Пермский национальный исследовательский политехнический университет

Лабораторные работы по

«Основам алгоритмизации и программированию»

за 1 семестр

вариант №4

Выполнил:

студент группы РИС-21-2бз (РИС-21-1бзу)

Дерябин Кирилл Николаевич

Проверила:

доцент кафедры ИТАС

Полякова О.А.

# Лабораторная работа № 7 7.1. "Перегрузка функций в Си++"

**Цель:** Знакомство с организацией перегруженных функций в Си++.

**1. Постановка задачи для конкретного варианта.**

Написать перегруженные функции и основную программу, которая их вызывает.

**2. Исходные данные.**

а) для вычитания целых чисел

б) для вычитания комплексных чисел

**3. Текст программы.**

#include <stdio.h>

int sub(int a, int b)

{

return a - b;

}

double sub(double ai, double bi, double a, double b)

{

return (a - b) + (ai - bi);

}

int main()

{

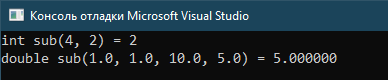
printf("int sub(4, 2) = %d\n", sub(4, 2));

printf("double sub(1.0, 1.0, 10.0, 5.0) = %lf\n", sub(1.0, 1.0, 10.0, 5.0));

return 0;

}

**4. Результаты выполнения программы.**

****

# 7.2. "Функции с переменным числом параметров"

**Цель:** Знакомство с организацией функций с переменным числом параметров.

**1. Постановка задачи для конкретного варианта.**

Решить указанную в варианте задачу, используя функции с переменным числом параметров.

**2. Исходные данные.**

Написать функцию sum с переменным числом параметров, которая находит сумму чисел типа int по формуле:

S=a1\*a2+a3\*a4+a5\*a6+. . . . .

Написать вызывающую функцию main, которая обращается к функции sum не менее трех раз с количеством параметров 8, 10, 12.

**3. Текст программы.**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

long long sum(int nparams, ...)

{

long long S = 0;

int \*pparam = (&nparams) + 1;

for (int i = 0; i < nparams; i += 2)

S += pparam[i] \* pparam[i + 1];

return S;

}

void gen\_parameters(int \*ptr)

{

srand(clock());

for (int i = 0; i < 12; i++)

ptr[i] = rand() % 12;

}

int main()

{

int params[12];

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

gen\_parameters(params);

printf("%d: Parameters 8, 10, 12: ( ( ( %d %d %d %d %d %d %d %d ) %d %d ) %d %d )\n", i + 1,

params[0], params[1], params[2], params[3], params[4], params[5],

params[6], params[7], params[8], params[9], params[10], params[11]);

printf("%d: sum 8 parameters: %lld\n", i + 1,

sum(8, params[0], params[1], params[2], params[3], params[4], params[5], params[6], params[7]));

printf("%d: sum 10 parameters: %lld\n", i + 1,

sum(10, params[0], params[1], params[2], params[3], params[4], params[5], params[6], params[7], params[8], params[9]));

printf("%d: sum 12 parameters: %lld\n", i + 1,

sum(12, params[0], params[1], params[2], params[3], params[4], params[5], params[6], params[7], params[8], params[9], params[10], params[11]));

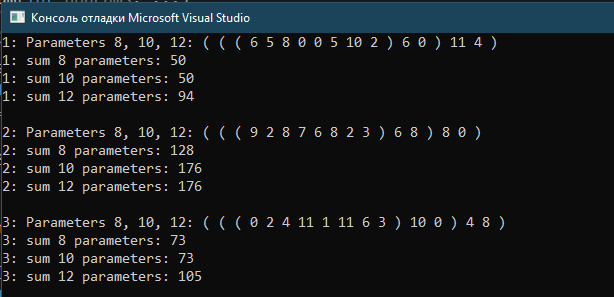
printf("\n");

}

return 0;

}

**4. Результаты выполнения программы.**

****

**Анализ выполнения работы**

В данной работе для реализации переменного числа параметров были использованы указатели, а также операторы адреса и разыменования для работы с ними.

Указатель — это переменная, значением которой является адрес в памяти. Указатели объявляются точно так же, как и обычные переменные, только со звёздочкой перед идентификатором.

Оператор адреса & позволяет узнать, какой адрес памяти присвоен определенной переменной.

Оператор разыменования \* позволяет получить значение по указанному адресу.

# Лабораторная работа №8 "Блоковый ввод-вывод"

**Цель:** Работа с двоичными файлами, организация ввода-вывода структурированной информации и ее хранение на внешних носителях.

**1. Постановка задачи.**

Сформировать двоичный файл из элементов, заданной в варианте структуры, распечатать его содержимое, выполнить удаление и добавление элементов в соответствии со своим.

Структура "Человек":

* фамилия, имя, отчество;
* домашний адрес;
* номер телефона;
* возраст.

Удалить все элементы с заданным возрастом, добавить элемент после элемента с заданным номером.

**2. Описание используемых типов данных.**

struct header

{

int count;

};

Структура header используется для хранения количества записей в файле. При чтении файла сначала читается заголовок и далее исходя из количества записей выполняется чтение самих записей.

struct human\_s

{

char firstname[32];

char lastname[32];

char patronymic[32];

char address[256];

char phonenumber[32];

int age;

};

Структура human\_s используется для хранения информации одного человека. В программе для хранения загруженных пользователей используется статический размер разметом 20 \* sizeof(human\_s).

**3. Текст функций**

Код записи пользователей в файл для дальнейших модификаций

void WriteInitData()

{

FILE \*fp = fopen(DATABASE, "wb");

if (!fp) {

printf("Failed to open file!\n");

return;

}

fwrite(&total\_objects, sizeof(int), 1, fp);

if (fwrite(&humans, sizeof(humans[0]), total\_objects, fp) != total\_objects)

printf("Error writing data to file!\n");

fclose(fp);

}

Код вывода данных файла

void PrintData()

{

for (int i = 0; i < current\_users; i++)

printf("%d: %s %s %s %s %s %d\n",

i,

humans\_data[i].lastname,

humans\_data[i].firstname,

humans\_data[i].patronymic,

humans\_data[i].address,

humans\_data[i].phonenumber,

humans\_data[i].age);

}

Код добавления записи

bool InsertHuman(human\_s \*p\_human\_data, int index)

{

if (current\_users == max\_data)

return false;

for (int i = index; i < max\_data - 1; i += 2)

humans\_data[i + 1] = humans\_data[i];

humans\_data[index] = \*p\_human\_data;

if (current\_users < max\_data) {

current\_users++;

printf("current\_users++;\n");

}

return true;

}

Код поиска и удаления пользователя по возрасту

bool DeleteHumanByAge(int age)

{

for (int i = 0; i < max\_data; i++) {

if (humans\_data[i].age == age) {

for (int j = i; j < max\_data - 1; j++) {

humans\_data[j] = humans\_data[j + 1];

}

current\_users--;

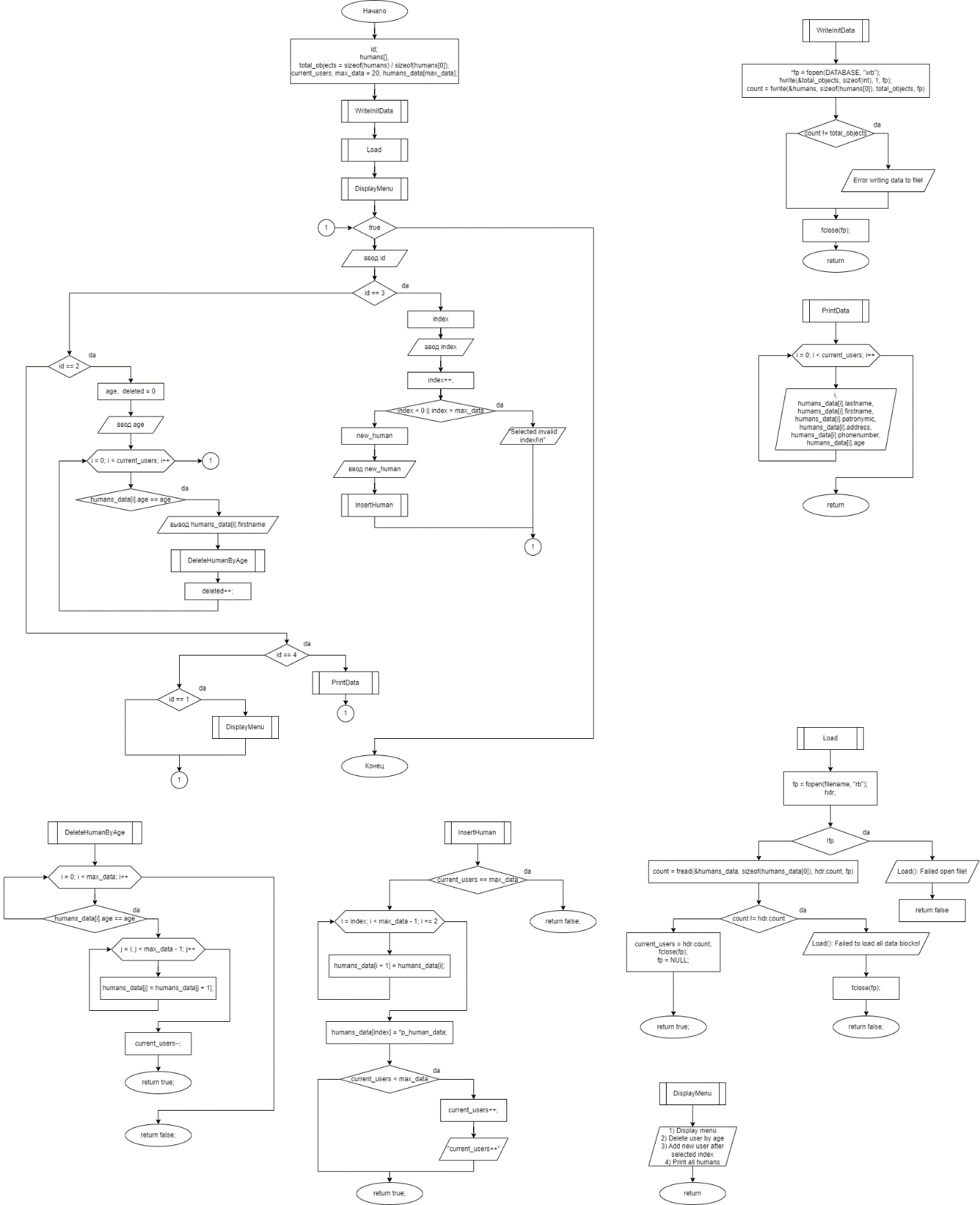
return true;

}

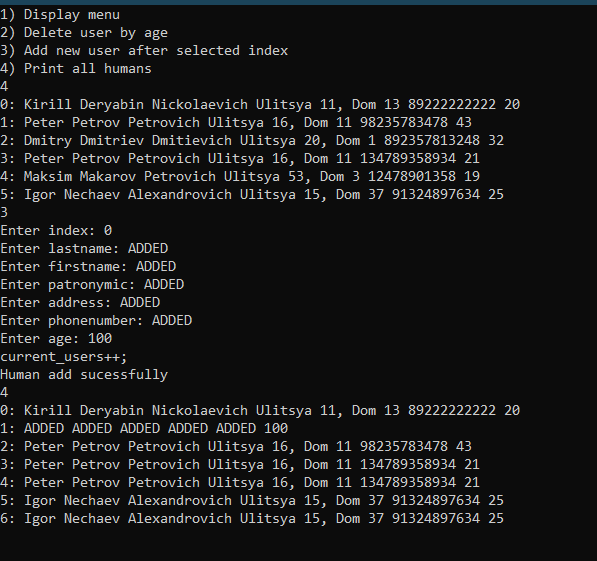
}

return false;

}

****

**4. Результат решения конкретного варианта.**

****

**Анализ выполнения работы**

В данной работе для удобного хранения данных о каждом человеке была использована структура. Для хранения данных людей был использован массив структур. Каждая структура содержит в себе поля имени, фамилии, отчества, адреса, номера телефона и возраста. Для хранения данных использован статический массив. Данные задаются при старте программы и записываются в файл. Далее читаются и загружаются в память.

**Лабораторная работа № 9**

**"Строковый ввод-вывод"**

**Цель**: Работа с текстовыми файлами, ввод-вывод текстовой информации и ее хранение на внешних носителях.

**1. Постановка задачи.**

1. Создать текстовый файл F1 не менее чем из 10 строк и записать в него информацию
2. Выполнить задание.

**2. Текст функций**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void create\_file\_with\_text(const char \*filename)

{

FILE \*fp = fopen(filename, "wt");

if (!fp) {

printf("Failed to open file!\n");

return;

}

fputs(<глобальный массив с текстом>, fp);

fclose(fp);

}

int wordlen(const char \*str)

{

int len = 0;

while (\*str) {

if (!ispunct(\*str))

len++;

str++;

}

return len;

}

int main()

{

create\_file\_with\_text("F1.txt");

FILE \*fpfrom, \*fpto;

if (!(fpfrom = fopen("F1.txt", "rt"))) {

printf("Failed open file F1.txt!\n");

return 1;

}

if(!(fpto = fopen("F2.txt", "w+"))) {

printf("Failed open file F2.txt!\n");

return 1;

}

int lineidx = 1;

char buffer[512];

while (!feof(fpfrom) && !ferror(fpfrom)) {

fgets(buffer, sizeof(buffer), fpfrom);

if (lineidx >= 4)

fputs(buffer, fpto);

lineidx++;

}

fclose(fpfrom);

rewind(fpto);

while (!feof(fpto) && !ferror(fpto))

fgets(buffer, sizeof(buffer), fpto);

int lastword\_len;

char \*ptok = strtok(buffer, " ");

while (ptok) {

lastword\_len = wordlen(ptok);

ptok = strtok(NULL, " ");

}

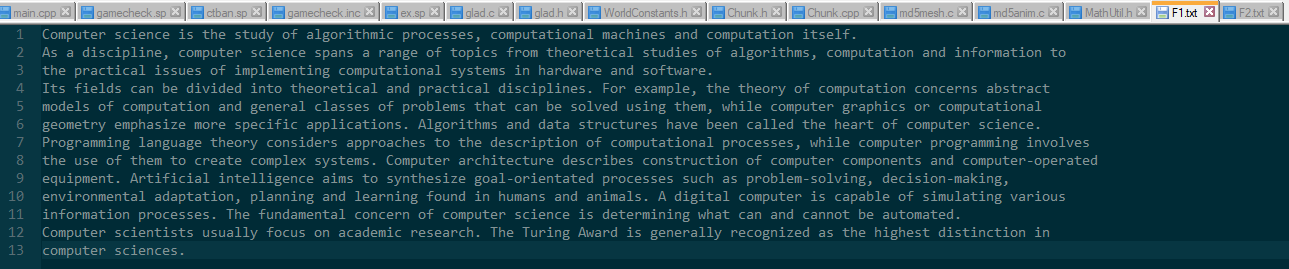
printf("%d\n", lastword\_len);

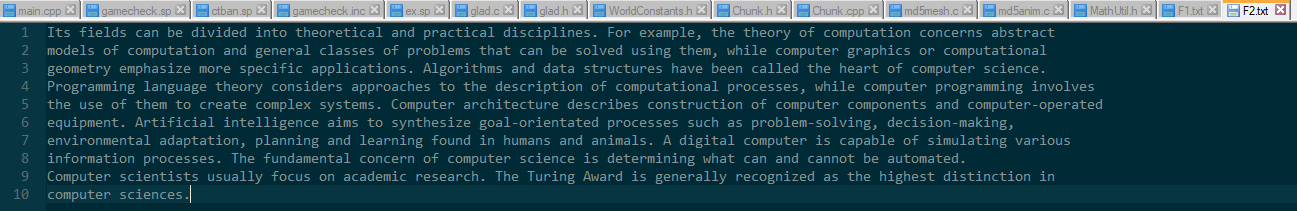
fclose(fpto);

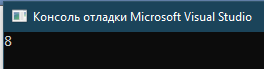
return 0;

}

**3. Результат решения конкретного варианта.**

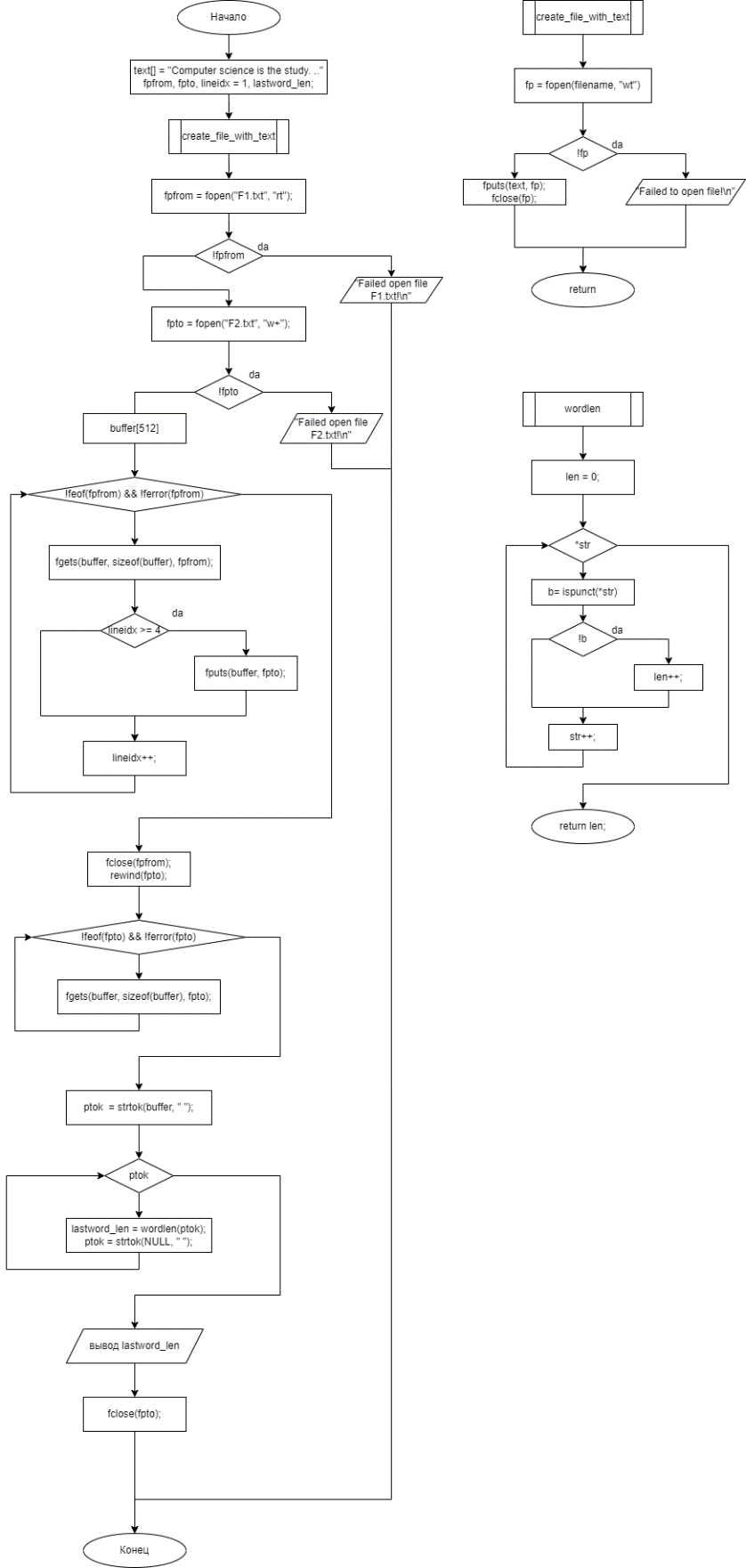
****

****

****

**Анализ выполнения работы**

Для работы с файлами была использована функция библиотеки stdio.h fopen. Первым параметром принимает имя файла который требуется создать или открыть, а вторым параметром функция принимает режим работы с файлом. Если файл требуется прочитать, вторым параметром указывается “r”. Так же возможны различные совмещения режимов работы с файлом. Чтобы записать в файл строку, используется функция fputs. Первым параметром передается адрес строки которую нужно скопировать в файл. Копирование символов производится до терминального нуля '\0'. Чтобы прочитать строку из файла, используется функция fgets. Так же, при чтении из файла необходимо проверять, достигнут ли конец файла и не произошло ли ошибок при чтении. Проверить это можно с помощью функций feof и ferror. Если не конец файла и нет ошибок, можно продолжать чтение. По завершению работы с файлом, требуется закрыть файл с помощью функции fclose. Проверка знаков пунктуации производится с помощью функции ispunkt. Используется данная функция в функции подсчитывающей длину слова wordlen. Функция create\_file\_with\_text создает файл с текстом. Так как функция fopen считается небезопасной, для того чтобы разрешить её использование, был определен макрос \_CRT\_SECURE\_NO\_WARNINGS перед подключением всех заголовочных файлов.

****

**Лабораторная работа № 10**

**"Динамические массивы"**

**Цель:** Организация динамических массивов.

**1. Постановка задачи.**

Написать программу, в которой создаются динамические массивы и выполнить их обработку в соответствии со своим вариантом.

**2. Код программы**

//lab10.cpp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <assert.h>

typedef unsigned int uint;

typedef unsigned char uchar;

template <typename \_type>

class DynamicHeap

{

public:

DynamicHeap() : m\_pData(nullptr), m\_nSize(0), m\_nOffset(0) {}

DynamicHeap(int size) : m\_pData(nullptr), m\_nSize((size > 0) ? size : 1), m\_nOffset(0) {

Realloc(m\_nSize);

assert(m\_pData);

}

~DynamicHeap() {

if (m\_pData) free(m\_pData);

}

bool DeleteElement(int index) {

for (int i = index; i < m\_nSize - 1; i++)

m\_pData[i] = m\_pData[i + 1];

m\_nSize--;

if (m\_nOffset >= m\_nSize)

m\_nOffset--;

return true;

}

void Push(\_type elem) {

CheckAndAutorealloc();

m\_pData[m\_nOffset] = elem;

m\_nOffset++;

}

int Size() { return m\_nSize - 1; }

int Offset() { return m\_nOffset; }

\_type operator[](int ind) {

assert(ind < m\_nSize);

return m\_pData[ind];

}

void AddAt(int index, \_type \*elems, int count) {

int needed\_size = m\_nSize + count;

if (needed\_size >= m\_nSize) {

m\_nSize = needed\_size;

Realloc(needed\_size);

}

for (int i = Size() - count; i >= index; i--)

m\_pData[i + count] = m\_pData[i];

memcpy(&m\_pData[index], elems, sizeof(\_type) \* count);

}

private:

inline bool Realloc(int newsize) {

m\_pData = (\_type \*)realloc(m\_pData, newsize \* sizeof(\_type));

assert(m\_pData);

return !!m\_pData;

}

inline void CheckAndAutorealloc() {

if ((m\_nOffset + 1) == m\_nSize) {

m\_nSize++;

Realloc(m\_nSize);

}

}

\_type \*m\_pData;

uint m\_nSize;

uint m\_nOffset;

};

int main()

{

DynamicHeap<int> heap(10);

for (int i = 0; i < 10; i++)

heap.Push(i);

heap.DeleteElement(0);

int newelems[] = { 1, 1, 1, 1, 1 };

heap.AddAt(2, newelems, 5);

for (int i = 0; i < heap.Size(); i++)

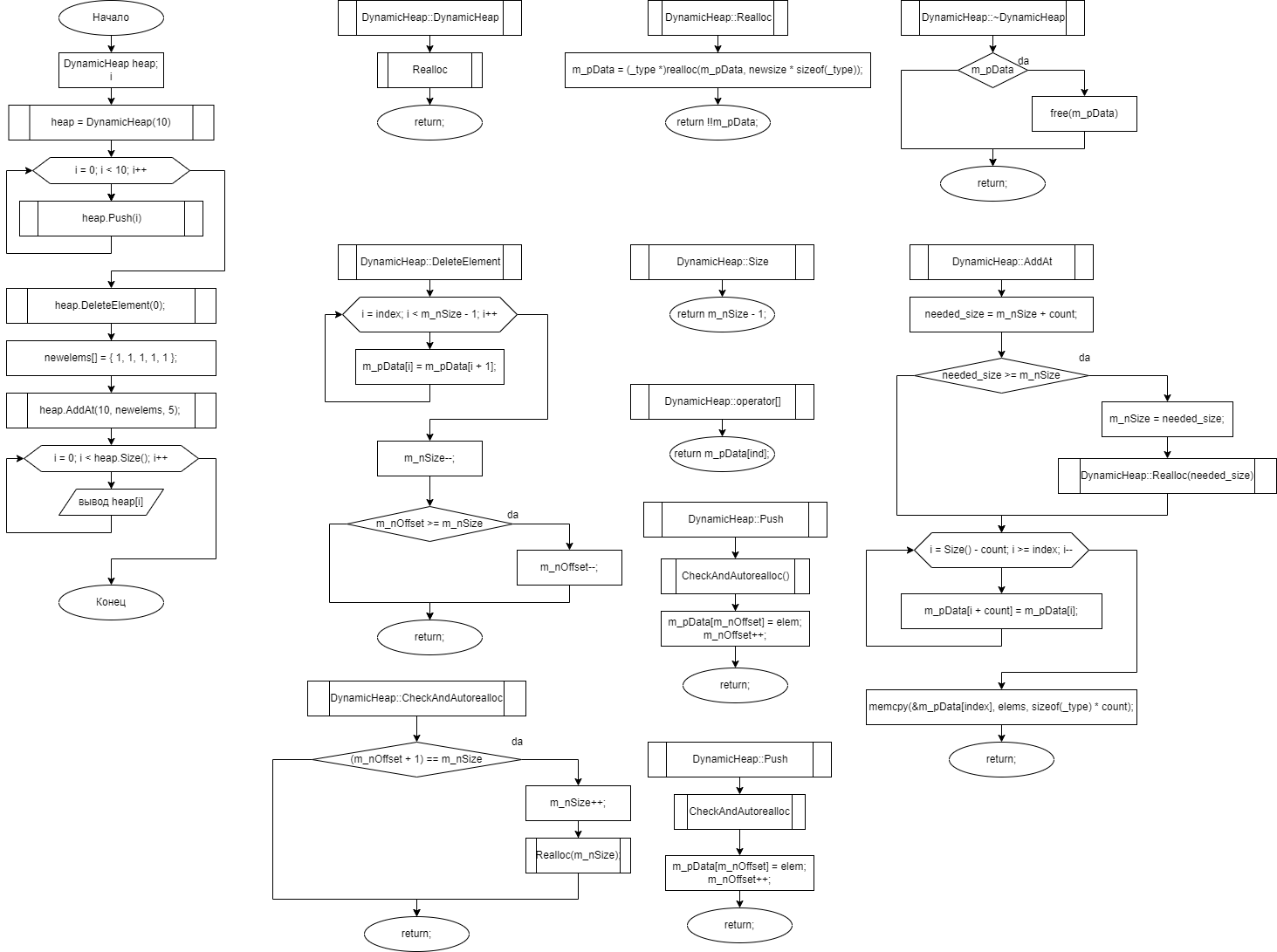
printf("%d\n", heap[i]);

return 0;

}

**3. Результаты выполнения работы.**





**Анализ выполнения работы**

Для выделения памяти динамически, в программе была использована функция realloc. Для удобной работы с функциями автоматически перераспределяемой памяти, был использован класс с шаблоном, в который передается имя типа данных, используемого далее внутри класса. При вызове конструктора, указатель на блок памяти инициализируется нулем, размер устанавливается равным одному элементу, если размер не был указан, иначе размер устанавливается равным тому, который был задан в конструкторе при объявлении экземпляра класса. Все утверждения проверяются на верность с помощью макроса assert. При компиляции программы в Release, макросы assert игнорируются. В деструкторе, при условии что память ранее была уже выделена (m\_pData != nullptr), вызывается функция free, которая освобождает выделенную память. Inline метод Realloc определенная в классе, занимается перераспределением памяти со встроенным assert, и возвратом bool значения в случае если возвращаемый адрес блока памяти равен нулю. Поле m\_pData хранит адрес блока памяти, m\_nSize хранит размер текущего блока памяти, m\_nOffset хранит положение в текущем блоке памяти, измеряемое в элементах, а не в байтах. Метод класса Push, занимается добавлением в конец нашего динамического массива одного элемента. В свою очередь он вызывает метод CheckAndRealloc, который проверяет, не равна ли позиция в буфере его размеру, и если равна, то увеличивает размер буфера на единицу и перераспределяет память. Чтобы исключить очень частые перераспределения памяти, можно выделять память с запасом, прибавляя к размеру не 1 а количество резервируемых элементов. Метод DeleteElement удаляет указанный элемент и сдвигает влево все последующие лежащие в массиве элементы. Для копирования элементов используется функция memcpy, заголовочного файла string.h.