Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Операционные системы»
3 семестр
Задание 5

Группа:	М8О-208Б-18, №12
Студент:	Коростелев Дмитрий Васильевич
Преподаватель:	Миронов Евгений Сергеевич
Оценка:	
Дата:	17.12.2019

Москва, 2019

Содержание

1.Задание

Составить и отладить динамически подключаемую библиотеку на языке C++ с использованием WinApi, продемонстрировать основные принципы создания библиотек и их подключения, составить и отладить программу на языке c++, которая будет использовать и подключать заранее написанные функции из DLL библиотеки.

2.Адрес репозитория на GitHub

https://github.com/Dmitry4K/labOS5

3.Код программы

Source.c:

```
#define LIB_LOCATION L"MYDLL.dll"
#include<Windows.h>
#include<iostream>
#include"Header.h"

int main() {
	HMODULE hDLL = LoadLibrary(LIB_LOCATION);
	if (!hDLL) {
		std::cout << "Error, DLL not find\n";
		return 1;
	}
	cTree* (*cTreeCreate)(int);
	(FARPROC&)cTreeCreate = GetProcAddress(hDLL, "cTreeCreate");
	if (cTreeCreate == NULL) {
		std::cout << "error\n"<<GetLastError();
	}
	void (*cTreePrint)(cTree*);
	(FARPROC&)cTreePrint = GetProcAddress(hDLL, "cTreePrint");
	void (*cTreePrint_)(cTree*,int);
	(FARPROC&)cTreePrint_ = GetProcAddress(hDLL, "cTreePrint_");
	cNode* (*cTreeFindNodeByKey)(cNode, int);
	(FARPROC&)cTreeFindNodeByKey = GetProcAddress(hDLL, "cTreeFindNodeByKey");
	void (*cTreeAddNode)(cTree* , int , int );//проверено
	(FARPROC&)cTreeAddNode = GetProcAddress(hDLL, "cTreeAddNode");
	void (*cTreeDeleteNode)(cTree * , int );
	(FARPROC&)cTreeDeleteNode = GetProcAddress(hDLL, "cTreeDeleteNode");
	void (*cTreeClear)(cNode * );//проверено
	(FARPROC&)cTreeClear = GetProcAddress(hDLL, "cTreeClear");
	void (*cTreeDestroy)(cTree * );//проверено
	(FARPROC&)cTreeDestroy = GetProcAddress(hDLL, "cTreeDestroy");
	void (*cTreePrintTo)(cTree * , HANDLE );
	(FARPROC&)cTreePrintTo = GetProcAddress(hDLL, "cTreePrintTo");
	void (*cTreePrintTo_)(cNode * , int , HANDLE , DWORD * );
	(FARPROC&)cTreePrintTo_ = GetProcAddress(hDLL, "cTreePrintTo_");
	std::string (*cTreePrintToPtr)(cTree*);
	(FARPROC&)cTreePrintToPtr = GetProcAddress(hDLL, "cTreePrintToPtr");
	void (*cTreePrintToPtr_)(cNode * , int , std::string * );
	(FARPROC&)cTreePrintToPtr_ = GetProcAddress(hDLL, "cTreePrintToPtr_");
```

```
	cTree* ctree = nullptr;
	ctree = cTreeCreate(0);
	cTreeAddNode(ctree, 0, 1);
	cTreeAddNode(ctree, 0, 1);
	cTreeAddNode(ctree, 1, 2);
	cTreeAddNode(ctree, 2, 3);
	cTreeDeleteNode(ctree, 3);
	cTreePrint(ctree);
	cTreeDestroy(ctree);
	cTreePrint(ctree);

	return 0;
}
```

Header.h:

```
#pragma once
extern  struct cNode {
	int key;
	cNode* parent;
	cNode* son;
	cNode* brother;
};

extern  struct cTree {
	cNode* root;
};
```

MyDLL.h

```
// Приведенный ниже блок ifdef — это стандартный метод создания макросов, упрощающий
процедуру
// экспорта из библиотек DLL. Все файлы данной DLL скомпилированы с использованием
символа MYDLL_EXPORTS
// Символ, определенный в командной строке. Этот символ не должен быть определен в каком-
либо проекте,
// использующем данную DLL. Благодаря этому любой другой проект, исходные файлы которого
включают данный файл, видит
// функции MYDLL_API как импортированные из DLL, тогда как данная DLL видит символы,
// определяемые данным макросом, как экспортированные.
#ifdef MYDLL_EXPORTS
#define MYDLL_API __declspec(dllexport)
#else
#define MYDLL_API __declspec(dllimport)
#endif


#include<string>
// Этот класс экспортирован из библиотеки DLL
/*
class MYDLL_API CMyDLL {
public:
	CMyDLL(void);
	// TODO: добавьте сюда свои методы.
};

extern MYDLL_API int nMyDLL;

MYDLL_API int fnMyDLL(void);
*/
```

```
extern MYDLL_API struct cNode {
	int key;
	cNode* parent;
	cNode* son;
	cNode* brother;
};

extern MYDLL_API struct cTree {
	cNode* root;
};


MYDLL_API cTree* cTreeCreate(int key);//проверено
MYDLL_API cNode* cTreeFindNodeByKey(cNode* tree, int key);//проверено
MYDLL_API void cTreeAddNode(cTree* tree, int to, int key);//проверено
MYDLL_API void cTreeDeleteNode(cTree* tree, int key);
MYDLL_API void cTreeClear(cNode* node);//проверено
MYDLL_API void cTreeDestroy(cTree* tree);//проверено
MYDLL_API void cTreePrint(cTree* tree);
MYDLL_API void cTreePrint_(cNode* node, int count);
MYDLL_API void cTreePrintTo(cTree* tree, HANDLE outH);
MYDLL_API void cTreePrintTo_(cNode* node, int count, HANDLE outH, DWORD* writebytes);
MYDLL_API std::string cTreePrintToPtr(cTree* tree);
MYDLL_API void cTreePrintToPtr_(cNode* node, int count, std::string* res);
```

MyDLL.cpp

```
// MyDLL.cpp : Определяет экспортируемые функции для DLL.
//

#include "pch.h"
#include "framework.h"
#include "MyDLL.h"

#include<malloc.h>
#include<iostream>
#include<string>
/*
// Пример экспортированной переменной
MYDLL_API int nMyDLL=0;

// Пример экспортированной функции.
MYDLL_API int fnMyDLL(void)
{
    return 0;
}

// Конструктор для экспортированного класса.
CMyDLL::CMyDLL()
{
    return;
}*/
MYDLL_API cTree* cTreeCreate(int key) {
	cTree* res = (cTree*)malloc(sizeof(cTree));
	res->root = (cNode*)malloc(sizeof(cNode));
	res->root->key = key;
	res->root->parent = nullptr;
	res->root->brother = nullptr;
	res->root->son = nullptr;
	return res;
}

MYDLL_API cNode* cTreeFindNodeByKey(cNode* node, int key) {
```

```
	cNode* res = nullptr;
	if (node->key == key)
		return node;
	if (node->son)
		res = cTreeFindNodeByKey(node->son, key);
	if (node->brother && !res)
		res = cTreeFindNodeByKey(node->brother, key);
	return res;
}

MYDLL_API void cTreeAddNode(cTree* tree, int to, int key) {
	cNode* fnode = nullptr;
	fnode = cTreeFindNodeByKey(tree->root, to);
	if (!fnode) {
		return;
	}
	if (!fnode->son) {
		fnode->son = (cNode*)malloc(sizeof(cNode));
		fnode->son->key = key;
		fnode->son->parent = fnode;
		fnode->son->brother = nullptr;
		fnode->son->son = nullptr;
		return;
	}
	else {
		cNode* bnode = nullptr;
		bnode = fnode->son;
		while (bnode->brother)
			bnode = bnode->brother;
		bnode->brother = (cNode*)malloc(sizeof(cNode));
		bnode->brother->key = key;
		bnode->brother->parent = fnode;
		bnode->brother->brother = nullptr;
		bnode->brother->son = nullptr;
		return;
	}
	return;
}

MYDLL_API void cTreeDeleteNode(cTree* tree, int key) {
	cNode* fnode = nullptr;
	fnode = cTreeFindNodeByKey(tree->root, key);
	if (!fnode)
		return;

	if (fnode->son)
		cTreeClear(fnode->son);

	cNode* inode = fnode->parent->son;
	if (inode == fnode) {
		if (fnode->brother)
			fnode->parent->son = fnode->brother;
		else
			fnode->parent->son = nullptr;
		free(fnode);
	}
	else {
		while (inode->brother != fnode) {
			inode = inode->brother;
		}
		if (fnode->brother) {
			inode->brother = fnode->brother;
			free(fnode);
		}
```

```
			else {
				inode->brother = nullptr;
				free(fnode);
			}
		}
}
MYDLL_API void cTreeClear(cNode* node) {
		if (node->son)
			cTreeClear(node->son);
		if (node->brother)
			cTreeClear(node->brother);
		node->brother = nullptr;
		node->son = nullptr;
		if (node->parent)
			if (node->parent->son == node)
				node->parent->son = nullptr;
		free(node);
}

MYDLL_API void cTreeDestroy(cTree* tree) {
		cTreeClear(tree->root);
		tree->root = nullptr;
}

MYDLL_API void cTreePrint(cTree* tree) {
		if (tree->root)
			cTreePrint_(tree->root, 0);
}

MYDLL_API void cTreePrint_(cNode* node, int count) {
		for (int i = 0; i < count; i++)
			printf("\t");
		printf("%d\n", node->key);
		if (node->son)
			cTreePrint_(node->son, count + 1);
		if (node->brother)
			cTreePrint_(node->brother, count);
}


MYDLL_API void cTreePrintTo(cTree* tree, HANDLE outH) {
		DWORD writebytes = 0;
		char c = '\0';
		if (tree->root)
			cTreePrintTo_(tree->root, 0, outH, &writebytes);
		WriteFile(outH, &c, sizeof(char), &writebytes, 0);
}

MYDLL_API void cTreePrintTo_(cNode* node, int count, HANDLE outH, DWORD* writebytes) {
		char c = '\t';
		for (int i = 0; i < count; i++)
			WriteFile(outH, &c, sizeof(char), writebytes, 0);
		c = node->key + '0';
		WriteFile(outH, &c, sizeof(char), writebytes, 0);
		c = '\n';
		WriteFile(outH, &c, sizeof(char), writebytes, 0);
		if (node->son)
			cTreePrintTo_(node->son, count + 1, outH, writebytes);
		if (node->brother)
			cTreePrintTo_(node->brother, count, outH, writebytes);
}
MYDLL_API void cTreePrintToPtr_(cNode* node, int count, std::string* res) {
		char c = '\t';
		for (int i = 0; i < count; i++)
```

```
			(*res) += c;
		//WriteFile(outH, &c, sizeof(char), writebytes, 0);
		c = node->key + '0';
		(*res) += c;
		//WriteFile(outH, &c, sizeof(char), writebytes, 0);
		c = '\n';
		(*res) += c;
		//WriteFile(outH, &c, sizeof(char), writebytes, 0);
		if (node->son)
			cTreePrintToPtr_(node->son, count + 1, res);
		if (node->brother)
			cTreePrintToPtr_(node->brother, count, res);
}
MYDLL_API std::string cTreePrintToPtr(cTree* tree) {
		//char c = '\0';
		std::string res;
		if (tree->root)
			cTreePrintToPtr_(tree->root, 0, &res);
		res += '\0';
		return res;
		//WriteFile(outH, &c, sizeof(char), &writebytes, 0);
}
```

Source.def

```
LIBRARY
EXPORTS
cTreeCreate=cTreeCreate
cTreeFindNodeByKey=cTreeFindNodeByKey
cTreeAddNode=cTreeAddNode
cTreeDeleteNode=cTreeDeleteNode
cTreeClear=cTreeClear
cTreeDestroy=cTreeDestroy
cTreePrint=cTreePrint
cTreePrint_=cTreePrint_
cTreePrintTo=cTreePrintTo
cTreePrintTo_=cTreePrintTo_
cTreePrintToPtr=cTreePrintToPtr
cTreePrintToPtr_=cTreePrintToPtr_
```

4.Результаты выполнения тестов

```
 0
          1
                    2
          1
```

5.Объяснение результатов работы программы

Целью лабораторной работы являлась реализации DLL билиотеки, что собственно и было мной продемонстрировано. За основу билиотеки были взяты процедуры для обработки дерева общего вида, помещеные в специальные файлы, добавлены соглашения о вызовах данных функций.

Использование библиотеки подразумевает в себе явное ее подключение по следующему алгоритму – создается указатель на функцию и далее с

помощью WinApi функции указатель связывается с нужной функцией. О том, что все работает корректно говорит правильная и логичная работа, представленная в файле source.c

6.Вывод

DLL – особый вид библиотек, который динамически подключается непосредственно во время работы программы. DLL крайне удобны в использовании в тех ситуациях, когда программисту нужно использовать свой набор инструментов, для этого достаточно иметь скомпилированный файл в своем рабочем пространстве. Кроме того при помощи DLL программист может изменить отдельно библиотеку, тем самым изменить способ выполнения определенной программы не прибегая к долгой компиляции и сборке всего проекта.