**Министерство Образования, Культуры и Исследований**

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

**Отчет по лабораторной работе №1**

**по предмету:**

**«Алгоритмы, Структуры Данных»**

***Выполнила:*** Спыну Екатерина,

группа IA2102 (группа:дизайна)

***Проверяющий: Mихаил Кроитор***

**Кишинев, 2023**

1. Создать текстовый файл, содержащий не менее 50 записей и минимум 5 полей (не менее 2 типов данных). Текстовый файл обязательно должен содержать ключевое поле.
2. Написать программу на языке C++, в которой реализованы методы поиска в неупорядоченных и упорядоченных таблицах по ключевому полю:
   1. Линейный поиск.
   2. Бинарный поиск.
   3. Интерполяционный поиск.
   4. Поиск методом Фибоначчи.
   5. Поиск в бинарном дереве.
3. Рассмотреть теоретическую и практическую длину метода поиска.
4. Описать пошаговый алгоритм метода.

**Выбор темы и названия полей:**

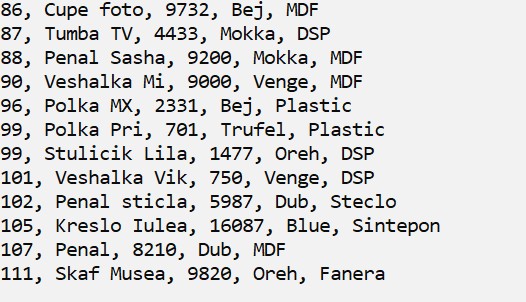
Я выбрала тему мебельный магазин. В качестве 5 полей было выбрано (Название товара -name ,айди- id , цвет -color, цена- price ,название материала -material )

Данные были представлены в блокноте, всего было два вида: (mebeli .csv) неотсортированные данные.



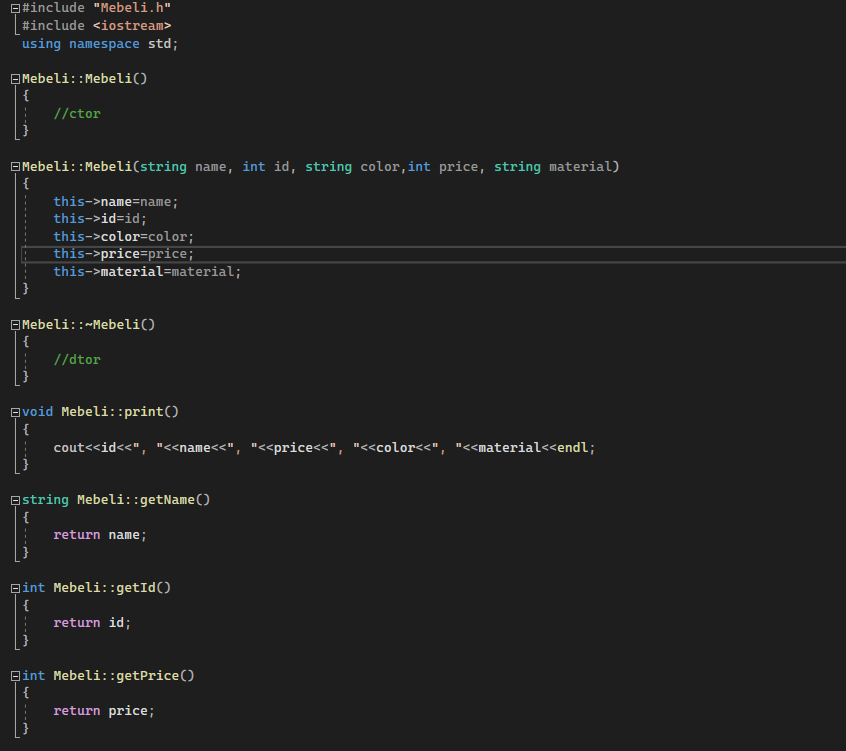
Отсортированные данные:



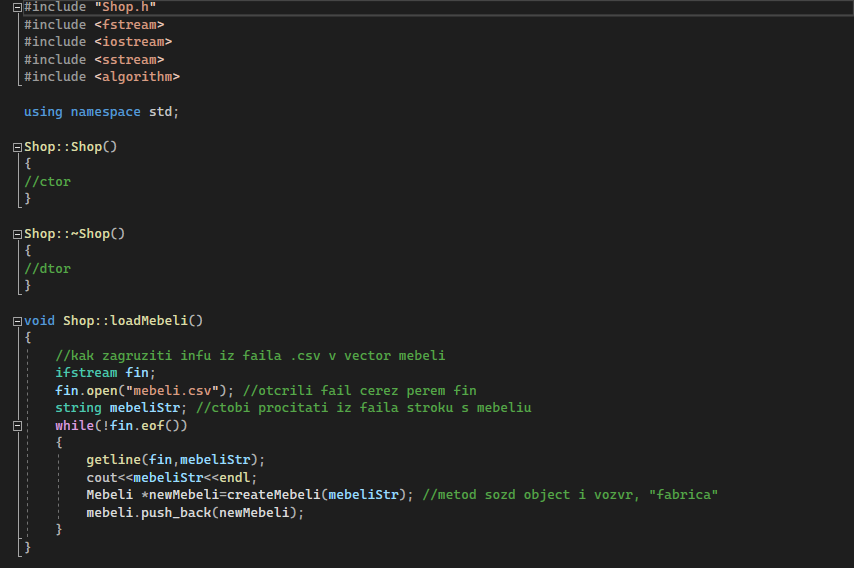


**Реализация ввода данных**

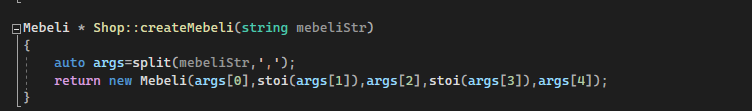
В проекте я создала Класс: Mebeli.h, в котором создала конструктор с параметрами, в который входят 5 полей: (string name, int id, string color, int price ,string material. )

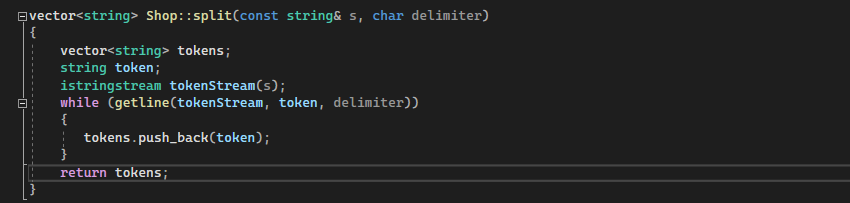


Загрузка текстового файла в программу была введена через функцию loadMebeli() следующим образом:

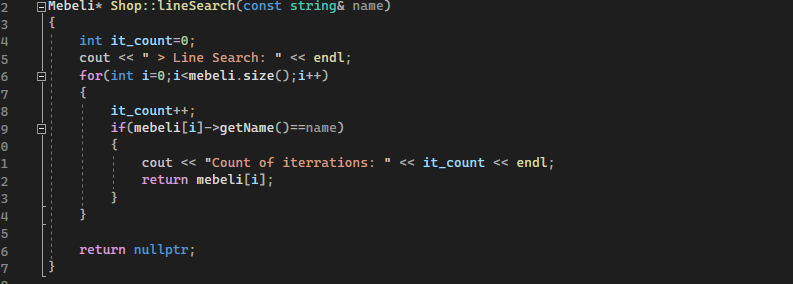


Далее данные из текстового файла надо переместить в мой вектор мебель, вектор- «умный массив.»

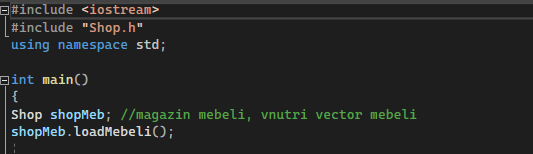




В классе Shop объявляю все функции методов поиска. В файле shop.cpp выполняю реализацию всех функции методов поиска.



## Главная программа main.cpp

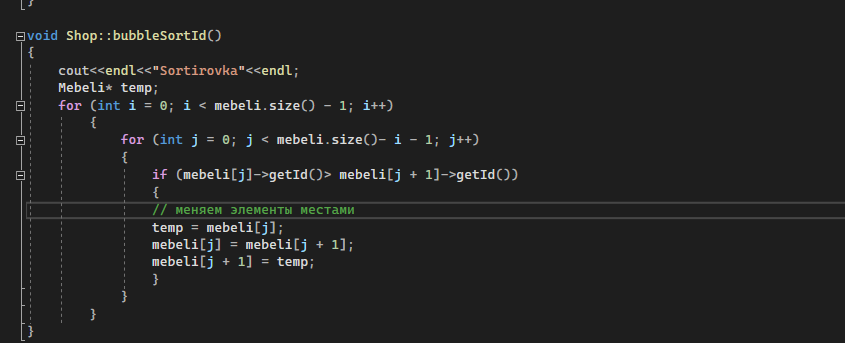


**Shop shopMeb** объект типа Shop. Абстрактно это наш магазин. А в самом магазине находится товар, наша мебель, в программе это вектор с мебелью.

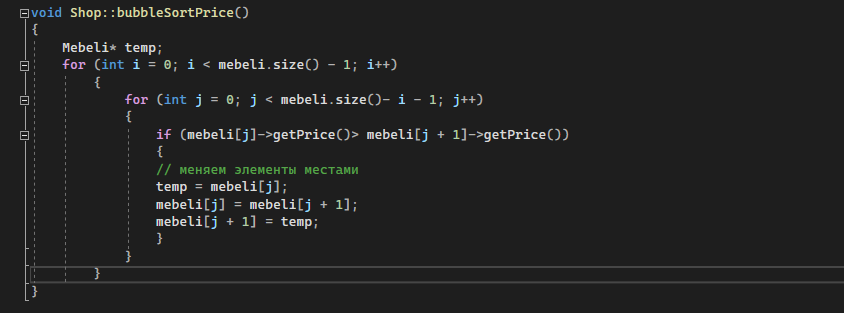
**shopMeb.loadMebeli() ;** это функция для вызова функции разных методов. Она загружает мебель.

Далее присутствуют вызовы функции методов, которые я представлю ниже, на каждом методе поиска.

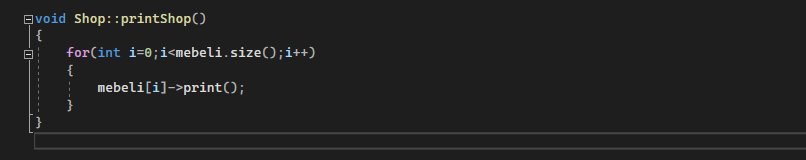
Сортировка по id товара:



Сортировка по цене:



Распечатка всего магазина



**1)Линейный метод поиска**

**Линейный** **поиск** — алгоритм нахождения заданного значения произвольной функции на некотором отрезке. Данный алгоритм является простейшим алгоритмом **поиска**, имеет простейшую реализацию и единственный может выполняться как для отсортированных данных, так и не для отсортированных данных. Но линейный метод является самым затратным по времени выполнения.

**Алгоритм поиска:**

Пусть K – некий ключ таблицы.

Данные таблицы представлены в виде массива A с N элементами, искомое значение равно Z.

Шаг 1. Начальная инициализация i := 0 (начало массива).

Шаг 2. Если A[i].K = Z, вернуть i.

Шаг 3. Иначе i := i +1.

Шаг 4. Если i ≤ N переходим на шаг 2.

Шаг 5. Иначе вернуть -1 ( пустой элемент, я использовала return nullptr)

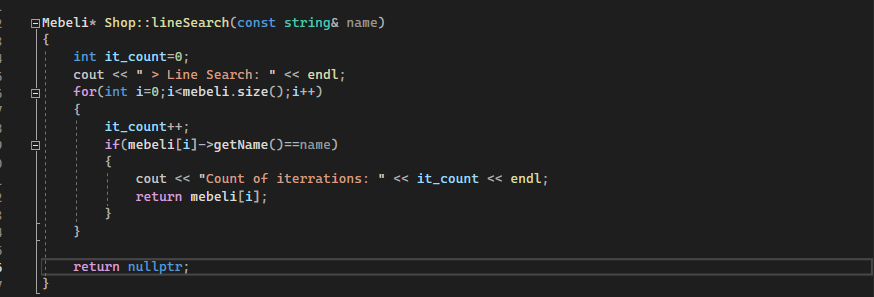
**Функция на языке C++** Параметры функции:

* vector <mebeli> – Я создала вектор ( умный массив) mebeli. Данный вектор будет использоваться во всех функциях как массив с данными о мебели.
* mebeli.size() –количество записей в массиве.
* name – числовой ключ. Возвращение значений (return).

Индекс записи – ключ найден.

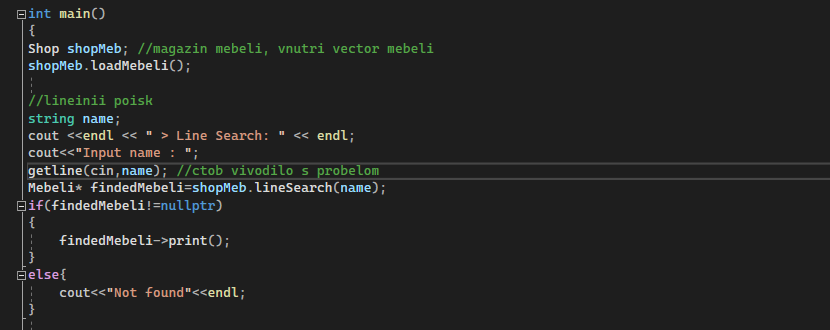
* nullptr (код завершения) ) нулевой указатель – неудачный поиск.

**Код программы:**

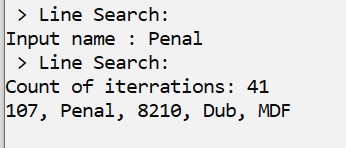


**Выполнение алгоритма функции:**

Поиск по полю название:



Количество итераций: 41 итерация.



Данный вид итерации более простой в плане кода, но затратный в плане количества итераций, так как поиск происходит в не отсортированных данных.

## 2)Бинарный поиск

Бинарный поиск в сравнении с линейным является поиском в отсортированных таблицах, т.е. ключи находятся в зависимости 𝐾1 < 𝐾2 < 𝐾3 < ⋯ < 𝐾𝑛. Данный метод поиска применяются для отсортированных данных.

Плюсом является то, что в данном методе на каждом шаге отсекается 1/2 часть массива, что в двое ускоряет метод поиска. Данные делятся по половинке. Этот метод можно сравнить с численным методом половинного деления.

**Алгоритм поиска:**

Пусть K – некий ключ таблицы. Данные таблицы представлены в виде массива A с N элементами, искомое значение равно Z. l и r – правая и левая граница массива соответственно.

Шаг 1. Выбирается элемент 𝑐 – элемент в середине массива A (𝑐 = 𝑙+𝑟 2 ).

Шаг 2. Если A[c].K = Z, возвращаем c.

Шаг 3. Иначе, если A[c].K > Z, то r := c.

Шаг 4. Иначе, если A[c].K < Z, то l := r.

Шаг 5. Если l < r переходим на шаг 2.

Шаг 6. Иначе вернуть -1.

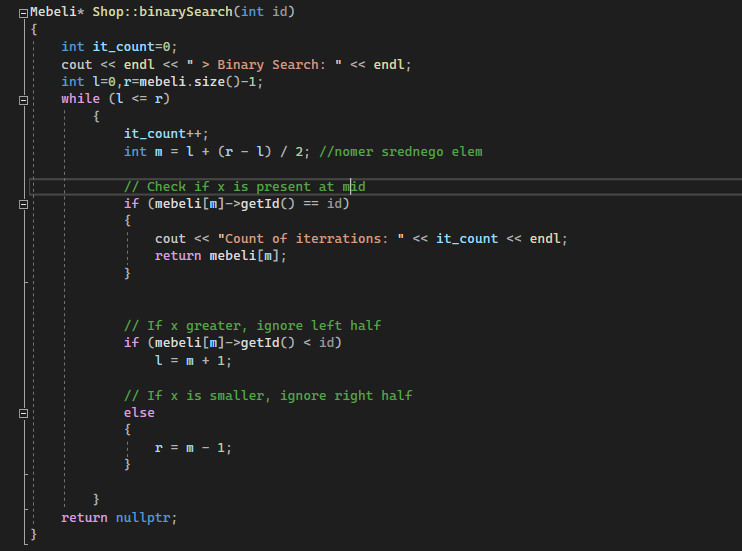
**Функция на языке C++** Параметры функции:

* vector <mebeli> Данный вектор будет использоваться во всех функциях как массив с данными о мебели.
* id– числовой ключ.

getId()- функция получения id

* mebeli.size() – количество записей в массиве.
* nullptr (код завершения) нулевой указатель – неудачный поиск.

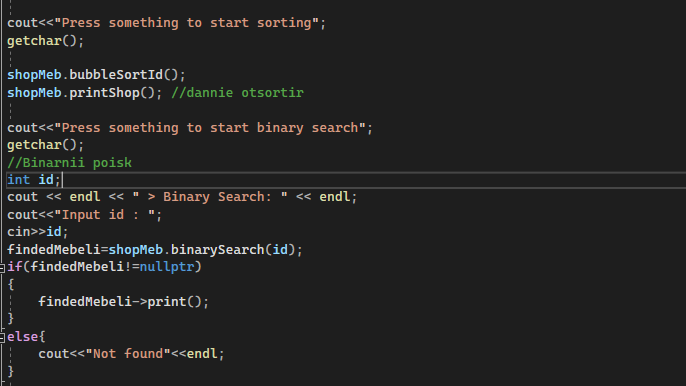
**Код программы (BinarySearch):**



**Выполнение алгоритма**

Вызов функции: findedMebeli=shopMeb.binarySearch(id);

Поиск по полю id



Количество итераций: 6



В сравнении с прошлым методом количество итераций намного меньше. Это благодаря тому, что поиск совершается уже в отсортированных данных.

## Интерполяционный поиск

Интерполяционный поиск работает только с упорядоченными массивами. Интерполяционный поиск не делит последовательность на две равные части, а вычисляет приблизительное расположение ключа (искомого элемента), ориентируясь на расстояние между искомым и текущим значением элемента.

Идея алгоритма напоминает хорошо знакомый старшим поколениям поиск телефонного номера в обычном справочнике: список имен абонентов упорядочен, поэтому не составит труда найти нужный телефонный номер, так как, если мы, например, ищем абонента с фамилией, начинающейся на букву «Ю», то для дальнейшего поиска разумно будет перейти в конец справочника.

**Алгоритм поиска:**

Пусть K – некий ключ таблицы. Данные таблицы представлены в виде массива A с N элементами, искомое значение равно Z. left и right – правая и левая граница массива соответственно.

Шаг 1. Проверяем, если левая и правая позиция массива не равна (left != right).

Шаг 2. Если правая и левая позиция равна, то проверяем, если A[left].K = Z.

Шаг 3. Если условие выполняется, то возвращаем значение left. Иначе, возвращаем значение -1.

Шаг 4. Считаем новую позицию 𝑝𝑜𝑠𝑖𝑡𝑖𝑜𝑛 ≔ 𝑙𝑒𝑓𝑡 + 𝑟𝑖𝑔ℎ𝑡−𝑙𝑒𝑓𝑡 𝐴[𝑟𝑖𝑔ℎ𝑡].K−𝐴[𝑙𝑒𝑓𝑡].K ∗ (𝑍 − 𝐴[𝑙𝑒𝑓𝑡]).

Шаг 5. Если A[position].K = Z, то возвращаем position. Иначе, если A[position].K < Z, то left := position + 1, иначе, right := position + 1.

Шаг 6. Если left <= right, Z >= A[left].K, key <= A[right].k переходим на шаг 1 . Иначе возвращаем значение -1.

**Функция на языке C++** Параметры функции:

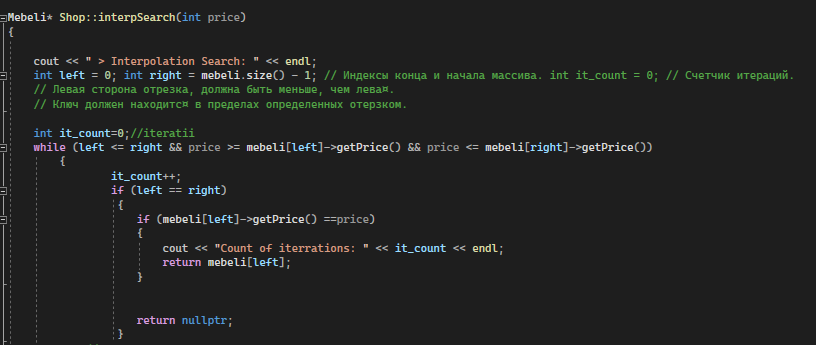
* vector <mebeli> Данный вектор будет использоваться во всех функциях как массив с данными о мебели.
* price – числовой ключ.

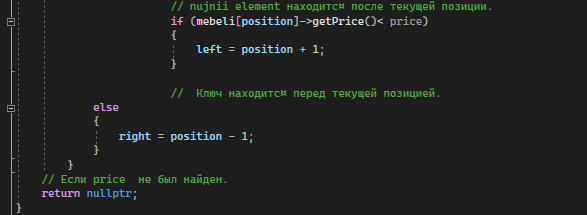
getPrice ()- функция получения id

* mebeli.size() – количество записей в массиве.
* nullptr (код завершения) нулевой указатель – неудачный поиск.

**Код программы: (InterpolationSearch)**

**Поиск по полю цена.**

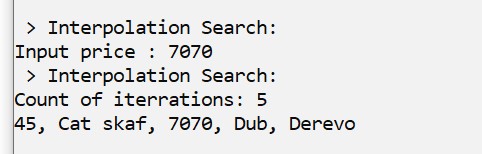




**Выполнение алгоритма**

Вызов функции: findedMebeli=shopMeb.interpSearch(price);

Количество итераций: 5



Данный метод является более эффективным и быстрым, так как итерации меньше, чем в предыдущем методе, который тоже ищет нужный элемент в отсортированных данных.

## Поиск методом Фибоначчи

Метод Фиббоначи начали использовать не так давно, хотя сам метод появился еще давно. Алгоритм применяется только в отсортированных данных. У него есть преимущество в том, что он не использует операции деления, благодаря чему уменьшаются вероятность ошибок, а также увеличивается время выполнения. Числа Фибоначчи можно вычислять, используя сложение, которое выполняется быстрее, чем деление).

**Алгоритм поиска**

Пусть K – некий ключ таблицы. Данные таблицы представлены в виде массива A с N элементами, искомое значение равно Z. left и right – правая и левая граница массива соответственно.

Шаг 1. Предполагается, что количество элементов массива A N+1 <= Fm+1 (m+1-е число Фибоначчи).

Шаг 2. Находим k такое, что 𝐹𝑚+1 ≥ 𝑁.

Шаг 3. Определяем сдвиг offset = -1, для корректировки размера массива.

Шаг 4. Определяем начальные параметры: 𝐹𝑚−2 − (m-2) элемент списка Фибоначчи; 𝐹𝑚−1 − (m-1) элемент списка Фибоначчи ; 𝐹𝑚 − m элемент списка Фибоначчи.

Шаг 5. Проверяем, если число Фибоначчи на правильной позиции. Если offset + 𝐹𝑚−2 < N - 1, то i := offset + 𝐹𝑚−2, иначе i := count – 1.

Шаг 6. Если A[i].K < Z, то 𝐹𝑚 = 𝐹𝑚−1, 𝐹𝑚−1 = 𝐹𝑚−2 ; 𝐹𝑚−2 = 𝐹𝑚 − 𝐹𝑚−1. Иначе, если A[i].K > Z, то 𝐹𝑚 = 𝐹𝑚−2, 𝐹𝑚−1 = 𝐹𝑚−1 − 𝐹𝑚−2, 𝐹𝑚−2 = 𝐹𝑚 − 𝐹𝑚−1. Иначе, если A[i].K = Z, то возвращаем i.

Шаг 7. Если 𝐹𝑚−1 > 1, то переходим на шаг 5.

Шаг 8. Иначе, если 𝐹𝑚−1 ≤ 1, то проверяем, если 𝐹𝑚−1! = 0 и A[offset + 1].K = Z, то возвращаем значение offset + 1. Шаг 9. Возвращаем значение -1.

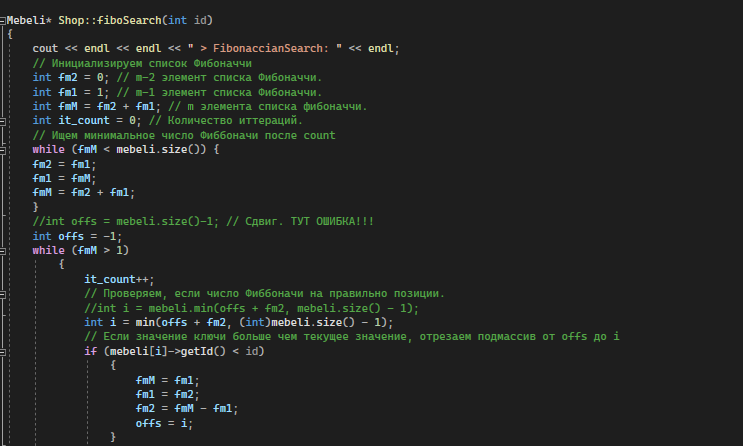
**Функция на языке C++** Параметры функции:

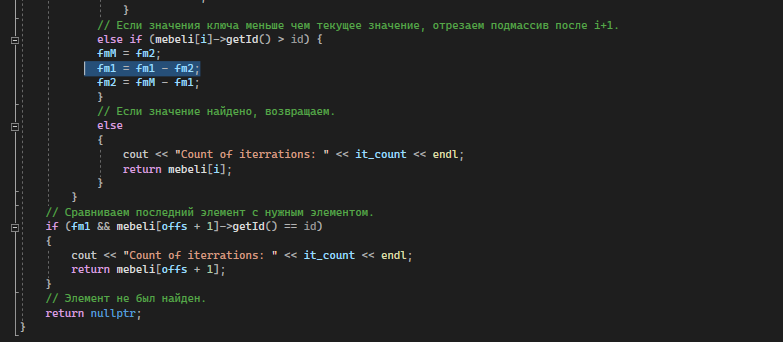
* vector <mebeli> Данный вектор будет использоваться во всех функциях как массив с данными о мебели.
* id– числовой ключ.

getId()- функция получения id

* mebeli.size() – количество записей в массиве.
* nullptr (код завершения) нулевой указатель – неудачный поиск.

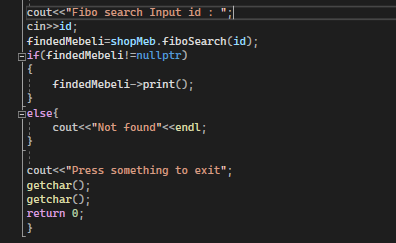
**Код программы (FibonacciSearch)**



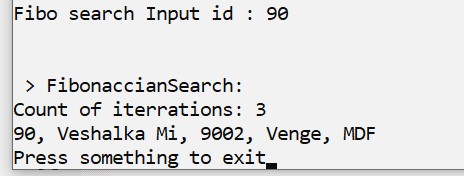


**Выполнение алгоритма**

Вызов функции: findedMebeli=shopMeb.interpSearch(price);



Количество итераций: 3



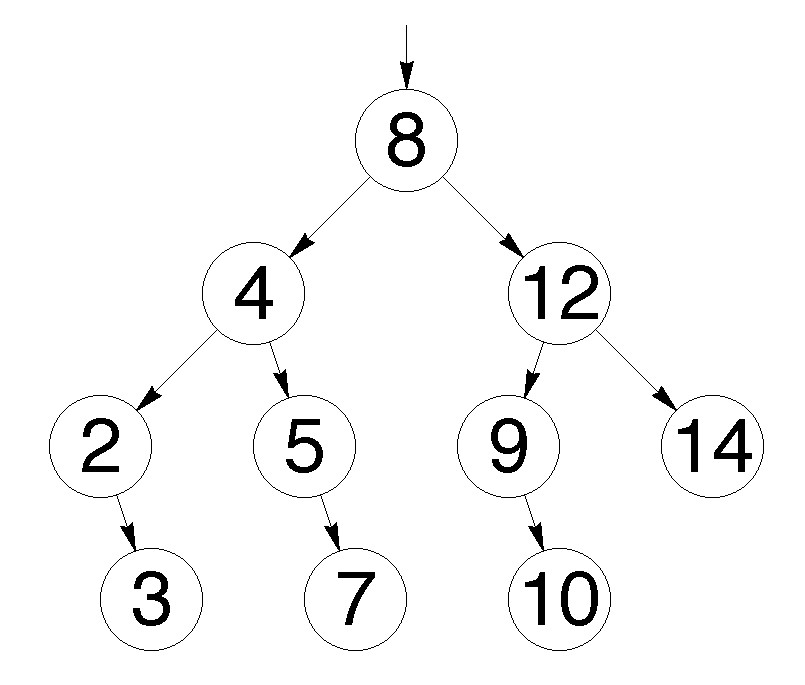
Данный метод еще более эффективный, так как ему потребовалось меньше итерации, только 3.

**5)Поиск в бинарном дереве**

Алгоритм поиска в бинарном дереве является методом, который отличается от прежних, так как для него требуется представление данных в виде бинарного (двоичного) дерева поиска.

В бинарном дереве каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим-либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.



**Алгоритм поиска**

Пусть K – некий ключ структуры A.

T – бинарное дерево хранящие данные типа A, а искомое значение равно Z.

Представим Алгоритм в виде псеводкода.

ПОИСК\_БИН (Дерево T, Поиск Z): ЕСЛИ T = 0 или T.K = Z то: ВЕРНУТЬ  T

ИНАЧЕ

ЕСЛИ Z > T.K то: T  ПОИСК\_БИН (правый\_потомок , Z) ИНАЧЕ: T  ПОИСК\_БИГ (левый\_потомок, Z).

ВЕРНУТЬ  T.

**Параметры функции:**

* tree – указать на вершину бинарного дерева.
* key – строковый ключ.
* it\_count –итерации.
* Нулевой указатель – неудачный поиск

**Выполнение алгоритма**

Данный метод для меня показался очень понятным и простым в плане понимания и выполнения алгоритма. Но более сложным, в написания кода. С данным методом возникли ошибки при написании кода.

**Вывод:**

В данной лабораторной работе я ознакомилась с различными алгоритмами методов поиска, такими как: Линейный, Бинарный, Интерполяционный, Метод Фибоначчи, Поиск в Бинарном Дереве.

Проанализировав информацию о сложности и количестве итераций, можно сделать вывод, что из вышеописанных алгоритмов самыми эффективными являются: Интерполяционный Поиск и Метод Фибоначчи.

Но мне больше всего понравился бинарный поиск, в принципе он тоже эффективный.

Так же научилась передавать текстовый файл в саму программу и данные с файла преобразовывать в вектор, над которыми уже можно производить действия, сортировать, проводить поиски по нужному элементу ключа.

**Библиография**

1. Архив myCoolProject.zip
2. Теоретическая часть moodle – курс ASDC – «Поиск» -

https://moodle.usm.md/mod/book/view.php?id=262060