

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ»

Структуры данных

Лабораторный практикум

Минск 2022

ПРЕДИСЛОВИЕ

Практикум содержит задания для выполнения лабораторных работ на основе приложения **Microsoft Visual Studio**. В каждой работе имеются краткие теоретические сведения по рассматриваемым вопросам.

Преподаватель определяет, какие лабораторные работы должны выполнять студенты и в каком объеме. Предполагается, что выполнение большинства лабораторных работ занимает у студентов два академических часа.

Задания для выполнения лабораторных работ содержат кнопки, при нажатии на которые открываются тесты, предназначенные для контроля знаний студентов. Тестирование происходит по команде преподавателя и занимает несколько минут.

Для работы тестирующих программ предварительно в приложении Word надо разрешить использование макросов. При этом тексты ответов на формах располагаются каждый раз случайным образом, и ответить на вопросы можно только один раз, так как после нажатия на кнопку «Результаты» форма с вопросами и вариантами ответов исчезает.

При **оформлении отчетов по лабораторным работам** необходимо использовать приложение **Word**. Каждый отчет должен содержать название работы, условия задач, блок-схемы алгоритмов, тексты разработанных программ, скриншоты результатов выполнения программ.

В верхнем колонтитуле записывается фамилия студента и номер группы, в нижнем – номера страниц. Шрифт – 10 или 12, интервал – одинарный, поля – по 1,5 см. Все отчеты сохраняются в **одном** файле.

ОГЛАВЛЕНИЕ

Лабораторная работа №1 Перечисление. Объединение. STL контейнеры. Массив. Дек. Однодвух связный список. Вектор;

Лабораторная работа №2 Схема БД. Меню программы;

Лабораторная работа №3 Двоичные и текстовые файлы;

Лабораторная работа №4 Индексирование записей. Простой/сложный индекс;

Лабораторная работа №5 Запись/чтение массива структур в файл;

Лабораторная работа №6 Редактирование файлов: удаление, изменение поля;

Лабораторная работа №7 Сортировка записей;

Лабораторная работа №8 Фильтрация данных;

Лабораторная работа №9 Поиск записи по значению/индексу.

Лабораторная работа №1

Перечисление. Объединение. STL контейнеры.

Массив. Дек. Одно/двух связный список. Вектор;

Структура данных — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

Перечисление (или «*перечисляемый тип*») — это тип данных, где любое значение (или «*перечислитель*») определяется как символьная константа. Объявить перечисление можно с помощью **ключевого слова enum**. объявление перечислений не требует выделения памяти. Только когда переменная перечисляемого типа определена, только тогда выделяется память для этой переменной.

Обратите внимание, каждый перечислитель отделяется запятой, а само перечисление заканчивается точкой с запятой.

```
#include <iostream>

using namespace std;

enum Colors {
    RED,
    GREEN,
    BLUE,
};

int main()
{
    setlocale(LC_ALL, "rus");
    Colors car = RED;
    if (car == RED)
        cout << "Машина красного цвета" << endl;
    else
        cout << "Машина зеленая или синяя" << endl;
}
```

Результат выполнения программы:
Машина красного цвета

Объединение – это группирование переменных, которые разделяют одну и ту же область памяти. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение начинаются с одной границы.

Объединение позволяет представить в компактном виде данные, которые могут изменяться. Одни и те же данные могут быть представлены разными способами с помощью объединений.

Точно также как и **структуры**, объединения требуют объявления типа (шаблона) и объявления переменной этого типа.

```
#include <iostream>

using namespace std;

union NewUnion {
    short int a;
    int b;
    double c;
};

int main()
{
```

Пример:

Контейнерный класс (или «*класс-контейнер*») в языке C++ — это **класс**, предназначенный для хранения и организации нескольких объектов определенного типа данных (пользовательских или фундаментальных). Существует много разных контейнерных классов, каждый из которых имеет свои преимущества, недостатки или ограничения в использовании. Безусловно, наиболее часто используемым контейнером в программировании является **массив**, который мы уже использовали во многих примерах. Хотя в языке C++ есть стандартные обычные массивы, большинство программистов используют контейнерные классы-массивы: **std::array** или **std::vector** из-за преимуществ, которые они предоставляют. В отличие от стандартных массивов, контейнерные классы-массивы имеют возможность динамического изменения своего размера, когда элементы добавляются или удаляются. Это не только делает их более удобными, чем обычные массивы, но и безопаснее.

Обычно, **функционал классов-контейнеров** языка C++ следующий:

1. Создание пустого контейнера (через **конструктор**).
2. Добавление нового объекта в контейнер.
3. Удаление объекта из контейнера.
4. Просмотр количества объектов, находящихся на данный момент в контейнере.
5. Очистка контейнера от всех объектов.
6. Доступ к сохраненным объектам.
7. Сортировка объектов/элементов (не всегда).

Стандартная библиотека шаблонов (сокр. «**STL**» от «*Standard Template Library*») — это часть Стандартной библиотеки C++, которая содержит набор шаблонов контейнерных классов (например, **std::vector** и **std::array**), алгоритмов и итераторов. Положительным моментом является то, что вы можете использовать эти классы без необходимости писать и отлаживать их самостоятельно (и разбираться в том, как они реализованы). Кроме того, вы получаете достаточно эффективные (и уже много раз

протестированные) версии этих классов. Недостатком является то, что не всё так просто/очевидно с функционалом Стандартной библиотеки шаблонов и это может быть несколько непонятно новичку, так как большинство классов на самом деле являются **шаблонами классов**.

Пример:

```
#include <iostream>
#include <array>

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");
    array<int, 4> newarray = { 8, 6, 4, 1 };
    cout << "2-ой элемент массива: " << newarray[1] << endl;
    cout << "Длина: " << newarray.size() << endl;
    cout << "3-ий элемент массива: " << newarray.at(2) << endl;
}
```

Результат выполнения программы:

6 4 4

Массив — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона. Представленный в C++11, **std::array** — это фиксированный массив, который не распадается в указатель при передаче в функцию. **std::array** определяется в **заголовочном файле** `array`, внутри **пространства имен** `std`. В отличие от стандартных фиксированных массивов, в **std::array** вы не можете пропустить (не указывать) длину массива.

Класс vector (или просто «**вектор**») — это **динамический массив**, способный увеличиваться по мере необходимости для содержания всех своих элементов. Класс `vector` обеспечивает произвольный доступ к своим элементам через **оператор индексации** `[]`, а также поддерживает добавление и удаление элементов.

В следующей программе мы добавляем 5 целых чисел в вектор и с помощью **перегруженного оператора индексации** `[]` получаем к ним доступ для их последующего вывода:

Пример:

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");
    vector<int> vect;
    for (int i = 0; i < 5; i++)
        vect.push_back(10 - i); // добавляем числа в конец массива

    for (int i = 0; i < vect.size(); i++)
        cout << vect[i] << " ";

    cout << "\n";
}
```

Результат выполнения программы:

10 9 8 7 6



Корень списка

Класс deque (или просто «дек») — это двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов. Например:

```
#include <iostream>
#include <deque>

int main()
{
    std::deque<int> deq;
    for (int count = 0; count < 4; ++count)
    {
        deq.push_back(count); // вставляем числа в конец массива
        deq.push_front(10 - count); // вставляем числа в начало массива
    }

    for (int index = 0; index < deq.size(); ++index)
        std::cout << deq[index] << ' ';

    std::cout << '\n';
}
```

Результат выполнения программы:

```
7 8 9 10 0 1 2 3
```

Односвязный список - это динамическая структура данных, состоящая из узлов. Каждый узел будет иметь какое-то значение и указатель на следующий узел. Картинка демонстрирует как будет выглядеть список. Напрямую в списке `forward_list` можно получить только первый элемент. Для этого применяется функция **front()**.

Чтобы удалить элемент из контейнера `forward_list` можно использовать следующие функции:

- **clear()**: удаляет все элементы
- **pop_front()**: удаляет первый элемент
- **erase_after(p)**: удаляет элемент после элемента, на который указывает итератор `p`. Возвращает итератор на элемент после удаленного
- **erase_after(begin, end)**: удаляет диапазон элементов, на начало и конец которого указывают соответственно итераторы `begin` и `end`. Возвращает итератор на элемент после последнего удаленного

Для добавления элементов в `forward_list` применяются следующие функции:

- **`push_front(val)`**: добавляет объект `val` в начало списка
- **`emplace_front(val)`**: добавляет объект `val` в начало списка
- **`emplace_after(p, val)`**: вставляет объект `val` после элемента, на который указывает итератор `p`. Возвращает итератор на вставленный элемент. Если `p` представляет итератор на позицию после конца списка, то результат неопределен.
- **`insert_after(p, val)`**: вставляет объект `val` после элемента, на который указывает итератор `p`. Возвращает итератор на вставленный элемент.
- **`insert_after(p, n, val)`**: вставляет `n` объектов `val` после элемента, на который указывает итератор `p`. Возвращает итератор на последний вставленный элемент.
- **`insert_after(p, begin, end)`**: вставляет после элемента, на который указывает итератор `p`, набор объектов из другого контейнера, начало и конец которого определяется итераторами `begin` и `end`. Возвращает итератор на последний вставленный элемент.

```
#include <iostream>
#include <forward_list>

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");
    forward_list<int> newlist = { 6, 2, 8, 4, 5 };
    cout << "1-ый член массива: " << newlist.front() << endl;
    // newlist.back() не работает, так как в односвязном списке напрямую
    // можно получить доступ только к 1-ому элементу
    auto iter = newlist.begin(); // итератор указывает на 1-ой элемент
    newlist.emplace_after(iter, 15); // добавляем после 1-ого элемента
    for (int n : newlist)
    {
        cout << n << "\t";
    }
}
```

List (или просто «*список*») — это двусвязный список, каждый элемент которого содержит 2 указателя: один указывает на следующий элемент списка, а другой — на предыдущий элемент списка. `List` предоставляет доступ только к началу и к концу списка — произвольный доступ запрещен. Если вы хотите найти значение где-то в середине, то вы должны начать с одного конца и перебирать каждый элемент списка до тех пор, пока не найдете то, что ищете. Преимуществом двусвязного списка является то, что добавление элементов происходит очень быстро, если вы, конечно, знаете, куда хотите добавлять. Обычно для перебора элементов двусвязного списка используются итераторы.

Тем не менее для контейнера list можно использовать функции **front()** и **back()**, которые возвращают соответственно первый и последний элементы.

push_back(val): добавляет значение val в конец списка

push_front(val): добавляет значение val в начало списка



Задания для самостоятельного выполнения к лабораторной работе 1:

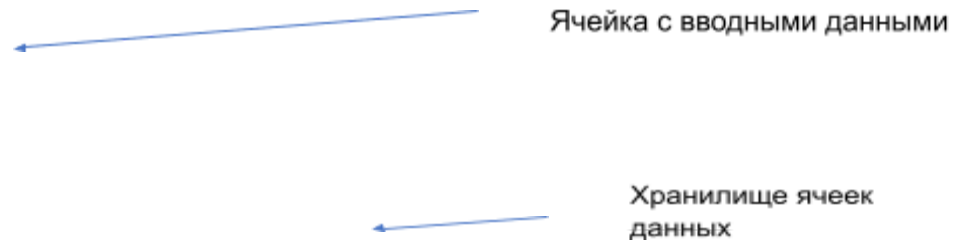
1. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить состояния РС (вкл, выкл, спящий режим), с использованием перечислений.

2. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить цвета радуги, с использованием перечислений.
3. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить название планеты Солнечной системы, с использованием перечислений.
4. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить дни недели, с использованием перечислений.
5. Написать программу, в которой в конец односвязного списка добавляется n элементов и вывести на экран.
6. Написать программу, в которой в начало двусвязного списка добавляется n элементов и вывести на экран.
7. Написать программу, в которой нужно найти максимальный элемент вектора и вывести на экран.
8. Написать программу, в которой нужно найти минимальный элемент дэки и добавить его в начало, с выводом на экран .
9. Написать программу, в которой нужно найти сумму минимального и максимального элементов массива, с выводом на экран.
10. Написать программу, в которой нужно найти сумму всех элементов вектора и вывести ее на экран.
11. Написать программу, в которой удалить n элемент из двусвязного списка и добавить такой же в начало, с выводом на экран.
12. Написать программу, в которой нужно найти произведение всех элементов деки и вывести его на экран.
13. Написать программу, в которой нужно удалить все элементы односвязного списка и добавить n новых, с выводом на экран.

Лабораторная работа №2 Схема БД. Меню программы;

Порой, чтобы продемонстрировать работу консольной программы бывает удобно воспользоваться меню — элементом пользовательского интерфейса, позволяющим выбрать одну из нескольких перечисленных опций программы. Иногда же наличие меню является обязательным условием задания по программированию.

На рисунке изображена схема ввода данных и обращения к хранилищу этих данных:



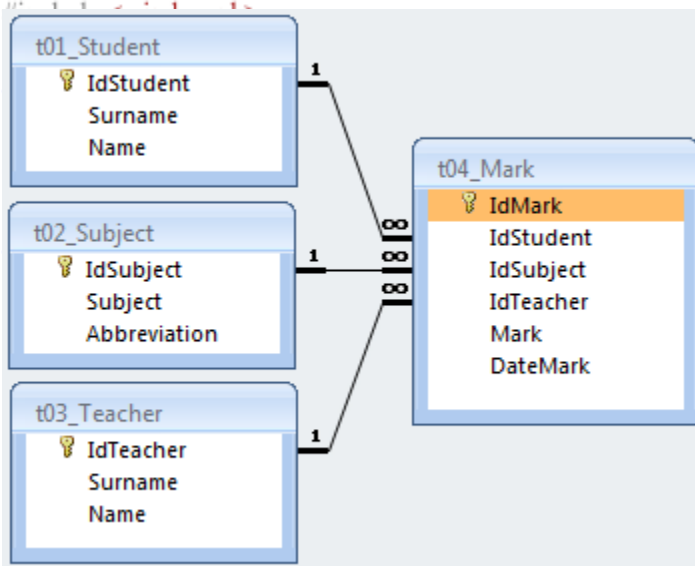
Прежде чем добавлять меню в программу, следует разбить программу на процедуры, которые должны будут выполняться при выборе соответствующих пунктов меню.

Наиболее часто встречаются задачи по:

1. Выводу пунктов меню на экран
2. Считыванию введенного пункта меню
3. Обработке выбранного пункта (посредством создания собственного обработчика для каждого пункта)

Пример кода для меню по схеме приведенной выше:

```
#include <iostream>
```



```
int main()
{
    setlocale(LC_ALL, "rus");
    Menu();
    while (_stateMenu != 0) {
```

Результат выглядит следующим образом:

```
Выберите действие:  
(1) Ввод данных  
(2) Вывод  
(3) Удаление  
Ваш выбор:
```

Существуют разные способы реализации консольного меню, в этом случае был использован способ со **switch-конструкцией**.

Задания к лабораторной работе (брать задания в соответствии со своим вариантом), написать консольное меню и составить схему для следующих случаев:

ПРИМЕЧАНИЕ! От исполнителя на данном этапе не требуется создания функционального программного продукта, только рабочий интерфейс в котором учтены все поставленные условия!

Функциональная настройка подвергнется рассмотрению в следующих лр!

Гостиница предоставляет номера клиентам на определенный срок. Каждый номер характеризуется вместимостью,

1. комфортностью (люкс, полулюкс, обычный) и ценой. Вашими клиентами являются различные лица, о которых Вы собираете определенную информацию (фамилия, имя, отчество и некоторый комментарий).
Сдача номера клиенту производится при наличии свободных мест в номерах, подходящих клиенту по указанным выше параметрам. При поселении фиксируется дата поселения. При выезде из гостиницы для каждого места запоминается дата освобождения.
2. Вы работаете в страховой компании. Вашей задачей является отслеживание финансовой деятельности компании.
Компания имеет различные филиалы по всей стране. Каждый филиал характеризуется названием, адресом и телефоном. Деятельность компании организована следующим образом: к Вам обращаются различные лица с целью заключения договора о страховании.
В зависимости от принимаемых на страхование объектов и страхуемых рисков, договор заключается по определенному виду страхования

(например, страхование автотранспорта от угона, страхование домашнего имущества, добровольное медицинское страхование). При заключении договора Вы фиксируете дату заключения, страховую сумму, вид страхования, тарифную ставку и филиал, в котором заключался договор. Нужно учесть, что договоры заключают страховые агенты. Помимо информации об агентах (фамилия, имя, отчество, адрес, телефон), нужно еще хранить информацию о филиале, в котором работают агенты. Кроме того, нужно иметь возможность рассчитывать заработную плату агентам. Заработная плата составляет некоторый процент от страхового платежа (страховой платеж — это страховая сумма, умноженная на тарифную ставку). Процент зависит от вида страхования, по которому заключен договор.

3. Вы работаете в ломбарде. Вашей задачей является отслеживание финансовой стороны работы ломбарда.
Деятельность Вашей компании организована следующим образом: к Вам обращаются различные лица с целью получения денежных средств под залог определенных товаров. У каждого из приходящих к Вам клиентов Вы запрашиваете фамилию, имя, отчество и другие паспортные данные. После оценивания стоимости принесенного в качестве залога товара Вы определяете сумму, которую готовы выдать на руки клиенту, а также свои комиссионные. Кроме того, определяете срок возврата денег. Если клиент согласен, то Ваши договоренности фиксируются в виде документа, деньги выдаются клиенту, а товар остается у Вас. В случае если в указанный срок не происходит возврата денег, товар переходит в Вашу собственность. После перехода прав собственности на товар, ломбард может продавать товары по цене, меньшей или большей, чем была заявлена при сдаче.
4. Вы работаете в компании, занимающейся оптово-розничной продажей различных товаров. Вашей задачей является отслеживание финансовой стороны работы компании.
Деятельность Вашей компании организована следующим образом: Ваша компания торгует товарами из определенного спектра. Каждый из этих товаров характеризуется наименованием, оптовой ценой, розничной ценой и справочной информацией.
В Вашу компанию обращаются покупатели. Для каждого из них Вы запоминаете в базе данных стандартные данные (наименование, адрес, телефон, контактное лицо) и составляете по каждой сделке документ, запоминая наряду с покупателем количество купленного им товара и дату покупки.
Обычно покупатели в рамках одной сделки покупают не один товар, а сразу несколько.

5. Вы работаете в бюро по трудоустройству. Вашей задачей является отслеживание финансовой стороны работы компании.
Деятельность Вашего бюро организована следующим образом: Ваше бюро готово искать работников для различных работодателей и вакансии для ищущих работу специалистов различного профиля.
При обращении к Вам клиента-работодателя, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. При обращении к Вам клиента-соискателя, его стандартные данные (фамилия, имя, отчество, квалификация, профессия, иные данные) также фиксируются в базе данных.
По каждому факту удовлетворения интересов обеих сторон составляется документ. В документе указываются соискатель, работодатель, должность и комиссионные (доход бюро).
6. Вы работаете в нотариальной конторе. Вашей задачей является отслеживание финансовой стороны работы компании.
Деятельность Вашей нотариальной конторы организована следующим образом: Ваша фирма готова предоставить клиенту определенный комплекс услуг.
Для наведения порядка Вы формализовали эти услуги, составив их список с описанием каждой услуги. При обращении к Вам клиента, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных.
По каждому факту оказания услуги клиенту составляется документ. В документе указываются услуга, сумма сделки, комиссионные (доход конторы), описание сделки.
В рамках одной сделки клиенту может быть оказано несколько услуг. Стоимость каждой услуги фиксирована.
7. Вы работаете в фирме, занимающейся продажей запасных частей для автомобилей. Вашей задачей является отслеживание финансовой стороны работы компании.
Основная часть деятельности, находящейся в Вашем ведении, связана с работой с поставщиками. Фирма имеет определенный набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков Вы приобретаете детали.
Каждая деталь наряду с названием характеризуется артикулом и ценой (считаем цену постоянной). Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей.
Выяснилось, что цена детали может меняться от поставки к поставке.

Поставщики заранее ставят Вас в известность о дате изменения цены и о его новом значении.

8. Вы работаете в учебном заведении и занимаетесь организацией курсов повышения квалификации.

В Вашем распоряжении имеются сведения о сформированных группах студентов. Группы формируются в зависимости от специальности и отделения. В каждой из них включено определенное количество студентов. Проведение занятий обеспечивает штат преподавателей. Для каждого из них в базе данных зарегистрированы стандартные анкетные данные (фамилия, имя, отчество, телефон) и стаж работы. В результате распределения нагрузки Вы получаете информацию о том, сколько часов занятий проводит каждый преподаватель с соответствующими группами. Кроме того, хранятся также сведения о виде проводимых занятий (лекции, практика), предмете и оплате за 1 час.

9. Вы являетесь руководителем информационно-аналитического центра коммерческого банка. Одним из существенных видов деятельности Вашего банка является выдача кредитов юридическим лицам. Вашей задачей является отслеживание динамики работы кредитного отдела. В зависимости от условий получения кредита, процентной ставки и срока возврата все кредитные операции делятся на несколько основных видов. Каждый из этих видов имеет свое название.

Кредит может получить юридическое лицо (клиент), при регистрации предоставивший следующие сведения: название, вид собственности, адрес, телефон, контактное лицо. Каждый факт выдачи кредита регистрируется банком, при этом фиксируются сумма кредита, клиент и дата выдачи.

10. Вы являетесь коммерческим директором театра, и в Ваши обязанности входит вся организационно-финансовая работа, связанная с привлечением актеров и заключением контрактов.

Вы поставили дело следующим образом: каждый год театр осуществляет постановку различных спектаклей. Каждый спектакль имеет определенный бюджет.

Для участия в конкретных постановках в определенных ролях Вы привлекаете актеров. С каждым из актеров Вы заключаете персональный контракт на определенную сумму. Каждый из актеров имеет некоторый стаж работы, некоторые из них удостоены различных наград и званий. В рамках одного спектакля на одну и ту же роль привлекается несколько актеров. Контракт определяет базовую зарплату актера, а по итогам реально отыгранных спектаклей актеру назначается премия.

11. Вы являетесь руководителем службы планирования платной поликлиники. Вашей задачей является отслеживание финансовых

показателей работы поликлиники.

В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются больные. Все больные проходят обязательную регистрацию, при которой в базу данных заносятся стандартные анкетные данные (фамилия, имя, отчество, год рождения). Каждый больной может обращаться в поликлинику несколько раз, нуждаясь в различной медицинской помощи. Все обращения больных фиксируются, при этом устанавливается диагноз, определяется стоимость лечения, запоминается дата обращения. При обращении в поликлинику пациент обследуется и проходит лечение у разных специалистов. Общая стоимость лечения зависит от стоимости тех консультаций и процедур, которые назначены пациенту.

12. Вы работаете в крупном торговом центре, сдающим в аренду коммерсантам свои торговые площади.

Вашей задачей является наведение порядка в финансовой стороне работы торгового центра.

Работы Вашего торгового центра построена следующим образом: в результате планирования Вы определили некоторое количество торговых точек в пределах Вашего здания, которые могут сдаваться в аренду.

Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. Со всех потенциальных клиентов Вы собираете стандартные данные (название, адрес, телефон, реквизиты, контактное лицо).

При появлении потенциального клиента Вы показываете ему имеющиеся свободные площади. При достижении соглашения Вы оформляете договор, фиксируя в базе данных торговую точку, клиента, период (срок) аренды.

13. Вы являетесь руководителем коммерческой службы телевизионной компании. Вашей задачей является отслеживание расчетов, связанных с прохождением рекламы в телеэфире.

Работа построена следующим образом: заказчики просят поместить свою рекламу в определенной передаче в определенный день. Каждый рекламный ролик имеет определенную продолжительность.

Для каждой организации-заказчика известны банковские реквизиты, телефон и контактное лицо для проведения переговоров. Передачи имеют определенный рейтинг. Стоимость минуты рекламы в каждой конкретной передаче известна (определяется коммерческой службой, исходя из рейтинга передачи и прочих соображений).

14. Вы являетесь сотрудником коммерческого отдела компании, продающей различные товары через Интернет. Вашей задачей является отслеживание финансовой составляющей работы компании.

Работа Вашей компании организована следующим образом: на Интернет-сайте компании представлены (выставлены на продажу) некоторые товары. Каждый из них имеет некоторое название, цену и единицу измерения (штуки, килограммы, литры).

Для проведения исследований и оптимизации работы магазина Вы пытаетесь собирать данные с Ваших клиентов. При этом для Вас определяющее значение имеют стандартные анкетные данные, а также телефон и адрес электронной почты для связи.

По каждому факту продажи Вы автоматически фиксируете клиента, товары, количество, дату продажи, дату доставки.

15. Вы работаете в парикмахерской. Ваша парикмахерская стрижет клиентов в соответствии с их пожеланиями и некоторым каталогом различных видов стрижки. Так, для каждой стрижки определены название, принадлежность полу (мужская, женская), стоимость работы.

Для наведения порядка Вы составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). После того, как закончена очередная работа, в кассе фиксируются стрижка, клиент и дата производства работ.

У Вашей парикмахерской появился филиал, и Вы хотели бы видеть, в том числе, и отдельную статистику по филиалам.

16. Вы работаете в химчистке. Ваша химчистка осуществляет прием у населения вещей для выведения пятен.

Для наведения порядка Вы составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). Все оказываемые Вами услуги подразделяются на виды, имеющие название, тип и стоимость, зависящую от сложности работ.

Работа с клиентом первоначально состоит в определении объема работ, вида услуги и, соответственно, ее стоимости. Если клиент согласен, он оставляет вещь (при этом фиксируется услуга, клиент и дата приема) и забирает ее после обработки (при этом фиксируется дата возврата).

У Вашей химчистки появился филиал, и Вы хотели бы видеть, в том числе, и отдельную статистику по филиалам.

Лабораторная работа №3 Двоичные и текстовые файлы

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать, читать, открывать файлы. Что же такое файл? Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Ну, теперь ясно, что под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, например **файл.txt**. В одной директории не могут находиться файлы с одинаковыми именами. Под

именем файла понимается не только его название, но и расширение, например: **file.txt** и **file.dat** — разные файлы, хоть и имеют одинаковые названия (первый является текстовым файлом, а второй двоичным). Существует такое понятие, как полное имя файлов – это полный адрес к директории файла с указанием имени файла, например: **D:\docs\file.txt**. Важно понимать эти базовые понятия, иначе сложно будет работать с файлами.

Различают два вида файлов: текстовые и двоичные.

Текстовый поток – это последовательность символов. Они организуются по строкам, каждая из которых заканчивается символом «конца строки». Конец самого файла обозначается символом «конца файла». При записи информации в текстовый файл, просмотреть который можно с помощью любого текстового редактора, все данные преобразуются к символьному типу и хранятся в символьном виде. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки). В текстовом режиме каждый разделительный символ строки автоматически преобразуется в пару (возврат каретки – переход на новую строку).

Двоичный поток – это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве. В двоичных файлах информация считывается и записывается в виде блоков определенного размера, в которых могут храниться данные любого вида и структуры.

Открытый файл представляется как последовательность считываемых или записываемых данных. При открытии файла с ним связывается поток ввода-вывода. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Основные операции, которые выполняются с файлами:

1. *Определение* переменной типа FILE.
2. *Открытие файл*, для того, чтобы к нему можно было обращаться. Соответственно, открывать файл можно для чтения, записи, чтения и записи, переписывания или записи в конец файла и т.п. При открытии файла могут возникнуть ошибки – файла может не существовать, это может быть файл не того типа, отсутствие прав на работу с файлом и т.д. Всё это необходимо учитывать.
3. *Работа с файлом* – запись и чтение. Здесь также нужно помнить, что операции выполняются не с памятью с произвольным доступом, а с буферизированным потоком (абстрактным логическим устройством), что добавляет свою специфику.
4. *Закрытие файл*. Так как файл является внешним по отношению к программе ресурсом, то если его не закрыть, то он продолжит находиться в памяти, возможно, даже после закрытия программы (например, нельзя будет удалить открытый файл или внести изменения и т.п.). Кроме того, иногда необходимо не закрывать, а «переоткрыть» файл для того, чтобы, например, изменить режим доступа.

Вся информация хранится в компьютере в виде 0 и 1, т. е. в двоичном виде. Двоичные файлы отличаются от текстовых только методами работы с ними. Например, если мы записываем в текстовый файл цифру «4», то она записывается как символ, и для ее хранения нужен один байт. Соответственно и размер файла будет равен одному байту. Текстовый файл, содержащий запись: «145687», будет иметь размер шесть байт.

Если же записать целое число 145 687 в двоичный файл, то он будет иметь размер четыре байта, так как именно столько необходимо для хранения данных типа `int`. То есть двоичные файлы более компактны и в некоторых случаях более удобны для обработки.

Для работы с файлами необходимо подключить заголовочный файл `<fstream>`. В `<fstream>` определены несколько классов и подключены заголовочные файлы `<ifstream>` — файловый ввод и `<ofstream>` — файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие — это то, что ввод/вывод выполняется не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов `cin` и `cout`, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам `cin` и `cout`.

Например, необходимо создать текстовый файл и записать в него строку **Работа с файлами в C++**. Для этого необходимо проделать следующие шаги:

1. создать объект класса `ofstream`;
2. связать объект класса с файлом, в который будет производиться запись;
3. записать строку в файл;
4. закрыть файл.


```
int main() {
    ofstream out("data.txt");
    if (!out.is_open()) {
        cout << "Ошибка: файл не открыт." << endl;
        return 1;
    }
    out << "Работа с файлами в C++" << endl;
    out.close();
    return 0;
}
```

Осталось проверить правильность работы программы, а для этого открываем файл `study.txt` и смотрим его содержимое, должно быть — [Работа с файлами в C++](#).

Для того чтобы прочитать файл понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями (пример кода приведен выше):

1. создать объект класса `ifstream` и связать его с файлом, из которого будет производиться считывание;
2. прочитать файл;
3. закрыть файл.

Результат выполнения программы:



```
Работа
с файлами в C++
```

В программе показаны два способа чтения из файла, первый — используя операцию передачи в поток, второй — используя функцию `getline()`. В первом случае считывается только первое слово, а во втором случае считывается строка, длиной 50 символов. Но так как в файле осталось меньше 50 символов, то считываются символы включительно до последнего. Обратите внимание на то, что считывание во второй раз продолжилось, после первого слова, а не с начала, так как первое слово было прочитано через `fin`.

Программа сработала правильно, но не всегда так бывает, даже в том случае, если с кодом всё в порядке. Например, в программу передано имя несуществующего файла или в имени допущена ошибка. Что тогда? В этом случае ничего не произойдёт вообще. Файл не будет найден, а значит и прочитать его не возможно. Поэтому компилятор проигнорирует строки, где выполняется работа с файлом. В результате корректно завершится работа программы, но ничего, на экране показано не будет. Казалось бы это вполне нормальная реакции на такую ситуацию. Но простому пользователю не будет понятно, в чём дело и почему на экране не появилась строка из файла. Так вот, чтобы всё было предельно понятно в C++ предусмотрена такая функция — `is_open()`, которая возвращает целые значения: 1 — если файл был успешно открыт, 0 — если файл открыт не был. Доработаем программу с открытием

файла, таким образом, что если файл не открыт выводилось соответствующее сообщение.

Результат программы (пример кода приведен ниже):

```
Файл не может быть открыт!
```

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");
    char buff[50];
    ifstream fin("study.doc"); // (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

    if (!fin.is_open()) // если файл не открыт
        cout << "Файл не может быть открыт!\n"; // сообщить об этом
    else
    {
        fin.getline(buff, 50);
        fin.close();
        cout << buff << endl;
    }
}
```

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе `ios_base` предусмотрены константы, которые определяют режим открытия файлов.

Константа	Описание
<code>ios_base::in</code>	открыть файл для чтения
<code>ios_base::out</code>	открыть файл для записи
<code>ios_base::ate</code>	при открытии переместить указатель в конец файла
<code>ios_base::app</code>	открыть файл для записи в конец файла
<code>ios_base::trunc</code>	удалить содержимое файла, если он существует
<code>ios_base::binary</code>	открытие файла в двоичном режиме

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции `open()`.

```
ofstream fout("study.txt", ios_base::app); // открываем файл для добавления
информации к концу файла
fout.open("study.txt", ios_base::app); // открываем файл для добавления
информации к концу файла
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции **или** `|`, например: `ios_base::out | ios_base::trunc` — открытие файла для записи, предварительно очистив его.

Объекты класса `ofstream`, при связке с файлами по умолчанию содержат режимы открытия файлов `ios_base::out | ios_base::trunc`. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи. Объекты класса `ifstream` связываясь с файлом, имеют по умолчанию режим открытия файла `ios_base::in` — файл открыт только для чтения. Режим открытия файла ещё называют — флаг, для удобочитаемости в дальнейшем будем использовать именно этот термин. В таблице 1 перечислены далеко не все флаги, но для начала этих должно хватить.

Обратите внимание на то, что флаги `ate` и `app` по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг `app` позволяет производить запись, только в конец файла, а флаг `ate` просто переставляет флаг в конец файла и не ограничивает места записи.

Путь к файлу можно задать тремя способами:

1. "file.txt" — открывается файл file.txt из текущей папки;

Замечание: при запуске exe-файла «текущая папка» — та, где он находится; при компиляции проекта через IDE Visual Studio «текущая папка» — та, находится файл `cpp` проекта.

2. "C://study/file.txt" — открывается файл file.txt из папки study находящейся на диске C.
3. С помощью переменной типа `string`.

```
string fileName = "study.txt";  
ofstream fout(fileName);
```

Для того, чтобы удалить файл, нужно использовать функцию `remove("название файла")`:

```
remove("study.txt");
```

Задание к лабораторной работе №3:

Написать функции:

1. Создание и запись данных в файл, название которого вводится с клавиатуры;
2. Чтение файла (название файла не нужно вводить с клавиатуры);
3. Удаление содержимого в файле;
4. Удаление файла.

Лабораторная работа №4 Индексирование записей. Простой/сложный индекс

К структурам данных в C++ применимы следующие операции:

Название операции	Знак операции
выбор элемента через имя (селектор)	. (точка)
выбор элемента через указатель (селектор)	-> (минус и знак больше)
присваивание	=
взятие адреса	&

Операция создания заключается в выделении памяти для структуры данных. Память может выделяться в процессе выполнения программы при первом появлении имени переменной в исходной программе или на этапе компиляции. В ряде языков (например, в C) для структурированных данных, конструируемых программистом, операция создания включает в себя также установку начальных значений параметров создаваемой структуры.

Например: в результате описания типа

int I;

char C;

будет выделена память для соответствующих переменных. 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. Программист может и сам выделять память для структур данных, используя имеющиеся в системе программирования процедуры и функции для выделения и освобождения памяти. В объектно-ориентированных языках программирования при разработке нового объекта для него должны быть определены функции его создания и уничтожения.

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.

Операция уничтожения структур данных противоположна по своему действию операции создания. Некоторые языки, такие как BASIC, FORTRAN, не дают возможности программисту уничтожать созданные структуры данных. В языках PL/1, C, PASCAL структуры данных, имеющиеся внутри блока, уничтожаются в процессе выполнения программы при выходе из этого блока. Операция уничтожения помогает эффективно использовать память.

Операция выбора используется программистами для доступа к данным внутри самой структуры. Форма операции доступа зависит от типа структуры данных, к которой осуществляется обращение. Метод доступа - один из наиболее важных свойств структур, особенно в связи с тем, что это свойство имеет непосредственное отношение к выбору конкретной структуры данных.

Операция обновления позволяет изменить значения данных в структуре данных. Примером операции обновления является операция присваивания или более сложная форма - передача параметров.

Вышеуказанные четыре операции обязательны для всех структур и типов данных. Помимо этих общих операций для каждой структуры данных могут быть определены операции специфические, работающие только с данными указанного типа (данной структуры). Специфические операции применяются при рассмотрении каждой конкретной структуры данных.

1.5. Структурность данных и технология программирования

Знание структуры данных позволяет организовать их хранение и обработку максимально эффективным образом - с точки зрения минимизации затрат как памяти, так и процессорного времени. Другим, не менее, а может быть и более, важным преимуществом, которое обеспечивается структурным подходом к данным, является возможность структурирования сложного программного изделия. Современные промышленно выпускаемые программные пакеты - изделия чрезвычайно сложные, объем их исчисляется тысячами и миллионами строк кода, а трудоемкость разработки - сотнями человеко-лет. Естественно, что разработать такое программное изделие "все сразу" невозможно, оно должно быть представлено в виде какой-то структуры - составных частей и связей между ними. Правильное структурирование изделия дает возможность на каждом этапе разработки сосредоточить внимание разработчика на одной обозримой части изделия или поручить реализацию разных его частей разным исполнителям.

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).

Другой подход к структуризации основывается на данных. Программисту, который хочет, чтобы его программа имела реальное применение в некоторой прикладной области, не следует забывать о том, что программирование - это обработка данных. В программах можно изобретать сколь угодно замысловатые и изощренные алгоритмы, но у реального программного изделия всегда есть Заказчик. У Заказчика есть входные данные, и он хочет, чтобы по ним были получены выходные данные, а какими средствами это обеспечивается - его не интересует. Таким образом, задачей любого программного изделия является преобразование входных данных в выходные. Инструментальные средства программирования предоставляют набор базовых (простых, примитивных) типов данных и операции над ними. Интегрируя базовые типы, создаются более сложные типы данных и определяются новые операции над сложными типами. Можно здесь провести аналогию со строительными работами: базовые типы - "кирпичики", из которых создаются сложные типы - "строительные блоки". Полученные на первом шаге композиции "строительные блоки" используются в качестве базового набора для следующего шага, результатом которого будут еще более сложные конструкции данных и еще более мощные операции над ними и т.д. В идеале последний шаг композиции дает типы данных, соответствующие входным и выходным данным задачи, а операции над этими типами реализуют в полном объеме задачу проекта.

Программисты, поверхностно понимающие структурное программирование, часто противопоставляют нисходящее проектирование восходящему, придерживаясь одного выбранного ими подхода. Реализация любого реального проекта всегда ведется встречными путями, причем, с постоянной коррекцией структур алгоритмов по результатам разработки структур данных и наоборот.

Еще одним чрезвычайно продуктивным технологическим приемом, связанным со структуризацией данных, является инкапсуляция. Смысл ее состоит в том, что сконструированный 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 данных представляется для него в виде "черного ящика", для которого известны входы 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.

Современные языки программирования блочного типа (PASCAL, C) обладают достаточно развитыми возможностями построения программ с модульной структурой и управления доступом модулей к данным и процедурам. Расширения же языков дополнительными возможностями конструирования типов и их инкапсуляции делают язык объектно-ориентированным. Сконструированные и полностью закрытые типы данных представляют собой объекты, а функции, работающие с их внутренней структурой, - методы работы с объектами. При этом в значительной степени меняется и сама концепция программирования. Программист, оперирующий объектами, указывает в программе ЧТО нужно сделать с объектом, а не КАК это надо делать.

+Технология баз данных развивалась параллельно с технологией языков программирования и не всегда согласованно с ней. Отчасти этим, а отчасти и объективными различиями в природе задач, решаемых системами управления базами данных (СУБД) и системами программирования, вызваны некоторые терминологические и понятийные различия в подходе к данным в этих двух сферах. Ключевым понятием в СУБД является понятие модели данных, в основном тождественное понятию логической структуры данных. Отметим, что физическая структура данных в СУБД не рассматривается вообще. Но сами СУБД являются программными пакетами, выполняющими отображение физической структуры в логическую (в модель данных). Для реализации этих пакетов используются те или иные системы программирования, разработчики СУБД, следовательно, имеют дело со структурами данных в терминах систем программирования. Для пользователя же внутренняя структура СУБД и физическая структура данных совершенно прозрачна; он имеет дело только с моделью данных и с другими понятиями логического уровня.

Массивы структур

Структуры часто образуют массивы. Чтобы объявить массив структур, вначале необходимо определить структуру (то есть определить агрегатный тип данных), а затем объявить переменную массива этого же типа. Например, чтобы объявить

100-элементный массив структур типа `addr`, который был определен ранее, напишите следующее:

```
struct addr addr_list[100];
```

Это выражение создаст 100 наборов переменных, каждый из которых организован так, как определено в структуре `addr`.

Чтобы получить доступ к определенной структуре, указывайте имя массива с индексом. Например, чтобы вывести ZIP-код из третьей структуры, напишите следующее:

```
printf("%d", addr_list[2].zip);
```

Как и в других массивах переменных, в массивах структур индексирование начинается с 0.

Для справки: чтобы указать определенную структуру, находящуюся в массиве структур, необходимо указать имя этого массива с определенным индексом. А если нужно указать индекс определенного элемента в структуре, то необходимо указать индекс этого элемента. Таким образом, в результате выполнения следующего выражения первому символу члена `name`, находящегося в третьей структуре из `addr_list`, присваивается значение 'X'.

```
addr_list[2].name[0] = 'X';
```

Пример со списком рассылки

Чтобы показать, как используются структуры и массивы структур, в этом разделе создается простая программа работы со списком рассылки, и в ее массиве структур будут храниться адреса и связанная с ними информация. Эта информация записывается в следующие поля: `name` (имя), `street` (улица), `city` (город), `state` (штат) и `zip` (почтовый код, индекс).

Вся эта информация, как показано ниже, находится в массиве структур типа `addr`:

```
struct addr {  
    char name[30];  
    char street[40];  
    char city[20];  
    char state[3];  
    unsigned long int zip;  
} addr_list[MAX];
```

Обратите внимание, что поле `zip` имеет целый тип `unsigned long`. Правда, чаще можно встретить хранение почтовых кодов, в которых используются строки символов, потому что этот способ подходит для почтовых кодов, в которых вместе с цифрами используются и буквы (как, например, в Канаде и других странах). Однако в нашем примере почтовый индекс хранится в виде целого числа; это делается для того, чтобы показать использование числового элемента в структуре.

Вот `main()` — первая функция, которая нужна программе:

```

int main(void)
{
    char choice;

    init_list(); /* инициализация массива структур */

    for (;;) {
        choice = menu_select();
        switch (choice) {
            case 1: enter();
                break;
            case 2: delete();
                break;
            case 3: list();
                break;
            case 4: exit(0);
        }
    }

    return 0;
}

```

Функция начинает выполнение с инициализации массива структур, а затем реагирует на выбранный пользователем пункт меню.

Функция init_list() готовит массив структур к использованию, обнуляя первый байт поля name каждой структуры массива. (В программе предполагается, что если поле name пустое, то элемент массива не используется.) А вот сама функция init_list():

```

/* Инициализация списка. */
void init_list(void)
{
    register int t;

    for (t = 0; t < MAX; ++t) addr_list[t].name[0] = '\0';
}

```

Функция menu_select() выводит меню на экран и возвращает то, что выбрал пользователь.

```

/* Получения значения, выбранного в меню. */
int menu_select(void)
{
    char s[80];
    int c;

    printf("1. Введите имя\n");
    printf("2. Удалите имя\n");
    printf("3. Выведите список\n");
    printf("4. Выход\n");

    do {

```

```

        printf("\nВведите номер нужного пункта: ");
        gets(s);
        c = atoi(s);
    } while (c < 0 || c > 4);

    return c;
}

```

Функция enter() подсказывает пользователю, что именно требуется ввести, и сохраняет введенную информацию в следующей свободной структуре. Если массив заполнен, то выводится сообщение Список заполнен. Функция find_free() ищет в массиве структур свободный элемент.

```

/* Ввод адреса в список. */
void enter(void)
{
    int slot;
    char s[80];

    slot = find_free();
    if (slot == -1) {
        printf("\nСписок заполнен");
        return;
    }

    printf("Введите имя: ");
    gets(addr_list[slot].name);

    printf("Введите улицу: ");
    gets(addr_list[slot].street);

    printf("Введите город: ");
    gets(addr_list[slot].city);

    printf("Введите штат: ");
    gets(addr_list[slot].state);

    printf("Введите почтовый код: ");
    gets(s);
    addr_list[slot].zip = strtoul(s, '0', 10);
}

/* Поиск свободной структуры. */
int find_free(void)
{
    register int t;

    for (t = 0; addr_list[t].name[0] && t < MAX; ++t);

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

```

```
}
```

Обратите внимание, что если все элементы массива структур заняты, то `find_free()` возвращает -1. Это удобное число, потому что в массиве нет -1-го элемента.

Функция `delete()` предлагает пользователю указать индекс той записи с адресом, которую требуется удалить. Затем функция обнуляет первый байт поля `name`.

```
/* Удаление адреса. */
void delete(void)
{
    register int slot;
    char s[80];

    printf("Введите № записи: ");
    gets(s);
    slot = atoi(s);
    if (slot >= 0 && slot < MAX)
        addr_list[slot].name[0] = '\0';
}
```

И последняя функция, которая требуется программе, — это `list()`, которая выводит на экран весь список рассылки. Из-за большого разнообразия компьютерных сред язык C не определяет стандартную функцию, которая бы отправляла вывод на принтер. Однако все нужные для этого средства имеются во всех компиляторах C. Возможно, вам самим захочется сделать так, чтобы программа работы со списками могла еще и распечатывать список рассылки.

```
/* Вывод списка на экран. */
void list(void)
{
    register int t;

    for (t = 0; t < MAX; ++t) {
        if (addr_list[t].name[0]) {
            printf("%s\n", addr_list[t].name);
            printf("%s\n", addr_list[t].street);
            printf("%s\n", addr_list[t].city);
            printf("%s\n", addr_list[t].state);
            printf("%lu\n\n", addr_list[t].zip);
        }
    }
    printf("\n\n");
}
```

Указатели на структуры

На структуры, как и на объекты других типов, можно определять указатели.

Например, указатель на структуру `person` :

```
struct person* p;
```

Указатели на структуры можно создавать и для безымянных структурных типов :

```
struct
{
    int age;
    char name[20];
} *p1, *p2;
```

В качестве значения такому указателю присваивается адрес объекта структуры того же типа :

```
struct person kate = { 31, "Kate" };
struct person* p_kate = &kate;
```

Используя указатель на структуру, можно получить доступ к ее элементам. Для этого можно воспользоваться двумя способами. Первый способ представляет применение операции разыменования :

```
(*указатель_на_структуру).имя_элемента
```

Второй способ предполагает использование операции -> (операция стрелка) :
указатель_на_структуру->имя_элемента

Используем оба этих способа для обращения к элементам структуры :

```
#include <stdio.h>

struct person
{
    int age;
    char name[20];
};

int main(void)
{
    struct person kate = { 31, "Kate" };
    struct person* p_kate = &kate;

    char* name = p_kate->name;
    int age = (*p_kate).age;

    printf("name = %s \t age = %d \n", name, age);

    // изменим элемент age в структуре
    p_kate->age = 32;
    printf("name = %s \t age = %d \n", kate.name, kate.age);
    return 0;
}
```

Здесь определяется указатель `p_kate` на переменную `kate`. И используя указатель, мы можем получить или изменить значения элементов структуры.

Задание для лабораторной работы №6:

Для вашей БД (из лабораторной работы №2) создать индексы для каждой структуры и линейный поиск данных по индексу (при вводе индекса, выводило всю структуру).

Лабораторная работа №5 Запись/чтение массива структур в файл

Структуры часто образуют массивы. Чтобы объявить массив структур, вначале необходимо определить структуру (то есть определить агрегатный тип данных), а затем объявить переменную массива этого же типа. Для начала рассмотрим как объявить структуру:

Поскольку структуры определяются программистом, то вначале мы должны сообщить компилятору, как она вообще будет выглядеть. Для этого используется **ключевое слово** `struct`:

```
struct Employee
{
    short id;
    int age;
    double salary;
};
```

Мы определили структуру с именем `Employee`. Она содержит 3 переменные:

`id` типа `short`;

`age` типа `int`;

`salary` типа `double`.

Эти переменные, которые являются частью структуры, называются **членами структуры** (или **«полями структуры»**). `Employee` — это простое объявление структуры. Хотя мы и указали компилятору, что она имеет переменные-члены, память под нее сейчас не выделяется. Имена структур принято писать с заглавной буквы, чтобы отличать их от имен переменных.

Предупреждение: Одна из самых простых ошибок в C++ — забыть точку с запятой в конце объявления структуры. Это приведет к ошибке компиляции в

следующей строке кода. Современные компиляторы, такие как Visual Studio версии 2010, а также более новых версий, укажут вам, что вы забыли точку с запятой в конце, но более старые компиляторы могут этого и не сделать, из-за чего такую ошибку будет трудно найти.

Чтобы использовать структуру `Employee`, нам нужно просто объявить переменную типа `Employee`:

```
Employee john; // имя структуры Employee начинается с заглавной буквы, а  
переменная john - с маленькой
```

Здесь мы определили переменную типа `Employee` с именем `john`. Как и в случае с обычными переменными, определение переменной, типом которой является структура, приведет к выделению памяти для этой переменной.

Объявить можно и несколько переменных одной структуры:

```
Employee john; // создаем отдельную структуру Employee для John  
Employee james; // создаем отдельную структуру Employee для James
```

Доступ к членам структур

Когда мы объявляем переменную структуры, например, `Employee john`, то `john` ссылается на всю структуру. Для того, чтобы получить доступ к отдельным её членам, используется **оператор выбора члена** (`.`). Например, в коде, приведенном ниже, мы используем оператор выбора членов для инициализации каждого члена структуры:

```
Employee john; // создаем отдельную структуру Employee для John  
    john.id = 8; // присваиваем значение члену id структуры john  
    john.age = 27; // присваиваем значение члену age структуры john  
    john.salary = 32.17; // присваиваем значение члену salary структуры john  
  
Employee james; // создаем отдельную структуру Employee для James  
    james.id = 9; // присваиваем значение члену id структуры james  
    james.age = 30; // присваиваем значение члену age структуры james  
    james.salary = 28.35; // присваиваем значение члену salary структуры james
```

Как и в случае с обычными переменными, переменные-члены структуры не инициализируются автоматически и обычно содержат мусор. Инициализировать их нужно вручную.

В примере, приведенном выше, легко определить, какая переменная относится к структуре `John`, а какая — к структуре `James`. Это обеспечивает гораздо более

высокий уровень организации, чем в случае с обычными отдельными переменными.

Инициализация структур

Инициализация структур путем присваивания значений каждому члену по порядку — занятие довольно громоздкое (особенно, если этих членов много), поэтому в языке C++ есть более быстрый способ инициализации структур — с помощью **списка инициализаторов**. Он позволяет инициализировать некоторые или все члены структуры во время объявления переменной типа struct:

```
struct Employee
{
    short id;
    int age;
    double salary;
};

Employee john = { 5, 27, 45000.0 }; // john.id = 5, john.age = 27,
john.salary = 45000.0
Employee james = { 6, 29 }; // james.id = 6, james.age = 29, james.salary =
0.0 (инициализация по умолчанию)
```

Если в списке инициализаторов не будет одного или нескольких элементов, то им присвоятся значения по умолчанию (обычно, 0). В примере, приведенном выше, члену james.salary присваивается значение по умолчанию 0.0, так как мы сами не предоставили никакого значения во время инициализации.

Вложенные структуры

Одни структуры могут содержать другие структуры. Например:

```
struct Employee
{
    short id;
    int age;
    double salary;
};

struct Company
{
    Employee CEO; // Employee - это структура внутри структуры Company
    int numberOfEmployees;
};

Company myCompany;
```

В этом случае, если бы мы хотели узнать, какая зарплата у CEO (исполнительного директора), то нам бы пришлось использовать оператор выбора членов дважды:

```
myCompany.CEO.salary = 2000;
```

Сначала мы выбираем поле CEO из структуры myCompany, а затем поле salary из структуры Employee.

Вы можете использовать вложенные списки инициализаторов с вложенными структурами:

```
Company myCompany = { { 3, 35, 55000.0f }, 7 };
```

Теперь рассмотрим массив структур: структуры часто образуют массивы. Чтобы объявить массив структур, вначале необходимо определить структуру (то есть определить агрегатный тип данных), а затем объявить переменную массива этого же типа.

Например, чтобы объявить 100-элементный массив структур типа addr, который был определен ранее, напишите следующее:

```
Company myCompany[100];
```

Это выражение создаст 100 наборов переменных, каждый из которых организован так, как определено в структуре Company.

Чтобы получить доступ к определенной структуре, указывайте имя массива с индексом. Например, чтобы вывести зарплату из третьей структуры, напишите следующее:

```
cout << myCompany[2].CEO.salary;
```

Как и в других массивах переменных, в массивах структур индексирование начинается с 0.

Для того, чтобы записать массив структур в файл или же считать его с файла, используем библиотеку fstream из Лабораторной работы №3. Пример:

```
#include <iostream>
#include <fstream>
#include <string>
```

```
using namespace std;
```

```
// Создаем структуру из элементов типа string
struct Initial {
    string surname,
        name,
```

```

        patronymic;
};

void ReadingData(Initial* (&d), int& n, string fileName)
{
    //создаём поток для чтения
    ifstream reading(fileName);

    // условие: если файл открылся
    if (reading) {
        reading >> n;

        d = new Initial[n];

        for (int i = 0; i < n; i++) {
            reading >> d[i].surname;
            reading >> d[i].name;
            reading >> d[i].patronymic;
        }
        cout << "Данные считаны!" << endl;
    }
    else
        cout << "Ошибка открытия файла!" << endl;
    reading.close();
}

void SavingData(Initial* d, int n, string fileName)
{
    // создается поток для записи
    // открывает fileName и делает так, чтобы он был пустой
    ofstream record(fileName, ios::out);

    // условие: если файл открылся
    if (record) {
        record << n << endl;

        for (int i = 0; i < n; i++) {
            record << d[i].surname << endl;
            record << d[i].name << endl;
            record << d[i].patronymic << endl;
        }
        cout << "Данные записаны!" << endl;
    }
    else
        cout << "Ошибка открытия файла!" << endl;

    record.close();
}

```

```

int main()
{
    SetConsoleCP(1251); // установка кодовой страницы win-cp 1251 в поток ввода
    SetConsoleOutputCP(1251); // установка кодов страницы win-cp 1251 в поток вывода
    int size = 0;
    // Массив структур
    Initial* arr = new Initial[size];
    ReadingData(arr, size, "Input.txt");
    SavingData(arr, size, "Output.txt");
}

```

Задание к лабораторной работе:

В соответствии со своими заданиями из лабораторной работы №2, составить массив структур для своей БД и реализовать функции считывания с файла и вывод структуры на экран, записи в файл и добавления данных в массив. Для функций считывание и записи должен быть выбор (в виде меню), в котором пользователь может сам ввести название файла с клавиатуры либо использовать уже заготовленный файл.

Лабораторная работа №6 Редактирование файлов: удаление, изменение поля

Наиболее понятным и эффективным способом удаления и редактирования данных для баз данных в С++ можно считать следующие функции:

Функция выборочного удаления:

```

#include <iostream>
#include <windows.h>
#include <string>

using namespace std;

```

```

struct Initial {
    string surname,
        name,
        patronymic;
};

struct Date {
    int day,
        month,
        year;
};

struct Data {
    Initial _initial;
    Date _date;
};

void Copy(Data* (&d_n), Data* (&d_o), int n)
{
    for (int i = 0; i < n; i++) {
        d_n[i] = d_o[i];
    }
}

void DeleteData(Data* (&d), int& n)
{
    int _n;
    cout << "Введите номер элемента (от 1 до " << n << "): ";
    cin >> _n;
    _n--;
    system("cls");

    if (_n >= 0 && _n < n) {
        // временный массив
        Data* buf = new Data[n];

        Copy(buf, d, n);

        // выделяем новую память
        --n;
        d = new Data[n];
        int q = 0;

        // заполняем неудаленные данные
        for (int i = 0; i <= n; i++) {
            if (i != _n) {
                d[q] = buf[i];
                ++q;
            }
        }
    }
}

```

```

    }

    system("cls");
    delete[] buf;
    cout << "Данные удалены!" << endl;
}
else
    cout << "Номер введён неверно!" << endl;
}

int main()
{
    setlocale(LC_ALL, "rus");
    // переменная хранящая кол-во структур массива
    int amountOfData = 0;

    // массив данных
    Data* d = new Data[amountOfData];

    //ввод из файла
    ReadingData(d, amountOfData, "Input.txt");

    if (amountOfData != 0) {
        DeleteData(d, amountOfData);
    }
    else
        cout << "Данные пусты!" << endl;
}

```

В данном и последующих примерах данные вводятся через функцию ReadingData() (из лабораторной работы №5) из файла Input.txt.

Функция полного удаления:

Для того, чтобы полностью удалить данные нам не нужна новая функция, для этого достаточно в main очистить динамический массив с помощью delete. И создать новый динамический массив.

```

amountOfData = 0;
delete[] d;
d = new Data[amountOfData];

```

Функция изменения:

```

#include <iostream>
#include <windows.h>
#include <string>

```

```

using namespace std;

struct Initial {
    string surname,
        name,
        patronymic;
};

struct Date {
    int day,
        month,
        year;
};

struct Data {
    Initial _initial;
    Date _date;
};

void Copy(Data* (&d_n), Data* (&d_o), int n)
{
    for (int i = 0; i < n; i++) {
        d_n[i] = d_o[i];
    }
}

void DataChange(Data* (&d), int n)
{
    int _n;
    cout << "Введите номер элемента (от 1 до " << n << "): ";
    cin >> _n;
    _n--;
    system("cls");

    // проверка, что ввели правильное значение
    if (_n >= 0 && _n < n) {
        cout << "Введите ФИО: ";
        cin >> d[_n]._initial.surname;
        cin >> d[_n]._initial.name;
        cin >> d[_n]._initial.patronymic;

        cout << "Введите дату: ";
        cin >> d[_n]._date.day;
        cin >> d[_n]._date.month;
        cin >> d[_n]._date.year;

        system("cls");

        cout << "Данные изменены!" << endl;
    }
}

```



```

    }
    else
        cout << "Номер введён неверно!" << endl;
}

int main()
{
    setlocale(LC_ALL, "rus");
    // переменная хранящая кол-во структур массива
    int amountOfData = 0;

    // массив данных
    Data* d = new Data[amountOfData];

    //ввод из файла
    ReadingData(d, amountOfData, "Input.txt");

    if (amountOfData != 0) {
        DataChange(d, amountOfData);
    }
    else
        cout << "Данные пусты!" << endl;
}

```

Задание для лабораторной работы №6:

Написать функции удаления (выборочное и полное) и изменения массива структур для вашей БД (из лабораторной работы №2).

Лабораторная работа №7 Сортировка записей

Было подсчитано, что до четверти времени централизованных компьютеров уделяется сортировке данных. Это потому, что намного легче найти значение в массиве, который был заранее отсортирован. В противном случае поиск немного похожит на поиск иголки в стоге сена.

Есть программисты, которые всё рабочее время проводят в изучении и внедрении алгоритмов сортировки. Это потому, что подавляющее большинство программ в бизнесе включает в себя управление базами данных. Люди ищут информацию в базах данных всё время. Это означает, что поисковые алгоритмы очень востребованы.

В то время как компьютеры находятся без пользователей в некоторые моменты времени, алгоритмы сортировки продолжают работать с базами данных. Снова приходят пользователи, осуществляющие поиск, а база данных уже отсортирована, исходя из той или иной цели поиска.

Рассмотрим несколько базовых реализаций сортировок.

Сортировка выбором (Selection sort)

Для того, чтобы отсортировать массив в порядке возрастания, следует на каждой итерации найти элемент с наибольшим значением. С ним нужно поменять местами последний элемент. Следующий элемент с наибольшим значением становится уже на предпоследнее место. Так должно происходить, пока элементы, находящиеся на первых местах в массиве, не окажутся в надлежащем порядке.

```
void SelectionSort(int* a, int n)
{
    int smallest_id;

    for (int i = 0; i < n; i++) {
        smallest_id = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[smallest_id])
                smallest_id = j;
        }
        //меняем местами элементы
        swap(a[smallest_id], a[i]);
    }
}
```

Пузырьковая сортировка (Bubble sort)

При пузырьковой сортировке сравниваются соседние элементы и меняются местами, если следующий элемент меньше предыдущего. Требуется несколько проходов по данным. Во время первого прохода сравниваются первые два элемента в массиве. Если они не в порядке, они меняются местами и затем сравниваются элементы в следующей паре. При том же условии они так же меняются местами. Таким образом сортировка происходит в каждом цикле пока не будет достигнут конец массива.

```
void BubbleSort(int* a, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (a[i] > a[j]) {
                //меняем местами элементы
                swap(a[i], a[j]);
            }
        }
    }
}
```

Сортировка вставками (Insertion sort)

При сортировке вставками массив разбивается на две области: упорядоченную и неупорядоченную. Изначально весь массив является неупорядоченной областью. При первом проходе первый элемент из неупорядоченной области изымается и помещается в правильное положение в упорядоченной области.

На каждом проходе размер упорядоченной области возрастает на 1, а размер неупорядоченной области сокращается на 1.

Основной цикл работает в интервале от 1 до N-1. На j-й итерации элемент [i] вставлен в правильное положение в упорядоченной области. Это сделано путем сдвига всех элементов упорядоченной области, которые больше, чем [i], на одну позицию вправо. [i] вставляется в интервал между теми элементами, которые меньше [i], и теми, которые больше [i].

```
void insertionSort(int data[], int lenD)
{
    int key = 0;
    int i = 0;
    for(int j = 1; j < lenD; j++){
        key = data[j];
        i = j-1;
        while(i >= 0 && data[i] > key){
            data[i+1] = data[i];
            i = i-1;
            data[i+1] = key;
        }
    }
}
```

Сортировка слиянием (Merge sort)

При рекурсивной сортировке слиянием массив сначала разбивается на мелкие кусочки - на первом этапе - на состоящие из одного элемента. Затем эти кусочки объединяются в более крупные кусочки - по два элемента и элементы при этом сравниваются, а в результате в новом кусочке меньший элемент занимает место слева, а больший - справа. Далее происходит слияние в ещё более крупные кусочки и так до конца алгоритма, когда все кусочки будут объединены в один, уже отсортированный массив.

```
void MergeSort(int* a, int left, int right)
{
    if (left < right)
    {
        int mid = (left + right) / 2;
        MergeSort(a, left, mid);
        MergeSort(a, mid + 1, right);
        Merge(a, left, mid, right);
    }
}

void Merge(int* a, int left, int mid, int right)
{
    int i = left, j = mid + 1, k = 0;
    int* temp = new int[right - left + 1];
    while (i <= mid && j <= right)
    {
        if (a[i] < a[j]) temp[k++] = a[i++];
        else temp[k++] = a[j++];
    }
    while (i <= mid) temp[k++] = a[i++];
    while (j <= right) temp[k++] = a[j++];
    for (i = left; i <= right; i++) a[i] = temp[i - left];
    delete[] temp;
}
```

Быстрая сортировка (Quick sort)

Быстрая сортировка использует алгоритм "разделяй и властвуй". Она начинается с разбиения исходного массива на две области. Эти части находятся слева и справа от отмеченного элемента, называемого опорным. В конце процесса одна часть будет содержать элементы меньшие, чем опорный, а другая часть будет содержать элементы больше опорного.

```
void QuickSort(int* a, int left, int right)
{
    if (left < right)
    {
        int l_hold = left, // левая граница
            r_hold = right, // правая граница
            pivot = a[left]; // разрешающий элемент
        while (left < right) // пока границы не сомкнутся
        {
            while (a[left] < pivot) left++;
            while (a[right] > pivot) right--;
            if (left < right) swap(a[left], a[right]);
        }
        swap(a[l_hold], a[right]);
        QuickSort(a, left, right - 1);
        QuickSort(a, left + 1, right);
    }
}
```

Задания к лабораторной работе №7:

В соответствии со своей БД из лабораторной работы №2, реализовать сортировку по каким-либо параметрам (сортировку можно использовать любую, однако параметров должно быть не менее 2).

Дополнительные задания к лабораторной работе №7:

Дополнительно нужно реализовать одну из сортировок в этом пункте, если вы ее не использовали в основном задании.

1. При помощи пузырьковой сортировки реализовать программу, которая отсортирует данные одного типа по возрастанию.
2. При помощи пузырьковой сортировки реализовать программу, которая отсортирует данные одного типа по убыванию.
3. При помощи сортировки вставками реализовать программу, которая отсортирует данные одного типа по возрастанию.
4. При помощи сортировки вставками реализовать программу, которая отсортирует данные одного типа по убыванию.
5. При помощи сортировки выбором реализовать программу, которая отсортирует данные одного типа по возрастанию.
6. При помощи сортировки выбором реализовать программу, которая отсортирует данные одного типа по убыванию.
7. При помощи сортировки слиянием реализовать программу, которая отсортирует данные одного типа по возрастанию.
8. При помощи сортировки слиянием реализовать программу, которая отсортирует данные одного типа по убыванию.
9. При помощи быстрой сортировки реализовать программу, которая отсортирует данные одного типа по возрастанию.
10. При помощи быстрой сортировки реализовать программу, которая отсортирует данные одного типа по убыванию.

Лабораторная работа №8 Фильтрация данных.

Далее мы разберём на примерах, как может выглядеть алгоритм поиска подстроки в строке. Примеры будут основываться на функциях стандартных библиотек, ведь именно в таких функциях и проявляются все удобства написания программ. А вот классический разбор алгоритма, основанный на циклах и сравнениях, также достаточно примечателен. Поэтому мы его рассмотрим в этом же уроке.

Сам алгоритм в принципе очень прост. Есть две строки. Например "Hello world" и "lo"

Работать будем в два цикла:

1. Первый будет выполнять проход по всей строке, и искать местоположение **первой буквы** искомой строки ("lo").
2. Второй, начиная с найденной позиции первой буквы – сверять, какие буквы стоят после неё и сколько из них подряд совпадают.

Проиллюстрируем поиск подстроки в строке:

1-й шаг цикла	Н	Е	Л	Л	О		W	O	R	L	D
	Л										
2-й шаг цикла	Н	Е	Л	Л	О		W	O	R	L	D
		Л									
3-й шаг цикла	Н	Е	Л	Л	О		W	O	R	L	D
			Л	О							
4-й шаг цикла	Н	Е	Л	Л	О		W	O	R	L	D
				Л	О						

На первых двух итерациях цикла сравниваемые буквы не будут совпадать (выделено красным). На третьей итерации искомая буква (первый символ

искомого слова) совпала с символом в строке, где происходит поиск. При таком совпадении в работу включается второй цикл.

Он призван отсчитывать количество символов после первого в искомой строке, которые будут совпадать с символами в строке исходной. Если один из следующих символов не совпадает – цикл завершает свою работу. Нет смысла гонять цикл впустую, после первого несовпадения, так как уже понятно, что искомого тут нет.

На третьей итерации совпал только первый символ искомой строки, а вот второй уже не совпадает. Придется первому циклу продолжить работу. Четвертая итерация дает необходимые результаты – совпадают все символы искомой строки с частью исходной строки. А раз все символы совпали – подстрока найдена. Работу алгоритма можно закончить.

Посмотрим, как выглядит классический код поиска подстроки в строке в C++:

```
#include <iostream>
// Функция для поиска подстроки в строке
// + поиск позиции, с которой начинается подстрока
int pos(char* s, char* c, int n)
{
    int i, j;          // Счетчики для циклов
    int lenC, lenS;    // Длины строк
    //Находим размеры строки исходника и искомого
    for (lenC = 0; c[lenC]; lenC++);
    for (lenS = 0; s[lenS]; lenS++);
    for (i = 0; i <= lenS - lenC; i++) // Пока есть возможность поиска
    {
        for (j = 0; s[i + j] == c[j]; j++); // Проверяем совпадение посимвольно
        // Если посимвольно совпадает по длине искомого
        // Вернем из функции номер ячейки, откуда начинается совпадение
        // Учитывать 0-терминатор ( '\0' )
        if (j - lenC == 1 && i == lenS - lenC && !(n - 1))
            return i;
        if (j == lenC)
            if (n - 1)
                n--;
            else
                return i;
    }
    //Иначе вернем -1 как результат отсутствия подстроки
    return -1;
}

int main()
{
    char* s = "parapapa";
    char* c = "pa";

    int i, n = 0;

    for (i = 1; n != -1; i++)
    {
        n = pos(s, c, i);

        if (n == 0)
```

Два цикла выполняют каждый свою задачу. Один топает по строке в надежде найти “голову” искомого слова (первый символ). Второй выясняет, есть ли после найденной “головы” “тело” искомого. Причем проверяет, не лежит ли это “тело” в конце строки. Т.е. не является ли длина найденного слова на единицу больше длины искомой строки, если учитывать, что в эту единицу попадает нуль-терминатор ('\0').

```
0
4
Для продолжения нажмите любую клавишу . . .
```

Мы видим, что программа нашла начало подстроки **pa** в ячейках символьного массива с индексом 0 и 4. Но почему? Ведь в слове **parapapa** **3** таких подстроки. Все дело в '\0' .

В целом смысл самого алгоритма на этом заканчивается. Больше никаких сложностей кроме нуля в конце строки нет. Однако, следует обратить внимание на множественность поиска. Что, если нам необходимо найти в строке несколько позиций?

Сколько раз искомое слово встречается в строке и в каких местах? Именно это и призван контролировать третий параметр – **int n** – номер вхождения в строку. Если поставить туда единицу – он найдет первое совпадение искомого. Если двойку, он заставит первый цикл пропустить найденное первое, и искать второе. Если тройку – искать третье и так далее. С каждым найденным искомым

словом, этот счетчик вхождений уменьшается на единицу. Это и позволяет описать поиск в цикле:

```
for (i = 1; n != -1; i++)
{
    n = pos(s, c, i);

    if (n >= 0)
        std::cout << n << std::endl;
}
```

То есть найти первое, второе, третье, четвертое совпадение. Пока функция не вернет **-1**, что укажет на отсутствие N-ного искомого в строке.

Теперь, для сравнения, поиск подстроки в строке C++ с хедером **string**.


```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");
    string s = "Hello world";
    cout << "Найдено в позиции " << s.find("lo") << endl;
}
```

```
Найдено в позиции 3
Для продолжения нажмите любую клавишу . . .
```

Как же множественность? Да пожалуйста:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s = "parapapa";
    int i = 0;

    for (i = s.find("pa", i++); i != string::npos; i = s.find("pa", i + 1))
        cout << i << endl;
}
```

Функция **find()** принимает вторым параметром номер символа, с которого начать поиск. Т.е. найдя первое вхождение, его значение увеличивается на единицу и **find()** продолжает поиск со следующего символа после найденной головы. Результат:

```
0
4
6
Для продолжения нажмите любую клавишу . . .
```

Всё это есть в C++, и сам класс **string** достаточно удобный для работы со строками именно, как строками, а не просто с массивом символов.

Задание к лабораторной работе №8:

Для вашей БД из лабораторной работы № 2 реализовать поиск подстроки в строке (фильтрацию, например по фамилии).

Лабораторная работа №9 Поиск записи по значению/индексу.

100% программистов, во время обучения, рано или поздно столкнутся с необходимостью проверить наличие в массиве определённого значения. Существует несколько общеизвестных алгоритмов поиска в языках программирования. Сейчас мы рассмотрим самый простой из них (но далеко не самый эффективный) – линейный поиск либо последовательный поиск. За счет того, что поиск проводится путем полного последовательного перебора элементов массива и сравнения его значений с заданным ключом, скорость работы алгоритма достаточно низкая.

В ниже приведенном примере объявим массив на 50 элементов и заполним его используя генератор случайных чисел `rand()`. Предложим пользователю ввести искомое значение с клавиатуры и реализуем проверку на наличие этого значения в нашем массиве. Если значение будет найдено в каком-либо элементе массива – выведем на экран индекс этого элемента.

```
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace std;

//прототипы функций
int linSearch(int arr[], int requiredKey, int size); // линейный поиск
void showArr(int arr[], int size); // показ массива

int main()
{
    setlocale(LC_ALL, "rus");
    const int arrSize = 50;
    int arr[arrSize];
    int requiredKey = 0; // искомое значение (ключ)
    int nElement = 0; // номер элемента массива
    srand(time(NULL));

    //запись случ. чисел в массив от 1 до 50
```

```

for (int i = 0; i < arrSize; i++)
{
    arr[i] = 1 + rand() % 50;
}

showArr(arr, arrSize);

cout << "Какое число необходимо искать? ";
cin >> requiredKey; // ввод искомого числа

//поиск искомого числа и запись номера элемента
nElement = linSearch(arr, requiredKey, arrSize);

if (nElement != -1)
{
    //если в массиве найдено искомое число - выводим индекс элемента на экран
    cout << "Значение " << requiredKey << " находится в ячейке с индексом: " <<
nElement << endl;
}
else
{
    //если в массиве не найдено искомое число
    cout << "В массиве нет такого значения" << endl;
}
return 0;
}

//вывод массива на экран
void showArr(int arr[], int arrSize)
{
    for (int i = 0; i < arrSize; i++)
    {
        cout << setw(4) << arr[i];
        if ((i + 1) % 10 == 0)
        {
            cout << endl;
        }
    }
    cout << endl << endl;
}

//линейный поиск
int linSearch(int arr[], int requiredKey, int arrSize)
{
    for (int i = 0; i < arrSize; i++)
    {
        if (arr[i] == requiredKey)
            return i;
    }
    return -1;
}

```

}

Функция выполняющая линейный поиск определена в строках 62-70. Она возвращает в программу -1 в том случае, если значение, которое ищет пользователь, не будет найдено в массиве. Если же значение будет найдено – функция вернет индекс элемента массива, в котором это значение хранится.

Результат выполнения программы в случае нахождения нужного значения:

```
29 40 9 42 50 9 27 32 28 25
1 20 17 22 6 15 30 36 19 45
22 8 23 27 23 43 22 38 34 13
17 29 6 27 18 35 46 10 29 11
20 31 11 26 32 21 25 18 3 25

Какое число необходимо искать? 27
Значение 27 находится в ячейке с индексом: 6
Для продолжения нажмите любую клавишу . . .
```

Результат работы программы в случае отсутствия нужного значения:

```
30 18 26 33 23 8 30 15 12 44
35 48 25 40 6 20 31 1 15 34
42 26 38 33 11 50 17 18 7 50
2 6 25 7 47 37 26 26 47 34
36 10 44 6 17 46 45 8 36 47

Какое число необходимо искать? 55
В массиве нет такого значения
Для продолжения нажмите любую клавишу . . .
```

Посмотрев на первый снимок, вы сразу обратите внимание, что в ячейке с индексом 6 искомое значение найдено и программа завершает работу, хотя в ячейках 23 и 33 этого массива находятся такие же значения. Если вас это устраивает, то индекс первого найденного элемента и будет результатом работы программы. Иначе программу надо дорабатывать, чтобы найти и записать (например в отдельный массив) все индексы ячеек, хранящих искомое число (ключ).

Обычно линейный поиск применяется для одиночного поиска в небольшом массиве, который не отсортирован. В других случаях, лучше и эффективней

сначала отсортировать массив и применять другие алгоритмы поиска. Например двоичный (бинарный) поиск либо другие.

Существуют другие алгоритмы поиска. Двоичный (бинарный) поиск является более эффективным (проверяется асимптотическим анализом алгоритмов) решением в случае, если массив заранее отсортирован.

Предположим, что массив из 12-ти элементов отсортирован по возрастанию

		ind 0	ind 1	ind 2	ind 3	ind 4	ind 5	ind 6	ind 7	ind 8	ind 9	ind 10	ind 11	
исходный массив	///	1	2	3	4	5	6	7	8	9	10	11	12	///

Пользователь задает искомое значение (ключ поиска). Допустим 4. На первой итерации массив делится на две части (ищем средний элемент – midd): $(0 + 11) / 2 = 5$ (0.5 отбрасываются). Сначала, проверяется значение среднего элемента массива. Если оно совпадает с ключом – алгоритм прекратит работу и программа выведет сообщение, что значение найдено. В нашем случае, ключ не совпадает со значением среднего элемента.

		ind 0	ind 1	ind 2	ind 3	ind 4	ind 5	ind 6	ind 7	ind 8	ind 9	ind 10	ind 11	
I шаг	///	1	2	3	4	5	6	7	8	9	10	11	12	///
		left					midd						right	

Если ключ меньше значения среднего элемента, алгоритм не будет проводить поиск в той половине массива, которая содержит значения больше ключа (т.е. от среднего элемента до конца массива). Правая граница поиска сместится ($\text{midd} - 1$). Далее снова деление массива на 2.

		ind 0	ind 1	ind 2	ind 3	ind 4	ind 5	ind 6	ind 7	ind 8	ind 9	ind 10	ind 11	
II шаг	///	1	2	3	4	5	///	///	///	///	///	///	///	///
		left		midd		right								

Ключ снова не равен среднему элементу. Он больше него. Теперь левая граница поиска сместится ($\text{midd} + 1$).

		ind 0	ind 1	ind 2	ind 3	ind 4	ind 5	ind 6	ind 7	ind 8	ind 9	ind 10	ind 11	
III war	///	///	///	///	4	5	///	///	///	///	///	///	///	
					left / midd	right								

На третьей итерации средний элемент – это ячейка с индексом 3: $(3 + 4) / 2 = 3$. Он равен ключу. Алгоритм завершает работу.

Пример:

```

def search(arr, key):
    """Функция поиска элемента в массиве"""
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return -1

# Пример использования
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
key = 4
result = search(arr, key)
print(f"Элемент {key} находится по индексу {result}")

```

Результат работы программы:

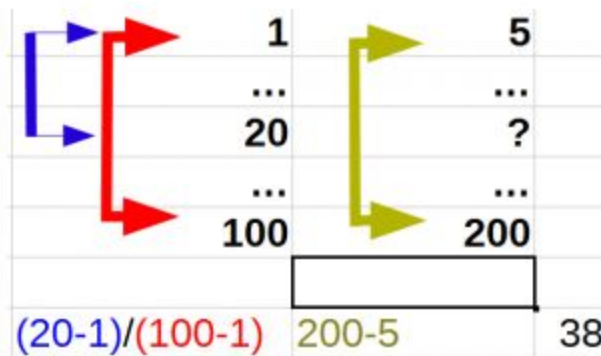
```

1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
Введите любое число: 4
Указанное число находится в ячейке с индексом: 3
Для продолжения нажмите любую клавишу . . .

```

Недостатками такого алгоритма является требование упорядоченности данных, а также возможности доступа к любому элементу данных за постоянное (не зависящее от количества данных) время. Таким образом алгоритм не может работать на неупорядоченных массивах и любых структурах данных, построенных на базе связанных списков.

Далее подвергнем рассмотрению **интерполирующий поиск**. Не стоит пугаться сложного слова, всё гораздо проще чем можно себе представить. Помните по геометрии подобие треугольников? Те, у которых одинаковые по значению углы, но пропорции разные. Здесь практически тот же принцип. Вычисляется длина области поиска, и длина от начала области до некоего числа (скажем до центрального элемента в массиве). Вычисление это проводится как с номерами элемента, так и с их значениями, после чего полученная длина области умножается на длину между значениями, и результат прибавленный к значению из начала области дает искомое.



Примерно так это будет выглядеть в виде картинки

Формула достаточно проста – вычисляется длина между номерами первого элемента и искомого (задаваемого точнее). Такая же длина считается между первым и последним номерами. Длины между собой делятся, как раз и получая вычисление подобия. То же самое происходит со значениями элементов – так же вычисляется расстояние между граничными значениями в массиве. Специально выделяю цветом понятия и связанные с ними части формулы.

Полученная длина номеров элементов массива умножается на длину значений в этих (граничащих) элементах и прибавляется значение в первой ячейке массива.

Получается: $1 + (20-1)/(100-1) * (200-5) = 38$ с копейками.

Результат и есть то самое искомое. Т.е. при таких значениях как на картинке в ячейке №20 будет стоять 38. Вот и весь смысл интерполяции – составления подобия между номерами элементов и между значениями элементов.

На C++ это выглядит так:

```
#include <iostream>
using namespace std;
```

```

int main()
{
    //Массив значений в котором пойдет поиск
    int MyArray[] { 1, 2, 4, 6, 7, 89, 123, 231, 1000, 1235 };

    int x = 0; //Текущая позиция массива, с которым сравнивается искомое
    int a = 0; //Левая граница области, где ведется поиск
    int b = 9; //Правая граница области, где ведется поиск

    int WhatFind = 123; //Значение, которое нужно найти
    bool found; //Переменная-флаг, принимающая True если искомое найдено

    /***** Начало интерполяции *****/

    //Цикл поиска по массиву, пока искомое не найдено
    //или пределы поиска еще существуют
    for (found = false; (MyArray[a] < WhatFind) && (MyArray[b] > WhatFind) && !found; )
    {
        //Вычисление интерполяцией следующего элемента, который будет
сравниваться с искомым
        x = a + ((WhatFind - MyArray[a]) * (b - a)) / (MyArray[b] - MyArray[a]);
        //Получение новых границ области, если искомое не найдено
        if (MyArray[x] < WhatFind)
            a = x + 1;
        else if (MyArray[x] > WhatFind)
            b = x - 1;
        else
            found = true;
    }

    /***** Конец интерполяции *****/

    //Если искомое найдено на границах области поиска, показать на какой границе оно
    if (MyArray[a] == WhatFind)
        cout << WhatFind << " founded in element " << a << endl;
    else if (MyArray[b] == WhatFind)
        cout << WhatFind << " founded in element " << b << endl;
    else
        cout << "Sorry. Not found" << endl;

    return 0;
}

```

Таким образом сам цикл просто вычисляет по формуле область массива, где может находиться искомое используя этот самый принцип интерполяции в C++, подбирая подобия так сказать. Если вычисленное не равно искомому, значит нужно сдвинуть границы области, где проходит вычисление.

Если вычисленное больше – сдвигается правая граница области поиска, если меньше – левая. Так отрезая (как в бинарном поиске) кусок массива за куском постепенно достигается нужная ячейка массива, ну или границы области поиска сужаются до таких величин, в пределах которого уже искать нечего, когда дистанция между границами равна 1 (т.е. между точкой А и В нет более элементов для вычисления) решение говорит о том, что значение в массиве не найдено.

Как видите сам цикл не настолько уж и сложен и ужасен. Вся соль то скрыта в его форуме. А остальное – просто проверки: нашли или дальше искать.

Задание к лабораторной работе №9:

В соответствии с содержанием своей БД (из лабораторной работы № 2) реализовать алгоритм интерпретирующего поиска индекса в вашем массиве структур и бинарный поиск по любому из типов данных (например string, по фамилии).