Отчет по лабораторной работе № 1 «Введение в С. Microsoft Visual Studio»			
дата	Оценка (max 3)	Бонус за сложность	подпись

Цели работы:

-Ознакомление с базовыми возможностями и средствами языка С

Задачи работы:

-знакомство с основами С

-разработка примеров простейших программ на С

Краткий конспект теоретической части (ответы на контрольные вопросы)

Функция main() - это начальная точка выполнения для всех программ на языках С и
С++
Функция prinf позволяет выводить информацию на экран при программировании в
консольном режиме
Цикл for - цикл с постусловием — цикл, в котором условие проверяется после
выполнения тела цикла.

Задание 1

Запустите программу “Hello, World!”.

Исходный код

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Результат выполнения

```
gcc 1_1.c
./a.out
Hello, World!
```

Задание 2

Выведите следующую картинку с помощи функции printf:

****

* <Ваше имя> *

****

Исходный код

```
#include <stdio.h>

int main() {
    printf("**********\n");
    printf("* <Ваше имя>  *\n");
    printf("**********\n");
}
```

Результат выполнения

```
**********
* <Ваше имя>  *
**********
```

Задание 3

Напишите программу, печатающую таблицу умножения для чисел от 0 до 9 в десятичной системе счисления.

Исходный код

```
#include <stdio.h>

int main() {
    for (int i = 1; i < 10; i++) {
        for (int j = 1; j < 10; j++) {
            printf("%3d", i * j);
        }
        printf("\n");
    }
}
```

Результат выполнения

```
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  4  8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81
```

Контрольные вопросы

1. Функция main?
2. Функция printf?
3. Цикл for?

Отчет по лабораторной работе № 2 «Основы С»			
дата	Оценка (max 3)	Бонус за сложность	подпись

Цели работы:

-Ознакомление с базовыми возможностями и средствами языка С

Задачи работы:

-знакомство с основами С

-разработка примеров простейших программ на С

Краткий конспект теоретической части (ответы на контрольные вопросы)

Процесс создания программного обеспечения: структура, согласно которой построена
разработка программного обеспечения (ПО)
Компиляция - трансляция программы, составленной на исходном языке высокого уровня,
в эквивалентную программу на низкоуровневом языке, близком машинному коду
(абсолютный код, объектный модуль, иногда на язык ассемблера), выполняемая
компилятором.
Переменные - в языках программирования именованная часть памяти, в которую
могут помещаться разные значения переменной
Функции - отдельная система (подсистема, подпрограмма), на вход которой
поступают управляющие воздействия в виде значений аргументов. На выходе системы
получаем результат выполнения программы, который может быть как скалярной
величиной, так и векторным значением.
Константы - некоторая величина, не изменяющая своё значение в рамках
рассматриваемого процесса.
Определение и объявление: При объявлении переменной, функции или даже класса все,
что вы делаете, это говорите компилятору, что есть что-то с определенным именем и
определенного типа. Компилятор может обрабатывать большинство (но не все)
использований этого имени, без необходимости полного определения этого имени.
Объявление значения, не определяя его, позволяет писать код, понятный компилятору,
опуская дополнительные детали

Задание 1

Напишите программу, выводящую каждое новое слово с новой строки.

Исходный код

```
#include <stdio.h>

int main() {
    char tmpChar = NULL;
    while (tmpChar != '\n') {
        tmpChar = getchar();
        if (tmpChar != ' ')
            putchar(tmpChar);
        else
            putchar('\n');
    }
}
```

Результат выполнения

```
Привет, Иноземец! Эта строка будет разбита по словам!
Привет,
Иноземец!
Эта
строка
будет
разбита
по
словам!
```

Задание 2

Напишите программу, которая заменяет символы табуляции, перехода на новую строку и остальные управляющие последовательности на \t, \n и т.д.

Исходный код

```
#include <stdio.h>

int main() {
    char tmpChar = NULL;
    while (tmpChar != '*') {
        tmpChar = getchar();
        switch (tmpChar){
            case '\n':
                putchar('/');
                putchar('n');
                break;
            case '\t':
                putchar('/');
                putchar('t');
                break;
            default:
                putchar(tmpChar);
        }
    }
}
```

Результат выполнения

Итак, это строка
Итак, это строка/n

Задание 3

Напишите программу, для подсчета количества пробелов, табуляций и символов перехода на новую строку.

Исходный код

```
#include <stdio.h>

int main() {
    int counterForSpace = 0, counterForTabs = 0, counterForLine = 0;
    char tmpChar = NULL;
    while (tmpChar != '*') {
        tmpChar = getchar();
        switch (tmpChar) {
            case '\n':
                counterForLine++;
                break;
            case '\t':
                counterForTabs++;
                break;
            case ' ':
                counterForSpace++;
                break;
        }
    }
    printf("Количество пробелов: %d\n", counterForSpace);
    printf("Количество табов: %d\n", counterForTabs);
    printf("Количество линий: %d\n", counterForLine);
    printf("Количество 'особых' символов: %d\n", counterForLine + counterForSpace
+ counterForTabs);
}
```

Результат выполнения

```
Привет, это форматированный текст!
Возможно, это кого-то удивит, но тут есть пробелы
Но табов нет ((((
*
Количество пробелов: 13
Количество табов: 0
Количество линий: 3
Количество 'особых' символов: 16
```

Контрольные вопросы

4. Процесс создания программного обеспечения?
5. Компиляция?
6. Переменные?
7. Функции?
8. Константы?
9. Объявления и определения?

Отчет по лабораторной работе № 3 «Типы данных, операции и выражения»			
дата	Оценка (max 3)	Бонус за сложность	подпись

Цели работы:

-Ознакомление с типами данных, операциями и выражениями С

Задачи работы:

-знакомство с основными типами данных, операциями и выражениями С

-разработка примеров простейших программ на С

Краткий конспект теоретической части (ответы на контрольные вопросы)

Переменные - Переменная в императивном программировании — поименованная, либо
адресуемая иным способом область памяти, адрес которой можно использовать
для осуществления доступа к данным.
Типы данных и их размеры- char (1 байт), int (2 байта), long (4 байта), long long (8 байт),
float(в стандарте не описан, обычно 64 бит), double (обычно 64 бит), long double (80 бит)
Символьная константа - Символьная константа – это некоторый символ алфавита,
заключенный в одиночные кавычки
Константное выражение - Константное выражение -- это выражение, состоящее из
одних констант. Такие выражения обрабатываются во время компиляции, а не при
прогоне программы, и соответственно могут быть использованы в любом месте,
где можно использовать константу, как, например в
Строковая константа - это нуль или более символов Unicode, заключенных в
одинарные или двойные кавычки.
Перечислимый тип - в программировании тип данных, чьё множество значений
представляет собой ограниченный список идентификаторов.
Виды операций и их приоритет – так же как и в арифметике

Задание 1

Напишите программу, которая преобразует строку шестнадцатеричных цифр в ее целочисленный эквивалент. Допустимые символы: 0-9, a-f, A-F.

Исходный код

```
int charToInt(char numeric) {
    if (numeric >= '0' && numeric <= '9')
        return numeric - '0';
    else if (numeric >= 'A' && numeric <= 'F')
        return numeric - 'A' + 10;
    else if (numeric >= 'a' && numeric <= 'f')
        return numeric - 'a' + 10;
    else return 0;
}

long long getLongFromHEX(size_t size, char *arr) {
    long long returnValue = 0;
    for (int i = (int) (size - 1), j = 0; i > -1; i--, j++) {
        returnValue += charToInt(arr[i]) *  pow(16.0, (double) j);
    }
    return returnValue;
}
```

Результат выполнения

```
B3A73CF8186
12345678987654
```

Задание 2

Напишите программу, удаляющую из строки s1 все символы, которые содержит строка s2. Удаление символов оформить в виде функции.

Исходный код

```
void removeSymbolsFromString(size_t s1Size, size_t s2Size, size_t outputSize,
char *s1, char *s2, char *outputArray) {
    outputSize = 0;
    bool contains = false;

    for (int i = 0; i < s1Size; i++) {
        for (int j = 0; j < s2Size && !contains; j++)
            if (s2[j] == s1[i])
                contains = true;
        if (!contains)
            outputArray[outputSize++] = s1[i];
        contains = false;
    }
    outputArray[outputSize] = '\0';
}
```

Результат выполнения

```
Privet!
asdfvg
Priet!
```

Задание 3

Напишите программу, которая бы разворачивала сокращенную запись наподобие a-z в строке s1 в полный список abc...xyz в строке s2. Учитывайте буквы в любом регистре, цифры.

Исходный код

```
int calculateNewSize(size_t stringSize, char *string) {
    int newSize = 0;
    for (int i = 0; i < stringSize; i++) {
        newSize++;
        if (string[i] == '-')
            newSize += string[i + 1] - string[i - 1] - 2;
    }
    return newSize + 1;
}

char* getFullString(size_t stringSize, size_t string2Size, char *string1, char *string2) {
    string2Size = (size_t) calculateNewSize(stringSize, string1);
    string2 = malloc(sizeof(char) * string2Size);
    int curPos = 0;

    for (int i = 0; i < stringSize; i++) {
        if (string1[i] != '-')
            string2[curPos++] = string1[i];
        else
            for (char ch = (char) (string1[i - 1] + 1); ch < string1[i + 1]; ch++)
                string2[curPos++] = ch;
    }
    string2[curPos] = '\0';
    return string2;
}
```

Результат выполнения

```
a-zA-Z0-9
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

Контрольные вопросы

1. Переменные?
2. Типы данных и их размеры?
3. Символьная константа?
4. Константное выражение?
5. Строковая константа?
6. Перечислимый тип?
7. Виды операций и их приоритет?

Отчет по лабораторной работе № 4 «Указатели и массивы»			
дата	Оценка (max 3)	Бонус за сложность	подпись

Цели работы:

-Ознакомление с типами данных, операциями и выражениями С

Задачи работы:

-знакомство с основными типами данных, операциями и выражениями С

-разработка примеров простейших программ на С

Краткий конспект теоретической части (ответы на контрольные вопросы)

Указатель - это переменная, которая хранит адрес памяти
Массив - тип или структура данных в виде набора компонентов (элементов массива),
расположенных в памяти непосредственно друг за другом. При этом доступ к
отдельным элементам массива осуществляется с помощью индексации, то есть через
ссылку на массив с указанием номера (индекса) нужного элемента. За счёт этого, в
отличие от списка, массив является структурой данных, пригодной для осуществления
произвольного доступа к её ячейкам[1].
Массив указателей - простейшая структура данных, в которой проявляется различие
между физическим и логическим порядком следования элементов. Способ организации
данных ясен уже из самого определения: это массив, каждый элемент которого содержит
указатель на переменную (объект).
Многомерные массивы - это массив, элементы которого сами являются массивами.
Например, первый элемент трехмерного массива является двумерным массивом.

Задание 1

Напишите программу, которая будет выводить только те строки из входного потока, которые заканчиваются на определенный шаблон. Для проверки напишите функцию, например, int strend(s, t) которая бы возвращала 1, если t является концом s и 0 в противном случае.

Исходный код

```
#include <malloc.h>
#include <string.h>

int strend(char *string, char *templare);

int main() {
    char string[256] = {1};
    char *templare = "nik";
    while (string[0] != 0) {
        scanf("%s", string);
        if (strend(string, templare))
            printf("%s", string);
    }
    return 0;
}

int strend(char *string, char *templare) {
    size_t lenTemplare = strlen(templare);
    size_t lenString = strlen(string);

    if (lenString < lenTemplare)
        return 0;

    for (size_t i = lenString, j = lenTemplare; i > 0 & j > 0; i--, j--)
        if (string[i] != templare[j])
            return 0;

    return 1;
}
```

Результат выполнения

```
привет
приветnik
приветnik
```

Задание 2

Напишите программу, которая будет выводить последние N строк входного потока. Значение N по умолчанию – 10. Пользователь может изменять значение N.
Дополнительные условия:

1. Строки должны храниться в массиве указателей
2. Число N должно передаваться как аргумент командной строки
3. Программа должна устойчиво обрабатывать все возможные варианты пользовательского ввода

Исходный код
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
const int DEFAULT_BUFFER = 128;
// По идее, надо сделать Дек, но эт слишком сложно для дз по Информатике
int main() {
int N = 0;
scanf("%d", &N);
int curPos = 0;
char **strings = calloc((size_t) N, sizeof(char *));
int tmpChar;
int *size;
int *bufferSize;
char *workspace;
while ((tmpChar = getchar()) != EOF) {
if (strings[curPos] == NULL) {
strings[curPos] = (char *) calloc(DEFAULT_BUFFER, sizeof(char));
// Первые 8 байт в строке - служебные
size = (void *) strings[curPos];
bufferSize = (void *) size + sizeof(int);
workspace = (void *) bufferSize + sizeof(int);
*bufferSize = DEFAULT_BUFFER - sizeof(int) - sizeof(int);
}
if (tmpChar == '\n') {
if (++curPos == N) {
char *tmpString = strings[0];
memmove(strings, strings + 1, sizeof(char *) * (N - 1));
curPos = N - 1;
strings[curPos] = tmpString;
size = (void *) strings[curPos];
bufferSize = (void *) size + sizeof(int);
workspace = (void *) bufferSize + sizeof(int);
*size = 0;

```
        }
    } else {
        workspace[(*size)++] = (char) tmpChar;
        if (*size == *bufferSize) {
            *bufferSize *= 1.5;
            strings[curPos] = (char *) realloc(strings[curPos],
                                    sizeof(char) * (*bufferSize) + sizeof(int) + sizeof(int));
        }
    }
  }

  if (N == curPos)
    curPos = N - 1;
  for (int i = 0; i <= curPos; i++) {
    for (int j = 0; j <= *((int *) strings[i]); j++)
      putchar(strings[i][j + sizeof(int) * sizeof(int)]);
    putchar('\n');
  }
  return 0;
}
```

Результат выполнения

```
> 2
> Привет
> Здравствуй
> Приветсвую
Здравствуй
Приветсвую
```

Задание 3

Реализуйте как минимум любые 2 алгоритма сортировки из представленных ниже:

1. Поразрядная сортировка (Radix Sort)
2. Сортировка подсчетом (Counting Sort)
3. Сортировка слиянием (Merge Sort)
4. Один из вариантов улучшения алгоритма быстрой сортировки: randomized quick sort, 3-way quick sort, комбинированная быстрая сортировка (применение сортировкой вставками на маленьких массивах)

Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/**
 * @var number max is 7
 */
int getByte(long long var, int number) {
    return (int) ((var >> (number * 8)) & 255);
}

void radixSort(unsigned int *array, int n) {
    int *keyArray = (int *) malloc(256 * sizeof(int));
    unsigned int *tmpArray = (unsigned int *) malloc(n * sizeof(unsigned int));
    int tmp;
    int lastVar;
    for (int byteNumber = 0; byteNumber < 4; byteNumber++) {
        memset(keyArray, 0, 256 * sizeof(int));

        for (int i = 0; i < n; i++)
            keyArray[getByte(array[i], byteNumber)]++;

        lastVar = 0;
        for (int i = 0; i < 256; i++) {
            tmp = keyArray[i];
            keyArray[i] = lastVar;
            lastVar += tmp;
        }

        for (int i = 0; i < n; i++) {
            tmp = getByte(array[i], byteNumber);
            tmpArray[keyArray[tmp]] = array[i];
            keyArray[tmp]++;
        }

        memmove(array, tmpArray, n * sizeof(unsigned int));
    }
    free(tmpArray);
    free(keyArray);
}

```

```
int main() {
    int n = 0;
    scanf("%d", &n);

    unsigned int *array = (unsigned int *) malloc(sizeof(unsigned int) * n);
    for (int i = 0; i < n; i++)
       scanf("%u", &array[i]);

    radixSort(array, n);

    for (int i = 0; i < n; i++)
        printf("%u ",array[i]);

    free(array);
    return 0;
}
```

Результат выполнения

```
> 9
> 9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9
```

```
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <stdio.h>

void merge(int *arr, int firstStart, int firstEnd, int secondStart, int secondEnd) {
    int start = firstStart;
    int end = secondEnd;
    firstStart -= start;
    firstEnd -= start;
    secondStart -= start;
    secondEnd -= start;
    int size = secondEnd - firstStart + 1;

    int *buffer = malloc(size * sizeof(int));
    memcpy(buffer, arr + start, size * sizeof(int));

    for (int i = start; i <= end; i++) {
        if (secondEnd >= secondStart)
            if (firstEnd >= firstStart) {
                arr[i] = (buffer[firstStart] > buffer[secondStart] ? buffer[secondStart++] : buffer[firstStart
            } else arr[i] = buffer[secondStart++];
        else arr[i] = buffer[firstStart++];
    }

    assert(secondStart > secondEnd);
    assert(firstStart > firstEnd);

    free(buffer);
}
```

```
void mergeSort(int *arr, int start, int end) {
    if (end - start < 2) {
        if (arr[start] > arr[end]) {
            int tmp = arr[start];
            arr[start] = arr[end];
            arr[end] = tmp;
        }
        return;
    }

    int size = end - start + 1;

    int firstStart = start;
    int firstEnd = start + size / 2;
    int secondStart = firstEnd + 1;
    int secondEnd = end;

    assert(firstEnd >= firstStart);
    assert(secondEnd >= secondStart);
    assert(size > 0);

    mergeSort(arr, firstStart, firstEnd);
    mergeSort(arr, secondStart, secondEnd);

    merge(arr, firstStart, firstEnd, secondStart, secondEnd);
}

int main() {
    int n = 0;
    scanf("%d", &n);

    int *array = (int *) malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
        scanf("%u", &array[i]);

    mergeSort(array, 0, n - 1);

    for (int i = 0; i < n; i++)
        printf("%u ", array[i]);

    free(array);
    return 0;
}
```

Результат выполнения

```
> 9
> 9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9
```

Контрольные вопросы

1. Указатели?
2. Массивы?
3. Массивы указателей?

Отчет по лабораторной работе № 5 «Ввод-вывод в языке С»			
дата	Оценка (max 5)	Бонус за сложность	подпись

Цели работы:

-Ознакомление с вводом-выводом в С

Задачи работы:

-знакомство с вводом-выводом в С

-разработка примеров простейших программ на С

Краткий конспект теоретической части (ответы на контрольные вопросы)

Стандартные средства ввода-вывода - - устройства взаимодействия компьютера с
внешним миром: с пользователями или другими компьютерами. Устройства ввода
позволяют вводить информацию в компьютер для дальнейшего хранения и обработки,
а устройства вывода - получать информацию из компьютера. Консоль, файл и прочее
Форматированный ввод-вывод - это совокупность операций, обеспечивающая
ввод/вывод высокого уровня переменных с применением определённого формата
ввода/вывода.
Списки аргументов переменной длинны – это массивы, которые увеличиваются и
уменьшаются в зависимости от количества данных
Работа с файлами – это операции, которые мы можем совершать над файлами из
программы. Открыть файл с помощь fopen, закрыть с помощью fclose()
Обработка ошибок - механизм языков программирования, предназначенный для
описания реакции программы на ошибки времени выполнения и другие возможные
проблемы (исключения), которые могут возникнуть при выполнении программы и
приводят к невозможности (бессмысленности) дальнейшей отработки программой
её базового алгоритма
Ввод-вывод строк – вывод/ввод массива символов в поток консоли/приложения

Задание 1

На вход программы поступают имена трех файлов. Необходимо записать содержимое первых двух файлов в третий в вперемешку. Пример:

File1.txt	File2.txt	File3.txt
Abc Cde Fgh	Qwe Asd	Abc Qwe Cde Asd Fgh

Исходный код

```
#include <stdio.h>
#include <stdbool.h>
int main(int argc, char *argv[]) {
   const int bufferSize = 256;
   if (argc < 4) {
      printf("Необходимо указать название трех файлов");
      return 0;
   }
   FILE *file1 = fopen(argv[1], "r");
   FILE *file2 = fopen(argv[2], "r");
   FILE *file3 = fopen(argv[3], "w");
   char buffer[bufferSize];
   if (file1 == NULL || file2 == NULL || file3 == NULL)
      printf("Проблема при открытии одного из файлов");
   else {
      bool isAdded = true;
      while (isAdded) {
         isAdded = false;

         if (fgets(buffer, bufferSize, file1) != NULL) {
            fputs(buffer, file3);
            isAdded = true;
         }

         if (fgets(buffer, bufferSize, file2) != NULL) {
            fputs(buffer, file3);
            isAdded = true;
         }
      }
   }
   if (file1 != NULL)
      fclose(file1);
   if (file2 != NULL)
      fclose(file2);
   if (file3 != NULL)
      fclose(file3);

   return 0;
}
```

Результат выполнения

```
> ./a.out File1.txt File2.txt File3.txt
```

Задание 2

На вход программы поступает имя файла. Программа должна найти и распечатать все уникальные слова, которые встречаются в файле, и количество их повторений. Для выполнения задания следует воспользоваться структурой данных хэш-таблица.

Исходный код

FILE: hashmap.h

```
#ifndef VYZ_HASHMAP_H
#define VYZ_HASHMAP_H

#define firstBufferSize 16

typedef struct tHashMapElement {
    long hashStr; // Жертвуем ОЗУ в пользу производительности
    const char *str; // Для исключения коллизии
    int element;
} tHashMapElement;

typedef struct tHashMap {
    int size;
    int bufferSize;
    struct tHashMapElement **elementArray;
} tHashMap;


void initHashMap(tHashMap *hashMp);
tHashMapElement *get(const char *str, tHashMap *hsh);
void printElement(tHashMapElement *element);
void addElement(tHashMap *hash, tHashMapElement *element);
void printHash(tHashMap *hash);
unsigned long hash(const char *str);
tHashMapElement *add(tHashMap *hsh, const char *str, int value);
void removeHash(tHashMap *hashMap);
#endif
```

Результат выполнения

FILE: hashmap.c

```
#include <stdlib.h>
#include <string.h>
#include "stdio.h"
#include "hashmap.h"

void initHashMap(tHashMap *hashMp) {
   hashMp->size = 0;
   hashMp->bufferSize = firstBufferSize;
   hashMp->elementArray = (tHashMapElement **) calloc(firstBufferSize,
sizeof(tHashMapElement *));
}

int binarySearch(long hashStr, tHashMap *hsh) {
   int average_index = 0, // переменная для хранения индекса среднего элемента массива
         first_index = 0, // индекс первого элемента в массиве
         last_index = hsh->size - 1; // индекс последнего элемента в массиве
   if (last_index == -1) return 0;
   while (first_index < last_index) {
      average_index = first_index + (last_index - first_index) / 2;
// меняем индекс среднего значения
      if (hashStr <= hsh->elementArray[average_index]->hashStr)
         last_index = average_index;
      else
         first_index = average_index + 1;
   }
   return last_index;
}

void addElement(tHashMap *hash, tHashMapElement *element) {
   if (hash->size == hash->bufferSize) {
      tHashMapElement **newArray = (tHashMapElement **) malloc(
            sizeof(tHashMapElement *) * (hash->bufferSize + firstBufferSize));
      for (int i = 0; i < hash->size; i++)
         newArray[i] = hash->elementArray[i];
      free(hash->elementArray);
      hash->elementArray = newArray;
      hash->bufferSize = hash->bufferSize + firstBufferSize;
   }
   if (hash->size == 0) {
      hash->size = 1;
      hash->elementArray[0] = element;
      return;
   }
   int k = binarySearch(element->hashStr, hash);
   if (hash->elementArray[k]->hashStr < element->hashStr)
      k++;
   for (int i = hash->size - 1; i >= k; i--)
      hash->elementArray[i + 1] = hash->elementArray[i];
   hash->size++;
   hash->elementArray[k] = element;
}
```

```
tHashMapElement *get(const char *str, tHashMap *hsh) {
    long hashStr = hash(str);
    int pos = binarySearch(hashStr, hsh);
    if (hsh->elementArray[pos] != NULL && hsh->elementArray[pos]->hashStr == hashStr)
        if (strcmp(hsh->elementArray[pos]->str, str) == 0)
            return hsh->elementArray[pos];
        else {
            int curPos = pos;
            while (hsh->elementArray[--curPos]->hashStr == hashStr)
                if (strcmp(hsh->elementArray[curPos]->str, str) == 0)
                    return hsh->elementArray[curPos];

            curPos = pos;
            while (hsh->elementArray[++curPos]->hashStr == hashStr)
                if (strcmp(hsh->elementArray[curPos]->str, str) == 0)
                    return hsh->elementArray[curPos];
        }
    return NULL;
}

tHashMapElement *add(tHashMap *hsh, const char *str, int value) {
    tHashMapElement *element = (tHashMapElement *) malloc(sizeof(tHashMapElement));
    char *copy = (char *) malloc(sizeof(char) * strlen(str));
    memcpy(copy, str, sizeof(char) * strlen(str));
    element->str = copy;
    element->hashStr = hash(str);
    element->element = value;
    addElement(hsh, element);
    return element;
}

void printElement(tHashMapElement *element) {
    printf("Хеш: %lu \nСтрока: \"%s\"\nЗначение: \"%d\"\n\n", element->hashStr, element->str,
           element->element);

}

void printHash(tHashMap *hash) {
    for (int i = 0; i < hash->size; i++) {
        printf("Элемент %d:\n", i);
        printElement(hash->elementArray[i]);
    }
}

void removeHash(tHashMap *hashMap) {
    for (int i = 0; i < hashMap->size; i++) {
        free((void *) hashMap->elementArray[i]->str);
        free(hashMap->elementArray[i]);
    }
    free(hashMap->elementArray);
}
```

```
/**
* Алгоритм хеширования djb2 был взят отсюда: http://www.cse.yorku.ca/~oz/hash.html
*
* @param str
* @return НЕ обладает 100% уникальностью
*/
unsigned long hash(const char *str) {
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

FILE: main.c

```
const int BUFFER_SIZE = 128;

char *nextWord(FILE *file) {
    int bufferSize = BUFFER_SIZE;
    int count = 0;
    char *word = (char *) calloc((size_t) bufferSize, sizeof(char));
    int tmpChar = 0;
    while ((tmpChar = fgetc(file)) != EOF && tmpChar != ' ' && tmpChar != ',' && tmpChar != '\n') {
        word[count++] = (char) tmpChar;
        if (count == bufferSize) {
            bufferSize *= 1.5;
            word = (char *) realloc(word, bufferSize * sizeof(char));
        }
    }
    word[count] = 0;
    if (count == 0) {
        free(word);
        word = NULL;
    }
    return word;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "hashmap.h"

char *nextWord(FILE *file);

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Необходимо указать имя файла");
        return 0;
    }
    FILE *file = fopen(argv[1], "r");

    if (file == NULL) {
        printf("Невозможно открыть файл");
        return 0;
    }
    char *word = NULL;

    tHashMap hashMap;
    initHashMap(&hashMap);
    tHashMapElement *tmpElement;

    while ((word = nextWord(file)) != NULL) {
        tmpElement = get(word, &hashMap);
        if (tmpElement == NULL) {
            add(&hashMap, word, 1);
        } else {
            tmpElement->element++;
        }
        free(word);
    }

    printHash(&hashMap);
    removeHash(&hashMap);

    if (file != NULL)
        fclose(file);
    return 0;
}
```

Задание 3

На вход программы поступает имя файла. Программа вывести список всех слов файла в алфавитном порядке. Для каждого слова необходимо указать номера страниц и строк, где оно встречается. Под страницей следует понимать 20 или 50 строк. Для выполнения задания следует воспользоваться структурой данных бинарное дерево. Более сложный вариант задания: использовать одну из реализаций сбалансированного бинарного дерева (например, красно-черное дерево).

Исходный код

```
#include <stdlib.h>
#include "stdio.h"

const int BUFFER_SIZE = 16;
const int linePerPage = 20;

int compare(char *str1, char *str2) {
    if (str1 == NULL || str2 == NULL)
        if (str1 == NULL && str2 == NULL)
            return 0;
        else if (str1 == NULL)
            return -1;
        else return 1;
    for (int i = 0; str1[i] != '\0' && str2[i] != '\0'; i++) {
        if (str1[i] != str2[i])
            return str1[i] - str2[i];
    }
    return 0;
}

typedef struct Word {
    char *key;
    int lineNumber;
    int pageNumber;
} Word;

typedef struct Node {
    Word key;
    unsigned char height;
    struct Node *left;
    struct Node *right;
} Node;
```

Результат выполнения

```
<< ./a.out File1.txt
>> Abc 1 1
>> Cde 2 1
>> Fgh 3 1
```

```
Word nextWord(FILE *file, int *lineNumber, int *page);

unsigned char height(Node *p) {
    return p ? p->height : (unsigned char) 0;
}

int bfactor(Node *p) {
    return height(p->right) - height(p->left);
}

void fixheight(Node *p) {
    unsigned char hl = height(p->left);
    unsigned char hr = height(p->right);
    p->height = (unsigned char) ((hl > hr ? hl : hr) + 1);
}
Node *rotateright(Node *p) // правый поворот вокруг p
{
    Node *q = p->left;
    p->left = q->right;
    q->right = p;
    fixheight(p);
    fixheight(q);
    return q;
}
Node *initNode(Word word) {
    Node *node = malloc(sizeof(Node));
    node->key = word;
    node->right = NULL;
    node->left = NULL;
    node->height = 0;
    return node;
}
Node *rotateleft(Node *q) // левый поворот вокруг q
{
    Node *p = q->right;
    q->right = p->left;
    p->left = q;
    fixheight(q);
    fixheight(p);
    return p;
}
Node *insert(Node *p, Word k) // вставка ключа k в дерево с корнем p
{
    if (!p) {
        return initNode(k);
    };
    if (compare(k.key, p->key.key) < 0)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}
```

```
Node *balance(Node *p) // балансировка узла p
{
    fixheight(p);
    if (bfactor(p) == 2) {
        if (bfactor(p->right) < 0)
            p->right = rotateright(p->right);
        return rotateleft(p);
    }
    if (bfactor(p) == -2) {
        if (bfactor(p->left) > 0)
            p->left = rotateleft(p->left);
        return rotateright(p);
    }
    return p; // балансировка не нужна
}

void freeNode(Node *p) {
    if (p != NULL) {
        freeNode(p->left);
        freeNode(p->right);
        free(p->key.key);
        free(p);
    }
}

void printNode(Node *node) {
    if (node == NULL)
        return;
    printNode(node->left);
    printf("%s: %d %d\n", node->key.key, node->key.lineNumber, node->key.pageNumber);
    printNode(node->right);
}
```

```
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Необходимо указать название файла");
        return 0;
    }
    Node *rootNode;
    FILE *file = fopen(argv[1], "r");
    int page = 1;
    int line = 1;

    if (file == NULL) {
        printf("Невозможно открыть файл");
        return 0;
    }
    Word word = nextWord(file, &line, &page);
    rootNode = initNode(word);
    while ((word = nextWord(file, &line, &page)).key != NULL) {
        insert(rootNode, word);
    }

    printNode(rootNode);
    freeNode(rootNode);
    fclose(file);
    return 0;
}
Word nextWord(FILE *file, int *lineNumber, int *page) {
    int bufferSize = BUFFER_SIZE;
    int count = 0;
    char *word = (char *) calloc((size_t) bufferSize, sizeof(char));
    int tmpChar = 0;
    Word toOut;
    toOut.lineNumber = *lineNumber;
    toOut.pageNumber = *page;
    while ((tmpChar = fgetc(file)) != EOF && tmpChar != ' ' && tmpChar != ',') {
        if (tmpChar == '\n') {
            (*lineNumber)++;
            break;
        }
        word[count++] = (char) tmpChar;
        if (count == bufferSize) {
            bufferSize *= 1.5;
            word = (char *) realloc(word, bufferSize * sizeof(char));
        }
    }
    word[count] = 0;
    if (count == 0) {
        free(word);
        word = NULL;
    }
    toOut.key = word;
    if (*lineNumber > linePerPage) {
        *lineNumber = 1;
        (*page)++;
    }
    return toOut;
}
```

Контрольные вопросы

1. Система ввода-вывода языка С?
2. Форматированный ввод-вывод?
3. Списки аргументов переменной длинны?
4. Работа с файлами в С?
5. Обработка ошибок?
6. Ввод-вывод строк в языке С?
7. Бинарное дерево?

СПИСОК ЛИТЕРАТУРЫ

1. Норенков И.П. Системы автоматизированного проектирования. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.
2. Норенков И.П. Основы автоматизированного проектирования. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2000.
3. Норенков И.П., Трудоношин В.А. Телекоммуникационные технологии и сети - М.: Изд-во МГТУ им. Н.Э. Баумана, 2000.
4. Керниган Б., Ритчи Д. Язык программирования С. – М.: Издательский дом «Вильямс», 2010.
5. Немет Э., Снайдер Г., Хейн Т. Руководство администратора Linux. – М.: Издательский дом «Вильямс», 2010.
6. Шилдт Г. Полный справочник по C++. - М.: Издательский дом «Вильямс», 2010.
7. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. М.: Издательский дом «Вильямс», 2011.