Вступление

СУБД (Система управления базой данных)— совокупность программных средств(функций), обеспечивающих управление созданием, удалением и редактированием базы данных.

База данных (БД) — некоторая совокупность объектов(узлов) и их значений, систематизированных таким образом (в данном случае иерархически), чтобы эти объекты могли быть найдены и обработаны пользователем БД.

Иерархическая модель данных — это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из узлов различных уровней и значений, которые хранятся в узлах.

Узел БД - это объект (запись в БД), содержащий некоторые значения, который хранится в БД и может редактироваться.

На схеме иерархического дерева узлы представляются вершинами графа разного уровня. Каждый узел на более низком уровне(потомок) связан только с одним узлом, находящимся на более высоком уровне(предок). Узел корня единственный и расположен на самом высоком (первом) уровне. Узлы, находящиеся на одном уровне и имеющие общего предка, называются братьями. Каждый предок связан только с одним из своих потомков, все его остальные потомки связаны друг с другом как братья.

Связанные узлы — это узлы, расположенные таким образом, что от узла на более высоком уровне можно перейти к узлу на более низком уровне, не нарушая иерархичности БД.

Каждый узел базы данных характеризуется следующим образом:

- <u>Имя узла</u>. Идентификатор, по которому пользователь БД может обращаться к узлу. Является строковой переменной с длиной, не более 255 символов. Узлы-братья не могут иметь одно имя
- <u>Путь к узлу</u> уникальный идентификатор, позволяющий выделить конкретный узел из всех узлов БД. Составляется как набор из имён узлов через которые проходит путь в дереве(графе) БД к данному узлу от корневого узла
- <u>Ключ узла</u> уникальный сгенерированный код необходимый для записи и восстановления БД из файла. Используется программно и недоступен пользователю
- Предок узла
- Любой из существующих потомков
- Соседние братья (справа и слева)
- Значения (представленные в виде списка), содержащиеся в данном узле

Структура узла(пример):

```
typedef struct NODE
{
    char NodeName[255]; //имя узла
    int key; // ключ узла
    struct NODE* UpNode; //указатель на предка
    struct NODE* PreviousNode; //указатель на предшествующего брата в списке
    struct NODE* NextNode; //указатель на следующего брата в списке
    struct NODE* DownNode; //указатель на потомка
    struct VALUE * Values; //указатель на значения
}NODE;
```

Значение - это содержащееся в конкретном узле информация (его характеристика). Может быть представлена в одном из следующих типов (INT, FLOAT, DOUBLE, STRING)

Каждое значение представляется в следующем виде:

- Спецификатор (ключ значения). Уникальный идентификатор, позволяющий выделить конкретное значение из всех значений данного узла. Является строковой переменной с длиной, не более 255 символов
- Тип значения
- Указатель на хранимую информацию
- Указатель на следующее значение этого узла если оно существует

Структура значения(пример):

Описание

Целью данного проекта являлась разработка и реализация API, который позволяет осуществлять работу с базой данных, фактически создание API для простейшей СУБД.

Реализация проекта велась на языке программирования С(Си).

АРІ включает в себя следующие возможности для организации работы с БД на программном уровне:

• Работа с узлами:

- о Создание нового узла в базе данных (в т.ч. создание новой базы данных)
- о Удаление узла из базы данных (в т.ч. удаление базы данных)
- о Поиск узла в базе данных
- о Поиск среди потомков узла
- о Переход в узел по пути
- о Сохранение базы данных в файл
- о Чтение базы данных из файла
- о Вывод всех потомков узла
- о Вывод всех ветвей дерева БД (от опр. вершины)
- о Вывод пути от вершины дерева до узла

• Работа со значениями и ключами(спецификаторами):

- о Добавление значения в узел
- о Удаление значения из узла
- о Удаление всех значений (значений определенного типа) из узла
- о Копирование значения
- о Вставка значения
- о Поиск значения в узле
- о Вывод всех значений (значений определенного типа) узла
- о Изменение значения и/или типа у данного спецификатора
- о Изменение спецификатора у значения

Подробное описание функций в этих приложениях.

Принцип сохранения БД в локальных файлах (см. здесь)

Реализация

Для работы с функционалом необходимо использовать среду разработки Microsoft Visual Studio.

Поддерживается версия Microsoft Visual Studio 2015.

- о Заголовочный файл DataBase15.h
- о Библиотека DataBase15.h

Совместимость с другими версиями не гарантируется. Для возможности работы поставляется файл-архив с файлами исходного кода функций заголовочными файлами для возможности собрать библиотеку средствами компилятора других версий.

Функция input tree

Прототип функции input_tree

NODE* input tree(NODE* currPtr, char* newName)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция input_tree создает в базе данных новый узел, являющийся потомком для текущего узла.

Параметры

• currPtr

Указатель на узел типа NODE, для которого требуется создать узел-потомок. Если передать NULL, то функция создаёт корень нового дерева

newName

Имя в виде си-строки, которое будет присвоено вновь созданному узлу. Не может превышать 255 символов по длине. Имена узлов-братьев не могут совпадать

Возвращаемое значение

В случае успеха возвращается указатель на созданный объект типа NODE.

В случае ошибки возвращается NULL.

Пример: исходный код программы

```
Возвращаемый результат:
```

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* newNode = NULL;
    root = input_tree(NULL, "root");
    directory(root);
    newNode = input_tree(root, "NewNode");
    directory(root);
    getch();
}
```

```
Directory is empty.
Directory contains:
NewNode
```

Функция delete node

Прототип функции delete_node

int delete_node(NODE** currentNode, NODE** root)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	С
DataBase15.lib	

Описание

Функция delete_node удаляет из базы данных узел со всеми его потомками. Удаление рекурсивно (т.е. удаляются также все потомки потомков и так далее)

Параметры

• currentFolder

Адрес указателя на узел типа NODE, к которому требуется применить операцию удаления

• root

#include "DataBase15.h"

Адрес указателя на корневой узел типа NODE

Возвращаемое значение

1 в случае успешного удаления

0 в случае ошибки (попытка удаления несуществующего узла)

Пример: исходный код программы:

```
int main()
{
    NODE* root = NULL;
    NODE* temp = NULL;
    int i;
    root = input_tree(NULL, "root");
    input_tree(root, "Child1");
    temp = input_tree(root, "Child2");
    input_tree(root, "Child3");
    directory(root);
    input_tree(temp, "Child1");
    i = delete(&temp, &root);
    directory(root);
    getch();
}
```

```
Directory is empty.
Directory contains:
NewNode
Directory contains:
Child3
Child2
Child1
Child1
Child2 deleted
Directory contains:
Child3
Child3
Child3
Child3
```

Функция find_node

Прототип функции find_node

NODE* find_node(char* fileName, NODE* beginf)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	С

Описание

Функция find_node осуществляет поиск узла с именем fileName, среди всех узловпотомков (в т.ч. их потомков и так далее, т.е. рекурсивно).

Параметры

• beginf

Указатель на узел типа NODE, на уровнях ниже которого среди связанных с ним узлов необходимо обнаружить искомый.

• fileName

Си-строка с искомым именем объекта

Возвращаемое значение

В случае успеха возвращается указатель на искомый объект типа NODE.

В случае ошибки возвращается NULL.

Пример: исходный код программы

```
#include "DataBase15.h"
int main()
{
    NODE* root = NULL;
    NODE* NewNode = NULL;
    root = input_tree(NULL, "root");
    directory(root);
    NewNode = input_tree(root, "NewNode");
    directory(root);
    NewNode = input_tree(NewNode, "NewNode2");
    NewNode = find_node("NewNode", root);
    directory(NewNode);
    getch();
}
```

```
Directory is empty.
Directory contains:
NewNode
Directory contains:
NewNode2
```

Функция go to node

Прототип функции go_to_node

NODE* go_to_node(char* s1, NODE* node)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция go_to_node осуществляет поиск узла с именем s1, среди всех узлов-потомков текущего узла.

Параметры

• node

Указатель на узел типа NODE, среди потомков которого необходимо обнаружить искомый узел.

• s1

Си-строка с искомым именем объекта

Возвращаемое значение

В случае успеха возвращается указатель на искомый объект типа NODE.

В случае ошибки возвращается NULL.

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* temp = NULL;
    root = input_tree(NULL, "root");
    temp = input_tree(root, "NewName1");
    input_tree(root, "NewName2");
    temp = input_tree(temp, "NewName11");
    directory(root);
    directory(temp);
    temp = go_to_node("NewName1", root);
    directory(temp);
    getch();
}
```

```
Directory contains:
NewName2
NewName1
Directory is empty.
Directory contains:
NewName11
```

Функция go to path

Прототип функции go_to_path

NODE* go_to_path(NODE * root, char* path)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция go_to_path возвращает указатель на узел базы данных расположенный по указанному пути.

Параметры

• root

Указатель на узел типа NODE, адрес которого нужно получить

path

Си-строка, содержащая путь к узлу Root. Узлы в пути нужно разделять "." (прим. MilkyWay.SolarSystem.Earth). Необходимо передавать указатель на начало строки. Передача самой строки непосредственно (прим. "MilkyWay.SolarSystem.Earth" вместо char s[]="MilkyWay.SolarSystem.Earth") недопустимо.

Возвращаемое значение

getch();

}

В случае успеха возвращается указатель на узел базы данных расположенный по указанному пути.

В случае ошибки возвращается NULL

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* temp = NULL;
    char s[] = "MilkyWay.SolarSystem";
    root = input_tree(NULL, "MilkyWay");
    temp=input_tree(root, "SolarSystem");
    add_value(temp, "Wide", INT, "100");
    temp = input_tree(temp, "Earth");
    temp=go_to_path(root, s);
    print_values(temp, ALL);
```

Возвращаемый результат:

Wide: 100N (int)

Функция record_tree

Прототип функции record_tree

int record_tree(NODE* CurrentNode, FILE* FileWithNodes, FILE*
FileWithValues)

Название	Язык
pi15db.h	С

Описание

Функция record_tree создает локальную копию базы данных в 2-х файлах формата .dat по образцу виртуальной копии

Параметры

CurrentNode

Указатель на объект типа NODE, который является дерева сохраняемой базы данных

• FileWithNodes

Указатель на объект типа FILE формата .dat, в котором будет создана локальная копия базы данных (данные об узлах). Файл необходимо открывать в режиме записи "rw"

• FileWithNodes

Указатель на объект типа FILE формата .dat, в котором будет создана локальная копия базы данных (данные о значениях). Файл необходимо открывать в режиме записи "rw"

Возвращаемое значение

Возвращает 0 если указатель на вершину дерева пустой; 1 если произошла ошибка открытия файла

2 в случае успеха

#include"DataBase15.h"

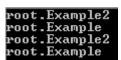
```
int main()
{
    FILE* node = fopen("1.dat", "wb");
    FILE* value = fopen("2.dat", "wb");
    NODE* root = NULL;
    root = input_tree(NULL, "root");
    input_tree(root, "Example");
    input_tree(root, "Example2");
    path_list(root);
    record_tree(root, node, value);
    fclose(node);
    fclose(value);
    node = fopen("1.dat", "rb");
    value = fopen("2.dat", "rb");
}
```

root = scan_file(node, value);

path_list(root);
fclose(node);
fclose(value);

}

Пример: исходный код программы



Функция scan_file

Прототип функции scan_file

NODE* scan file(FILE* FileWithNodes, FILE* FileWithValue)

Название	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция scan_file восстанавливает виртуальную копию базы данных по локальной копии, сохраненной в 2-х файлах формата .dat

Параметры

• FileWithNodes

Указатель на объект типа FILE формата .dat, в котором расположена локальная копия базы данных (данные об узлах). Файл необходимо открывать в режиме чтения "rb"

• FileWithNodes

Указатель на объект типа FILE формата .dat, в котором расположена локальная копия базы данных (данные о значениях). Файл необходимо открывать в режиме чтения "rb"

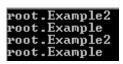
Возвращаемое значение

В случае успеха возвращается указатель на основание базы данных, т.н. корень (объект типа NODE).

В случае ошибки или если файл пустой возвращается NULL.

Пример: исходный код программы #include"DataBase15.h"

```
int main()
        FILE* node = fopen("1.dat", "wb");
        FILE* value = fopen("2.dat", "wb");
        NODE* root = NULL;
        root = input_tree(NULL, "root");
        input_tree(root, "Example");
input_tree(root, "Example2");
        path_list(root);
        record_tree(root, node, value);
        fclose(node);
        fclose(value);
        node = fopen("1.dat", "rb");
value = fopen("2.dat", "rb");
root = scan_file(node, value);
        path_list(root);
        fclose(node);
        fclose(value);
        getch();
}
```



Функция directory

Прототип функции directory

void directory(NODE *current)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	С
DataBase15.lib	

Описание

Функция directory выводит на экран всех потомков узла.

Параметры

• current

Указатель на узел типа NODE, для которого требуется распечатать потомков

Возвращаемое значение

Функция типа void. Нет возвращаемых значений

Пример: исходный код программы

```
Возвращаемый результат:
```

```
#include "DataBase15.h"
int main()
{
    NODE* root = NULL;
    NODE* NewNode = NULL;
    root = input_tree(NULL, "root");
    directory(root);
    NewNode = input_tree(root, "NewNode");
    directory(root);
    getch();
}
```

```
Directory is empty.
Directory contains:
NewNode
```

Функция path_list

Прототип функции path_list

int path_list(NODE * currentNode)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	С
DataBase15.lib	

Описание

Функция path_list выводит на экран представление БД в виде дерева.

Параметры

currentNode

Указатель на узел типа NODE на вершину (корневой узел) дерева.

Возвращаемое значение

0 если дерево пустое

1 в случае успешного вывода на экран

Пример: исходный код программы

```
пример. ислооный коо программы
```

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* temp = NULL;
    root = input_tree(NULL, "MilkyWay");
    temp=input_tree(root, "SolarSystem");
    input_tree(temp, "Earth");
    input_tree(root, "System503");
    path_list(root);
    getch();
}
```

Возвращаемый результат:

MilkyWay.System503 MilkyWay.SolarSystem.Earth

Функция way

Прототип функции way

```
void way(NODE * beginf)
```

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	С
DataBase15.lib	

Описание

Функция way распечатывает путь от корневого узла до текущего.

Параметры

• beginf

Указатель на узел типа NODE, для которого требуется распечатать путь.

Возвращаемое значение

Функция типа void, нет возвращаемого значения.

Пример: исходный код программы

Возвращаемый результат:

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* temp = NULL;
    int i = 1;
    root = input_tree(NULL, "root");
    temp = input_tree(root, "NewNode");
    temp = input_tree(temp, "NewNode2");
    way(temp);
    getch();
}
```

root.NewNode.NewNode2

Функция add value

Прототип функции add_value

VALUE* add_value(NODE* currentPtr, char* qualf, TYPE type, char* info)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция add_value добавляет новое значение для узла в базе данных.

Параметры

• currentPtr

Указатель на узел типа NODE, для которого требуется добавить значение.

• qualf

Спецификатор в виде си-строки, который будет присвоен вновь созданному значению. Не может превышать 255 символов по длине

• type

Тип значения (INT, STRING, FLOAT, DOUBLE)

info

Си-строка содержащая само значение

Возвращаемое значение

В случае успеха возвращается указатель на созданный объект типа VALUE.

В случае ошибки возвращается NULL.

Пример: исходный код программы

Возвращаемый результат:

Value2: 15PI (int) Value1: 2016 (int)

```
#include "DataBase15.h"
int main()
{
    NODE* root = NULL;
    root = input_tree(NULL, "root");
    add_value(root, "Value1", INT, "2016");
    add_value(root, "Value2", INT, "15PI");
    print_values(root,ALL);
    getch();
}
```

Функция delete val

Прототип функции delete_val

int delete_val(VALUE* toDelete, NODE* start)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	С

Описание

Функция delete_val удаляет все значения/значения определённого типа из узла в БД.

Параметры

- toDelete
 - Указатель на значение, которое требуется удалить.
- start

Указатель на узел типа NODE, для которого требуется удалить значение.

Возвращаемое значение

0 в случае ошибки

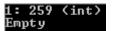
}

1 в случае успешного удаления

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    VALUE* temp = NULL;
    root = input_tree(NULL, "MilkyWay");
    temp = add_value(root, "1", INT, "259");
    print_values(root, ALL);
    delete_val(temp,root);
    print_values(root, ALL);
    getch();
```



Функция delete_all_values

Прототип функции delete_all_values

void delete_all_values(NODE *currPtr, TYPE type)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция delete_all_values удаляет все значения/значения определённого типа из узла в БД.

Параметры

- currPtr
 - Указатель на узел типа NODE, для которого требуется удалить значения.
- type

Тип значения (INT, STRING, FLOAT, DOUBLE, ALL если требуется удалить все)

Возвращаемое значение

Функция типа void. Нет возвращаемого значения

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    root = input_tree(NULL, "MilkyWay");
    add_value(root, "1", INT, "259");
    add_value(root, "2", CHAR, "Value");
    add_value(root, "3", INT, "346");
    print_values(root, ALL);
    delete_all_values(root, INT);
    print_values(root, ALL);
    getch();
}
```

```
3: 346 (int) 2: Value (char) 1: 259 (int)
2: Value (char)
```

Функция сору

Прототип функции сору

VALUE* copy(VALUE* valueFrom)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция сору копирует значение из узла в буфер. Буфер должен соответствовать типу VALUE.

Параметры

• valueFrom

Указатель на значение типа VALUE, которое требуется скопировать

Возвращаемое значение

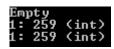
В случае успеха возвращается указатель типа VALUE на буфер со скопированным значением.

В случае неудачи возвращается NULL

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* rootN = NULL;
    VALUE* temp = NULL;
    root = input_tree(NULL, "MilkyWay");
    temp = add_value(root, "1", INT, "259");
    temp = copy(temp);
    rootN = input_tree(root, "Solar");
    print_values(rootN, ALL);
    paste(rootN, temp);
    print_values(root, ALL);
    print_values(rootN, ALL);
    getch();
}
```



Функция paste

Прототип функции paste

int paste(NODE* pasteTo, VALUE* copy)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция paste добавляет указанное значение в узел

Параметры

- pasteTo
 - Указатель на узел типа NODE, в который требуется поместить значение
- copy

Указатель на значение типа VALUE, которое требуется поместить в узел

Возвращаемое значение

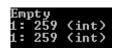
В случае успеха возвращается 2

При попытке вставить значение в узел, где уже существует значение с таким спецификатором возвращается 1

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    NODE* rootN = NULL;
    VALUE* temp = NULL;
    root = input_tree(NULL, "MilkyWay");
    temp = add_value(root, "1", INT, "259");
    temp = copy(temp);
    rootN = input_tree(root, "Solar");
    print_values(rootN, ALL);
    paste(rootN, temp);
    print_values(root, ALL);
    print_values(rootN, ALL);
    getch();
}
```



Функция find_value_in_node

Прототип функции find_value_in_node

VALUE* find_value_in_node(NODE* nodePtr, char* key)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция find_value_in_node находит значение с именем key в текущем узле.

Параметры

• nodePtr

Указатель на узел типа NODE, среди значений которого требуется обнаружить искомое

kev

Си-строка с именем искомого объекта

Возвращаемое значение

В случае успеха возвращается указатель типа VALUE

Если найти значение не удалось возвращается NULL

Пример: исходный код программы

```
#include "DataBase15.h"

int main()
{

    NODE* root = NULL;
    VALUE* temp = NULL;
    root = input_tree(NULL, "root");
    temp = add_value(root, "Value1", CHAR, "NewValue");
    temp = add_value(root, "Value2", CHAR, "NewValue");
    print_values(root, ALL);
    temp=find_value_in_node(root, "Value1");
    delete_val(temp, root);
    print_values(root, ALL);
    getch();
}
```

Функция print values

Прототип функции print_values

int print values(NODE *head, TYPE type)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция print_values выводит на экран все значения/значения определённого типа, которые содержит узел.

Параметры

- head
 - Указатель на узел типа NODE, значения которого необходимо распечатать.
- type
 Тип значения (INT, STRING, FLOAT, DOUBLE)

Возвращаемое значение

0 если в узле нет значений

Число значений в узле если они существуют

Пример: исходный код программы

Возвращаемый результат:

Value2: 15PI (int) Value1: 2016 (int)

```
#include "DataBase15.h"

int main()
{
    NODE* root = NULL;
    root = input_tree(NULL, "root");
    add_value(root, "Value1", INT, "2016");
    add_value(root, "Value2", INT, "15PI");
    print_values(root,ALL);
    getch();
}
```

Функция change_value

Прототип функции change_value

int change_value(VALUE * value, TYPE type, char * string)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция change_value меняет в данном узле выбранное значение и/или его тип.

Параметры

- value
 - Указатель на значение типа VALUE, которое требуется изменить
- type
 - Новый тип, который нужно присвоить (INT, STRING, DOUBLE, FLOAT) или ALL если тип менять не требуется
- string
 - Си-строка с новым значением или NULL если менять значение не требуется

Возвращаемое значение

0 в случае неверного указателя

2 случае успеха

3 при попытке выбора типа не из перечисленного списка (INT, CHAR, DOUBLE, FLOAT, ALL)

Пример: исходный код программы #include "DataBase15.h"



```
int main()
{
    NODE* root = NULL;
    VALUE* temp = NULL;
    root = input_tree(NULL, "root");
    temp = add_value(root, "Value2", INT, "2016");
    print_values(root, ALL);
    change_value(temp, CHAR, "HSE");
    print_values(root, ALL);
    getch();
}
```

Функция change_qualifier

Прототип функции change_qualifier

int change_qualifier(NODE * current, VALUE * value, char * string)

Заголовочный файл

Название и библиотека	Язык
DataBase15.h	C
DataBase15.lib	

Описание

Функция change_qualifier меняет в данном узле спецификатор выбранного значения.

Параметры

• current

Указатель на узел типа NODE, спецификатор значения которого требуется изменить.

• value

Указатель на значение типа VALUE, спецификатор которого требуется изменить

• string

Си-строка с новым спецификатором для значения

Возвращаемое значение

0 в случае неверного указателя

1 при попытке присвоить спецификатор, который принадлежит другому значению в этом узле

2 случае успеха

Пример: исходный код программы #include "DataBase15.h"

```
int main()
{
    NODE* root = NULL;
    VALUE* temp = NULL;
    root =input_tree(NULL, "root");
    temp = add_value(root, "Value2", INT, "2016");
    print_values(root, ALL);
    change_qualifier(root, temp, "HSE");
    print_values(root, ALL);
    getch();
}
```

Принцип работы БД с локальными копиями

С помощью функций record_tree и scan_file можно организовать сохранение и загрузку локальной копии базы данных.

Локальная копия сохраняется функцией record_tree в 2-х файлах с расширением .dat

В первый файл записывается информация обо всех узлах дерева БД со всеми их характеристиками, а также иерархическим порядком (за это отвечает ключ и смещение над подузлы). Информация об узлах хранится попарно: предок-потомок. Информация представляется в следующем виде:

‹‹

(Флаг Удаления 1 байт)(Ключ типа int)(Символьное имя типа char *255)(Смещение на данные во втором файле 8 байт)(Смещение на подузлы в первом файле 8 байт) // корень (Флаг Удаления 1 байт)(Ключ типа int)(Символьное имя типа char *255)(Смещение на данные во втором файле 8 байт)(Смещение на подузлы в первом файле 8 байт) //потомок

(Ключ int) //npedok

(Флаг Удаления 1 байт)(Ключ типа int)(Символьное имя типа char *255)(Смещение на данные во втором файле 8 байт)(Смещение на подузлы в первом файле 8 байт) //потомок

(Ключ int) //предок

(Флаг Удаления 1 байт)(Ключ типа int)(Символьное имя типа char *255)(Смещение на данные во втором файле 8 байт)(Смещение на подузлы в первом файле 8 байт) //потомок

<...>

>>

Во второй файл записывается информация обо всех значениях дерева БД со всеми их характеристиками. Чтобы получить к ним доступ используется информация о смещении из первого файла. Информация представляется в следующем виде:

‹‹

(Кол-во значений int == k)((Длина спецификатора int == n)(Спецификатор char* [n])(Длина значения int == p)(Значение char* [p])(Тип Int))*k(Смещение на новые зн. 8 байт) (Кол-во значений int == k)((Длина спецификатора int == n)(Спецификатор char* [n])(Длина значения int == p)(Значение char* [p])(Тип Int))*k(Смещение на новые зн. 8 байт)

<...>

>>