МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра информационных систем и технологий

Отчёт по лабораторному практикуму  
по курсу «Операционные системы»

Вариант №2

Выполнил:  
Алеев И.И.

гр. 6304-090301D

Проверил:

Востокин С. В.

**Цель лабораторной работы №3:** 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 сумму элементов строки в динамическом массиве размерностью NxM

**Листинг программы:**

Выбор между методами работы с динамическй памятья происходит с помощью задания макроса USENATIVE

#include <stdio.h>

#include <locale.h>

#define USENATIVE

#ifdef USENATIVE

#include <Windows.h>

void\* MyAlloc(size\_t siz)

{

HANDLE h = GetProcessHeap();

return (void\*)HeapAlloc(h, 0, siz);

}

BOOL MyFree(void\* ptr)

{

return HeapFree(GetProcessHeap(), 0, (LPVOID)ptr);

}

#define ALLOC(X) MyAlloc(X)

#define FREE(X) MyFree(X)

#else

#include <stdlib.h>

#define ALLOC(X) malloc(X)

#define FREE(X) free(X)

#endif

void main(void)

{

int arr\_dim1, arr\_dim2;

setlocale(LC\_ALL, "ru");

printf("Введите размеры массива: ");

scanf\_s("%d %d", &arr\_dim1, &arr\_dim2);

int\*\* arr = ALLOC(sizeof(int\*) \* arr\_dim1);

if (!arr)

{

fprintf(stderr, "unable alloc() for %d bytes\n", (int)sizeof(int) \* arr\_dim1 \* arr\_dim2);

return;

}

for (int i = 0; i < arr\_dim1; i++)

{

arr[i] = ALLOC(sizeof(int) \* arr\_dim2);

}

for (size\_t i = 0; i < arr\_dim1; i++)

{

for (size\_t j = 0; j < arr\_dim2; j++)

{

arr[i][j] = i \* arr\_dim2 + j;

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

}

putchar('\n');

}

putchar('\n');

for (size\_t i = 0; i < arr\_dim1; i++)

{

float sum = 0;

for (size\_t j = 0; j < arr\_dim2; j++)

sum += arr[i][j];

for (size\_t j = 0; j < arr\_dim2; j++)

\*((float\*)&arr[i][j]) = arr[i][j] / sum;

for (size\_t j = 0; j < arr\_dim2; j++)

printf("%01.4lf\t", \*((float\*)&arr[i][j]));

putchar('\n');

}

for (int i = 0; i < arr\_dim1; i++)

{

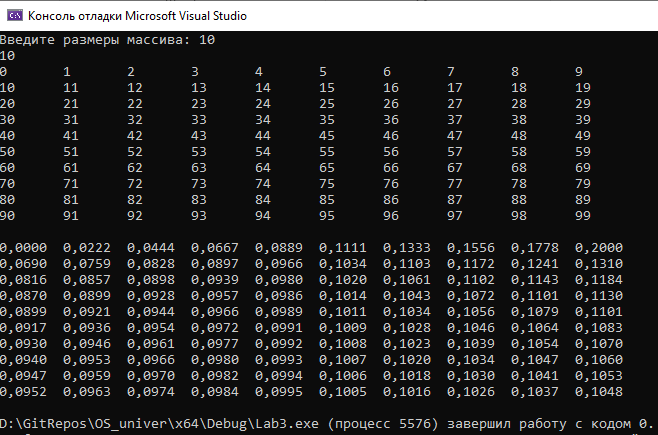
FREE(arr[i]);

}

FREE(arr);

}

**Результат выполнения программы:**



**Цель лабораторной работы №4:** 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 связный список.

**Листинг программы:**

Фаил LinkedList.h

#pragma once

#include <stdint.h>

#include <Windows.h>

struct LinkedListNode\_s;

struct LinkedList\_s;

struct LinkedList\_s

{

struct LinkedListNode\_s\* head;

};

struct LinkedListNode\_s

{

void\* data;

size\_t data\_size;

struct LinkedListNode\_s \*next;

struct LinkedListNode\_s \*prev;

};

typedef struct LinkedList\_s LinkedList;

typedef struct LinkedListNode\_s LinkedListNode;

LinkedList\* new\_LinkedList ();

void append\_Linkedlist (LinkedList\*, void\*, size\_t);

void removeAfter\_Linkedlist (LinkedListNode\*);

void removeBefore\_Linkedlist(LinkedListNode\*);

void remove\_LinkedList(LinkedListNode\*);

void addAfter\_Linkedlist (LinkedListNode\*, void\*, size\_t);

void addBefore\_Linkedlist(LinkedListNode\*, void\*, size\_t);

void clear\_Linkedlist (LinkedList\*);

size\_t len\_LinkedList (LinkedList\*);

void del\_LinkedList(LinkedList\*);

void savebin\_LinkedList(HANDLE, LinkedList\*);

void restorebin\_LinkedList(HANDLE, LinkedList\*);

Фаил AssociativeArray.h

#pragma once

#include "LinkedList.h"

struct Pair\_s;

struct AssociativeArray\_s;

struct Pair\_s

{

void\* key\_ptr;

size\_t key\_size;

void\* value\_ptr;

size\_t value\_size;

};

struct AssociativeArray\_s

{

LinkedList\* l;

};

typedef struct AssociativeArray\_s AssociativeArray;

typedef struct Pair\_s Pair;

AssociativeArray\* new\_AssociativeArray();

void addPair\_AssociativeArray(Pair, AssociativeArray\*);

void addKeyValue\_AssociativeArray(void\* key, size\_t key\_size, void\* value, size\_t value\_size, AssociativeArray\*);

size\_t len\_AssociativeArray(AssociativeArray\* a);

Pair\* at\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

void remove\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

void del\_AssociativeArray(AssociativeArray\*);

void savebin\_AssociativeArray(HANDLE, AssociativeArray\*);

void restorebin\_AssociativeArray(HANDLE, AssociativeArray\*);

Фаил LinkedList.c

#pragma once

#include <stdlib.h>

#include <string.h>

#include "LinkedList.h"

// Cycle double linked list

LinkedList\* new\_LinkedList()

{

LinkedList\* ret = (LinkedList\*)malloc(sizeof(LinkedList));

if (ret)

{

ret->head = (LinkedListNode\*)malloc(sizeof(LinkedListNode));

ret->head->data = 0;

ret->head->data\_size = 0;

ret->head->next = ret->head;

ret->head->prev = ret->head;

}

return ret;

}

void append\_Linkedlist(LinkedList\* l, void\* data\_ptr, size\_t siz)

{

addBefore\_Linkedlist(l->head, data\_ptr, siz);

}

void removeAfter\_Linkedlist(LinkedListNode\* n)

{

if (!n && !n->next)

return;

remove\_LinkedList(n->next);

}

void removeBefore\_Linkedlist(LinkedListNode\* n)

{

if (!n && !n->prev)

return;

remove\_LinkedList(n->prev);

}

void remove\_LinkedList(LinkedListNode\* n)

{

if (!n)

return;

n->prev->next = n->next;

n->next->prev = n->prev;

free(n->data);

free(n);

}

void addAfter\_Linkedlist(LinkedListNode\* n, void\* data\_ptr, size\_t siz)

{

if (!n)

return;

LinkedListNode\* nn = malloc(sizeof(LinkedListNode));

nn->data = malloc(siz);

nn->data\_size = siz;

nn->next = n->next;

nn->prev = n;

n->next = nn;

n->next->prev = nn;

memcpy\_s(nn->data, nn->data\_size, data\_ptr, siz);

}

void addBefore\_Linkedlist(LinkedListNode\* n, void\* data\_ptr, size\_t siz)

{

if (!n)

return;

LinkedListNode\* nn = malloc(sizeof(LinkedListNode));

nn->data = malloc(siz);

nn->data\_size = siz;

nn->next = n;

nn->prev = n->prev;

nn->prev->next = nn;

n->prev = nn;

memcpy\_s(nn->data, siz, data\_ptr, siz);

}

void clear\_Linkedlist(LinkedList\* l)

{

if (!l)

return;

LinkedListNode\* n = l->head->next;

l->head->next = l->head;

l->head->prev = l->head;

while (n != l->head)

{

free(n->data);

n = n->next;

if (n->prev != l->head)

free(n->prev);

}

}

size\_t len\_LinkedList(LinkedList\* l)

{

if (!l)

return -1;

size\_t ret = 1;

LinkedListNode\* n = l->head->next;

while (n != l->head)

{

n = n->next;

ret++;

}

return ret;

}

void del\_LinkedList(LinkedList\* l)

{

clear\_Linkedlist(l);

remove\_LinkedList(l->head);

free(l);

}

Фаил AssociativeArray.c

#include <stdlib.h>

#include <string.h>

#include "AssociativeArray.h"

uint8\_t byte\_cmp(void\* p1, void\* p2, size\_t s, size\_t ss)

{

uint8\_t ret = s == ss;

while (ret && s >= 0)

{

s--;

ret = (((uint8\_t\*)p1)[s] == ((uint8\_t\*)p1)[s]);

}

return ret;

}

AssociativeArray\* new\_AssociativeArray()

{

AssociativeArray\* ret = malloc(sizeof(AssociativeArray));

if (ret)

ret->l = new\_LinkedList();

return ret;

}

void addPair\_AssociativeArray(Pair p, AssociativeArray\* a)

{

append\_Linkedlist(a->l, &p, sizeof(p));

}

void addKeyValue\_AssociativeArray(void\* key, size\_t key\_size, void\* value, size\_t value\_size, AssociativeArray\* a)

{

// copy key and data

void

\* dup\_key = malloc(key\_size),

\* dup\_val = malloc(value\_size);

if (!dup\_key || !dup\_val)

return;

memcpy(dup\_key, key, key\_size);

memcpy(dup\_val, value, value\_size);

Pair p = (Pair){ .key\_ptr = dup\_key, .key\_size = key\_size, .value\_ptr = dup\_val, .value\_size = value\_size };

append\_Linkedlist(a->l, &p, sizeof(Pair));

}

Pair\* at\_AssociativeArray(void\* key\_ptr, size\_t key\_size, AssociativeArray\* a)

{

LinkedListNode\* n = a->l->head->next;

while (

n != a->l->head &&

((Pair\*)n->data)->key\_size == key\_size &&

memcmp(((Pair\*)n->data)->key\_ptr, key\_ptr, key\_size) )

{

n = n->next;

}

if (n != a->l->head)

return n->data;

else

return NULL;

}

size\_t len\_AssociativeArray(AssociativeArray\* a)

{

return len\_LinkedList(a->l);

}

void remove\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\* a)

{

LinkedListNode\* n = a->l->head->next;

while (n != a->l->head && !byte\_cmp(key, n->data, key\_size, n->data\_size))

{

n = n->next;

}

if (n == a->l->head)

return;

remove\_LinkedList(n);

}

void del\_AssociativeArray(AssociativeArray\* a)

{

del\_LinkedList(a->l);

free(a);

}

Фаил main.c

#include <stdio.h>

#include "LinkedList.h"

#include "AssociativeArray.h"

#define NODE\_ITERATE

int main(int argc, char\* argv[])

{

AssociativeArray\* l = new\_AssociativeArray();

for (size\_t i = 0; i < 32; i++)

{

int sq = i \* i;

addKeyValue\_AssociativeArray(&i, sizeof(i), &sq, sizeof(sq), l);

}

#ifdef NODE\_ITERATE

LinkedListNode\* n = l->l->head->next;

while (n != l->l->head)

{

Pair\* p = n->data;

if (p)

printf("k: %02lli\tv: %04u\t%p\n", \*((size\_t\*)p->key\_ptr), \*((int\*)p->value\_ptr), p);

else

printf("%p\n", p);

n = n->next;

}

#else

size\_t len = len\_AssociativeArray(l);

printf("len with head: %llu\n", len);

for (size\_t i = 0; i < len - 1; i++)

{

Pair\* p2 = at\_AssociativeArray(&i, sizeof(i), l);

if (p2)

printf("%02llu\tk: %02llu\tv: %04u\t%p\n", i, \*((size\_t\*)p2->key\_ptr), \*((int\*)p2->value\_ptr), p2);

else

printf("%02lld\tNULL\n", i);

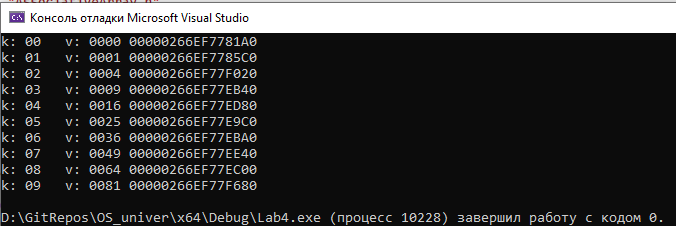
}

#endif

return 0;

}

**Результат выполнения программы:**

****

**Цель лабораторной работы №5:** 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 сохраненного файла.

**Задание:** Требуется реализовать структуру данных «ассоциативный массив», используя связный список.

**Листинг программы:**

**Результат выполнения программы:**

mem.h

#pragma once

#include <Windows.h>

void\* MyAlloc(size\_t siz);

BOOL MyFree(void\* ptr);

int MyMemcpy(void\*, void\*, size\_t);

int MyMemcmp(void\*, void\*, size\_t);

FileIO.h

#pragma once

#include <Windows.h>

enum FMODE

{

FMODE\_READ = 1 << 1,

FMODE\_WRITE = 1 << 2,

FMODE\_APPEND=(1 << 2) | (1 << 3),

};

enum FPOS

{

FPOS\_BEGIN = 0,

FPOS\_ADD = 1,

FPOS\_END = 2

};

HANDLE file\_open(char\*,enum FMODE, DWORD);

void file\_close(HANDLE);

size\_t file\_read(HANDLE, void\*, size\_t);

size\_t file\_write(HANDLE, void\*, size\_t);

size\_t file\_pos(HANDLE, UINT64, enum FPOS);

BOOL file\_iseof(HANDLE);

LinkedList.h

#pragma once

#include <stdint.h>

#include <Windows.h>

struct LinkedListNode\_s;

struct LinkedList\_s;

struct LinkedList\_s

{

struct LinkedListNode\_s\* head;

};

struct LinkedListNode\_s

{

void\* data;

size\_t data\_size;

struct LinkedListNode\_s \*next;

struct LinkedListNode\_s \*prev;

};

typedef struct LinkedList\_s LinkedList;

typedef struct LinkedListNode\_s LinkedListNode;

LinkedList\* new\_LinkedList ();

void append\_Linkedlist (LinkedList\*, void\*, size\_t);

void removeAfter\_Linkedlist (LinkedListNode\*);

void removeBefore\_Linkedlist(LinkedListNode\*);

void remove\_LinkedList(LinkedListNode\*);

void addAfter\_Linkedlist (LinkedListNode\*, void\*, size\_t);

void addBefore\_Linkedlist(LinkedListNode\*, void\*, size\_t);

void clear\_Linkedlist (LinkedList\*);

size\_t len\_LinkedList (LinkedList\*);

void del\_LinkedList(LinkedList\*);

void savebin\_LinkedList(HANDLE, LinkedList\*);

void restorebin\_LinkedList(HANDLE, LinkedList\*);

AssociativeArray.h

#pragma once

#include "LinkedList.h"

struct Pair\_s;

struct AssociativeArray\_s;

struct Pair\_s

{

void\* key\_ptr;

size\_t key\_size;

void\* value\_ptr;

size\_t value\_size;

};

struct AssociativeArray\_s

{

LinkedList\* l;

};

typedef struct AssociativeArray\_s AssociativeArray;

typedef struct Pair\_s Pair;

AssociativeArray\* new\_AssociativeArray();

void addPair\_AssociativeArray(Pair, AssociativeArray\*);

void addKeyValue\_AssociativeArray(void\* key, size\_t key\_size, void\* value, size\_t value\_size, AssociativeArray\*);

size\_t len\_AssociativeArray(AssociativeArray\* a);

Pair\* at\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

void remove\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

void del\_AssociativeArray(AssociativeArray\*);

void savebin\_AssociativeArray(HANDLE, AssociativeArray\*);

void restorebin\_AssociativeArray(HANDLE, AssociativeArray\*);

FileIO.c

#include <Windows.h>

#include "FileIO.h"

HANDLE file\_open(char\* path, enum FMODE mode, DWORD conv)

{

DWORD access = 0;

if (mode & FMODE\_READ)

access |= GENERIC\_READ;

if (mode & FMODE\_WRITE)

access |= GENERIC\_WRITE;

//if (mode & FMODE\_APPEND)

// access |= GENERIC\_WRITE;

HANDLE fd = CreateFileA(path, access, 0, NULL, conv, FILE\_ATTRIBUTE\_NORMAL, NULL);

return fd;

}

void file\_close(HANDLE fd)

{

CloseHandle(fd);

}

size\_t file\_read(HANDLE fd, void\* buf, size\_t buff\_size)

{

size\_t ret = 0;

ReadFile(fd, buf, buff\_size, &ret, NULL);

return ret;

}

size\_t file\_write(HANDLE fd, void\* buf, size\_t buff\_size)

{

size\_t ret = 0;

WriteFile(fd, buf, buff\_size, &ret, NULL);

return ret;

}

size\_t file\_pos(HANDLE fd, UINT64 offset, enum FPOS dir)

{

size\_t ret;

SetFilePointerEx(fd, (LARGE\_INTEGER) { .QuadPart = offset }, &ret, dir);

return ret;

}

BOOL file\_iseof(HANDLE fd)

{

size\_t tmp = file\_pos(fd, 0, FPOS\_ADD);

size\_t end = file\_pos(fd, 0, FPOS\_END);

file\_pos(fd, tmp, FPOS\_BEGIN);

return tmp == end;

}

mem.c

#include "mem.h"

void\* MyAlloc(size\_t siz)

{

HANDLE h = GetProcessHeap();

return (void\*)HeapAlloc(h, 0, siz);

}

BOOL MyFree(void\* ptr)

{

return HeapFree(GetProcessHeap(), 0, (LPVOID)ptr);

}

int MyMemcpy(void\* \_1, void\* \_2, size\_t \_3)

{

return memcpy(\_1, \_2, \_3);

}

int MyMemcmp(void\* \_1, void\* \_2, size\_t \_3)

{

return memcmp(\_1, \_2, \_3);

}

LinkedList.c

#pragma once

#include <stdlib.h>

#include <string.h>

#include "LinkedList.h"

#include "FileIO.h"

// Cycle double linked list

LinkedList\* new\_LinkedList()

{

LinkedList\* ret = (LinkedList\*)malloc(sizeof(LinkedList));

if (ret)

{

ret->head = (LinkedListNode\*)malloc(sizeof(LinkedListNode));

ret->head->data = 0;

ret->head->data\_size = 0;

ret->head->next = ret->head;

ret->head->prev = ret->head;

}

return ret;

}

void append\_Linkedlist(LinkedList\* l, void\* data\_ptr, size\_t siz)

{

addBefore\_Linkedlist(l->head, data\_ptr, siz);

}

void removeAfter\_Linkedlist(LinkedListNode\* n)

{

if (!n && !n->next)

return;

remove\_LinkedList(n->next);

}

void removeBefore\_Linkedlist(LinkedListNode\* n)

{

if (!n && !n->prev)

return;

remove(n->prev);

}

void remove\_LinkedList(LinkedListNode\* n)

{

if (!n)

return;

n->prev->next = n->next;

n->next->prev = n->prev;

free(n->data);

free(n);

}

void addAfter\_Linkedlist(LinkedListNode\* n, void\* data\_ptr, size\_t siz)

{

if (!n)

return;

LinkedListNode\* nn = malloc(sizeof(LinkedListNode));

nn->data = malloc(siz);

nn->data\_size = siz;

nn->next = n->next;

nn->prev = n;

n->next = nn;

n->next->prev = nn;

memcpy\_s(nn->data, nn->data\_size, data\_ptr, siz);

}

void addBefore\_Linkedlist(LinkedListNode\* n, void\* data\_ptr, size\_t siz)

{

if (!n)

return;

LinkedListNode\* nn = malloc(sizeof(LinkedListNode));

nn->data = malloc(siz);

nn->data\_size = siz;

nn->next = n;

nn->prev = n->prev;

nn->prev->next = nn;

n->prev = nn;

memcpy\_s(nn->data, siz, data\_ptr, siz);

}

void clear\_Linkedlist(LinkedList\* l)

{

if (!l)

return;

LinkedListNode\* n = l->head->next;

l->head->next = l->head;

l->head->prev = l->head;

while (n != l->head)

{

free(n->data);

n = n->next;

if (n->prev != l->head)

free(n->prev);

}

}

size\_t len\_LinkedList(LinkedList\* l)

{

if (!l)

return;

size\_t ret = 1;

LinkedListNode\* n = l->head->next;

while (n != l->head)

{

n = n->next;

ret++;

}

return ret;

}

void del\_LinkedList(LinkedList\* l)

{

clear\_Linkedlist(l);

remove\_LinkedList(l->head);

free(l);

}

void savebin\_LinkedList(HANDLE fd, LinkedList\* l)

{

LinkedListNode \*n = l->head->next;

while (n != l->head)

{

file\_write(fd, &(n->data\_size), sizeof(n->data\_size));

file\_write(fd, n->data, n->data\_size);

n = n->next;

}

}

void restorebin\_LinkedList(HANDLE fd, LinkedList\* l)

{

void\* buf;

size\_t siz;

while (!file\_iseof(fd))

{

file\_read(fd, &siz, sizeof(siz));

buf = malloc(siz);

file\_read(fd, buf, siz);

addBefore\_Linkedlist(l->head, buf, siz);

free(buf);

}

}

AssociativeArray.c

#include <stdlib.h>

#include <string.h>

#include "AssociativeArray.h"

uint8\_t byte\_cmp(void\* p1, void\* p2, size\_t s, size\_t ss)

{

uint8\_t ret = s == ss;

while (ret && s >= 0)

{

s--;

ret = (((uint8\_t\*)p1)[s] == ((uint8\_t\*)p1)[s]);

}

return ret;

}

AssociativeArray\* new\_AssociativeArray()

{

AssociativeArray\* ret = malloc(sizeof(AssociativeArray));

if (ret)

ret->l = new\_LinkedList();

return ret;

}

void addPair\_AssociativeArray(Pair p, AssociativeArray\* a)

{

void

\* dup\_key = malloc(p.key\_size),

\* dup\_val = malloc(p.value\_size);

memcpy(dup\_key, p.key\_ptr, p.key\_size);

memcpy(dup\_val, p.value\_ptr, p.value\_size);

append\_Linkedlist(a->l, &p, sizeof(p));

}

void addKeyValue\_AssociativeArray(void\* key, size\_t key\_size, void\* value, size\_t value\_size, AssociativeArray\* a)

{

// copy key and data

void

\* dup\_key = malloc(key\_size),

\* dup\_val = malloc(value\_size);

if (!dup\_key || !dup\_val)

return;

memcpy(dup\_key, key, key\_size);

memcpy(dup\_val, value, value\_size);

Pair p = (Pair){ .key\_ptr = dup\_key, .key\_size = key\_size, .value\_ptr = dup\_val, .value\_size = value\_size };

append\_Linkedlist(a->l, &p, sizeof(Pair));

}

Pair\* at\_AssociativeArray(void\* key\_ptr, size\_t key\_size, AssociativeArray\* a)

{

LinkedListNode\* n = a->l->head->next;

while (

n != a->l->head &&

((Pair\*)n->data)->key\_size == key\_size &&

memcmp(((Pair\*)n->data)->key\_ptr, key\_ptr, key\_size) )

{

n = n->next;

}

if (n != a->l->head)

return n->data;

else

return NULL;

}

size\_t len\_AssociativeArray(AssociativeArray\* a)

{

return len\_LinkedList(a->l);

}

void remove\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\* a)

{

LinkedListNode\* n = a->l->head->next;

while (n != a->l->head && !byte\_cmp(key, n->data, key\_size, n->data\_size))

{

n = n->next;

}

if (n == a->l->head)

return;

remove\_LinkedList(n);

}

void del\_AssociativeArray(AssociativeArray\* a)

{

del\_LinkedList(a->l);

free(a);

}

void savebin\_AssociativeArray(HANDLE fd, AssociativeArray\* a)

{

LinkedListNode\* n = a->l->head->next;

while (n != a->l->head)

{

Pair\* p = n->data;

file\_write(fd, &p->key\_size, sizeof(p->key\_size));

file\_write(fd, p->key\_ptr, p->key\_size);

file\_write(fd, &p->value\_size, sizeof(p->value\_size));

file\_write(fd, p->value\_ptr, p->value\_size);

n = n->next;

}

}

void restorebin\_AssociativeArray(HANDLE fd, AssociativeArray\* a)

{

void\* buf;

size\_t siz;

while (!file\_iseof(fd))

{

Pair\* p = malloc(sizeof(Pair));

file\_read(fd, &p->key\_size, sizeof(p->key\_size));

p->key\_ptr = malloc(p->key\_size);

file\_read(fd, p->key\_ptr, p->key\_size);

file\_read(fd, &p->value\_size, sizeof(p->value\_size));

p->value\_ptr = malloc(p->value\_size);

file\_read(fd, p->value\_ptr, p->value\_size);

addKeyValue\_AssociativeArray(p->key\_ptr, p->key\_size, p->value\_ptr, p->value\_size, a);

free(p->key\_ptr);

free(p->value\_ptr);

free(p);

}

}

Выбор между операциями записи и чтения выполняется с помощью макроса Save

Lab5.c

#include <stdio.h>

#include <Windows.h>

#include "AssociativeArray.h"

#include "FileIO.h"

//#define Save

int main(int argc, char\* argv[])

{

AssociativeArray\* a = new\_AssociativeArray();

#ifdef Save

for (size\_t i = 0; i <= 10; i++)

{

uint32\_t pow3 = i\*i\*i;

addKeyValue\_AssociativeArray(&i, sizeof(i), &pow3, sizeof(pow3), a);

}

HANDLE fd = file\_open("D:\\bin\_test.bin", FMODE\_WRITE | FMODE\_READ, CREATE\_ALWAYS);

savebin\_AssociativeArray(fd, a);

#else

HANDLE fd = file\_open("D:\\bin\_test.bin", FMODE\_READ, OPEN\_ALWAYS);

restorebin\_AssociativeArray(fd,a);

LinkedListNode\* n = a->l->head->next;

while (n != a->l->head)

{

Pair\* p = n->data;

if (p)

printf("k: %02lli\tv: %06u\t%p\n", \*((size\_t\*)p->key\_ptr), \*((int\*)p->value\_ptr), p);

else

printf("%p\n", p);

n = n->next;

}

#endif

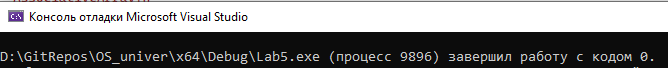
file\_close(fd);

return 0;

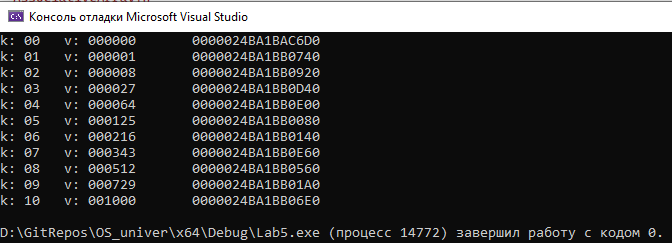
}

**Результат выполнения программы:**

Сохранение:



Чтение:



**Цель лабораторной работы №6:** 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 связный список.

**Листинг программы:**

Проект динамической библиотеки

mem.h

#pragma once

#include <Windows.h>

typedef void\* (\*Allocator)(size\_t);

typedef void (\*Deallocator)(void\*);

int MyMemcpy(void\*, void\*, size\_t);

int MyMemcmp(void\*, void\*, size\_t);

LinkedList.h

#pragma once

#include <stdint.h>

#include <Windows.h>

#include "mem.h"

struct LinkedListNode\_s;

struct LinkedList\_s;

struct LinkedList\_s

{

struct LinkedListNode\_s\* head;

Allocator a;

Deallocator d;

};

struct LinkedListNode\_s

{

void\* data;

size\_t data\_size;

struct LinkedListNode\_s \*next;

struct LinkedListNode\_s \*prev;

};

typedef struct LinkedList\_s LinkedList;

typedef struct LinkedListNode\_s LinkedListNode;

\_CONTAINER\_API LinkedList\* new\_LinkedList (Allocator, Deallocator);

\_CONTAINER\_API void append\_Linkedlist (LinkedList\*, void\*, size\_t, LinkedList\*);

\_CONTAINER\_API void removeAfter\_Linkedlist (LinkedListNode\*, LinkedList\*);

\_CONTAINER\_API void removeBefore\_Linkedlist(LinkedListNode\*, LinkedList\*);

\_CONTAINER\_API void remove\_LinkedList(LinkedListNode\*, LinkedList\*);

\_CONTAINER\_API void addAfter\_Linkedlist (LinkedListNode\*, void\*, size\_t, LinkedList\*);

\_CONTAINER\_API void addBefore\_Linkedlist(LinkedListNode\*, void\*, size\_t, LinkedList\*);

\_CONTAINER\_API void clear\_Linkedlist (LinkedList\*);

\_CONTAINER\_API size\_t len\_LinkedList (LinkedList\*);

\_CONTAINER\_API void del\_LinkedList(LinkedList\*);

\_CONTAINER\_API void savebin\_LinkedList(HANDLE, LinkedList\*);

\_CONTAINER\_API void restorebin\_LinkedList(HANDLE, LinkedList\*);

FileIO.h

#pragma once

#include <Windows.h>

enum FMODE

{

FMODE\_READ = 1 << 1,

FMODE\_WRITE = 1 << 2,

FMODE\_APPEND=(1 << 2) | (1 << 3),

};

enum FPOS

{

FPOS\_BEGIN = 0,

FPOS\_ADD = 1,

FPOS\_END = 2

};

\_CONTAINER\_API HANDLE file\_open(char\*,enum FMODE, DWORD);

\_CONTAINER\_API void file\_close(HANDLE);

\_CONTAINER\_API size\_t file\_read(HANDLE, void\*, size\_t);

\_CONTAINER\_API size\_t file\_write(HANDLE, void\*, size\_t);

\_CONTAINER\_API size\_t file\_pos(HANDLE, UINT64, enum FPOS);

\_CONTAINER\_API BOOL file\_iseof(HANDLE);

Containers.h

#pragma once

#include "pch.h"

struct Pair\_s;

struct AssociativeArray\_s;

struct LinkedListNode\_s;

struct LinkedList\_s;

typedef struct LinkedList\_s LinkedList;

typedef struct LinkedListNode\_s LinkedListNode;

typedef struct AssociativeArray\_s AssociativeArray;

typedef struct Pair\_s Pair;

struct LinkedList\_s

{

struct LinkedListNode\_s\* head;

};

struct LinkedListNode\_s

{

void\* data;

size\_t data\_size;

struct LinkedListNode\_s\* next;

struct LinkedListNode\_s\* prev;

};

struct Pair\_s

{

void\* key\_ptr;

size\_t key\_size;

void\* value\_ptr;

size\_t value\_size;

};

struct AssociativeArray\_s

{

LinkedList\* l;

};

typedef void\* (\*Allocator)(size\_t);

typedef void (\*Deallocator)(void\*);

#ifndef LAB6\_ONLYTYPE

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

CONTAINER\_API HANDLE file\_open(char\*, enum FMODE, DWORD);

CONTAINER\_API void file\_close(HANDLE);

CONTAINER\_API size\_t file\_read(HANDLE, void\*, size\_t);

CONTAINER\_API size\_t file\_write(HANDLE, void\*, size\_t);

CONTAINER\_API size\_t file\_pos(HANDLE, UINT64, enum FPOS);

CONTAINER\_API BOOL file\_iseof(HANDLE);

CONTAINER\_API LinkedList\* new\_LinkedList(Allocator, Deallocator);

CONTAINER\_API void append\_Linkedlist(LinkedList\*, void\*, size\_t, LinkedList\*);

CONTAINER\_API void removeAfter\_Linkedlist(LinkedListNode\*, LinkedList\*);

CONTAINER\_API void removeBefore\_Linkedlist(LinkedListNode\*, LinkedList\*);

CONTAINER\_API void remove\_LinkedList(LinkedListNode\*, LinkedList\*);

CONTAINER\_API void addAfter\_Linkedlist(LinkedListNode\*, void\*, size\_t, LinkedList\*);

CONTAINER\_API void addBefore\_Linkedlist(LinkedListNode\*, void\*, size\_t, LinkedList\*);

CONTAINER\_API void clear\_Linkedlist(LinkedList\*);

CONTAINER\_API size\_t len\_LinkedList(LinkedList\*);

CONTAINER\_API void del\_LinkedList(LinkedList\*);

CONTAINER\_API void savebin\_LinkedList(HANDLE, LinkedList\*);

CONTAINER\_API void restorebin\_LinkedList(HANDLE, LinkedList\*);

CONTAINER\_API AssociativeArray\* new\_AssociativeArray(Allocator, Deallocator);

CONTAINER\_API void addPair\_AssociativeArray(Pair, AssociativeArray\*);

CONTAINER\_API void addKeyValue\_AssociativeArray(void\* key, size\_t key\_size, void\* value, size\_t value\_size, AssociativeArray\*);

CONTAINER\_API size\_t len\_AssociativeArray(AssociativeArray\* a);

CONTAINER\_API Pair\* at\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

CONTAINER\_API void remove\_AssociativeArray(void\* key, size\_t key\_size, AssociativeArray\*);

CONTAINER\_API void del\_AssociativeArray(AssociativeArray\*);

CONTAINER\_API void savebin\_AssociativeArray(HANDLE, AssociativeArray\*);

CONTAINER\_API void restorebin\_AssociativeArray(HANDLE, AssociativeArray\*);

#endif

Файлы создаваемые по-умолчанию:

framework.h

#pragma once

#define WIN32\_LEAN\_AND\_MEAN // Исключите редко используемые компоненты из заголовков Windows

// Файлы заголовков Windows

#include <windows.h>

pch.h

// pch.h: это предварительно скомпилированный заголовочный файл.

// Перечисленные ниже файлы компилируются только один раз, что ускоряет последующие сборки.

// Это также влияет на работу IntelliSense, включая многие функции просмотра и завершения кода.

// Однако изменение любого из приведенных здесь файлов между операциями сборки приведет к повторной компиляции всех(!) этих файлов.

// Не добавляйте сюда файлы, которые планируете часто изменять, так как в этом случае выигрыша в производительности не будет.

#ifndef PCH\_H

#define PCH\_H

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

// Добавьте сюда заголовочные файлы для предварительной компиляции

#include "framework.h"

#endif //PCH\_H

pch.c

// pch.cpp: файл исходного кода, соответствующий предварительно скомпилированному заголовочному файлу

#include "pch.h"

// При использовании предварительно скомпилированных заголовочных файлов необходим следующий файл исходного кода для выполнения сборки.

**Результат выполнения программы:**

**Цель лабораторной работы №7:** 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 нее – 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 путем чтения файла на диске и печати его содержимого по окончании добавления.

**Задание:** Требуется реализовать структуру данных «ассоциативный массив», используя связный список.

**Листинг программы:**

Header.h

Source.cpp

Lab7\_OS.cpp

**Результат выполнения программы:**

**Цель лабораторной работы №8:** 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 связный список.

**Листинг программы:**

**Решение Lab8\_OS\_Process:**

Lab8\_OS\_Process.cpp

Lab8\_OS.cpp

**Результат выполнения программы:**