1. Реализовать структуру для хранения представленных в задании данных. Информация в элемент структуры вводится со стандартного потока ввода.
2. Для хранения набора структур использовать указанный вид дерева (сбалансированное (АВЛ) дерево, двоичная heap (куча), дерево поиска). Написать функцию добавления элемента в дерево add(), удаления remove(), поиска и извлечения элемента find(), сохранения дерева в файл* save() и извлечения элементов дерева из файла* load(). Реализовать указанный вид поиска: под фильтром предполагаем удовлетворение условию >, <, ==, под поиском – только ==.
3. Хотя бы одна реализованная функция должна быть рекурсивной. Должна присутствовать функция, перечисляющая содержимое дерева.
4. Фильтр возвращает новое дерево элементов, удовлетворяющих условию.

*кроме фильтра по указанным полям, можно написать универсальный фильтр по произвольному полю.

Вариант 1

Структура «Государство».
Минимальный набор полей: название, столица, язык, численность населения, площадь.
Структура дерева: дерево поиска.
Поиск по названию страны, фильтр по площади.

Вариант 2

Структура «Человек».
Минимальный набор полей: фамилия, имя, пол, рост, вес, дата рождения, телефон, адрес.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по фамилии, фильтр по дате рождения.

Вариант 3

Структура «Школьник».
Минимальный набор полей: фамилия, имя, пол, класс, дата рождения, адрес.
Структура дерева: двоичная куча (heap).
Поиск по фамилии, фильтр по классу.

Вариант 4

Структура «Покупатель».
Минимальный набор полей: фамилия, имя, город, улица, номера дома и квартиры, номер счёта, средний сумма чека.
Структура дерева: дерево поиска.
Поиск по фамилии, фильтр по средней сумме чека.

Вариант 5

Структура «Пациент».
Минимальный набор полей: фамилия, имя, дата рождения, телефон, адрес, номер карты, группа крови.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по фамилии, фильтр по группе крови.

Вариант 6

Структура «Команда».
Минимальный набор полей: название, город, число побед, поражений, ничьих, количество очков.
Структура дерева: двоичная куча (heap).

Поиск по названию, фильтр по числу побед.

Вариант 7

Структура «Стадион».
Минимальный набор полей: название, виды спорта, год постройки, вместимость, количество арен.
Структура дерева: дерево поиска.
Поиск по названию, фильтр по году постройки.

Вариант 8

Структура «Автовладелец».
Минимальный набор полей: фамилия, имя, номер автомобиля, дата рождения, номер техпаспорта.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по номеру техпаспорта, фильтр по фамилии.

Вариант 9

Структура «Автомобиль».
Минимальный набор полей: марка, цвет, серийный номер, количество дверей, год выпуска, цена.
Структура дерева: двоичная куча (heap).
Поиск по серийному номеру, фильтр по цене.

Вариант 10

Структура «Фильм».
Минимальный набор полей: фамилия, имя режиссёра, название, страна, год выпуска, стоимость, доход.
Структура дерева: дерево поиска.
Поиск по названию, фильтр по доходу.

Вариант 11

Структура «Альбом».
Минимальный набор полей: имя или псевдоним исполнителя, название альбома, количество композиций, год выпуска.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по названию альбома, фильтр по имени.

Вариант 12

Структуры «Комплексное число», «Уравнение с комплексными коэффициентами».
Минимальный набор полей: коэффициенты в уравнении , набор корней уравнения.
Структура дерева: двоичная куча (heap).
Поиск по коэффициентам.
Дополнительная функция: поиск корней уравнения.

Вариант 13

Структуры «Комплексное число», «Дробь».
Минимальный набор полей: комплексные коэффициенты a, b, c, d в выражении , набор корней уравнения.
Структура дерева: дерево поиска.
Поиск по коэффициентам.
Дополнительная функция: поиск значения выражения.

Вариант 14

Структура «Матрица».
Минимальный набор полей: размерности, коэффициенты матрицы.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по следу, фильтр по размерности.
Дополнительная функция: вычисление определителя и следа, произведения.

Вариант 15

Структура «Вектор».
Минимальный набор полей: размерность, коэффициенты вектора.
Структура дерева: двоичная куча (heap).
Поиск по длине, фильтр по размерности.
Дополнительная функция: вычисление длины вектора, скалярного произведения двух векторов.

Вариант 16

Структура «Определённый интеграл».
Минимальный набор полей: указатель на подынтегральную функцию, шаг вычисления (точность), пределы интегрирования.
Структура дерева: дерево поиска.
Фильтр по значению интеграла.
Дополнительная функция: приближённое вычисление значения интеграла.

Вариант 17

Структура «Уравнение».
Минимальный набор полей: указатель на функцию f(x) в уравнении, шаг вычисления (точность), левая и правая границы области, в которой производится поиск решения уравнения.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по решениям уравнения.
Дополнительная функция: приближённое вычисление решения уравнения.

Вариант 18

Структура «Государство».
Минимальный набор полей: название, столица, язык, численность населения, площадь.
Структура дерева: двоичная куча (heap).
Поиск по численности населения, фильтр по языку.

Вариант 19

Структура «Человек».
Минимальный набор полей: фамилия, имя, пол, рост, вес, дата рождения, телефон, адрес.
Структура дерева: дерево поиска.
Поиск по фамилии, фильтр по адресу.

Вариант 20

Структура «Школьник».
Минимальный набор полей: фамилия, имя, пол, класс, дата рождения, адрес.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по фамилии, фильтр по дате рождения.

Вариант 21

Структура «Покупатель».

Минимальный набор полей: фамилия, имя, город, улица, номера дома и квартиры, номер счёта.
Структура дерева: двоичная куча (heap).
Поиск по номеру счета, фильтр по городу.

Вариант 22

Структура «Пациент».
Минимальный набор полей: фамилия, имя, дата рождения, телефон, адрес, номер карты, группа крови.
Структура дерева: дерево поиска.
Поиск номеру карты, фильтр по группе крови..

Вариант 23

Структура «Команда».
Минимальный набор полей: название, город, число побед, поражений, ничьих, количество очков.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по названию, фильтр по числу побед.

Вариант 24

Структура «Автовладелец».
Минимальный набор полей: фамилия, имя, номер автомобиля, дата рождения, номер техпаспорта.
Структура дерева: двоичная куча (heap).
Поиск по номеру автомобиля, фильтр по имени.

Вариант 25

Структура «Государство».
Минимальный набор полей: название, столица, язык, численность населения, площадь.
Структура дерева: дерево поиска.
Поиск по столице, фильтр по численности населения.

Вариант 26

Структура «Программист».
Минимальный набор полей: фамилия, имя, email, skype, telegram, основной язык программирования, текущее место работы, уровень(число от 1 до 10).
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по email, фильтр по уровню.

Вариант 27

Структура «Спортсмен».
Минимальный набор полей: фамилия, имя, возраст, гражданство, вид спорта, количество медалей.
Структура дерева: двоичная куча (heap).
Поиск по фамилии, фильтр по числу медалей.

Вариант 28

Структура «Полином».
Минимальный набор полей: степень полинома, коэффициенты
Структура дерева: дерево поиска.
Поиск по коэффициентам, фильтр по степени полинома.
Дополнительная функция: поиск корня полинома на указанном отрезке.

Вариант 29

Структура «Профиль в соц.сети».
Минимальный набор полей: псевдоним, адрес страницы, возраст, количество друзей, интересы, любимая цитата.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по псевдониму, фильтр по количеству друзей.

Вариант 30

Структура «Велосипед».
Минимальный набор полей: марка, тип, тип тормозов, количество колес, диаметр колеса, наличие амортизаторов, детский или взрослый.
Структура дерева: двоичная куча (heap).
Поиск по марке, фильтр по диаметру колеса.

Вариант 31

Структура «Ноутбук».
Минимальный набор полей: производитель, модель, размер экрана, процессор, количество ядер, объем оперативной памяти, объем диска, тип диска, цена.
Структура дерева: дерево поиска.
Поиск по марке, фильтр по цене.

Вариант 32

Структура «Смартфон».
Минимальный набор полей: марка, размер экрана, количество камер, объем аккумулятора, максимальное количество часов без подзарядки, цена.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по марке, фильтр по размеру экрана.

Вариант 33

Структура «Фотоаппарат».
Минимальный набор полей: производитель, модель, тип, размер матрицы, количество мегапикселей, вес, тип карты памяти, цена.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по модели, фильтр по размеру матрицы.

Вариант 33

Структура «Супергерой».
Минимальный набор полей: псевдоним, настоящее имя, дата рождения, пол, суперсила, слабости, количество побед, рейтинг силы.
Структура дерева: двоичная куча (heap).
Поиск по псевдониму, фильтр по рейтингу силы.

Вариант 34

Структура «Сериал».
Минимальный набор полей: название, режиссер, количество сезонов, популярность, рейтинг, дата запуска, страна.
Структура дерева: дерево поиска.
Поиск по режиссеру, фильтр по популярности.

Вариант 35

Структура «Программа».

Минимальный набор полей: название, версия, лицензия, есть ли версия для android, iOS, платная ли, стоимость, разработчик, открытость кода, язык кода.
Структура дерева: сбалансированное (АВЛ) дерево.
Поиск по названию, фильтр по платности.

Вариант 36

Структура «Сайт».
Минимальный набор полей: название, адрес, дата запуска, язык, тип (блог, интернет-магазин и т.п.), cms, дата последнего обновления, количество посетителей в сутки.
Структура дерева: двоичная куча (heap).
Поиск по адресу, фильтр по количеству посетителей.

Пример кода:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
 * Определяет, начинается ли строка с подстроки
 * @param TestArray      Строка, в которой проверяем
 * @param StringToFind   Строка, которую ищем
 * @param count          Число символов для проверки
 * @return 1 - да, 0 - нет
 **/
int StringStartsWith(char TestArray[], char StringToFind[], int count)
{
        // TODO: необходимодобавить проверку на случай, если строки разной
длины
        for(int i=0;i<count;i++)
        {
                if(TestArray[i]!=StringToFind[i])
                        return 0;
        }
        return 1;
}

/**
 * Структура элемента в дереве "Книга"
 * @property name     Название книги
 * @property author   Автор книги
 * @property Left     Указатель на левое поддерево
 * @property Right    Указатель на правое поддерево
 * @property Parent   Указатель на родительский элемент
 **/
typedef struct BookNode
{
        char name[80];
        char author[80];
```

```
        struct BookNode* Left;
        struct BookNode* Right;
        struct BookNode* Parent;
} BookNode;

/**
 * Инициализация элемента дерева начальными значениями
 * @param B  Указатель на элемент дерева
 **/
void InitNode(BookNode* B)
{
        B->Left = NULL;
        B->Right = NULL;
        B->Parent = NULL;
}

/**
 * Печать в консоль информации о книге
 * @param B  Указатель на элемент дерева
 **/
void print(BookNode* B)
{
        if(B!=NULL)
                printf("\nName: %s\nAuthor: %s", B->name, B->author);
}

/**
 * Дерево книг
 * @property Root  Указатель на корень дерева
 **/
typedef struct BookTree
{
        BookNode* Root;
} BookTree;

/**
 * Инициализация дерева начальным значением
 * @param B  Указатель на структуру дерева
 **/
void InitTree(BookTree* T)
{
        T->Root = NULL;
}

/**
 * Добавление нового элемента в дерево (с рекурсивным проходом)
 * @param T        Указатель на структуру дерева
 * @param N        Указатель на новый элемент
 * @param Current  Указатель на текущий элемент (при первом вызове равен
NULL)
 **/
void Add_R(BookTree* T, BookNode* N, BookNode* Current)
{
```

```
        if(T==NULL) return;
        if(T->Root==NULL)
        {
                T->Root = N;
                return;
        }
        if(Current==NULL) Current = T->Root;
        // Сравниваем элементы по названию
        int c = strcmp(Current->name, N->name);
        // Если новое имя больше текущего, то переходим в левое поддерево
        if(c<0)
        {
                if(Current->Left!=NULL)
                        Add_R(T,N,Current->Left);
                else
                        Current->Left = N;
        }
        // Если новое имя меньше текущего, то переходим в правое поддерево
        if(c>0)
        {
                if(Current->Right!=NULL)
                        Add_R(T,N,Current->Right);
                else
                        Current->Right = N;
        }
        if(c==0)
        {
                //нашли совпадение имён
        }
        return;
}

/**
 * Добавление нового элемента в дерево (нерекурсивная версия)
 * @param T        Указатель на структуру дерева
 * @param N        Указатель на новый элемент
 **/
void Add(BookTree* T, BookNode* N)
{
        if(T==NULL) return;
        if(T->Root==NULL)
        {
                T->Root = N;
                return;
        }
        BookNode* Current = T->Root;
        while(Current!=NULL)
        {
                int c = strcmp(Current->name, N->name);
                if(c<0)
                {
                        if(Current->Left!=NULL)
                        {
                                Current = Current->Left;
```

```
                        continue;
                    }
                    Current->Left = N;

                    break;
                }
                if(c>0)
                {
                    if(Current->Right!=NULL)
                    {
                        Current = Current->Right;
                        continue;
                    }
                    Current->Right = N;

                    break;
                }
                if(c==0)
                {
                    //нашли совпадение имён
                }
        }
}

/**
 * Удаление элемента из дерева по названию
 * @param T          Указатель на структуру дерева
 * @param BookName   Название книги
 **/
void Remove(BookTree* T, char BookName[])
{// Здесь ваш код }


/**
 * Поиск элемента в дереве по названию
 * @param T          Указатель на структуру дерева
 * @param BookName   Название книги
 * @return Найденный элемент или NULL
 **/
BookNode* Find(BookTree* T, char BookName[])
{
        if(T==NULL) return NULL;
        BookNode* Current = T->Root;
        while(Current!=NULL)
        {
                int c = strcmp(Current->name, BookName);
                if(c<0)
                {
                    Current = Current->Left;
                    continue;
                }
                if(c>0)
                {
                    Current = Current->Right;
```

```
                        continue;
                }
                if(c==0)
                        return Current;
        }
        return NULL;
}

/**
 * Поиск элемента в дереве по названию (рекурсивная версия)
 * @param T          Указатель на структуру дерева
 * @param BookName   Название книги
 * @param Current    Указатель на текущий элемент (при первом вызове равен
NULL)
 * @return Найденный элемент или NULL
 **/
BookNode* Find_R(BookTree* T, char BookName[], BookNode* Current)
{
        if(T==NULL) return NULL;
        if(Current==NULL) Current = T->Root;
        int c = strcmp(Current->name, BookName);
        if(c<0)
        {
                if(Current->Left!=NULL) return Find_R(T, BookName, Current-
>Left);
                else return NULL;
        }
        if(c>0)
        {
                if(Current->Right!=NULL) return Find_R(T, BookName, Current-
>Right);
                else return NULL;
        }
        if(c==0)
                        return Current;
        return NULL;
}

/**
 * Вызывает переданную по указателю функцию для каждого элемента дерева
(рекурсивно обходит дерево)
 * @param Node  Указатель на текущий элемент дерева
 * @param f     Указатель на функцию-действие
 **/
void MakeAction(BookNode* Node, void (*f)(BookNode*))
{
        if(Node!=NULL)
                f(Node);
        if(Node->Left!=NULL)
                MakeAction(Node->Left, f);
        if(Node->Right!=NULL)
                MakeAction(Node->Right, f);
}
```

```
/**
 * Загружает дерево из файла
 * @param T          Дерево
 * @param FileName   Имя файла
 **/
void Load(BookTree* T, char FileName[])
{
        if(T==NULL) return;
        FILE* fp;
        fp = fopen(FileName, "r");
        if(fp==NULL)
        {
                printf("\nCouldn't open the file: %s", FileName);
                return;
        }
        char Name[80]; char Author[80];
        while(1)
        {
                //fscanf(fp, "%s%s", Name, Author);
                fgets(Name, 80, fp);
                fgets(Author, 80, fp);
                //if(strcmp(Name, "NULL")!=0 && strcmp(Author, "NULL")!=0)
                if(!StringStartsWith(Name, "NULL", 4) && !
StringStartsWith(Author, "NULL", 4))
                {
                        BookNode* N = (BookNode*)malloc(sizeof(BookNode));
InitNode(N);
                        strcpy(N->author, Author); strcpy(N->name, Name);
                        Add(T, N);
                }
                else
                        break;
        }
        fclose(fp);
}

/**
 * Сохраняем элемент в файл
 * @param Node  Элемент
 * @param fp    Указатель на файл
 **/
void SaveNode(BookNode* Node, FILE* fp)
{
        if(fp==NULL)
                return;
        if(Node!=NULL)
        {
                //fprintf(fp, "%s%s", Node->name, Node->author);
                fputs(Node->name, fp);
                fputs("\n", fp);
                fputs(Node->author, fp);
                fputs("\n", fp);
                if(Node->Left!=NULL)
                        SaveNode(Node->Left, fp);
```

```
                if(Node->Right!=NULL)
                        SaveNode(Node->Right, fp);
        }
}

/**
 * Сохраняем дерево в файл
 * @param T          Дерево
 * @param FileName   Имя файла
 **/
void Save(BookTree* T, char FileName[])
{
        if(T==NULL) return;
        FILE *ptr;
        ptr = fopen(FileName, "w");
        if(ptr==NULL)
        {
                printf("\nCouldn't open the file: %s", FileName);
                return;
        }

        SaveNode(T->Root,ptr);
        fputs("NULL\n", ptr);
        fputs("NULL", ptr);
        fclose(ptr);
}

/**
 * Производит прямой обход (CLR) дерева и вызывает для каждого узла
переданную функцию
 * @param Node  Указатель на текущий элемент дерева
 * @param f     Указатель на функцию-действие
 **/
void PreOrder(BookNode* Node, void (*f)(BookNode*))
{
        if(Node!=NULL)
                f(Node);
        if(Node->Left!=NULL)
                PreOrder(Node->Left, f);
        if(Node->Right!=NULL)
                PreOrder(Node->Right, f);
}

/**
 * Производит центрированный (симметричный) (LCR) обход дерева и вызывает для
каждого узла переданную функцию
 * @param Node  Указатель на текущий элемент дерева
 * @param f     Указатель на функцию-действие
 **/
void InOrder(BookNode* Node, void (*f)(BookNode*))
{
        if(Node->Left!=NULL)
                InOrder(Node->Left, f);
        if(Node!=NULL)
```

```
                f(Node);
        if(Node->Right!=NULL)
                InOrder(Node->Right, f);
}

/**
 * Производит обратный обход (LRC) дерева и вызывает для каждого узла
переданную функцию
 * @param Node  Указатель на текущий элемент дерева
 * @param f     Указатель на функцию-действие
 **/
void PostOrder(BookNode* Node, void (*f)(BookNode*))
{
        if(Node->Left!=NULL)
                PostOrder(Node->Left, f);
        if(Node->Right!=NULL)
                PostOrder(Node->Right, f);
        if(Node!=NULL)
                f(Node);
}

int main()
{
        // Создаем дерево
        BookTree Tree;
        InitTree(&Tree);
        BookTree* ptr = &Tree;

        // Создаем несколько элементов
        BookNode* London = (BookNode*)malloc(sizeof(BookNode));
InitNode(London);
        strcpy(London->author, "J.London"); strcpy(London->name, "Little Lady
of the big house");
        BookNode* T = (BookNode*)malloc(sizeof(BookNode)); InitNode(T);
        strcpy(T->author, "L.Tolstoy"); strcpy(T->name, "War and Peace");
        BookNode* S = (BookNode*)malloc(sizeof(BookNode)); InitNode(S);
        strcpy(S->author, "M.Sholohov"); strcpy(S->name, "Calm Don");
        BookNode* Scott = (BookNode*)malloc(sizeof(BookNode));
InitNode(Scott);
        strcpy(Scott->author, "W.Scott"); strcpy(Scott->name, "Weverly");

        // Добавляем элементы в дерево
        Add_R(ptr, London, NULL);
        Add_R(ptr, T, NULL);
        Add_R(ptr, S, NULL);
        Add_R(ptr, Scott, NULL);

        // Поиск элемента
        BookNode* B = Find_R(ptr, "Weverly", NULL);
        if(B!=NULL)
                print(B);
        printf("\n");
```

```
        // Вывод узлов дерево с обходом в глубину тремя способами
        printf("\n-----\nPreorder:");
        void (*f_ptr)(BookNode*); f_ptr = print;
        PreOrder(Tree.Root, f_ptr);
        printf("\n-----\nInorder:");
        InOrder(Tree.Root, f_ptr);
        printf("\n-----\nPostorder:");
        PostOrder(Tree.Root, f_ptr);

        // Здесь необходимо вызвать освобождение памяти из-под элементов
дерева

        char c; scanf("%c", &c);
        return 0;
}
```