МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский государственный университет аэрокосмического приборостроения»
ГУАП

Е.П. Овсянников

Программирование на языке Си.

Учебно-методическое пособие

УДК 681.3.06
Д69

Рецензенты:
Б.Д. Кудряшов - д-р техн. наук, проф. Санкт-Петербургского национального исследовательского университета информационных технологий механики и оптики;
А.Н.Трофимов - канд. техн. наук, доц. Санкт-Петербургского государственного университета аэрокосмического приборостроения.

Учебно-методическое пособие представляет собой лабораторный практикум для изучения языка программирования Си в рамках дисциплины "Основы программирования". Содержит 11 лабораторных работ, целью которых является получение знаний и навыков работы с основными типами данных и конструкциями языка Си. Для каждой лабораторной работы приведён и разобран пример и представлена комментированная программа на языке Си.

Учебно-методическое пособие подготовлено на кафедре №14 "Кафедра аэрокосмических компьютерных и программных систем" Санкт-Петербургского государственного университета аэрокосмического приборостроения и предназначено для студентов, обучающихся по информационным специальностям, в том числе по направлению 09.03.01 "Информатика и вычислительная техника" и 09.05.01 "Применение и эксплуатация автоматизированных систем специального назначения".

Оглавление

Введение

Данное пособие предназначено для изучения языка программирования Си и содержит набор задач, который предполагает, что для выполнения очередной работы будет необходимо изучить и использовать очередную конструкцию языка Си. Для успешного выполнения каждой лабораторной работы рекомендуется сначала, воспользовавшись конспектом лекций или соответствующими разделами из рекомендованной литературы [1 - 4], ответить на контрольные вопросы. Это значительно облегчит понимание задач, приведённых в качестве примера, и ускорит написание собственных программ.

Каждая лабораторная работа завершается её защитой, на которую нужно предъявить отчёт, содержащий

- титульный лист,
- цель работы,
- задание,
- формализацию задачи,
- набор тестовых примеров (как минимум два) для проверки программы,
- схема алгоритма,
- листинг программы,
- выводы, доказывающие правильность функционирования программы.

На защите лабораторной работы с целью проверки глубины знаний автору программы может быть предложено выполнить её модификацию, не требующую значительных изменений алгоритма.

Критериями оценки программы являются

- функциональность программы, т.е. правильное выполнение поставленной задачи на широком наборе входных данных,
- сроки выполнения, указываемые при выдаче задания,
- правильность и полнота выполнения отчёта,
- знания, показанные при защите лабораторной работы.

Лабораторная работа №1. Линейные алгоритмы

Цель работы: изучение основной структуры линейной программы на языке Си.

Методические указания.

Линейные алгоритмы являются простейшим видом алгоритмов. Чаще всего они не являются самостоятельными задачами, а скорее представляют собой фрагменты более сложных вычислительных процессов. Особенностью линейного алгоритма является то, что каждый его шаг выполняется ровно один раз. Рассмотрим на примере реализацию линейного алгоритма на языке Си.

Задача. Из пункта А в пункт Б, находящийся на расстоянии L км , выехала машина со скорость v_1 км/ч. Навстречу ей выехал мотоциклист со скоростью v_2 км/ч. На каком расстоянии от пункта А они встретятся?

Несмотря на всю простоту задачи программа, реализующая её решение, обладает всеми необходимыми атрибутами Си-программы, и может быть использована как шаблон. Основные элементы программы помечены комментариями, которые начинаются с двойной наклонной черты.

```
#include <stdio.h> // необходимо для корректного вызова
                   //функций ввода/вывода

int main( void ) // главная функция main не принимает
                 // данных от операционной системы,
                 // возвращает статус завершения

{    // начало тела программы
```

```
// описание переменных
float L;          // расстояние между пунктами
float v1;         // скорость машины
float v2;         // скорость мотоциклиста
float t;          // время до встречи
float s;          // искомое расстояние

//  ввод исходных данных
// вывод на экран подсказки: L =
printf("L = ");

// ввод с клавиатуры значения переменной L
scanf("%f", &L);

// вывод на экран подсказки: v1 =
printf("v1 = ");

// ввод с клавиатуры значения переменной v1
scanf("%f", &v1);

// вывод на экран подсказки: v2 =
printf("v2 = ");

// ввод с клавиатуры значения переменной v2
scanf("%f", &v2);

//  решение задачи
t = L / (v1 + v2);// вычисление времени до встречи
s = t * v1;       // вычисление искомого расстояния
```

```
    //  вывод результатов
    // вывод на экран значения переменной s
    printf("distance = %f\n", s);

    // программа возвращается  признак корректного
    // завершения
    return 0;
}   // конец тела программы
```

Заметим, что приведённая выше программа является консольным приложением. То есть она не обладает оконным интерфейсом и лишена графических возможностей. Все программы, которые будут представлены в этом пособии, будут являться точно такими же консольными приложениями. В рамках данного курса всё внимание будет уделяться изучению языка Си, а не способам "красивого" представления результатов работы программы на экране компьютера.

Структура данной программы весьма проста. Сначала идёт блок описания переменных, которые будут использованы в дальнейшем. Для каждой переменной указывается её имя и тип - в данном случае это float, то есть переменная, в которой можно хранить вещественное число. После описания переменных следуют так называемые "исполняемые" операторы. Они и являются реализацией линейного алгоритма решения задачи.

В программе реализован простой диалог, позволяющий ввести требуемые значения расстояния и скорости. Затем вычисляется время до встречи и искомое расстояние. В конце программы на экран выводится результат вычислений.

Для ввода значения переменной с клавиатуры использована библиотечная функция scanf. Заметим, что перед именем переменной, значение которой будет введено, стоит символ & (амперсанд). Его

необходимость будет объяснена в дальнейшем, когда будут изучаться такие элементы языка Си как указатели.

Для вывода значения переменной на экран использована библиотечная функция printf.

Варианты лабораторных заданий.

Задача 1.1

Из пункта А в пункт Б, выехала машина со скорость v_1 *км/ч*. Навстречу ей выехал мотоциклист со скоростью v_2 *км/ч*. На каком расстоянии находятся пункты А и Б, если машина и мотоциклист встретились через t часов?

Задача 1.2

Бассейн объёмом V литров полностью заполнен водой. По одной трубе вода из бассейна вытекает со скоростью v_1 литров в минуту, по другой трубе вода поступает со скоростью v_2 литров в минуту. Через какое время бассейн станет пустым, если вода вытекает быстрее, чем поступает?

Задача 1.3

Бассейн объёмом V литров полностью заполнен водой. По одной трубе вода из бассейна вытекает со скоростью v_1 литров в минуту, по другой трубе вода поступает. Бассейн стал пустым через время t. Какова скорость поступления воды v_2?

Задача 1.4

Катер проходит расстояние L по течению реки за время t_1 минут, а против течения - за время t_2 минут. Определить скорость течения реки.

Задача 1.5

Самолёт касается взлётно-посадочной полосы со скоростью v_1 км/час и за t секунд успевает сбросить скорость до v_2 км/час. Какое расстояние успевает проехать самолёт за это время по взлётно-посадочной полосе?

Задача 1.6

Самолёт касается взлётно-посадочной полосы со скоростью v_1 км/час и останавливается, преодолев расстояние L. Сколько времени длится торможение?

Задача 1.7

Из круга радиуса R вырезали квадрат со стороной A. Какова площадь оставшейся фигуры?

Задача 1.8

У конуса высотой H с радиусом основания R отрезали верхнюю часть. Каков объём оставшейся фигуры, если D - высота отрезанной части ?

Задача 1.9

Из пункта А в пункт Б, расстояние между которыми L км, одновременно вышли два пешехода. Первый двигался со скоростью v_1 км/час. Какова скорость второго, если они встретились через t часов?

Задача 1.10

Шарик, брошенный с высоты H вертикально вниз достиг поверхности за время t. Какова была начальная скорость шарика?

Контрольные вопросы.

1. Как называется главная функция программы, написанной на языке Си?
2. С помощью какой функции осуществляется вывод на экран?
3. С помощью какой функции осуществляется ввод данных с клавиатуры?
4. Какие типы данных существуют в языке Си?
5. Какие спецификации формата следует применять для ввода/вывода вещественных переменных?
6. Какие спецификации формата следует применять для ввода/вывода целочисленных
переменных?
7. Какие арифметические операции поддерживаются в языке Си?

Лабораторная работа №2. Циклические алгоритмы

Цель работы: изучение способов организации циклических вычислений с известными и неизвестным числом итераций.

Методические указания.

Циклические алгоритмы представляют собой один из примеров разветвляющихся алгоритмов. От линейных алгоритмов их отличает то, что в процессе выполнения задачи разные шаги алгоритма могут выполняться неодинаковое количество раз, а некоторые - не выполняться вообще. Это зависит от исходных данных, которые вводятся в программу.

Задача. Для заданного значения x *вычислить сумму первых* n *элементов ряда*

$$\frac{1}{x}-\frac{3}{2x}+\frac{5}{3x}-\frac{7}{4x}+\ldots.$$

Основной трудностью решения подобных задач является поиск закономерности, которая описывает изменение элементов ряда. В данном случае нетрудно вывести формулу для i-го члена ряда:

$$a_i=\frac{1+2i}{(i+1)\cdot x}, i=0,1,2,...$$

Соответственно искомая сумма описывается формулой

$$S_n=\sum_{i=0}^{n-1}(-1)^i a_i .$$

Ниже приведена программа, реализующая вычисления по данным формулам.

```
#include <stdio.h>

int main( void )
{
    float s;      // сумма ряда
    float a;      // очередной элемент ряда
```

```
    float x;     // входной параметр ряда
    int n;       // число элементов ряда
    int i;       // номер очередного элемента ряда
    int sign;    // знак очередного элемента ряда

    // ввод параметров ряда
    printf("n=");
    scanf("%d", &n);
    printf("x=");
    scanf("%f", &x);

    // начальное присваивание переменных
    s = 0;
    sign = 1;
    i = 0;

    // цикл
    while( i < n )
    {
        a = (1+i*2) / ((i+1)*x);
        s = s + a*sign;
        sign = -sign;
        i = i + 1;
    }

    // вывод результата
    printf("s = %f\n", s);

    return 0;
}
```

Иногда достаточно затруднительно найти простую формулу для i- го элемента ряда. Тогда задачу можно решить, подобрав рекуррентные зависимости отдельно для числителя и знаменателя. В нашем случае это

$$\begin{aligned} c_i &= c_{i-1} + 2 \\ b_i &= b_{i-1} + x \end{aligned},$$

где c_i и b_i- соответственно числитель и знаменатель i- го члена ряда. При этом начальные значения для числителя и знаменателя имеют вид

$$\begin{aligned} c_0 &= 1 \\ b_0 &= x \end{aligned}.$$

Программа, реализующая вычисление суммы элементов ряда на основе рекуррентных соотношений, представлена ниже.

```
int main( void )
{
    float s; // сумма ряда
    float a; // очередной элемент ряда
    float c; // числитель очередного элемента ряда
    float b; // знаменатель очередного элемента ряда
    float x; // входной параметр ряда
    int n;   // число элементов ряда
    int i;   // номер очередного элемента ряда
    int sign;    // знак очередного элемента ряда

    // ввод параметров ряда
    printf("n=");
    scanf("%d", &n);
    printf("x=");
    scanf("%f", &x);
```

```
    // начальное присваивание переменных
    s = 0;
    sign = 1;
    i = 0;
    c = 1;
    b = x;
    // цикл
    while( i < n )
    {
        a = c / b;
        s = s + a*sign;
        sign = -sign;
        c = c + 2;
        b = b + x;
        i = i + 1;
    }

    // вывод результата
    printf("s = %f\n", s);

    return 0;
}
```

Рассмотрим ещё одну задачу, в которой применяется другое правило остановки вычислений.

Задача. Для заданного значения x вычислить сумму первых элементов ряда $\frac{1}{x}-\frac{3}{x^2}+\frac{5}{x^3}-\frac{7}{x^4}+\ldots$. *Вычисления проводить, пока модуль очередного элемента больше* ε.

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>

int main( void )
{
    float s; // сумма ряда
    float a; // очередной элемент ряда
    float c; // числитель очередного элемента ряда
    float b; // знаменатель очередного элемента ряда
    float x; // входной параметр ряда
    float eps;   // пороговое значение
    int i;       // номер очередного элемента ряда
    int sign;    // знак очередного элемента ряда

    // ввод параметров ряда
    printf("eps=");
    scanf("%f", &eps);
    printf("x=");
    scanf("%f", &x);

    // начальное присваивание переменных
    s = 0;
    sign = 1;
    i = 0;
```

```
    c = 1;
    b = x;
    a = c / b;

    // цикл
    while( a > eps )
    {
        s = s + a*sign;
        sign = -sign;
        c = c + 2;
        b = b * x;
        a = c / b;
        i = i + 1;
    }

    // вывод результата
    printf("n = %d, s = %f\n", i, s);

    return 0;
}
```

Варианты лабораторных заданий.

Задача 2.1

Вычислить сумму первых N элементов ряда:

$$1-\frac{2}{2}+\frac{3}{4}-\frac{4}{8}+\ldots$$

.

Задача 2.2

Вычислить сумму первых N элементов ряда:

$$1+\frac{1\cdot 3}{1\cdot 4}+\frac{1\cdot 3\cdot 5}{1\cdot 4\cdot 7}+\frac{1\cdot 3\cdot 5\cdot 7}{1\cdot 4\cdot 7\cdot 10}+\ldots \ .$$

Задача 2.3

Вычислить сумму первых N элементов ряда:

$$\frac{1}{4}+\frac{2}{9}+\frac{3}{16}+\frac{4}{25}+\ldots \ .$$

Задача 2.4

Вычислить сумму первых N элементов ряда:

$$\frac{3}{1}-\frac{4}{4}+\frac{5}{7}-\frac{6}{10}+\ldots \ .$$

Задача 2.5

Вычислить сумму первых N элементов ряда:

$$\frac{3}{1\cdot 2}-\frac{4}{2\cdot 3}+\frac{5}{3\cdot 4}-\frac{6}{4\cdot 5}+\ldots \ .$$

Задача 2.6

Для заданного значения x вычислить сумму первых N элементов ряда:

$$1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots \ .$$

Задача 2.7

Для заданного значения x вычислить сумму первых N элементов ряда:

$$\frac{x^1}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots \ .$$

Задача 2.8

Для заданного значения x вычислить сумму первых N элементов ряда:

$$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}+\ldots \ .$$

Задача 2.9

Для заданного значения x вычислить сумму первых N элементов ряда:

$$x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}+\ldots \ .$$

Задача 2.10

Для заданного значения x вычислить сумму первых N элементов ряда:

$$\frac{1}{x}+\frac{2}{x^2}+\frac{3}{x^3}+\frac{4}{x^4}+\ldots \quad .$$

Задача 2.11

Вычислить сумму первых элементов ряда:

$$1-\frac{2}{2}+\frac{3}{4}-\frac{4}{8}+\ldots \quad .$$

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.12

Вычислить сумму первых элементов ряда:

$$1+\frac{1\cdot 3}{1\cdot 4}+\frac{1\cdot 3\cdot 5}{1\cdot 4\cdot 7}+\frac{1\cdot 3\cdot 5\cdot 7}{1\cdot 4\cdot 7\cdot 10}+\ldots .$$

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.13

Вычислить сумму первых элементов ряда:

$$\frac{1}{4}+\frac{2}{9}+\frac{3}{16}+\frac{4}{25}+\ldots \quad .$$

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.14

Вычислить сумму первых элементов ряда:

$$\frac{3}{1}-\frac{4}{4}+\frac{5}{7}-\frac{6}{10}+\ldots \quad .$$

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.15

Вычислить сумму первых элементов ряда:

$$\frac{3}{1\cdot 2}-\frac{4}{2\cdot 3}+\frac{5}{3\cdot 4}-\frac{6}{4\cdot 5}+\ldots \quad .$$

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.16

Для заданного значения x вычислить сумму первых элементов ряда:

$$1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+...$$.

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.17

Для заданного значения x вычислить сумму первых элементов ряда:

$$\frac{x^1}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+...$$.

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.18

Для заданного значения x вычислить сумму первых элементов ряда:

$$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}+...$$.

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.19

Для заданного значения x вычислить сумму первых N элементов ряда:

$$x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}+...$$.

Вычисления проводить, пока модуль очередного элемента больше ε.

Задача 2.20

Для заданного значения x вычислить сумму первых N элементов ряда:

$$\frac{1}{x}+\frac{2}{x^2}+\frac{3}{x^3}+\frac{4}{x^4}+...$$.

Вычисления проводить, пока модуль очередного элемента больше ε.

Контрольные вопросы.

1. Какие циклы с предусловием существуют в языке Си?
2. Как выглядит цикл с постусловием на языке Си?
3. В чём отличие цикла с предусловием от цикла с постусловием?
4. Как выглядит графическое представление цикла с предусловием?
5. Как выглядит графические представление цикла с постусловием?

6. Как работает оператор break?

7. Как работает оператор continue?

8. Что такое вложенные циклы?

Лабораторная работа №3. Обработка потока символов

Цель работы: изучение организации ввода/вывода потока символов и разбиения его на лексемы.

Методические указания.

Обработка символьной информации - весьма часто встречающаяся на практике задача. Подсчет слов в потоке символов, порождаемых с клавиатуры, это достаточно простой пример, позволяющий изучить приёмы чтения потока и разбиения его на слова.

Для упрощения алгоритма наложим ограничения на разделители слов. Будем считать, что слова в потоке символов разделяются пробелами, точками, запятыми, концом строки и любыми их комбинациями. Следовательно, перенос слов между строками запрещён.

Очевидно, что для разбора символьных строк уже давно существует набор библиотечных функций. Но, поскольку именно организация разбора потока является одним из ключевых моментов обучения в данной лабораторной работе, введём дополнительное условие: не допускается использование библиотек работы с символьными строками.

Задача. В потоке символов сосчитать число слов, заканчивающихся на две одинаковых буквы.

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>

// определение символических констант
#define YES   1
#define NO   0
```

```
int main( void )
{
    int c;        // текущий символ из потока
    int prev_c;   // предыдущий символ
    int flag;     // признак слова
    int cnt;      // счётчик
    int found;    // индикатор того, что искомый признак
                  // обнаружен

    // начальные присваивания (инициализация)
    cnt = 0;
    flag = NO;
    found = NO;
    prev_c = ' ';

    // цикл чтения символов из потока, связанного с
    // клавиатурой
    while( (c = getchar()) != EOF )
    {
        if( c == ' ' || c == '.' || c == '\n' ||
            c == ',' )
        {
            // найден разделитель
            if( flag == YES )
            {
                // это первый разделитель после слова
                if( found == YES )
                {
                    // две последних буквы в слове
                    // были одинаковы
```

```
                    cnt = cnt + 1;
                }
            }
            flag = NO;
        }
        else
        {
            // найдена буква

            // проверка совпадения текущего и
            // предыдущего символов
            if( prev_c == c )
                found = YES;
            else
                found = NO;

            flag = YES;
        }
        prev_c = c;
    }
    printf("number of words = %d\n", cnt );
    return 0;
}
```

Следует отметить, что задания 3.18, 3.19 и 3.20 имеют повышенную сложность по сравнению с остальными.

Варианты лабораторных заданий.

Задача 3.1

В потоке символов сосчитать число слов, у которых первые две буквы совпадают.

Задача 3.2

В потоке символов сосчитать число слов, содержащих букву ‘а’.

Задача 3.3

В потоке символов сосчитать среднюю длину слова.

Задача 3.4

В потоке символов сосчитать число слов, содержащих гласные буквы.

Задача 3.5

В потоке символов сосчитать число слов, начинающихся с гласной буквы.

Задача 3.6

В потоке символов сосчитать число слов, заканчивающихся на согласную букву.

Задача 3.7

В потоке символов сосчитать число слов, начинающихся на гласную букву и заканчивающихся на согласную букву.

Задача 3.8

В потоке символов сосчитать число слов, в которых встречаются подряд идущие одинаковые буквы.

Задача 3.9

В потоке символов сосчитать число слов, состоящих из более чем N букв.

Задача 3.10

В символьной строке сосчитать число слов, состоящих из нечётного количества букв.

Задача 3.11

В потоке символов сосчитать число слов, не содержащих гласных букв.

Задача 3.12

Определить, являются ли все слова из потока символов словами одинаковой длины.

Задача 3.13

В потоке символов найти самое длинное слово.

Задача 3.14

В потоке символов сосчитать слова, начинающиеся и заканчивающиеся на одну и ту же букву.

Задача 3.15

В потоке символов сосчитать количество десятичных чисел.

Задача 3.16

В потоке символов сосчитать число слов в каждом предложении. Предложения разделяются точкой.

Задача 3.17

В потоке символов сосчитать число слов, в которых согласных букв меньше чем гласных.

Задача 3.18

В потоке символов сосчитать число слов – палиндромов.

Задача 3.19

В потоке символов найти самое длинное слово – палиндром.

Задача 3.20

В потоке символов сосчитать число слов, состоящих только из разных букв.

Контрольные вопросы.

1. Какую функцию следует использовать для чтения символа из потока?
2. Почему функция getchar() возвращает не char, а int?
3. Для чего используется символическая константа EOF?
4. Что из себя представляет таблица ASCII?
5. Что такое escape-последовательность?

Лабораторная работа №4. Обработка символьных строк

Цель работы: изучение формата и правил описания символьных строк, а также методов обработки строковых данных.

Методические указания.

Работа с символьными строками чаще всего сводится либо к поиску слов с нужными свойствами (см. предыдущий раздел), либо к изменению содержимого строки по заданному правилу. Следует помнить, что строка - это массив. И если в результате обработки строка удлиняется, то программисту следует позаботиться о том, чтобы новое содержимое строки не вышло за пределы массива. В текущем разделе будут рассматриваться задачи, в которых строка укорачивается. Поэтому все изменения строки происходят "на месте", т.е. в том же самом массиве. Дополнительное ограничение - не допускается использование библиотек работы с символьными строками.

Задача. В символьной строке удалить слова, заканчивающиеся на две одинаковых буквы.

Программа, реализующая данную задачу, представлена ниже. Основная идея алгоритма состоит в том, что для каждого слова исходной строки надо вынести решение - сохранить его или нет. Если в слове не обнаружены искомые признаки, в данном случае две одинаковых последних буквы, то оно копируется в выходную строку. В данной программе входная и выходная строки это один и тот же массив. Поэтому, если слов с заданным признаком в строке нет, массив будет посимвольно копироваться сам в себя. В общем случае, массив будет копироваться сам в себя ("тривиальное" копирование) до тех пор, пока не встретится первое слово с искомыми признаками. Затем до конца строки копирование будет уже

"нетривиальным". Предложенный алгоритм можно легко модифицировать для того, чтобы исключить "тривиальное" копирование.

```
#include <stdio.h>
// определение символических констант
#define YES   1
#define NO   0
#define MAXLINE 1000

void process_line( char buffer[] );

int main( void )
{
    char line[MAXLINE];

    gets( line );
    process_line( line );
    puts( line );
    return 0;
}

void process_line( char buffer[] )
{
    char c;      // текущий символ
    int prev_c;  // предыдущий символ
    int flag;    // признак слова
    int found;   // индикатор того, что искомый признак
                 // в слове обнаружен
    int i;       // позиция текущего символа исходной
                 // строки
```

```
int pos;     // позиция текущего символа
             // результирующей строки
int start;   // позиция начала слова
int j;

// начальные присваивания (инициализация)
flag = NO;
found = NO;
prev_c = ' ';
start = 0;
i = 0;
pos = 0;

// цикл чтения символов из строки
do
{
    c = buffer[i];   // взять текущий символ из
                     // буфера

    if( c == ' ' || c == '.' || c == ',' ||
        c == '\n' || c == '\0')
    {
        // найден разделитель
        if( flag == YES )
        {
            // это первый разделить после слова

            // проверить, обнаружен ли в слове
            // искомый признак
            if( found == NO )
```

```
            {
                // слово не подлежит удалению
                // оно копируется в результирующую
                // строку
                for( j = start; j < i; j++ )
                    buffer[pos++] = buffer[j];
            }
        }
        flag = NO;
        buffer[pos++] = c;
    }
    else
    {
        // найдена буква
        if( flag == NO )
            start = i;   // запомнить позицию
                         // начала слова
        // проверка совпадения текущего и
        // предыдущего символов
        // (обнаружение искомого признака в слове)
        if( prev_c == c )
            found = YES;
        else
            found = NO;

        flag = YES;
    }

    prev_c = c;
    i++;
```

```
    }
    while( c != '\0' );
}
```

Варианты лабораторных заданий.

Следует отметить, что задания 4.18, 4.19 и 4.20 имеют повышенную сложность по сравнению с остальными.

Задача 4.1

В символьной строке удалить все слова, начинающиеся с гласной буквы.

Задача 4.2

В символьной строке удалить все слова, начинающиеся и заканчивающиеся на одну и ту же букву.

Задача 4.3 В символьной строке удалить все слова, состоящие из нечетного количества букв.

Задача 4.4

В символьной строке удалить все слова, состоящие более чем из N букв.

Задача 4.5

В символьной строке удалить все слова с четным номером.

Задача 4.6

В символьной строке удалить предпоследнее слово.

Задача 4.7

В символьной строке удалить все слова, содержащие две подряд идущие гласные буквы.

Задача 4.8

В символьной строке оставить только те слова, в которых встречаются подряд идущие одинаковые буквы.

Задача 4.9

В символьной строке удалить все лишние подряд идущие пробелы.

Задача 4.10

В символьной строке удалить самое длинное слово.

Задача 4.11

В символьной строке удалить все слова, в которых число гласных букв больше числа согласных букв.

Задача 4.12

В символьной строке удалить все числа.

Задача 4.13

В символьной строке удалить все слова, содержащие чётное количество гласных букв.

Задача 4.14

В символьной строке удалить слово, стоящее после запятой.

Задача 4.15

В символьной строке оставить только те слова, которые состоят только из разных букв.

Задача 4.16

В символьной строке удалить все слова, начинающиеся с заданной приставки.

Задача 4.17

В символьной строке удалить все слова палиндромы

Задача 4.18

В символьной строке удалить повторяющиеся слова.

Задача 4.19

В символьной строке переставить слова в зеркальном порядке.

Задача 4.20

В символьной строке удалить все слова, которые содержат буквы, не входящие в разрешенный набор символов.

Контрольные вопросы.

1. В каком формате хранится символьная строка в языке Си?
2. В чем различие ввода символьной строки при помощи функции scanf() и gets()?
3. Как правильно описать массив для хранения символьной строки?
4. Символьная строка какой длины может храниться в массиве, описанном как
char line[100]?
5. По какому принципу должна работать библиотечная функция, определяющая длину символьной строки?

Лабораторная работа №5. Одномерные массивы

Цель работы: получение навыков работы с одномерными массивами.

Методические указания.

Вычислительные задачи зачастую связаны с обработкой наборов однородных данных, которые с точки зрения программирования и представляют собой массивы. При работе с массивами следует помнить, что в отличие от некоторых других языков программирования язык Си при обращении к элементу не контролирует выход за границы массива. С одной стороны, этим достигается высокое быстродействие программ. С другой стороны, это предполагает усиленное внимание к тестированию программ, поскольку выход индекса элемента за пределы массива может проявляться не сразу и иметь отложенное по времени влияние на поведение программы.

Задача. В массиве из 10 целых чисел обнулить все элементы, большие чем среднее арифметическое по массиву .

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>
#define N 10

int main( void )
{
    int x[N];    // массив из N элементов
    int aver;    // среднее арифметическое
    int i;

    // ввод массива
```

```
    for( i = 0; i < N; i++ )
        scanf("%d", &x[i]);

    // вычисление среднего арифметического значения
    aver = 0;
    for( i = 0; i < N; i++ )
        aver = aver + x[i];
    aver = aver / N;

    // выборочное обнуление элементов
    for( i = 0; i < N; i++ )
    {
        if( x[i] > aver )
            x[i] = 0;
    }

    // вывод массива
    for( i = 0; i < N; i++ )
        printf("%d ", x[i]);
    printf("\n");

    return 0;
}
```

Заметим, что переменная aver, в которой будет накапливаться сумма элементов массива, имеет такой же тип, что и сами элементы. Это означает, что в процессе накопления возможно арифметическое переполнение этой переменной. Стандартный выход из такой ситуации состоит в том, чтобы накапливать сумму в переменной, которая имеет более широкий диапазон значений, чем элементы массива. Переполнение может возникнуть и при

вводе элементов массива, если их значения, вводимые с клавиатуры, не укладываются в диапазон целых чисел. Чтобы избежать на данном этапе обучения лишних усложнений, будем считать, что переполнения игнорируются. Хотя программист всегда обязан помнить о возможных арифметических переполнениях.

Варианты лабораторных заданий.

Задача 5.1

В массиве из 10 целых чисел найти и поменять местами минимальный и максимальный элементы.

Задача 5.2

В массиве из 10 целых чисел обнулить элементы, находящиеся между минимальным и максимальным элементами.

Задача 5.3

В массиве из 10 целых чисел обнулить все отрицательные элементы, если сумма их модулей меньше суммы положительных элементов.

Задача 5.4

В массиве из 10 целых чисел переставить элементы в зеркальном порядке.

Задача 5.5

В массиве из 10 целых чисел поделить все отрицательные элементы на максимальный элемент массива.

Задача 5.6

В массиве из 10 целых чисел все элементы, стоящие левее максимального, заменить нулями, а элементы, стоящие правее максимального, заменить максимальным элементом.

Задача 5.7

В массиве из 10 целых чисел сосчитать сумму чётных и сумму нечётных чисел. Вывести наибольшую из этих сумм.

Задача 5.8

В массиве из 10 целых чисел определить номер пары соседних элементов с максимальной суммой.

Задача 5.9

В массиве из 10 целых чисел обнулить все элементы, значение которых меньше среднего арифметического элементов данного массива.

Задача 5.10

В массиве из 10 целых чисел обнулить все отрицательные элементы, если их число меньше числа положительных элементов. В противном случае обнулить все положительные элементы.

Задача 5.11

В массиве из 10 целых чисел обнулить все нечётные элементы, если их число меньше числа чётных элементов. В противном случае обнулить все чётные элементы.

Задача 5.12

В массиве из 10 целых чисел сосчитать сумму элементов с чётными номерами и сумму элементов с нечётными номерами. Обнулить элементы с нечётными номерами, если их сумма меньше. В противном случае обнулить элементы с чётными номерами.

Задача 5.13

В массиве из 10 целых чисел обнулить первую половину, если ее сумма элементов меньше суммы элементов второй половины массива. В противном случае обнулить вторую половину массива.

Задача 5.14

В массиве из 10 целых чисел определить, составляют ли его элементы арифметическую прогрессию.

Задача 5.15

В массиве из 10 целых чисел определить, составляют ли его элементы геометрическую прогрессию.

Задача 5.16

В массиве из 10 целых чисел определить, является ли он зеркальным отображением самого себя.

Задача 5.17

В массиве из 10 целых чисел найти число с максимальной суммой цифр.

Задача 5.18

В массиве из 10 целых чисел найти среднее арифметическое цифр, из которых состоят элементы массива.

Задача 5.19

В массиве из 10 целых чисел обнулить те N соседних элементов массива, которые имеют минимальную сумму.

Задача 5.20

В массиве из 10 целых чисел поменять в зеркальном порядке элементы массива, стоящие левее максимального элемента. Элементы, стоящие правее следует обнулить.

Контрольные вопросы.

1. Что такое массив данных?
2. Начиная с какого индекса нумеруются элементы массива в языке Си?
3. Существует ли в языке Си проверка выхода значения индекса за границы массива?
4. Какие операторы целесообразно использовать для полного или частичного перебора элементов массива?
5. Можно ли в языке Си описать массив переменного размера?

Лабораторная работа №6. Двумерные массивы

Цель работы: получение навыков работы с многомерными массивами на примере двумерных массивов.

Методические указания.

Как уже было отмечено ранее, массив это совокупность однородных объектов. В частном случае каждый объект массива может также являться массивом. Подобную конструкцию можно считать двумерным массивом или матрицей. Описывая такой двумерный массив, например, строчкой

int X[10][20];

мы говорим о том, что массив X состоит из десяти элементов, причем каждый элемент - это массив из двадцати целых чисел. Формально можно считать, что описан двумерный массив, состоящий из десяти строк и двадцати столбцов. Соответственно, элемент X[k][n] можно рассматривать как число на пересечении строки с номером k и столбца с номером n. Манипулируя индексами n и k, можно обеспечить доступ к любому требуемому элементу матрицы X. Обработка двумерных массивов отличается от обработки одномерных массивов лишь тем, что перебор по всем элементам должен быть организован не как отдельный цикл, а как два вложенных цикла, например:

```
S = 0;
for( k = 0; k < 10; k++ )
{
     for( n = 0; n < 20; n++ )
     {
          S = S + X[k][n];
```

```
    }
}
```

Это означает, что для каждого значения индекса k следует перебрать все возможные значения индекса n.

Задача. В целочисленном массиве размера K на N обнулить все элементы, большие чем среднее арифметическое по массиву.

Программа, реализующая данную задачу, представлена ниже. Как и в предыдущей лабораторной работе будем считать, что возможные арифметические переполнения игнорируются.

```
#include <stdio.h>
#define K 3
#define N 4

int main( void )
{
    int x[K][N];      // массив из K на N элементов
    int aver;         // среднее арифметическое
    int i, j;

    // ввод массива
    for( i = 0; i < K; i++ )
        for( j = 0; j < N; j++ )
            scanf("%d", &x[i][j]);

    // вычисление среднего арифметического значения
    aver = 0;
    for( i = 0; i < K; i++ )
```

```
        for( j = 0; j < N; j++ )
            aver = aver + x[i][j];

    aver = aver / (K*N);

    // выборочное обнуление элементов
    for( i = 0; i < K; i++ )
    {
        for( j = 0; j < N; j++ )
        {
            if( x[i][j] > aver )
                x[i][j] = 0;
        }
    }

    // вывод массива
    for( i = 0; i < K; i++ )
    {
        for( j = 0; j < N; j++ )
            printf("%4d ", x[i][j]);
        printf("\n");
    }
    return 0;
}
```

Варианты лабораторных заданий.

Следует заметить, что задача 6.20 имеет повышенную сложность. Время её выполнения может быть увеличено.

Задача 6.1

В двумерном целочисленном массиве размера N на K обнулить все строки, находящиеся ниже минимального элемента.

Задача 6.2

В двумерном целочисленном массиве размера N на K проверить, являются ли строки арифметической прогрессией.

Задача 6.3

В двумерном целочисленном массиве размера N на K проверить, являются ли строки геометрической прогрессией.

Задача 6.4

В двумерном целочисленном массиве размера N на K обнулить строки, являющиеся зеркальным отражением самих себя.

Задача 6.5

В двумерном целочисленном массиве размера N на K поменять строку с максимальной суммой элементов со строкой с минимальной суммой элементов.

Задача 6.6

В двумерном целочисленном массиве размера N на N обнулить элементы над главной диагональю, если их сумма меньше суммы элементов под главной диагональю. В противном случае обнулить элементы под главной диагональю.

Задача 6.7

В двумерном целочисленном массиве размера N на K все отрицательные элементы заменить минимальным элементом массива, а все положительные – максимальным.

Задача 6.8

В двумерном целочисленном массиве размера N на K обнулить столбцы, в которых чётных элементов было больше чем нечётных.

Задача 6.9

В двумерном целочисленном массиве размера N на K обнулить строку с минимальной суммой элементов.

Задача 6.10

В двумерном целочисленном массиве размера N на N поменять местами строку и столбец, на пересечении которых находится максимальный элемент массива.

Задача 6.11

В двумерном целочисленном массиве размера N на K обнулить диагональ с максимальной суммой элементов. Следует рассмотреть все диагонали, а не только главные.

Задача 6.12

В двумерном целочисленном массиве размера N на K обнулить строки, среднее арифметическое которых меньше среднего арифметического по всему массиву.

Задача 6.13

В двумерном целочисленном массиве размера N на K обнулить все элементы, кроме строки, содержащей одновременно и минимальный, и максимальный элементы массива.

Задача 6.14

В двумерном целочисленном массиве размера N на K поменять местами строку, содержащую минимальный элемент массива, со строкой, содержащей максимальный элемент массива.

Задача 6.15

В двумерном целочисленном массиве размера N на K обнулить строку и столбец, на пересечении которых находится минимальный элемент массива.

Задача 6.16

В двумерном целочисленном массиве размера N на K обнулить все отрицательные элементы, если их число было меньше числа положительных элементов. В противном случае обнулить все положительные элементы.

Задача 6.17

Двумерный целочисленный массив размера N на N повернуть на 90 градусов по часовой стрелке.

Задача 6.18

В двумерном целочисленном массиве размера N на K отсортировать элементы по возрастанию.

Задача 6.19

В двумерном целочисленном массиве размера N на K упорядочить строки в порядке убывания среднего арифметического элементов строки.

Задача 6.20

Задать лабиринт в виде двумерного целочисленного массива размера N на K, где, например, стены определяются единицами, а проходы - нулями. Программа должна находить выход из заданной точки лабиринта любым способом. Вариант отображения маршрута выбирается студентом по желанию.

Контрольные вопросы.

1. Как реализовано хранение в памяти многомерных массивов?
2. Сколько байтов памяти будет отведено для хранения массива, описанного как int a[5][6] ?
3. Существует ли в языке Си ограничение на размерность массивов?

Лабораторная работа №7. Поразрядная обработка целых чисел

Цель работы: ознакомление с поразрядными операциями и сдвигами.

Методические указания.

Поразрядные (побитовые) операции языка Си относятся к набору команд, которые достаточно редко используются в программировании. Сложные математические вычисления, статистическая обработка данных, обработка символьной информации, моделирование, создание графических оконных интерфейсов - все это задачи, как правило, не требуют манипуляций с отдельными двоичными разрядами. Однако существуют задачи, решение которых в значительной степени основано на использовании поразрядных операций. Это, например, архиваторы файлов и алгоритмы сжатия аудио и видеоданных. Не следует забывать и системное программирование, где поразрядные операции также могут присутствовать.

Поразрядные операции используются только с целыми типами данных. Они применяются для того, чтобы извлечь из переменной один двоичный разряд или группу двоичных разрядов. Либо для того, чтобы изменить в переменной один разряд или группу разрядов.

Для извлечения разрядов необходимо к содержимому переменной применить команду "поразрядное логическое умножение" с подходящей маской, например:

```
x = y & mask;
```

Маска mask как раз и определяет те разряды, которые будут извлечены из переменной y. Было бы удобно задавать значение маски в двоичной форме, но такой формат не поддерживается языком Си. Поэтому удобнее всего использовать шестнадцатеричное представление маски, которое легко в уме переводится в двоичную форму. Заметим, что разряды принято

нумеровать начиная с нулевого. Таким образом, для выделения нулевого разряда маска должна получить значение 0x01. Для выделения группы от четвёртого до седьмого разряда маска должна иметь значение 0x0f0. Если номер отдельного выделяемого разряда заранее неизвестен, то маску можно сконструировать по правилу

```
mask = 1 << n;
```

где переменная n хранит номер интересующего нас разряда.

Для записи разрядов следует использовать команду "поразрядное логическое сложение", например:

```
x = x | mask;
```

В данном случае, разряды, которые были равны единице в маске, станут единичными и в переменной x. Остальные разряды переменной x не изменятся.

Если же требуется обнулить некоторые разряды в переменной, то следует использовать две поразрядных операции:

```
x = x & ~mask;
```

Поясним, что при этом происходит. Пусть маска содержит единицы в позициях, подлежащих обнулению. Тогда поразрядная операция "дополнение" создаст маску, в которой единицы стоят во всех разрядах, кроме тех, которые должны быть обнулены. И поэтому, последующее поразрядное логическое умножение обнулит требуемые двоичные разряды, не затронув остальные.

Задача. В длинном целом числе сосчитать число единичных разрядов на чётных и на нечётных позициях. Группу двоичных разрядов (чётных или нечётных) с наименьшим числом единиц необходимо обнулить.

При решении такого рода задач существует соблазн сначала распаковать двоичные разряды в одномерный массив, каждый элемент которого будет хранить единицу или ноль. Затем преобразовать массив в соответствии с условиями задачи и запаковать обратно в целое число. Такой подход имеет право на существование, но является очень плохим с точки зрения оптимальности, поскольку такого рода программа и работать будет медленнее, и код будет иметь длиннее, чем программа, основанная на поразрядных операциях.

Группа чётных разрядов состоит из разрядов с номерами 0, 2, 4, ..., поэтому для неё подходит маска 0x55555555. Соответственно для выделения группы нечётных разрядов следует использовать маску 0xaaaaaaaa.

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>
#define MASK0 0x55555555  // маска всех чётных разрядов
#define MASK1 0xaaaaaaaa  // маска всех нечётных
                          // разрядов

int main( void )
{
    long z;               // исходное число
    unsigned long y;      // беззнаковая версия
                          // исходного числа
    unsigned long mask;   // маска
    int counter0;         // счётчик чётных позиций
```

```
    int counter1;         // счётчик нечётных позиций
    int i;

    counter0 = 0;
    counter1 = 0;
    mask = 1;

    printf("z = ");
    scanf("%x", &z); // ввод в шестнадцатиричном
                     // формате
    y = (unsigned int)z;

    while( y != 0 )
    {
        counter0 += y & mask;
        y >>= 1;
        counter1 += y & mask;
        y >>= 1;
    }

    if( counter0 != counter1 )
    {
        mask =  counter0 > counter1 ? MASK1 : MASK0;
        z = z & ~mask;
    }

    printf("result: %x\n", z);

    return 0;
}
```

Варианты лабораторных заданий.

Задача 7.1

В длинном целом числе N поменять местами нулевой разряд с первым разрядом, второй разряд с третьим разрядом и т.д.

Задача 7.2

Циклически сдвинуть длинное целое число N на K разрядов влево.

Задача 7.3

Определить, является ли симметричным двоичное представление длинного целого числа N.

Задача 7.4

В длинном целом числе N поменять двоичные разряды в зеркальном порядке.

Задача 7.5

Определить положение старшей единицы в длинном целом числе N.

Задача 7.6

Определить, составляют ли байты длинного целого числа N арифметическую прогрессию.

Задача 7.7

В длинном целом числе N поменять пары двоичных разрядов в зеркальном порядке.

Задача 7.8

В длинном целом числе N числе поменять байты в зеркальном порядке.

Задача 7.9

Определить положение младшей единицы в длинном целом числе N.

Задача 7.10

В длинном целом числе N поменять тетрады (четыре соседних двоичных разряда) в зеркальном порядке.

Задача 7.11

Циклически сдвинуть длинное целое число N на K разрядов вправо.

Задача 7.12

В каждом байте длинного целого числа N поменять местами двоичные разряды в зеркальном порядке.

Задача 7.13

Определить число единиц в каждом байте длинного целого числа N.

Задача 7.14

Для длинного целого числа N определить, если возможно, номер двоичного разряда, для которого выполняется условие: число единичных разрядов справа от него равно числу единичных разрядов слева от него.

Задача 7.15 В длинном целом числе N найти длину самой большой серии из единичных разрядов.

Задача 7.16

В длинном целом числе N все серии единиц, состоящие из трёх и более единиц, заменить на нули.

Задача 7.17

В длинном целом числе N все единицы сгруппировать в левой стороне, а все нули – в правой.

Задача 7.18

В длинном целом числе N обнулить байты, содержащие менее 4 единиц.

Задача 7.19

В длинном целом числе N обнулить разряды, стоящие правее старшей единицы.

Задача 7.20

В длинном целом числе N обнулить разряды, стоящие левее младшей единицы.

Контрольные вопросы.

1. Над какими типами данных допустимы поразрядные операции?

2. Приведите таблицы истинности для известных в языке Си поразрядных логических операций.

3. Можно ли в общем случае сказать, что сдвиг вправо на один разряд эквивалентен делению на 2 и почему?
4. Можно ли в общем случае сказать, что сдвиг влево на один разряд эквивалентен умножению на 2 и почему?
5. Сколько и какие унарные поразрядные операции существуют в языке Си?

Лабораторная работа №8. Указатели

Цель работы: изучение команд адресной арифметики.

Методические указания.

Указатели и адресная арифметика это один из краеугольных камней, обеспечивающий высокое быстродействие программ, написанных на языке Си. Известно, что адресную арифметику следует применять только для обработки массивов. На момент создания первых компиляторов языка Си грамотное использование указателей позволяло ускорить некоторые фрагменты программ в несколько раз, по сравнению с алгоритмами, основанными на использовании индексов при доступе к элементам массива. Современные оптимизирующие компиляторы существенно снизили этот выигрыш. И даже если в программе используются индексы элементов массива, оптимизирующий компилятор неявно для программиста строит машинный код, основанный на использовании указателей. Более того, в некоторых случаях явное использование указателей может привести к проигрышу в быстродействии.

Целью данной лабораторной работы является ознакомление с командами адресной арифметики и написание, по возможности, наиболее оптимального с точки зрения быстродействия кода программы.

Напомним, что указатель, например ptr, - это переменная, которая хранит адрес другой переменной, например X. Настроить указатель на переменную можно с помощью операции

```
ptr = &X;
```

Теперь извлечь содержимое переменной X, используя указатель, можно с помощью операции косвенного доступа

Y = *ptr;

Напомним, что команды адресной арифметики в качестве операндов используют адрес переменной, на которую они ссылаются, а не содержимое этой переменной. Сама адресная арифметика допускает только четыре действия с указателями:

- сложение указателя с целочисленной величиной;
- автоинкремент (автоувеличение) и автодекремент (автоуменьшение) указателя;
- сравнение указателей;
- вычитание указателей.

Задача. В символьной строке удалить слова, заканчивающиеся на две одинаковых буквы.

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>
// определение символических констант
#define YES   1
#define NO   0
#define MAXLINE 1000

void process_line( char buffer[] );

int main( void )
{
    char line[MAXLINE];

    gets( line );
```

```
    process_line( line );
    puts( line );

    return 0;
}

void process_line( char buffer[] )
{
    char c;       // текущий символ
    int prev_c;   // предыдущий символ
    int flag;     // признак слова
    int found;    // индикатор того, что искомый признак
                  // в слове обнаружен
    char *in_ptr;     // указатель на текущий символ
                      // входной строки
    char *out_ptr;    // указатель на текущий символ
                      // выходной строки
    char *word_ptr;   // указатель на начало слова

    // начальные присваивания (инициализация)
    flag = NO;
    found = NO;
    prev_c = ' ';
    word_ptr = buffer;
    in_ptr   = buffer;
    out_ptr  = buffer;

    // цикл чтения символов из строки
    do
    {
```

```
c = *in_ptr; // взять текущий символ из буфера

if( c == ' ' || c == '.' || c == ',' ||
    c == '\n' || c == '\0')
{
    // найден разделитель
    if( flag == YES )
    {
        // это первый разделить после слова

        // проверить, обнаружен ли в слове
        // искомый признак
        if( found == NO )
        {
            // слово не подлежит удалению,
            // оно копируется в результирующую
            // строку
            while( word_ptr < in_ptr )
                *out_ptr++ = *word_ptr++;
        }
    }
    flag = NO;
    *out_ptr++ = c;
}
else
{
    // найдена буква

    if( flag == NO )
```

```
                word_ptr = in_ptr; // запомнить адрес
                                   // начала слова

            // проверка совпадения текущего и
            // предыдущего символов
            // (обнаружение искомого признака в слове)
            if( prev_c == c )
                found = YES;
            else
                found = NO;

            flag = YES;
        }

        prev_c = c;
        in_ptr++;    // продвинуть вперёд указатель на
                     // текущий символ
    }
    while( c != '\0' );
}
```

Варианты лабораторных заданий.

В качестве задач для самостоятельного решения следует использовать задачи из лабораторной работы "Обработка символьных строк", добавив дополнительное условие - обращение к элементам массива следует реализовать с помощью указателей, а не индексов. Кроме этого желательно, чтобы в тексте программы были использованы все четыре возможных команды адресной арифметики. При этом сохраняется запрет на использование библиотечных функций обработки символьных строк.

Контрольные вопросы.

1. Что такое указатель?
2. Как настроить указатель на адрес соответствующей переменной?
3. Как извлечь содержимое переменной, на которую указывает соответствующий указатель?
4. Какие операции входят в состав адресной арифметики?
5. С помощью каких операций можно продвинуть указатель на следующий элемент массива?
6. Что получается в результате вычитания двух указателей, настроенных на разные элементы одного и того же массива?

Лабораторная работа №9. Работа с файлами

Цель работы: получение навыков чтения и записи информации из файлов.

Методические указания.

В языке Си существует два стиля работы с файлами: либо с помощью файлового указателя, либо с помощью файлового дескриптора. Последний способ иногда называют низкоуровневым доступом к файлам. Он чуть более сложен, чем первый, но зато обладает более широкими возможностями. В этой лабораторной работе предполагается знакомство с доступом к файлам посредством файлового указателя.

Файловый указатель это указатель вида

```
FILE *fp;
```

где FILE это структура, описанная в одном из стандартных заголовочных файлов языка Си.

В программе можно одновременно работать с неограниченным количеством файлов, но каждый из них должен быть сопоставлен со своим файловым указателем. Для этого файл надо открыть с помощью библиотечной функции fopen(), указав в качестве параметров имя файла и способ его использования (чтение или запись). Перед выходом из программы с помощью функции fclose() следует закрыть каждый открытый файл. В принципе, при выходе из программы операционная система самостоятельно закроет все открытые файлы, так что файлы, открытые для чтения, можно не закрывать. А вот файлы, открытые для записи, закрывать строго необходимо, иначе может потеряться часть содержимого этих файлов.

После того, как файл успешно открыт, можно читать из него или писать в него информацию, используя стандартные библиотечные функции, например: fgets(), fputs(), fread(), fwrite(), fscanf(), fprintf() и т.д.

Важным моментом при чтении из файла является организация цикла чтения, пока содержимое файла не будет прочитано полностью. Для этого,

как правило, используется библиотечная функция feof(). Она возвращает нулевое значение до тех пор, пока файл не исчерпан, и единичное значение, если достигнут конец файла.

Задача. Скопировать содержимое текстового файла, удалив все слова, кроме первого слова каждой строки .

Программа, реализующая данную задачу, представлена ниже.

```
#include <stdio.h>
#define MAXLINE 1024

int main( void )
{
    // указатели на структуру типа FILE для
    // входной и выходного файлов
    FILE *fpin;
    FILE *fpout;

    char line[MAXLINE];   // текущая строка
    char *ptr;

    // открыть файл для чтения
    fpin = fopen( "test.txt", "rt" );
    if( fpin == NULL )
        return;      // ошибка при открытии файла

    // открыть файл для записи
    fpout = fopen("result.txt", "wt" );
    if( fpout == NULL )
```

```
    return;      // ошибка при открытии файла

while( !feof( fpin ) )// цикл до конца файла
{
    // чтение строки
    ptr = fgets( line, MAXLINE, fpin );

    if( ptr == NULL )
        break;   // файл исчерпан

    while( *ptr != '\0' ) // цикл до конца строки
    {
        if( *ptr == ' ' || *ptr == '.' ||
            *ptr == ',' )
        {
            // найден разделитель слова
            *ptr++ = '\n'; // ставим символ "конец
                           //строки"
            *ptr = '\0';   // ставим ограничитель
                           //строки
            break;
        }

        ptr++;   // продвигаем указатель по строке
    }
    fputs( line, fpout ); // запись строки
}

fclose( fpin );  // закрыть входной файл
fclose( fpout ); // закрыть выходной файл
```

```
        return 0;
}
```

Варианты лабораторных заданий.

Задание 9.20 следует выдавать только по желанию.

Задача 9.1
Скопировать содержимое текстового файла, удалив строку с наибольшим количеством слов.
Задача 9.2
Скопировать содержимое текстового файла, удалив в каждой строке лишние пробелы.
Задача 9.3
Скопировать содержимое текстового файла, ограничив длину строки N символами. Слова, не помещающиеся в строку заданной длины, не копировать.
Задача 9.4
Скопировать содержимое текстового файла, удалив в каждой строке самое длинное слово.
Задача 9.5
Скопировать содержимое текстового файла, удалив в каждой строке слова четной длины.
Задача 9.6
Скопировать содержимое текстового файла, удалив в каждой строке предпоследнее слово.
Задача 9.7
Скопировать содержимое текстового файла, удалив все числа.

Задача 9.8

Скопировать содержимое текстового файла, удалив в каждой строке слово номер N.

Задача 9.9

Скопировать содержимое текстового файла, ограничив длину строки N символами. Слова, не помещающиеся в строку заданной длины, скопировать в виде новой строки.

Задача 9.10

Скопировать содержимое текстового файла, удалив строку с наибольшим числом различных символов.

Задача 9.11

Скопировать содержимое текстового файла, заменив все цифры на их словесный эквивалент.

Задача 9.12

Скопировать содержимое текстового файла, удалив строку, содержащую самое длинное слова.

Задача 9.13

Скопировать содержимое текстового файла, удалив строку с максимальным количеством слов.

Задача 9.14

Скопировать содержимое текстового файла, удалив строку, содержащую самое большое число, если такая строка имеется.

Задача 9.15

Скопировать содержимое текстового файла, отформатировав каждую строку по ширине с помощью равномерного добавления пробелов между словами.

Задача 9.16

По текстовому файлу составить статистику длин строк и вывести строки в порядке возрастания их длин.

Задача 9.17

По текстовому файлу составить статистику частоты появления символов и вывести символы в порядке убывания их популярности.

Задача 9.18

По текстовому файлу составить статистику длин слов и вывести слова в порядке возрастания их длин.

Задача 9.19

Скопировать содержимое текстового файла, удаляя комментарии, организованные по правилам языка Си.

Задача 9.20

По текстовому файлу составить словарь и вывести его в алфавитном порядке.

Контрольные вопросы.

1. Какой тип имеет файловый указатель?
2. Какая функция должна быть использована для открытия файла?
3. Почему файлы, открытые для записи, обязательно должны быть закрыты перед выходом из программы?
4. Чем текстовый режим, задаваемый при открытии файла, отличается от двоичного режима?
5. Чем функции двоичного ввода/вывода в файл отличаются от функций форматированного ввода/вывода?
6. Как организовать чтение из файла до окончания его содержимого?

Лабораторная работа №10. Динамические массивы

Цель работы: изучение методов организации массивов переменной размерности и работы с ними.

Методические указания.

В языке Си объявление массива подразумевает, что в явном виде должен быть указан его размер. В программировании такого рода массивы принято называть статическими. Не надо путать со статическими переменными, которые объявляются при помощи ключевого слова static. Чтобы избежать путаницы в терминах заметим, что статический массив может быть автоматической переменной, если он объявлен внутри какой-либо функции. Если статический массив объявлен вне функций, то он является внешней переменной, а если он объявлен с ключевым словом static, то является статической переменной. Разница между автоматической, внешней или статической переменной состоит в том, что они располагаются в разных сегментах памяти и имеют разное время жизни и область видимости. Но в любом случае их размер определяется на этапе компиляции и не может быть изменён во время выполнения задачи. Именно это и является главным недостатком статических массивов. Если программисту заранее не известен объём данных, которые будет обрабатывать его программа, то приходится перестраховываться и объявлять массивы максимально возможного размера, даже если в процессе работы из него будет задействована лишь ничтожная часть. Во-первых, это приводит к неразумному использованию памяти. Во-вторых, если массив является автоматическим, то он будет расположен в стеке, и следовательно не может превышать размеры стека. Если же массив является внешней или статической переменной, то его размер будет влиять на размер построенной программы.

Альтернативой статическому массиву является динамический массив, то есть массив, размер которого можно задавать в процессе выполнения задачи, исходя, скажем, из введённых в программу данных. Более того, размер массива можно изменять в процессе решения задачи. На языке Си динамический массив можно создать с помощью указателя, соответствующего типа, настроив его на динамически выделенную область памяти. Это можно сделать с помощью библиотечных функций malloc или calloc, причём последняя не только выделит запрашиваемую память, но и обнулит её, например:

```
int *array = (int*)malloc( n * sizeof(int) );
char *buffer = (char*)calloc( n, sizeof(char) );
```

Обращение к i-му элементу динамической памяти можно реализовать по правилам работы с указателями:

```
*(array + i) = 0;
*(buffer+k) = 'a';
```

Или, учитывая тесную связь указателей и массивов в синтаксисе языка Си:

```
array[i] = 0;
buffer[k] = 'a';
```

Теперь переменные array и buffer действительно выглядят массивами, а не указателями.
Динамический массив обязательно надо освобождать по выходе из программы:

```
free( arrar);
```

free(buffer);

Неосвобождённые по выходе из программы массивы приводят к тому, что распределённая для них память не может быть использована другими задачами. Такая ситуация называется "утечкой памяти".

По мере необходимости размер динамического массива можно изменять с помощью функции realloc(). Если размер динамического массива увеличивается, то всё содержимое старого массива автоматически копируется в новый массив. Именно по этому принято считать функцию realloc() достаточно затратной по времени.

Задача. Написать функцию, которая сосчитает количество слов в массиве строк.

Для тестирования функции будем использовать файл неизвестного содержания и размера. Программа будет содержать две похожих функции. Одна будет работать со статическим массивом строк, а вторая - с динамическим массивом строк.

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

#define YES  1
#define NO   0

#define MAXLEN   1000// максимальная длина строки
#define NLINES   1000  // максимальное число строк
```

```
int word_counter0( char lines[][MAXLEN], int n );
int word_counter1( char **lines, int n );

int main( void )
{
    FILE *fp;    // входной файл
    int nlines0, nlines1; // число строк в файле
    static char lines[NLINES][MAXLEN]; // статический
                                       // массив для
                                       // хранения
                                       // строк
    char **lines_ptr = NULL;           // указатель на
                                       // динамический
                                       // массив
    int npointers;        // число указателей в
                          // динамическом массиве
    int i;
    int nwords0, nwords1; // число слов в массиве строк

    // открыть файл для чтения
    fp = fopen( "test.txt", "rt" );
    if( fp == NULL )
        return;  // ошибка при открытии файла

    // заказать и обнулить динамический массив
    // указателей (два варианта)
#if 1
    lines_ptr = (char**)malloc( NLINES*sizeof(char*) );
    // обнулить массив указателей
```

```
    memset( lines_ptr, 0, NLINES*sizeof(char*) );
#else
    lines_ptr = (char**)calloc( NLINES, sizeof(char*));
#endif
    npointers = NLINES;

    nlines0 = 0;
    nlines1 = 0;
    while( !feof( fp )    )    // цикл до конца файла
    {
        char line[MAXLEN];
        // чтение строки
        char *ptr = fgets( line, MAXLEN, fp );
        int len;

        if( ptr == NULL )
            break;   // файл исчерпан

        if( nlines0 < NLINES )
        {
            // статический массив ещё не исчерпан

            // сохранить строку в статическом массиве
            strcpy( &lines[nlines0][0], line );
            nlines0++;
        }

        // определить длину строки
        len = (int)strlen( line );
        if( nlines1 == npointers )
```

```
        {
            // динамический массив исчерпан
            npointers += NLINES;// увеличить число
                                //указателей в массиве

            // перезаказать память для массива
            // указателей
            lines_ptr = (char**)realloc( lines_ptr,
                        npointers * sizeof(char*) );

            // обнулить новую часть массива указателе
            memset( &lines_ptr[nlines1], 0,
                    NLINES * sizeof(char*) );
        }

        // заказать в памяти место для строки
        // плюс один байт
        lines_ptr[nlines1] = (char*)malloc( len+1 );

        // сохранить строку в динамическом массиве
        strcpy( lines_ptr[nlines1], line );
        nlines1++;
    }

    nwords0 = word_counter0( lines, nlines0 );
    nwords1 = word_counter1( lines_ptr, nlines1 );

    printf("number of words: %d\n", nwords0 );
    printf("number of words: %d\n", nwords1 );
```

```
    // отказаться от памяти для строк
    for( i = 0; i < nlines1; i++ )
    {
        if( lines_ptr[i] != NULL )
            free( lines_ptr[i] );
    }

    // отказаться от динамического массива указателей
    if( lines_ptr != NULL )
        free( lines_ptr );

    fclose( fp );         // закрыть входной файл

    return 0;
}



int word_counter0( char lines[][MAXLEN], int n )
{
    int i, j;
    int counter; // счётчик слов
    int flag;    // признак слова

    counter = 0;
    flag = NO;

    for( i = 0; i < n; i++ )
    {
        // цикл по строкам
```

```
        char *ptr = &lines[i][0]; // указатель на
                                  // текущую строку
        for( j = 0; ptr[j] != '\0'; j++ )
        {
            if( ptr[j] == ' ' || ptr[j] == ',' ||
                ptr[j] == '.' || ptr[j] == '\n' )
            {
                // текущий символ - разделитель
                flag = NO;
            }
            else
            {
                // текущий символ - буква
                if( flag == NO )
                {
                    // первая буква в слове
                    counter++;
                }
                flag = YES;
            }
        }
        if( flag == YES )
        {
            // строка закончилась буквой
            counter++;
        }
    }

    return counter;
}
```

```
int word_counter1( char **lines, int n )
{
    int i, j;
    int counter; // счётчик слов
    int flag;    // признак слова

    counter = 0;
    flag = NO;

    for( i = 0; i < n; i++ )
    {
        // цикл по строкам

        // получить адрес текущей строки
        // ( два варианта )
#if 1
        char *ptr = *(lines + i); // указатель на
                                  // текущую строку

#else
        char *ptr = lines[i]; // указатель на текущую
                              // строку
#endif

        for( j = 0; ptr[j] != '\0'; j++ )
        {
            if( ptr[j] == ' ' || ptr[j] == ',' ||
                ptr[j] == '.' || ptr[j] == '\n' )
            {
```

```
                // текущий символ - разделитель
                flag = NO;
            }
            else
            {
                // текущий символ - буква
                if( flag == NO )
                {
                    // первая буква в слове
                    counter++;
                }
                flag = YES;
            }
        }
        if( flag == YES )
        {
            // строка закончилась буквой
            counter++;
        }
    }

    return counter;
}
```

Функции word_counter0() и word_counter1() практически идентичны за исключением интерфейса. Функция word_counter0() работает со статическим массивом строк или, что тоже самое, со статическим двумерным массивом символов. Массив передается посредством параметра char lines[][MAXLEN], причём по правилам языка Си первую размерность указывать необязательно. Функция word_counter1() работает с динамическим массивом строк, который можно интерпретировать как динамический двумерный массив символов.

Причем длина строк этого двумерного массива различна. Динамический массив передаётся посредством параметра char **lines. Это так называемый "двойной указатель". Вспомнив о тесной связи массивов и указателей, этот параметр можно привести к виду char *lines[]. Теперь всё становится понятней - lines это массив, каждый элемент которого является указателем на символ. Другая интерпретация - lines это массив, каждый элемент которого является указателем на символьную строку. Таким образом становится понятной структура динамических данных и стратегия работы с ней. В вызывающей программе заказывается динамический массив указателей lines_ptr, содержащий NLINES элементов.

Считав строку из файла, программа определяет её длину и заказывает динамическую строку соответствующего размера, записывая указатель на неё в соответствующий элемент массива lines_ptr. Если обнаруживается, что массив указателей уже заполнен целиком, то его размер увеличивается с помощью функции realloc. Таким образом программа может обрабатывать файлы любой длины. Можно говорить о том, что размер файла ограничен размером оперативной памяти компьютера.

Для статического двумерного массива ситуация складывается гораздо хуже. Не зная число строк и размер самой длинной строки, мы вынуждены заказывать статическую память по максимуму. За это в программе отвечают константы MAXLEN и NLINES. И если файл содержит больше чем NLINES строк, то некоторое их количество после прочтения будет проигнорировано.

Варианты лабораторных заданий.

Все таблицы должны быть организованы как динамические массивы.

Задача 10.1

По файлу с программой на Си построить дерево вызовов функций.

Задача 10.2

Создать таблицу имён внешних переменных по группе файлов, содержащих модули проекта на языке Си.

Задача 10.3

Создать таблицу внутренних переменных по файлу с программой на языке Си.

Задача 10.4

Скопировать текстовый файл, зеркально поменяв порядок строк.

Задача 10.5

Скопировать текстовый файл, зеркально поменяв порядок слов.

Задача 10.6

Создать список интерфейсов функций, определённых в файле с программой на языке Си.

Задача 10.7

Скопировать текстовый файл, отсортировав строки по алфавиту.

Задача 10.8

Скопировать текстовый файл, отсортировав строки по числу слов в строке.

Задача 10.9

По содержимому текстового файла составить словарь (в алфавитном порядке).

Задача 10.10

Скопировать содержимое текстового файла, удаляя строки, если они уже встречались ранее.

Задача 10.11

По содержимому текстового файла составить список предложений (в алфавитном порядке).

Контрольные вопросы.

1. Какие функции следует использовать в процессе выполнения задачи,

чтобы заказать в памяти область заданного размера для размещения данных.

2. Какая функция освобождает заказанную область памяти?

3. Для чего в конце программы следует обязательно освободить всю динамически распределённую память?

4. В чём основные отличия статического двумерного массива от динамического двумерного массива?

5. Чем ограничивается размер динамического массива?

Лабораторная работа №11. Синтаксический разбор текста

Цель работы: закрепление знаний об основных конструкциях языка Си, закрепление навыков самостоятельной разработки алгоритмов и выбора инструментария для их реализации.

Методические указания.

Данная лабораторная работа является последней в цикле изучения языка программирования Си. При её выполнении автор должен показать, насколько хорошо он владеет основными конструкциями языка. Все задания связаны с обработкой текстовой информации, при этом разрешается использовать любые библиотечные функции, входящие в стандарт языка Си.

Задача. Вывести на экран содержимое файла выделяя в нем слова.

Самый наглядный способ выделения фрагментов текста основан на изменении цвета символов на экране монитора. По умолчанию вывод в консольное окно Windows предполагает черный цвет фона и белый цвет символов. Для изменения этих цветов необходимо выполнить два шага: с помощью библиотечной функции GetStdHandle() получить у операционной системы дескриптор окна консоли, изменить цвета фона и символа этого окна с помощью функции SetConsoleTextAttribute(). В программе, представленной ниже, показано как это правильно сделать. Цветовая палитра для консоли, предлагаемая операционной системой Windows достаточно скудна. На цвет фона и цвет символа отводится по четыре двоичных разряда, при этом в каждой четверке один разряд отвечает за яркость, а три - за цветность. Эти двоичные разряды определяются константами

```
FOREGROUND_INTENSITY - интенсивность символа;
FOREGROUND_BLUE      - синий цвет символа;
FOREGROUND_GREEN     - зелёный цвет символа;
FOREGROUND_RED       - красный цвет символа;
```

```
BACKGROUND_INTENSITY - интенсивность фона;
BACKGROUND_BLUE      - синий цвет фона;
BACKGROUND_GREEN     - зелёный цвет фона;
BACKGROUND_RED       - красный цвет символа.
```

Любая из восьми комбинаций цветов символа или фона является допустимой, а сами комбинации получаются с помощью операции логического сложения базовых цветов.

В представленной ниже задаче выделение слов осуществляется следующим образом. Информация читается из файла построчно. Поток символов классифицируется на буквы, которые собираются в слова в массиве word, или на разделители, которые собираются в том же массиве. Как только появляется разделитель, цвет фона и цвет символов меняется на цвета, соответствующие словам, и на экран выводится накопленное содержимое массива word, который содержит слово, предшествующее принятому разделителю. После этого массив подготавливается к приёму разделителей. Соответственно, как только появляется символ, цвет фона и цвет символов меняется на цвета, соответствующие разделителям, и на экран выводится накопленное содержимое того же самого массива. Только теперь этот массив содержит последовательность разделителей, предшествующих принятому символу.

В конце программы восстанавливаются стандартные цвета консольного окна.

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

#define LETTER   1
#define NOT_THE_LETTER    0
```

```
#define MAXLEN   1000

int main( void )
{
    HANDLE hStdout;           // дескриптор консольного
                              // окна
    FILE *fp;                 // входной файл
    char line[MAXLEN];        // строчка из файла
    char word[MAXLEN];        // слово
    int i;
    int flag;             // флаг слово/разделитель
    int prev_flag;        // предыдущее значение флага
    char *ptr;            // указатель на строчку
    WORD foregroundColor0;    // цвет слов
    WORD foregroundColor1;    // цвет разделителей
    WORD foregroundColor;     // цвет символа
    WORD backgroundColor;     // цвет фона
    WORD textAttribute;       // атрибут текста - цвет
                              // символа и фона

    // Получить стандартный дескриптор
    hStdout= GetStdHandle (STD_OUTPUT_HANDLE) ;

    // определить цвета символов, разделителей и фона
    foregroundColor0 = FOREGROUND_INTENSITY |
                       FOREGROUND_BLUE |
                       FOREGROUND_GREEN /*|
                       FOREGROUND_RED*/;
    foregroundColor1 = FOREGROUND_INTENSITY |
                       FOREGROUND_RED;
```

```
backgroundColor = BACKGROUND_INTENSITY |
                  BACKGROUND_BLUE |
                  BACKGROUND_GREEN |
                  BACKGROUND_RED;

SetConsoleTextAttribute (hStdout,
        foregroundColor0 | backgroundColor );

// открыть файл для чтения
fp = fopen( "test.txt", "rt" );
if( fp == NULL )
    return;      // ошибка при открытии файла

while( !feof( fp ) )
{
    ptr = fgets( line, MAXLEN, fp );

    if( ptr == NULL )
        break;

    i = 0;
    prev_flag = flag = NOT_THE_LETTER;

    word[0] = '\0';

    while( *ptr != 0 )
    {
        prev_flag = flag;// запомнить флаг
```

```
        if( *ptr == ' ' || *ptr == ',' ||
            *ptr == '.' || *ptr == '\n' )
            flag = NOT_THE_LETTER;
        else
            flag = LETTER;

        if( flag != prev_flag )
        {
            // флаг изменился - слово закончилось
            word[i] = '\0';  // закрыть слово

            // установить цвет символа (буква или
            // разделитель)
            foregroundColor =
                (prev_flag == LETTER) ?
                foregroundColor0 :
                foregroundColor1;
            textAttribute =
                foregroundColor | backgroundColor;
            SetConsoleTextAttribute (hStdout,
                                textAttribute );

            printf( "%s", word ); // вывести слово
            i = 0;           // начать новое слово
        }

        word[i++] = *ptr++;   // запомнить символ
    }
```

```
            // проверить, выведено ли последнее слово в
            // строке
            if( i != 0 )
            {
                word[i] = '\0';  // закрыть слово

                // установить цвет символа (буква или
                // разделитель)
                foregroundColor =
                            (prev_flag == LETTER) ?
                            foregroundColor0 :
                            foregroundColor1;
                textAttribute =
                    foregroundColor | backgroundColor;
                SetConsoleTextAttribute (hStdout,
                                textAttribute );

                printf( "%s", word ); // вывести слово
            }
        }
        printf("\n");

        // белые символы, черный фон
        SetConsoleTextAttribute (hStdout, 7 );

        return 0;
}
```

Варианты лабораторных заданий.

Задача 11.1

Вывести на экран содержимое файла с программой на языке С, выделяя в строке ключевые слова.

Задача 11.2

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все целые константы.

Задача 11.3

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все строковые константы.

Задача 11.4

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все директивы препроцессора.

Задача 11.5

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все макроимена.

Задача 11.6

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все идентификаторы.

Задача 11.7

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все внешние переменные.

Задача 11.8

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все обращения к функциям.

Задача 11.9

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все идентификаторы массивов.

Задача 11.10

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все заголовки функций.

Задача 11.11

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все восьмеричные константы.

Задача 11.12

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все шестнадцатеричные константы.

Задача 11.13

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все вещественные константы.

Задача 11.14

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все комментарии, включая многострочные.

Задача 11.15

Вывести на экран содержимое файла с программой на языке С, выделяя в строке все описания и объявления переменных.

Список литературы

1. Керниган, Б. В. Язык программирования Си / Б. Керниган, Д. Ритчи; пер. с англ. под ред. Вс. С. Штаркмана. - 3. изд., испр. - СПб. : Нев. диалект, 2001. - 351 с.

2. Кучин,Н.В. Основы программирования на языке СИ : Учеб. пособие / Н. В. Кучин, М. М. Павлова; М-во образования Рос. Федерации. С.-Петерб. гос. ун-т аэрокосм. приборостроения. - СПб. : СПбГУАП, 2001. - 85 с.

3. Подбельский, В. В.. Программирование на языке Си : Учеб. пособие для студентов вузов, обучающихся по направлениям: "Прикл. математика и информатика". "Информатика и вычисл. техника", специальностям "Прикл. математика" и "Вычисл. машины, комплексы, системы и сети упр." / В. В. Подбельский, С. С. Фомин. - 2. изд., доп. - М. : Финансы и статистика, 2001. - 600 с.

4. Kernigan, B.W. The C programming language / B. Kernigan, D. Ritchie. - 2nd edition. - New Jersey: Prentice Hill PRT, 1988. - 272p.