ЗМІСТ

[ВСТУП 5](#_Toc514792630)

[1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ДИНАМІЧНІ СТРУКТУРИ ДАНИХ 8](#_Toc514792631)

[1.1 Черга.[4] 10](#_Toc514792632)

[1.2 Стек.[5] 11](#_Toc514792633)

[1.3 Односпрямовані і двонаправлені списки.[6] 14](#_Toc514792634)

[1.4 Бінарні дерева. [7] 16](#_Toc514792635)

[2. ПРОГРАМНА РЕАЛІЗАЦІЯ 22](#_Toc514792636)

[2.1. Основні відомості 22](#_Toc514792637)

[2.2 Опис роботи функцій 24](#_Toc514792638)

[3. ІНСТРУКЦІЯ КОРИСТУВАЧА 36](#_Toc514792639)

[ВИСНОВКИ 46](#_Toc514792640)

[ПЕРЕЛІК ПОСИЛАНЬ ТА СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 47](#_Toc514792641)

[КОД ПРОГРАМИ 48](#_Toc514792642)

# ВСТУП

Мова програмування С. [1]

C - компільована статично типізована мова програмування загального призначення, розроблена в 1969-1973 роках співробітником Bell Labs Денніс Рітчі як розвиток мови Бі. Спочатку була розроблений для реалізації операційної системи UNIX, але, в наслідку, була перенесениа на безліч інших платформ. Згідно дизайну мови Сі, Її конструкції близько зіставляються типовим машинним інструкціям, завдяки чому він знайшов застосування в проектах, для яких був властивий мову асемблера, в тому числі як в операційних системах, так і в різному прикладному ПО для безлічі пристроїв - від суперкомп'ютерів до вбудованих систем. Мова програмування Сі зробила істотний вплив на розвиток індустрії програмного забезпечення, а Її синтаксис став основою для таких мов програмування, як C ++, C #, Java і Objective-C.

Мова програмування С - найшвидша у світі високорівнева мова програмування.  
 Для мови С характерні лаконічність, стандартний набір конструкцій управління потоком виконання, структур даних і великий набір операцій.  
С цінують за його ефективність. Він є найпопулярнішою мовою для створення системного програмного забезпечення. Його також часто використовують для створення прикладних програм. Незважаючи на те, що С не розроблявся для новачків, він активно використовується для навчання програмуванню.  
Сі - сучасна мову. Вона включає в себе ті керуючі конструкції, які рекомендуються теоретичним і практичним програмуванням. Його структура спонукає програміста використовувати в своїй роботі спадне проектування, структурне програмування і покрокову розробку модулів. Результатом такого підходу є надійна і читаєма програма.  
 Сі - ефективна мова. Її структура дозволяє найкращим чином використовувати можливості сучасних ЕОМ. На мові Сі програми зазвичай відрізняються компактністю і швидкістю виконання.  
 Сі - переносна, або мобільна, мова. Це означає, що програма, написана на Сі для однієї обчислювальної системи, може бути перенесена з невеликими змінами (або взагалі без них) на іншу. Якщо модифікації все-таки необхідні, то часто вони можуть бути зроблені шляхом простої зміни декількох елементів в "головному" файлі, який супроводжує головній програмі.

Мова програмування С++.[2]

C++ (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом (англ. Bjarne Stroustrup) в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) у 1979. Базується на мові С.

 У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java.

При створенні С++ прагнули зберегти сумісність з мовою С. Більшість програм на С справно працюватимуть і з компілятором С++.

С++ має наступні нововведення порівняно з С:

- підтримка об'єктно-орієнтованого програмування через класи;  
- підтримка узагальненого програмування через шаблони;  
- доповнення до стандартної бібліотеки;  
- додаткові типи даних;  
- обробка винятків;  
- простори імен;  
- вбудовані функції;  
- перевантаження операторів;  
- перевантаження імен функцій;  
- посилання і оператори управління вільно розподіленою пам'яттю.

Назва «Сі++» була вигадана Ріком Масситті (Rick Mascitti) і вперше було використана в грудні 1983 року. Раніше, на етапі розробки, нова мова називалася «Сі з класами». Ім'я, що вийшло у результаті, походить від оператора Сі «++» (збільшення значення змінної на одиницю) і поширеному способу присвоєння нових імен комп'ютерним програмам, що полягає в додаванні до імені символу «+» для позначення поліпшень. Згідно зі Страуструпом, «ця назва вказує на еволюційну природу змін Ci».

C++ працює під керуванням багатьох операційних систем. До них входять: Mac, Linux (включаючи дистрибутив Ubuntu усіх версій), Windows.

Особливості роботи у Linux.

1. ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

Динамічні структури даних[3] - це структури даних, пам'ять під які виділяється і звільняється в міру необхідності.

Динамічні структури даних в процесі існування в пам'яті можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами. При цьому не враховується зміна вмісту самих елементів даних. Така особливість динамічних структур, як мінливість їх розміру і характеру відносин між елементами, призводить до того, що на етапі створення машинного коду програма-компілятор не може виділити для всієї структури в цілому ділянку пам'яті фіксованого розміру, а також не може зіставити з окремими компонентами структури конкретні адреси. Для вирішення проблеми адресації динамічних структур даних використовується метод, званий динамічним розподілом пам'яті, тобто пам'ять під окремі елементи виділяється в момент, коли вони "починають існувати" в процесі виконання програми, а не під час компіляції. Компілятор в цьому випадку виділяє фіксований обсяг пам'яті для зберігання адреси динамічно розміщується елемента, а не самого елемента.

Динамічна структура даних характеризується тим що:

-вона не має імені;

-їй виділяється пам'ять в процесі виконання програми;

-кількість елементів структури може не фіксуватися;

-розмірність структури може змінюватися в процесі виконання програми;

-в процесі виконання програми може змінюватися характер взаємозв'язку між елементами структури.

Кожній динамічної структурі даних зіставляється статична змінна типу покажчик (її значення - адреса цього об'єкта), за допомогою якої здійснюється доступ до динамічної структурі.

Самі динамічні величини не вимагають опису в програмі, оскільки під час компіляції пам'ять під них не виділяється. Під час компіляції пам'ять виділяється тільки під статичні величини. Покажчики - це статичні величини, тому вони вимагають опису.

Необхідність в динамічних структурах даних зазвичай виникає в наступних випадках.

Використовуються змінні, що мають досить великий розмір (наприклад, масиви великої розмірності), необхідні в одних частинах програми і абсолютно не потрібні в інших.

У процесі роботи програми потрібен масив, список або інша структура, розмір якої змінюється в широких межах і важко передбачуваний.

Коли розмір даних, що обробляються в програмі, перевищує обсяг сегмента даних.

Динамічні структури, за визначенням, характеризуються відсутністю фізичної суміжності елементів структури в пам'яті, непостійністю і непередбачуваністю розміру (числа елементів) структури в процесі її обробки.

Оскільки елементи динамічної структури розташовуються по непередбачуваним адресами пам'яті, адреса елемента такої структури не може бути обчислений з адреси початкового або попереднього елемента. Для встановлення зв'язку між елементами динамічної структури використовуються покажчики, через які встановлюються явні зв'язки між елементами. Таке уявлення даних в пам'яті називається зв'язковим.

Переваги зв'язкового представлення даних - в можливості забезпечення значною мінливості структур:

-розмір структури обмежується тільки доступним об'ємом машинної пам'яті;

-при зміні логічної послідовності елементів структури потрібно не переміщення даних в пам'яті, а тільки корекція покажчиків;

велика гнучкість структури.

Разом з тим, чіткий уявлення не позбавлене і недоліків, основними з яких є наступні:

-на поля, що містять покажчики для зв'язування елементів один з одним, збільшують споживання пам'ять;

-доступ до елементів зв'язної структури може бути менш ефективним за часом.

Останній недолік є найбільш серйозним і саме їм обмежується можливість застосування зв'язкового представлення даних. Якщо в суміжному поданні даних для обчислення адреси будь-якого елементу нам у всіх випадках досить було номера елемента і інформації, що міститься в дескрипторі структури, то для зв'язкового уявлення адреса елемента не може бути обчислений з вихідних даних. Дескриптор зв'язковою структури містить один або кілька покажчиків, що дозволяють увійти в структуру, далі пошук необхідного елемента виконується проходженням по ланцюжку покажчиків від елемента до елементу. Тому чіткий уявлення практично ніколи не застосовується в задачах, де логічна структура даних має вид вектора або масиву - з доступом за номером елемента, але часто застосовується в задачах, де логічна структура вимагає іншої вихідної інформації доступу (таблиці, списки, дерева і т.д .).

Порядок роботи з динамічними структурами даних наступний:

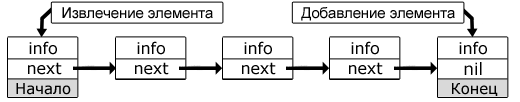
-створити (відвести місце в динамічній пам'яті);

-працювати за допомогою покажчика;

-видалити (звільнити зайняте структурою місце).

* 1. Черга.[4]

Черга - структура даних типу «список», що дозволяє додавати елементи лише в кінець списку, і витягувати їх з його початку. Вона функціонує за принципом FIFO (First In, First Out - «першим прийшов - першим вийшов»), для якого характерно, що всі елементи a1, a2, ..., an-1, an, додані раніше елемента an + 1, повинні бути видалені перш, ніж буде вилучено елемент an + 1. Також чергу може бути визначена як окремий випадок однозв'язного списку, який обслуговує елементи в порядку їх надходження. Як і в «живій» черзі, тут першим буде обслужений той, хто прийшов першим.



Мал. 1.1.1. Черга та її організація.

Стандартний набір операцій (часто у різних авторів він не ідентичний), які виконуються над чергами, збігається з тим, що використовується при обробці стеків:

* додавання елемента;
* видалення елемента;
* читання першого елемента.

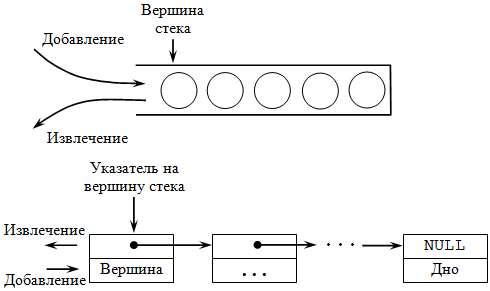
Тільки, якщо стосовно стека в момент додавання або видалення елемента допустимо залучення лише його вершини, то щодо черзі ці дві операції повинні бути застосовані так, як це регламентовано в визначенні цієї структури даних, т. Е. Додавання - в кінець, видалення - з початку. Далі, при реалізації інтерфейсу черзі, список стандартних операцій буде розширено.

Виділяють два способи програмної реалізації черги. Перший з них заснований на базі масиву, а другий на базі покажчиків (зв'язного списку). Перший спосіб - статичний, т. К. Чергу представляється у вигляді простого статичного масиву, другий - динамічний.

* 1. Стек.[5]

Стек (англ. Stack - стопка) - це структура даних, в якій новий елемент завжди записується в її початок (вершину) і черговий читається елемент також завжди вибирається з її початку (рис. 30.1). У стеках використовується метод доступу до елементів LIFO (Last Input - First Output, "останнім прийшов - першим вийшов"). Найчастіше принцип роботи стека порівнюють зі стопкою тарілок: щоб взяти другу зверху, потрібно спочатку взяти верхню.

Стек - це список, у якого доступний один елемент (одна позиція). Цей елемент називається вершиною стека. Взяти елемент можна тільки з вершини стека, додати елемент можна тільки в вершину стека. Наприклад, якщо записані в стек числа 1, 2, 3, то при подальшому витяганні отримаємо 3,2,1.



Мал. 1.2.1 Стек і його організація

Опис стека виглядає наступним чином:

struct імя\_типа {

                 інформаційне поле;

                 адресне поле;

                };

-де інформаційне поле - це поле будь-якого раніше оголошеного або стандартного типу;

-адресне поле - це покажчик на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента стека.

наприклад:

struct list {

             type pole1;

             list \* pole2;

            } Stack;

Стек як динамічну структуру даних легко організувати на основі лінійного списку. Оскільки робота завжди йде з заголовком стека, тобто не потрібно здійснювати перегляд елементів, видалення і вставку елементів в середину або кінець списку, то досить використовувати економічний по пам'яті лінійний односпрямований список. Для такого списку досить зберігати покажчик вершини стека, який вказує на перший елемент списку. Якщо стек порожній, то списку не існує, і покажчик приймає значення NULL.

Опис елементів стека аналогічно опису елементів лінійного односпрямованого списку. Тому оголосимо стек через оголошення лінійного односпрямованого списку:

struct Stack {

              Single\_List \* Top; // вершина стека

             };

. . . . . . . . . .

Stack \* Top\_Stack; // покажчик на вершину стека

Основні операції, вироблені зі стеком:

-створення стека;

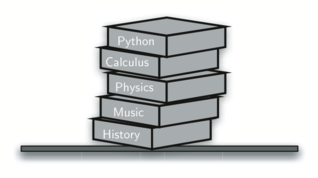
-друк (перегляд) стека;

-додавання елемента в вершину стека;

-витяг елемента з вершини стека;

-перевірка порожнечі стека;

-очищення стека.



Мал. 1.2.2 – Стек з книг

* 1. Односпрямовані і двонаправлені списки.[6]

Списком називається впорядкована множина, що складається з змінного числа елементів, до яких застосовні операції включення, виключення. Список, що відображає відносини сусідства між елементами, називається лінійним.

Довжина списку дорівнює числу елементів, що містяться в списку, список нульової довжини називається порожнім списком. Списки є спосіб організації структури даних, при якій елементи деякого типу утворюють ланцюжок. Для зв'язування елементів в списку використовують систему покажчиків. У мінімальному випадку, будь-який елемент лінійного списку має один покажчик, який вказує на наступний елемент у списку або є порожнім покажчиком, що інтерпретується як кінець списку.

Структура, елементами якої служать записи з одним і тим же форматом, пов'язані один з одним за допомогою покажчиків, що зберігаються в самих елементах, називають пов'язаним списком. У пов'язаному списку елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а покажчиками, що входять до складу елементів списку. Кожен список має особливий елемент, званий покажчиком початку списку (головою списку), який зазвичай за змістом відмінний від інших елементів. В поле покажчика останнього елемента списку знаходиться спеціальний ознака NULL, який свідчить про кінець списку.

Лінійні зв'язні списки є найпростішими динамічними структурами даних. З усього різноманіття пов'язаних списків можна виділити наступні основні:

-односпрямовані (одинзв'язні) списки;

-двонаправлені (двусвязного) списки;

-циклічні (кільцеві) списки.

В основному вони відрізняються видом взаємозв'язку елементів і / або допустимими операціями.

Найбільш простий динамічною структурою є односпрямований список, елементами якого служать об'єкти структурного типу.

Односпрямований (однозв'язний) список - це структура даних, що представляє собою послідовність елементів, в кожному з яких зберігається значення і покажчик на наступний елемент списку (рис. 29.1). В останньому елементі покажчик на наступний елемент дорівнює NULL.



Мал. 1.3.1. Лінійний односпрямований список

Опис найпростішого елемента такого списку виглядає наступним чином:

struct імя\_тіпа {інформаційне поле; адресне поле; };

де інформаційне поле - це поле будь-якого, раніше оголошеного або стандартного, типу;

адресне поле - це покажчик на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку.

наприклад:

struct Node {

             int key; // інформаційне поле

             Node \* next; // адресне поле

            };

Інформаційних полів може бути кілька. Наприклад:

struct point {

              char \* name; // інформаційне поле

              int age; // інформаційне поле

              point \* next; // адресне поле

             };

Кожен елемент списку містить ключ, який ідентифікує цей елемент. Ключ зазвичай буває або цілим числом, або рядком.

Основними операціями, здійснюваними з односпрямованим списками, є:

* створення списку;
* друк (перегляд) списку;
* вставка елемента в список;
* видалення елемента зі списку;
* пошук елемента в списку
* перевірка порожнечі списку;
* видалення списку.

Особливу увагу слід звернути на те, що при виконанні будь-яких операцій з лінійним односпрямованим списком необхідно забезпечувати позиціонування будь-якого покажчика на перший елемент. В іншому випадку частина або весь список буде недоступний.

* 1. Бінарні дерева. [7]

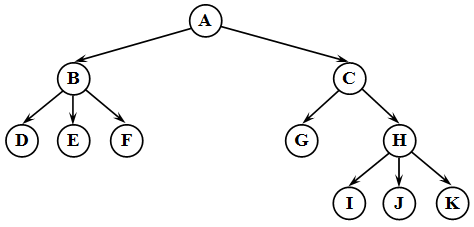
Дерево - це структура даних, що представляє собою сукупність елементів і відносин, що утворюють ієрархічну структуру цих елементів (рис. 31.1). Кожен елемент дерева називається вершиною (вузлом) дерева. Вершини дерева з'єднані спрямованими дугами, які називають гілками дерева. Початковий вузол дерева називають коренем дерева, йому відповідає нульовий рівень. Листям дерева називають вершини, в які входить одна гілка і не виходить жодної гілки.

Кожне дерево має такі властивості:

1. існує вузол, в який не входить ні однієї дуги (корінь);

2. в кожну вершину, крім кореня, входить одна дуга.

Дерева особливо часто використовують на практиці при зображенні різних ієрархій. Наприклад, популярні генеалогічні дерева.



Мал. 1.4.1. Дерево

Всі вершини, в які входять гілки, що виходять з однієї загальної вершини, називаються нащадками, а сама вершина - предком. Для кожного предка може бути виділено кілька. Рівень нащадка на одиницю перевищує рівень його предка. Корінь дерева не має предка, а листя дерева не мають нащадків.

Висота (глибина) дерева визначається кількістю рівнів, на яких розташовуються його вершини. Висота порожнього дерева дорівнює нулю, висота дерева з одного кореня - одиниці. На першому рівні дерева може бути тільки одна вершина - корінь дерева, на другому - нащадки кореня дерева, на третьому - нащадки нащадків кореня дерева і т.д.

Піддерево - частина древообразная структури даних, яка може бути представлена ​​у вигляді окремого дерева.

Ступенем вершини в дереві називається кількість дуг, яке з неї виходить. Ступінь дерева дорівнює максимальному ступені вершини, що входить в дерево. При цьому листям в дереві є вершини, що мають ступінь нуль. За величиною ступеня дерева розрізняють два типи дерев:

– виконавчі - ступінь дерева не більше двох;

– сильноветвящиеся - ступінь дерева довільна.

Впорядковане дерево - це дерево, у якого гілки, що виходять з кожної вершини, впорядковані за певним критерієм.

Дерева є рекурсивними структурами, так як кожне піддерево також є деревом. Таким чином, дерево можна визначити як рекурсивну структуру, в якій кожен елемент є:

– або порожній структурою;

– або елементом, з яким пов'язано кінцеве число піддерев.

Дії з рекурсивними структурами найзручніше описуються за допомогою рекурсивних алгоритмів.

Списочное уявлення дерев засноване на елементах, відповідних вершин дерева. Кожен елемент має поле даних і два поля покажчиків: покажчик на початок списку нащадків вершини і покажчик на наступний елемент у списку нащадків поточного рівня. При такому способі подання дерева обов'язково слід зберігати покажчик на вершину, що є коренем дерева.

Для того, щоб виконати певну операцію над усіма вершинами дерева необхідно все його вершини переглянути. Таке завдання називається обходом дерева.

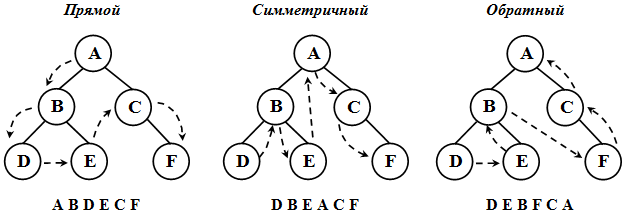
Обхід дерева - це впорядкована послідовність вершин дерева, в якій кожна вершина зустрічається тільки один раз.

При обході все вершини дерева повинен відвідувати у певному порядку. Існує кілька способів обходу всіх вершин дерева. Виділимо три найбільш часто використовуваних способу обходу дерева (рис. 31.2):

– прямий;

– симетричний;

– зворотний.

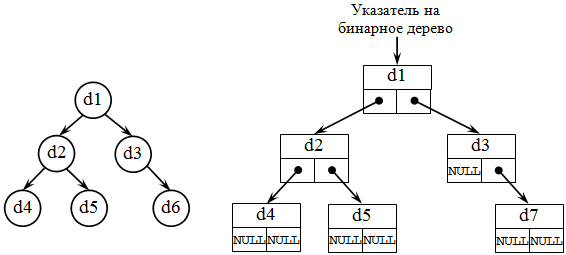


Мал. 1.4.2. обходи дерев

Існує велике різноманіття деревовидних структур даних. Виділимо найпоширеніші з них: бінарні (двійкові) дерева, червоно-чорні дерева, В-дерева, АВЛ-дерева, матричні дерева, змішані дерева і т.д.

Бінарні дерева є деревами зі ступенем не більше двох.

Бінарне (двійкове) дерево - це динамічна структура даних, що представляє собою дерево, в якому кожна вершина має не більше двох нащадків (рис. 31.3). Таким чином, бінарне дерево складається з елементів, кожен з яких містить інформаційне поле і не більше двох посилань на різні бінарні піддерева. На кожен елемент дерева є рівно одна посилання.



Мал. 1.4.3. Бінарне дерево і його організація

Кожна вершина бінарного дерева є структурою, що складається з чотирьох видів полів. Вмістом цих полів будуть відповідно:

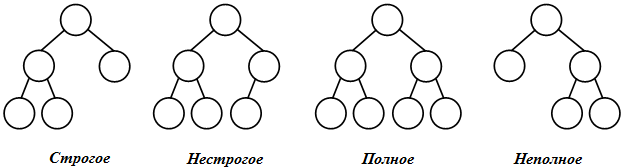
– інформаційне поле (ключ вершини);

– службове поле (їх може бути декілька або жодного);

– покажчик на ліве піддерево;

– покажчик на праве піддерево.

За ступенем вершин бінарні дерева діляться на (рис. 31.4):



Мал. 1.4.4.

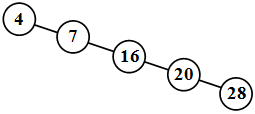
– строгі - вершини дерева мають ступінь нуль (у листя) або два (у вузлів);

* несуворі - вершини дерева мають ступінь нуль (у листя), один або два (у вузлів).

У загальному випадку у бінарного дерева на k -му рівні може бути до 2k-1 вершин. Бінарне дерево називається повним, якщо воно містить тільки повністю заповнені рівні. В іншому випадку воно є неповним.

Дерево називається збалансованим, якщо довжини всіх шляхів від кореня до зовнішніх вершин рівні між собою. Дерево називається майже збалансованим, якщо довжини всіляких шляхів від кореня до зовнішніх вершин відрізняються не більше, ніж на одиницю.

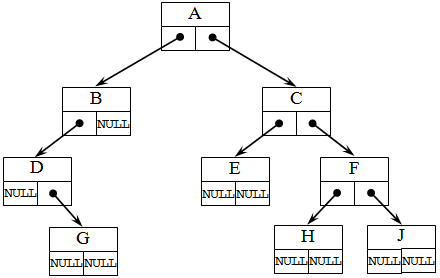
Бінарне дерево може являти собою порожня множина. Бінарне дерево може виродитися в список (рис. 31.5).



Мал. 1.4.5. Список як окремий випадок бінарного дерева

Структура дерева відбивається у вхідному потоці даних так: кожній вводиться порожній зв'язку відповідає умовний символ, наприклад, '\*' (зірочка). При цьому спочатку описуються ліві нащадки, потім, праві. Для структури бінарного дерева, представленого на рисунку 6, вхідний потік має вигляд:

ABD \* G \*\*\* CE \*\* FH \*\* J \*\*.



Мал. 1.4.6. Адресація в бінарному дереві

Бінарні дерева можуть застосовуватися для пошуку даних в спеціально побудованих деревах (бази даних), сортування даних, обчислень арифметичних виразів, кодування (метод Хаффмана) і т.д.

# 2. ПРОГРАМНА РЕАЛІЗАЦІЯ

## 2.1. Основні відомості

Поставленою задачею було реалізувати динамічну структуру даних – базу даних автомобілів у автосалоні, що містить наступну інформацію

марка автомобіля;

– пробіг;

– назва фірми-виробника;

– рік випуску;

– об’єм двигуна;

– ціна

З можливістю:

– додавання елементів;

– видалення елементів;

– коригування даних;

– виведення всіх даних;

–  початкове формування даних про всі автомобілі у вигляді таблиці;

* висновок даних про автомобілі старше заданого року випуску;
* пошук автомобілів заданої фірми виробника;
* пошук автомобілів, чий пробіг перевищує задану величину;
* сортування всіх автомобілів за обсягом двигуна;

– сортування всіх автомобілів за ціною.

Для виконання цих функцій була створена динамічна структура бінарного дерева «CBinaryNode», яка містить такі поля:

– пробіг;

– назва фірми-виробника;

– рік випуску;

– об’єм двигуна;

– ціна

– вказіваник на лівого нащадка

* вказіваник на правого нащадка
* вказіваник на батька

Програма складається з 29 функцій:

int menu ();

struct CBinaryNode \* addnode (struct CBinaryNode \*ptr, struct CBinaryNode \*tree);

int treeprint (struct CBinaryNode \*tree);

struct CBinaryNode \*del\_elem (struct CBinaryNode \*tree, struct CBinaryNode \*del\_elem );

struct CBinaryNode \*root\_del (struct CBinaryNode \*del\_elem);

struct CBinaryNode \*correct (struct CBinaryNode \*tree);

void sort\_price ( struct database \*tree void sort\_capacity ( struct database \*tree );

int year\_output ( struct CBinaryNode \*treeint company\_output( struct CBinaryNode \*tree);

int mileage\_output ( struct CBinaryNode \*tree

void save(struct CBinaryNode \*tree);

void deltree\_mem (struct CBinaryNode \*tree);

///основні функції

int one\_el\_print( struct CBinaryNode \*tree, int num);

struct CBinaryNode \*input ( struct CBinaryNode \*tree );

struct database \*sort\_mas ( struct CBinaryNode \*tree );

struct CBinaryNode \*TREE\_scrl ( struct CBinaryNode \*tree, int del\_number);

///допоміжні функції

int yes\_no();

void errors (int err\_x);

void finish (int succ\_x);

void header();

void privet ();

void SetColor (int text, int Fon);

void TITUL ();

///косметичні функції

struct CBinaryNode \*empty\_check ( struct CBinaryNode \*tree );

int num\_check (int razmer);

float float\_num\_check (int razmer2);

char \*str\_check (int razmer3, int if\_integer);

///функції перевірки

Покажчики \*left, \*right, \*parent вказують на правого та лівого нащадків та предка відповідно. \*root = NULL - структуру дерева, \*ptr = & NODES - структура вузла, \*current - батьківська змінна, яка вказує на попередній вузол.

У програмі реалізовано створення двох файлів з даними. data - файл для читання і запису даних і Table - формування даних у вигляді таблиці. А так-же два додаткових файли для збереження відсортованих даних.

## 2.2 Опис роботи функцій

**int main ()**

У головній функції main, після налагодження користувальницької консолі під потреби програми, на екран виводиться привітання викликом функцій TITUL() і privet(). Програма надає користувачеві можливість продовжити роботу з попереднім збереженням файлу. Залежно від відповіді користувача файл або відкривається для читання та заносить данні в пам'ять бінарного дерево через виклик відповідної функції, або файл очищається. (В разі читання данних з порожнього файлу – виводиться ошибка і питання про продовження роботи).

Далі в функції реалізована функція циклічного виклику меню - вихід з циклу відбувається тільки при введенні 0. За допомогою функції switch був реалізований вибір між наступними діями.

| 1 | Відображення всіх даних про автомобілі.

| 2 | Додавання даних про новий автомобіль.

| 3 | Видалення даних про автомобіль.

| 4 | Зміна даних про автомобіль.

| 5 | Вивід даних про автомобіль старше заданого року випуску.

| 6 | Вивід даних про автомобілі заданої фірми виробника.

| 7 | Вивід даних про автомобілі, чий пробіг перевищує задану величину.

| 8 | Сортування всіх автомобілів за обсягом двигуна.

| 9 | Сортування всіх автомобілів за ціною.

| 10 | Зберегти дані

| 0 | Вихід з програми

1 – Перевірка файлу на порожнечу, якщо дерево порожнє – вивід помилки. Вивід на екран всього дерева у вигляді таблиці викликом відповідних функцій.

2 - Виклик функції додавання дерева.

3 - Перевірка файлу на порожнечу, якщо дерево порожнє – вивід помилки. Якщо ні – вивід дерева і введення номера видаляється елемента. Якщо введене число відповідає існуючому елементу відбувається виклик функції проходу по дереву для пошуку потрібного адреси. Після повторного питання про видалення відбувається виклик функцій видалення (виклик фукнції видалення кореня в разі введення користувачем 1). Якщо користувач ввів «НІ» в питанні про видалення елемента - пропонується вибрати інший елемент.

4 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Якщо ні - вивід дерева і введення номера змінюваного елемента. Якщо введене відповідає існуючому елементу відбувається виклик функції проходу по дереву для пошуку потрібного адреси. Після повторного питання про зміну даних відбувається виклик функції коригування. Якщо користувач ввів «НІ» в питанні про видалення елемента - пропонується вибрати інший елемент.

5 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Через функцію перевірки введених значень користувач вводить потрібний рік, після чого викликається функція виведення авто старше заданого року - якщо таких немає - вивід помилки.

6 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Через функцію перевірки введених значень користувач вводить потрібну фірму, після чого викликається функція виведення авто даної фірми - якщо таких немає - вивід помилки.

7 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Через функцію перевірки введених значень користувач вводить потрібний пробіг, після чого викликається функція виведення авто з пробігом вище даного числа - якщо таких немає - вивід помилки.

8 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Виклик функції заношеними даних в масив для сортування. Виклик самої функції сортування за обсягом двигуна.

9 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Виклик функції заношеними даних в масив для сортування. Виклик самої функції сортування по ціні.

10 - Перевірка файлу на порожнечу, якщо дерево порожнє - вивід помилки. Виклик функції відображенні дерева на екран. Після запитання про збереження даних, файл або відкривається з видаленням і викликається функція збереження, або повертає користувача в меню.

0 - Вивід питання для уточнення про вихід з програми. Якщо так - питання про збереження даних в файл (очищення файлу і виклик функція збереження, або завершення без збереження), якщо немає - повернення в меню.

Після чого викликається функція очищення пам'яті і програма закривається.

**int one\_el\_print (struct CBinaryNode \* tree, int num)**

*/// Друк одного еллемента*

Послідовний вивід на екран всіх даних про один елемент у вигляді таблиці. В якості аргументів функція приймає номер виведеного елемента і покажчик на структуру дерева.

**int treeprint (struct CBinaryNode \* tree)**

*/// Висновок вузлів дерева (обхід в інфіксной формі)*

Умовним оператором if при кожному виклику функції перевіряється не порожній вузол. Якщо немає - вивід одного елемента і рекурсивний виклик самої себе для лівого піддерева і для правого піддерева. Повертає кількість елементів. В якості аргументів функція приймає покажчик на структуру дерева.

**struct CBinaryNode \* addnode (struct CBinaryNode \* ptr, struct CBinaryNode \* tree)**

*/// Додавання вузлів в дерево*

В якості аргументів функція приймає покажчик на структуру дерева і покажчик на вузол. Якщо дерево порожнє - виділяємо пам'ять і заносимо дані в вузол. Гілки іниціализуємо порожнечею і створюємо змінну - покажчик на батька. Якщо дерево не пусте в залежності від випадку виконуються наступні дії:

- якщо вводиться ціна менша за ціну в кореневому вузлі, рекурсивно викликаємо функцію з покажчиком на лівий елемент і повторно виконуємо дії до тих пір, поки не зустрінеться порожній вузол - після чого дані вводяться, а гілки іниціализуються порожнечею. Створюємо змінну - покажчик на батька.

- якщо вводиться ціна більша за ціну в кореневому вузлі, рекурсивно викликаємо функцію на цей раз з покажчиком на правий елемент і повторно виконуємо дії до тих пір поки не зустрітися порожній вузол - після чого дані вводяться, а гілки ініціалізуються порожнечею. Створюємо змінну - покажчик на батька.

- якщо ціна нового елемента дорівнює ціні кореневого елемента, рекурсивно викликаємо функцію з покажчиком на правий елемент і повторно виконуємо дії до тих пір поки не зустрітися порожній вузол - після чого дані вводяться, а гілки ініціалізуються порожнечею. Створюємо змінну - покажчик на батька.

Тим самим реалізуємо бінарне дерево пошуку

В кінці функції додавання - інкрементується змінна загального числаелементів і повертається показчик на елемент**.**

**struct CBinaryNode \* del\_elem (struct CBinaryNode \* tree, struct CBinaryNode \* del\_elem)**

*/// Видалення*

В якості аргументів функція приймає покажчик на структуру дерева і покажчик на адресу видаляється елемента. На початку функції створюється покажчик на батька видаляється елемента.

Функція видалення працює для 3 випадків:

1) Вилучення листа. Якщо елемент - лист, то розглядаються два випадки - видаляється елемент-ліве поддервео для свого батька, або праве. В обох випадках об'єкт видаляється і гілка ініціалізується порожнечею через покажчик на відповідне поддерево батька.

2) Видаляємий елемент має одного нащадка. Розглядаються два випадки в залежності від того, яке піддерево є у видаляємого елемента. Для обох випадків перевіряється яким піддеревом є видаляємий елемент щодо батьківського і відбувається перепрів’язка - покажчик на піддерево батьківського елемента (лівому або правому піддерев) прив'язується до поддерва видаляємого елемента (лівому або правому)

3) Видаляємий елемент має двох нащадків. Ми оголошуємо змінну, яка стане заміною для видаляємого елемента. Далі розглядаємо два випадки: якщо у видаляємого елемента в правому піддереві немає лівого піддерева в залежності від того, яким піддерево є сам видаляємий елемент - просто ставимо на його місце його правий елемент. Робимо прив'язку так, що лівий елемент видаляємого став лівим нащадком для нового елемента, а батько видаляєтмого елемента став батьком нового елемента. В іншому випадку ми циклічно рухаємося по лівому піддерево правого піддерева видаляємого елемента до тих пір поки не зустрінемо елемент, у якого немає лівого нащадка - який стає заміною видаляється елемента. Елемент який став заміною ініціалізуємо порожнечею і в залежності від того, яким піддеревом для батьків є видаляємий елемент, ставимо на місце видаленого, відповідно прив'язуючись до батьків і до піддерев видаляємого елемента. Так само і у випадку якщо у заміни є елементи правого піддерева - ми прив'язуємо їх до батька елемента, який замінюємо. Звільняємо пам'ять і декріментіруем змінну загальної кількості автомобілів. Повертаємо покажчик на дерево.

**struct CBinaryNode \* root\_del (struct CBinaryNode \* del\_elem)**

*/// Видалення кореня*

Видалення першого елемента бінарного дерева - кореня, відбувається за тим-же принципом розгляду трьох випадків.

1) Якщо у кореня немає нащадком - звичайне звільнення пам'яті і повернення змінної заміни.

В інших випадках відбувається поіск вдалою заміни.

2) Якщо у кореня тільки один - правий нащадок, то ми перевіряємо чи є у нього в свою чергу лівий нащадок. Якщо немає - заміною просто стає правий елемент правого поддервеа кореня, якщо є, то ми рекурсивно йдемо по лівому піддереву правого елемента кореня до тих пір поки не дійдемо до елемента без лівого нащадка -він і буде вдалою заміною, після чого перевіряємо - чи є у заміни правий нащадок, якщо є, взаємно прив'язуємо його до батьків заміни, якщо немає - ініціалізуємо місце в пам'яті заміни порожнечею. Далі ставимо заміну на місце кореня ініціалізувавши його батька порожнечею і прив'язуючи до нього правий нащадок, звільняємо пам'ять і повертаємо заміну. Дзеркальні дії проводяться і для випадку коли у кореня тільки лівий нащадок.

3) У разі якщо у кореня є два сина перевіряється на наявність лівий елемент правого поддерва кореня.

Якщо немає - на місце кореня постає його правий нащадок, батько якого инициализируется порожнечею, лівий елемент переприв’язується і батьком лівого нащадка стає заміна. Звільняється пам'ять і повертається заміна.

Якщо є - заміна рекурсивно шукається в лівому поддереве правого нащадка кореня до тих пір поки не зустрінеться елемент без лівого нащадка - він і станемо заміною. Якщо у заміни є правий нащадок - його батьком стає батько заміни. Якщо немає - батько заміни онулируется прив'язку. Після чого заміна встановлює прив'язки до лівого і правого елементу кореня, і ініціалізує свого батька в порожнечу. Для правого і лівого елемента кореня заміна - стає батьком. Звільняється пам'ять видаляється елемента і повертається заміна.

**struct CBinaryNode \*correct (struct CBinaryNode \*tree);**

*/// Коригування даних*

В якості аргументів функція пріймає адресою корректируемого елемента. Залежно від введеної в switch цифри - інформація про даний параметрі введеного заздалегідь елемента переписується. У разі введення 7 віклікається функція введення даних з відповіднім аргументом адреси. Програма буде просити користувача вводити нові дані поки той не ввдет 0 і не вийде з циклу. Возвращается покажчик на коректованій елементів.

**void save (struct CBinaryNode \* tree)**

*/// Збереження*

Як аргумент функція приймає покажчик на страктуру даних. Відкриває файл з даними і з таблицею для запису у кінець. При першому виклику функції в файл з таблицею заноситься заголовок. Поки не зустрінеться порожній вузол функція буде рекурсивно викликати сама себе, з кожної ітерацией збільшуючи номер заносимого в файли елемента на 1.

**struct database \* sort\_mas (struct CBinaryNode \* tree)**

Функція приймає в якості аркумента покажчик на страктуру даних.

Функція є допоміжною для сортування даних. Головним її завданням є копіювання даних з пам'яті в окремий масив, який надалі можна буде сортувати. Масив створюється на основі бази даних про автомобілі. Функція буде рекурсивно повторюватися до тих пір поки не зустріне порожній вузол. З кожної итерацией лічильник збільшується на одиницю. Функція повертає масив no\_sort.

**void sort\_price (struct database \* tree)**

*/// Сортування за ціною sorting*

Функція приймає покажчик на базу даних елементів (масив створений функцією sort\_mas). Алгоритм сортування - бульбашкова сортування. Реалізована шляхом створення двоїчного циклу. В ході роботи кожен елемент порівнюється за ціною з усіма іншими елементами. Якщо ціна більше, елемент стає на місце елемента з меншою ціною внаслідок чого отримуємо відсортований за зростанням масив. Після сортування результат виводиться на екран і користувачеві пропонується зберегти результат в файл шляхом очищення старого файлу сортування і внесення нових даних в нього.

**void sort\_capacity (struct database \* tree)**

*/// Сортування за об'ємом двигуна*

Функція приймає покажчик на базу даних елементів (масив створений функцією sort\_mas). Алгоритм сортування - бульбашкова сортування. Реалізована шляхом створення двоїчного циклу. В ході роботи кожен елемент порівнюється по пробігу з усіма іншими елементами. Якщо пробіг більше, елемент стає на місце елемента з меншим пробігом внаслідок чого отримуємо відсортований за зростанням масив. Після сортування результат виводиться на екран і користувачеві пропонується зберегти результат в файл шляхом очищення старого файлу сортування і внесення нових даних в нього.

**int year\_output (struct CBinaryNode \* tree)**

*/// Вивід даних про автомобіль старше заданого року випуску.*

Функція як аргумент приймає покажчик на корінь дерева. Функція буде викликати себе рекурсивно поки не зустріне порожній вузол. Якщо на шляху функції зустрінеться елемент з більшим роком, він вивидеться на екран (якщо перший раз - з заголовком). З кожною ітерацією збільшуючи кількість елементів на одиницю. Повертає кількість.

**int company\_output (struct CBinaryNode \* tree)**

*/// Вивід даних про автомобілів заданої фірми виробника.*

Функція як аргумент приймає покажчик на корінь дерева. Функція буде викликати себе рекурсивно поки не встетіть порожній вузол. Якщо на шляху функції зустрінеться елемент з компанією - відповідної до заздалегідь введеної, він вивидеться на екран (якщо перший раз - з заголовком). З кожною ітерацією збільшуючи кількість елементів на одиницю. Повертає кількість.

**int mileage\_output (struct CBinaryNode \* tree)**

*/// Вивід даних про автомобілів чий пробіг перевищує задану величину.*

Функція як аргумент приймає покажчик на корінь дерева. Функція буде викликати себе рекурсивно поки не встетіть порожній вузол. Якщо на шляху функції зустрінеться елемент з більшим пробігом, він вивидеться на екран (якщо перший раз - з заголовком). З кожною ітерацією збільшуючи кількість елементів на одиницю. Повертає кількість.

**int yes\_no ()**

Допоміжна функція призначена для спрощення взаємодії користувача з програмою. Якщо користувач ввів Y, повертає 1. Якщо користувач ввів N, повертає 2. Якщо щось інше - виводить помилку і рекурсивно запускається ще раз.

**void SetColor (int Text, int Fon)**

Функція спрощує роботу зі зміною кольору в консолі. Приймає два значення. Перше - колір тексту. Друге - колір фону. Після чого за допомогою функції з бібліотеки <windows.h> змінює колір.

**void privet ()**

Функція для спрощення виведення тексту на екран. Відкриває файл для читання. Зчитує рядки з файлу в масив, поки не дійде до кінця файлу і виводить їх на екран. Після чого закриває файл. Виводить вітання користувача на початку програми.

**void TITUL ()**

Функція для спрощення виведення тексту на екран. Відкриває файл для читання. Зчитує рядки з файлу в масив, поки не дійде до кінця файлу і виводить їх на екран. Після чого закриває файл. Виводить титульний лист програми перед початком роботи.

**int menu ()**

Функція для спрощення виведення меню на екран. Відкриває файл для читання. Зчитує рядки з файлу в масив. Зчитує рядки з файлу поки не дійде до кінця файлу і виводить їх на екран. Після чого закриває файл. Виводить меню для взаємодії з користувачем. Після чого чекає відповіді користувача - введення числа відповідного певної дії.

**void errors (int err\_x)**

Функція для спрощення виводу помилок на екран. Приймає номер помилки і виводить відповідне повідомлення червоним кольором, після чого закривається.

**void header ()**

Функція виводить заголок таблиці даних на екран. Скорочує код і спрощує роботу з ним.

**void finish (int succ\_x)**

Функція приймає номер відповідного закінчення кейса з меню і виводить відповідне повідомлення зеленим кольором, закривається, або виходить з програми в разі якщо аргумент 0. Чекаємо відповіді користувача і чистить екран.

**struct CBinaryNode \* empty\_check (struct CBinaryNode \* tree)**

Як аргумент функція приймає покажчик на корінь дерева. Перевіряє наявність кореня - якщо немає - дерево пусте - вивід повідомлення.

**struct CBinaryNode \* TREE\_scrl (struct CBinaryNode \* tree, int del\_number)**

Допоміжна функція для видалення і коригування даних. Шукає адресу елемента з зазделегіть введеним номером. У якості аргументу приймає покажчик на корінь і номер елемента з яким будуть зроблена дія. Поки не зустрінеться порожній вузол функція буде рекурсивно повторюватися і порівнювати номер кожної ітерації з потрібним. Повертає адресу елемента.

**int num\_check (int razmer)**

Як аргумент функція приймає максимальний розмір рядка. Перевірка числа типу int відбувається за алгоритмом ініціалізації циклу, в який кожен раз користувач вводить новий рядок і вихід з якого виконується тільки якщо помилок в введенні немає. Спочатку перевіряється розмір рядка порівнянням довжини з максимальним значенням (якщо більше - виводиться помилка). Якщо перша перевірка пройдена успішно - відбувається перевірка рядка на наявність у ньому букв - проходом по всьому рядку і перевіркою кожного елемента функцією isdigit; Якщо рядок пройшла перевірку - він переводиться в тип даних int за допомогою функції atoi і повертає це значення.

**float float\_num\_check (int razmer2)**

Як аргумент функція приймає максимальний розмір рядка. Перевірка числа з плаваючою комою відбувається за алгоритмом ініціалізації циклу, в який кожен раз користувач вводить новий рядок і вихід з якого виконується тільки якщо помилок в введенні немає. Спочатку перевіряється розмір рядка порівнянням довжини з максимальним значенням (якщо більше - виводиться помилка). Якщо перша перевірка пройдена успішно - відбувається перевірка рядка на на явність у ньому букв: в циклі проходу по всьому рядку проходить пошук крапки чи коми, після чого перевіряються значення до і після неї функцією isdigit; якщо число ціле - перевіряється весь рядок без поділу. Якщо рядок пройшов перевірку – він переводиться в тип даних float за допомогою функції atof і повертає це значення.

**char \* str\_check (int razmer3, int if\_integer)**

Як аргумент функція приймає 2 значени: максимальний розмір рядка і 1 в разі якщо в рядку допускаються цифри і 0 в разі якщо цифри заборонені.

Якщо 0 - ыныцыалызуэться цикл, в який кожен раз користувач вводить новий рядо і вихід з якого виробляється тільки якщо рядок проходить перевірку (кожен раз рядок перевіряється на вміст в ньому цифр від 0 до 9 і в разі наявності - виводиться помилка). Далі розмір рядка порывнюэться з максимальним. Якщо рядок більше - повідомлення про помилку і рекурсивний виклик функції.

Якщо 1 - перевіряється тільки довжина рядка методом циклу, в який кожен раз користувач вводить нове значніє і вихід з якого відбувається тільки тоді, коли перевірка пройдена успішно.

**struct CBinaryNode \* input (struct CBinaryNode \* tree)**

Функція приймає покажчик на місце в пам'яті куди буде введено данні. Після виведення підказок через функції перевірки введених значень користувач вводить дані про автомобілі, кожен раз очищаючи потік введення. Повертає покажчик на введений елемент.

**void deltree\_mem (struct CBinaryNode \* tree)**

Функція звільняє пам'ять всього дерева. Як аргумент приймає покажчик на корінь. Викликає себе рекурсивно до тих пір, поки не видалить все дерево і не зустріне порожній вузол**.**

# ІНСТРУКЦІЯ КОРИСТУВАЧА

Дана програма дозволяє:

-Відображення всіх даних про автомобілі.

-Додавання даних про новий автомобіль.

-Видалення даних про автомобіль.

-Зміна даних про автомобіль.

-Вивід даних про автомобіль старше заданого року випуску.

-Вивід даних про автомобілі заданої фірми виробника.

-Вивід даних про автомобілі, чий пробіг перевищує задану величину.

-Сортування всіх автомобілів за обсягом двигуна.

-Сортування всіх автомобілів за ціною

Після запуску програми вас зустріне титульний ліст роботи.

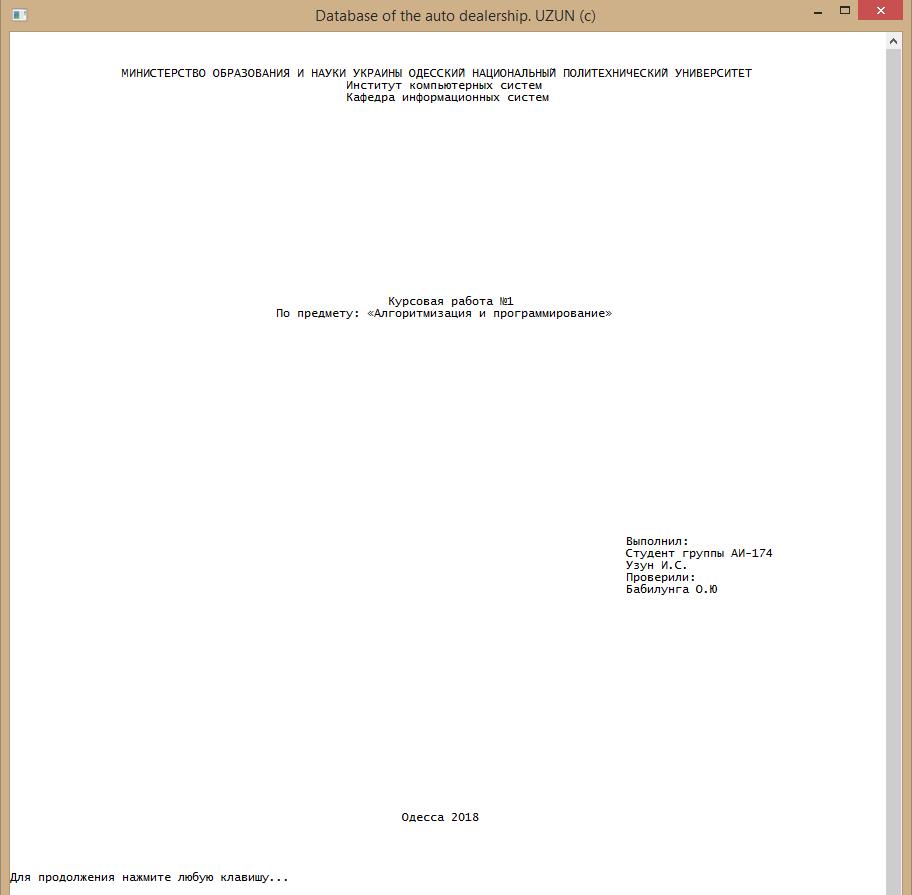


Рисунок 3.1 – Титульний лист програми.

После натіскання будь-якої клавіші вам запропонують продовжити роботу з останнім збереженням файлу або очистити файл і почати роботу з нуля.

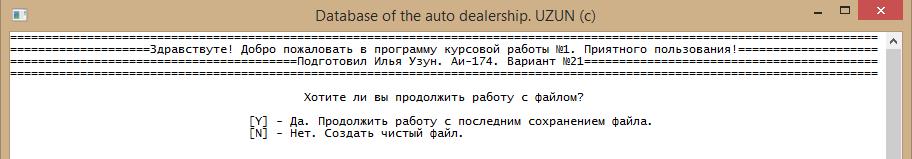


Рисунок 3.2 – Початок роботи.

Якщо ви введете 'Y', але файл виявиться порожній, вас запитають чи хочете ви завершити роботу або продовжити далі.

Після вибору Y або N на екрані відобразиться головне меню програми.

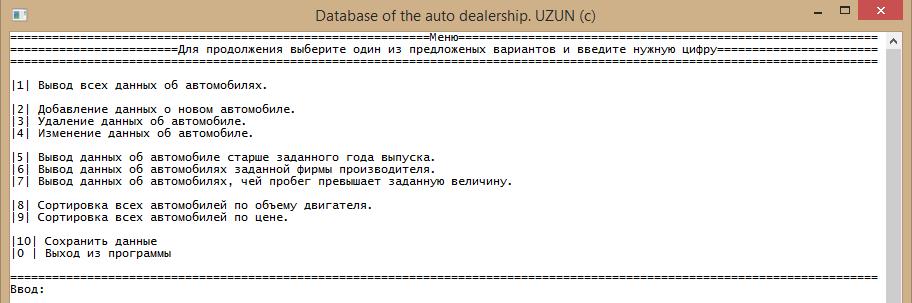


Рисунок 3.3 – Головне меню.

Залежно від введеної цифри буде виконано дію відповідне до цієї цифри. При введенні інших цифр або букв буде виведено повідомлення про помилку і виведений повторний запит на введення.

Якщо дані ще не були введені, при виклику всіх функцій крім 2 і 0 буде про це повідомлено.



Рисунок 3.4 –Помилка про відсутність даних.

При виборі пункту 2 в меню буде організовано розміщення інформації про автомобіьль в бінарне дерево: натіскайте Enter и послідовно заповняйті Кожне поле. Після заповнення всіх даних буде виведено повідомлення про успішне додаванні елемента і вас буде переведено до головного меню.

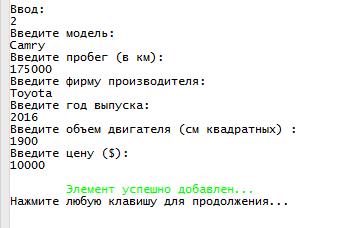


Рисунок 3.5 – Введення нових даних.

Якщо буде введена інформація, яка не відповідає обмеженням введення – програма виведе помилку та запросить інформацію в користувача знову.

При віборі пункту 1 в меню буде виведена таблиця з уже введеними даними.

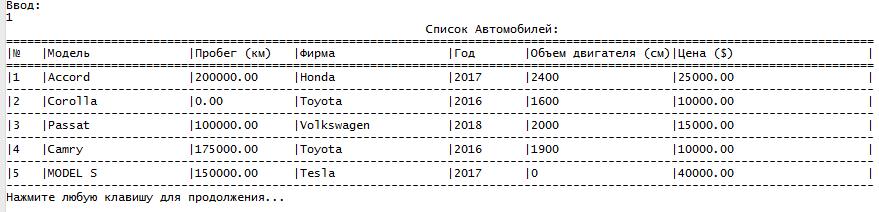


Рисунок 3.6 – Вивід таблиці.

При виборі пункту 5 буде запропоновано ввести рік - після чого таблиця з автомобілями старше даного року буде виведена на екран. Якщо таких автомобілів немає - про це буде сказано. У разі неправильного введення буде виведена помилка і викликаний повторний запит на введення.



Рисунок 3.7 – Вивід авто старше введеного року

При виборі пункту 6 буде запропоновано ввести марку - після чого таблиця з автомобілями цієї фірми буде виведена на екран. Якщо таких автомобілів немає - про це буде сказано. У разі неправильного введення буде виведена помилка і викликаний повторний запит на введення.

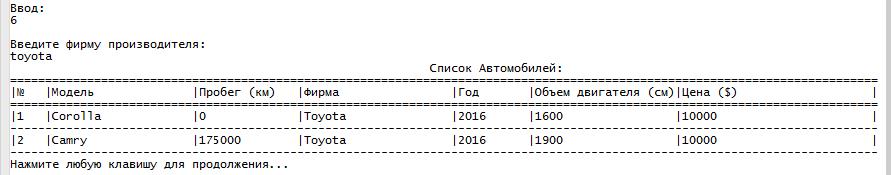


Рисунок 3.8 – Вивід авто введеної фірми.

При виборі пункту 7 буде запропоновано ввести пробіг автомобіля - після чого таблиця з автомобілями, чий пробіг більше, буде виведена на екран. Якщо таких автомобілів немає - про це буде сказано. У разі неправильного введення буде виведена помилка і викликаний повторний запит на введення.

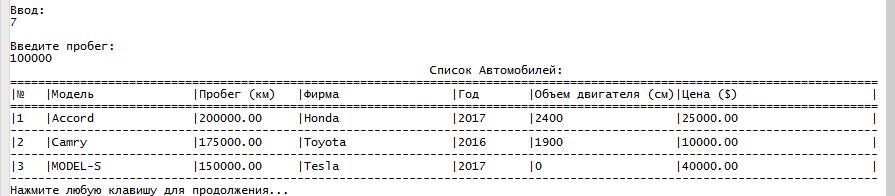


Рисунок 3.9 – Вивід авто з більшим пробігом.

При виборі пункту 8 на екран буде виведена таблиця з усіма даними відсортованими за обсягом двигуна (за зростанням). Після чого вам запропонують зберегти відсортовані дані в файл.

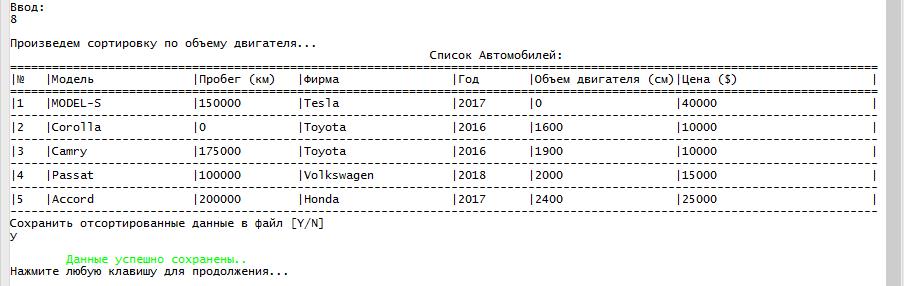


Рисунок 3.10 – Вивід авто відсортований за зростанням об’єму двигуна.

При виборі пункту 8 на екран буде виведена таблиця з усіма даними відсортованими за зростанням ціни. Після чого вам запропонують зберегти відсортовані дані в файл.

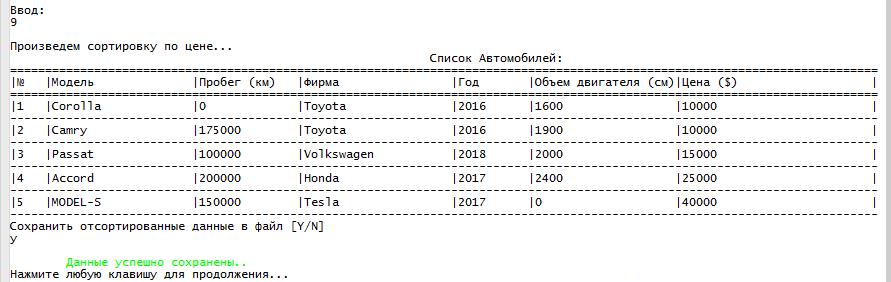


Рисунок 3.11 – Вивід авто відсортований за зростанням ціни.

При виборі пунка 3 буде виведена вся таблиця і запропоновано вибрати один елемент, який ви хочете видалити. Після уточнення ваших намеряній файл буде видалений (якщо відповідь позитивна) або вас повернуть в головне меню (якщо відповідь негативна).

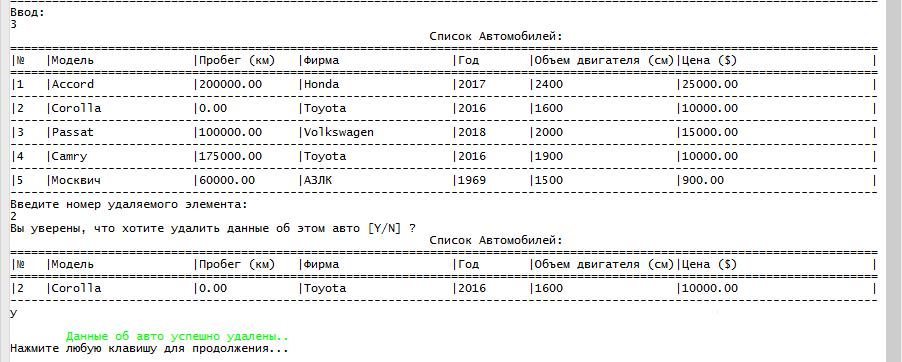


Рисунок 3.12 –Успішне видалення елемента.

При виборі пункту 10 вам запропонують зберегти дані в файл.

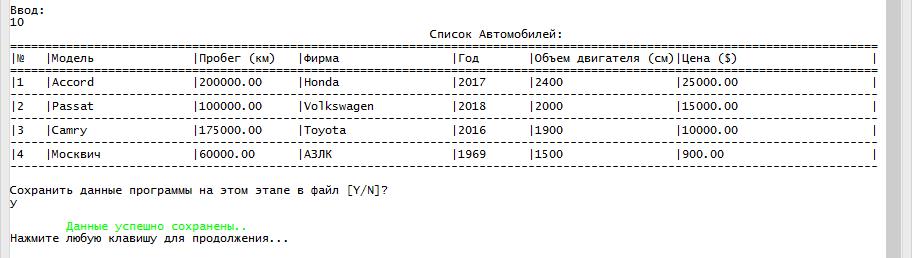


Рисунок 3.13 – Запит на збереження файлу.

При виборі пунка 4 буде виведена вся таблиця і запропоновано вибрати один елемент, дані якого ви хотіли б змінити. Після уточнення ваших намірів вас попросять вибрати номер параметру елемента із запропонованих, в дані якого ви хочете внести зміни (якщо відповідь позитивна), або запит на введення нового елемента (якщо відповідь негативна). Після вибору введіть нове значення і натисніть Enter. Операція буде повторювати до тих пір поки ви не введете 0 в меню. При виборі 7 - відбувається нове введення всіх даних про автомобіль. У разі відмови від корегування вас повернуть в меню (якщо відповідь негативна). У разі введення номера неіснуючого елемента буде виведена помилка та запит на введення нового елемента.

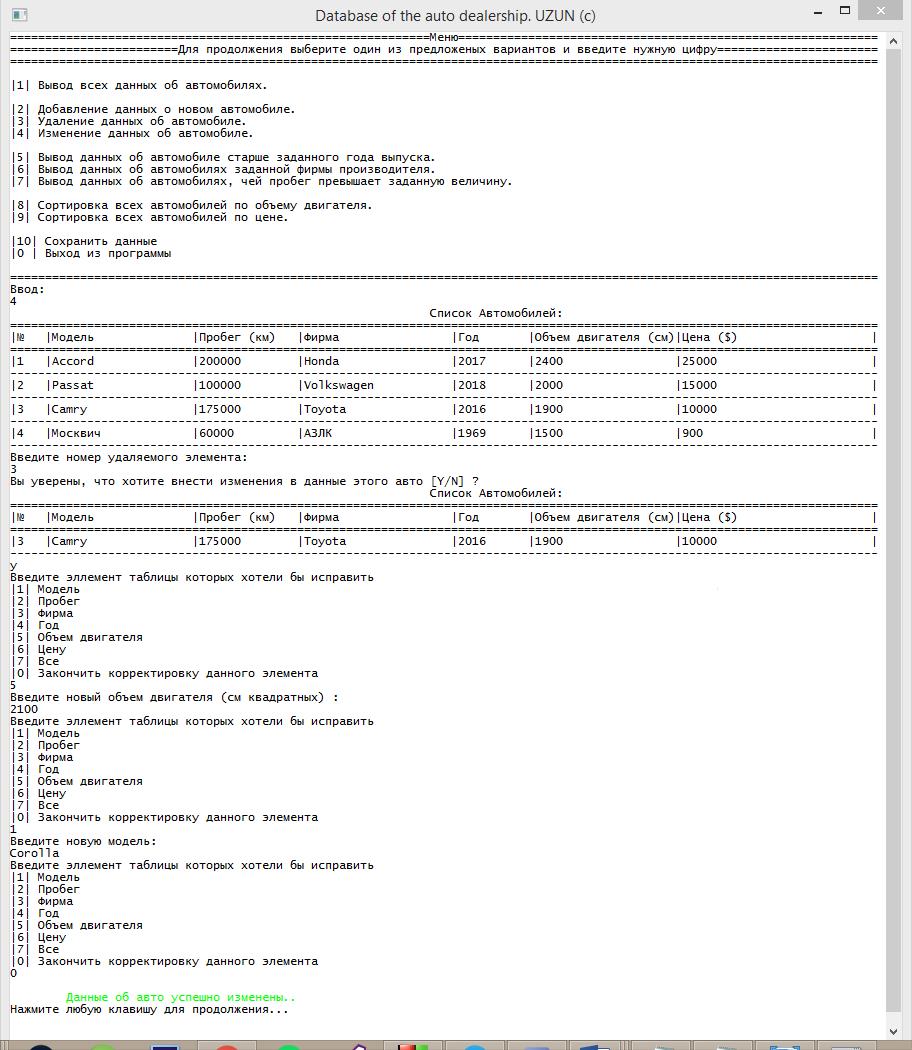


Рисунок 3.14 – Успішне корегування данних про авто.

При виборі пункту 0 вам запропонують вийти з програми без збереження, або зберегти результат в файл.

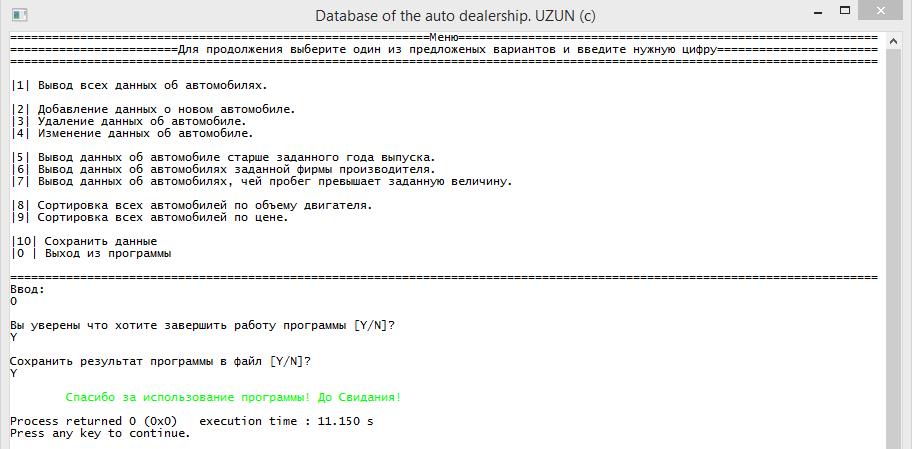


Рисунок 3.15 – Успішне завершення роботи.

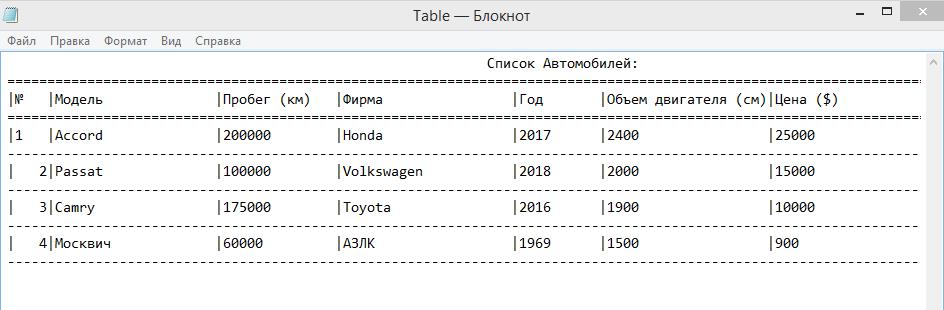
Після збереження файли з данними будуть розташовані у кореневому каталозі програми, а файл з відсортуйованими елементами у каталозі saves. 

Рисунок 3.16 – Файл із збереженою таблицею з даними.

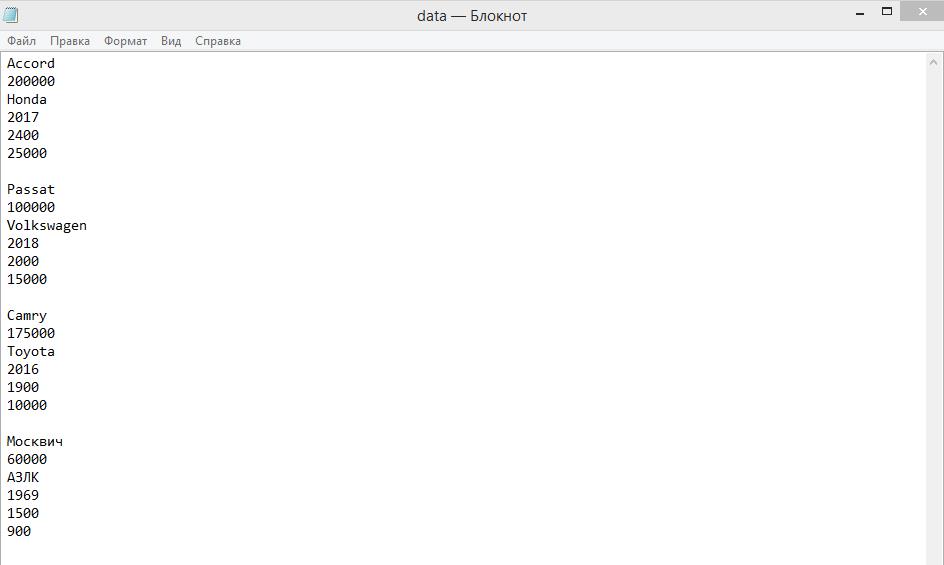


Рисунок 3.17 – Файл із збереженими даними для подальшого використання.

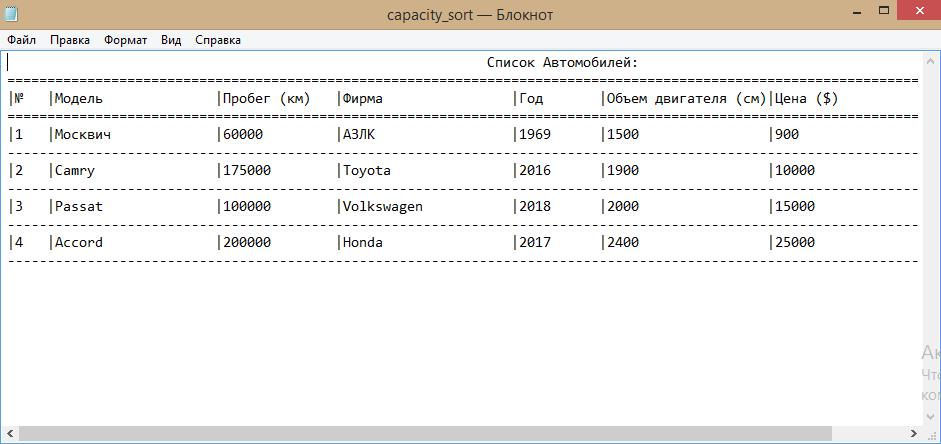


Рисунок 3.18 – Файл із відсортованими за об’ємом двигуна елементами.

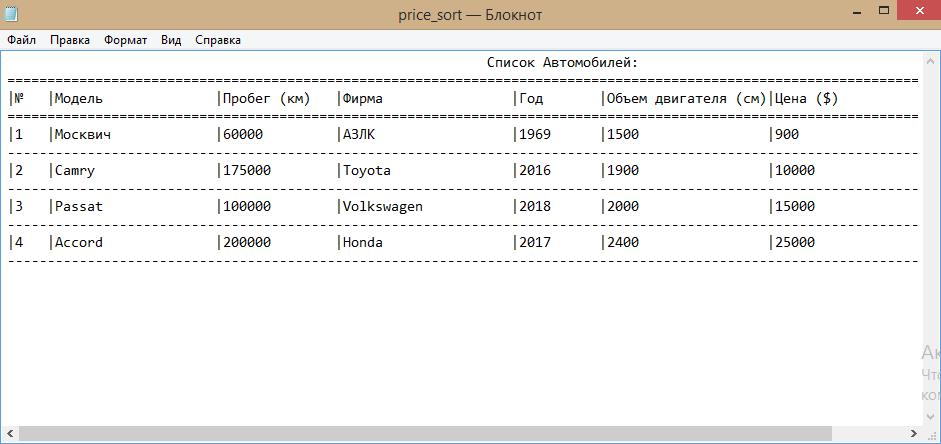


Рисунок 3.19 – Файл із відсортованими за ціною елементами.

# ВИСНОВКИ

В ході виконання даної курсової роботи були набуті навички роботи з абстрактними тапами даних. Зокрема навички програмної реалізації динамічної структури бінарне дерево і реалізації функцій на її основі таких як: вивід даних і сортування даних, видалення, додавання і корегування елемента дерева, а також пошук потрібного значення за заданим значенням поля.

Була створена програма для роботи з дінаміечкой структурної даних у вигляді реалізації бази даних автосалону, де в якості елементів виступають: марка втомобиля, пробіг, фірма, ціна, рік випуску і об'єм двигуна.

Додавання елемента виконується шляхом вношенія елемента в дерево впорядковано за ціною, реалізуючи тим самим бінарне дерево пошуку.

Видалення було поділене на дві функції: видалення будь-якого елементу і видалення кореня. Воно відбувалося за алгоритмом пошуку вдалою заміни і перепрівязки всіх задіяних в цьому елементів.

Коригування даних відбувається з будь-якого обраного полю.

Реалізовані функції сортування по ціні і обсягом двигуна (за принципом занесення даних в масив і його безпосередній сортування) і виведення на екран елементів, відповідних зазначеним умовам.

У багатьох функціях був використаний алгоритм рекурсивного виклику, що часто допомагало обходить дерево для пошуку потрібного елемента або порожнього вузла.

У програмі було Созданно 29 функцій з яких 12 основних і 17 допоміжних, косметичних, функцій перевірки введення і функцій для спрощення і скорочення коду.

Програмний код оснащений коментарями, і готовий для подальщій модифікації.

# ПЕРЕЛІК ПОСИЛАНЬ ТА СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] – <http://damp.biz/mova-programuvannya-c-campuswiki/>

[2] – [http:/programer.in.ua/index.php/prohramuvannia/prohramuvannia-na-movi-c/4-bazovyi-kurs-prohramuvannia-na-s-urok-1-znaiomstvo%20](http://programer.in.ua/index.php/prohramuvannia/prohramuvannia-na-movi-c/4-bazovyi-kurs-prohramuvannia-na-s-urok-1-znaiomstvo%20)

[3] – <https://www.intuit.ru/studies/courses/648/504/lecture/11455>

[4] – <http://kvodo.ru/queue.html>

[5] – <https://www.intuit.ru/studies/courses/648/504/lecture/11457%3Fpage%3D3>

[6] – <https://www.intuit.ru/studies/courses/648/504/lecture/11456>

[7] – <https://www.intuit.ru/studies/courses/648/504/lecture/11458>

# КОД ПРОГРАМИ

//ДОБРО ПОЖАЛОВАТЬ В КОД КУРСОВОЙ РАБОТЫ №1. ВЫПОЛНИЛ: Илья Узун. АИ-174. ВАРИАНТ №21.

#include <iostream>

#include <fstream>

#include <string.h>

#include <conio.h>

#include <iomanip>

#include <cstdlib>

#include <windows.h>

#include <stdlib.h>

#include <string>

using namespace std;

using std::ofstream;

const int col = 179; // ширина окна консоли

const int row = 84; // высота консоли

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE); // дескриптор устройства стандартного вывода

const int N=256; // константный размер строки

///основные переменные

int x = 0, x2 = 0, i = 0, j=0; // переменные счетчики // x - переход по кейсам

int number = 0; // счетчик общего количества

int number3 = 0; // счетчик для treeprint

int COUNTER = 0; // для функций

///счетчики

int changing\_number;

int del\_number;

int year\_search;

char company\_search[30];

int mileage\_search;

///переменные для задания входных значений для работы с функциями определенного вывода

int flagg = 0; //флаг

int kolvo = 0; //для функций вывода определенных значений

char answer = '\0'; //для yes\_no();

///вспомагательные переменные для функций

int menu (); //меню

struct CBinaryNode \* addnode (struct CBinaryNode \*ptr, struct CBinaryNode \*tree); //добавление элемента в дерево

int treeprint (struct CBinaryNode \*tree); //вывод всего дерева

struct CBinaryNode \*del\_elem (struct CBinaryNode \*tree, struct CBinaryNode \*del\_elem ); //удаление элемента из дерева

struct CBinaryNode \*root\_del (struct CBinaryNode \*del\_elem); //удаление корня из дерева

struct CBinaryNode \*correct (struct CBinaryNode \*tree); //изменение данных об элементе

void sort\_price ( struct database \*tree ); //ф-я сортировки массива по цене

void sort\_capacity ( struct database \*tree ); //ф-я сортировки массива по пробегу

int year\_output ( struct CBinaryNode \*tree); //вывод авто одного года выпуска

int company\_output( struct CBinaryNode \*tree); //вывод авто одной фирмы производителя

int mileage\_output ( struct CBinaryNode \*tree); //вывод авто одной фирмы производителя

void save(struct CBinaryNode \*tree); //сохранение в файл

void deltree\_mem (struct CBinaryNode \*tree); //освобождение памяти дерева

///основные функции

int one\_el\_print( struct CBinaryNode \*tree, int num); //печать одного эллемента

struct CBinaryNode \*input ( struct CBinaryNode \*tree ); //ввод

struct database \*sort\_mas ( struct CBinaryNode \*tree ); //для сортировки

struct CBinaryNode \*TREE\_scrl ( struct CBinaryNode \*tree, int del\_number); //пробег по дереву

///вспомогательные функции

int yes\_no();

void errors (int err\_x);

void finish (int succ\_x);

void header();

void privet ();

void SetColor (int text, int Fon);

void TITUL ();

///косметические функции

struct CBinaryNode \*empty\_check ( struct CBinaryNode \*tree );

int num\_check (int razmer);

float float\_num\_check (int razmer2);

char \*str\_check (int razmer3, int if\_integer);

///функции проверки

struct database //база данных

{

char car\_model[20]; //модель

float mileage; //пробег в км

char company[20]; //компания производитель

int year; //год

int engine\_capacity; //объем двигателя

float price; //цена

}no\_sort[N];

struct CBinaryNode //структура бинарного дерева поиска

{

database CAR; //база данных

struct CBinaryNode \*left; //левый потомок

struct CBinaryNode \*right; //правый потомок

struct CBinaryNode \*parent; //родитель

}NODES;

struct CBinaryNode \*root = NULL; // Объявляем структуру дерева

struct CBinaryNode \*ptr = &NODES; //структура узла

struct CBinaryNode \*current = NULL; //родительская переменная, указывающая на предыдущий узел

struct CBinaryNode \*PTR; //вспомогательная для прохода по дереву

FILE \*D\_INP;

FILE \*T\_INP; //файлы вывода

int main ( )

{

system ("color F0"); //меняем цвет

//setlocale(LC\_ALL,"Russian"); //меняем язык

SetConsoleTitle("Database of the auto dealership. UZUN (c)");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("mode con cols=125 lines=85"); //подстраиваем консоль под свои нужды

TITUL();

privet(); //вывод текста из файлов

if (yes\_no()==1) //читаем из файла

{

FILE \*D\_INP=fopen("data.txt", "r");

while (( fscanf ( D\_INP, "%s %f %s %d %d %f\n", &ptr -> CAR.car\_model, &ptr -> CAR.mileage, &ptr -> CAR.company, &ptr -> CAR.year, &ptr -> CAR.engine\_capacity, &ptr -> CAR.price)) != EOF)

root = addnode ( ptr, root);

fclose(D\_INP);

if ( root == NULL ) //проверка файла на пустоту

{

errors(5);

if (yes\_no()==1)

goto MENU;

else

exit (0);

}

}

else //очистка файла

{

ofstream D\_INP("data.txt", ios\_base::trunc);

ofstream T\_INP("Table.txt", ios\_base::trunc);

D\_INP.close();

T\_INP.close();

}

MENU:

finish(1);

do //меню

{

menu();

switch ( x )

{

case 1:///Вывод дерева

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

header();

number3=0;

treeprint(root);

finish(1);

break;

}

case 2:///Добавление элемента

{

root = addnode ( input(ptr), root); // размещаем введенный узел на дереве

finish(2);

break;

}

case 3:///Удаление елемента

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

header();

number3=0;

treeprint(root);

LL2:

del\_number=0;

cout<<"Введите номер удаляемого элемента:"<<endl;

del\_number = num\_check(N);

if ( del\_number > number || del\_number <= 0)

{

errors(2);

cout<<"Хотите выбрать другой элемент [Y/N] ?"<<endl;

if (yes\_no() == 1)

goto LL1;

else

break;

}

COUNTER = 1;

TREE\_scrl(root, del\_number); //ишем адрес удаляемого элемента

cout<<"Вы уверены, что хотите удалить данные об этом авто [Y/N] ?"<<endl;

header();

one\_el\_print(PTR, del\_number);

if (yes\_no() == 1 )

{

if ( del\_number == 1 )

root = root\_del(root); //удаляем корень

else

del\_elem (root, PTR); //удаляем елементы

}

else

{

cout<<"Хотите выбрать другой элемент [Y/N] ?"<<endl;

if (yes\_no() == 1)

goto LL2;

else

break;

}

finish(3);

break;

}

case 4:///Изменение данных

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

header();

number3=0;

treeprint(root);

LL1:

changing\_number=0;

cout<<"Введите номер удаляемого элемента:"<<endl;

changing\_number = num\_check(N);

if ( changing\_number > number || changing\_number <= 0)

{

errors(2);

cout<<"Хотите выбрать другой элемент [Y/N] ?"<<endl;

if (yes\_no() == 1)

goto LL1;

else

break;

}

COUNTER = 1;

TREE\_scrl(root, changing\_number); //адрес изменяемого элемента

cout<<"Вы уверены, что хотите внести изменения в данные этого авто [Y/N] ?"<<endl;

header();

one\_el\_print(PTR, changing\_number);

if (yes\_no() == 1)

correct ( PTR );

else

{

cout<<"Хотите выбрать другой элемент [Y/N] ?"<<endl;

if (yes\_no() == 1)

goto LL1;

else

break;

}

finish(4);

break;

}

case 5:///Вывод авто старше заданного года выпуска

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

cout<<"\nВведите год выпуска:\n";

year\_search=num\_check(10);

fflush(stdin);

number3 = 1;

flagg=0;

int z = year\_output (root);

if ( z == 0 )

cout<<"Таких автомобилей нет.\n";

finish(1);

break;

}

case 6:///Вывод данных об автомобилей заданной фирмы производителя.

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

cout<<"\nВведите фирму производителя:\n";

strcpy(company\_search, str\_check(23, 0));

fflush(stdin);

number3 = 1;

flagg=0;

int z2 = company\_output (root);

if ( z2 == 0 )

cout<<"Таких автомобилей нет.\n";

finish(1);

break;

}

case 7:///Вывод данных об автомобилей чей пробег превышает заданную величину.

{

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

cout<<"\nВведите пробег:\n";

mileage\_search = float\_num\_check(14);

fflush(stdin);

number3 = 1;

flagg=0;

int z3 = mileage\_output(root);

if ( z3 == 0 )

cout<<"Таких автомобилей нет.\n";

finish(1);

break;

}

case 8:///Вывод отсортированного по объему двигателя списка

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

COUNTER = 0;

sort\_mas(root);

number3 = 1;

flagg=0;

cout<<"\nПроизведем сортировку по объему двигателя...\n";

header();

sort\_capacity (no\_sort);

break;

case 9:///Вывод отсортированного по цене списка

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

COUNTER = 0;

sort\_mas( root );

number3 = 1;

flagg=0;

cout<<"\nПроизведем сортировку по цене...\n";

header();

sort\_price (no\_sort);

break;

case 10:///Сохранение

if ( empty\_check( root ) == NULL) //проверка на пустоту

break;

header();

number3=0;

treeprint(root); // выводим элементы дерева

cout<<"\nСохранить данные программы на этом этапе в файл [Y/N]?"<<endl;

if (yes\_no()==1)

{

ofstream D\_INP("data.txt", ios\_base::trunc); //чистим файл и заносим данные

ofstream T\_INP("Table.txt", ios\_base::trunc);

flagg=0;

D\_INP.close();

T\_INP.close();

number3=0;

flagg=0;

save(root);

finish(5);

break;

}

else

{

finish(5);

break;

}

case 0:

{

cout<<"\nВы уверены что хотите завершить работу программы [Y/N]?"<<endl;

if (yes\_no()==1)

{

cout<<"\nСохранить результат программы в файл [Y/N]?"<<endl;

if (yes\_no()==1)

{

ofstream D\_INP("data.txt", ios\_base::trunc); //чистим файл и заносим данные

ofstream T\_INP("Table.txt", ios\_base::trunc);

D\_INP.close();

T\_INP.close();

number3=0;

flagg=0;

save(root);

finish(0);

}

else

{

finish(0);

}

}

else

{

finish(1);

goto MENU;

}

}

default:

errors (3);

}

} while ( x!=0 );

deltree\_mem(root);

return 0;

}

///Печать одного эллемента

int one\_el\_print( struct CBinaryNode \*tree, int num)

{

cout <<"|"<<setw( 4 ) << num

<<"|"<<setw( 20 ) << left << tree -> CAR.car\_model

<<"|"<<setw( 14 ) << tree -> CAR.mileage

<<"|"<<setw( 21 ) << tree -> CAR.company

<<"|"<<setw( 10 ) << tree -> CAR.year

<<"|"<<setw( 20 ) << tree -> CAR.engine\_capacity

<<"|"<<setw( 27 ) << tree -> CAR.price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------"<<endl;

}

/// Вывод узлов дерева (обход в инфиксной форме)

int treeprint( struct CBinaryNode \*tree )

{

if ( tree != NULL ) //Пока не встретится пустой узел

{

number3++;

one\_el\_print( tree, number3 );

treeprint ( tree -> left ); //Рекурсивная функция вывода левого поддерева

treeprint ( tree -> right ); //Рекурсивная функция вывода правого поддерева

}

return ( number3 );

}

/// Добавление узлов в дерево

struct CBinaryNode \*addnode(struct CBinaryNode \*ptr, struct CBinaryNode \*tree) //ГОТОВО

{

if (tree == NULL) // Если дерева нет, то формируем корень

{

tree = new CBinaryNode;

strcpy(tree -> CAR.car\_model, ptr -> CAR.car\_model);//заносим данные

tree -> CAR.mileage = ptr -> CAR.mileage;

strcpy(tree -> CAR.company, ptr -> CAR.company);

tree -> CAR.year = ptr -> CAR.year;

tree -> CAR.engine\_capacity = ptr -> CAR.engine\_capacity;

tree -> CAR.price = ptr -> CAR.price;

number++;

tree->left = NULL;

tree->right = NULL; //ветви инициализируем пустотой

tree->parent = current; //передаем указатель на родителя

}

else

{

if ( ptr -> CAR.price < tree -> CAR.price ) //Если цена меньше корневого, уходим влево

{

current = tree;

tree -> left = addnode ( ptr, tree->left ); //Рекурсивно добавляем элемент

}

else if ( ptr -> CAR.price > tree -> CAR.price ) //иначе уходим вправо

{

current = tree;

tree -> right = addnode ( ptr, tree->right ); //Рекурсивно добавляем элемент

}

else //( ptr -> CAR.price = tree -> CAR.price ) усли равны - приходится уходить вправо

{

current = tree;

tree -> right = addnode ( ptr, tree->right ); //Рекурсивно добавляем элемент

}

};

return(tree);

}

///Удаление

struct CBinaryNode \*del\_elem (struct CBinaryNode \*tree, struct CBinaryNode \*del\_elem )

{

struct CBinaryNode \*p = del\_elem->parent; //ук-ль на родителя удаляемого элемента

if ( (del\_elem->left == NULL) && (del\_elem->right == NULL) ) // первый случай: удаляем лист

{

if ( p->left == del\_elem )

{

delete (del\_elem);

p->left = NULL;

}

else

{

delete(del\_elem);

p->right = NULL;

}

}

else if ( del\_elem->left == NULL || del\_elem->right == NULL ) // второй случай: удаляемый элемент имеет одного потомка

{

if ( del\_elem->left == NULL )

{

if ( p->left == del\_elem )

{

p->left = del\_elem->right;

del\_elem->right->parent = p;

delete (del\_elem);

}

else

{

p->right = del\_elem->right;

del\_elem->right->parent = p;

delete (del\_elem);

}

}

else

{

if ( p->left == del\_elem )

{

p->left = del\_elem->left;

del\_elem->left->parent = p;

delete (del\_elem);

}

else

{

p->right = del\_elem->left;

del\_elem->left->parent = p;

delete (del\_elem);

}

}

}

else //третий случай: удаление, где 2 сына

{

struct CBinaryNode \*first\_right = del\_elem->right; //укакзатель на правый элемент от удаляемого

struct CBinaryNode \*change = NULL; //для левого крайнего узла правого узла удаляемого элемента

if ( del\_elem->right->left == NULL )

{

if ( p->left == del\_elem )

{

del\_elem->left->parent = del\_elem->right;

p->left = del\_elem->right;

del\_elem->right->left = del\_elem->left;

del\_elem->right->parent = p;

delete (del\_elem);

}

else

{

p->right = del\_elem->right;

del\_elem->right->parent = p;

del\_elem->left->parent = del\_elem->right;

del\_elem->right->left = del\_elem->left;

delete (del\_elem);

}

}

else

{

while ( first\_right -> left != NULL )

{

first\_right = first\_right->left;

change = first\_right;

}

change -> parent ->left = NULL;

if ( p->left == del\_elem )

{

p->left = change;

change->parent->left = NULL;

change->parent = p;

change->left = del\_elem->left;

change->right = del\_elem->right;

change->left->parent = change;

change->right->parent = change;

delete (del\_elem);

}

else

{

p->right = change;

change->parent->left = NULL;

change->parent = p;

change->left = del\_elem->left;

change->right = del\_elem->right;

change->left->parent = change;

change->right->parent = change;

delete (del\_elem);

}

}

number--;

}

return (tree);

}

///Удаление корня

struct CBinaryNode \*root\_del (struct CBinaryNode \*del\_elem)

{

number--;

struct CBinaryNode \*zamena = NULL; //для левого крайнего узла правого узла удаляемого элемента

if ((del\_elem->right == NULL) && (del\_elem->left==NULL))

{

delete (del\_elem);

return ( zamena );

}

else if ( del\_elem->left == NULL || del\_elem->right == NULL )

{

if ( del\_elem->left == NULL )

{

if ( del\_elem->right->left == NULL )

{

zamena = del\_elem->right; //указатель на правый элемент от удаляемого

zamena->parent = NULL;

delete (del\_elem);

return ( zamena );

}

else

{

struct CBinaryNode \*zamena = del\_elem -> right;

while ( zamena -> left != NULL )

zamena = zamena -> left;

if ( zamena -> right != NULL )

{

zamena -> parent -> left = zamena -> right;

zamena -> right -> parent = zamena -> parent;

}

else

zamena -> parent -> left = NULL;

zamena -> parent -> left = NULL;

zamena->right = del\_elem->right;

zamena->parent = NULL;

zamena -> parent -> left = zamena -> right;

del\_elem->right->parent = zamena;

delete (del\_elem);

return ( zamena );

}

}

else

{

if ( del\_elem->left->right == NULL )

{

zamena = del\_elem->left;

zamena->parent = NULL;

delete (del\_elem);

return ( zamena );

}

else

{

struct CBinaryNode \*zamena = del\_elem -> left;

while ( zamena -> right != NULL )

zamena = zamena -> right;

if ( zamena -> left != NULL )

{

zamena -> parent -> right = zamena -> left;

zamena -> left -> parent = zamena -> parent;

}

else

zamena -> parent -> right = NULL;

zamena -> parent -> right = zamena -> left;

zamena->left = del\_elem->left;

zamena->parent= NULL;

del\_elem->left->parent = zamena;

delete (del\_elem);

return ( zamena );

}

}

}

else //случай если два 2 сына

{

if ( del\_elem->right->left == NULL )

{

zamena = del\_elem->right; //ук-ль на правый элемент от удаляемого

zamena->parent = NULL;

zamena->left = del\_elem->left;

del\_elem->left->parent = zamena;

delete (del\_elem);

return ( zamena );

}

else

{

struct CBinaryNode \*zamena = del\_elem -> right;

while ( zamena -> left != NULL )

zamena = zamena -> left;

if ( zamena -> right != NULL )

{

zamena -> parent -> left = zamena -> right;

zamena -> right -> parent = zamena -> parent;

}

else

zamena -> parent -> left = NULL;

zamena->right = del\_elem->right;

zamena->left = del\_elem->left;

zamena->parent = NULL;

del\_elem->left->parent = zamena;

del\_elem->right->parent = zamena;

delete (del\_elem);

return ( zamena );

}

}

}

///Корректировка данных

struct CBinaryNode \*correct (struct CBinaryNode \*tree)

{

do

{

cout<<"Введите эллемент таблицы которых хотели бы исправить\n"

<<"|1| Модель\n"

<<"|2| Пробег\n"

<<"|3| Фирма\n"

<<"|4| Год\n"

<<"|5| Объем двигателя\n"

<<"|6| Цену\n"

<<"|7| Все\n"

<<"|0| Закончить корректировку данного элемента"<<endl;

x2=num\_check(N);

switch (x2)

{

case 1:

{

cout<<"Введите новую модель: "<<endl;

strcpy(tree -> CAR.car\_model, str\_check(20, 1));

fflush(stdin); //чистка потока

break;

}

case 2:

{

cout<<"Введите новый пробег (в км):"<<endl;

tree -> CAR.mileage = float\_num\_check(14);

fflush(stdin);

break;

}

case 3:

{

cout<<"Введите новую фирму производителя:"<<endl;

strcpy(tree -> CAR.company, str\_check(23, 0));

fflush(stdin);

break;

}

case 4:

{

cout<<"Введите новый год выпуска:"<<endl;

tree -> CAR.year = num\_check(10);

fflush(stdin);

break;

}

case 5:

{

cout<<"Введите новый объем двигателя (см квадратных) :"<<endl;

tree -> CAR.engine\_capacity = num\_check(20);

fflush(stdin);

break;

}

case 6:

{

cout<<"Введите новую цену ($):"<<endl;

tree -> CAR.price = float\_num\_check(27);

fflush(stdin);

break;

}

case 7:

{

tree = input ( tree );

break;

}

case 0:

return (tree);

default:

errors (3);

}

}while (x2!=0);

}

///Сохранение

void save(struct CBinaryNode \*tree)

{

ofstream D\_INP("data.txt", ios\_base::app); //заносим данные в конец

ofstream T\_INP("Table.txt", ios\_base::app);

if (flagg==0) //первый раз - заносим заголовок

{

T\_INP<<setw( 80 ) << right << "Список Автомобилей:\n"

<<"============================================================================================================================\n"

<<"|"<<setw( 4 ) << left <<"№"

<<"|"<<setw( 20 ) << left << "Модель"

<<"|"<<setw( 14 ) << "Пробег (км)"

<<"|"<<setw( 21 ) << "Фирма"

<<"|"<<setw( 10 ) << "Год"

<<"|"<<setw( 20 ) << "Объем двигателя (см)"

<<"|"<<setw( 27 ) << "Цена ($)"

<<"|\n"<<"============================================================================================================================"<<endl;

}

if ( tree != NULL ) //Пока не встретится пустой узел

{

number3++;

T\_INP<<"|"<<setw( 4 ) << number3

<<"|"<<setw( 20 ) << left << tree -> CAR.car\_model

<<"|"<<setw( 14 ) << tree -> CAR.mileage

<<"|"<<setw( 21 ) << tree -> CAR.company

<<"|"<<setw( 10 ) << tree -> CAR.year

<<"|"<<setw( 20 ) << tree -> CAR.engine\_capacity

<<"|"<<setw( 27 ) << tree -> CAR.price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------\n";

D\_INP<< left << tree -> CAR.car\_model<<"\n"

<< tree -> CAR.mileage<<"\n"

<< tree -> CAR.company<<"\n"

<< tree -> CAR.year<<"\n"

<< tree -> CAR.engine\_capacity<<"\n"

<< tree -> CAR.price<<"\n\n";

flagg=1;

D\_INP.close();

T\_INP.close();

save ( tree -> left ); //Рекурсивная функция вывода левого поддерева

save ( tree -> right ); //Рекурсивная функция вывода правого поддерева

}

}

///Функция создания массива для сортировки sorted

struct database \*sort\_mas( struct CBinaryNode \*tree)

{

if ( tree != NULL ) // если дерево не пустое

{

no\_sort[COUNTER].mileage = tree->CAR.mileage;

no\_sort[COUNTER].year = tree->CAR.year;

no\_sort[COUNTER].engine\_capacity = tree->CAR.engine\_capacity;

no\_sort[COUNTER].price = tree->CAR.price;

strcpy (no\_sort[COUNTER].car\_model, tree->CAR.car\_model);

strcpy (no\_sort[COUNTER].company, tree->CAR.company);

COUNTER++;

sort\_mas (tree->left); // рекурсивно копируем левую ветку

sort\_mas (tree->right); // рекурсивно копируем правую ветку

}

return (no\_sort);

}

///Сортировка по цене sorting

void sort\_price ( struct database \*tree ) //сортировка по цене

{

for( i = 0 ; i < COUNTER ; i++ )

for( j = 0 ; j < COUNTER ; j++ )

if ( no\_sort[i].price < no\_sort[j].price)

{ //заносим остортированные данные в массив

char tmp1[20];

strcpy (tmp1, no\_sort[i].car\_model);

strcpy (no\_sort[i].car\_model, no\_sort[j].car\_model);

strcpy(no\_sort[j].car\_model, tmp1);

char tmp6[20];

strcpy (tmp6, no\_sort[i].company);

strcpy (no\_sort[i].company, no\_sort[j].company);

strcpy(no\_sort[j].company, tmp6);

float tmp2 = no\_sort[i].price;

no\_sort[i].price = no\_sort[j].price;

no\_sort[j].price = tmp2;

float tmp3 = no\_sort[i].mileage;

no\_sort[i].mileage = no\_sort[j].mileage;

no\_sort[j].mileage = tmp3;

int tmp4 = no\_sort[i].year;

no\_sort[i].year = no\_sort[j].year;

no\_sort[j].year = tmp4;

int tmp5 = no\_sort[i].engine\_capacity;

no\_sort[i].engine\_capacity = no\_sort[j].engine\_capacity;

no\_sort[j].engine\_capacity = tmp5;

}

number3=1;

for ( i = 0; i < COUNTER; i++, number3++ )

{

cout <<"|"<<setw( 4 ) << number3

<<"|"<<setw( 20 ) << left << no\_sort[i].car\_model

<<"|"<<setw( 14 ) << no\_sort[i].mileage

<<"|"<<setw( 21 ) << no\_sort[i].company

<<"|"<<setw( 10 ) << no\_sort[i].year

<<"|"<<setw( 20 ) << no\_sort[i].engine\_capacity

<<"|"<<setw( 27 ) << no\_sort[i].price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------"<<endl;

}

number3=1;

cout<<"Сохранить отсортированные данные в файл [Y/N]"<<endl;

if(yes\_no()==1)

{

ofstream price\_sort("saves/price\_sort.txt", ios\_base::trunc);

for ( i = 0; i < COUNTER; i++, number3++ )

{

if (i==0)

{

price\_sort<<setw( 80 ) << right << "Список Автомобилей:\n"

<<"============================================================================================================================\n"

<<"|"<<setw( 4 ) << left <<"№"

<<"|"<<setw( 20 ) << left << "Модель"

<<"|"<<setw( 14 ) << "Пробег (км)"

<<"|"<<setw( 21 ) << "Фирма"

<<"|"<<setw( 10 ) << "Год"

<<"|"<<setw( 20 ) << "Объем двигателя (см)"

<<"|"<<setw( 27 ) << "Цена ($)"

<<"|\n"<<"============================================================================================================================"<<endl;

}

price\_sort<<"|"<<setw( 4 ) << number3

<<"|"<<setw( 20 ) << left << no\_sort[i].car\_model

<<"|"<<setw( 14 ) << no\_sort[i].mileage

<<"|"<<setw( 21 ) << no\_sort[i].company

<<"|"<<setw( 10 ) << no\_sort[i].year

<<"|"<<setw( 20 ) << no\_sort[i].engine\_capacity

<<"|"<<setw( 27 ) << no\_sort[i].price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------"<<endl;

}

price\_sort.close();

finish(5);

}

else

finish(1);

}

///Сортировка по обьему двигателя

void sort\_capacity ( struct database \*tree )

{

for( i = 0 ; i < COUNTER ; i++ )

for( j = 0 ; j < COUNTER ; j++ )

if ( no\_sort[i].engine\_capacity < no\_sort[j].engine\_capacity)

{ //заносим остортированные данные в массив

char tmp1[20];

strcpy (tmp1, no\_sort[i].car\_model);

strcpy (no\_sort[i].car\_model, no\_sort[j].car\_model);

strcpy(no\_sort[j].car\_model, tmp1);

char tmp6[20];

strcpy (tmp6, no\_sort[i].company);

strcpy (no\_sort[i].company, no\_sort[j].company);

strcpy(no\_sort[j].company, tmp6);

float tmp2 = no\_sort[i].price;

no\_sort[i].price = no\_sort[j].price;

no\_sort[j].price = tmp2;

float tmp3 = no\_sort[i].mileage;

no\_sort[i].mileage = no\_sort[j].mileage;

no\_sort[j].mileage = tmp3;

int tmp4 = no\_sort[i].year;

no\_sort[i].year = no\_sort[j].year;

no\_sort[j].year = tmp4;

int tmp5 = no\_sort[i].engine\_capacity;

no\_sort[i].engine\_capacity = no\_sort[j].engine\_capacity;

no\_sort[j].engine\_capacity = tmp5;

}

number3=1;

for ( i = 0; i < COUNTER; i++, number3++ )

{

cout <<"|"<<setw( 4 ) << number3

<<"|"<<setw( 20 ) << left << no\_sort[i].car\_model

<<"|"<<setw( 14 ) << no\_sort[i].mileage

<<"|"<<setw( 21 ) << no\_sort[i].company

<<"|"<<setw( 10 ) << no\_sort[i].year

<<"|"<<setw( 20 ) << no\_sort[i].engine\_capacity

<<"|"<<setw( 27 ) << no\_sort[i].price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------"<<endl;

}

number3=1;

cout<<"Сохранить отсортированные данные в файл [Y/N]"<<endl;

if(yes\_no()==1)

{

ofstream capacity\_sort("saves/capacity\_sort.txt", ios\_base::trunc);

for ( i = 0; i < COUNTER; i++, number3++ )

{

if (i==0)

{

capacity\_sort<<setw( 80 ) << right << "Список Автомобилей:\n"

<<"============================================================================================================================\n"

<<"|"<<setw( 4 ) << left <<"№"

<<"|"<<setw( 20 ) << left << "Модель"

<<"|"<<setw( 14 ) << "Пробег (км)"

<<"|"<<setw( 21 ) << "Фирма"

<<"|"<<setw( 10 ) << "Год"

<<"|"<<setw( 20 ) << "Объем двигателя (см)"

<<"|"<<setw( 27 ) << "Цена ($)"

<<"|\n"<<"============================================================================================================================"<<endl;

}

capacity\_sort<<"|"<<setw( 4 ) << number3

<<"|"<<setw( 20 ) << left << no\_sort[i].car\_model

<<"|"<<setw( 14 ) << no\_sort[i].mileage

<<"|"<<setw( 21 ) << no\_sort[i].company

<<"|"<<setw( 10 ) << no\_sort[i].year

<<"|"<<setw( 20 ) << no\_sort[i].engine\_capacity

<<"|"<<setw( 27 ) << no\_sort[i].price

<<"|\n"<<"----------------------------------------------------------------------------------------------------------------------------"<<endl;

}

capacity\_sort.close();

finish(5);

}

else

finish(1);

}

///Вывод данных об автомобиле старше заданного года выпуска.

int year\_output( struct CBinaryNode \*tree)

{

if ( tree != NULL ) // пока дерево не пустое

{

if ( year\_search < tree -> CAR.year )

{

if ( flagg == 0 )

{

header();

flagg=1;

}

one\_el\_print( tree, number3 );

number3++;

kolvo++;

}

year\_output ( tree -> left ); //Рекурсивная функция вывода левого поддерева

year\_output ( tree -> right ); //Рекурсивная функция вывода правого поддерева

}

return ( kolvo );

}

///Вывод данных об автомобилей заданной фирмы производителя.

int company\_output( struct CBinaryNode \*tree)

{

if ( tree != NULL ) // пока дерево не пустое

{

if ( !stricmp ( company\_search, tree -> CAR.company ) )

{

if ( flagg == 0 )

{

header();

flagg=1;

}

one\_el\_print( tree, number3 );

kolvo++;

number3++;

}

company\_output ( tree -> left ); //Рекурсивная функция вывода левого поддерева

company\_output ( tree -> right ); //Рекурсивная функция вывода правого поддерева

}

return ( kolvo );

}

///Вывод данных об автомобилей чей пробег превышает заданную величину.

int mileage\_output( struct CBinaryNode \*tree)

{

if ( tree != NULL ) // пока дерево не пустое

{

if ( mileage\_search < tree -> CAR.mileage )

{

if ( flagg == 0 )

{

header();

flagg=1;

}

one\_el\_print( tree, number3 );

kolvo++;

number3++;

flagg++;

}

mileage\_output ( tree -> left ); //Рекурсивная функция вывода левого поддерева

mileage\_output ( tree -> right ); //Рекурсивная функция вывода правого поддерева

}

return ( kolvo );

}

///Да/Нет

int yes\_no ()

{

answer = '\0';

cin>>answer;

fflush(stdin);

if ( answer == 'Y' || answer == 'y' )

return 1;

else if ( answer == 'N' || answer == 'n' )

return 2;

else

{

errors (6);

yes\_no();

}

}

///Функция для цветов

void SetColor (int Text, int Fon)

{

SetConsoleTextAttribute(h, (Fon << 4) + Text);

}

///Начало работы

void privet()

{

char S[N]={""}; //В S будут считываться строки

ifstream f("input/privet.txt"); //Открыли файл для чтения

while (!f.eof()) //Будем читать информацию пока не дойдем до конца файла

{

f.getline(S,N); //Построчное считывание информации в S

cout<<S<<endl; //Вывод очередной строки на экран

}

f.close(); //Закрыли открытый файл

}

///Титульный лист

void TITUL()

{

char S[N]={""}; //В S будут считываться строки

ifstream f("input/titul.txt"); //Открыли файл для чтения

while (!f.eof()) //Будем читать информацию пока не дойдем до конца файла

{

f.getline(S,N); //Построчное считывание информации в S

cout<<S<<endl; //Вывод очередной строки на экран

}

f.close(); //Закрыли открытый файл

getch();

system("cls"); //clearscreen

}

///Меню

int menu ()

{

char S[N]={""}; //В S будут считываться строки

ifstream f("input/menu.txt"); //Открыли файл для чтения

while (!f.eof()) //Будем читать информацию пока не дойдем до конца файла

{

f.getline(S,N); //Построчное считывание информации в S

cout<<S<<endl; //Вывод очередной строки на экран

}

f.close(); //Закрыли открытый файл

cout<<"Ввод: "<<endl;

x=num\_check(N);

return (x);

}

///Вывод ошибок

void errors (int err\_x)

{

SetColor(12, 15);

switch (err\_x)

{

case 1:

cout<<"\n\tОшибка! Ввод числовых значений недопустим. Повторите попытку..."<<endl;

SetColor(0, 15);

break;

case 2:

cout<<"\n\tОшибка! Такого элемента не существует. Попробуйте еще раз..."<<endl;

SetColor(0, 15);

break;

case 3:

cout<<"\n\tОшибка! Вы ввели недопустимое значение. Выберите номер из предложенных.."<<endl;

SetColor(0, 15);

break;

case 4:

cout<<"\n\tОшибка! Ввод символов кроме, числовых недопустим. Повторите попытку..."<<endl;

SetColor(0, 15);

break;

case 5:

cout<<"\n\tОшибка! Файл с данными пуст... Продолжить работу [Y/N]?"<<endl;

SetColor(0, 15);

break;

case 6:

cout<<"\n\tВведите 'Y' - для подтверждения действия или 'N' - для отмены. "<<endl;

SetColor(0, 15);

break;

case 7:

cout<<"\n\tОшибка! Невозможно открыть файл."<<endl;

SetColor(0, 15);

break;

case 8:

cout<<"\n\tОшибка! Вы ввели слишком большое значение."<<endl;

SetColor(0, 15);

break;

case 9:

cout<<"Пока-что вами не был введен ни один элемент"<<endl;

SetColor(0, 15);

break;

}

}

///Заголовок

void header()

{

cout<<setw( 80 ) << right << "Список Автомобилей:\n"

<<"============================================================================================================================\n"

<<"|"<<setw( 4 ) << left <<"№"

<<"|"<<setw( 20 ) << left << "Модель"

<<"|"<<setw( 14 ) << "Пробег (км)"

<<"|"<<setw( 21 ) << "Фирма"

<<"|"<<setw( 10 ) << "Год"

<<"|"<<setw( 20 ) << "Объем двигателя (см)"

<<"|"<<setw( 27 ) << "Цена ($)"

<<"|\n"<<"============================================================================================================================"<<endl;

}

///Функция завершения каждого кейса

void finish (int succ\_x)

{

SetColor(10, 15);

switch (succ\_x)

{

case 1:

SetColor(0, 15);

break;

case 2:

cout<<"\n\tЭлемент успешно добавлен..."<<endl;

SetColor(0, 15);

break;

case 3:

cout<<"\n\tДанные об авто успешно удалены.."<<endl;

SetColor(0, 15);

break;

case 4:

cout<<"\n\tДанные об авто успешно изменены.."<<endl;

SetColor(0, 15);

break;

case 5:

cout<<"\n\tДанные успешно сохранены.."<<endl;

SetColor(0, 15);

break;

case 0:

cout<<"\n\tСпасибо за использование программы! До Свидания!"<<endl;

SetColor(0, 15);

exit(0);

}

cout<<"Нажмите любую клавишу для продолжения..."<<endl;

getch();

system ("cls");

}

///Для проверки наличия в дереве элементов

struct CBinaryNode \*empty\_check ( struct CBinaryNode \*tree )

{

if ( tree == NULL )

{

errors(9);

finish(1);

return NULL;

}

}

/\*МОЖНО СДЕЛАТЬ СОРТИРОВКУ ПО ЛЮБОМУ СТОЛБЦУ ТАБЛИЦЫ - ЗАМЕНИМ ДВЕ ФУНКЦИИ СОРТИРОВКУ НА ОДНУ - С ПЕРЕДАЧЕЙ ЗНАЧЕНИЯ НУЖНОГО СТОЛБЦЫ ТЕМ САМЫМ СОКРАТИВ КОД И ДОБАВИВ ВОЗМОЖНОСТЕЙ (ТОЖЕ САМОЕ КАСАЕТСЯ ФУНКЦИЙ УДАЛЕНИЯ, КОРРЕКТИРОВКИ И ВЫВОДА\*/ ///НА БУДУюЩЕЕ

///Проход по дереву

struct CBinaryNode \*TREE\_scrl ( struct CBinaryNode \*tree, int del\_number )

{

if ( tree != NULL ) //Пока не встретится пустой узел

{

if ( del\_number == COUNTER )

PTR = tree;

COUNTER++;

TREE\_scrl ( tree -> left, del\_number ); //Рекурсивная функция вывода левого поддерева

TREE\_scrl ( tree -> right, del\_number ); //Рекурсивная функция вывода правого поддерева

}

return ( PTR );

}

///Проверка - int

int num\_check (int razmer)

{

int bk; //переменная возврата

char znach[N];

int err=0;

do

{

memset(znach,'\0', sizeof (znach)); //заполнения всей строки нулями

err=0;

cin.getline(znach, 256);

fflush(stdin); //чистка потока ввода

if ((strlen(znach)) > razmer) //проверка длины

{

errors(8);

err = 1;

}

if (err==0) //проверка на цифры

{

for ( i = 0 ; i < strlen ( znach ) ; i++ )

if ( ( isdigit ( ( unsigned char ) znach [i] ) ) == 0 )

err=2;

if (err==2)

errors(4);

}

if (err==0)

bk = atoi ( znach );

} while (err != 0);

return bk;

}

///Проверка - float

float float\_num\_check (int razmer2)

{

cout.setf(ios\_base::fixed);

cout.precision(2);

int err = 0;

string znach;

int j = 0;

int ser=0;

do

{

//memset(znach,'\0', sizeof (znach));

ser = 0;

err = 0;

getline(cin, znach);

fflush(stdin); //чистка потока

if ((znach.length() > razmer2)) //проверка длины

{

errors(8);

err = 1;

}

if (err==0) //поиск запятой

{

for ( i = 0 ; i < znach.length() ; i++ )

if ( (znach [i] == ',') || (znach [i] == '.') )

{

znach [i] = ',';

ser = i;

}

if ( ser != 0 ) //проверка на цифры

{

for ( i = 0; i < ser; i++)

if ( isdigit ( znach [i] ) == 0 )

err=1;

for ( j = ser+1; j < znach.length() ; j++ )

if ( isdigit ( znach [j] ) == 0 )

err=1;

}

else

for ( i = 0 ; i < znach.length() ; i++ )

if ( ( isdigit ( ( unsigned char ) znach [i] ) ) == 0)

err=1;

if (err==1)

errors(4);

}

} while (err != 0);

float bk = atof (znach.c\_str());

return bk;

}

///Проверка - строка

char \*str\_check (int razmer3, int if\_integer)

{

char str[N];

char \*PTR = str;

int err=0;

if (if\_integer==0)

{

do

{

memset(str,'\0', sizeof (str)); //заполнения всей строки нулями

cin.getline(str, N);

fflush(stdin); //чистка потока

for ( i = 0 ; str[i] ; i++ )

if ( (str[i] >= '0') && (str[i] <= '9'))

break;

if ( str[i] )

errors (1);

}while (str[i]);

if ((strlen(str)) > razmer3)

{

errors(8);

str\_check(razmer3, if\_integer);

}

else

return PTR;

}

if (if\_integer == 1)

{

do

{

err=0;

memset(str,'\0', sizeof (str));

cin.getline(str, N);

fflush(stdin);

if (strlen(str)>razmer3)

{

err=1;

errors(8);

}

}while (err != 0);

return PTR;

}

}

///Ввод данных

struct CBinaryNode \*input ( struct CBinaryNode \*tree )

{

cout<<"Введите модель: "<<endl;

strcpy(tree -> CAR.car\_model, str\_check(20, 1));

fflush(stdin); //чистка потока

cout<<"Введите пробег (в км):"<<endl;

tree -> CAR.mileage = float\_num\_check(14);

fflush(stdin);

cout<<"Введите фирму производителя:"<<endl;

strcpy(tree -> CAR.company, str\_check(23, 0));

fflush(stdin);

cout<<"Введите год выпуска:"<<endl;

tree -> CAR.year = num\_check(10);

fflush(stdin);

cout<<"Введите объем двигателя (см квадратных) :"<<endl;

tree -> CAR.engine\_capacity = num\_check(20);

fflush(stdin);

cout<<"Введите цену ($):"<<endl;

tree -> CAR.price = float\_num\_check(27);

fflush(stdin);

return tree;

}

///Освобождение памяти дерева

void deltree\_mem( struct CBinaryNode \*tree)

{

if ( tree != NULL ) // если дерево не пустое

{

deltree\_mem(tree->left); // рекурсивно удаляем левую ветку

deltree\_mem(tree->right); // рекурсивно удаляем правую ветку

delete (tree); // удаляем корень

}

}