

Федеральное агентство по образованию
Государственное образовательное учреждение высшего
профессионального образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**С. И. Борисов
М. А. Песков**

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ
ПРОГРАММИРОВАНИЕ
часть 1**

Учебно-методическое пособие

Томск – 2006

Борисов С. И., Песков М. А.

Объектно-ориентированное программирование. Часть 1 : учеб. метод. пособие / С. И. Борисов, М. А. Песков. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2006. – 43 с.

В первой части учебно-методического пособия «Объектно-ориентированное программирование» студентам предлагается выполнить лабораторные работы по основам программирования на языке C++. Эти лабораторные работы иллюстрируют основные принципы построения сравнительно небольших структурированных программ. В качестве окружения языка C++ используются стандартные библиотеки потокового ввода-вывода этого языка.

Лабораторные работы построены по принципу нарастания сложности, однако, включают ряд заданий (не обязательных), требующих нестандартного, творческого подхода для их решения. Это позволяет использовать данные учебно-методические пособия в учебном процессе для студентов разного начального уровня подготовки.

Пособие предназначено для студентов высших технических учебных заведений.

© Борисов С. И., Песков М. А., 2006

© Том. гос. ун-т систем упр. и
радиоэлектроники, 2006

Оглавление

1	Простейшие программы.....	6
1.1	Первая программа	6
1.2	Сложение двух целых чисел	7
1.3	Вывести на экран сумму чисел	7
1.4	Ввести с клавиатуры целое число	8
1.5	Варианты индивидуальных заданий	8
1.5.1	Вычисление длины окружности.....	8
1.5.2	Поиск максимума из двух чисел	9
1.5.3	Поменять значения двух переменных	9
1.5.4	Вычислить корень линейного уравнения.....	10
1.6	Вычислить корни квадратного уравнения.....	10
2	Функции.....	10
2.1	Функция «максимум»	10
2.2	Варианты индивидуальных заданий	11
2.2.1	Четность числа	11
2.2.2	Високосный год	11
2.2.3	Пробельный символ	12
2.2.4	Корень линейного уравнения	12
2.3	Корни квадратного уравнения	13
2.4	Числа Фибоначчи	14
3	Массивы.....	15
3.1	Печать массива	15
3.2	Функция печати массива	15
3.3	Варианты индивидуальных заданий	16
3.3.1	Поиск максимума.....	16
3.3.2	Сумма массива	16
3.3.3	Среднее арифметическое значение массива	17
3.3.4	Сортировка массива	17

3.4	Разупорядочивание массива	18
3.5	Простые числа	18
3.6	Многомерные массивы	18
3.7	Точное значение числа Фибоначчи	19
4	Строки и указатели	21
4.1	Длина строки	21
4.2	Копирование строки	22
4.3	Конкатенация строк	23
4.4	Количество пробелов	23
4.5	Пробельный символ	24
4.6	Варианты индивидуальных заданий	24
4.6.1	Количество пробельных символов	24
4.6.2	Удаление пробельных символов	24
4.6.3	Удаление дублирующихся пробельных символов	24
4.6.4	Проверка палиндрома	25
4.7	Сравнение строк	25
4.8	Печать строки в кодах символов	25
4.9	Замена символа табуляции на пробелы	25
4.10	Замена пробелов на символ табуляции	26
4.11	Модуль str	26
5	Структуры, объединения, перечисления и синонимы	30
5.1	Перечисления	30
5.2	Булев тип данных	30
5.3	Варианты индивидуальных заданий	30
5.3.1	Структуры: point	30
5.3.2	Структуры: person	32
5.4	Объединения	33
6	Динамические структуры данных	34
6.1	Обработка массива	34

6.2	Варианты индивидуальных заданий	35
6.2.1	Организация динамических массивов	35
6.2.2	Организация списков.....	37
7	Файловый ввод-вывод.....	41
7.1	Конвертация файла	41
7.2	Варианты индивидуальных заданий	42
7.2.1	Запись таблицы на диск/чтение таблицы с диска	42
7.2.2	Запись списка на диск/чтение списка с диска	43

1 Простейшие программы

1.1 Первая программа

Напишем первую программу на языке Си.

```
1 #include <iostream.h>
2 int main()
3 {
4     cout << "Hello world!" << endl;
5     return 0;
6 }
```

В строке 1 данной программы при помощи директивы препроцессора `include` подключается заголовочный файл `iostream.h`. В данном файле описаны функции по вводу и выводу на экран и в файлы. Подробнее о директивах препроцессора рассмотрено соответствующий раздел в курсе лекций.

В строке 2 описана функция с именем `main`, возвращающая пустой тип данных `int` – целое число (для чего нужно это значение будет рассмотрено позже), список входных параметров функции пуст (список параметров заключен в круглые скобки).

В строке 3 записана открывающая, а в строке 6 закрывающая фигурные скобки, внутри которых записывается тело функции.

В строке 4 записано собственно тело функции – в данном случае состоящее всего из двух операторов вывода на экран строки `"Hello world!"` и перевода строки

Наберите исходный текст данной программы в файле `first.cpp`, скомпилируйте и запустите ее. Если результат выполнения программы быстро появился в черном окне, которое сразу же исчезло, то это означает, что Ваша Windows так настроена. Для того, чтобы увидеть результат рабо-

ты Вашей программы можно поступить одним из двух нижеследующих способов.

Запускать программу из командной строки или консольной оболочки, например FAR.

Поставить в конце главной функции (перед `return ...`) вызов функции `getch()` – получение символа из консоли с ожиданием, эта функция описана в заголовочном файле `conio.h`, который Вам соответственно необходимо подключить. В этом случае Ваша программа будет ждать от Вас нажатия любой клавиши после выполнения своей задачи.

1.2 Сложение двух целых чисел

В новом файле (`add2.cpp`), на основе программы `first.cpp` создадим новую программу. По сравнению с предыдущей в ней изменится только содержимое функции `main`. Внутри функции `main` (вместо строки 4) зададим три переменные целого типа:

```
7 int a,b,c;
```

Присвоим переменным `a` и `b` начальные значения. Например:

```
8 a = 13;
```

```
9 b = 5;
```

И запишем в `c` сумму двух предыдущих переменных:

```
10 c = a+b;
```

Данная программа, если Вы ее запустите, ничего не выдаст на экран, так как нет печати результата. В следующей программе мы сделаем печать результата на экране

1.3 Вывести на экран сумму чисел

В файле `add3.cpp` создадим новую программу на основе программы `first.cpp`. Ее отличия будут только в записи печати на экране:

передадим в `cout` выражение типа следующего: `5+7`, после чего `endl` так же как и в прошлый раз. После запуска программы на экране появится результат введенного выражения.

Попробуйте подставить другие числа и другие операции, такие как умножить (*), разделить (/) и так далее.

1.4 Ввести с клавиатуры целое число

На основе программы `add2.cpp` создадим новую программу, `add4.cpp`. Модифицируем данную программу следующим образом: вместо присваивания начальных значений переменным `a` и `b` в строках 7, 8 сделаем ввод значений данных переменных с клавиатуры:

```
11 cin >> a;
```

```
12 cin >> b;
```

Все остальное оставим без изменений. После запуска данной программы необходимо ввести с клавиатуры два целых числа, разделив их символом пробел, табуляция или перевод строки. Затем, после ввода чисел и нажатия клавиши `<Enter>` программа выведет на печать сумму введенных чисел.

1.5 Варианты индивидуальных заданий

1.5.1 Вычисление длины окружности

В файле `mulf1.cpp` модифицируем предыдущую программу следующим образом: необходимо ввести с клавиатуры одно вещественное число (`float`). Вывести на экран данное число, помноженное на константу `2*3.14159`. Ввод и вывод вещественных (`float`) чисел осуществляется аналогично целым.

1.5.2 Поиск максимума из двух чисел

В файле `max1.cpp` создайте программу выбора максимума из двух чисел. Для данной программы необходимо использовать условный оператор `if`. В круглых скобках в данном операторе записывается целочисленное выражение, например: $(a > b)$. Если результат данного выражения будет отличен от нуля, то будут выполнены оператор, следующий сразу за скобками выражения, в противном случае будет выполнен оператор указанный после ключевого слова `else`. У операций сравнения результат равен 1, в случае истинности выражения и 0 в противном случае. Таким образом, если значение переменной `a` будет больше, чем значение переменной `b`.

1.5.3 Поменять значения двух переменных

В файле `swap.cpp` создайте программу, вводящую значения двух переменных, поменяйте местами значения этих двух переменных и выведите их на экран.

Для того, чтобы поменять местами две переменные необходимо использовать третью. Представьте, что у вас есть два стакана, в одном из них налито молоко, а в другом вода. Теперь, пусть вам необходимо поменять содержимое стаканов – в первый стакан надо налить молоко из второго стакана, а во второй, наоборот воду из первого. Самый простой способ сделать это будет следующим образом: берем третий стакан, переливаем в него содержимое первого стакана (молоко), затем переливаем содержимое второго стакана (вода) в первый (который в это время уже свободен), а затем из третьего стакана переливаем молоко во второй, освободившийся стакан.

При обмене значениями переменных нужно поступить аналогично. Есть лишь небольшая разница: значения переменных не переносятся, а ко-

пируются. То есть, если мы сделаем следующего вида присваивание: $a=b$ значение остается как переменной a так и в переменной b .

1.5.4 Вычислить корень линейного уравнения

В файле `sqlin.cpp` вычислите значение корня линейного уравнения вида $kx+b=0$, вещественные значения k , b вводите с клавиатуры.

1.6 Вычислить корни квадратного уравнения

В файле `sqsq.cpp` вычислите значения корней квадратного уравнения вида $ax^2+bx+c=0$. Вещественные значения коэффициентов квадратного уравнения вводите с клавиатуры. Корни квадратного уравнения вычисляются по следующим формулам: $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, где $D = b^2 - 4ac$. Будем считать, что при отрицательном D , корней нет. При $D = 0$, корни равные.

2 Функции

2.1 Функция «максимум»

Разработайте программу `max2.cpp`, в которой оформите поиск максимума в виде функции. Данная функция должна принимать на входе два вещественных параметра и возвращать больший из них (тоже вещественный). Например, так:

```
1 #include <iostream.h>
2 float get_max(float a, float b)
3 {
4     if (a>b)
5         return a;
6     return b;
7 }
8 void main()
```

```

9  {
10  float x, y, c;
11  cin >> x;
12  cin >> y;
13  c = get_max(x, y);
14  cout << c << endl;
15 }

```

В строках 2 – 7 задается функция, возвращающая вещественное число (`float`) и принимающая на вход два вещественных числа. В круглых скобках указаны *формальные* параметры `a` и `b`. Формальные параметры являются локальными переменными функции `get_max`. В строках 5 и 6 при помощи оператора `return` возвращаем значение функции `a` или `b` в зависимости от их значений, проверяемых в операторе `if` в строке 4.

В 13 строке осуществляется вызов записанной выше функции. В качестве *фактических* параметров в функцию передаются значения переменных `a` и `b`. Данное значение будет результатом функции `get_max` и, в данном случае (строка 13) это значение будет записано в переменную `c`.

2.2 Варианты индивидуальных заданий

2.2.1 Четность числа

В файле `even.cpp` написать функцию `iseven`, определяющую четность числа, переданного в качестве параметра. Написать программу, демонстрирующую работу данной функции для любого числа, введенного с клавиатуры. Число передается в качестве параметра функции.

2.2.2 Високосный год

В файле `vis.cpp` написать функцию `isvis`, определяющую является ли заданный год високосным. Функция имеет на входе целое число – номер года, а на выходе возвращает 1 или 0, соответственно, високосный и

не високосный. Високосным годом считается тот, чей номер делится на 4. Годы, которые делятся на 100, но не делятся на 400, високосными не являются. Это можно записать в виде следующего правила: Високосные года, это года, номера которых делятся на 4, кроме тех, которые делятся на 100, кроме тех, которые делятся на 400. Это правило легко алгоритмизируется.

То есть, например, 1993 год не високосный – он не делится на 4, и нам нет смысла его рассматривать. 1992 год на 4 делится, но не делится на 100 – данный год является високосным. Год 1900 делится на 4 и делится на 100, но не делится на 400, поэтому он не високосный. Год 2000 делится на 4 (поэтому он должен быть високосным), делится на 100 (поэтому он НЕ должен быть високосным), но делится на 400, поэтому он високосный.

2.2.3 Пробельный символ

В файле `spc1.cpp` напишите функцию `isspace`, определяющую является ли заданный в качестве параметра символ пробельным. 1 – является, 0 – не является. Аналогично предыдущим заданиям необходимо в функции `main` создать пример использования данной функции. Нет необходимости в данном случае вводить символы с клавиатуры достаточно задать символ в качестве параметра, непосредственно в вызове функции

Замечание: будем считать, что пробельным символом помимо собственно пробела являются также символ табуляции (`' \t '`) и перевод строки (`' \n '`).

2.2.4 Корень линейного уравнения

В файле `sqlin2.cpp` оформите в виде функции поиск корня линейного уравнения (см. 1.1.8).

2.3 Корни квадратного уравнения

В файле `sqsq2.cpp` по аналогии с предыдущим необходимо сделать функцию, вычисляющую корни квадратного уравнения (см. 1.1.9). Обратите внимание, что в данной функции может быть от 0 до 2 корней. Значит самое логичное сделать, чтобы данная функция возвращала количество вычисленных корней, а значения корней записывались по указателям, переданным в качестве параметров. Декларация функции должна выглядеть, например, так:

```
1 int get_sqr_root(double a, double b, double c,  
2                 double *x1, double *x2)  
3 {  
4     double D;  
5     D = ...  
6     if (D<0.0) return 0;  
7     if (D==0.0) {  
8         *x1 = *x2 = ...  
9         return 1;  
10    }  
11    *x1 = ...  
12    *x2 = ...  
13    return 2;  
14 }  
15
```

В первых трех параметрах передаются коэффициенты квадратного уравнения. По указателям, переданным в качестве `x1` и `x2`, записываются полученные значения корней уравнения.

2.4 Числа Фибоначчи

Цель данного задания найти число ряда Фибоначчи. Число Фибоначчи математически задается следующим образом: $f_n = f_{n-1} + f_{n-2}$, $f_1 = 1$, $f_2 = 1$. Таким образом, первые значения ряда Фибоначчи выглядят так: 1, 1, 2, 3, 5, 8, 13, 21, ... Вычислите числа Фибоначчи f_5 , f_{20} , f_{100} . Оформите данную программу в файле `fib1.cpp`. Реализуйте вычисление ряда в виде функции `fib`. Данная функция должна принимать в качестве входного параметра номер числа Фибоначчи и возвращать значение данного числа. Суть алгоритма сводится к следующему:

```

Пусть n - номер числа Фибоначчи
Если n равен 1 или 2, то вернем 1
Пусть fn - очередное число Фибоначчи
Пусть f1 и f2 соответственно предыдущее и
предпредыдущее числа Фибоначчи
Переменным f1 и f2 присвоим 1
Для всех i от 3 до n включительно сделаем
    Начало
        fn = f1 + f2
        f1 = f2
        f2 = fn
    конец
вернем fn

```

Поэкспериментируйте с типами переменных, с которыми вы оперируете (попробуйте использовать помимо `int` еще `long` и `double`). Какие результаты Вы получили? Можете объяснить эти результаты?

3 Массивы

3.1 Печать массива

Файл `arr.cpp`. Задать глобальный массив целых переменных, присвоив его элементам начальные значения. Распечатать содержимое массива на экране при помощи цикла `for`.

Для того чтобы создать массив необходимо сделать следующее описание:

```
int m[10];
```

При этом компилятор выделяет 10 ячеек типа `int` (то есть 20 байт, при 2 байтном целом).

Для того, чтобы сразу (при создании) присвоить значения элементам этого массива надо описать его следующим образом: `int m[10]={5,-1,7,14,8,5,2,8,0,4};` При этом компилятор выделенные 10 ячеек сразу заполнит указанными значениями. Если значений не достаточно, то оставшиеся ячейки остаются неинициализированными. Если же значений указано больше, чем размерность массива, то компилятор выдаст сообщение об ошибке, так как он не «знает» куда ему деть остальные значения.

Далее необходимо в цикле перебрать все элементы массива и вывести каждый элемент на экран. Например, так: `for (i=0;i<10;i++) cout << m[i];` Обратите внимание, что для индексации массива используются числа от 0 до 9 включительно (как раз ровно 10 элементов).

3.2 Функция печати массива

Файл `arrprint.cpp`. Модифицировать предыдущую программу: необходимо, чтобы печать массива была оформлена в виде функции “`arr_print`”, в данную функцию необходимо передавать два параметра – указатель на `int` (собственно массив) и целое число (количество элементов

массива). Организовать в главной функции вызов созданной функции, передавая в качестве параметра локальный массив. Это должно выглядеть, например, так:

```
void arr_print(int *m, int n)
{
    ...
}

void main()
{
    int a[10]={5,-1,7,14,8,5,2,8,0,4};
    arr_print(a,10);
}
```

При этом в качестве формального параметра `m` – указателя на `int` в функции `arr_print` будет подставлен адрес начала массива `a` – то есть адрес его нулевого элемента.

3.3 Варианты индивидуальных заданий

3.3.1 Поиск максимума

Файл `max3.cpp`. Добавить к предыдущей программе функцию поиска максимального числа в массиве `maximum`. В качестве параметров функции необходимо также как и в функцию печати массива передавать два параметра: указатель на `int` и размер массива.

3.3.2 Сумма массива

В файле `sum.cpp` добавить к программе `arrprint.cpp` функцию подсчета суммы массива. То есть необходимо просуммировать значения всех элементов массива. Параметры функции необходимо сделать аналогично предыдущему заданию.

3.3.3 Среднее арифметическое значение массива

На основе программы `sum.c` сделать программу подсчета среднего арифметического (`avg.cpp`). Пусть среднее арифметическое вычисляется для массива вещественных чисел, то есть в качестве первого параметра функции будет использоваться `double*`. Возвращать данная функция будет также вещественное число. Однако, второй параметр (размер массива) по-прежнему останется целым числом (размер массива это количество элементов - оно не может быть дробным).

3.3.4 Сортировка массива

В предыдущую программу, скопированную в файл `sort.cpp`, добавить функцию сортировки (упорядочивания) массива. Пусть в эту функцию передается массив вещественных чисел, и, как всегда, размер массива. Функция должна сортировать переданный массив по возрастанию (в начале массива должны оказаться самые маленькие числа в конце самые большие). Подробно об алгоритмах сортировки можно прочитать в специальной литературе [Кнут, т.3]. Мы не будем подробно останавливаться на этих алгоритмах. Реализуем один из самых простых алгоритмов, который известен, как метод пузырька. Его суть заключается в следующем: перебираем все элементы массива, сравнивая два соседних элемента, если элементы не удовлетворяют нас по признаку сортировки (по возрастанию или по убыванию) меняем два этих соседних элемента. Если в процессе перебора была произведена хоть одна операция перестановки, значит можно предположить, что массив еще не весь упорядочен, и весь цикл необходимо повторить. Вот так этот алгоритм можно записать на формальном языке:

Повторять

 Флаг = ложь

 Для i от 0 до $n-1$ делать

```

Если  $m_i > m_{i+1}$  начало
    поменять  $m_i$  и  $m_{i+1}$ 
    Флаг = истина
конец
Пока флаг равен истине

```

Переменная «Флаг» соответствует логическому утверждению «перестановки элементов были». Необходимо помнить, что в данном случае модифицируется (сортируется) оригинальный массив, то есть тот массив, который как раз и был передан в качестве параметра.

3.4 Разупорядочивание массива

В файле `unsort.cpp` напишите функцию разупорядочивания массива. Это задание творческое и не слишком простое, попробуйте самостоятельно придумать какой либо способ реализации данной задачи. Исходные данные: есть упорядоченный массив вещественных чисел, необходимо получить неупорядоченный массив тех же чисел.

3.5 Простые числа

В файле `prost.cpp` напишите функцию нахождения первых 100 простых чисел. Простое число, это число, которое делится без остатка только на 1 и на само себя. Создайте массив беззнаковых целых (`unsigned`) на 100 элементов. Первое простое число – 2, затем каждое очередное число проверяем, не делится ли оно на какое либо из уже найденных простых чисел. Если не делится, то добавляем его в тот же массив. Так да тех пор пока массив не заполнится. Выведете этот массив на экран (у вас уже есть функция, которая печатает массив целых чисел).

3.6 Многомерные массивы

В файле `mtr1.cpp` напишите программу, вычисляющую произведение двух матриц. Напишите функцию ввода матрицы с клавиатуры

(mtr_in), вывода матрицы (mtr_out) на экран, и, собственно, перемножения двух матриц (mtr_mul). Две матрицы A и B размерностью соответственно $m \cdot n$ и $n \cdot s$ перемножаются по следующей формуле:

$$c_{i,j} = \sum_{k=0}^n a_{i,k} \cdot b_{k,j}, i = \overline{1, m}, j = \overline{1, s}$$

При этом размер матрицы C получается равным $m \cdot s$.

Описать двумерный массив можно, например, так: `double a[N][M];`, получить доступ к любому элементу данного массива можно, например, следующим образом: `a[i][j]`. Для простоты будем считать, что при разработке функций мы точно знали количество столбцов, тогда декларация функций будет выглядеть следующим образом:

```
void mtrx_input(double *a[N]);
```

3.7 Точное значение числа Фибоначчи

Файл `fib2.cpp`. Программа, написанная Вами в пункте 1.2.7, страдает одним крупным недостатком: при помощи данной программы невозможно вычислить точное значение числа f_{100} , например. При использовании чисел типа `int` и `long` разрядной сетки процессора не хватает, а при использовании типа `double` (самого точного стандартного типа данных) значение числа получается приближенным. Компьютер штука очень ограниченная и дискретная, она не может автоматически подбирать разрядную сетку числа так, как ей захочется. По крайней мере, пока Вы этого не запрограммировали.

Цель данного задания как раз таки вычислить точное значение числа Фибоначчи номер 100. С одной стороны это число имеет порядок 21 , и равно приближенно $3.5 \cdot 10^{20}$. С другой стороны получить точное значение можно лишь с использованием целочисленной арифметики. Таким образом, нам необходимо длинное целое, в которое вмещается как минимум 21 десятичная цифра. Это можно сделать в виде массива, например `char`.

Проанализировав внимательно код программы `fib1.cpp` можно выделить какие операции нам необходимы. Нам нужно:

Сделать функцию сложения двух десятичных чисел представленных в виде массивов с помещением результата в третий массив. Для реализации этой операции создадим функцию `arr_add`, которая принимает в качестве входных параметров 3 массива (2 слагаемых и их сумма) и их размерность (целое число). Вспомните, как Вы осуществляете сложение чисел в столбик. Реализуйте данную функцию аналогично.

Сделать функцию копирования чисел. Для этого сделаем функцию `arr_cpy` принимающую на входе два массива (входной и выходной) и, опять же размерность массивов.

Функцию присвоения числу начального значения. Это необходимо для начальных присвоений вместо `f1=1`. При этом надо обнулить все разряды, кроме самого младшего, в который необходимо записать значение 1.

Функцию печати числа на экране. В эту функцию будет передаваться массив и его размерность. Выдать на экран надо весь массив в символьном представлении (то есть надо к каждому значению, хранящемуся в массиве прибавить код символа '0', тогда Вы как раз получите код символа, соответствующего данной цифре). Например так : `cout << char(a[i] + '0') ;`

Теперь можно реализовывать функцию вычисления чисел Фибоначчи.

Это задание творческое и не тривиальное. Подумайте внимательно над реализацией данной программы.

4 Строки и указатели

4.1 Длина строки

В файле `str.cpp` напишите функцию вычисления длины строки (`str_lenght`). Саму строку опишите в качестве локальной переменной в функции `main`, например следующим образом: `char s[100]="Hello";` или так: `char *s="world";`. Разница между этими двумя формами объявления в том, что в первом случае создается массив байт в первые 6 элементов которого заносятся значения символов 'H', 'e', 'l', 'l', 'o' и нуль-символ в конец. Передавайте в качестве параметра функции указатель на `char`. Возвращаемым значением данной функции должно быть целое значение – количество символов в строке отличных от нуль-символа.

Эта очень простая функция может быть реализована одним из следующих методов:

```
1 int i; for(i=0;s[i];i++);
2 int i=0; while(s[i++]);
3 int i=0; while(*s++) i++;
```

Замечание: во втором и в третьем варианте предполагается, что переменная `i` имеет начальное значение равное 0.

В первых двух вариантах строка рассматривается как массив символов и обращение к элементам массива осуществляется через индексную переменную `i`. Причем обратите внимание, что во втором варианте значение переменной `i` будет на 1 больше, чем количество ненулевых символов.

Третий вариант более сложный, но при этом более оптимальный. В данном варианте вся обработка строки (подсчет количества символов, в данном случае) происходит через указатель на строку. Перед каждой итерацией цикла проверяется текущее значение ячейки по адресу `s`. Если эта

ячейка не равна 0, то выполняется тело цикла ($i++$). В любом случае после взятия значения ячейки по адресу s производится инкремент данной переменной. Таким образом, в данном цикле указатель s последовательно перемещается по всем символам строки, пока не дойдет до конца строки.

Для этого и для всех последующих заданий данного раздела сделайте в главной функции пример использования.

4.2 Копирование строки

В предыдущую программу добавьте функцию копирования строк (`str_copy`). У этой функции будет уже два параметра типа `char*` – входная и выходная строка. Основной цикл работы данной программы может выглядеть, например, так:

```
1 while (*d++=*s++);
```

При компиляции данного куска текста компилятор, скорее всего, выдаст предупреждение типа: «Possibly incorrect assignment» («возможно некорректное присваивание»). В данном случае можно не обращать внимания на это предупреждение. Компилятор просто сообщает Вам, что, *возможно*, вы ошибочно указали знак присваивания '=' вместо знака сравнения '=='. Но нам здесь нужно как раз таки присваивание.

Разберитесь, как работает данный цикл. Убедитесь в том, копируется завершающий нуль-символ. Обратите внимание на фактические параметры, которые вы будете использовать при вызове данной функции. Убедитесь в том, что у выходного массива выделена память под символы. Например:

```
1 void main() {
2     char *src="Hello world!", dst[20];
3     str_copy(src,dst);
4     cout << src << endl << dst << endl;
5 }
```

Здесь переменная `src` описана как указатель на строку (то есть выделяется память под указатель – 2 либо 4 байта в зависимости от реализации компилятора), которому сразу присваивается значение – адрес строки "Hello world!". Кроме того, выделяется память под саму строчку (13 символов в данном случае, считая нуль символ в конце строки). При передаче в функцию переменной `src` в стек записывается значение данной переменной (адрес начала строки в данном случае).

Под переменную `dst` выделяется 20 байт памяти (2 – 4 байта под указатель не выделяются), а при передаче в функцию в качестве параметра в стек записывается адрес нулевого элемента данного массива.

4.3 Конкатенация строк

Добавьте в тот же файл функцию конкатенации строк (`str_concat`). Данная функция должна добавлять содержимое второй строки к первой. Фактически данная функция есть объединение двух предыдущих – необходимо, во-первых, узнать длину первой строки, а точнее сказать адрес ее нуль символа, а затем скопировать вторую строку, начиная с вычисленного адреса.

4.4 Количество пробелов

Напишите функцию (`str_spaces`), вычисляющую в строке количество символов «пробел» (код 32 десятичного). Основной цикл будет похож на цикл при вычислении длины строки, однако тело цикла изрядно модифицируется: теперь Вам нужно производить инкремент переменной `i` не в каждой итерации, а только в случае, если текущий символ ' ' («пробел»). Кроме того, инкремент указателя `s` нужно производить не при проверке условия цикла, а в теле цикла.

4.5 Пробельный символ

Напишите функцию (`str_isspace`) принимающую на вход одно целое число типа `char` и возвращающую 1, если это число соответствует коду символов «пробел», «табуляция» (символ `'\t'`) или «перевод строки» (символ `'\n'`). Все эти символы считаются «пробельными символами».

4.6 Варианты индивидуальных заданий

4.6.1 Количество пробельных символов

В том же файле написать функцию (`str_spcs`) возвращающую количество пробельных символов в строке. В основном цикле данной функции (сделанной по аналогии с 1.4.4) воспользуйтесь предыдущей функцией (1.4.5), для определения является ли данный символ пробельным.

4.6.2 Удаление пробельных символов

Напишите функцию (`str_delspc`) имеющую два входных параметра (начальная и конечная строки). Данная функция копирует содержимое начальной строки в выходную строку помимо пробельных символов. Для определения является ли символ пробельным, воспользуйтесь функцией из 1.4.5.

4.6.3 Удаление дублирующихся пробельных символов

Напишите функцию `str_deldspc` – более сложный вариант предыдущей функции. В выходную строку копируются не более одного пробела подряд. То есть если встречаются последовательности пробельных символов, то эти последовательности должны быть заменены в выходной строке ровно одним пробелом.

4.6.4 Проверка палиндрома

Написать функцию `palindrom`. Данная функция возвращает значение 1, если строка (параметр функции) является палиндромом, то есть ее можно прочитать одинаково как слева направо, так и справа налево. Например: “город хорог - горох дорог”. Для упрощения задачи будем считать, что все символы набраны в одном регистре (все буквы маленькие).

4.7 Сравнение строк

Написать функцию `str_compare` производящую посимвольное сравнение двух строк. Функция должна возвращать значение +1, если первая строка больше, -1, если первая строка меньше, и 0, если строки идентичны. Строка считается больше, если хоть код символа данной строки больше, чем соответствующий код символа второй строки. Проверку кодов символов необходимо делать слева направо.

4.8 Печать строки в кодах символов

Написать функцию `str_prncod` – печатающую десятичные коды символов переданной в качестве параметров строки.

4.9 Замена символа табуляции на пробелы

Напишите функцию `str_tab2spc`. Все знаки табуляции в исходной строке заменить на последовательность пробелов в выходной. Остальные символы оставить неизменными.

Существуют стандартные правила для табуляций. Один символ табуляции должен передвигать курсор на ближайшую позицию кратную 8 вправо.

Например, строка: “Мама\tмыла” заменятся строкой “Мама!!!!мыла” (‘\t’ – символ табуляции, а вместо пробела, для наглядности записаны символы ‘!’). Другой пример: “Оля\tмыла\траму” преобразуется в строку: “Оля!!!!мыла!!!!раму”.

Для наглядности при выводе строки на экран воспользуйтесь функцией `str_prncod`. При выводе строки на экран в обычном режиме у вас начальная строка должна выглядеть точно так же, как конечная.

4.10 Замена пробелов на символ табуляции

Написать функцию `str_spc2tab` – обратную предыдущей функции: последовательность пробелов, которую можно заменить одним знаком табуляции необходимо сделать это.

4.11 Модуль `str`

В завершении работы составьте заголовочный файл `str.h` в котором запишите декларации всех функций, реализованных в этом файле. Оформите данный заголовочный файл по стандартным правилам:

```
1 #ifndef __STR_H
2 #define __STR_H
3 int str_length(char*s);
4 void str_copy(char*s, char*d);
5 ...
6 #endif
```

То, что Вы сейчас сделали (связка `str.cpp + str.h`), фактически есть модуль, который Вы можете в дальнейшем использовать в любом вашем проекте. Назовем этот модуль `str`. Однако, учитывая, что в программе должна быть только одна функция `main` Вам необходимо исключить из модуля `str` эту функцию (например, просто закомментировать) или вынести в другой модуль (`strmain.c`, например). Вот так может выглядеть главный модуль:

```
1 #include <stdio.h>
2 #include "str.h"
3 void main() {
```

```

4   char *s1="Hello world!",s2[80];
5   printf("\nlen s1: %d\n",str_length(s1));
6   str_copy(s1,s2);
7   printf("s2: %s\n",s2);
8   ...
9  }

```

Здесь, в строке 2 подключается заголовочный файл `str.h`, в котором перечислены декларации функций, которые используются в данном модуле (`str_length` и `str_copy`). Поскольку имя заголовочного файла в директиве `#include` указано в кавычках, препроцессор будет искать данный файл в текущем каталоге. В строке 1 имя указано в угловых скобках, поэтому данный файл будет искааться в стандартном каталоге.

Следующим шагом будет объединить эти два модуля (`str` и `strmain`) в один проект. Как это сделать зависит от той инструментальной системы, которой Вы пользуетесь. Например в интегрированной среде разработки BorlandC++5.02 можно проделать нижеследующее.

Создайте новый консольный проект `strtest` указав в качестве цели (target) `strmain`.

Добавьте к этому проекту помимо добавленного автоматически модуля `strmain.cpp` также модуль `str.cpp`. Для этого на имени исполняемой программы (`strtest.exe`) в дереве проекте щелкните правой кнопкой мыши и из появившегося контекстного меню выберите пункт `Add`. Выберите файл `str.cpp`.

Выберите пункт меню `Project → Make` для сборки всей программы (также это можно сделать из контекстного меню в дереве проектов или клавишей F9). При этом будут скомпилированы все некомпиллированные модули и из полученных объектных файлов при помощи редактора связей будет собрана исполняемая программа (`strtest.exe`).

Одним из универсальных способов, более или менее подходящим для всех инструментальных систем является использование программы `make` (в других средах разработки, например, VisualC эта утилита называется `nmake` и имеет чуть-чуть другой входной язык). Создайте файл `str.mak`:

```
1 OBJS = str.obj strmain.obj
2 CC = bcc32
3 TARGET = strtest
4 $(TARGET) : $(OBJS)
5     $(CC) -e$(TARGET) $(OBJS)
6 str.obj : str.c str.h
7     $(CC) -c str.c
8 strmain.obj : strmain.c str.h
9     $(CC) -c strmain.c
```

В строке 1 создается переменная с именем `OBJS` и значением `"str.obj strmain.obj"`. Эта переменная необходима для удобства, в ней необходимо указывать имена всех объектных файлов, участвующих в проекте. В строке 2 описывается переменная `CC`, в которой указано имя компилятора командной строки, который Вы используете, в данном случае – `bcc32` (Borland C Compiler 32 bit). В третьей строке описывается основная цель проекта – программа `strtest`. В строке 4 указаны `strtest`, в данном случае, `strtest` зависит от `str.obj` и `strmain.obj`. В строке 5 стоит указание для программы `make` каким образом ей нужно получить результат. В данном случае сказано, что необходимо выполнить командную строку:

```
bcc32 -estrtest str.obj strmain.obj
```

Опция компилятора `'-e'` задает имя исполняемого файла. Указание `$ (имя_переменной)` означает подстановку значения данной переменной.

В строках 6 – 7 указываются зависимости и способ получения файла `str.obj`, а в строках 8 – 9 для получения файла `strmain.obj`. Опция компилятора `'-c'` означает «только компиляция» (без редактирования связей).

Теперь наберите в командной строке:

```
make -f str.mak
```

И программа `make` по заданным ей указаниям сформирует исполняемую программу. Если назвать файл просто `makefile` (без расширения, то опция `-f` будет не нужна).

На самом деле здесь Вас может поджидать множество подводных камней, Дать какие-то однозначные рекомендации по их преодолению сложно. Внимательно читайте руководство к вашей среде разработки и читайте, что Вам сообщает программа `make` и Ваш компилятор командной строки.

5 Структуры, объединения, перечисления и синонимы

5.1 Перечисления

В файле `enums.cpp` создайте перечислимые типы данных: `animal` – животные (кошка, собака, лошадь), `week` – дня недели (понедельник, вторник и так далее), `suit` – масть карты (червы, бубны и так далее).

И опишите синоним для этого типа данных. Например:

```
typedef enum t_animal {cat, dog, horse, cow } tanimal;
```

Здесь: описывается перечисление `t_animal` с 4-мя возможными значениями (`cat`, `dog`, `horse`, `cow`). А также описывается синоним: `'tanimal'` для `'enum t_animal'`.

Попробуйте сделать операции присваивания переменных описанных типов. Вывод на печать, ввод с клавиатуры.

5.2 Булев тип данных

В файле `boolean.h` задайте перечисление для типа `bool` два значения: `false` и `true`, соответственно первый из них должен иметь значение 0, а второй 1. В стандарте языка C++ 1998 года тип `bool` уже есть, поэтому более или менее современные системы программирования на языке C++ (например, Borland C++ 5.02) при переопределении данного типа компилятор выдаст сообщение об ошибке типа: «Invalid type redeclaration». В таких случаях будем считать, что такой тип у Вас уже есть и вводить его не надо.

5.3 Варианты индивидуальных заданий

5.3.1 Структуры: `point`

В файле `point.cpp` запишите структуру `t_point` – точка, в которой содержится три поля: `x`, `y` и `z` вещественных типов. И синоним `point` для данной структуры.

Напишите нижеследующие функции. Функцию скалярного умножения `scrlmult`, у которой есть два входных параметра – указателя на структуру типа `point` и один выходной – вещественное число – скалярное произведение данных векторов. Функцию векторного умножения `vctrmult` (три входных параметра – два множимых вектора и один результирующий – все указатели на `point`). Функцию ввода вектора с клавиатуры, и функцию печати вектора на экране.

Например, так:

```

1 typedef struct t_point {
2     double X;
3     double Y;
4     double Z;
5 } point;
6 double scrlmult(point *a, point *b) {
7     return a->X*b->X + a->Y*b->Y + a->Z*b->Z;
8 }
9 void vctrmult(point *a, point *b, point *c) {
10    point r;
11    r.X = a->Y*b->Z - b->Y*a->Z;
12 ...
13 }
14 void pointprn(point *p) {
15    point a;
16    cout << a.X << ", " << a.Y << ", " << a.Z;
17    *p = a;
18 }
```

Напишите главную функцию, используя данные функции.

5.3.2 Структуры: person

В файле `person.cpp` разработайте структуру `person`, содержащую информацию о человеке (имя, возраст, пол и т.д.). Например, следующим образом:

```
1 typedef enum t_sex {female,male} sex;
2 typedef struct t_person {
3     char Name[20];
4     int Age;
5     sex Sex;
6 } person;
```

Разработайте функции `person_input` – ввод персоны с клавиатуры и `person_print` – печать персоны на экране. В качестве параметра этих функций необходимо использовать указатель на `person`. Для доступа к полям структуры используйте оператор «точка»:

```
7 person John;
8 John.age=19;
9 str_copy(John.Name, "John");
```

Используйте созданный ранее модуль `str` для копирования строк (если в этом есть необходимость). Изучите, как в вашей системе можно создавать многомодульные программы.

Для доступа к полям структуры через указатель на структуру используйте оператор «стрелка»:

```
10 void person_print (person* p) {
11     cout << p->Name << " " << p->Age << " " << p-
        >Sex?'M':'W';
12 }
```


Создайте заголовочный файл `person.h`, в который вынесите объявление структуры `person` и занесите декларацию всех функций.

5.4 Объединения

В файле `union.cpp` запишите структуру `animal` – «животное» в данной структуре должно быть поле тип животного (`animal type`; - см. 1.5.1), название животного (`char name[20]`) и поле специфичных данных. Специфичные данные должны быть описаны, как объединение, включающее в себя структуры данных: `thorse`, `tcow`, `tcat`, `tdog`. В каждой из этих структур данных содержаться поля, специфичные для данного животного – для коровы, например, ее средний годовой удой, для собаки ее порода, масть и так далее.

6 Динамические структуры данных

6.1 Обработка массива

В данной программе (`darr.cpp`) необходимо выдать все характеристики массива вещественных чисел, такие как, максимальное число, минимальное число, сумма всех чисел, среднее арифметическое. Кроме того, надо отсортировать массив по убыванию и по возрастанию. Необходимо распечатать массив до сортировки и после каждой из сортировок. Для выполнения всех этих действий можно (и нужно) воспользоваться функциями, разработанными в предыдущем разделе.

Отличие от предыдущего раздела заключается в том, что массив, для которого необходимо вызывать данные функции необходимо создавать динамически, например: так:

```
1 void main() {
2     double *m;
3     int n;
4     cin >> n;
5     m = (double*)calloc(n, sizeof(double));
6     if (m==NULL) {
7         cerr << "Не хватает памяти. Выход." << endl;
8         exit(-1); }
9     ...
10    free(m);
11 }
```

Когда Вы описываете массив, например, так: `double a[10];` выделяется память под 10 ячеек типа `double`. Вы можете обращаться к элементам этого массива через имя `a`, при этом обращение компилятор будет брать адрес нулевого элемента этого массива. При передаче в функцию массив, фактически передается адрес его нулевого элемента.

При работе с указателями ситуация немножко другая. В строке 2 описывается указатель на `double`, при этом в памяти выделяется место только под указатель. Указатель на любой тип данных занимает в памяти 2 либо 4 байта (в зависимости от реализации компилятора). В строке 5 вызывается функция `malloc`, которая выделяет место в специальной области памяти (в так называемой «куче»), на заданное количество байт ($n \cdot \text{размер_типа_double}$, в данном случае). В указатель записывается адрес начала этого массива. В дальнейшем при обращении к элементам этого массива (например, так: `m[i]`) компилятор генерирует следующие действия: вычисляет значения выражения в квадратных скобках, умножает полученное значение на размер базового типа (для `double` 8 байт), прибавляет к полученному значению

6.2 Варианты индивидуальных заданий

6.2.1 Организация динамических массивов

В файле `table.cpp` создайте глобальные переменные:

```
1 person *tblptr;
2 int tbln;
3 int tblcount;
```

`tblptr` – указатель для хранения динамического массива. `tbln` – размер выделенной под таблицу памяти, то есть максимально возможное количество записей в таблице. `tblcount` – количество записей в таблице уже заполненных, оно же является номером первой свободной ячейки.

Создайте функции:

`table_create` – создание таблицы. В качестве входных параметров принимает количество элементов (максимальный размер) таблицы - `n`. Функция инициализирует переменные `tbln` и `tblcount` начальными значениями (`n` и 0 соответственно). В `tblptr` записывается указатель на

блок выделенной памяти. Блок памяти можно выделить, например, следующим образом:

```
1 tblptr = (person*)calloc(n, sizeof(person));
```

После чего необходимо проверить смогла ли система выделить положенный блок памяти и, если не смогла, то выйти из программы.

- `table_destroy` – уничтожение таблицы. Основная забота данной функции освободить память, выделенную под таблицу при ее создании. А также очистить (обнулить) все переменные, связанные с таблицей.
- `table_add` – добавить одну запись в конец таблицы. В качестве параметра данной функции передается указатель на `person`. Запись записывается в первую же свободную ячейку, (если есть еще место в таблице). После чего производится инкремент переменной `tblcount`. Не забудьте проверить наличие свободного места в таблице (сравните значение `tblcount` и `tbln`).
- `table_print` – печать таблицы на экране.
- `table_insert` – вставка записи на заданную позицию. В качестве параметра передается номер позиции (целое число), в которой будет находиться новая запись и сама запись. Данная функция должна сдвинуть все записи, следующие за указанной записью. То есть таблица должна «раздвинуться».
- `table_delete` – удаление записи из таблицы. Передается номер удаляемой записи (целое число). Все записи, находящиеся в таблице ниже удаляемой записи должны быть сдвинуты.

Создайте заголовочный файл (`table.h`), в котором запишите декларации созданных функций. Поскольку в данном файле Вы используете для декларации функций структуру `person`, описанную в файле `person.h` включите (`#include`) файл `person.h` в файл `table.h`.

6.2.2 Организация списков

В файле `list.h` опишите структуру для двусвязного списка:

```

1 typedef struct t_list_item {
2     person *Data;
3     struct t_list_item *Prev, *Next;
4 } list_item;
5
6 В файле list.cpp создайте экземпляры переменных:
7 list_item *Head; /* голова списка */
8 list_item *Tail; /* хвост списка */
9 list_item *Curr; /* текущий элемент списка */

```

Реализуйте следующие функции:

Инициализация, деинициализация:

- `list_init` – без входных и выходных параметров. Начальная инициализация списка – список считается пустым, поэтому всем переменным списка (`Head`, `Tail` и `Curr`) нужно присвоить значение `NULL`.
- `list_destroy` – без входных и выходных параметров. Уничтожение списка. Необходимо пройти по всем элементам списка и освободить память, выделенную под элементы списка. Это можно следующим способом:

```

1 void list_destroy()
2     list_item *p, *t;
3     for (p=Head; p;) {
4         t=p->Next;
5         free(p);
6         p=t;
7     }
8 }

```

Добавление элементов:

- `list_add_tail` – добавление элементов в хвост списка – один входной параметр – добавляемые данные (Персона). Алгоритмически это нужно сделать следующим способом:

Выделить память под новый элемент списка, проверить смогла ли система выделить нужный кусок памяти, и предпринять соответствующие действия, если не смогла (завершить работу программы, выдав предупреждение на экран).

Заполнить поля выделенной структуры данными (поле `Data`), а поля «предыдущий» и «следующий» обнуляем (`NULL`)

Если добавляемый элемент – первый элемент списка, то тогда данный элемент является головой, хвостом и текущим элементом списка.

Иначе:

Направляем указатель «следующий» последнего элемента на новый элемент.

Направляем указатель «предыдущий» на текущий хвост.

Направляем указатель «хвост» на новый элемент.

```

1 void list_add_tail(person *d) {
2     list_item* t =
        (list_item*)malloc(sizeof(list_item));
3     t->Prev = t->Next = NULL;
4     t->data = d;
5     if (!Head)
6         Head = Tail = Curr = t;
7     else {
8         t->Prev = Tail;
9         Tail->Next = t;
10        Tail = t;

```

```

11     }
12 }

```

- `list_add_head` – аналогично предыдущему, только с головы (все зеркально).
- `list_add_after` – добавление нового элемента после текущего. Лучше сперва проверить не является ли текущий хвостом, и в этом случае вызвать уже реализованную ранее функцию. Не забудьте проверить, не пуст ли список, в этом случае работа немножко особенная.
- `list_add_before` – аналогично предыдущему за исключением того, что добавляется **перед** текущим.
- Навигация:
 - `list_go_head` – простая функция – переход в начало списка. Реализуется просто: `Curr = Head;`
 - `list_go_tail` – аналогично предыдущему, но в конец списка.
 - `list_go_n` – переход на заданный номер, в качестве параметра – целое число – номер записи, на которую нужно перейти. Необходимо организовать цикл прохода по списку.
 - `list_go_data` – переход на заданные данные, передается `person` в качестве параметра (образца) записи, которую надо сделать текущей.

Удаление:

- `list_delete` – удаление текущего элемента.

Прочие:

- `list_print` – печать списка на экране. Входных/выходных параметров нет. Цикл может выглядеть следующим образом:

```

1  list_item *p;
2  for (p=Head; p; p=p->Next)

```

```
3    person_print(p->Data);
```

- `list_for_each` – для каждого. Выполнить переданную в качестве параметра функцию для каждого элемента списка. Один входной параметр: `bool (*f)(person*)` – указатель на функцию, которая в качестве параметра принимает указатель на данные (Персона, в нашем случае), а на выходе возвращает значение типа `bool`. Функция `list_for_each` сама возвращает также значение типа `bool`. Цикл должен выполняться пока функция `f` не вернет значение `false`. В этом случае функция возвращает ложь, иначе (если функция выполняется до конца), возвращается истина.
- `list_get_curr` – возвращает персону из текущей ячейки (`Curr->Data`).

Рекомендуется выполнять работу в следующей последовательности:

Реализовать методы `list_init`, `list_destroy`, `list_add_head` и `list_print`. Отладить данные функции.

Затем сделать первые 3 функции из списка функций навигации. Слегка модифицировать функцию печати – внести в нее возможность отображения текущего элемента (например, выводить перед текущим элементом символ '>') дабы можно было проконтролировать работу функций навигации.

Реализовать и отладить функцию удаления.

Последовательно сделать и отладить оставшиеся 3 функции добавления.

Реализовать итератор `for_each`. Сделать печать на основе этого итератора.

Остальные функции.

7 Файловый ввод-вывод

7.1 Конвертация файла

Цель данной задачи написать простую программу (`convert.cpp`). Данная программа должна считать информацию из заданного файла (например, `"src.txt"`) в следующем формате: `<a (целое число)> <k (вещественное число)> <b (целое число)>`. Например:

```
3 5.85 6
5 6.975 8
3 2.3 2
```

В выходной файл (например, `"dst.txt"`) необходимо записать результаты преобразования этих строк по следующей формуле: $a*k+b$. Для вышеприведенного входного файла выходной будет выглядеть следующим образом:

```
23.55
42.875
8.9
```

Это можно реализовать следующим образом:

```
1 void main() {
2     int a,b;
3     int r;
4     float k,res;
5     ifstream fin("src.txt");
6     ofstream fout("dst.txt");
7     while(fin){
8         fin >> a;
9         fin >> k;
10        fin >> b;
```

```

11     if (!fin) break;
12     res = a*k+b;
13     fout << res << endl;
14 }
15 }

```

В строке 5 объявляется объект типа `ifstream` – потоковый ввод из файла, а в строке 6 соответственно объект типа `ofstream` – потоковый вывод в файл. Эти классы объявлены в заголовочном файле `ifstream.h`. к этим классам применимы уже знакомые нам операции `>>` и `<<` также, как и к объектам `cin` и `cout`. Через эти объекты осуществляется доступ к информации о файле функциями, предназначенными для работы с файлами. Эти файлы сразу же открываются.

В строке 7 проверяем, не кончился ли файл. Далее в строках 8-10 считываем из входного файла 3 поля. Если после этого файл оказался пустым (строка 11 проверяет все ли в порядке с входным файлом), то считаем, что считать 3 поля не удалось и потому, что файл уже кончился и выходим из тела цикла. В противном случае вычисляем по формуле результат (строка 12) и записываем это значение в вещественном виде в файл (строка 13).

При выходе из функции объекты `fin` и `fout` удаляются автоматически.

7.2 Варианты индивидуальных заданий

7.2.1 Запись таблицы на диск/чтение таблицы с диска

Добавьте в модуль `person` следующие функции:

- `person_write` – запись персоны на диск. Два параметра – указатель на `person` – собственно персона, которую нужно вывести в файл и `ostream&` (ссылка на `ostream`) – открытый файл, в который надо вывести заданную персону.

- `person_read` – чтение персоны с диска. Аналогично предыдущему – два параметра – куда надо считать персону (`person*`) и откуда надо ее считать (`istream&`).

В модуле `table` добавьте функцию `table_write` с одним параметром типа `char*` - имя файла. В этой функции необходимо открыть заданный файл на запись, записать в него целое число – количество записей в таблице, и, затем, записать при помощи функции `person_write` на диск всю таблицу. Не забудьте закрыть файл по окончании работы с ним.

Добавьте функцию `table_read` – чтение файла с диска. Один параметр – имя файла. При помощи функции `person_read` считать содержимое заданного файла. Для этого надо открыть файл на чтение, считать количество записей, а потом прочитать в цикле известное количество записей. Перед тем как начинать считывать записи из файла желательно убедиться, что в таблице достаточно места для того количества записей, которое есть в файле.

7.2.2 Запись списка на диск/чтение списка с диска

Реализуйте функции `list_read` и `list_write` в модуле `list` аналогично предыдущему, но для списка. В данном случае нет необходимости записывать количество записей при записи в файл. И аналогично при считывании нам нет необходимости проверять вместимость. Однако, здесь вам придется немного модифицировать функцию `person_read` – необходимо, что данная функция возвращала булево значение – удалось ли ей считать заданное количество записей. Тогда можно организовать цикл аналогичный тому, что приведен в 1.7.1.