

САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА
(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчёт по лабораторному практикуму
по курсу «Операционные системы»

Вариант №4.

Выполнил:
Власов Г.В.
гр. 6303-090301D

Проверил:
Востокин С. В.

Самара 2019

Цель лабораторной работы №1,2:

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:

Напечатать все простые числа, не превосходящие заданное число М.

Листинг программы:

lab1.cpp

```
#include <iostream>

int simple(int m)
{
    int i, k, j;
    for (i = 1; i <= m; i++)
    {
        k = 0;
        for (j = 1; j <= i; j++)
            if (i % j == 0)
                k++;
        if (k == 2)
        {
            printf("%d\n", i);
        }
    }
    return 0;
}

int main()
{
    int m;
    printf_s("Please, enter number M:");
    scanf_s("%d", &m);
    simple(m);
}
```

Результат выполнения программы:



```
Консоль отладки Microsoft Visual Studio
Please, enter number M:104
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
C:\Users\Георгий\Desktop\OS - копия (2)\Лаб1,2\х64\Debug\Laba1,2.exe (процесс 2320) завершает работу с кодом 0.
```

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:

По динамическому массиву из M вещественных чисел необходимо рассчитать выборочное среднее и выборочную дисперсию.

Листинг программы:

```
#include <iostream>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include "windows.h"
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
    setlocale(LC_ALL, "RUSSIAN");

    int n = 0;
    double MX = 0;
    double DX = 0;
    double sum = 0;
    double summ = 0;

    printf("Введите количество элементов массива: ");
    scanf_s("%d", &n);

    double* arr;
    arr = new double[n];

    /*
    HANDLE heap = HeapCreate(0, n * sizeof(double), n * sizeof(double));
    double *arr;
    arr = (double*)HeapAlloc(heap, HEAP_ZERO_MEMORY, n * sizeof(double));
    */

    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = 0.01 * (rand());
    }

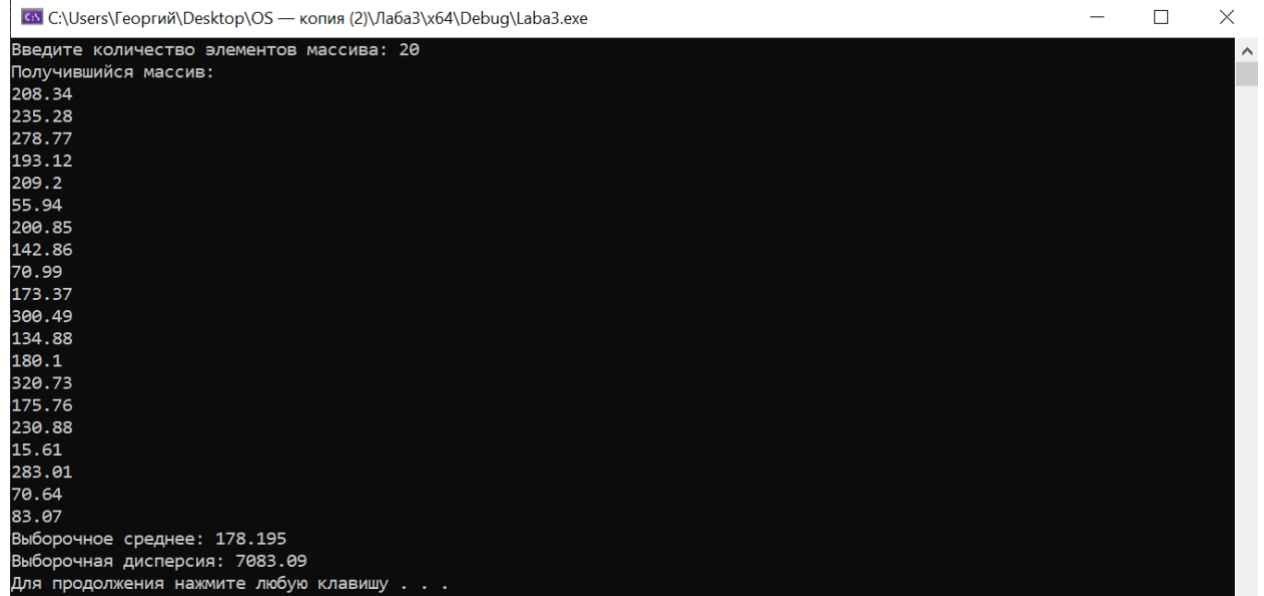
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    MX = sum / n;

    for (int i = 0; i < n; i++) {
        summ += (arr[i] - MX) * (arr[i] - MX);
    }
    DX = summ / n;

    printf("Получившийся массив: \n");
    for (int i = 0; i < n; i++) {
        cout << arr[i] << "\n";
    }
}
```

```
printf("Выборочное среднее: ");  
cout << MX << "\n";  
printf("Выборочная дисперсия: ");  
cout << DX << "\n";  
  
free(arr);  
//HeapDestroy(heap);  
system("pause");  
return 0;  
}
```

Результат выполнения программы:



```
C:\Users\Георгий\Desktop\OS — копия (2)\Лаба3\x64\Debug\Laba3.exe  
Введите количество элементов массива: 20  
Получившийся массив:  
208.34  
235.28  
278.77  
193.12  
209.2  
55.94  
200.85  
142.86  
70.99  
173.37  
300.49  
134.88  
180.1  
320.73  
175.76  
230.88  
15.61  
283.01  
70.64  
83.07  
Выборочное среднее: 178.195  
Выборочная дисперсия: 7083.09  
Для продолжения нажмите любую клавишу . . .
```

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 хэш-таблицу, построенную по методу цепочек.

Листинг программы:

List.h

```
#pragma once
#include<iostream>
#include <string>

class List
{
public:
    struct Node {
        std::string key;
        std::string value;
        Node(const std::string& k, const std::string& v)
        {
            key = k;
            value = v;
            next = nullptr;
        }
        Node* next;
        ~Node() {}
    };
    Node* first;
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    List();
    ~List();
};

inline std::ostream& operator << (std::ostream& os, const List& list)
{
    auto* p = list.first;
    while (p != nullptr)
    {
        os << p->key << " - " << p->value << "\t\n";
        p = p->next;
    }
    return os;
}
```

List.cpp

```
#include "List.h"

List::List()
{
    first = nullptr;
}
List::~~List() {}

void List::add(const std::string& key, const std::string& value)
{
    if (!first)
```

```

        {
            first = new Node(key, value);
        }
        else
        {
            Node* p = first;
            first = new Node(key, value);
            first->next = p;
        }
    }

void List::del(const std::string& key)
{
    if (first->key == key)
    {
        Node* p = first;
        first->next = first->next->next;
    }
    else
    {
        Node* p = first;
        while (p->next && p->next->key != key)
        {
            p = p->next;
        }
        if (!p->next) return;
        Node* n = p->next;
        p->next = p->next->next;
        delete n;
    }
}

```

Assoc_arr.h

```

#pragma once
#include "List.h"
class Assoc_arr
{
public:
    List list[5];
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    static int hashCode(const std::string& value);
};

inline std::ostream& operator <<(std::ostream& os, const Assoc_arr& arr)
{
    for (int i = 0; i < 5; i++)
        os << arr.list[i] << "\t\n";
    return os;
}

```

Assoc_arr.cpp

```

#include "Assoc_arr.h"

int Assoc_arr::hashCode(const std::string& value)
{
    int res = 0;
    for (unsigned int i = 0; i < value.length(); i++)
    {
        res += value[i];
    }
    return res % 5;
}

```

```

void Assoc_arr::add(const std::string& key, const std::string& value)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    while (p != nullptr)
    {
        if (p->key == key)
        {
            p->value = value;
            return;
        }
        p = p->next;
    }
    if (p == nullptr)
    {
        List::Node* temp = list[hashKey].first;
        list[hashKey].first = new List::Node(key, value);
        list[hashKey].first->next = temp;
        return;
    }
}

void Assoc_arr::del(const std::string& key)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    if (p == nullptr)
        return;
    if (p->key == key)
    {
        list[hashKey].first = p->next;
        delete p;
        return;
    }
    List::Node* temp = p;
    while (p && p->key != key)
    {
        temp = p;
        p = p->next;
    }
    if (p != nullptr)
    {
        temp->next = p->next;
        delete p;
    }
}

```

lab4.cpp

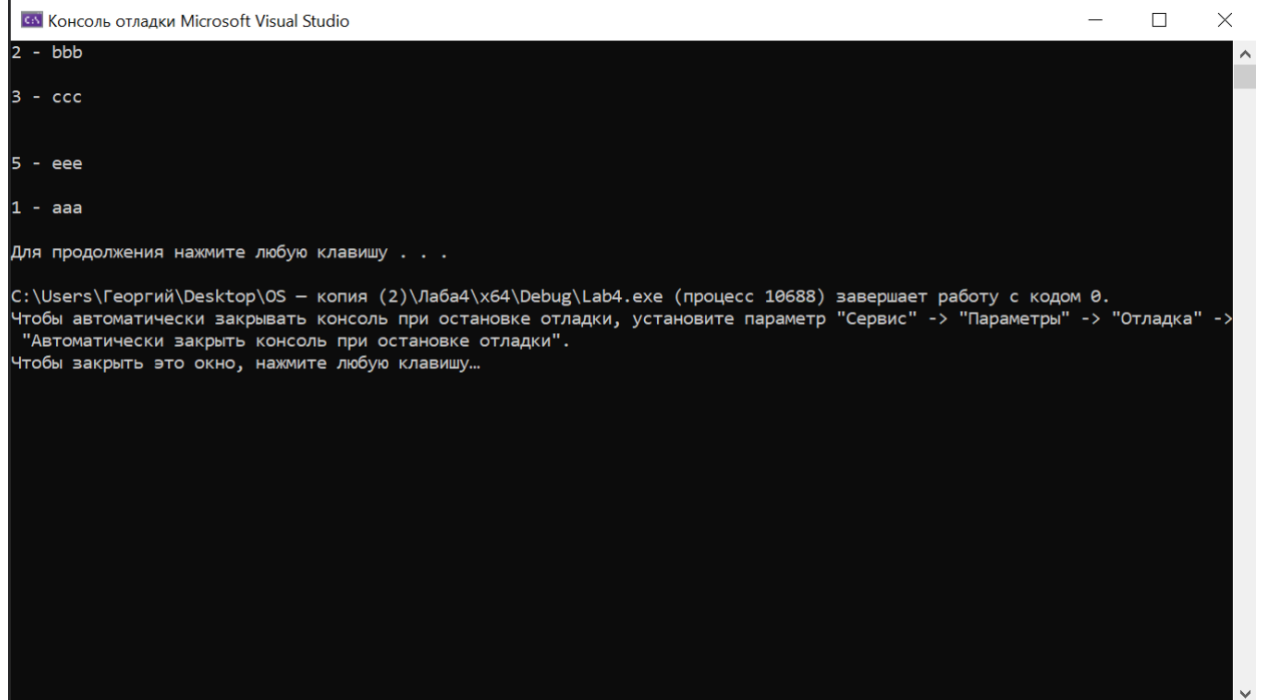
```

#include <iostream>
#include "Assoc_arr.h"

int main()
{
    Assoc_arr array;
    array.add("1", "aaa");
    array.add("2", "bbb");
    array.add("3", "ccc");
    array.add("4", "ddd");
    array.add("5", "eee");
    array.del("4");
    std::cout << array;
    system("pause");
    return 0;
}

```

Результат выполнения программы:



Консоль отладки Microsoft Visual Studio

```
2 - bbb
3 - ccc
5 - eee
1 - aaa

Для продолжения нажмите любую клавишу . . .

C:\Users\Георгий\Desktop\OS – копия (2)\Лаба4\x64\Debug\Lab4.exe (процесс 10688) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```


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 хэш-таблицу, построенную по методу цепочек.

Листинг программы:

List.h

```
#pragma once
#include<iostream>
#include <string>

class List
{
public:
    struct Node {
        std::string key;
        std::string value;
        Node(const std::string& k, const std::string& v)
        {
            key = k;
            value = v;
            next = nullptr;
        }
        Node* next;
        ~Node() {}
    };
    Node* first;
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    List();
    ~List();
};

inline std::ostream& operator << (std::ostream& os, const List& list)
{
    auto* p = list.first;
    while (p != nullptr)
    {
        os << p->key << " - " << p->value << "\t\n";
        p = p->next;
    }
    return os;
}
```

List.cpp

```
#include "List.h"

List::List()
{
    first = nullptr;
}
List::~~List() {}

void List::add(const std::string& key, const std::string& value)
{
    if (!first)
```

```

        {
            first = new Node(key, value);
        }
        else
        {
            Node* p = first;
            first = new Node(key, value);
            first->next = p;
        }
    }

void List::del(const std::string& key)
{
    if (first->key == key)
    {
        Node* p = first;
        first->next = first->next->next;
    }
    else
    {
        Node* p = first;
        while (p->next && p->next->key != key)
        {
            p = p->next;
        }
        if (!p->next) return;
        Node* n = p->next;
        p->next = p->next->next;
        delete n;
    }
}

```

Assoc_arr.h

```

#pragma once
#include "List.h"
class Assoc_arr
{
public:
    List list[5];
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    static int hashCode(const std::string& value);
};

inline std::ostream& operator <<(std::ostream& os, const Assoc_arr& arr)
{
    for (int i = 0; i < 5; i++)
        os << arr.list[i] << "\t\n";
    return os;
}

```

Assoc_arr.cpp

```

#include <iostream>
#include <fstream>
#include <Windows.h>
#include "sstream"
#include "iostream"
#include "List.h"
#include "Assoc_Arr.h"

int Assoc_arr::hashCode(const std::string& value)
{
    int res = 0;
    for (unsigned int i = 0; i < value.length(); i++)

```

```

        {
            res += value[i];
        }
        return res % 5;
    }
}

void Assoc_arr::add(const std::string& key, const std::string& value)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    while (p != nullptr)
    {
        if (p->key == key)
        {
            p->value = value;
            return;
        }
        p = p->next;
    }
    if (p == nullptr)
    {
        List::Node* temp = list[hashKey].first;
        list[hashKey].first = new List::Node(key, value);
        list[hashKey].first->next = temp;
        return;
    }
}

void Assoc_arr::del(const std::string& key)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    if (p == nullptr)
        return;
    if (p->key == key)
    {
        list[hashKey].first = p->next;
        delete p;
        return;
    }
    List::Node* temp = p;
    while (p && p->key != key)
    {
        temp = p;
        p = p->next;
    }
    if (p != nullptr)
    {
        temp->next = p->next;
        delete p;
    }
}

void Assoc_arr::write(HANDLE handle)
{
    for (int i = 0; i < 5; i++)
    {
        List::Node* mark = list[i].first;
        while (mark != nullptr)
        {
            size_t buf = mark->key.length();
            WriteFile(handle, &buf, sizeof size_t, 0, nullptr);
            WriteFile(handle, mark->key.c_str(), mark->key.length(), 0, nullptr);
            buf = mark->value.length();
            WriteFile(handle, &buf, sizeof size_t, 0, nullptr);
            WriteFile(handle, mark->value.c_str(), mark->value.length(), 0,
nullptr);
            mark = mark->next;
        }
    }
}

```

```

    }
}
void Assoc_arr::read(HANDLE handle)
{
    DWORD dword;
    size_t length;
    ReadFile(handle, &length, sizeof(size_t), &dword, nullptr);
    std::string key(length, '\\0');
    ReadFile(handle, (void*)key.c_str(), length, &dword, nullptr);
    ReadFile(handle, &length, sizeof(size_t), &dword, nullptr);
    std::string value(length, '\\0');
    ReadFile(handle, (void*)value.c_str(), length, &dword, nullptr);
    while (dword != 0)
    {
        add(key, value);
        ReadFile(handle, &length, sizeof(size_t), &dword, nullptr);
        key = std::string(length, '\\0');
        ReadFile(handle, (void*)key.c_str(), length, &dword, nullptr);
        ReadFile(handle, &length, sizeof(size_t), &dword, nullptr);
        value = std::string(length, '\\0');
        ReadFile(handle, (void*)value.c_str(), length, &dword, nullptr);
    }
}

```

lab5.cpp

```

#include <string>
#include <Windows.h>
#include <iostream>
#include "Assoc_Arr.h"

int main()
{
    Assoc_arr arr1;
    arr1.add("1", "a");
    arr1.add("2", "b");
    arr1.add("3", "c");
    arr1.add("4", "d");
    arr1.add("5", "e");

    HANDLE handle = CreateFileA("test1.txt", GENERIC_WRITE, FILE_SHARE_WRITE, nullptr,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, nullptr);
    arr1.write(handle);
    CloseHandle(handle);
    arr1.del("4");
    handle = CreateFileA("test1.txt", GENERIC_READ, 0, nullptr, OPEN_EXISTING, NULL,
    nullptr);

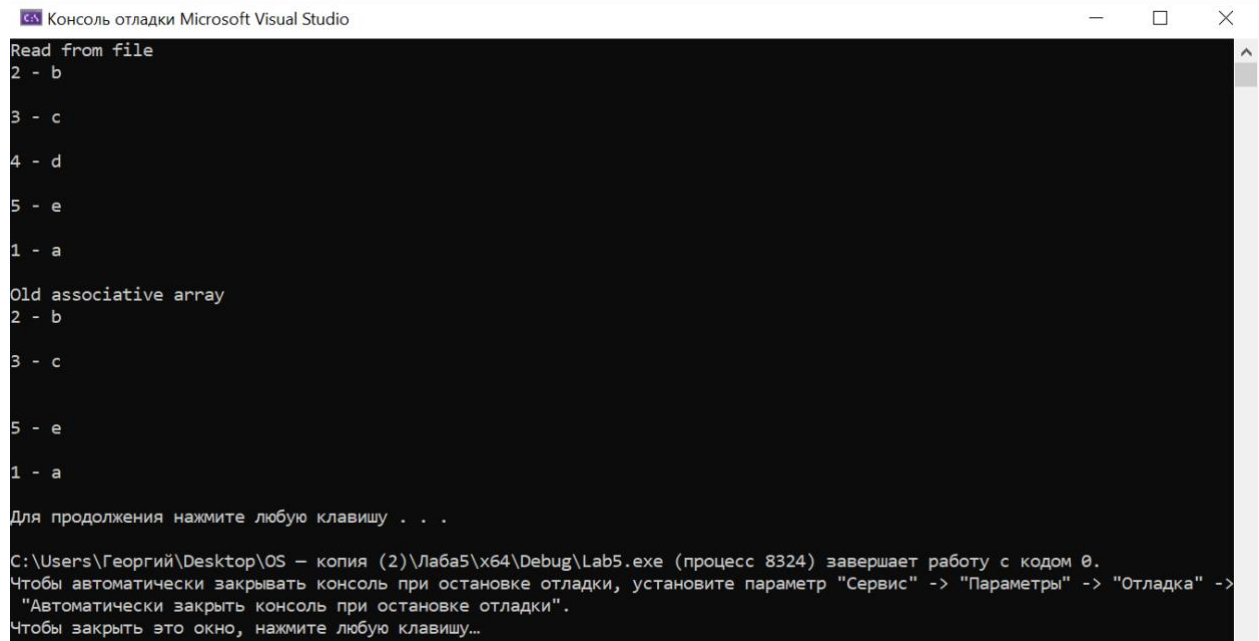
    Assoc_arr arr2;
    arr2.read(handle);
    CloseHandle(handle);

    printf("Read from file \n");
    std::cout << arr2;
    printf("Old associative array \n");
    std::cout << arr1;

    system("pause");
    return 0;
}

```

Результат выполнения программы:



Консоль отладки Microsoft Visual Studio

```
Read from file
2 - b
3 - c
4 - d
5 - e
1 - a

Old associative array
2 - b
3 - c
5 - e
1 - a

Для продолжения нажмите любую клавишу . . .

C:\Users\Георгий\Desktop\OS – копия (2)\Лаба5\х64\Debug\Lab5.exe (процесс 8324) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

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 хэш-таблицу, построенную по методу цепочек.

Листинг программы:

List.h

```
#pragma once
#include <ostream>
#include <string>

class __declspec(dllexport) List
{
public:
    struct Node {
        std::string key;
        std::string value;
        Node(const std::string& k, const std::string& v)
        {
            key = k;
            value = v;
            next = nullptr;
        }
        Node* next;
        ~Node() {}
    };
    Node* first;
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    List();
    ~List();
};

inline std::ostream& operator << (std::ostream& os, const List& list)
{
    auto* p = list.first;
    while (p != nullptr)
    {
        os << p->key << " - " << p->value << "\t\n";
        p = p->next;
    }
    return os;
}
```

List.cpp

```
#include "List.h"

List::List()
{
    first = nullptr;
}

List::~~List() {}

void List::add(const std::string& key, const std::string& value)
{
    if (!first)
    {
        first = new Node(key, value);
    }
}
```

```

    }
    else
    {
        Node* p = first;
        first = new Node(key, value);
        first->next = p;
    }
}

void List::del(const std::string& key)
{
    if (first->key == key)
    {
        Node* p = first;
        first->next = first->next->next;
    }
    else
    {
        Node* p = first;
        while (p->next && p->next->key != key)
        {
            p = p->next;
        }
        if (!p->next) return;
        Node* n = p->next;
        p->next = p->next->next;
        delete n;
    }
}

```

Assoc_arr.h

```

#pragma once
#include <string>
#include <ostream>
#include "List.h"

class __declspec(dllexport) Assoc_arr
{
public:
    List list[5];
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    static int hashCode(const std::string& value);
    void write(HANDLE structFile);
    void read(HANDLE structFile);
};

inline std::ostream& operator <<(std::ostream& os, const Assoc_arr& arr)
{
    for (int i = 0; i < 5; i++)
        os << arr.list[i] << "\t\n";
    return os;
}

```

Assoc_arr.cpp

```

#include <iostream>
#include <fstream>
#include <Windows.h>
#include "sstream"
#include "Assoc_arr.h"

int Assoc_arr::hashCode(const std::string& value)
{

```

```

        int res = 0;
        for (unsigned int i = 0; i < value.length(); i++)
        {
            res += value[i];
        }
        return res % 5;
    }

void Assoc_arr::add(const std::string& key, const std::string& value)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    while (p != nullptr)
    {
        if (p->key == key)
        {
            p->value = value;
            return;
        }
        p = p->next;
    }
    if (p == nullptr)
    {
        List::Node* temp = list[hashKey].first;
        list[hashKey].first = new List::Node(key, value);
        list[hashKey].first->next = temp;
        return;
    }
}

void Assoc_arr::del(const std::string& key)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    if (p == nullptr)
        return;
    if (p->key == key)
    {
        list[hashKey].first = p->next;
        delete p;
        return;
    }
    List::Node* temp = p;
    while (p && p->key != key)
    {
        temp = p;
        p = p->next;
    }
    if (p != nullptr)
    {
        temp->next = p->next;
        delete p;
    }
}

void Assoc_arr::write(HANDLE handle)
{
    DWORD dw = 0;
    for (int i = 0; i < 5; i++)
    {
        List::Node* mark = list[i].first;
        while (mark != nullptr)
        {
            size_t buf = mark->key.length();
            WriteFile(handle, &buf, sizeof size_t, &dw, nullptr);

```



```

        WriteFile(handle, mark->key.c_str(), mark->key.length(), &dw,
nullptr);
        buf = mark->value.length();
        WriteFile(handle, &buf, sizeof size_t, &dw, nullptr);
        WriteFile(handle, mark->value.c_str(), mark->value.length(), &dw,
nullptr);
        mark = mark->next;
    }
}

void Assoc_arr::read(HANDLE handle)
{
    DWORD dw;
    size_t length;
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string key(length, ' ');
    ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string value(length, ' ');
    ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    while (dw != 0)
    {
        add(key, value);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        key = std::string(length, ' ');
        ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        value = std::string(length, ' ');
        ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    }
}

extern "C"
{
    __declspec(dllexport) void add(Assoc_arr* arr, const std::string& key, const
std::string& value)
    {
        arr->add(key, value);
    }

    __declspec(dllexport) void writeToCout(Assoc_arr* arr)
    {
        std::cout << *arr;
    }
}

```

lab6.cpp

```

#include <iostream>
#include <string>
#include <fstream>
#include <Windows.h>
#include <tchar.h>
#include "Assoc_arr.h"
#include "C:\Users\Георгий\Desktop\OS\Laba6\Laba6\Assoc_arr.h"

//Неявное
int main1()
{
    Assoc_arr arr1;
    arr1.add("1", "a");
    arr1.add("2", "b");
    arr1.add("3", "c");
    arr1.add("4", "d");
    arr1.add("5", "e");
}

```

```

        arr1.del("4");
        std::cout << arr1;
        system("pause");
        return 0;
    }

    //Явное

    typedef void(*addFunction)(Assoc_arr*, const std::string&, const std::string&);
    typedef void(*writeFunction)(Assoc_arr*);

    int main2()
    {
        Assoc_arr arr1;
        arr1.add("1", "a");
        arr1.add("2", "b");
        arr1.add("3", "c");
        arr1.del("3");
        LPCWSTR buf = _T("Laba6.dll");
        HINSTANCE hDll = LoadLibrary(buf);
        if (hDll != NULL)
        {
            addFunction pfn;
            (FARPROC&)pfn = GetProcAddress(hDll, "add");
            pfn(&arr1, "4", "ttt");
            writeFunction p;
            (FARPROC&)p = GetProcAddress(hDll, "writeToCout");
            p(&arr1);
        }
        system("pause");
        return 0;
    }

    int main() {
        main1();
        main2();
    }

```

Результат выполнения программы:

```

Консоль отладки Microsoft Visual Studio
2 - b
3 - c
5 - e
1 - a
Для продолжения нажмите любую клавишу . . .
2 - b
4 - ttt
1 - a
Для продолжения нажмите любую клавишу . . .
C:\Users\Георгий\Desktop\OS - копия (2)\Лабa6\х64\Debug\Laba6.exe (процесс 2144) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрывать консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...

```

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 хэш-таблицу, построенную по методу цепочек.

Листинг программы:

List.h

```
#pragma once
#include <ostream>
#include <string>

class __declspec(dllexport) List
{
public:
    struct Node {
        std::string key;
        std::string value;
        Node(const std::string& k, const std::string& v)
        {
            key = k;
            value = v;
            next = nullptr;
        }
        Node* next;
        ~Node() {}
    };
    Node* first;
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    List();
    ~List();
};

inline std::ostream& operator << (std::ostream& os, const List& list)
{
    auto* p = list.first;
    while (p != nullptr)
    {
        os << p->key << " - " << p->value << "\t\n";
        p = p->next;
    }
    return os;
}
```

List.cpp

```
#include "List.h"

List::List()
{
    first = nullptr;
}
```

```

}
List :: ~List() {}

void List::add(const std::string& key, const std::string& value)
{
    if (!first)
    {
        first = new Node(key, value);
    }
    else
    {
        Node* p = first;
        first = new Node(key, value);
        first->next = p;
    }
}

void List::del(const std::string& key)
{
    if (first->key == key)
    {
        Node* p = first;
        first->next = first->next->next;
    }
    else
    {
        Node* p = first;
        while (p->next && p->next->key != key)
        {
            p = p->next;
        }
        if (!p->next) return;
        Node* n = p->next;
        p->next = p->next->next;
        delete n;
    }
}

```

Assoc_arr.h

```

#pragma once
#include "List.h"

class Assoc_arr
{
public:
    List list[5];
    Assoc_arr();
    HANDLE hThread;
    HANDLE hEventBufEmpty;
    HANDLE hEventBufFull;
    std::string buf1;
    std::string buf2;
    void threadAdd(const std::string& key, const std::string& value);
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    static int hashCode(const std::string& value);
    void write(HANDLE structFile);
    void read(HANDLE structFile);
};

inline std::ostream& operator <<(std::ostream& os, const Assoc_arr& arr)
{
    for (int i = 0; i < 5; i++)
        os << arr.list[i] << "\t\n";
}

```

```

        return os;
    }

```

Assoc_arr.cpp

```

#include <iostream>
#include <fstream>
#include <Windows.h>
#include "sstream"
#include "iostream"
#include "List.h"
#include "Assoc_Arr.h"

Assoc_arr::Assoc_arr()
{
    buf1 = "";
    buf2 = "";
    hEventBufEmpty = CreateEvent(NULL, FALSE, TRUE, NULL);
    hEventBufFull = CreateEvent(NULL, FALSE, FALSE, NULL);
    hThread = CreateThread(NULL, 0, [](PVOID s)->DWORD
    {
        Assoc_arr* arr = (Assoc_arr*)s;
        while (true)
        {
            WaitForSingleObject(arr->hEventBufFull, 100);
            arr->add(arr->buf1, arr->buf2);
            SetEvent(arr->hEventBufEmpty);
        }
        return 0;
    }, this, 0, 0);
}

int Assoc_arr::hashCode(const std::string& value)
{
    int res = 0;
    for (unsigned int i = 0; i < value.length(); i++)
    {
        res += value[i];
    }
    return res % 5;
}

void Assoc_arr::threadAdd(const std::string& key, const std::string& value)
{
    WaitForSingleObject(hEventBufEmpty, 100);
    buf1 = key;
    buf2 = value;
    SetEvent(hEventBufFull);
}

void Assoc_arr::add(const std::string& key, const std::string& value)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    while (p != nullptr)
    {
        if (p->key == key)
        {
            p->value = value;
            return;
        }
        p = p->next;
    }
    if (p == nullptr)
    {

```

```

        List::Node* temp = list[hashKey].first;
        list[hashKey].first = new List::Node(key, value);
        list[hashKey].first->next = temp;
        return;
    }
}

void Assoc_arr::del(const std::string& key)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    if (p == nullptr)
        return;
    if (p->key == key)
    {
        list[hashKey].first = p->next;
        delete p;
        return;
    }
    List::Node* temp = p;
    while (p && p->key != key)
    {
        temp = p;
        p = p->next;
    }
    if (p != nullptr)
    {
        temp->next = p->next;
        delete p;
    }
}

void Assoc_arr::write(HANDLE handle)
{
    DWORD dw;
    for (int i = 0; i < 5; i++)
    {
        List::Node* p = list[i].first;
        while (p != nullptr)
        {
            size_t buf = p->key.length();
            WriteFile(handle, &buf, sizeof(size_t), &dw, nullptr);
            WriteFile(handle, p->key.c_str(), p->key.length(), &dw, nullptr);
            buf = p->value.length();
            WriteFile(handle, &buf, sizeof(size_t), &dw, nullptr);
            WriteFile(handle, p->value.c_str(), p->value.length(), &dw, nullptr);
            p = p->next;
        }
    }
}

void Assoc_arr::read(HANDLE handle)
{
    DWORD dw;
    size_t length = 0;
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string key(length, ' ');
    ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string value(length, ' ');
    ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    while (dw != 0)
    {
        add(key, value);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        key = std::string(length, ' ');
    }
}

```

```

        ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        value = std::string(length, ' ');
        ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    }
}

```

Lab7.cpp

```

#include <iostream>
#include <string>
#include <fstream>
#include <Windows.h>
#include <tchar.h>
#include "Assoc_arr.h"

int main()
{
    Assoc_arr arr1;
    arr1.threadAdd("1", "a");
    arr1.threadAdd("2", "b");
    arr1.threadAdd("3", "c");
    arr1.threadAdd("4", "d");
    arr1.threadAdd("5", "e");

    HANDLE handle = CreateFileA("test1.txt", GENERIC_WRITE, FILE_SHARE_WRITE, nullptr,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, nullptr);
    arr1.write(handle);
    CloseHandle(handle);
    handle = CreateFileA("test1.txt", GENERIC_READ, 0, nullptr, OPEN_EXISTING, NULL,
    nullptr);

    arr1.del("4");

    Assoc_arr arr2;
    arr2.read(handle);
    CloseHandle(handle);

    printf("Read from file \n");
    std::cout << arr2;
    printf("Old associative array \n");
    std::cout << arr1;

    system("pause");
    return 0;
}

```

Результат выполнения программы:

```
Консоль отладки Microsoft Visual Studio
Read from file
2 - b

3 - c

4 - d

1 - a

Old associative array
2 - b

3 - c

5 - e

1 - a

Для продолжения нажмите любую клавишу . . .

C:\Users\Георгий\Desktop\05 – копия (2)\Лабa7\х64\Debug\Laba7.exe (процесс 5388) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```


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 хэш-таблицу, построенную по методу цепочек.

Листинг программы:

List.h

```
#pragma once
#include <iostream>
#include <string>

class List
{
public:
    struct Node {
        std::string key;
        std::string value;
        Node(const std::string& k, const std::string& v)
        {
            key = k;
            value = v;
            next = nullptr;
        }
        Node* next;
        ~Node() {}
    };
    Node* first;
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    List();
    ~List();
};

inline std::ostream& operator << (std::ostream& os, const List& list)
{
    auto* p = list.first;
    while (p != nullptr)
    {
        os << p->key << " - " << p->value << "\t\n";
        p = p->next;
    }
    return os;
}
```

List.cpp

```
#include "List.h"

List::List()
{
    first = nullptr;
}

List::~~List() {}

void List::add(const std::string& key, const std::string& value)
{
    if (!first)
    {
        first = new Node(key, value);
    }
    else
```

```

    {
        Node* p = first;
        first = new Node(key, value);
        first->next = p;
    }
}

void List::del(const std::string& key)
{
    if (first->key == key)
    {
        Node* p = first;
        first->next = first->next->next;
    }
    else
    {
        Node* p = first;
        while (p->next && p->next->key != key)
        {
            p = p->next;
        }
        if (!p->next) return;
        Node* n = p->next;
        p->next = p->next->next;
        delete n;
    }
}

```

Assoc_arr.h

```

#pragma once
#include "List.h"

class Assoc_arr
{
public:
    List list[5];
    Assoc_arr();
    HANDLE hThread;
    HANDLE hEventBufEmpty;
    HANDLE hEventBufFull;
    std::string buf1;
    std::string buf2;
    void asyncAdd(const std::string& key, const std::string& value);
    void add(const std::string& key, const std::string& value);
    void del(const std::string& key);
    static int hashCode(const std::string& value);
    void write(HANDLE structFile);
    void read(HANDLE structFile);
};

inline std::ostream& operator <<(std::ostream& os, const Assoc_arr& arr)
{
    for (int i = 0; i < 5; i++)
        os << arr.list[i] << "\t\n";
    return os;
}

```

Assoc_arr.cpp

```

#include <iostream>
#include <fstream>
#include <Windows.h>
#include "sstream"
#include "iostream"

```

```

#include "List.h"
#include "Assoc_Arr.h"

Assoc_arr::Assoc_arr()
{
    DWORD id;
    buf1 = "";
    buf2 = "";
    hEventBufEmpty = CreateEvent(NULL, FALSE, TRUE, NULL);
    hEventBufFull = CreateEvent(NULL, FALSE, FALSE, NULL);
    hThread = CreateThread(NULL, 0, [](PVOID s)->DWORD
    {
        Assoc_arr* arr = (Assoc_arr*)s;
        while (true)
        {
            WaitForSingleObject(arr->hEventBufFull, INFINITE);
            arr->add(arr->buf1, arr->buf2);
            SetEvent(arr->hEventBufEmpty);
        }
        return 0;
    }, this, 0, &id);
}

int Assoc_arr::hashCode(const std::string& value)
{
    int res = 0;
    for (unsigned int i = 0; i < value.length(); i++)
    {
        res += value[i];
    }
    return res % 5;
}

void Assoc_arr::asyncAdd(const std::string& key, const std::string& value)
{
    WaitForSingleObject(hEventBufEmpty, INFINITE);
    buf1 = key;
    buf2 = value;
    SetEvent(hEventBufFull);
}

void Assoc_arr::add(const std::string& key, const std::string& value)
{
    int hashKey = hashCode(key);
    List::Node* p = list[hashKey].first;
    while (p != nullptr)
    {
        if (p->key == key)
        {
            p->value = value;
            return;
        }
        p = p->next;
    }
    if (p == nullptr)
    {
        List::Node* temp = list[hashKey].first;
        list[hashKey].first = new List::Node(key, value);
        list[hashKey].first->next = temp;
        return;
    }
}

void Assoc_arr::del(const std::string& key)
{

```

```

int hashKey = hashCode(key);
List::Node* p = list[hashKey].first;
if (p == nullptr)
    return;
if (p->key == key)
{
    list[hashKey].first = p->next;
    delete p;
    return;
}
List::Node* temp = p;
while (p && p->key != key)
{
    temp = p;
    p = p->next;
}
if (p != nullptr)
{
    temp->next = p->next;
    delete p;
}
}

void Assoc_arr::write(HANDLE handle)
{
    DWORD dw;
    for (int i = 0; i < 5; i++)
    {
        List::Node* p = list[i].first;
        while (p != nullptr)
        {
            size_t buf = p->key.length();
            WriteFile(handle, &buf, sizeof size_t, &dw, nullptr);
            WriteFile(handle, p->key.c_str(), p->key.length(), &dw, nullptr);
            buf = p->value.length();
            WriteFile(handle, &buf, sizeof size_t, &dw, nullptr);
            WriteFile(handle, p->value.c_str(), p->value.length(), &dw, nullptr);
            p = p->next;
        }
    }
}

void Assoc_arr::read(HANDLE handle)
{
    DWORD dw;
    size_t length = 5;
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string key(length, ' ');
    ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
    ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
    std::string value(length, ' ');
    ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    while (dw != 0)
    {
        add(key, value);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        key = std::string(length, ' ');
        ReadFile(handle, (void*)key.c_str(), length, &dw, nullptr);
        ReadFile(handle, &length, sizeof(size_t), &dw, nullptr);
        value = std::string(length, ' ');
        ReadFile(handle, (void*)value.c_str(), length, &dw, nullptr);
    }
}

```

lab8.cpp

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include <Windows.h>
#include <tchar.h>
#include "Assoc_arr.h"

#define DATA_BUFFER_SIZE 256

const TCHAR FileInMemoryName[] = TEXT("file1");
const TCHAR FileInMemoryName2[] = TEXT("file2");
HANDLE hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0,
DATA_BUFFER_SIZE, FileInMemoryName);
HANDLE hMapFile2 = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0,
DATA_BUFFER_SIZE, FileInMemoryName2);

int main1() {
    Assoc_arr arr;
    auto putBufKey = (char*)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0,
DATA_BUFFER_SIZE);
    auto putBufValue = (char*)MapViewOfFile(hMapFile2, FILE_MAP_ALL_ACCESS, 0, 0,
DATA_BUFFER_SIZE);
    std::string value((int)*putBufValue, ' ');
    std::string key((int)*putBufKey, ' ');
    CopyMemory((void*)key.c_str(), (void*)(putBufKey + 1), (int)*putBufKey);
    CopyMemory((void*)value.c_str(), (void*)(putBufValue + 1), (int)*putBufValue);
    arr.add(key, value);
    std::cout << arr;
    system("pause");
    return 0;
}

int main2()
{
    std::string key = "12345";
    std::string value = "abcde";
    auto putBufKey = (char*)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0,
DATA_BUFFER_SIZE);
    auto putBufValue = (char*)MapViewOfFile(hMapFile2, FILE_MAP_ALL_ACCESS, 0, 0,
DATA_BUFFER_SIZE);
    *putBufKey = (char)key.length();
    *putBufValue = (char)value.length();
    CopyMemory((void*)(putBufKey + 1), key.c_str(), key.length());
    CopyMemory((void*)(putBufValue + 1), value.c_str(), value.length());

    HANDLE structfile;
    STARTUPINFO startupInfo;
    PROCESS_INFORMATION processInfo;
    ZeroMemory(&startupInfo, sizeof(STARTUPINFO));
    startupInfo.cb = sizeof(STARTUPINFO);
    BOOL RET =
CreateProcess(L"C:\\Users\\Георгий\\Desktop\\OS\\Laba8\\Debug\\Laba8.exe",
NULL, NULL, NULL, FALSE, CREATE_NEW_CONSOLE, NULL, NULL, &startupInfo,
&processInfo);

    WaitForSingleObject(processInfo.hProcess, INFINITE);

    CloseHandle(processInfo.hProcess);
    system("pause");
    return 0;
}
```

```
int main() {  
    main1();  
    main2();  
    return 0;  
}
```

Результат выполнения программы:

