МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»  
  
Факультет информатики  
Кафедра информационных систем и технологий  
  
Дисциплина  
**«Операционные системы»  
  
  
  
ОТЧЕТ**по лабораторному практикуму

Вариант № 3

Студент: Гижевская В.Д.  
Группа: 6213-020302D  
  
Преподаватель: Востокин С.В.

Самара 2020

Лабораторные работы № 1,2

Цель:Изучение синтаксиса языка Си. В лабораторной работе №1 реализуется алгоритм решения выбранной задачи. В лабораторной работе №2 решение оформляется в виде функций, выполняющих ввод данных пользователя с клавиатуры в консольном режиме, обработку данных согласно заданию и вывод результата.

Задание: В массиве A(N) каждый элемент равен 0, 1 или 2. Переставить элементы массива так, чтобы сначала располагались все нули, затем все двойки и, наконец, все единицы.

Листинг программы:

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <conio.h>

#include <stdio.h>

void SortArr(int\* arr, int\* res, int N)

{

int order[3] = { 0, 2, 1 };

int j = 0;

for (int k = 0; k < 3; k++) {

for (int i = 0; i < N; i++) {

if (arr[i] == order[k]) {

res[j] = arr[i];

j++;

}

}

}

puts("\nОтсортированный массив:");

for (int i = 0; i < N; i++)

printf("%d ", res[i]);

}

int main(void)

{

setlocale(LC\_ALL, "Rus");

printf("Вариант 3 | Выполнила: Гижевская Валерия | Группа: 6213 \n");

int N = 10;

int arr[10];

int res[10];

puts("Неотсортированный массив:");

for (int i = 0; i < N; i++) {

arr[i] = rand() % 3;

printf("%d ", arr[i]);

}

SortArr(arr, res, N);

}

Лабораторная работа № 3

Цель: Изучение методов работы с динамической памятью средствами программного интерфейса Win32. В первой части работы пишется программа с функциями создания динамического массива требуемого размера, обработки его согласно заданию, вывода и очистки с использованием функций библиотеки времени исполнения C/C++. Во второй части работы вызовы библиотеки времени исполнения заменяются вызовами программного интерфейса Win32. Задание: Необходимо каждый элемент строки разделить на наибольший элемент строки в динамическом массиве размерностью NxM.

Листинг программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <conio.h>

#include <windows.h>

#include <stdio.h>

void PrintArray(float\* a, int n, int m)

{

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

printf("%10.3f", \*(a + i \* m + j));

}

printf("\n");

}

}

void InputArray(float\* a, int n, int m)

{

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

printf("a[%d][%d] = ", i, j);

scanf("%f", (a + i \* m + j));

}

}

}

void DivByMaxEl(float\* a, int n, int m)

{

float\* max = NULL;

HANDLE heap = GetProcessHeap();

max = (float\*)HeapAlloc(heap, NULL, n \* sizeof(float));

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

if (a[j] > max[i])

{

max[i] = a[j];

}

}

for (j = 0; j < m; j++) // цикл по столбцам

{

printf("%10.3f", \*(a + i \* m + j) / max[i]);

}

printf("\n");

}

}

int main()

{

setlocale(LC\_ALL, "");

int i, j, n, m;

printf(" Вариант 3 | Гижевская Валерия | 6213 \n");

printf("\nВведите количество строк: ");

scanf("%d", &n);

printf("Введите количество столбцов: ");

scanf("%d", &m);

HANDLE heap = GetProcessHeap();

if (heap != NULL) {

float\* a = NULL;

a = (float\*)HeapAlloc(heap, NULL, n \* m \* sizeof(float));

if (a != NULL)

{

InputArray(a, n, m);

PrintArray(a, n, m);

printf("\nМассив, в котором каждый элемент строки разделён на наибольший элемент строки: \n");

DivByMaxEl(a, n, m);

HeapFree(heap, NULL, a);

}

}

getchar();

return 0;

}

Лабораторная работа № 4

Цель: Изучение методов работы с типами данных, определяемых пользователем на языке Си. Требуется реализовать в виде отдельной единицы компиляции (модуля) набор функций и объявлений данных, необходимых для работы с указанным в задании типом данных. В отдельном модуле пишется код для тестирования функций модуля.

Задание: Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Листинг программы:

BinaryTree.h:

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct bstree {

int key; // Ключ

char\* value; // Данные

struct bstree\* left;

struct bstree\* right;

};

//создание элемента

struct bstree\* bstree\_create(int key, const char\* value);

//добавление элемента

void bstree\_add(struct bstree\* tree,

int key, const char\* value);

//поиск элемента в дереве

struct bstree\* bstree\_lookup(struct bstree\* tree,

int key);

//поиск минимального элемента дерева

struct bstree\* bstree\_min(struct bstree\* tree);

//поиск максимального элемента дерева

struct bstree\* bstree\_max(struct bstree\* tree);

void writeInFile();

void readFile();

BinaryTree.cpp:

#include "BinaryTree.h"

#include <windows.h>

const TCHAR szCounterFileName[] = L"file.txt";

//создание элемента

struct bstree\* bstree\_create(int key, const char\* value)

{

struct bstree\* node;

node = (bstree\*)malloc(sizeof(\*node));

if (node != NULL) {

node->key = key;

node->value = \_strdup(value);

//node->value = value;

node->left = NULL;

node->right = NULL;

}

return node;

}

//добавление элемента

void bstree\_add(struct bstree\* tree,

int key, const char\* value)

{

struct bstree\* parent, \* node;

parent = NULL;

if (tree == NULL)

return;

/\* Отыскиваем листовой узел \*/

while (tree != NULL) {

parent = tree;

if (key < tree->key) {

tree = tree->left;

}

else if (key > tree->key) {

tree = tree->right;

}

else {

return;

}

}

/\* Создаем элемент и связываем с узлом \*/

node = bstree\_create(key, value);

if (key < parent->key)

parent->left = node;

else

parent->right = node;

}

//поиск элемента в дереве

struct bstree\* bstree\_lookup(struct bstree\* tree,

int key)

{

while (tree != NULL) {

if (key == tree->key) {

return tree;

}

else if (key < tree->key) {

tree = tree->left;

}

else {

tree = tree->right;

}

}

return tree;

}

//поиск минимального элемента дерева

struct bstree\* bstree\_min(struct bstree\* tree)

{

if (tree == NULL)

return NULL;

while (tree->left != NULL)

tree = tree->left;

return tree;

}

//поиск максимального элемента дерева

struct bstree\* bstree\_max(struct bstree\* tree)

{

if (tree == NULL)

return NULL;

while (tree->right != NULL)

tree = tree->right;

return tree;

}

Lab4.cpp:

#include "BinaryTree.h"

#include <iostream>

#include <tchar.h>

#include <windows.h>

int main()

{

setlocale(LC\_ALL, "");

struct bstree\* tree, \* node;

printf("Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево\n");

tree = bstree\_create(28, "Ситникова");

bstree\_add(tree, 23, "Гижевская");

bstree\_add(tree, 60, "Борисова");

node = bstree\_lookup(tree, 60);

printf("Value = %s\n", node->value);

node = bstree\_lookup(tree, 28);

printf("Value = %s\n", node->value);

return 0;

}

Лабораторная работа № 5

Цель: Изучение функций ввода/вывода в программном интерфейсе Win32. Интерфейс модуля для работы с заданной структурой данных из задания №4 расширяется функциями для сохранения структуры данных на диске целиком и восстановления структуры данных из сохранённого файла.

Задание: Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Листинг программы:

BinaryTree.h:

#pragma once

#include <Windows.h>

#include <cstdio>

struct nodeTree {

int key; // Ключ

char\* value; // Данные

struct nodeTree\* left;

struct nodeTree\* right;

};

class BinaryTree {

nodeTree\* root;

int size;

void leftPrint(nodeTree\* tree);

void deleteNode(nodeTree\* tree);

public:

BinaryTree();

//добавление элемента

void add(int key, const char\* value);

//левый обход дерева

void leftPrint();

//дружественные функцию

friend void writeInFile(BinaryTree\* tree, HANDLE file);

~BinaryTree();

};

BinaryTree\* readInFile(HANDLE file);

//создание элемента

nodeTree\* createNode(int key, const char\* value);

BinaryTree.cpp:

#include "BinaryTree.h"

void writeFile(nodeTree\* tree, HANDLE file) {

if (tree == NULL) return;

DWORD dw = 0;

int l = strlen(tree->value);

WriteFile(file, &(tree->key), sizeof(int), &dw, NULL);

WriteFile(file, &l, sizeof(int), &dw, NULL);

WriteFile(file, tree->value, strlen(tree->value), &dw, NULL);

writeFile(tree->left, file);

writeFile(tree->right, file);

}

void writeInFile(BinaryTree\* tree, HANDLE file)

{

writeFile(tree->root, file);

}

BinaryTree\* readInFile(HANDLE file)

{

BinaryTree\* tree=new BinaryTree();

DWORD end = SetFilePointer(file, 0, 0, FILE\_END);

SetFilePointer(file, 0, 0, FILE\_BEGIN);

DWORD dw = 0;

int key;

int length = 0;

char\* buf = new char[256];

if (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

while (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

delete[] buf;

return tree;

}

nodeTree\* createNode(int key, const char\* value)

{

struct nodeTree\* node = new nodeTree;

node->key = key;

node->value = \_strdup(value);

node->left = NULL;

node->right = NULL;

return node;

}

void BinaryTree::leftPrint(nodeTree\* tree)

{

if (tree == NULL) return;

leftPrint(tree->left);

leftPrint(tree->right);

printf("key: %d; value: %s\n", tree->key, tree->value);

}

void BinaryTree::deleteNode(nodeTree\* tree)

{

if (tree) {

deleteNode(tree->left);

deleteNode(tree->right);

free(tree);

}

tree = NULL;

}

BinaryTree::BinaryTree()

{

root = NULL;

size = 0;

}

void BinaryTree::add(int key, const char\* value)

{

struct nodeTree\* parent;

parent = NULL;

if (root == NULL) {

root = createNode(key, value);

size++;

}

else {

nodeTree\* tree = root;

while (tree != NULL) {

parent = tree;

if (key < tree->key) {

tree = tree->left;

}

else if (key > tree->key) {

tree = tree->right;

}

else {

return;

}

}

/\* Создаем элемент и связываем с узлом \*/

nodeTree\* node = createNode(key, value);

if (key < parent->key)

parent->left = node;

else

parent->right = node;

size++;

}

}

void BinaryTree::leftPrint()

{

leftPrint(root);

}

BinaryTree::~BinaryTree()

{

deleteNode(root);

}

Main.cpp:

#include <iostream>

#include "BinaryTree.h"

void main() {

setlocale(LC\_ALL, "russian");

printf("Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево\n");

BinaryTree\* a = new BinaryTree();

a->add (28, "Ситникова");

a->add (23, "Гижевская");

a->add (60, "Борисова");

a->add (11, "Булатник");

a->add (22, "Петрова");

a->add (33, "Корень");

a->add (44, "Азаматов");

a->add (55, "Сухов");

a->add (66, "Кротков");

a->add (77, "Абрамов");

a->add (88, "Мбингу");

a->add (99, "Черкасов");

printf("Исходное дерево (левый обход):\n");

a->leftPrint();

HANDLE fileTo = CreateFile(L"file.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

writeInFile(a, fileTo);

CloseHandle(fileTo);

HANDLE fileFrom = CreateFile(L"file.txt", GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

BinaryTree\* b = readInFile(fileFrom);

CloseHandle(fileFrom);

printf("\n\nДерево, считанное из файла (левый обход):\n");

b->leftPrint();

system("pause");

}

Лабораторная работа № 6

Цель: Изучение методов работы с динамически подключаемыми библиотеками в программном интерфейсе Win32. Из модуля для работы с заданным типом данных, реализованным в задании №5, строится динамически подключаемая библиотека. Тестирующий код выполняет подключение библиотеки с использованием явного (парой вызовов LoadLibrary/GetProcAddress) и неявного (конфигурированием проекта) связывания.

Задание: Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Листинг программы:

BinaryTreeDll:

BinaryTreeDll.h:

#pragma once

#ifdef BINARYTREEDLL\_EXPORTS

#define BINARYTREEDLL\_API \_\_declspec(dllimport)

#else

#define BINARYTREEDLL\_API \_\_declspec(dllexport)

#endif //

#include <Windows.h>

#include <cstdlib>

#include <cstdio>

struct BINARYTREEDLL\_API nodeTree {

int key; // Ключ

char\* value; // Данные

struct nodeTree\* left;

struct nodeTree\* right;

};

class BINARYTREEDLL\_API BinaryTree {

nodeTree\* root;

int size;

void leftPrint(nodeTree\* tree);

void deleteNode(nodeTree\* tree);

public:

BinaryTree();

const nodeTree\* getRoot();

//добавление элемента

void add(int key, const char\* value);

//левый обход дерева

void leftPrint();

~BinaryTree();

friend BINARYTREEDLL\_API void writeInFile(BinaryTree\* tree, HANDLE file);

};

BINARYTREEDLL\_API BinaryTree\* readInFile(HANDLE file);

//создание элемента

BINARYTREEDLL\_API nodeTree\* createNode(int key, const char\* value);

BinaryTreeDll.cpp:

#include "pch.h"

#include "BinaryTreeDll.h"

void writeFile(const nodeTree\* tree, HANDLE file) {

if (tree == NULL) return;

DWORD dw = 0;

int l = strlen(tree->value);

WriteFile(file, &(tree->key), sizeof(int), &dw, NULL);

WriteFile(file, &l, sizeof(int), &dw, NULL);

WriteFile(file, tree->value, strlen(tree->value), &dw, NULL);

writeFile(tree->left, file);

writeFile(tree->right, file);

}

BinaryTree\* readInFile(HANDLE file)

{

BinaryTree\* tree = new BinaryTree();

DWORD end = SetFilePointer(file, 0, 0, FILE\_END);

SetFilePointer(file, 0, 0, FILE\_BEGIN);

DWORD dw = 0;

int key;

int length = 0;

char\* buf = new char[256];

if (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

while (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

delete[] buf;

return tree;

}

nodeTree\* createNode(int key, const char\* value)

{

struct nodeTree\* node = new nodeTree;

node->key = key;

node->value = \_strdup(value);

node->left = NULL;

node->right = NULL;

return node;

}

void BinaryTree::leftPrint(nodeTree\* tree)

{

if (tree == NULL) return;

leftPrint(tree->left);

leftPrint(tree->right);

printf("key: %d; value: %s\n", tree->key, tree->value);

}

void BinaryTree::deleteNode(nodeTree\* tree)

{

if (tree) {

deleteNode(tree->left);

deleteNode(tree->right);

free(tree);

}

tree = NULL;

}

BinaryTree::BinaryTree()

{

root = NULL;

size = 0;

}

const nodeTree\* BinaryTree::getRoot()

{

return root;

}

void BinaryTree::add(int key, const char\* value)

{

struct nodeTree\* parent;

parent = NULL;

if (root == NULL) {

root = createNode(key, value);

size++;

}

else {

nodeTree\* tree = root;

while (tree != NULL) {

parent = tree;

if (key < tree->key) {

tree = tree->left;

}

else if (key > tree->key) {

tree = tree->right;

}

else {

return;

}

}

/\* Создаем элемент и связываем с узлом \*/

nodeTree\* node = createNode(key, value);

if (key < parent->key)

parent->left = node;

else

parent->right = node;

size++;

}

}

void BinaryTree::leftPrint()

{

leftPrint(root);

}

BinaryTree::~BinaryTree()

{

deleteNode(root);

}

void writeInFile(BinaryTree\* tree, HANDLE file)

{

writeFile(tree->root, file);

}

dllmain.cpp:

#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

Lab6:

Main.cpp

#include <iostream>

#include "BinaryTreeDll.h"

void main() {

setlocale(LC\_ALL, "russian");

printf("Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево\n");

BinaryTree\* a = new BinaryTree();

a->add (28, "Ситникова");

a->add (23, "Гижевская");

a->add (60, "Борисова");

a->add (11, "Булатник");

a->add (22, "Петрова");

a->add (33, "Корень");

a->add (44, "Азаматов");

a->add (55, "Сухов");

a->add (66, "Кротков");

a->add (77, "Абрамов");

a->add (88, "Мбингу");

a->add (99, "Черкасов");

printf("Исходное дерево (левый обход):\n");

a->leftPrint();

HANDLE fileTo = CreateFile(L"file.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

writeInFile(a, fileTo);

CloseHandle(fileTo);

HANDLE fileFrom = CreateFile(L"file.txt", GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

BinaryTree\* b = readInFile(fileFrom);

CloseHandle(fileFrom);

printf("\n\nДерево, считанное из файла (левый обход):\n");

b->leftPrint();

system("pause");

}

Yavn:

Main.cpp:

#include <iostream>

#include <Windows.h>

#include "..\\BinaryTreeDll\\BinaryTreeDll.h"

void main() {

setlocale(LC\_ALL, "RUS");

HINSTANCE dll = LoadLibrary(L"..\\Debug\\BinaryTreeDll.dll");

if (dll == NULL) {

printf("Ошибка при загрузке dll\n");

system("pause");

return;

}

printf("Требуется реализовать структуру данных Ђассоциативный массивї, использу¤ бинарное дерево\n");

void(BinaryTree:: \* constructor)();

(FARPROC&)constructor = GetProcAddress(dll, "??0BinaryTree@@QAE@XZ");

void (BinaryTree:: \* add)(int, const char\*);

(FARPROC&)add = GetProcAddress(dll, "?add@BinaryTree@@QAEXHPBD@Z");

void (BinaryTree:: \* leftPrint)();

(FARPROC&)leftPrint = GetProcAddress(dll, "?leftPrint@BinaryTree@@QAEXXZ");

char \_c[sizeof(BinaryTree)];

BinaryTree& a = \*(BinaryTree\*)\_c;

(a.\*constructor)();

(a.\*add)(28, "Ситникова");

(a.\*add)(23, "Гижевская");

(a.\*add)(60, "Борисова");

(a.\*add)(11, "Булатник");

(a.\*add)(22, "Петрова");

(a.\*add)(33, "Корень");

(a.\*add)(44, "Азаматов");

(a.\*add)(55, "Сухов");

(a.\*add)(66, "Кротков");

(a.\*add)(77, "Абрамов");

(a.\*add)(88, "Мбингу");

(a.\*add)(99, "Черкасов");

printf("Исходное дерево (левый обход):\n");

(a.\*leftPrint)();

system("pause");

Лабораторная работа № 7

Цель:Изучение методов написания многопоточных приложений и синхронизации потоков в программном интерфейсе Win32. В библиотеку функций для работы с заданной структурой данных, реализованную в задании №5 или №6, добавляется следующая функциональность. Вызов функции для добавления элемента в структуру выполняется в одном потоке, обработка вызова с действительным помещением элементов в неё – в другом потоке. Передача аргументов вызова осуществляется через буфер в памяти, доступ к которому синхронизируется. При каждом добавлении элемента в структуру данных происходит ее сохранение на диск целиком, как в задании №5. Тестирующая программа демонстрирует корректность записи элементов путём чтения файла на диске и печати его содержимого по окончании добавления.

Задание: Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Листинг программы:

BinaryTreeDll:

BinaryTree.h:

#pragma once

#ifdef BINARYTREEDLL\_EXPORTS

#define BINARYTREEDLL\_API \_\_declspec(dllimport)

#else

#define BINARYTREEDLL\_API \_\_declspec(dllexport)

#endif //

#include <Windows.h>

#include <cstdlib>

#include <cstdio>

struct BINARYTREEDLL\_API nodeTree {

int key; // Ключ

char\* value; // Данные

struct nodeTree\* left;

struct nodeTree\* right;

};

class BINARYTREEDLL\_API BinaryTree {

nodeTree\* root;

int size;

int buf\_key;

char\* buf\_value;

void leftPrint(nodeTree\* tree);

void deleteNode(nodeTree\* tree);

public:

HANDLE hThread;

HANDLE bufEmpty;

HANDLE bufFull;

HANDLE stopThread;

BinaryTree();

BinaryTree(bool f);

const nodeTree\* getRoot();

//добавление элемента

void add(int key, const char\* value);

//левый обход дерева

void leftPrint();

~BinaryTree();

friend BINARYTREEDLL\_API void writeInFile(BinaryTree\* tree, HANDLE file);

friend DWORD WINAPI ThreadFunc(PVOID p);

void callAdd(int key, const char\* value);

};

BINARYTREEDLL\_API BinaryTree\* readInFile(HANDLE file);

//создание элемента

BINARYTREEDLL\_API nodeTree\* createNode(int key, const char\* value);

BinaryTree.cpp:

#include "pch.h"

#include "BinaryTreeDll.h"

void writeFile(const nodeTree\* tree, HANDLE file) {

if (tree == NULL) return;

DWORD dw = 0;

int l = strlen(tree->value);

WriteFile(file, &(tree->key), sizeof(int), &dw, NULL);

WriteFile(file, &l, sizeof(int), &dw, NULL);

WriteFile(file, tree->value, strlen(tree->value), &dw, NULL);

writeFile(tree->left, file);

writeFile(tree->right, file);

}

DWORD \_\_stdcall ThreadFunc(PVOID p)

{

BinaryTree\* tree = ((BinaryTree\*)p);

while (WaitForSingleObject(tree->stopThread, 0) != WAIT\_OBJECT\_0) {

WaitForSingleObject(tree->bufFull, INFINITE);

tree->add(tree->buf\_key, \_strdup(tree->buf\_value));

HANDLE fileTo = CreateFile(L"file.txt", GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

writeInFile(tree, fileTo);

CloseHandle(fileTo);

SetEvent(tree->bufEmpty);

}

return 0;

}

BinaryTree\* readInFile(HANDLE file)

{

BinaryTree\* tree = new BinaryTree();

DWORD end = SetFilePointer(file, 0, 0, FILE\_END);

SetFilePointer(file, 0, 0, FILE\_BEGIN);

DWORD dw = 0;

int key;

int length = 0;

char\* buf = new char[256];

if (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

while (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

delete[] buf;

return tree;

}

nodeTree\* createNode(int key, const char\* value)

{

struct nodeTree\* node = new nodeTree;

node->key = key;

node->value = \_strdup(value);

node->left = NULL;

node->right = NULL;

return node;

}

void BinaryTree::leftPrint(nodeTree\* tree)

{

if (tree == NULL) return;

leftPrint(tree->left);

leftPrint(tree->right);

printf("key: %d; value: %s\n", tree->key, tree->value);

}

void BinaryTree::deleteNode(nodeTree\* tree)

{

if (tree) {

deleteNode(tree->left);

deleteNode(tree->right);

free(tree);

}

tree = NULL;

}

BinaryTree::BinaryTree()

{

root = NULL;

size = 0;

}

BinaryTree::BinaryTree(bool f)

{

root = NULL;

size = 0;

buf\_key = 0;

buf\_value = \_strdup("");

DWORD id;

bufEmpty = CreateEvent(NULL, FALSE, TRUE, L"bufEmpty");

bufFull = CreateEvent(NULL, FALSE, FALSE, L"bufFull");

stopThread = CreateEvent(NULL, TRUE, FALSE, L"stopThread");

HANDLE fileTo = CreateFile(L"file.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

CloseHandle(fileTo);

hThread = CreateThread(NULL, 0, ThreadFunc, this, 0, &id);

}

const nodeTree\* BinaryTree::getRoot()

{

return root;

}

void BinaryTree::add(int key, const char\* value)

{

struct nodeTree\* parent;

parent = NULL;

if (root == NULL) {

root = createNode(key, value);

size++;

}

else {

nodeTree\* tree = root;

while (tree != NULL) {

parent = tree;

if (key < tree->key) {

tree = tree->left;

}

else if (key > tree->key) {

tree = tree->right;

}

else {

return;

}

}

/\* Создаем элемент и связываем с узлом \*/

nodeTree\* node = createNode(key, value);

if (key < parent->key)

parent->left = node;

else

parent->right = node;

size++;

}

}

void BinaryTree::leftPrint()

{

leftPrint(root);

}

BinaryTree::~BinaryTree()

{

deleteNode(root);

}

void BinaryTree::callAdd(int key, const char\* value)

{

WaitForSingleObject(bufEmpty, INFINITE);

buf\_key = key;

buf\_value = \_strdup(value);

SetEvent(bufFull);

}

void writeInFile(BinaryTree\* tree, HANDLE file)

{

writeFile(tree->root, file);

}

dllmain.cpp:

#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

Main.cpp:

#include <iostream>

#include "BinaryTreeDll.h"

void main() {

setlocale(LC\_ALL, "russian");

printf("Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево\n");

BinaryTree\* a = new BinaryTree(true);

a->callAdd(28, "Ситникова");

a->callAdd(23, "Гижевская");

a->callAdd(60, "Борисова");

a->callAdd(11, "Булатник");

a->callAdd(22, "Петрова");

a->callAdd(33, "Корень");

a->callAdd(44, "Азаматов");

a->callAdd(55, "Сухов");

a->callAdd(66, "Кротков");

a->callAdd(77, "Абрамов");

a->callAdd(88, "Мбингу");

a->callAdd(99, "Черкасов");

WaitForSingleObject(a->bufEmpty, INFINITE);

SetEvent(a->stopThread);

printf("Дерево (левый обход):\n");

HANDLE fileFrom = CreateFile(L"file.txt", GENERIC\_READ, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

a = readInFile(fileFrom);

CloseHandle(fileFrom);

a->leftPrint();

system("pause");

}

Лабораторная работа № 8

Цель: Изучение методов работы с процессами в программном интерфейсе Win32. Задание выполняется по схеме задания №7 за исключением того, что поток, осуществляющий фактическое добавление элементов в структуру данных, реализуется в дочернем процессе.

Задание: Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Листинг программы:

BinaryTreeDll:

BinaryTree.h:

#pragma once

#ifdef BINARYTREEDLL\_EXPORTS

#define BINARYTREEDLL\_API \_\_declspec(dllimport)

#else

#define BINARYTREEDLL\_API \_\_declspec(dllexport)

#endif //

#include <Windows.h>

#include <cstdlib>

#include <cstdio>

struct BINARYTREEDLL\_API nodeTree {

int key; // Ключ

char\* value; // Данные

struct nodeTree\* left;

struct nodeTree\* right;

};

class BINARYTREEDLL\_API BinaryTree {

nodeTree\* root;

int size;

int buf\_key;

char\* buf\_value;

void leftPrint(nodeTree\* tree);

void deleteNode(nodeTree\* tree);

public:

HANDLE hThread;

HANDLE bufEmpty;

HANDLE bufFull;

HANDLE stopThread;

BinaryTree();

BinaryTree(bool f);

const nodeTree\* getRoot();

//добавление элемента

void add(int key, const char\* value);

//левый обход дерева

void leftPrint();

~BinaryTree();

friend BINARYTREEDLL\_API void writeInFile(BinaryTree\* tree, HANDLE file);

friend DWORD WINAPI ThreadFunc(PVOID p);

void callAdd(int key, const char\* value);

};

BINARYTREEDLL\_API BinaryTree\* readInFile(HANDLE file);

//создание элемента

BINARYTREEDLL\_API nodeTree\* createNode(int key, const char\* value);

BinaryTree.cpp:

#include "pch.h"

#include "BinaryTreeDll.h"

void writeFile(const nodeTree\* tree, HANDLE file) {

if (tree == NULL) return;

DWORD dw = 0;

int l = strlen(tree->value);

WriteFile(file, &(tree->key), sizeof(int), &dw, NULL);

WriteFile(file, &l, sizeof(int), &dw, NULL);

WriteFile(file, tree->value, strlen(tree->value), &dw, NULL);

writeFile(tree->left, file);

writeFile(tree->right, file);

}

DWORD \_\_stdcall ThreadFunc(PVOID p)

{

BinaryTree\* tree = ((BinaryTree\*)p);

while (WaitForSingleObject(tree->stopThread, 0) != WAIT\_OBJECT\_0) {

WaitForSingleObject(tree->bufFull, INFINITE);

tree->add(tree->buf\_key, \_strdup(tree->buf\_value));

HANDLE fileTo = CreateFile(L"..\\Debug\\file.txt", GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

writeInFile(tree, fileTo);

CloseHandle(fileTo);

SetEvent(tree->bufEmpty);

}

return 0;

}

BinaryTree\* readInFile(HANDLE file)

{

BinaryTree\* tree = new BinaryTree();

DWORD end = SetFilePointer(file, 0, 0, FILE\_END);

SetFilePointer(file, 0, 0, FILE\_BEGIN);

DWORD dw = 0;

int key;

int length = 0;

char\* buf = new char[256];

if (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

while (SetFilePointer(file, 0, 0, FILE\_CURRENT) < end)

{

ReadFile(file, &key, sizeof(int), &dw, NULL);

ReadFile(file, &length, sizeof(int), &dw, NULL);

ReadFile(file, buf, length, &dw, NULL);

buf[length] = '\0';

tree->add(key, buf);

}

delete[] buf;

return tree;

}

nodeTree\* createNode(int key, const char\* value)

{

struct nodeTree\* node = new nodeTree;

node->key = key;

node->value = \_strdup(value);

node->left = NULL;

node->right = NULL;

return node;

}

void BinaryTree::leftPrint(nodeTree\* tree)

{

if (tree == NULL) return;

leftPrint(tree->left);

leftPrint(tree->right);

printf("key: %d; value: %s\n", tree->key, tree->value);

}

void BinaryTree::deleteNode(nodeTree\* tree)

{

if (tree) {

deleteNode(tree->left);

deleteNode(tree->right);

free(tree);

}

tree = NULL;

}

BinaryTree::BinaryTree()

{

root = NULL;

size = 0;

}

BinaryTree::BinaryTree(bool f)

{

root = NULL;

size = 0;

buf\_key = 0;

buf\_value = \_strdup("");

DWORD id;

bufEmpty = CreateEvent(NULL, FALSE, TRUE, NULL);

bufFull = CreateEvent(NULL, FALSE, FALSE, NULL);

stopThread = CreateEvent(NULL, TRUE, FALSE, NULL);

HANDLE fileTo = CreateFile(L"..\\Debug\\file.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

CloseHandle(fileTo);

hThread = CreateThread(NULL, 0, ThreadFunc, this, 0, &id);

}

const nodeTree\* BinaryTree::getRoot()

{

return root;

}

void BinaryTree::add(int key, const char\* value)

{

struct nodeTree\* parent;

parent = NULL;

if (root == NULL) {

root = createNode(key, value);

size++;

}

else {

nodeTree\* tree = root;

while (tree != NULL) {

parent = tree;

if (key < tree->key) {

tree = tree->left;

}

else if (key > tree->key) {

tree = tree->right;

}

else {

return;

}

}

/\* Создаем элемент и связываем с узлом \*/

nodeTree\* node = createNode(key, value);

if (key < parent->key)

parent->left = node;

else

parent->right = node;

size++;

}

}

void BinaryTree::leftPrint()

{

leftPrint(root);

}

BinaryTree::~BinaryTree()

{

deleteNode(root);

}

void BinaryTree::callAdd(int key, const char\* value)

{

WaitForSingleObject(bufEmpty, INFINITE);

buf\_key = key;

buf\_value = \_strdup(value);

SetEvent(bufFull);

}

void writeInFile(BinaryTree\* tree, HANDLE file)

{

writeFile(tree->root, file);

}

dllmain.cpp:

#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

DoughterProcess:

Source.cpp:

#include <Windows.h>

#include "BinaryTreeDll.h"

int main() {

BinaryTree\* a = new BinaryTree(true);

a->callAdd(28, "Ситникова");

a->callAdd(23, "Гижевская");

a->callAdd(60, "Борисова");

a->callAdd(11, "Булатник");

a->callAdd(22, "Петрова");

a->callAdd(33, "Корень");

a->callAdd(44, "Азаматов");

a->callAdd(55, "Сухов");

a->callAdd(66, "Кротков");

a->callAdd(77, "Абрамов");

a->callAdd(88, "Мбингу");

a->callAdd(99, "Черкасов");

WaitForSingleObject(a->bufEmpty, INFINITE);

SetEvent(a->stopThread);

system("pause");

return 0;

}

Lab8:

Main.cpp:

#include <iostream>

#include "BinaryTreeDll.h"

void main() {

setlocale(LC\_ALL, "RUS");

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(STARTUPINFO));

ZeroMemory(&pi, sizeof(PROCESS\_INFORMATION));

si.cb = sizeof(STARTUPINFO);

CreateProcess(L"..\\Debug\\DoughterProcess.exe", NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi);

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

printf("Дерево (левый обход):\n");

HANDLE fileFrom = CreateFile(L"..\\Debug\\file.txt", GENERIC\_ALL, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

BinaryTree\* a = readInFile(fileFrom);

CloseHandle(fileFrom);

a->leftPrint();

system("pause");

}