как "восходящее" проектирование или "программирование снизу вверх".

В первом случае структурируют прежде всего действия, которые должна выполнять программа. Большую и сложную задачу, стоящую перед проектируемым программным изделием, представляют в виде нескольких подзадач меньшего объема. Таким образом, модуль самого верхнего уровня, отвечающий за решение всей задачи в целом, получается достаточно простым и обеспечивает только последовательность обращений к модулям, реализующим подзадачи. На первом этапе проектирования модули подзадач выполняются в виде "заглушек". Затем каждая подзадача в свою очередь подвергается декомпозиции по тем же правилам. Процесс дробления на подзадачи продолжается до тех пор, пока на очередном уровне декомпозиции получают подзадачу, реализация которой будет вполне обозримой. В предельном случае декомпозиция может быть доведена до того, что подзадачи самого нижнего уровня могут быть решены элементарными инструментальными средствами (например, одним оператором выбранного языка программирования).

Изображение выглядит как зарисовка, рисунок, диаграмма, Штриховая графика

Контент, сгенерированный ИИ, может содержать ошибки.

Чрезвычайно продуктивным технологическим приемом, связанным со структури-зацией данных, является инкапсуляция. Смысл ее состоит в том, что сконструи-рованный новый тип данных - "строительный блок" - оформляется таким образом, что его внутренняя структура становится недоступной для программиста-пользователя этого типа. Программист, использующий этот тип данных в своей программе (в модуле более высокого уровня), может оперировать с данными этого типа только через вызовы функций, определенных для этого. Новый тип дан-ных представляется для него в виде "черного ящика", для которого известны вхо-ды и выходы, но содержимое - неизвестно и недоступно. Инкапсуляция чрезвычайно полезна и как средство преодоления сложности, и как средство защиты от ошибок. Первая цель достигается за счет того, что сложность внутренней структуры нового типа данных и алгоритмов выполнения операций над ним исключается из поля зрения программиста-пользователя. Вторая цель достигается тем, что возможности доступа пользователя ограничиваются лишь заведомо корректными входными точками, следовательно, снижается и вероятность ошибок.

1. Указатель на структуру и массив структур

**Указатель на структуру** — это переменная, которая содержит адрес в памяти структуры. Она позволяет работать с полями структуры, используя адрес этой структуры.

Для определения указателя на структуру необходимо сначала определить саму структуру.

struct student {

char name[50];

int age;

float gpa;

};

Чтобы создать указатель на структуру student, необходимо использовать синтаксис указателей. Например:

struct student\* ptr;

Этот код создает указатель ptr на тип struct student. Затем можно использовать оператор & для получения адреса структуры. Например:

struct student s;

ptr = &s;

Чтобы обратиться к полям структуры через указатель, используйте операторы \* и ->. Например:

(\*ptr).age = 20;

ptr->gpa = 3.5;

Эти операторы используются для доступа к полям структуры через указатель. Первый оператор (\*ptr) разыменовывает указатель и возвращает значение структуры, а затем обращается к полю age. Второй оператор ptr->gpa автоматически разыменовывает указатель и обращается к полю gpa.

**Указатель на массив структур** - это способ обращения к элементам массива структур через указатель на первый элемент.

struct student {

char name[50];

int age;

float gpa;

};

struct student class[10];

Этот код определяет массив структур class из 10 элементов типа struct student. Каждый элемент массива содержит поля name, age и gpa.

Чтобы создать указатель на структуру student, необходимо использовать синтаксис указателей. Например:

struct student\* ptr;

Этот код создает указатель ptr на тип struct student. Затем можно использовать оператор & для получения адреса структуры. Например:

struct student s;

ptr = &s;

Этот код создает переменную s типа struct student и присваивает ей значения полей. Затем он создает указатель ptr на тип struct student и присваивает ему адрес структуры s.

Чтобы обратиться к полям структуры через указатель, используйте операторы \* и ->. Например:

(\*ptr).age = 20;

ptr->gpa = 3.5;

**Пример со списком рассылки**:

// объявление массива структур

….

void init\_list(void)

{

for (int t = 0; t < MAX; ++t) {

addr\_list[t].name = ""; // очистка полей структуры

addr\_list[t].street = "";

addr\_list[t].city = "";

addr\_list[t].state = "";

addr\_list[t].zip = 0;

}

}

int menu\_select(void)

{

string s;

int c;

cout << "1. Введите имя\n";

cout << "2. Удалите имя\n";

cout << "3. Выведите список\n";

cout << "4. Выход\n";

do {

cout << "\nВведите номер нужного пункта: ";

getline(cin, s);

c = stoi(s); // считывание номера выбранного пункта

} while (c < 0 || c>4); // проверка на корректность ввода

return c;

}

int find\_free(void)

{

int t{}; // инициализация переменной t

for (int t = 0; t < MAX && !addr\_list[t].name.empty(); ++t); // поиск свободной структуры

if (t == MAX) return -1; /\* свободных структур нет \*/

return t; // возврат номера свободной структуры

}

void enter(void)

{

int slot;

string s;

slot = find\_free(); // поиск свободной структуры

if (slot == -1) {

cout << "\nСписок заполнен";

return;

}

cout << "Введите имя: ";

getline(cin, addr\_list[slot].name); // считывание поля имени

cout << "Введите улицу: ";

getline(cin, addr\_list[slot].street); // считывание поля улицы

cout << "Введите город: ";

getline(cin, addr\_list[slot].city); // считывание поля города

cout << "Введите штат: ";

getline(cin, addr\_list[slot].state); // считывание поля штата

cout << "Введите почтовый код: ";

getline(cin, s);

addr\_list[slot].zip = stoul(s); // считывание поля почтового кода

}

void del(void)

{

int slot;

string s;

cout << "Введите № записи: ";

getline(cin, s);

slot = stoi(s); // считывание номера записи для удаления

if (slot >= 0 && slot < MAX)

addr\_list[slot].name = ""; // удаление поля имени

}

void list(void)

{

for (int t = 0; t < MAX; ++t) {

if (!addr\_list[t].name.empty()) { // проверка, что структура не пустая

cout << addr\_list[t].name << endl;

cout << addr\_list[t].street << endl;

cout << addr\_list[t].city << endl;

cout << addr\_list[t].state << endl;

cout << addr\_list[t].zip << endl << endl; // вывод полей структуры

}

}

cout << "\n\n";

}

int main(void)

{

// инициализация списка

….

// выбор пункта меню

….

// добавление записи

….

// удаление записи

….

// вывод списка

….

// выход из программы

}

}

return 0;

Решение ОСНОВНЫХ ЗАДАЧ

**Условие:**

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).

**Решение:**

*#include* <iostream>

*#include* <vector>

*#include* <unordered\_map>

*using* *namespace* std;

*struct* *Teacher* {

*int* id;

*string* name;

*string* subject;

*string* phone;

*string* city;

};

*class* *University* {

*private:*

vector*<Teacher>* teachers;

unordered\_map*<int*, *Teacher\*>* teacherIndex;

unordered\_map*<string*, *Teacher\*>* phoneIndex;

unordered\_multimap*<string*, *Teacher\*>* cityIndex;

*int* nextId *=* 1;

*public:*

*void* addTeacher(*const* *string&* name, *const* *string&* subject, *const* *string&* phone, *const* *string&* city) {

teachers.push\_back({ nextId, name, subject, phone, city });

teacherIndex[nextId] *=* *&*teachers.back();

phoneIndex[phone] *=* *&*teachers.back();

cityIndex.insert({ city, *&*teachers.back() });

cout << "Преподаватель добавлен с ID: " << nextId << endl;

nextId*++*;

}

*void* findTeacher(*int* id) {

*if* (teacherIndex.find(id) != teacherIndex.end()) {

*Teacher\** t *=* teacherIndex[id];

cout << "Найден преподаватель: " << t->name << ", предмет: " << t->subject << ", телефон: " << t->phone << ", город: " << t->city << endl;

} *else* {

cout << "Преподаватель с таким ID не найден." << endl;

}

}

*void* findTeacherByPhone(*const* *string&* phone) {

*if* (phoneIndex.find(phone) != phoneIndex.end()) {

*Teacher\** t *=* phoneIndex[phone];

cout << "Найден преподаватель: " << t->name << ", предмет: " << t->subject << ", телефон: " << t->phone << ", город: " << t->city << endl;

} *else* {

cout << "Преподаватель с таким номером телефона не найден." << endl;

}

}

*void* findTeachersByCity(*const* *string&* city) {

*auto* range *=* cityIndex.equal\_range(city);

*bool* found *=* *false*;

*for* (*auto* it *=* range.first; it != range.second; ++it) {

*Teacher\** t *=* it->second;

cout << "Найден преподаватель: " << t->name << ", предмет: " << t->subject << ", телефон: " << t->phone << ", город: " << t->city << endl;

found *=* *true*;

}

*if* (*!*found) {

cout << "Преподаватели в данном городе не найдены." << endl;

}

}

*void* removeTeacher(*int* id) {

*if* (teacherIndex.find(id) != teacherIndex.end()) {

*Teacher\** t *=* teacherIndex[id];

phoneIndex.erase(t->phone);

*auto* range *=* cityIndex.equal\_range(t->city);

*for* (*auto* it *=* range.first; it != range.second; ) {

*if* (it->second *==* t) {

it = cityIndex.erase(it);

} *else* {

++it;

}

}

teacherIndex.erase(id);

teachers.erase(remove\_if(teachers.begin(), teachers.end(), [id](*const* *Teacher&* t) {

*return* t.id *==* id;

}), teachers.end());

cout << "Преподаватель удален." << endl;

} *else* {

cout << "Преподаватель с таким ID не найден." << endl;

}

}

*void* listTeachers() {

*for* (*const* *auto&* t : teachers) {

cout << "ID: " << t.id << ", Имя: " << t.name << ", Предмет: " << t.subject << ", Телефон: " << t.phone << ", Город: " << t.city << endl;

}

}

};

*int* main() {

*University* uni;

*int* choice, id;

*string* name, subject, phone, city;

*while* (*true*) {

cout << "\n1. Добавить преподавателя\n2. Найти преподавателя по ID\n3. Найти преподавателя по телефону\n4. Найти преподавателей по городу\n5. Удалить преподавателя\n6. Список преподавателей\n7. Выйти\nВаш выбор: ";

cin >> choice;

*switch* (choice) {

*case* 1:

cout << "Введите имя: ";

cin >> name;

cout << "Введите предмет: ";

cin >> subject;

cout << "Введите телефон: ";

cin >> phone;

cout << "Введите город: ";

cin >> city;

uni.addTeacher(name, subject, phone, city);

*break*;

*case* 2:

cout << "Введите ID: ";

cin >> id;

uni.findTeacher(id);

*break*;

*case* 3:

cout << "Введите телефон: ";

cin >> phone;

uni.findTeacherByPhone(phone);

*break*;

*case* 4:

cout << "Введите город: ";

cin >> city;

uni.findTeachersByCity(city);

*break*;

*case* 5:

cout << "Введите ID: ";

cin >> id;

uni.removeTeacher(id);

*break*;

*case* 6:

uni.listTeachers();

*break*;

*case* 7:

*return* 0;

*default*:

cout << "Неверный ввод! Попробуйте снова." << endl;

}

}

}

Решение Дополнительных задач

**Условие:**

Напишите программу, которая считывает список файлов из указанной директории и выводит на экран их размеры в байтах. (3 балла)

**Решение:**

void AdditionalTask1() {

    string dir;

    cout << "Enter the directory path: ";

    cin >> dir;

*for* (const auto& entry : filesystem::directory\_iterator(dir)) {

        cout << entry.path().filename() << " : " << filesystem::file\_size(entry.path()) << " bytes" << endl;

    }

}

**Условие:**

Напишите программу, которая считывает список файлов из указанной директории и выводит на экран их размеры в байтах. (3 балла)

**Решение:**

void AdditionalTask1() {

    string dir;

    cout << "Enter the directory path: ";

    cin >> dir;

*for* (const auto& entry : filesystem::directory\_iterator(dir)) {

        cout << entry.path().filename() << " : " << filesystem::file\_size(entry.path()) << " bytes" << endl;

    }

}

**Условие:**

Напишите программу, которая считывает список файлов из указанной директории и выводит на экран их размеры в байтах. (3 балла)

**Решение:**

*void* findTeacherByPhone(*const* *string&* phone) {

*if* (phoneIndex.find(phone) != phoneIndex.end()) {

*Teacher\** t *=* phoneIndex[phone];

cout << "Найден преподаватель: " << t->name << ", предмет: " << t->subject << ", телефон: " << t->phone << ", город: " << t->city << endl;

} *else* {

cout << "Преподаватель с таким номером телефона не найден." << endl;

}

}