top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

ОБЪЕКТНО -ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА C++

Урок №9

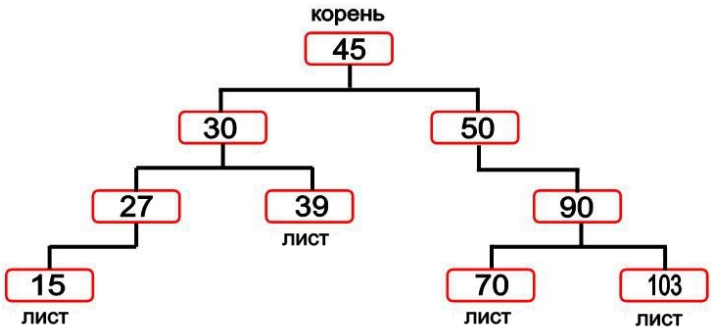
Деревья и работа с файлами

Содержание

Бинарное дерево

Сегодня мы с вами познакомимся с новой, отнюдь не линейной структурой данных. Называется эта структура двоичное (или бинарное) дерево. Для начала дадим определение самой структуре, а затем рассмотрим несколько, терминов использующихся при работе с ней.

Бинарное дерево (*binary tree*) ***— это упорядоченная древовидная динамическая структура.*** Каждый элемент (*узел*) дерева имеет не более двух элементов следующих за ним (*потомков*) и не более одного предыдущего (*родителя*). Рассмотрим схематичное изображение бинарного дерева:



Комментарии к изображению дерева. Терминология.

1. Самый главный принцип бинарного дерева заключается в том, что для каждого узла выполняется правило: ***в левой ветке содержатся только те ключи, которые имеют значения, меньшие, чем значение данного***

узла. В правой же ветке содержатся ключи, имеющие значения, большие, чем значение данного узла.

2. Каждый узел может иметь два, одного или ни одного потомка.
3. ***Лист*** — узел, не имеющий потомков.
4. Узел является родительским для своих потомков и дочерним для своего предка.
5. ***Левый потомок*** — дочерний узел слева от текущего узла.
6. ***Правый потомок*** — дочерний узел справа от текущего узла.
7. ***Корень*** — основной узел, не имеющий родителей.
8. Каждый узел состоит из четырех частей:
   - Значение.
   - Указатель на родителя.
   - Указатель на левого потомка.
   - Указатель на правого потомка.

Примечание: Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем самостоятельного дерева.

Организация работы с деревом

При работе с деревьями обычно используются рекурсивные алгоритмы. Использование рекурсивных функций менее эффективно, поскольку многократный вызов функции расходует системные ресурсы. Тем не менее, в данном случае использование рекурсивных функций является оправданным, поскольку не рекурсивные функции для работы с деревьями гораздо сложнее и для написания, и для восприятия кода программы. Здесь мы приведем схематичные алгоритмы работы с деревом. А, осмысленный, практический пример смотрите в следующем разделе урока. Вот некоторые значения, которые мы будем использовать в схемах:

- **x** — вершина бинарного дерева
- **left[x]** — левое поддерево
- **right[x]** — правое поддерево
- **key[x]** — ключ
- **p[x]** — родитель вершины

Обход всего дерева

Печать(x)
Начало

1. Если x не равен NULL
2. Тогда Печать(left[x])
3. Напечатать key[x]
4. Печать(right[x])

Конец

Поиск значения

Поиск (x, k)

Начало

1. Пока x не равен NULL и k не равно key[x]
2. Начало
3. Если k меньше key[x]
4. Тогда x равно left[x]
5. Иначе x равно right[x]
6. Конец
7. Вернуть x

Конец

Нахождение минимума и максимума

Минимум(x)

Начало

1. Пока left[x] не равен NULL
2. Начало
3. x = left[x]
4. Конец
5. Вернуть x

Конец

Максимум (x)

Начало

1. Пока right[x] не равен NULL
2. Начало
3. x = right[x]
4. Конец
5. Вернуть x

Конец

Получение следующего и предыдущего элементов

ПолучитьСледующийЭлемент(x)

Начало

1. Если right[x] не равен NULL, тогда вернуть Минимум(right[x])
2. y равно p[x]
3. Пока y не равно NULL и x равно right[x]
4. Начало
5. x равно y
6. y равно p[y]
7. Вернуть y

Конец

ПолучитьПредыдущийЭлемент(x)

Начало

1. Если left[x] не равен NULL, тогда вернуть Максимум(left[x])
2. y равно p[x]
3. Пока y не равно NULL и x равно left[x]
4. Начало
5. x равно y
6. y равно p[y]
7. Вернуть y Конец

Добавление значения

Вставка(T, z)

Начало

1. y равно NULL
2. x равно root[T]
3. Пока x не равно NULL
4. Начало

5. y равно x
6. Если key[z] меньше key[x], тогда x равно left[x]
7. Иначе x равно right[x]
8. Конец
9. p[z] равно y
10. Если y равно NULL, тогда root[T] равно z
11. Иначе если key[z] меньше key[y], тогда left[y] равно z
12. Иначе right[y] равно z

Конец

Удаление значения

Удаление (T, z)

Начало

1. Если left[z] равно NULL или right[z]=NULL, тогда y равно z
2. Иначе y равно ПолучитьСледующийЭлемент(z)
3. Если left[y] не равно NULL, тогда x равно left[y]
4. Иначе x равно right[y]
5. Если x не равно NULL, тогда p[x] равно p[y]
6. Если p[y] равно NULL, тогда root[T] равно x
7. Иначе если y равно left[p[y]], тогда left[p[y]] равно x
8. Иначе right[p[y]] равно x
9. Если y не равен z, тогда key[z] равно key[y]
10. Копируем дополнительные данные связанные с y
11. Удалить y

Конец

Применение

В заключение, отметим, что организация данных с помощью бинарных деревьев часто позволяет значительно сократить время поиска нужного элемента. Поиск элемента в линейных структурах данных обычно осуществляется путем последовательного перебора всех элементов, присутствующих в данной структуре. Поиск по дереву не требует перебора всех элементов, поэтому занимает значительно меньше времени. Максимальное число шагов при поиске по дереву равно высоте данного дерева, т.е. количеству уровней в иерархической структуре дерева.

Реализация бинарного дерева

Использования бинарного дерева для хранения результатов игр команд английского чемпионата по футболу.

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

using namespace std;
struct Elem
{
    int OwnerPoints;    //Очки хозяина
    int OppPoints;      //Очки соперника
    char Match[10];     //Счет
    char Name[20];      //Команда
    char Opponent[20];  //Соперник

    Elem * left, * right, * parent;
};
```

```
class Tree
{
    //корень
    Elem * root;
public:
    Tree();
    ~Tree();
    //печать от указанного узла
    void Print(Elem * Node);
    //поиск от указанного узла
    Elem * Search(Elem * Node, char * key);
    //min от указанного узла
    Elem * Min(Elem * Node);
    //max от указанного узла
    Elem * Max(Elem * Node);
    //следующий для указанного узла
    Elem * Next(Elem * Node);
    //предыдущий для указанного узла
    Elem * Previous(Elem * Node);
    //вставка узла
    void Insert(Elem * z);
    //удаление ветки для указанного узла,
    //0 - удаление всего дерева
    void Del(Elem * z = 0);
    //получить корень
    Elem * GetRoot();
};

Tree::Tree()
{
    root = NULL;
}

Tree::~Tree()
{
    Del();
}
```

```
//Рекурсивный обход дерева
void Tree::Print(Elem * Node)
{
    if(Node != 0)
    {
        Print(Node->left);
        cout << Node->Name
            << Node->Match
            << Node->Opponent
            << endl;
        Print(Node->right);
    }
}

Elem * Tree::Search(Elem * Node, char * k)
{
    //Пока есть узлы и ключи не совпадают
    while(Node != 0 && strcmp(k, Node->Name) != 0)
    {
        if(strcmp(k, Node->Name) < 0)
            Node = Node->left;
        else
            Node = Node->right;
    }
    return Node;
}

Elem * Tree::Min(Elem * Node)
{
    //Поиск самого "левого" узла
    if(Node != 0)
        while(Node->left != 0)
            Node = Node->left;
    return Node;
}
Elem * Tree::Max(Elem * Node)
{
```

```
    //Поиск самого "правого" узла
    if(Node != 0)
        while(Node->right != 0)
            Node = Node->right;

    return Node;
}

Elem * Tree::Next(Elem * Node)
{
    Elem * y = 0;
    if(Node != 0)
    {
        //если есть правый потомок
        if(Node->right != 0)
            return Min(Node->right);

        //родитель узла
        y = Node->parent;
        //если Node не корень и Node справа
        while(y != 0 && Node == y->right)
        {
            //Движемся вверх
            Node = y;
            y = y->parent;
        }
    }
    return y;
}

Elem * Tree::Previous(Elem * Node)
{
    Elem * y = 0;
    if(Node != 0)
    {
        //если есть левый потомок
        if(Node->left != 0)
            return Max(Node->left);
```

```
        //родитель узла
        y = Node->parent;
        //если Node не корень и Node слева
        while(y != 0 && Node == y->left)
        {
            //Движемся вверх
            Node = y;
            y = y->parent;
        }
    }
    return y;
}

Elem * Tree::GetRoot()
{
    return root;
}

void Tree::Insert(Elem * z)
{
    //потомков нет
    z->left = NULL;
    z->right = NULL;

    Elem * y = NULL;
    Elem * Node = root;

    //поиск места
    while(Node != 0)
    {
        //будущий родитель
        y = Node;
        if(strcmp(z->Name, Node->Name) < 0)
            Node = Node->left;
        else
            Node = Node->right;
    }
```

```
    //заполняем родителя
    z->parent = y;

    if(y == 0) //элемент первый (единственный)
        root = z;
    //чей ключ больше?
    else if(strcmp(z->Name, y->Name) < 0)
        y->left = z;
    else
        y->right = z;
}

void Tree::Del(Elem * z)
{
    //удаление куста
    if(z != 0)
    {
        Elem * Node, * y;

        //не 2 ребенка
        if(z->left == 0 || z->right == 0)
            y = z;
        else
            y = Next(z);

        if(y->left != 0)
            Node = y->left;
        else
            Node = y->right;

        if(Node != 0)
            Node->parent = y->parent;
        //Удаляется корневой узел?
        if(y->parent == 0)
            root = Node;
        else if(y == y->parent->left)
```

```
            //слева от родителя?
            y->parent->left = Node;
        else
            //справа от родителя?
            y->parent->right = Node;
        if(y != z)
        {
            //Копирование данных узла
            strcpy(z->Name, y->Name);
            strcpy(z->Opponent, y->Opponent);
            strcpy(z->Match, y->Match);
            z->OppPoints = y->OppPoints;
            z->OwnerPoints = y->OwnerPoints;
        }

        delete y;
    }
    else //удаление всего дерева
        while(root != 0)
            Del(root);
}

//Турнирная таблица
Tree tournament;

void Game(char Commands[][20], int N)
{
    int i, j;
    int p1, p2; //Счет

    //Каждая команда играет с каждой по 2 раза -
    //дома и в гостях
    int k;

    Elem * temp;
    for(k = 0; k < 2; k++)
        for(i = 0; i < N - 1; i++)
        {
```

```
for(j = i + 1; j < N; j++)
{
    temp = new Elem;
    if(k == 0)
    {
        //1 игра
        strcpy(temp->Name, Commands[i]);
      strcpy(temp->Opponent, Commands[j]);
    }
    else
    {
       //2 игра
       strcpy(temp->Name, Commands[j]);
       strcpy(temp->Opponent, Commands[i]);
    }

    p1 = rand() % 6;
    p2 = rand() % 6;

    if(p1 > p2)
    {
       temp->OwnerPoints = 3;
       temp->OppPoints = 0;
    }
    else if(p1 == p2)
    {
       temp->OwnerPoints = 1;
       temp->OppPoints = 1;
    }
    else
    {
       temp->OwnerPoints = 0;
       temp->OppPoints = 3;
    }

    //Запись счета
    sprintf(temp->Match, " %d : %d ", p1, p2);
```

```
                //Добавление записи
                tournament.Insert(temp);
            }
        }
}

void main()
{
    srand(time(0));

    const N = 4;
    char Commands[4][20] =
    {
        "Arsenal",
        "Liverpool",
        "Lids United",
        "Manchester United"
    };

    //Игра
    Game(Commands, N);
    //Вывод результатов
    tournament.Print(tournament.GetRoot());

}
```

Файлы

Введение

А, теперь, мы поговорим о файлах. Для программиста работа с файлами имеет очень большое значение. Вы прекрасно понимаете, что длительное хранение информации только в оперативной памяти — невозможно. Файл же, хранит информацию на диске, что позволяет обратиться к ней в любой момент. Поэтому изучение данной темы необходимо нам с вами как воздух. Приступим. Для начала ответьте себе на вопрос: «А, что — есть файл». Вариантов у вас, наверняка множество. Однако, давайте дадим, этому понятию четкое определение. Итак,

Файл — это именованный блок информации, расположенный на носителе информации.

Любой файл обладает следующим рядом особенностей:

- Файл не может располагаться на диске непрерывно, однако пользователю файл предоставляется цельным блоком последовательной байтовой информации.
- Название файла не может содержать символы: < > : " / \ |.
- Большинство файлов обладает расширением — сочетанием символов, с помощью которых операционная система определяет тип файла. Расширение — необязательная часть.
- У каждого файла есть, так называемые атрибуты, которые, например, определяют уровни доступа

к нему. Используя атрибуты, операционная система узнает, как нужно и, главное, можно, работать с данным файлом.

Теперь введем несколько новых терминов:

1. ***Дескриптор файла*** — уникальный номер, который операционная система присваивает любому открытому файлу, что бы отличать его от других. Когда файл закрывается, система «отбирает» у него дескриптор. Именно это уникальное число мы будем использовать для работы с конкретным файлом в наших программах.
2. ***Файловый указатель*** — переменная, которая автоматически присваивается открытому файлу и хранит текущую позицию в файле. Она перемещается по файлу в момент процессов записи и чтения. Для большего понимания, вы можете представить данную переменную в виде курсора в любом текстовом редакторе.

Существует несколько разновидностей файлов. В Windows таких разновидностей две: ***текстовый файл и бинарный (двоичный) файл.*** Данная классификация в качестве критерия использует способ хранения информации. Итак:

Сравнительная таблица двух категорий.

Текстовый файл	Двоичный файл
Число, записанное в файл — запишется как текст, то есть размер занятого пространства будет равен количеству цифр в этом числе.	Число, записанное в файл — запишется в двоичном формате, и размер занятого пространства будет равен размеру типа данных в этом числе.

Текстовый файл	Двоичный файл
Прочитать данный файл можно с помощью любого текстового редактора.	Прочитать файл можно только с помощью особой программы.
При потере нескольких байт, информацию можно восстановить по смыслу.	Потеря нескольких байт в двоичном файле может быть необратима
Считывание информации программным путем затруднено, так как файл представляет собой единый тестовый блок.	Считывание информации программным путем облегчено, так как файл представляет собой набор блоков информации с конкретными типами данных.
Используется как файл для чтения, редактирования, вывода на печать.	Используется как файл для удобного чтения, хранения, записи информации програмным путем.

Примечание: Файл можно открыть в текстовом или в двоичном режиме. Внимание, не следует это отождествлять с текстовым и двоичным форматами!!! Разница между режимами открытия состоит в том, что при открытии файла в двоичном режиме информация будет в том же виде, в котором она хранится на диске или в памяти. А, вот, при текстовом режиме символ окончания строки (\n) будет заменяться на двойное сочетание (\r\n). Это как обычно связано с «не стыковкой» систем MS DOS и Windows.

Функции для работы с файлами библиотеки языка С

После краткого экскурса в теорию файлов, пора переходить к практике. Данный раздел урока посвящается описанию функций для работы с файлами. Эти функции понадобятся нам для решения задач с использованием файлов.

Функции библиотеки stdio.h

```
FILE *fopen(const char *filename, const char *mode)
```

Функция открывает файл.

filename — путь к файлу.
mode — тип доступа.

- **r** — чтение, если файла нет, то данная функция генерирует ошибку (возвращает 0).
- **w** — запись, если файла нет, то файл создаётся, если есть исходное содержимое удаляется.
- **a** — добавление в конец, если файла нет, то он создаётся.
- **r+** — чтение и запись (файл должен существовать).
- **w+** — чтение и запись (принцип работы как у w).
- **a+** — добавление и чтение (принцип работы как у a).

Примечание: Все вышеописанные режимы предназначены для текстового открытия файла. Для двоичного открытия перед режимом достаточно добавить букву b. Например, br.

Если функция отработала успешно, из неё возвращается указатель на открытый файл, в противном случае — нуль.

Примечание: Указатель на открытый файл принято хранить в типе данных FILE*.

```
int fclose( FILE *stream )
```

Функция закрывает файл.

stream — указатель на закрываемый файл.

Если всё проходит успешно, то данная функция возвращает 0, или EOF в случае ошибки.

Примечание: EOF (End Of File) — обозначение конца файла.

```
char *fgets( char *string, int n, FILE *stream )
```

Считывает строку начиная с текущей позиции. Считывание останавливается:

- ...если был найден символ перехода на новую строчку (он помещается в строку);
- ...если достигнут конец файла;
- ...если считано n–1 символов.

string — строка, в которую попадают считанные данные

n — количество элементов в string.

stream — указатель на открытый файл.

Если всё прошло успешно функция возвращает считанную строку, если произошла ошибка или достигнут конец файла возвращается 0.

```
int fputs( const char *string, FILE *stream )
```

Записывает строку в файл, начиная с текущей позиции.

string — строка для записи.

stream — указатель на открытый файл, куда производиться запись.

Если функция отрабатывает успешно из неё возвращается неотрицательное значение. При ошибке возвращается EOF.

```
size_t fread( void *buffer, size_t size,
              size_t count, FILE *stream )
```

Функция считывает данные из файла в буфер.

buffer — адрес массива, куда запишутся данные.

size — размер элемента массива в байтах.

count — максимальное количество элементов для считывания.

stream — указатель на открытый файл.

Функция возвращает количество считанных байт.

Примечание: Тип данных size_t определен в библиотеке *stdio.h* следующим образом: typedef unsigned int size_t;. Другими словами, это обычный без знаковый int.

```
size_t fwrite( const void *buffer, size_t size,
               size_t count, FILE *stream )
```

Функция записывает массив данных в файл.

buffer — адрес массива, где содержатся данные.

size — размер элемента массива в байтах.

count — максимальное количество элементов для записи в файл.

stream — указатель на открытый файл.

Функция возвращает количество записанных байт.

```
int feof( FILE *stream )
```

Функция проверяет достигнут ли конец файла.

stream — указатель на открытый файл.

Функция возвращает ненулевое значение, если достигнут конец файла, нуль возвращается в противном случае.

```
int _fileno( FILE *stream )
```

Данная функция возвращает дескриптор файла.

stream — указатель на открытый файл.

```
int fseek ( FILE *stream, int offset [, int whence] )
```

Устанавливает смещение в файле.

stream — указатель на открытый файл.

offset — смещение, измеряемое в байтах от начала файла.

whence — точка, от которой производится смещение.

- SEEK_SET (0) — начало файла,
- SEEK_CUR (1) — позиция текущего указателя файла,
- SEEK_END (2) — конец файла (EOF).

Функция возвращает значение 0, если указатель файла успешно перемещен, и ненулевое значение в случае неудачного завершения.

Функции библиотеки io.h

```
int _access( const char *path, int mode )
```

Функция определяет разрешения файла или директории.

path — путь к файлу или директории.

mode — флаги для проверки.

- 00 — проверка на существование,
- 02 — проверка на разрешение на запись,
- 04 — проверка на разрешение на чтение,
- 06 — проверка на чтение и запись.

Если разрешение есть, функция возвращает ноль, в случае отсутствия –1.

Примечание: Директории можно проверять только на существование.

```
long _filelength( int handle )
```

Возвращает размер файла в байтах.

handle — дескриптор файла.

В случае ошибки функция возвращает –1.

```
int _locking( int handle, int mode, long nbytes )
```

Блокирует или разблокирует байты файла начиная с текущей позиции в файле.

handle — дескриптор файла.
mode — тип блокировки.

- _LK_LOCK — блокирует байты, если заблокировать байты не получается попытка повторяется через 1 секунду. Если после 10 попыток байты не заблокируются функция генерирует ошибку и возвращает –1,
- _LK_NBLCK — блокирует байты, если заблокировать байты не получается функция генерирует ошибку и возвращает –1,
- _LK_NBRLCK — то же самое, что и _LK_NBLCK,
- _LK_RLCK — то же самое, что и _LK_LOCK,
- _LK_UNLCK — разблокировка байт, которые были до этого заблокированы.

nbytes — количество байт для блокировки.

Функция locking возвращает –1, если происходит ошибка и 0 в случае успеха.

Примечание: Для работы этой функции кроме io.h, нужно подключить sys/locking.h.

Пример программы. Копирование файлов

Задача. Реализовать программу производящую копирование одного файла в другой. Пользователь с клавиатуры вводит путь к файлу, который будет копироваться, а также путь и имя для копии.

```
#include <iostream>
#include <windows.h>

#include <io.h>
```

```
#include <stdio.h>
using namespace std;

//Функция, выводящая на экран строку
void RussianMessage(char *str){
    char message[100];
    //перевод строки из кодировки Windows
    //в кодировку MS DOS
    CharToOem(str,message);
    cout<<message;
}
//Функция копирования файла
bool CopyFile(char *source,char *destination){
    const int size=65536;
    FILE *src,*dest;
    //Открытие Файла
    if(!(src=fopen(source,"rb"))){
        return false;
    }
    //Получение дескриптора файла
    int handle=_fileno(src);

    //выделение памяти под буффер
    char *data=new char[size];
    if(!data){
        return false;
    }

    //Открытие файла, куда будет производиться копирование
    if(!(dest=fopen(destination,"wb"))){
        delete []data;
        return false;
    }

    int realsize;
    while (!feof(src)){
        //Чтение данных из файла
        realsize=fread(data,sizeof(char),size,src);
```

```
        //Запись данных в файл
        fwrite(data,sizeof(char),realsize,dest);
    }

    //Закрытие файлов
    fclose(src);
    fclose(dest);
    return true;
}

void main(){
    //MAX_PATH - Константа, определяющая
    //максимальный размер пути.
    //Даная константа содержится в stdlib.h
    char source[_MAX_PATH];
    char destination[_MAX_PATH];
    char answer[20];
    RussianMessage("\nВведите путь и название
                    копируемого файлу:\n");
    //Получение пути к первому файлу
    cin.getline(source,_MAX_PATH);
    //Проверка Существует ли файл
    if(_access(source,00)==-1){
            RussianMessage("\nУказан неверный путь
                            или название файла\n");
        return;
    }

    RussianMessage("\nВведите путь и название нового
                    файла:\n");
    //Получение пути к второму файлу
    cin.getline(destination,_MAX_PATH);
    //Проверка на существование файла
    if(_access(destination,00)==0){
            RussianMessage("\nТакой Файл уже
                           существует  перезаписать
                           его(1 - Да /2 - Нет)?\n");
```

```
        cin.getline(answer,20);
        if(!strcmp(answer,"2")){
            RussianMessage("\nОперация отменена\n");
            return;
        }
        else if(strcmp(answer,"1")){
            RussianMessage("\nНеправильный ввод\n");
            return;
        }
        if(_access(destination,02)==-1){
            RussianMessage("\nНет доступа к записи.\n");
            return;
        }
    }
    //Копирование файла
    if(!CopyFile(source,destination)){
        RussianMessage("\nОшибка при работе
                        с файлом\n");
    }
}
```

Пример программы. Игра «Виселица»

Смысл игры состоит в том, что пользователь за некоторое количество попыток должен отгадать слово, в нашем случае английское. В папку с проектом, в котором вы будете компилировать данный код, необходимо положить файл *words.txt*. Этот файл должен содержать несколько английских слов, расположенных в столбик (одно под другим).

```
#include <windows.h>
#include <iostream>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <time.h<
#include <sys\locking.h>
#include <string.h>
#include <ctype.h>

using namespace std;

//Максимальная длина слова
#define MAX_WORD_LENGTH 21

//Кол-во попыток
int Tries = 10;

//Кол-во угаданных слов
int CountWords = 0;
```

```
//Загрузка слова
bool LoadWord(FILE * file, char * word)
{
    int i = 0;
    char s[MAX_WORD_LENGTH] = {0};
    //Кол-во слов в файле
    static int count = -1;

    if(count == -1)
    {
        //Подсчет количества слов
        while(!feof(file))
        {
            fgets(s, MAX_WORD_LENGTH, file);
            count++;
        }
        //Слов нет?
        if(count == 0)
            return false;
        //Возврат файлового указателя в начало файла
        fseek(file, 0, 0);
    }

    //Случайное слово
    int n = rand() % count;
    //Поиск слова
    while(i <= n)
    {
        fgets(s, MAX_WORD_LENGTH, file);
        i++;
    }
    //Определяем длину слова
    int wordlen = strlen(s);
    //Минимальная длина слова 2 буквы
    if(wordlen <= 1)
        return false;
```

```
    //Убираем Enter (в DOS'е 2 байта 13 10)
    if(s[wordlen - 1] == 10)
        s[wordlen - 2] = 0;
    else if(s[wordlen - 1] == 13)
        s[wordlen - 1] = 0;
    //Копируем слово
    strcpy(word, s);
    //Получаем дескриптор файла
    int hFile = _fileno(file);
    //Вычисляем размер файла
    int size = _filelength(hFile);

    //Блокировка файла
    fseek(file, 0, 0);
    _locking(hFile, _LK_NBLCK, size);
    return true;
}

//Игра
void Game(char * word)
{
    //Перевод в большие буквы
    strupr(word);

    int len = strlen(word);
    //Строка-копия
    char * copy = new char[len + 1];
    memset(copy, '*', len);
    copy[len] = 0;

    //Алфавит + пробелы
    char letters[52];

    int i, j = 0;
    for(i = 0; i < 26; i++)
    {
        letters[j++] = i + 'A';
```

```
        letters[j++] = ' ';
    }
    //Замыкающий ноль
    letters[51] = 0;

    //Буква
    char letter;

    char * pos;
    bool replace = false;

    do {
        //Очистка экрана
        system("cls");
        cout << copy << endl << endl;
        cout << letters << endl << endl;
        cout << "Count of tries: " << Tries << endl
             << endl;
        cout << "Input any letter:\t";

        cin >> letter;
        //Звуковой сигнал
        Beep(500, 200);

        //if(letter >= 'A' && letter <= 'Z'
        //|| letter >= 'a' && letter <= 'z')

        //Буква?
        if(!isalpha(letter))
        {
            cout << "It's not a letter" << endl;
            //Задержка на 1 секунду
            Sleep(1000);
            continue;
        }

        //Перевод буквы в большую
        letter = toupper(letter);
```

```
        //Поиск буквы в алфавите
        pos = strchr(letters, letter);

        //Такая буква уже была
        if(pos == 0)
        {
            cout << "This letter have been already
                     pressed" << endl;
            Sleep(1000);
            continue;
        }
        else
        {
            //Убираем букву из алфавита
            pos[0] = ' ';
        }

        //Поиск буквы в слове
        for(i = 0; i < len; i++)
        {
            if(word[i] == letter)
            {
                copy[i] = letter;
                replace = true;
            }
        }

        if(replace == false)
            Tries--;
        else
            replace = false;
        //Условие победы
        if(strcmp(word, copy) == 0)
        {
            system("cls");
            cout << copy << endl << endl;
            cout << letters << endl << endl;
```

```
            cout << "Count of tries: " << Tries <<
                endl << endl;
            cout << "Congratulation !!!" << endl;
            CountWords++;
            break;
        }

    } while(Tries != 0);
    delete [] copy;
}

void main()
{
    //Открываем файл на чтение в двоичном режиме
    FILE * f = fopen("words.txt", "rb");

    //Если файл не открылся
    if(f == 0)
    {
        //Ошибка
        perror("Open");
        return;
    }

    srand(time(0));

    char Word[20];
    //Пытаемся загрузить слово
    if(!LoadWord(f, Word))
    {
        //Если неудачно
        cout << "Error !!!" << endl;
        fclose(f);
        return;
    }

char answer;
//Играем, пока не надоест
do
{
```

```
    Game(Word);
    //Если попыток не осталось, то выход
    if(Tries == 0)
    {
        cout << "Count of words: " << CountWords << endl;
        cout << "Bye-bye" << endl;
        break;
    }
    //Если остались
    cout << "Continue ??? (Y/N)\t";

    cin >> answer;
    //Еще играем?
    if(answer == 'Y' || answer == 'y')
        if(!LoadWord(f, Word))
        {
            cout << "Error !!!" << endl;
            fclose(f);
            return;
        }

    }while(answer == 'Y' || answer == 'y');


    //получаем дескриптор
    int hFile = _fileno(f);

    //Разблокировка файла
    int size = _filelength(hFile);
    fseek(f, 0, 0);
    _locking(hFile, _LK_UNLCK, size);
    fclose(f);
}
```

Домашнее задание

1. Создать телефонный справочник для осуществления следующих операций:
   - Добавление абонентов в базу.
   - Удаление абонентов из базы.
   - Модификация данных абонента.
   - Поиск абонентов по телефонному номеру или фамилии.
   - Распечатка в алфавитном порядке абонентов из заданного диапазона номеров или фамилий; например, для номеров диапазон может быть: 222222–333333, а для фамилий: Иванаускас — Иванов (то есть Иванова в диапазон не входит).
   - Возможность сохранения найденной информации в файл.
   - Сохранение базы в файл.
   - Загрузка базы из файла.
2. Реализовать базу данных ГАИ по штрафным квитанциям с помощью бинарного дерева. Ключом будет служить номер автомашины, значением узла — список правонарушений. Если квитанция добавляется в первый раз, то в дереве появляется новый узел, а в списке данные по правонарушению; если нет, то данные заносятся в существующий список. Необходимо также реализовать следующие операции:

- Полная распечатка базы данных (по номерам машин и списку правонарушений, числящихся за ними).
- Распечатка данных по заданному номеру.
- Распечатка данных по диапазону номеров.