Лабораторная работа № 9

**Поиск по значению/ индексу**

**Цель:** изучить прием поиска, применить знания для осуществления поиска по значению и индексу.

Поиск — обработка некоторого множества данных с целью выявления подмножества данных, соответствующего критериям поиска.  
  
Все алгоритмы поиска делятся на

* поиск в неупорядоченном множестве данных;
* поиск в упорядоченном множестве данных.

Упорядоченность – наличие отсортированного ключевого поля.

**Линейный поиск** по сравнению с бинарным гораздо проще. Бинарный поиск — очень быстрый алгоритм с не сложной реализацией, который находит элемент с определенным значением в уже **отсортированном** массиве.

Очень важно помнить! Алгоритм будет работать правильно, только с отсортированным массивом. А если по случайности вы забыли отсортировать массив перед его использованием, то в большинстве случаев тот ответ, который подсчитал алгоритм, будет неверным.

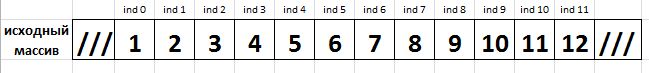
самый простой для реализации алгоритм поиска (но и самый затратный по времени) — линейный поиск.

Этот алгоритм перебирает все элементы в массиве, сравнивая их с заданным ключом, из-за полного перебора скорость поиска намного меньше, чем в других алгоритмах.

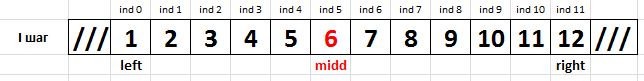
Его обычно используют, если в отрезке поиска находится мало элементов, в ином случае используют другие алгоритмы поиска (один из них бинарный поиск).

Двоичный (бинарный) поиск является более эффективным (проверяется [асимптотическим анализом алгоритмов](http://pro-prof.com/archives/1660)) решением в случае, если массив заранее отсортирован.

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

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c.jpg)

Пользователь задает искомое значение (ключ поиска). Допустим 4. На первой итерации [массив](https://purecodecpp.com/archives/1) 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.

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-2.jpg)

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 (midd – 1). Далее снова деление массива на 2.

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-31.jpg)

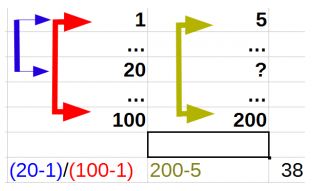
Ключ снова не равен среднему элементу. Он больше него. Теперь левая граница поиска сместится (midd + 1).

[](https://purecodecpp.com/wp-content/uploads/2015/06/binarnyi-dvoichnyi-poisk-c-4.jpg)

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

**Интерполяция** – это (если говорить на языке студента) определение области поиска, путем вычисления подобия расстояний между искомым значением и всей областью. 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.

Сложно сразу понять на словах, поэтому попробую показать картинкой:

[](https://purecodecpp.com/wp-content/uploads/2015/08/interpoliruyushii-poisk.jpg)

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.

# Поиск подстроки в строке

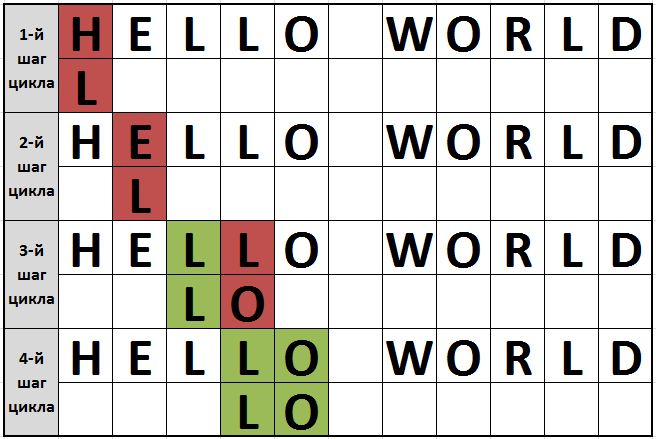
Сейчас мы разберём на примерах, как может выглядеть алгоритм поиска подстроки в строке. Примеры будут основываться на функциях стандартных библиотек, ведь именно в таких функциях и проявляются все удобства написания программ.  А вот классический разбор алгоритма, основанный на [циклах](https://purecodecpp.com/archives/623) 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:

[](https://purecodecpp.com/wp-content/uploads/2015/09/poisk-podstroki-v-stroke-algoritm.jpg)

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.

**Пример поиска по индексу:**

void search() {

int i, j, found = 0;

student s1;

FILE\* fp;

int rno;

printf("Введит код студента для поиска : ");

scanf("%d", &rno);

fp = fopen("mystudents1.txt", "r");

while (fread(&s1, sizeof(student), 1, fp)) {

if (s1.rno == rno) {

found = 1;

printf("\n%-5d%-20s%-20s%-20s", s1.rno, s1.name, s1.surname, s1.second\_name);

for (j = 0; j < 3; j++) {

printf("%4d", s1.sub[j].mark);

}

printf("%5d%7.2f", s1.total, s1.per);

}

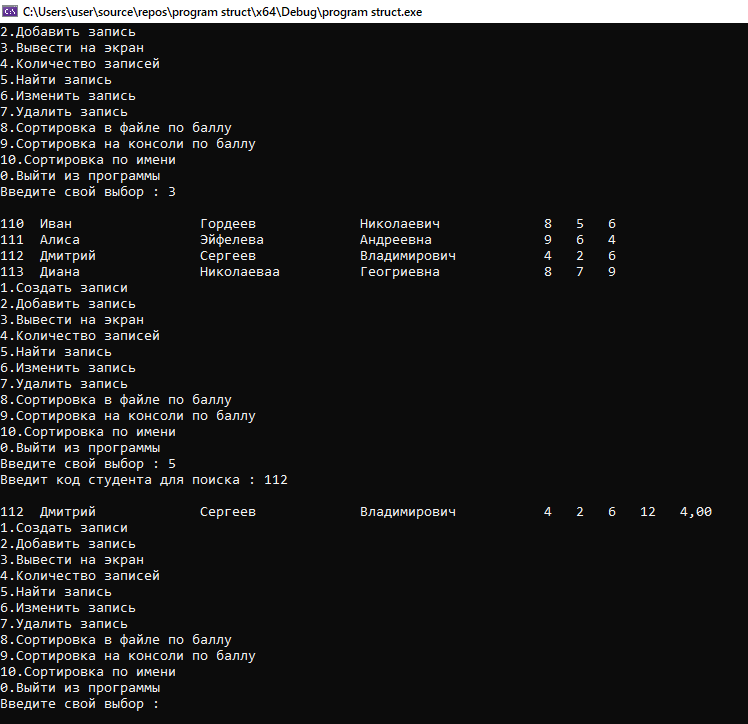
}

if (!found)

printf("\nНе найдено\n");

fclose(fp);

}



**ЗАДАНИЕ**

Создайте структуру Отель с полями Имя гостя, Фамилия, Номер паспорта, Национальность, Дата рождения, Номер комнаты, Дата заселения, Дата выезда, Цена пребывания в номере за 1 сутки, Карта лояльности сети отелей (есть / нет).

Подсчитать количество дней, проведенных в отеле и стоимость проживания. При наличии у гостя карты лояльности, применить скидку 15 %. Найдите самого частого клиента и выведите людей, проживающих на одном этаже (первая цифра номера комнаты – этаж). Вывести список и количество отобранных людей.