

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 C(Си).

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

- **Работа с узлами:**
 - [Создание нового узла в базе данных \(в т.ч. создание новой базы данных\)](#)
 - [Удаление узла из базы данных \(в т.ч. удаление базы данных\)](#)
 - [Поиск узла в базе данных](#)
 - [Поиск среди потомков узла](#)
 - [Переход в узел по пути](#)
 - Сохранение базы данных в файл
 - Чтение базы данных из файла
 - [Вывод всех потомков узла](#)
 - [Вывод всех ветвей дерева БД \(от опр. вершины\)](#)
 - [Вывод пути от вершины дерева до узла](#)
- **Работа со значениями и ключами(спецификаторами):**
 - [Добавление значения в узел](#)
 - [Удаление значения из узла](#)
 - [Удаление всех значений \(значений определенного типа\) из узла](#)
 - [Копирование значения](#)
 - [Вставка значения](#)
 - Поиск значения в узле
 - Вывод всех значений узла
 - Вывод значений определенного типа в узле
 - Изменение значения и/или типа у данного спецификатора
 - Изменение спецификатора у значения
 - Вывод спецификатора у значения

Подробное описание функций в приложениях.

Реализация

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

Поддерживается версия Microsoft Visual Studio 2015.

- Заголовочный файл DataBase15.h
- Библиотека DataBase15.h

Совместимость с другими версиями не гарантируется.

Функция `input_tree`

Прототип функции `input_tree`

`NODE* input_tree(NODE* currPtr, char* newName)`

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

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

```
#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();
}
```

Возвращаемый результат:

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

Функция delete

Прототип функции delete

```
int delete(NODE** currentNode, NODE** root)
```

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

Описание

Функция **delete** 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**
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:

```
#include "DataBase15.h"

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();
}
```

Возвращаемый результат:

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

Функция `find_node`

Прототип функции `find_node`

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

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

Описание

Функция `find_node` осуществляет поиск узла с именем `fileName`, среди всех узлов-потомков (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();
}
```

Возвращаемый результат:

```
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 DataBase15.lib	C

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();
}
```

Возвращаемый результат:

```
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 DataBase15.lib	C

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

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);
    getch();
}
```

Возвращаемый результат:

```
Wide: 100N <int>
```

Функция directory

Прототип функции directory

```
void directory(NODE *current)
```

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

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

```
#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();
}
```

Возвращаемый результат:

```
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 DataBase15.lib	C

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

```
#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 DataBase15.lib	C

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

```
#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();
}
```

Возвращаемый результат:

```
root.NewNode.NewNode2
```

Функция add_value

Прототип функции add_value

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

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

Описание

Функция **add_value** добавляет новое значение для узла в базе данных.

Параметры

- **currentPtr**
Указатель на узел типа **NODE**, для которого требуется добавить значение.
- **qualif**
Спецификатор в виде си-строки, который будет присвоен вновь созданному значению. Не может превышать 255 символов по длине
- **type**
Тип значения (INT, CHAR, 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

```
#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();
}
```

Возвращаемый результат:

```
Value2: 15PI <int> Value1: 2016 <int>
```

Функция delete_val

Прототип функции delete_val

```
int delete_val(VALUE* toDelete, NODE* start)
```

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

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();
}
```

Возвращаемый результат:

```
1: 259 <int>
Empty
```

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 DataBase15.lib	C

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, CHAR, FLOAT, DOUBLE)

Возвращаемое значение

Функция типа void. Нет возвращаемого значения

Пример: исходный код программы

Возвращаемый результат:

```
#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 `copy`

Прототип функции `copy`

`VALUE* copy(VALUE* valueFrom)`

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

Описание

Функция `copy` 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();
}
```

Возвращаемый результат:

```
Empty
1: 259 <int>
1: 259 <int>
```

Функция paste

Прототип функции paste

```
int paste(NODE* pasteTo, VALUE* copy)
```

Заголовочный файл

Название и библиотека	Язык
DataBase15.h DataBase15.lib	C

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

Возвращаемый результат:

```
#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();
}
```

```
Empty
1: 259 <int>
1: 259 <int>
```