Озёрский технологический институт –
филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

Александр Зубаиров

ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие

Сборник задач

Озёрск, 2020

УДК 004.432.2
ББК 32.973.26-018.1
З91

Зубаиров А. Ф.

З91 Программирование : учеб.-метод. пособие : сборник задач / А. Ф. Зубаиров – Озёрск : ОТИ НИЯУ МИФИ, 2020. – 124 с.

Учебно-методическое пособие предназначено для изучения дисциплины «Программирование» студентами, обучающимися по направлению «Информатика и вычислительная техника» и специальности «Применение и эксплуатация автоматизированных систем специального назначения».

Пособие состоит из 16 разделов, соответствующих отдельным важным темам, изучаемым в рамках дисциплины «Программирование».

Каждый раздел содержит необходимые теоретические сведения с примерами на языке программирования C/C++ и примеры решения типовых задач. По определенным темам приведены наборы вариантов заданий, которые могут использоваться в качестве упражнений для закрепления теоретического материала, в качестве дополнительных заданий на практических занятиях, качестве заданий для самостоятельных и контрольных работ.

Рецензенты:

Оглавление

Введение

Учебное пособие «Программирование» в первую очередь ориентировано на студентов, начинающих изучение курса программирования «с нуля». В качестве первого языка для изучения программирования предлагается язык С с элементами языка С++ за исключением его средств объектно-ориентированного программирования (далее – языки С/С++).

Для начала изучения программирования языки С/С++ выбраны не случайно.

Во-первых, языки С/С++ являются одними из наиболее популярных в настоящее время. Так, в соответствии с рейтингом TIOBE programming community index на февраль 2020 года язык С является самым популярным, а С++ стоит на пятом месте после языкоа Python, Java, Java Script, и C#. Рейтинг PYPL PopularitY of Programming Language показывает, что языки С/С++ входят в первую десятку самых популярных языков программирования судя по запросам в поисковой системе Google.

Во-вторых, языки С/С++, являясь языками программирования высокого уровня, в то же время позволяют выполнять операции, характерные для низкоуровневых языков программирования. А это значит, что для написания программ на языках С/С++ необходимо не только знать основы программирования и алгоритмизации, но и иметь представление о процессах обработки данных в компьютере, что является необходимым условием для успешной деятельности в сфере создания программного обеспечения.

Выполняя задания из данного пособия, студенты ОТИ НИЯУ МИФИ обязаны соблюдать требования к оформлению программных модулей, установленные в методическом пособии «ТПМ. Требования к программным модулям» (Вл. Пономарев. ТПМ. Требования к программным модулям. Методические указания. Ред. 1. Озерск: ОТИ МИФИ, 2006. – 40 с.).

1. Блок-схемы линейных алгоритмов

Блок-схема – графический способ записи алгоритмов, в котором отдельные шаги изображаются в виде блоков различной формы – символов блок-схемы, соединенных между собой линиями, указывающими направление выполнения алгоритма.

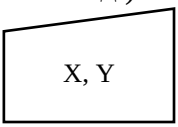
В Российской Федерации правила выполнения блок-схем определяет ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

Для описания алгоритмов в рамках выполнения работ из данного сборника необходимо использовать следующие символы:

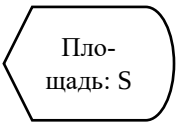
- символ «терминатор» отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы):

- символ «соединитель» отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение:

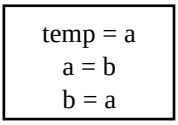
- символ «ручной ввод» отображает данные, вводимые вручную во время обработки с устройств любого типа (клавиатура, переключатели, кнопки и т.д.):



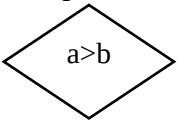
- символ «дисплей» отображает данные, представленные в человекочитаемой форме на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации):



- символ «процесс» отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации):

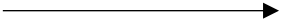


- символ «решение» отображает алгоритмическую структуру «ветвление», имеющую один вход и ряд альтернативных выходов, один, и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа.

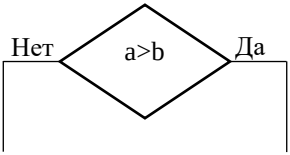


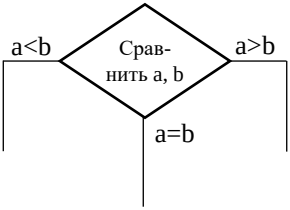
За исключением символов «терминатор», «соединитель» и «решение», каждый символ имеет один вход и один выход. «Терминатор» и «разделитель» имеет либо вход, либо выход. «Решение» имеет один вход и минимум два выхода.

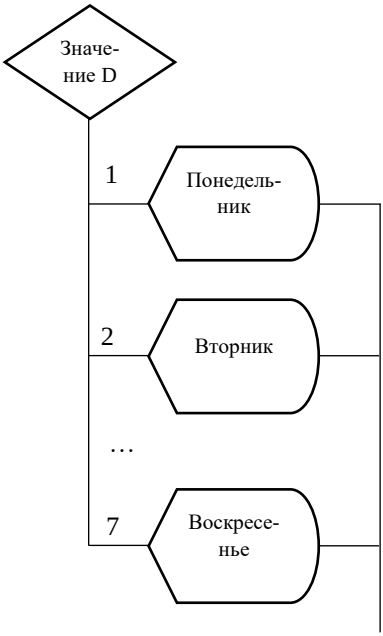
Линии, соединяющие символы, при необходимости или для повышения удобочитаемости могут иметь стрелки-указатели:



Для символов «решение» линии могут быть подписаны:







Варианты заданий

Описать в виде блок-схемы алгоритм решения задачи.

1. Три сопротивления `R1`, `R2` и `R3` соединены параллельно. Найти сопротивление соединения. Примечание: использовать формулу $R = R_1R_2R_3/(R_1 + R_2+R_3)$.

2. Определить время падения камня на поверхность земли с высоты `h`. Примечание: использовать формулу $t = \sqrt{2h/g}$, где $g = 9.81$.

3. Известна длина окружности `l`. Найти площадь круга, ограниченного этой окружностью. Примечание: использовать формулу $S = l^2/(2\pi)$.

4. Вычислить длину высоты треугольника, опущенной на сторону `a`, по известным значениям длин его сторон `a`, `b`, `c`. Примечание: использовать формулу $h = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$, где $p = (a + b + c)/2$.

5. Вычислить объем цилиндра с радиусом основания `r` и высотой `h`. Примечание: использовать формулу $V = \frac{1}{3}\pi r^2 h$.

6. Определить расстояние, пройденное физическим телом за время `t`, если тело движется с постоянным ускорением `a` и имеет в начальный момент времени скорость `v0`. Примечание: использовать формулу $S = v_0 t + \frac{at^2}{2}$.

7. Вычислить площадь треугольника по формуле Герона, если заданы его стороны `a`, `b` и `c`. Примечание: использовать формулу $S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = \frac{a+b+c}{2}$.

8. По данным сторонам прямоугольника `a` и `b` вычислить длину его диагонали. Примечание: использовать формулу $d = \sqrt{b^2 + a^2}$.

9. Даны два числа. Найти среднее арифметическое их квадратов.

10. Дана длина ребра куба. Найти площадь полной поверхности этого куба.

11. Найти площадь круга заданного радиуса `R`. Примечание: использовать формулу $S = \pi r^2$.

12. Найти площадь поверхности шарового слоя, если известны высота слоя `h` и радиус шара `r`. Примечание: использовать формулу $S = 2\pi rh$.

13. Груз заданной массы `m`, прикрепленный к пружине жесткости `k`. Определить период гармонических колебаний груза на пружине. Примечание: использовать формулу $T = 2\pi\sqrt{m/k}$.

14. Найти период колебания математического маятника заданной длины l. Примечание: использовать формулу $T = 2\pi\sqrt{\frac{l}{g}}$, где $g = 9.81$.

15. Скорость лодки в стоячей воде `v` км/ч, скорость течения реки `u` км/ч (`u < v`). Время движения лодки по озеру `t1` ч, а по реке (против течения) – `t2` ч. Определить путь `s`, пройденный лодкой.

16. Дана сторона равностороннего треугольника `a`. Найти площадь этого треугольника. Примечание: использовать формулу $S = \frac{\sqrt{3}}{4}a^2$.

17. Дана сторона основания правильной треугольной пирамиды `a` и высота пирамиды `h`. Найти объём этой пирамиды. Примечание: использовать формулу $V = \frac{ha^2}{4\sqrt{3}}$.

18. Дана длина ребра правильного тетраэдра `a`. Найти его объём. Примечание: использовать формулу $V = \frac{a^3\sqrt{2}}{12}$.

19. Тело брошено с высоты `h` с начальной скоростью `v0`. Найти дальность падения тела. Примечание: использовать формулу $l = v_0\sqrt{\frac{2h}{g}}$, где $g = 9.81$.

20. Найти площадь кольца, внутренний радиус которого равен `r1`, а внешний радиус равен `r2` (`r1 < r2`).

Пример выполнения задания

Вычислить площадь поверхности шарового сектора шара с радиусом основания `r` шара радиуса `R`.

Площадь поверхности шарового сектора вычисляется по формуле:

$$S = 2\pi R\left(R + \frac{r}{2} - \sqrt{R^2 + r^2}\right),$$

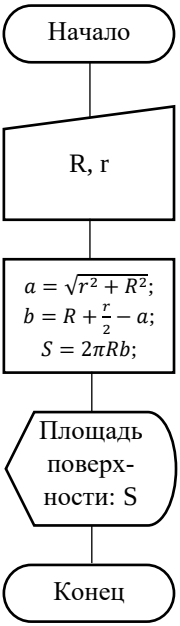
где `R` – радиус шара, `r` – радиус основания конуса.

Блок-схема алгоритма начинается и заканчивается символом «терминатор» с текстом «Начало» и «Конец» соответственно.

Перед началом выполнения алгоритма необходимо получить исходные данные из внешней среды (например, с клавиатуры пользователя). В символе «ручной ввод» указано, что в программу вводятся значения `R` и `r`.

Далее в символе «процесс» описывается процесс вычисления требуемой величины. Величина S вычисляется в три действия только затем, чтобы содержимое символа «процесс» не было громоздким. Не было бы ошибкой указать лишь одно действие $S = 2\pi R(R + \frac{r}{2} - \sqrt{R^2 + r^2})$.

Последним шагом работы алгоритма является вывод результата на экран. Вывод обозначается символом «дисплей», в котором указано «Площадь поверхности: S». Предполагается, что на экране должна появиться надпись «Площадь поверхности:» и вычисленное значение `S`.



2. Блок-схемы разветвляющихся алгоритмов

Варианты заданий

Описать в виде блок-схемы алгоритм решения задачи.

Указание: для получения отдельных цифр числа следует использовать операции деления по модулю (получения остатка от деления) `mod` и операцию целочисленного деления `div`. Например, результатом операции `153 mod 10` будет `3` – младшая цифра числа. Затем, если выполнить целочисленное деление `153 div 10`, результатом будет число `15`. Далее последовательное применение операций `mod` и `div` будет давать следующие цифры числа.

1. Определить и вывести на экран, является ли заданное число однозначным, двузначным или трехзначным.

2. Определить и вывести на экран, состоит ли двузначное число из одинаковых цифр.

3. Определить и вывести на экран, существует ли треугольник с заданными длинами сторон `x`, `y`, `z`.

4. Выбрать из заданных чисел `x`, `y`, `z` те, которые принадлежат интервалу (1, 3), и вывести их на экран.

5. Даны вещественные числа `a`, `b`, `c`, `d`. Найти `min{max{a, b}, max{c, d}}` и вывести его на экран.

6. Даны вещественные числа `x`, `y`. Если только одно из них отрицательно, заменить его значение на его абсолютную величину. Если отрицательны оба заменить их значения на `0`. В остальных случаях `x`, `y` оставить без изменения. Вывести новые значения на экран.

7. Определить и вывести на экран номер квадранта, в котором расположена точка `M(x,y)`, где `x` и `y` заданные вещественные числа. Известно, что точка не лежит ни на одной из осей.

8. Заданы числа `a`, `b`, `c`. Известно, что одно отлично от других, равных между собой. Вывести это число на экран.

9. Определить, какая из двух точек – M1(x1, y1) или M2(x2, y2) – расположена ближе к началу координат. Вывести на экран координаты этой точки.

10. Заданы сторона квадрата a, радиус круга r. Определить, какая из двух фигур (круг или квадрат) имеет большую площадь. Вывести на экран название и значение площади большей фигуры.

11. Определить и вывести на экран, попадает ли точка с заданными координатами x, y в круг радиусом r с центром в точке x0, y0.

12. Дано целое число в диапазоне 10–99. Вывести строку – словесное описание данного числа, например, 25 – «двадцать пять», 81 – «восемьдесят один».

13. Даны переменные x, y. Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное. Результат вывести на экран.

14. Дано трёхзначное число. Определить и вывести на экран, кратно ли четырем произведение его цифр.

15. Определить и вывести на экран, является ли треугольник, заданный координатами вершин (x1, y1), (x2, y2), (x3, y3) равносторонним.

16. Даны неотрицательное целое число a, положительное целое число b, числа r и s. Определить и вывести на экран, верно ли, что при делении a на b, получается остаток, равный r либо s.

17. Даны три вещественных числа a, b, c. Определить и вывести на экран, имеется ли среди них хотя бы одна пара чисел, имеющих противоположные значения.

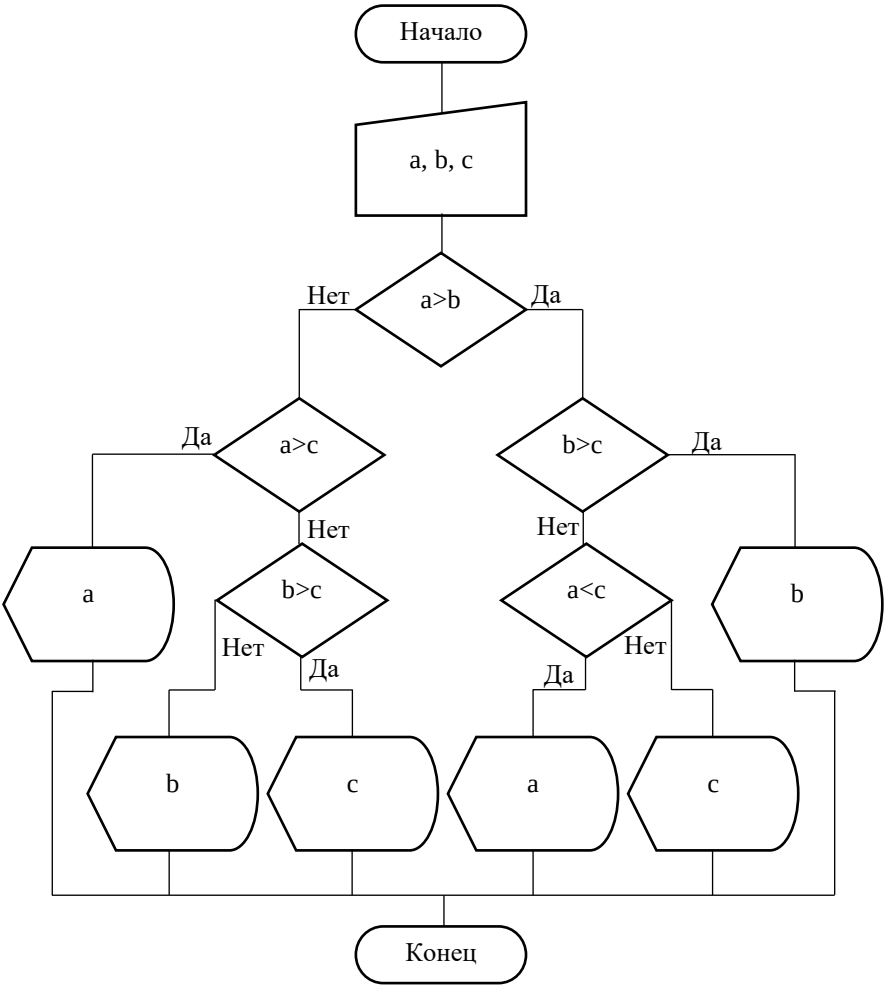
18. Даны целые числа n, r, a, b. Определить, поместятся ли n шаров радиуса r в коробку с длинами сторон a и b.

19. Определить и вывести на экран, является ли одно из заданных чисел `x`, `y` общим делителем заданных чисел `a` и `b`.

20. Определить, является указанный год високосным. Год является високосным если он кратен 4, но при этом не кратен 100, либо кратен 400.

Пример выполнения задания

Выбрать среднее по значению из трех различных чисел.



Блок-схема алгоритма начинается и заканчивается символом «терминатор» с текстом «Начало» и «Конец» соответственно.

Перед началом выполнения алгоритма необходимо получить исходные данные из внешней среды (например, с клавиатуры пользователя). В символе «ручной ввод» указано, что в программу вводятся некие значения `a`, `b` и `c`.

Далее необходимо выяснить, какое из чисел является средним по значению. Для этого предлагается выполнить ряд проверок, в ходе которых определяются отношения между тремя значениями.

Символ «решение» позволяет указать, по какому пути должен выполняться алгоритм в случае истинности или ложности некоторого проверяемого условия.

Сначала проверяется, больше ли `a`, чем `b`. Если больше, необходимо проверить отношение между `b` и `c`. Если `b` больше `c`, значит, что `b` является средним (`a` больше, чем `b`, и `b` больше, чем `c`) и далее его значение выводится на экран. В случае, если `c` больше чем `b`, необходимо сравнить `c` с `a`. Меньшее из них и будет средним числом.

В случае, если в результате первой проверки условия `a > b` алгоритм выполняется по ветви «Нет», логика рассуждений остается прежней.

Следует обратить внимание, что в итоге все линии из символов «дисплей» сливаются в одну, которая входит в конечный символ «терминатор», так как любой алгоритм имеет только один вход и только один выход.

3. Блок-схемы циклических алгоритмов

Варианты заданий

Описать в виде блок-схемы алгоритм решения задачи.

1. Вывести на экран все двузначные числа, сумма цифр которых не меняется при умножении числа на 2.

2. Вывести на экран все трехзначные числа, сумма цифр которых равна заданному целому числу.

3. Вывести на экран все трехзначные числа, средняя цифра которых равна сумме первой и третьей цифр.

4. Определить и вывести на экран делители заданного числа.

5. Вывести на экран все двузначные числа, сумма квадратов цифр которых делится на 17.

6. Найти и вывести на экран факториал заданного числа. Примечание: $n! = \prod_{i=1}^{n} i$ ($n!$ – обозначение для факториала числа n).

7. Найти и вывести на экран такое двузначное число, куб суммы цифр которого равен квадрату самого числа.

8. Найти и вывести на экран двузначное число, равное утроенному произведению его цифр.

9. Даны два целых числа `A` и `B` (`A < B`). Вывести все целые числа, расположенные между данными числами (не включая сами эти числа), в порядке их убывания, а также количество этих чисел.

10. Дано целое число `N > 1`. Вывести наибольшее целое `K`, при котором выполняется неравенство $3^K < N$, и само значение 3^K.

11. Дано вещественное число `A` и целое число `N > 0`. Вычислить при помощи цикла и вывести на экран значение A^N.

12. Дано вещественное число `A` и целое число `N > 0`. Вывести на экран значение суммы `A + 2A + 3A + ... + NA`.

13. Дано число `A` и целое число `B > 0`. Вывести на экран значение суммы `A/B + 2A/(B-1) + 3A/(B-2) + ... + BA/1`.

14. Определить и вывести на экран, является ли заданное число совершенным, то есть равным сумме всех своих положительных делителей, кроме самого этого числа (например, число 6 совершенно: 6=1+2+3).

15. Составить алгоритм вывода всех двузначных чисел, сумма квадратов цифр которых делится на 13.

16. Дано натуральное число `n`. Вычислить и вывести на экран $\frac{1}{\sin 1} + \frac{1}{\sin 1+\sin 2} + \cdots + \frac{1}{\sin 1+\cdots+\sin n}$.

17. Дано натуральное число `n`. Вычислить и вывести на экран $\frac{1}{x} + \frac{1}{x(x+1)} + \cdots + \frac{1}{x(x+1)\ldots(x+n)}$.

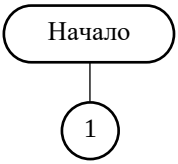
18. Дано натуральное число `n`. Вычислить и вывести на экран $\sum_{i=1}^{n} \frac{1+i}{i}$.

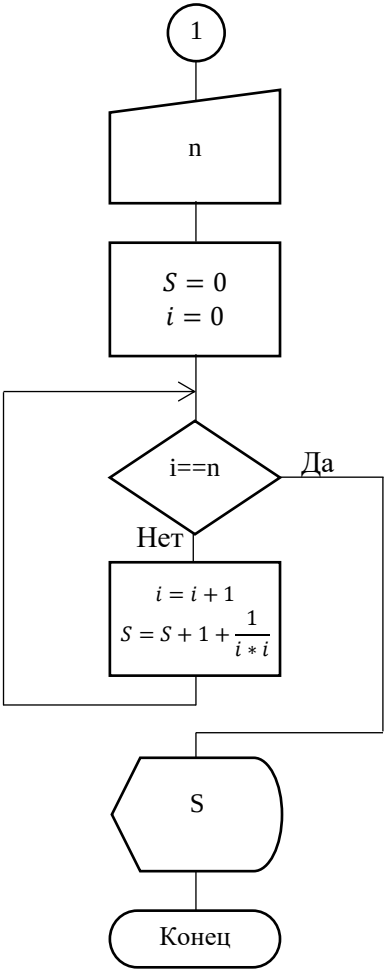
19. Дано десятизначное число. Определить и вывести на экран, упорядочены ли цифры в записи этого числа по убыванию.

20. Дано десятизначное число. Определить и вывести на экран, верно ли, что число состоит только из 0 и 1.

Пример выполнения задания

Дано натуральное число `n`. Вычислить $\left(1+\frac{1}{1^2}\right) + \left(1+\frac{1}{2^2}\right) + \left(1+\frac{1}{3^2}\right) + \cdots + \left(1+\frac{1}{n^2}\right)$.





Блок-схема алгоритма начинается и заканчивается символом «терминатор» с текстом «Начало» и «Конец» соответственно.

Так как вся блок-схема не помещается на одной странице, использован символ «разделитель», который прерывает блок-схему на одной странице и продолжает её на другой. Разделитель помечен значением «1».

Перед началом выполнения алгоритма необходимо получить исходные данные из внешней среды (например, с клавиатуры пользователя). В символе «ручной ввод» указано, что в программу вводится значение `n`.

Далее в символе «процесс» описывается инициализация переменных: в переменную `S` записывается `0` (в переменной `S` будет накапливаться сумма слагаемых), в переменную `i` записывается `0` (переменная `i` будет одновременно и счётчиком цикла, и числом, на квадрат которого в арифметическом выражении делится единица).

При помощи символа «решение» организован циклический алгоритм: очередное слагаемое вычисляется и прибавляется к общей сумме до тех пор, пока значение `i` не станет равно значению `n`, то есть пока не будут просуммированы все необходимые слагаемые. Как только требование равенства `i` и `n` будет выполнено, управление перейдёт к символу «Дисплей», который означает вывод результата `S` экран.

Внутри цикла значение переменной увеличивается на `1` и к общей сумме `S` прибавляется новое слагаемое. Затем линия возвращается к символу «решение» для зацикливания.

4. Линейные алгоритмы

Программа на языке С

Рассмотрите простую программу на языке С/С++ (здесь и далее числа в скобках в начале строк не являются частью программы, а обозначают строки, на которые необходимо обратить особое внимание):

```
(1) /* Первая программа на С */
(2) #include <stdio.h>
(3) void main(void) {
(4)   printf("Hello World!\n");
(5) }
```

Строка 1:

```
/* Первая программа на С */
```

Строка 1 является комментарием. Комментарии служат для документирования программ и для того, чтобы сделать программы более удобочитаемыми.

Комментарии не оказывают никакого влияния на работу компьютера во время выполнения программы.

В языке С комментарии могут иметь следующие формы:

- многострочный комментарий, который может иметь любую длину; он отмечается символами `/*` в начале комментария и `*/` в конце:

```
/* Это комментарий, который занимает несколько строк
   программного кода. Весь текст, начинающийся с
   последовательности символов слеш-звёздочка и
   заканчивающийся последовательностью звёздочка-слеш,
   является комментарием */
```

- однострочный комментарий отмечается символами `//` в начале строки и продолжается до конца строки:

```
// это комментарий в одной строке
// это другой комментарий в одной строке
```

Строка 2:

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

Директива препроцессора **#include** в данном примере служит для подключения файлов стандартной библиотеки С.

Синтаксис директивы следующий:

```
#include <название_библиотеки>
```

Сколько библиотек необходимо подключить, столько раз должна быть описана директива **#include.**

Данная директива подключает библиотеку `stdio.h`, в которой содержатся функции для ввода и вывода информации.

Строка 3:

```
void main(void)
```

void `main(`**void**`)` – основная функция программы. Все программы, написанные на языке С, содержат основную функцию, которая должна иметь имя `main`. Функция `main` служит в качестве начальной точки выполнения программы. Она обычно управляет выполнением программы, вызывая другие её функции. Выполнение программы завершается, как правило, в конце функции `main`,

Функция – главный строительный блок программы. Программа может содержать одну или несколько функций, но функция **void** `main(`**void**`)` должна быть обязательно.

Открывающаяся фигурная скобка `{` обозначает начало тела каждой функции. Закрывающаяся фигурная скобка `}` обозначает окончание тела каждой функции. Пара скобок и текст между ними называется блоком программы.

Строка 4

```
printf("Hello World!\n");
```

заставляет компьютер выполнить действие – вывести на экран символьную строку «`Hello World!`» без кавычек. Последовательность символов `\n` не отобразится на экране, так как она является escape-последовательностью (обычно используются для указания действий, например, возврата каретки или табуляции, на терминалах и принтерах; используются для вывода непечатаемых символов, а также символов, которые обычно имеют специальное

значение, например, двойных кавычек "). Escape-последовательность \n позволяет перевести курсор на новую строку.

Таким образом, простейшая программа на языке С имеет следующий вид:

```
директивы_препроцессора
void main(void) {
  тело_функции
}
```

Тело функции состоит из описания данных и описания команд.

Описание данных

В простейшем случае инструкция описания данных (объявления переменных) в программе на языке С/С++ имеет следующий формат:

```
тип идентификатор;
тип идентификатор1, идентификатор2, ...,
    идентификаторN;
```

Здесь тип – тип данных, которые будет описывать объявленная переменная; идентификатор – имя этой переменной.

Переменная должна быть обязательно объявлена до её использования.

Ниже представлены некоторые типы, поддерживаемые компилятором Visual C++:

Имя типа	Размер в памяти, байт	Диапазон значений
int	4	от -2147483648 до 2147483647
unsigned int	4	от 0 до 4294967295
char	1	от -128 до 127
unsigned char	1	от 0 до 255
short	2	от -32 768 до 32 767
unsigned short	2	от 0 до 65 535

float	4	3,4E +/- 38 (7 знаков)
double	8	1,7E +/- 308 (15 знаков)

```
/* Переменные a, b, c объявлены имеющими тип int */
int a, b, c;

/* Переменные alp и bet объявлены имеющими тип float */
float alp, bet;
```

Для объявления констант (объектов языка, которые не могут изменять свои значения в процессе выполнения программы), используется ключевое слово const:

```
/* Описание константы G типа double (для хранения
значения ускорения свободного падения), имеющей
значение 9.8 */
const double G = 9.8;
```

После такого объявления попытка присвоить значение константной переменной `G` приведет к ошибке.

Оператор присваивания

Оператор присваивания – инструкция языка программирования, позволяющая назначать и изменять значения переменных.

Для обозначения оператора присваивания в С используется символ «=».

Синтаксис оператора присваивания:

```
переменная = выражение;
```

Пример:

```
(1)     a = 4;
(2)     b = a + 4;
(3)     b = b + b;
(4)     c = a + b;
```

В строке 1 в переменную `a` записывается значение `4`. После выполнения этого оператора переменная `a` будет содержать значение `4` до тех пор, пока переменная существует, либо пока ей не будет присвоено новое значение.

В строке 2 в переменную b записывается значение выражения a + 4. Так как на момент выполнения этого оператора a имеет значение 4, в переменную b запишется 8. После выполнения этого оператора переменная b будет содержать значение 8 до тех пор, пока переменная существует, либо пока ей не будет присвоено новое значение.

В строке 3 в переменную b записывается значение выражения b + b. Так как на момент выполнения этого оператора b имеет значение 8, в переменную b запишется 16. После выполнения этого оператора переменная b уже не содержит значение 8, так как ей было присвоено новое значение 16.

В строке 4 в переменную с записывается значение выражения a + b. Так как на момент выполнения этого оператора переменная a имеет значение 4, а переменная b имеет значение 16, в переменную с запишется значение 20.

Арифметические операторы

Для составления арифметических выражений в языке С существуют два вида операторов:

- унарные (одноместные; требующие одного операнда) операторы, присоединённые спереди к операнду (имени переменной или выражению);

- бинарные (двуместные; требующие двух операндов) операторы, соединяющие два операнда (имени переменных или выражения).

В язык С/С++ включены следующие унарные арифметические операторы (в примерах использования a и b могут являться именами переменных, константами или выражениями):

Оператор	Назначение	Пример использования
+	Унарный плюс	+(a - 7)
-	Унарный минус	-a
++	Оператор инкремента (увеличения на 1)	i++ ++i
--	Оператор декремента (уменьшения на 1)	i-- --i

Оператор унарного плюса, предшествующий выражению в скобках, не изменяет значение выражения, но выполняет восходящее приведение целого типа. Тип результата – это тип операнда более высокого уровня. О типах результатов см. подраздел «Автоматическое преобразование типов».

Оператор унарного минуса (арифметического отрицания) создает отрицательную форму своего операнда – выполняет обычное арифметическое преобразование. Если, например, переменная `a` имеет значение `5`, то `-a` будет иметь значение `-5`.

Операторы инкремента и декремента имеют префиксную форму (записываются перед операндом) и постфиксную форму (записываются после операнда).

В случае префиксной записи `++i` или `--i` увеличение или уменьшение значения переменной `i` выполняется до его использования в более сложном выражении:

```
/* в x записывается 1 */
x = 1;
/* сначала значение x увеличивается на 1 (становится
   равным 2), затем к новому значению x прибавляется 5;
   в результате в a будет записано 7 */
a = ++x + 5;
```

В случае постфиксной записи `i++` или `i--` увеличение или уменьшение значения переменной `i` выполняется после его фактического использования в выражении.

```
/* в x записывается 1 */
x = 1;
/* сначала значение к значению x прибавляется 5,
   результат 6 записывается в a; после этого значение x
   увеличивается и становится равным 2 */
a = x++ + 5;
```

В язык C/C++ включены следующие бинарные арифметические операторы:

Оператор	Назначение	Пример использования
*	Умножение	`a * b`

/	Деление	a / b
%	Вычисление остатка	a % b
+	Сложение	a + b
-	Вычитание	a - b

Оператор умножения выполняет умножение двух операндов.

Оператор деления выполняет деление первого операнда на второй. Следует иметь в виду следующее: если два целочисленных операнда делятся друг на друга, и результат не является целым числом, дробная часть результата отбрасывается (в некоторых компиляторах результат деления может определяться другим способом, если один из операндов является отрицательным числом). Результат деления на 0 не определен. Во время трансляции или выполнения фиксируется ошибка.

Результатом выполнения оператора вычисления остатка является остаток от деления первого операнда на второй. Если правый операнд равен 0, результат не определён. Если один из операндов отрицательный, знак результата будет совпадать со знаком делимого (в некоторых компиляторах знак результата может определяться другим способом).

Оператор сложения выполняет сложение двух операндов.

Оператор вычитания вычитает второй операнд из первого.

Автоматическое преобразование типов

Если в состав арифметического или логического выражения входят операнды различных типов, то компилятор автоматически выполняет их приведение к общему типу:

1) если операция выполняется над данными двух различных типов, обе величины приводятся к «высшему» типу;

2) в операторе присваивания конечный результат вычисления выражения в правой части приводится к типу переменной, которой должно быть присвоено значение.

Последовательность имен типов, упорядоченных от «высшего» типа к «низшему», выглядит следующим образом: **`double`**, **`float`**, **`long`**, **`int`**, **`short`** и **`char`**. Применение ключевого слова

unsigned повышает ранг соответствующего типа данных со знаком.

Функции printf и scanf

При выполнении заданий из данного сборника следует использовать функцию форматированного ввода `scanf` и функцию форматированного вывода `printf`:

```
printf("Hello World!\n");
```

Функция форматированного вывода `printf` – функция из стандартной библиотеки `stdio.h`, которая позволяет осуществлять вывод на экран. Данные в скобках – параметры функции – то, что должна вывести функция. Точка с запятой обозначает конец оператора. Каждый оператор должен заканчиваться точкой с запятой.

Функция форматированного вывода `printf` переводит данные из внутреннего кода в символьное представление и выводит полученные изображения символов на экран.

Оператор вызова функции (здесь и далее в квадратных скобках указана необязательная часть оператора, если не сказано иное):

```
printf(форматная_строка[, список_аргументов]);
```

1) `форматная_строка` ограничена кавычками и может включать произвольный текст, управляющие символы и спецификации преобразования данных.

Обобщенный формат управляющих символов в форматной строке (показано для одного аргумента; каждому выводимому аргументу соответствует своя спецификация):

```
%[Ширина_поля][.Точность]Тип
```

`Ширина_поля` – целое положительное число, определяющее минимальное количество позиций для вывода значения. `Точность` – целое положительное число, указывающее максимальное количество символов, которые будут напечатаны в строках, количество значащих цифр или число цифр после десятичной точки

для значений с плавающей точкой, или минимальное число цифр которые будут напечатаны для целых значений.

`Тип` указывает, как должен интерпретироваться аргумент: как символ, строка, указатель, целое число или число с плавающей точкой. Символ `тип` – единственное необходимое поле спецификации формата; он находится после всех необязательных полей.

Примеры спецификаций:

- `%d` – для вывода целого числа;
- `%5d` – для вывода числа с использованием как минимум 5 позиций;
- `%3.2f` – для вывода числа типа с плавающей точкой с двумя знаками после запятой и как минимум тремя позициями под целую часть.

Некоторые значения символов типа:

Символ типа	Аргумент	Формат вывода
`c`	Символ	Однобайтовый символ
`d`	Целое	Десятичное целое число со знаком.
`i`	Целое	Десятичное целое число со знаком.
`f`	С плавающей точкой	Значение со знаком, имеющее формат [–]dddd.dddd где dddd — одна или несколько десятичных цифр. Количество цифр перед десятичной точкой зависит от порядка числа, а количество цифр после десятичной точки зависит от указанной точности.
`s`	Строковое	Определяет однобайтовую строку символов (массив типа char или указатель на char). Символы отображаются до первого символа с кодом 0 или до тех пор, пока не

		будет достигнуто значение точность.
p	Указатель	Отображает аргумент как адрес в шестнадцатеричных цифрах.

2) список аргументов – выражения через запятую, например: a, b + c, 3 – a, sin(f).

Каждая спецификация в форматной строке определяет формат вывода соответствующего аргумента из списка аргументов:

```
printf("%d плюс %d равно %d", a, b, a + b);
```

Значение переменной a будет выведено на месте первого вхождения %d в форматную строку; значение переменной b будет выведено на месте второго вхождения %d в форматную строку; значение выражения a + b будет выведено на месте третьего вхождения %d в форматную строку.

Функция форматированного ввода scanf выполняет чтение кодов, вводимых с клавиатуры, воспринимает коды, преобразует их во внутренний формат и передает программе. Возможно влияние на правила интерпретации входных кодов при помощи форматной строки.

Оператор вызова функции:

```
scanf(форматная_строка, список_аргументов);
```

Форматная строка аналогична форматной строке в функции printf. В отличие от функции printf, в функции scanf аргументами могут быть только непосредственные адреса переменных в памяти – адреса ячеек, в которые нужно записать считываемые значения.

Для определения адреса используется оператор взятия адреса объекта – & (амперсанд).

Выражение для получения адреса переменной: &имя_переменной. Если name – имя некоторой переменной, то &name – адрес этой переменной.

Каждая спецификация в форматной строке определяет формат ввода соответствующего аргумента из списка аргументов.

```
scanf("%d%f%f", &n, &z, &x);
```

Если переменная `n` описана как целая, а `z` и `x` как вещественные, то после чтения с клавиатуры последовательности символов `18 18 -0.431` переменная `n` получит значение `18`, `z` – `18.0`, `x` – значение `-0.431`.

При чтении входных данных функция `scanf` воспринимает в качестве разделителей полей данных «обобщенные пробельные символы» – пробелы, символы табуляции, символы перевода строки.

Функция `scanf` одинаково правильно введёт в программу данные, представленные, например, такими способами:

А) 18 18 -0.431

Б) 18 18 -0.431

В) 18 18 -0.431

Г)

18

18

-0.431

Варианты заданий

Написать программу вычисления заданного арифметического выражения (значение `x` задается с клавиатуры). Для использования математических функций следует использовать библиотеку `math.h`.

1.	$y = \dfrac{x}{1 + \dfrac{x^2}{1 + \dfrac{3x^3}{1 - \dfrac{x}{1+\lvert x\rvert}}}}$	2.	$y = \dfrac{e^{\frac{x^2}{\sqrt{2}}}\cos\left(\frac{x}{2} + \frac{\pi}{8}\right)}{\sqrt{2\pi\lvert x\rvert} + 1}$
3.	$y = \dfrac{\sin 2x \cdot \cos 2x \cdot e^x}{\sqrt[7]{x^4 - 0{,}58\sin x} + 1}$	4.	$y = \dfrac{\sqrt{\sqrt[3]{\lvert x\rvert} + \lvert\sin(2x+1)\rvert}}{\cos^2\dfrac{\pi x}{2} + \operatorname{tg}\dfrac{\pi}{3}}$
5.	$y = \dfrac{5\sin\lvert 0{,}2x\rvert - \sqrt[3]{\lvert x\rvert}}{6\sqrt{3\ln^2\lvert x\rvert}\operatorname{tg}\dfrac{x}{2}}$	6.	$y = \dfrac{(e^{\sin x} + \operatorname{tg}x)^3}{\sqrt[3]{\lvert(x^2 - 0{,}85)(e^x - \sin x)\rvert}}$

7.	$y = \frac{\mathrm{tg}^2 \sin^3 x^{0,5} + \sqrt[3]{\lvert 3x^3 - 4x^2 \rvert}}{6\sqrt{\lg 2\lvert x \rvert + e^{0,1x}}}$	8.	$y = \frac{\sqrt[5]{5x^2} + 0,6\sin^2 \frac{x}{2}}{\cos^3 x^{2,5} 6 \ln(\lvert x+2 \rvert)}$
9.	$y = \frac{\sqrt{\ln\left\lvert 0,5x - \frac{\pi}{2}\right\rvert} - 2\sin^2 x^x}{e^{0,7x} 3\mathrm{arctg} x^{1,2}}$	10.	$y = \frac{\mathrm{arctg}\left(\frac{x}{2}+1\right) + \log_2 x^{0,2}}{\sqrt[5]{x^{2x}} + e^{0,2x+1}}$
11.	$y = \frac{x^{-x\sin^2 x} + \ln 6,23x^2}{\sqrt[3]{x^2 + 24,8}}$	12.	$y = \frac{\cos\sqrt{x^3} - \ln\lvert x - 3,47 \rvert}{\cos^2 \sin\frac{x}{3} + 5\mathrm{tg}\frac{x}{4}}$
13.	$y = \frac{\ln\lvert \sin^2 x - \cos x^2 \rvert + \sqrt[3]{0,5x^2}}{6\sqrt{3\ln^2 \lvert x \rvert}\,\mathrm{tg}\frac{x}{2} + 1}$	14.	$y = \frac{\arcsin(1-x^2) + \arccos(1-x)}{0,2e^x - 0,1e^{-x}}$
15.	$y = \frac{\frac{\sqrt[3]{x^2 + e^{0,1x}}}{x}}{0,5x + \mathrm{tg}\frac{x}{2} + 1} - \sqrt{x} + 1$	16.	$y = \frac{2^{-\cos x} + \sqrt[3]{\lvert (0,75 - x^3)x \rvert}}{2^x - 3\cos x}$
17.	$y = \frac{\sqrt[4]{\lvert x^3 - 6\sin\frac{x}{2} \rvert} + 2,3}{\ln\left\lvert \frac{x}{2} - 4 \right\rvert - e^{0,3x}}$	18.	$y = \frac{\mathrm{arctg}\frac{x}{5} + \sqrt[5]{x^3}}{e^{\frac{x}{2}} \sin^2 x^{0,5} + 1}$
19.	$y = \frac{\sqrt[3]{(e^x - 8,35x^2)(x+1)}}{2^{\sin x - \ln\sin x}}$	20.	$y = \frac{3\cos^2 \sin^x x + 6\lg 0,4x}{0,1\sqrt{2\lvert x^3 \rvert} - e^{0,8x}}$

Пример выполнения задания

$$y = \frac{\cos\left(\frac{x}{2} + \frac{\pi}{8}\right)}{\sqrt{2\pi x}\sin\left(x + \sqrt{2\pi x}\right) + \frac{2}{3}}$$

```
      /* подключение библиотеки stdio.h */
(1)   #include <stdio.h>
      /* подключение библиотеки math.h */
(2)   #include <math.h>

      /* Основная функция */
(3)   void main(void) {
        /* объявление вещественных переменных x и y */
(4)     double x, y;
(5)     const double PI = 3.141592;
```

```
        /* вывод на экран запроса значения x */
(6)     printf("Введите значение x: ");
        /* запись значения, введенного пользователем, в
           переменную x */
(7)     scanf("%lf", &x);
        /* вычисление значения выражения и запись
           результата в y */
(8)     y = cos(x / 2.0 + PI / 8.0) / (sqrt(2.0 *
            PI) * sin(x + sqrt(2.0 * PI * x)) + 2.0
            / 3.0);
        /* вывод результата на экран */
(9)     printf("\ny = %.5f", y);
    }
```

Строка 1: подключение библиотеки `stdio.h` для того, чтобы использовать функцию ввода `scanf` и вывод `printf`.

Строка 2: подключение библиотеки `math.h`, содержащей математические функции (в том числе `cos` для вычисления значения косинуса, `sin` для вычисления значения синуса, `sqrt` для вычисления значения квадратного корня).

Строка 3: начало главной функции `main`.

Строка 4: объявление переменных `x` и `y` типа **`double`**. В переменную `x` будет считываться значение, вводимое пользователем с клавиатуры, в переменную `y` будет записано вычисленное значение выражения.

Строка 5: объявление константы `PI`.

Строка 6: вывод на экран строки `"Введите значение x: "` (без кавычек).

Строка 7: запись данных, введённых пользователем с клавиатуры, в переменную `x`. Так как функция `scanf` требует указания адреса в памяти, по которому следует записать считанное значение, перед переменной `x` стоит операция взятия адреса – `&`.

Строка 8: вычисление значения выражения и запись результата в переменную `y`. При вычислении внимание следует обратить на то, что для получения результата деления `2` на `3` используется не запись `2 / 3`, в которой используются целочисленные константы, а запись `2.0 / 3.0` с вещественными константами. При делении целочисленных констант получается целочислен-

ный результат, так как выполняется автоматическое преобразование типа результата. Таким образом результатом деления `2` на `3` является `0`, а при делении `2.0` на `3.0` результатом является `0.666…`, что и предполагается в данном примере.

Строка 9: вывод на экран строки `"y = "`, следом за которой будет выведено значение переменной `y`. Так как строка формата начинается с escape-последовательности `\n`, строка `"y = "` будет выводиться с новой строки. Так как в спецификации `%.5f` указано значение точности `.5`, независимо от фактического значения `y`, на экран будет выведено 5 знаков после запятой.

5. Условный оператор

Группа одного или большего числа простых операторов называется составным оператором.

```
/* один простой оператор */
c = 5;
/* другой простой оператор */
a = c + 6;

/* Один составной оператор */
{
  c = 5;
  a = c + 6;
}
```

Условный оператор позволяет запрограммировать алгоритмическую конструкцию «ветвление».

Синтаксис условного оператора (здесь и далее в квадратных скобках описывается необязательная часть оператора, если не указано иное):

```
if (логическое_выражение)
     оператор1
[ else
     оператор2 ]
```

В данном описании `логическое_выражение` – произвольное выражение, относительно которого можно сказать, истинно оно ли ложно; `оператор1` и `оператор2` – простые или составные операторы; **`if`** и **`else`** – ключевые слова.

Для формирования логических выражений как правило используются бинарные операторы отношения и равенства. Эти операторы сравнивают свои первые и вторые операнды для проверки истинности указанного отношения. Результат выражения отношения равен `1`, если проверенное отношение истинно, или `0`, если отношение ложно. Результат имеет тип **`int`**.

Оператор	Отношение
<	Первый операнд меньше второго операнда
>	Первый операнд больше второго операнда

<=	Первый операнд меньше или равен второму операнду
>=	Первый операнд больше или равен второму операнду
==	Первый операнд равен второму операнду
!=	Первый операнд не равен второму операнду

В логических выражениях может применяться унарный оператор логического отрицания `!`, который возвращает `0`, если операнд является истинным (не равен `0`), и `0`, если операнд является ложным (равен `0`). Например, если выражение `a < b` является ложным, то `!(a < b)` является истинным.

При выполнении условного оператора вычисляется значение логического выражения, стоящего в скобках, и полученный результат сравнивается с нулем, что соответствует его интерпретации в терминах «истина» / «ложь» («ложь» – значение 0; «истина» – значение, отличное от 0).

Если результатом является значение «истина», то выполняется `оператор1`.

В противном случае осуществляется переход к `оператор2` в else-части условного оператора.

Например, данный фрагмент программы позволяет вывести на экран строку «`Зачтено.`», если значение переменной `grade` больше `60`:

```
if (grade > 60) {
   printf("Зачтено. ");
}
```

В следующем фрагменте программы условный оператор имеет ветвь **`else`**:

```
if (a % 2 == 0) {
   printf("%d - чётное", a);
} else {
   printf("%d - нечётное", a);
}
```

В данном примере, если `a` является чётным (если остаток от деления `a` на `2` равен `0`), выполняется оператор `printf("%d -`

чётное", a);, в противном случае (если остаток от деления a на 2 не равен 0) выполняется оператор printf("%d - нечётное", a);.

В случае, если необходимо оценить истинность (ложность) нескольких выражений одновременно, используются логические операторы:

Оператор	Описание
&&	Логический оператор И возвращает значение 1 (истина), если оба операнда имеют ненулевое значение. Если любой из операндов имеет значение 0, результат равен 0. Используется, если необходимо убедиться, что несколько отношений являются истинными одновременно
||	Логический оператор ИЛИ возвращает значение 0 (ложь), если оба операнда имеют значение 0. Если любой из операндов имеет ненулевое значение, результат будет равен 1. Используется, если необходимо убедиться, что хотя бы одно из отношений является истинным.

Значение выражения $\sqrt{\frac{x}{b}}$ можно вычислить в случае, если знаменатель не равен 0, и если значение подкоренного выражения является неотрицательным. Для проверки данного сложного условия можно построить следующее логическое выражение при помощи оператора &&: (b != 0) && (x * b >= 0). Это выражение будет истинным, когда обе его части (часть (b != 0) и часть (x * b >= 0)) будут истинными.

Варианты заданий

Написать программу, используя условный оператор.

1. Определить и вывести на экран, верно ли, что все цифры заданного трехзначного числа различны.

2. Даны вещественные числа x, y, z. Вычислить и вывести на экран $\max\{x, y, z\}/\min\{x, y, z\}$.

3. Даны три вещественных числа a, b, c. Определить и вывести на экран, имеется ли среди них хотя бы одна пара чисел, имеющих противоположные значения.

4. Даны неотрицательное целое число a, положительное целое число b, числа r и s. Определить и вывести на экран, верно ли, что при делении a на b, получается остаток, равный r либо s.

5. Даны три различных целых числа. Вывести на экран среднее из них (большее наименьшего, но меньшее наибольшего).

6. Даны вещественны числа a, b, c, d. Если a <= b <= c <= d, то каждое число заменить наибольшим из них. Если a > b > c > d, то числа оставить без изменения; в противном случае все числа заменить их квадратами. Вывести получившиеся числа.

7. Определить, верно ли, что для заданного четырехзначного числа выполняется соотношение: сумма первой и последней цифры числа равна разности второй и третьей цифры.

8. Даны числа a, b, c, d. Известно, что одно отлично от других, равных между собой. Вывести это число на экран.

9. Определить и вывести на экран, является ли треугольник, заданный координатами вершин равнобедренным.

10. Определить и вывести на экран, является ли заданное шестизначное число счастливым (таким, что сумма первых трех его цифр равна сумме последних трех).

11. Найти произведение двух наименьших из трех различных чисел.

12. Три точки заданы своими координатами. Найти и вывести на экран наиболее удаленные друг от друга точки.

13. Определить и вывести на экран, является ли треугольник, заданный координатами вершин, равносторонним.

14. Даны две точки. Определить, какая из них находится ближе к началу координат.

15. Решить биквадратное уравнение (уравнение вида $Ax^4+Bx^2+C=0$). Результат вывести на экран.

16. Даны вещественные числа a, b, c, d. Найти и вывести на экран min{max{a, b}, max{c, d}}.

17. Задано четырёх число. Определить и вывести на экран, упорядочены ли по возрастанию цифры в записи этого числа.

18. Найти и вывести на экран сумму двух наибольших из трех различных чисел.

19. Дано число x. Напечатать в порядке возрастания числа $\sin(x)$, $1+|x|$, $(1+x^2)^2$.

20. Найти сумму различных чисел из трёх введённых чисел.

Пример выполнения задания

Дано трехзначное число; определить, верно ли, что все его цифры являются нечётными числами.

Первый решения задачи:

```
   #include <stdio.h>

   /* Основная функция */
   void main(void) {
     /* объявление целочисленных переменных a, a1, a2,
        a3 */
     int a, a1, a2, a3;
     /* вывод на экран запроса трехзначного числа */
     printf("Введите трехзначное целое число: ");
     /* запись введенного числа в переменную a */
     scanf("%d", &a);
     /* определение последней цифры числа и запись её в
        a1 */
(1)  a1 = a % 10;
     /* определение второй цифры числа и запись её в a2 */
(2)  a2 = a / 10 % 10;
     /* определение первой цифры числа и запись её в a3 */
```

```
(3)  a3 = a / 100;
     /* проверка, являются ли все три числа нечетными */
(4)  if (a1 % 2 != 0 && a2 % 2 != 0 && a3 % 2 != 0) {
       /* вывод сообщения, что все числа нечётные */
(5)    printf("\nВсе цифры числа %d нечётные.\n", a);
     } else {
       /* вывод сообщения, что не все числа нечётные */
(6)    printf("\nНе все цифры числа %d нечётные.\n", a);
     }
   }
```

Второй вариант решения задачи (присутствуют только значимые комментарии):

```
#include <stdio.h>

/* Основная функция */
void main(void) {
  int a, a1, a2, a3;
  printf("Введите трехзначное целое число: ");
  scanf("%d", &a);
  a1 = a % 10;
  a2 = a / 10 % 10;
  a3 = a / 100;
  /* проверка, является ли последнее число четным */
  if (a1 % 2 == 0) {
    /* если первое число четное,
       выводится соответствующее сообщение */
    printf("\nНе Все цифры числа %d нечётные.\n", a);
  /* если первое число нечётное, проверяется второе */
  } else if (a2 % 2 == 0) {
    /* если второе число четное,
       выводится соответствующее сообщение */
    printf("\nНе Все цифры числа %d нечётные.\n", a);
  /* если второе число нечётное, проверяется третье */
  } else if (a3 % 2 == 0) {
    /* если третье число четное,
       выводится соответствующее сообщение */
    printf("\nНе все цифры числа %d нечётные.\n", a);
  } else {
    /* если и третье число нечетное,
       выводится соответствующее сообщение */
    printf("\nВсе цифры числа %d нечётные.\n", a);
  }
}
```

В первом варианте программы используется один условный оператор для проверки нечётности всех трёх чисел одновременно.

Строка 1: в `a1` записывается последняя цифра числа (цифра, стоящая в младшем разряде числа). Для получения этой цифры используется операция взятия по модулю `%`. Если любое целое число взять по модулю `10`, результатом будет последняя цифры этого числа (остаток от деления числа на `10`).

Строка 2: в `a2` записывается вторая цифры числа. Для этого сначала исходное число делится на `10` (будет выполнено целочисленное деление, так как и делимое – переменная `a`, и делитель – число `10` являются целыми числами). Взятие результата по модулю `10` даст результатом вторую цифру исходного числа.

Строка 3: в `a3` записывается старшая цифра числа. Для этого исходное трёхзначное число делится нацело на `100`.

Строка 4: условный оператор проверяет, является ли заданное условие истинным. Проверяется следующее условие: `a1 % 2 != 0 && a2 % 2 != 0 && a3 % 2 != 0`. Это логическое выражение следует читать так: остаток от деления `a1` на `2` не равен `0` и остаток от деления `a2` на `2` не равен `0`, и остаток от деления `a3` на `2` не равен `0`. Идея проверки заключается в том, что если остаток от деления целого числа на `2` равен `0`, то такое число является чётным, в противном случае – нечётным. Так как в данном условии используется логический оператор `&&` («И»), проверяется, являются ли все три числа нечётными одновременно. Если условие истинно (все три числа нечётные), то выполняется строка 5, иначе (хотя бы одно из чисел является чётным) выполняется строка 6.

Во втором варианте программы каждое число проверяется отдельным условным оператором. При этом проверяется, является ли текущее число чётным. Если попадается чётное число, сразу выводится сообщение о том, что не все числа являются нечётными. Если рассматриваемое число нечётное, управление получает следующий условный оператор, проверяющий следующее число. Если ни одно из чисел не является чётным, управление передастся else-ветви последнего условного оператора, который выведет сообщение о том, что все числа нечётные.

6. Оператор выбора

Оператор выбора является специальным случаем условного оператора и позволяет осуществлять многовариантный выбор, заменяя группу вложенных условных операторов.

Оператор выбора имеет следующий синтаксис:

```
switch (выражение) {
    case значение1: операторы1
    case значение2: операторы2
    …
[  default: операторыN   ]
}
```

Любая **case-** или **default-** часть может отсутствовать, а также могут быть опущены операторы в любой из этих частей.

Здесь **switch**, **case** и **default** – ключевые слова; выражение – произвольное выражение целого типа; значение – константное выражение, вычисляемое во время трансляции программы; операторы – набор операторов.

Оператор выбора работает следующим образом.

Предварительно вычисленное значение выражения в круглых скобках сравнивается со значением во всех вариантах **case**, и управление передается той группе операторов, которая соответствует найденному значению.

В том случае, когда значение ни одного из константных выражений не совпало со значением выражения значение, выполняются операторы, связанные с меткой **default**, а при её отсутствии управление переходит к оператору, непосредственно следующему за оператором **switch**.

```
switch (operation) {
    case '+':
        z = x + y;
    case '-':
        z = x - y;
    case '*':
        z = x * y;
    case '/':
        z = x / y;
}
```

В данном примере предполагается, что в зависимости от значения переменной `operation`, имеющей тип **`char`**, выполнится конкретное арифметическое действие. При значении `'+'` – сложение `x` и `y` и запись результата в `z`, при значении `'-'` – вычитание из `x` значения `y` и запись результата в `z`.

Но результат выполнения данного оператора будет другим. В операторе **`switch`** будет выполняться каждый оператор, начиная с метки **`case`**, значение константного выражения которой совпало со значением управляющего выражения, вплоть до конца оператора **`switch`**, включая операторы метки **`default`**.

Таким образом, в случае, если переменная `operation` имеет значение `'-'`, то выполнение начнется с метки case `'-'`, при этом будут выполняться все операторы до конца оператора **`switch`**: `z = x - y;`, затем `z = x * y;`, затем `z = x / y;`.

Такой эффект может быть желателен при решении некоторых задач. Например, следующий оператор позволяет вывести на экран строку «`Это цифра.`» в случае, если значением символьной переменной `key` является символ, обозначающий цифру:

```
switch (key) {
    case '0':
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
        printf("Это цифра");
}
```

Для остановки выполнения оператора **`switch`** и передачи управления оператору, следующему за оператором **`switch`**, используется оператор **`break`**.

Следующий оператор **`switch`**, содержащий операторы **`break`**, будет выполнять поставленную ранее задачу корректно:

```
switch (operation) {
```

```
    case '+':
        z = x + y;
        break;
    case '-':
        z = x - y;
        break;
    case '*':
        z = x * y;
        break;
    case '/':
        z = x / y;
}
```

Варианты заданий

Написать программу, выводящую меню, содержащее пункты, перечисленные в задании, и выполняющую действия, соответствующие пунктам меню.

1. Перевод длины из метров в дециметры, сантиметры, миллиметры, микрометры, нанометры.

2. Перевод длины из метров в декаметры, гектометры, километры, мириаметры, мегаметры.

4. Перевод длины из метров в мили (1 миля = 1609 м), версты (1 верста = 1066,8 м), сажени (1 сажень = 2,1336 м), аршины (1 аршин = 0,7112 м), футы (1 фут = 0,3048 м).

5. Перевод количества информации из байтов в биты, килобайты, мегабайты, гигабайты, терабайты.

6. Вычисление значения следующих функций для заданной величины: синус, косинус, тангенс, котангенс, гиперболический синус.

7. Перевод массы из килограммов в стоуны (1 стоун = 6,35 кг), фунты (1 фунт = 0,453 кг), унции (1 унция = 0,02835 кг), драхмы (1 драхма = 0,001772 кг).

8. Перевод массы из килограммов в пуды (1 пуд = 16,38 кг), лоты (1 лот = 0,012797 г), золотники (1 золотник = 0,00426575 кг), безмены (1 безмен = 1,022 кг).

9. Перевод объема из литров в баррели (1 баррель = 158,988 л), пинты (1 пинта = 0,473176 л), американская кварта для жидкостей (0,9463 л), английская имперская кварта (1,1365 л).

10. Перевод температуры из градусов Цельсия в градусы Фаренгейта ($t_F = \frac{9}{5}t_C + 32$), кельвины ($t_K = t_C - 273{,}15$), градусы Реомюра ($t_R = \frac{5}{4}t_C$), градус Ранкина ($t_{Ra} = \frac{9}{5}(t_C + 273{,}15)$).

11. Перевод температуры из градусов Цельсия градусы Ранкина ($t_{Ra} = \frac{9}{5}(t_C + 273{,}15)$), градусы Ньютона ($t_\text{Т} = \frac{33}{100}t_C$), градусы Рёмера ($t_{Ro} = \frac{21}{40}t_C + 7{,}5$), градус Делиля ($t_D = \frac{3}{2}(100 - t_C)$).

12. Перевод мощности из ватт в калории в час (1 калория в час = $1{,}16\times10^{-3}$ Вт), электрическая лошадиная сила (1 электрическая лошадиная сила = 746 Вт), джоуль в час (1 джоуль в час = 1/3600 Вт).

13. Перевод давления из мегапаскалей в бары (10^{-5} Па), физическая атмосфера ($9{,}8692\cdot10^{-6}$ Па), миллиметр ртутного столба ($7{,}5006\cdot10^{-3}$ Па), миллиметр водяного столба ($1{,}0197\cdot10^{-4}$ Па).

14. Перевод единиц измерения скорости передачи данных из мегабит в секунду. Пункты меню: килобит в секунду, килобайт в секунду, гигабит в секунду, мегабайт в секунду, мегабайт в минуту.

15. Перевод плотности из килограмм на метр кубический в тонны на метр кубический, килограммы на литр, граммы на сантиметр кубический, граммы на миллилитр.

16. Перевод энергии из джоулей в калории (1 джоуль = 0,238846 калорий), килограммы-силы-метр (1 джоуль = 0,101972 кгс·м), киловатты-часы (1 кВт·ч = 3,6 мегаджоулей), эрги (1 джоуль = 10^7 эрг).

17. Перевод угловой меры из градусов в радианы (1 радиан = $180/\pi$°), в грады (1 град = 0,9°), в минуты (1 минута = 1/60°), в секунды (1 секунда = 1/3600°).

18. Перевод объемной скорости потока из кубических метров в секунду кубические метры в час, в кубические метры в минуту, в литры в секунду, в литры в минуту, в литры в час.

19. Вычисление значения функции для заданной величины. Пункты меню: десятичный логарифм, натуральный логарифм, двоичный логарифм.

20. Вычисление значения функции для заданной величины. Пункты меню: арксинус, арккосинус, арктангенс, гиперболический тангенс.

Пример выполнения задания

Перевод дней в минуты, часы, месяцы, годы.

```
   #include <stdio.h>
   /* основная функция */
   void main(void) {
     /* объявление переменной days для хранения
        количества дней */
     int days;

     /* объявление переменной key для хранения символа,
        введенного пользователем */
(1)  char key;
     /* запрос количества дней */
     printf("Введите количество дней: ");
     /* сохранение количества дней */
     scanf("%d", &days);
     /* вывод меню на экран */
     printf("\n1. Перевести в минуты.");
     printf("\n2. Перевести в часы.");
     printf("\n3. Перевести в месяцы.");
     printf("\n4. Перевести в годы.");
     /* вывод запроса на выбор действия */
     printf("\n\n Выберите действие и нажмите ВВОД: ");

     /* сохранение кода введенного символа */
```

```
(2)   scanf(" %c", &key, 1);

      /* проверка значение переменной key */
(3)   switch(key) {
        /* обработка случая, когда значение key – '1' */
(4)     case '1':
(5)       printf("\nКоличество минут: %d\n", days *
                  1440);
(6)       break;
        /* обработка случая, когда значение key – '2' */
(7)     case '2':
(8)       printf("\nКоличество часов: %d\n", days * 24);
(9)       break;

        /* обработка случая, когда значение key – '3' */
(10)    case '3':
(11)      printf("\nКоличество месяцев: %.2lf\n", days
                    / 30.42);
(12)      break;
        /* обработка случая, когда значение key – '4' */
(13)    case '4':
(14)      printf("\nКоличество лет: %.2lf\n", days /
                  365.0);
(15)      break;
        /* обработка случая, когда key имеет другое
           значение */
(16)    default:
(17)      printf("\nДействие с выбранным номером
                  отсутствует.\n");
      }
    }
```

Предложенная программа запрашивает у пользователя количество дней – число, которое будет переведено в другие единицы измерения, затем выводит на экран меню, запрашивает у пользователя номер действия, которое необходимо выполнить, и в зависимости от того, какое из четырёх возможных действий выбрал пользователь, переводит число из одних единиц измерения в другие; если введён номер, не соответствующий никакому действию, программа оповещает об этом пользователя.

Строка 1: объявление переменной `key` типа **`char`** для записи номера действия, выбранного пользователем. Не смотря на то, то

для выбора операции водится число от 1 до 4, для его хранения выбран тип **char**. Это связано, во-первых, с тем, что данные типа **char** занимают в памяти 1 байт, во-вторых, использование данного типа позволит, например, при расширении программы вводить операции с номерами, представленными не только цифрами, но и буквами (`a. b. c. ...`). Впрочем, для хранения номера действия, по своему усмотрению, программист может выбрать любой целочисленный тип данных.

Строка 2: для записи в память кода введённого символа, представляющего номер действия, используется небольшая хитрость: вместо ожидаемой форматной строки `"%c"` используется строка `" %c"` (с пробелом перед `%c`). Это сделано для того, чтобы функция `scanf` не посчитала символ перевода строки символом, код которого нужно записать в память, а восприняла его как пробельный символ.

Строка 3: начало оператора выбора **switch**, содержащее выражение, значение которого будет отыскиваться среди меток **case**. В данном случае отыскивается значение переменной `key`. Так как данная переменная хранит код символа, в качестве значений, приписанных к меткам **case**, могут выступать как символьные константы (`'1', '2', '3', '4'`), так и сами коды символов (`49, 50, 51, 52`).

Строки 4, 7, 10, 13: метки **case** с описанными ожидаемыми значениями переменной `key`. Переход будет осуществлен к той метке **case**, значение которой совпадает со значением переменной `key`.

Строки 5, 8, 11, 14: операторы, которые выполняются в зависимости от значения переменной `key`.

Строки 6, 9, 12, 15, 18: операторы **break**, которые заставляют завершить оператор выбора **switch** сразу после выполнения действий, описанных для конкретной метки. Отсутствие этих операторов приведет к тому, что после перехода по определенной метке начнут выполняться все операторы до конца оператора **switch**. Например, если буду отсутствовать операторы **break**, а

значением `key` будет `'2'`, то вместо выполнения одного оператора `printf("\nКоличество часов: %d\n", days * 24)` будут последовательно выполняться все следующие операторы:

```
printf("\nКоличество часов: %d\n", days * 24);
printf("\nКоличество месяцев: %.2lf\n", days / 30.42);
printf("\nКоличество лет: %.2lf\n", days / 365.0);
printf("\nДействие с выбранным номером
отсутствует.\n");
```

Строка 16: переход к метке **`default`** будет произведен в случае, если ни одна из меток **`case`** не соответствует значению переменной `key`.

7. Оператор цикла

Операторы цикла позволяют реализовать алгоритмическую структуру «цикл».

Цикл с параметром

В случае, если в программе необходимо повторить определенные действия заранее известное число раз, используется цикл с параметром **for** (цикл с заданным числом повторений)

Цикл **for** имеет следующий синтаксис:

```
for (выражение1; выражение2; выражение3)
   оператор
```

Здесь `оператор` – простой или составной оператор, называемый телом цикла.

Перед началом выполнения цикла вычисляется значение инициализирующего выражения `выражение1` и значение выражения условного выражения `выражение2` (условия продолжения итераций).

Если значение выражения `выражение2` истинно (отлично от нуля), то:

1) выполняется тело цикла;

2) вычисляется корректирующее выражение `выражение3`;

3) вновь проверяется истинность выражения `выражение2`.

Этот процесс повторяется до тех пор, пока условие продолжения итераций не станет ложным (равным нулю), после чего управление передается оператору, следующему за оператором **for**.

В случае отсутствия выражения `выражение2`, ему условно присваивается постоянное значение «истина», что равносильно бесконечному циклическому процессу.

Как правило, `выражение1` – объявление с инициализацией переменной-параметра (счётчика цикла), либо инициализация переменной-параметра;

`выражение2` – условное выражение, часто – проверка достижения параметром конечного значения;

`выражение3` – изменение параметра (операция инкремента, декремента и пр.).

Рассмотрим конкретный оператор **`for`**:

```
for (int i = 0; i < 10; i++)  {
   ...
}
```

Здесь в качестве счетчика выступает переменная `i`. В качестве начального значения ей присваивается значение `0`. Конечным значением параметра является значение `9`, так как при таком значении условие `i < 10` является истинным, а при следующем значении `10` условие истинным являться не будет. После каждого выполнения тела цикла значение переменной-параметра увеличивается на `1`, так как значение параметра изменяет оператор инкремента `i++`.

Следующая программа позволяет вывести на экран числа в диапазоне от `1` до `100`:

```
void main(void) {
  for (int i = 1; i <= 100; i++) {
    printf("%d\n", i);
  }
}
```

В операторе **`for`** выражения 1 и 3 могут являться списками выражений, разделённых запятыми:

```
for (i = 0, j = 100; i < j; i++, j--) {
    printf("%d %d\n", i, j);
}
```

В данном операторе используется два параметра: `i` с начальным значением `0` и `j` с начальным значением `100`. Цикл выполняется, пока значение параметра `i` меньше значения параметра `j`. При этом после каждой итерации значение `i` увеличивается на `1`, а значение `j` уменьшается на `1`.

Цикл с предусловием и цикл с постусловием

В случае, если заранее неизвестно, какое количество раз необходимо повторить определенные действия, используются циклы с условиями – такие, которые выполняются, пока значение проверяемого условного выражения является истинным.

Цикл с предусловием **while** выполняется, пока значение проверяемого логического выражения является истинным (не равно `0`):

```
while (логическое_выражение)
   оператор
```

Здесь `оператор` – простой или составной оператор, называемый телом цикла.

Значение параметра `логическое_выражение` проверяется ***перед*** каждым выполнением цикла, поэтому цикл **while** выполняется *ноль или более раз*.

Цикл с постусловием **do-while** выполняется, пока значение проверяемого логического выражения является истиным, как и в цикле **while**:

```
do
   оператор
while (логическое_выражение);
```

Но, так как проверка логического выражения выполняется ***после*** каждого выполнения цикла, цикл **do-while** выполняется *один или более раз*.

Операторы break и continue

Исполнение оператора **break** в операторах цикла **while**, **do-while**, **for** приводит к немедленному выходу из оператора цикла. Выполнение программы продолжается с первого оператора, следующего за прерванным оператором.

```
int x;
for (x = 1; x <= 10; x++) {
  if (x == 5) {
    break;
```

```
  }
  printf("%d\n", x);
}
```

В данном примере на экран будут выведены следующие значения: 1, 2, 3, 4. Последующие значения не будут выведены, так как при x равном 5 выполнится оператор **break**, который прекратит выполнение цикла.

Исполнение оператора **continue** в операторах цикла **while**, **do-while**, **for** приводит к пропуску оставшихся операторов в теле цикла и выполнению очередной итерации цикла.

```
int x;
for (x = 1; x <= 10; x++) {
  if (x == 5) {
    continue;
  }
  printf("%d\n", x);
}
```

В данном примере на экран будут выведены следующие значения: 1, 2, 3, 4, 6, 7, 8, 9, 10. Значение 5 не будет выведено, так как при x равном 5 выполнится оператор **continue**, который прекратит выполнение текущей итерации цикла и заставит выполняться следующую итерацию.

Варианты заданий

Написать программу, используя операторы цикла.

1. Вводится целое положительно число n. Найти среднее арифметическое целых четных чисел с 1 до n. Вывести на экран эти числа, их количество, сумму и среднее арифметическое.

2. Вводится последовательность из 15 целых чисел. Определить, является ли данная последовательность подпоследовательностью последовательности чисел Фибоначчи. Если не является, указать номер элемента, нарушающего последовательность.

3. Найти сумму и количество целых чисел больших 20, но меньших 100 и кратных 3. Вывести на экран числа, удовлетворяющие условию, их количество и найденную сумму.

4. Вводится последовательность из 20 целых чисел. Определить, есть ли в последовательности хотя бы одно число, оканчивающееся цифрой 7. Если есть, вывести на экран порядковый номер последнего из них.

5. Вводится целое положительное число `n`. Найти и вывести на экран сумму квадратов, произведение и среднее арифметическое целых нечетных чисел с 10 до n, и сами эти числа.

6. Вводится последовательность из 20 вещественных чисел. Определить, есть ли в последовательности хотя бы одна пара одинаковых соседних чисел. Если есть, вывести порядковые номера чисел первой из таких пар.

7. Найти количество и произведение целых положительных чисел больших 40, меньших 100 и кратных 4. Вывести на экран числа, удовлетворяющие условию, их количество и произведение.

8. Вводится последовательность из 20 целых чисел. Определить, является ли последовательность упорядоченной по возрастанию. Если не является, вывести на экран порядковый номер первого числа, нарушающего последовательность.

9. Определить, существует ли натуральное четырехзначное число, сумма пятых степеней цифр которого равна самому числу.

10. Вводится целое положительное число `n`. В интервале от `1` до `n` найти количество и сумму чётных чисел. Вывести на экран эти числа, их количество и произведение.

11. Подсчитать количество целых чисел в интервале от `102` до `987`, в записи которых отсутствуют одинаковые цифры.

12. Вводится целое положительное число `n`. Найти произведение и среднее арифметическое целых четных чисел, кратных `5`,

в интервале с 1 до n. Вывести на экран эти числа, их количество и произведение.

13. Дано натуральное число n. Найти наибольшую цифру числа (например, в числе 901123, наибольшей цифрой является 9).

14. Вводятся два целых числа m и n (m < n). Найти количество отрицательных четных и положительных нечетных чисел от m до n включительно. Вывести на экран эти числа и их общее количество.

15. Дано натуральное число n. Найти и вывести на экран среднее арифметическое значение нечетных цифр числа.

16. Найти и вывести на экран сумму целых чисел, больших 20, меньших 100, кратных 3 и заканчивающихся на 2, 4 или 8, и сами эти числа.

17. Вводится натуральное число n и однозначное число k. Определить и вывести на экран номер первого вхождения цифры k в числе n, считая от младших разрядов числа. Если такая цифра отсутствует, вывести -1.

18. Найти и вывести на экран сумму вторых степеней нечетных чисел и сумму третьих степеней чётных чисел в интервале, заданном значениями целочисленных переменных m и n.

19. Вывести на экран все двузначные числа, сумма вторых степеней цифр которых делится на 13.

20. Вводится целое положительное число n. В интервале от 1 до n найти и вывести на экран среднее арифметическое целых четных чисел, произведение целых нечетных чисел, сумму квадратов чисел кратных 7.

Пример выполнения задания

С клавиатуры вводится n чисел (n задается пользователем); определить, какое число встречалось чаще, минимальное или

максимальное, или указать, что минимальное и максимальное число встречалось равное количество раз.

```
#define <stdio.h>

/* основная функция */
void main(void) {
  /* объявление целочисленной переменной n для хранения
     количества чисел, которые необходимо считать */
  int n;
  /* объявление переменных: current – для хранения
     текущего введенного числа; min – для хранения
     минимального числа; max – для хранения
     максимального числа; min_count - для хранения
     количества, сколько раз встречается минимальное
     число; max_count – для хранения количества,
     сколько раз встречается максимальное число */
  int current, min, max, min_count, max_count;

  /* запрос количества чисел */
  printf("Укажите, сколько чисел будет вводиться: ");
  /* запись введенного количества в n */
  scanf("%d", &n);

    /* запрос первого числа */
(1) printf("\nВведите число № 1: ");
    /* сохранение первого числа */
(2) scanf("%d", &current);
    /* инициализация min первым числом */
(3) min = current;
    /* инициализация max первым числом */
(4) max = current;
    /* инициализация min_count и max_count */
(5) min_count = 1;
(6) max_count = 1;

    /* счетный цикл по i от 2 до n с шагом 1 */
(7) for (int i = 2; i <= n; i++) {
      /* запрос очередного числа */
      printf("\nВведите число № %d: ", i);
      /* считывание очередного числа */
      scanf("%d", &current);
      /* если текущее число меньше текущего
         минимального числа */
```

```
    if (current < min) {
      /* сохранение нового минимума в min */
      min = current;
      /* счетчик количества вхождения минимального
         числа сбрасывается в 1 */
      min_count = 1;
      /* иначе если текущее число совпадает с текущим
         минимальным числом */
    } else if (current == min) {
      /* увеличивается счетчик вхождения минимального
         числа */
      min_count++;
    }
    /* если текущее число больше текущего максимального
       числа */
    if (current > max) {
      /* сохранение нового максимума в max */
      max = current;
      /* счетчик количества вхождения максимального
         числа сбрасывается в 1 */
      max_count = 1;
      /* иначе если текущее число совпадает с текущим
         максимальным числом */
    } else if (current == max) {
      /* увеличивается счетчик вхождения максимального
         числа */
      max_count++;
    }
  }

  /* если минимальное число встретилось большее
     количество раз, чем максимальное */
  if (min_count > max_count) {
    /* вывод информации */
    printf("\nМинимальное число %d встречалось чаще
              максимального числа %d.\n", min, max);
    /* иначе если минимальное число встретилось меньшее
       количество раз, чем максимальное */
  } else if (min_count < max_count) {
    /* вывод информации */
    printf("\nМинимальное число %d встречалось реже
максимального числа %d.\n", min, max);
  } else {
    /* вывод информации */
    printf("\nМинимальное число %d и максимальное
```

```
                число %d встречались равное количетсво
                раз.\n", min, max);
  }
}
```

Строки 1, 2, 3, 4, 5, 6: отдельно описывают все действия, производимые над всеми числами, для самого первого числа. Это необходимо для того, чтобы сделать предположение, что первое число является минимальным (или максимальным). Далее в цикле будут обрабатываться числа со второго по последнее. И, так как первое число было обработано отдельно, будет с чем сравнивать второе и последующие числа при поиске минимума и максимума.

Строка 7: описывается счётный цикл **`for`**. Счётчиком в данном цикле является переменная `i`. Перед началом выполнения цикла ей присваивается значение `2` – начальное значение счётчика. Условием выполнения цикла будет являться условие, что `i <= n`. После каждого выполнения тела цикла значение счётчика будет увеличиваться на единицу. Таким образом, цикл позволит обработать (считать и проанализировать) числа со второго по `n`-е, что и требуется сделать в программе.

8. Массивы

Массив – группа ячеек памяти, связанных в том смысле, что все они имеют одно имя и один тип.

Для обращения к конкретной ячейке памяти – элементу массива – указывается имя массива и номер позиции этого элемента в массиве: `c[1]` – элемент с номером `1` в массиве `c`.

Массив является статической структурой данных – его размер не изменяется в процессе выполнения программы.

Первый элемент в массиве в языке С имеет порядковый номер `0`.

Номер позиции элемента называется индексом элемента массива. Индекс – целое число или арифметическое выражение, значение которого является целым числом: `c[6]`, `c[t]`, `c[i * 2 + 1]`.

Элемент массива может стоять как в левой части оператора присваивания, так и в правой:

```
/* увеличение значения элемента 0 массива c на 5.5 */
c[0] += 5.5;
/* присваивание переменной x значения, равного половине
значения элемента 6 массива c */
x = c[6] / 2;
/* вывод суммы первых трех элементов массива */
printf("%d", c[0] + c[1] + c[2]);
```

Массивы занимают необходимое количество последовательных ячеек памяти. При объявлении устанавливается имя массива, тип элементов массива и число элементов в массиве:

```
тип идентификатор[количество_элементов];
тип идентификатор1[количество_элементов1],
идентификатор2[количество_элементов2], ...,
идентификаторN[количество_элементовN];
```

Здесь `тип` – тип данных, которые будет хранить объявленный массив; `идентификатор` – имя этого массива. Количество элементов массива может быть задано только в виде целочисленной константы (в том числе, описанной при помощи директивы `#define`, см. далее).

```
/* объявление массива c из 12 элементов (номера
элементов массива - с 0 по 11) */
int c[12];
// объявление массива b из 100 элементов (с 0 по 99) и
x из 27 элементов (с 0 по 26)
int b[100], x[27];
```

Сразу после объявления массива элементы не имеют конкретных значений.

Массив можно инициализировать, например, с помощью цикла `for`:

```
void main(void) {
  int n[10];
  for (int i = 0; i < 10; i++) {
    n[i] = 0;
  }
  ...
}
```

Такой цикл запишет значения `0` во все элементы массива (с элемента 0 по элемент 9).

Массив может быть инициализирован при объявлении:

```
тип идентификатор[колич_элем] = { список_значений };
```

В качестве списка значений могут выступать константы соответствующего типа, разделенные запятыми:

```
int n[5] = {32, 27, 64, 18, -7};
```

После такого объявления элементы массива n будут соответственно содержать значения `32`, `27`, `64`, `18`, `-7`.

Если инициализирующих значений меньше, чем элементов массива, элементы, значения которых не заданы, автоматически инициализируются нулями:

```
int n[5] = {0};
```

При таком объявлении в элемент массива № 0 запишется инициализирующее значение `0`, а элементы с №№ 1-4 автоматически заполнятся значениями `0`, так как для них не указаны инициализирующие значения.

Следующее объявление приведёт к ошибке, так как в нём инициализирующих значений больше, чем элементов в массиве:

```
int n[5] = {32, 27, 64, 18, -7, 0};
```

Если размер массива при объявлении не указан, число элементов в массиве будет равно числу инициализирующих значений.

Следующее объявление создаст массив `n`, содержащий 2 элемента, имеющих значения `32` и `27`:

```
int n[] = {32, 27};
```

Следующий массив будет содержать 4 элемента:

```
int n[] = {27, 5, 7, 1};
```

Для указания размера массива удобно использовать символические константы, определенные при помощи директивы препроцессора **`#define`**.

Директива **`#define`** описывается в области директив препроцессора и имеет следующий синтаксис (в описываемом случае):

```
#define идентификатор константа
```

Директива **`#define`** указывает компилятору, что нужно подставить последовательность символов в текст программы, определенную аргументом `константа`, вместо каждой последовательности символов, определенной аргументом `идентификатор`. Последовательность символов `идентификатор` заменяется на последовательность символов `константа` только в том случае, если она находится вне комментария, вне символьной константы и не является частью более длинного идентификатора (например, имени переменной, функции и пр.).

Следующая директива позволит заменить все вхождения последовательности символов `MAX_SIZE` в исходном тексте программы на последовательность символов `1024`:

```
#define MAX_SIZE 1024
```

Рассмотрим следующий пример:

```
#include <stdio.h>

/* директива позволит заменить все вхождения
последовательности символов SIZE на последовательность
символов 10 */
#define SIZE 10

void main() {
  /* объявление массива s, содержащего 10 элементов */
  int s[SIZE];
  /* конечным значением счетчика станет значение SIZE -
     1, то есть 9; в цикле массив заполняется чётными
     числами */
  for (int j = 0; j < SIZE; j++) {
    s[j] = 2  + 2 * j;
  }
  printf("%s%13s\n", "Элемент", "Значение");

  /* в цикле происходит вывод массива в виде таблицы */
  for (int j = 0; j < SIZE; j++) {
    printf("%7d%13d\n", j, s[j]);
  }
}
```

Для того, чтобы изменить программу так, чтобы она работала с массивами другого размера, например, `100`, необходимо исправить одну единственную строку программы:

```
#define SIZE 100
```

В языке С отсутствует проверка на выход за пределы массива, предотвращающая ссылку на несуществующий элемент. Весь контроль должен осуществлять программист. К примеру, при выполнении цикла над элементами массива его индекс никогда не должен быть меньше `0` и должен быть строго меньше общего количества элементов.

Пусть, например, объявлен массив `length` типа **`int`**, содержащий 10 элементов:

```
int length[10] = {0};
```

В этом случае следующая операция будет допустимой с точки зрения языка С, но произойдёт выход за пределы массива и результат выполнения операции будет непредсказуем:

```
length[11] = 1;
```

Варианты заданий

Написать программу для работы с массивом. Значения элементов массива должны вводиться с клавиатуры.

1. Получить и вывести на экран сумму логарифмов квадратов четных элементов массива `a[N]` и сумму нечетных элементов.

2. В массиве `a[N]` поменять местами элементы каждой пары чисел. Если количество элементов в массиве нечетное, последний элемент не должен участвовать в обмене. Вывести на экран получившийся массив.

3. Найти среднее арифметическое элементов массива `a[N]` и заменить отрицательные элементы массива суммой значения соответствующего элемента и среднего арифметического всех элементов массива. Вывести на экран получившийся массив.

4. Найти значение наибольшего отрицательного элемента массива `a[N]` и наименьшего положительного элемента.

5. Поменять местами элемент массива `a[N]` с минимальным значением и элемент, находящийся в середине массива. Вывести на экран получившийся массив.

6. Поменять местами в массиве `a[N]` первый элемент и последний отрицательный. Вывести на экран получившийся массив.

7. Определить и вывести на экран количество и сумму элементов массива `a[N]`, лежащих за пределами отрезка `[a, b]`, границы вводятся пользователем.

8. Переставить в массиве `a[N]` первый элемент и элемент с наибольшим значением. Вывести на экран получившийся массив.

9. Наименьший элемент массива a[N] заменить остатком от деления наибольшего элемента на этот наименьший элемент. Вывести на экран получившийся массив.

10. Вычислить и вывести на экран произведение всех не равных нулю элементов массива a[N], имеющих нечетные номера.

11. Вводится натуральное число m (m < N / 2). Переставить в массиве a[N] первые m и последние m элементов массива, сохраняя порядок их следования. Вывести на экран получившийся массив.

12. Если хотя бы один элемент массива a[N] меньше или равен -2, все отрицательные элементы массива заменить их квадратами, а положительные – значением их квадратного корня. В противном случае умножить все элементы массива на 2. Вывести на экран получившийся массив.

13. Выяснить и вывести на экран, положительные или отрицательные числа встречаются в массиве a[N] чаще, или их количество совпадает.

14. Изменить знак (с плюса на минус или с минуса на плюс) у элементов той половины массива a[N], в которой расположен элемент массива с наибольшим абсолютным значением. Вывести на экран получившийся массив.

15. Заменить максимальный и минимальный элементы массива a[N] на среднее арифметическое остальных элементов массива. Вывести на экран получившийся массив.

16. Определить, имеются ли в массиве a[N] три подряд стоящих элемента, упорядоченных по возрастанию. Вывести на экран значения и номера элементов первой такой группы.

17. Заменить все элементы массива a[N], кратные трем, на третий по счету положительный элемент массива. Вывести на экран получившийся массив.

18. Заменить последний положительный элемент массива `a[N]` на первый отрицательный элемент массива. Если положительных и (или) отрицательных элементов нет, оставить массив без изменения. Вывести на экран получившийся массив.

19. Определить, есть ли в массиве `a[N]` два соседних положительных элемента. Если есть, вывести на экран номера первой и последней такой пары.

20. В массиве `a[N]` заменить каждый элемент, кроме первого, суммой всех предыдущих элементов. Вывести на экран получившийся массив.

Пример выполнения задания

Все отрицательные элементы массива `a[N]` заменить их квадратами, положительные элементы заменить их квадратными корнями, найти среднее арифметическое элементов массива, поменять местами последний и наименьший элемент массива, вывести на экран новый массив и среднее арифметическое его элементов.

```
    #include <stdio.h>
    /* подключение библиотеки math.h для использования
       функции sqrt */
    #include <math.h>

(1) #define N 10

    /* основная функция */
    void main(void) {
      /* объявление вещественного массива a из N
         элементов (а данном случае - 10); переменной
         temp для обмена значениями двух элементов
         массива */
      double a[N], temp;
      /* объявление переменной min для хранения номера
         минимального элемента массива */
      int min = 0;
      /* объявление переменной sum для записи суммы
         элементов массива */
      double sum = 0;
```

```
      /* цикл по всем элементам массива */
(2)   for (int i = 0; i < N; i++) {
        /* запрос значения элемента № i + 1 */
        printf("\nВведите элемент массива № %d: ",
                  i + 1);
        /* запись введенного значения в элемент массива
           a с номером i */
(3)     scanf("%lf", &a[i]);
        /* если введённое значение больше 0 */
        if (a[i] > 0) {
          /* заменить его на значение квадратного
             корня */
          a[i] = sqrt(a[i]);
        }
        /* если введённое значение меньше 0 */
        if (a[i] < 0) {
          /* заменить его на его квадрат */
          a[i] = a[i] * a[i];
        }
        /* прибавить текущий элемент к общей сумме */
        sum += a[i];
      }

      /* цикл по всем элементам массива */
(4)   for (int i = 0; i < N; i++) {
        /* если текущий элемент меньше элемента с
           номером min */
(5)     if (a[i] < a[min]) {
          /* значит текущий элемент может быть
             минимальным; запомнить номер текущего
             элемента*/
          min = i;
        }
      }
      /* вычисление среднего арифметического элементов
         массива (в sum содержалась их сумма, N – их
         количество) */
      sum /= N;

      /* обмен значениями последнего элемента массива и
         элемента массива с минимальным значением */
(6)   temp = a[min];
(7)   a[min] = a[N - 1];
(8)   a[N - 1] = temp;
```

```
  printf("\nНовый массив: ");
  /* цикл по всем элементам массива */
  for (int i = 0; i < N; i++) {
    /* вывод значения элемента с номером i */
    printf("%.2lf ", a[i]);
  }
  /* вывод среднего арифметического */
  printf("\nСреднее арифметическое элементов
          массива: %l.2f\n", sum);
}
```

Строка 1: при помощи директивы препроцессора **#define** описывается символическая переменная `N`. При компиляции все вхождения константы `N` в программу будут заменены на её значение – `10`.

Строка 2: в параметрическом цикле значение параметра `i` изменяется последовательно от `0` до `N-1` (до `9`), что соответствует номерам элементов массива `a`. Таким образом, подставляя счётчик `i` в качестве номера элемента массива (`a[i]`), можно получить доступ последовательно к каждому элементу массива и сделать с ним необходимые действия.

Строка 3: правила считывания значений в элементы массива ничем не отличаются от правил считывания значений в обычные переменные: точно также необходимо указать адрес в памяти, по которому необходимо записать полученное значение: `&a[i]`.

Строка 4: как только массив заполнен нужными значениями, можно приступать к поиску минимального элемента массива. Для этого также удобно использовать параметрический цикл.

Строка 5: при поиске минимального (максимального) элемента массива сначала следует предположить, что минимальным (максимальным) элементом является первый элемент массива (в данном случае элемент с номером `0`; поэтому при объявлении переменная min инициализирована значением `0`). Далее необходимо просматривать последовательно все элементы массива; если встретится элемент с меньшим (большим) значением, он становится новым претендентом на «звание» минимального (максимального) элемента. После просмотра всех элементов массива

переменная `min` будет содержать номер минимального элемента массива.

Строки 6, 7, 8: для обмена значениями двух переменных можно использовать третью переменную – буфер для временного хранения значения одной из переменных: в данном конкретном случае в `temp` сохраняется значение `a[min]`, затем значение `a[min]` смело переписывается значением `a[N - 1]`, а из `temp` в `a[N - 1]` помещается значение `a[min]`. Провести обмен значениями у двух переменных можно и без использования третьей переменной по такой схеме: `b = a + b; a = b - a; b= b - a;`.

9. Двумерные массивы

Массивы в языке C/C++ могут иметь несколько измерений (несколько индексов) – это многомерные массивы (часто применяются для представления таблиц, например, матриц) – массивы массивов.

Для идентификации элемента указывают два индекса:

- идентифицирующий строку элемента;
- идентифицирующий столбец.

Если индексов 2 – массивы называются двумерными.

Многомерные массивы объявляются следующим образом:

```
тип идент[колич_элем][колич_элем]...;
```

Например, объявляется двумерный массив `a` типа **`int`**, состоящий из двух строк и двух столбцов:

```
int a[2][2];
```

Ниже объявляется трехмерный (3 на 4 на 3) массив `b` типа **`int`**:

```
int b[3][4][3];
```

При инициализации при объявлении многомерных массивов значения рекомендуется группировать в фигурных скобках (инициализировать каждый одномерный массив, входящий в многомерный массив, отдельным списком инициализации):

```
int a[2][2] = {{1, 2}, {3, 4}};
```

В следующем примере после инициализации в массиве `a[0]` будут содержаться значения `1`, `0`, а в массиве `a[1]` – значения `3`, `4`:

```
int a[2][2] = {{1}, {3, 4}};
```

Обрабатывать многомерные массивы удобно, например, при помощи вложенных циклов. Следующий фрагмент программы может быть использован для вывода содержимого некоторого двумерного массива `b[2][3]` на экран:

```
for (int i = 0; i < 2; i++) {
   for (int j = 0; j < 3; j++) {
      printf("%d ", b[i][j]);
   }
   printf("\n");
}
```

Здесь параметр `i` фиксирует строку массива, и при фиксированном значении `i` параметр `j` изменяет свое значение, «проходя» по всем элементам строки. При использовании массивов большей мерности следует использовать циклы большей вложенности.

Варианты заданий

Написать программу для работы с двумерным массивом (матрицей). Значения элементов матрицы допустимо задавать непосредственно в тексте программы.

1. Определить и вывести на экран, является ли заданная матрица бинарной – состоящей из 0 и 1.

2. Вычислить и вывести на экран сумму максимальных элементов из каждого столбца матрицы.

3. Определить и вывести на экран, является ли заданная матрица ганкелевой матрицей – такой, у которой на всех диагоналях, перпендикулярных главной, стоят равные элементы.

4. Вывести на экран количество положительных элементов каждого столбца матрицы.

5. Определить и вывести на экран, является ли заданная матрица антидиагональной – такой, что все элементы, лежащие вне побочной диагонали, равны 0.

6. Найти и вывести на экран сумму элементов той строки матрицы, в которой расположен минимальный элемент матрицы.

7. Определить и вывести на экран, является ли заданная матрица бисимметричной – симметричной как относительно главной диагонали, так и относительно побочной диагонали.

8. Найти и вывести на экран среднее арифметическое наименьшего и наибольшего значений элементов матрицы.

9. Определить и вывести на экран, является ли заданная матрица порядка `n` центросимметричной – симметричной относительно своего центра, то есть $a_{i,j} = a_{n-i+1,n-j+1}$.

10. Найти и вывести на экран номер столбца матрицы с максимальной суммой элементов. Если таких столбцов несколько, вывести номер правого из них.

11. Определить, является ли заданная матрица диагонально-доминирующей: такой, что $|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}|$ для всех значений i.

12. Определить и вывести на экран, имеются ли в матрице одинаковые элементы.

13. Определить и вывести на экран, является ли заданная матрица Z-матрицей – такой, что её элементы, не лежащие на главной диагонали, меньше или равны нулю.

14. Найти и вывести на экран индексы последнего четного элемента в каждой строке матрицы.

15. Определить, является ли заданная матрица нижнетреугольной – такой, что все элементы выше главной диагонали нулевые.

16. Каждый столбец матрицы заменить суммой столбцов, находящихся справа, включая сам столбец. Вывести получившуюся матрицу на экран.

17. Определить и вывести на экран, является ли заданная матрица трёхдиагональной – такой, что все её ненулевые элементы располагаются на трёх диагоналях: главной, первой сверху и первой снизу.

18. Каждую строку матрицы заменить суммой нижележащих строк, включая саму строку. Вывести получившуюся матрицу на экран.

19. Определить и вывести на экран, является ли заданная матрица матрицей Фробениуса – такой, которая получается из единичной при помощи сдвига её влево и добавления нового столбца справа.

20. Все элементы матрицы, имеющие максимальное значение, заменить на 0. Вывести получившуюся матрицу на экран.

Пример выполнения задания

Определить и вывести на экран, является ли заданная матрица (предполагается, что задана бинарная матрица) матрицей перестановки – такой бинарной матрицей, в каждой строке и столбце которой находится лишь один единичный элемент.

```
#include <stdio.h>

/* определение символической константы N со значением
   10 при помощи директивы #define */
#define N 10

/* основная функци */
void main(void) {
  /* объявление двумерного массива matrix размером N на
     N и инициализация его списком значений */
  int matrix[N][N] =
    {
      { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
      { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
      { 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },
      { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },
      { 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 },
      { 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
      { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
    };

  /* объявление переменной для хранения суммы элементов
     в строке */
  int rows;
  /* объявление переменной для хранения суммы элементов
```

```
    в столбце */
  int cols;
  /* объявление переменной для хранения признака того,
     что строка или столбец не являются содержащими
     нули на всех позициях, кроме одной */
  int flag;

  /* цикл по всем строкам (столбцам) */
  for (int i = 0; i < N; i++) {
    /* для каждой новой строки (столбца) выполняется
       обнуление сумм значений элементов строки
       (столбца) и сброс флага */
    rows = 0;
    cols = 0;
    flag = 0;
    /* цикл по всем элементам конкретной строки
       (столбца) */
    for (int j = 0; j < N; j++) {
      /* вычисление суммы элементов в строке i */
      rows += matrix[i][j];
      /* вычисление суммы элементов в столбце j */
      cols += matrix[j][i];
    }
    /* если сумма элементов в текущей строке не равна
       1 или сумма элементов в текущем столбце не
       равна 1 */
    if (rows != 1 || cols != 1) {
      /* установить значение флага в 1 */
      flag = 1;
      /* иначе выйти из цикла */
      break;
    }
  }

  /* если в ходе проверки флаг был установлен в 1 */
  if (flag == 1) {
    /* вывести сообщение о том, что исходная матрица -
       не матрица перестановок*/
    printf("Не является матрицей перестановок.\n");
  } else {
    /* иначе вывести о том, что исходная матрица -
       матрица перестановок*/
    printf("Является матрицей перестановок.\n");
  }
}
```

10. Пользовательские функции

Функции

Всякая программа на языке С представляет собой совокупность функций, выполняющих работу по реализации некоторого алгоритма.

Каждая из них есть независимый набор описаний и операторов.

Используя функции, исходную задачу можно представить в виде последовательности более простых задач, каждая из которых реализует некоторую часть общего алгоритма.

Для композиции программ используются функции следующих видов:

- функции стандартной библиотеки С (`sin`, `cos`, `printf`, `scanf` и пр.);

- функции, определяемые программистом – пользовательские функции.

Описание пользовательских функций имеет следующий синтаксис:

```
тип_возвращаемого_значения имя(список_параметров) {
  тело_функции
}
```

Здесь `имя` – любой допустимый идентификатор; `тип_возвращаемого_значения` – тип результата, который возвращает функция; `тело_функции` – объявления и операторы; `список_параметров` – список объявлений, получаемых функцией при вызове (формальные параметры функции).

Функция выполняет вычисления над входными данными – параметрами функции – и выдаёт результат определенного типа – возвращает значение.

Если функция не возвращает значения, используется тип **`void`**.

Например, функция `srand` из библиотеки `stdlib.h` задает начальную точку для формирования последовательности псевдослучайных целых чисел, но не возвращает значения – она имеет

тип **void**. Функция `sin` вычисляет и возвращает синус аргумента – она имеет тип **double**.

Кроме этого, тип **void*** используется, если необходимо вернуть указатель неопределенного типа.

Данные из вызывающей функции передаются в вызываемую при помощи параметров.

Формальные параметры – аргументы, описанные в заголовке функции и используемые при её определении.

Фактические параметры – конкретные значения, придаваемые формальным параметрам функции при её вызове.

Формальные параметры указываются при объявлении функции следующим образом:

```
тип1 имя1, тип2 имя2, тип3 имя3...
```

Функция, определяющая максимум из трёх чисел, может быть описана так:

```
int max(int a, int b, int c) {
   /* операторы, позволяющие определить макс. число */
}
```

Здесь `max` – имя функции, **`int`** – тип возвращаемого значения, а список **`int`** `a,` **`int`** `b,` **`int`** `c` показывает, что функция `max` принимает три целочисленных параметра.

Объявления и операторы внутри фигурных скобок образуют тело функции. В теле функции возможно объявление новых переменных. Формальные параметры используются как переменные, уже имеющие значения, переданные при вызове функции.

Выполнение инструкций в теле функции начинается с самого первого оператора и продолжается до тех пор, пока не встретится оператор возврата **`return`**, либо пока не будет достигнут конец внешнего блока.

Для выхода из функции, имеющей тип **`void`**, оператор **`return`** записывается следующим образом:

```
return;
```

Для выхода из функции с возвращением значения оператор **return** записывается так (значение выражения `выражение` должно иметь тот же тип, что и сама функция):

```
return выражение;
```

Ниже описана функция, принимающая три целых числа и возвращающая наибольшее из них:

```
int max(int a, int b, int c) {
  /* если a больше, чем b и больше, чем c */
  if (a > b && a > c) {
    /* вернуть значение a */
    /* функция прекратит выполнение после return a */
    return a;
  } else {
    /* если b больше, чем c */
    if (b > c) {
      /* вернуть значение b */
      /* функции прекратит выполнение после return b */
      return b;
    } else {
      /* вернуть c */
      /* функция прекратит выполнение после return c */
      return c;
    }
  }
}
```

Вызов функций

Вызов функции осуществляется следующим образом:

```
имя_функции(список_фактических_параметров)
```

Функции, возвращающие результат типа **void**, вызываются как отдельные операторы.

Функции, возвращающие значения других типов, могут быть использованы в выражениях:

```
int z;
...
z = 1 + max(2, a + z, z * 2);
```

При вызове функции `max` указанным способом формальный параметр `a` получает значение `2`, формальный параметр `b` получает значение выражения `a + z`, формальный параметр `c` получает значение выражения `z * 2`. Значение, возвращаемое функцией `max`, увеличивается на `1` и общий результат записывается в `z`.

При вызове функции фактические параметры записываются в область памяти, выделенную под формальные параметры функции. То есть функция работает с копией передаваемых значений – в С реализована передача параметров по значению. Никакое изменение значений формальных параметров внутри функции не приведет к изменению значений фактических параметров.

Для корректного вызова функций должно выполняться требование, чтобы функция была описана выше того места, откуда она вызывается:

```
/* описана функция foo */
int foo() {
   ...
}

/* функция, вызывающая foo, описана ниже функции foo */
void main(void) {
  int a = foo();
}
```

Следующее описание программы приведёт к ошибке, так как функция описана ниже того места, откуда она вызывается:

```
void main(void) {
  int a = foo();
}

int foo() {
  ...
}
```

Прототип функции

Прототип функции – предварительное её описание – сообщает компилятору тип данных, возвращаемых функцией, число

параметров, получаемых функцией, тип и порядок следования параметров.

Прототипом функции `max` является:

```
int max(int, int, int);
```

При использовании прототипа описание функций будет располагаться в программном коде, например, так:

```
/* прототип функции max */
int max(int, int, int);
...

/* функция main, вызывающая функцию max */
void main() {
  ...
  a = max(v1, v2, v3);
  ...
}

/* функция max */
int max(int a, int b, int c) {
  ...
}
```

Функция является независимой в семантическом отношении частью программы и может быть определена в произвольном месте исходного файла, в отдельном файле, либо находиться во внешних библиотеках.

Так как функция может быть вызвана раньше, чем она определена в текущем файле, предварительное её прототипирование позволяет избежать дальнейших ошибок.

Аргументы по умолчанию

В случае, если функция многократно вызывается с одним и тем же аргументом для некоторого параметра, возможно использовать *аргумент по умолчанию* для данного параметра – значение, которое по умолчанию должно передаваться параметру.

Аргументы по умолчанию должны быть самыми правыми аргументами в списке параметров функции.

Аргументы по умолчанию должны быть указаны при первом появлении имени функции (в прототипе или в заголовке функции, если прототип отсутствует).

Если пропущенный аргумент не является самым правым, то все аргументы правее него также пропускаются.

Значения по умолчанию могут быть любыми выражениями, в том числе константными, глобальными переменными или вызовами функций.

Прототип с аргументами по умолчанию должен иметь следующий вид:

```
тип имя(тип1 [имя1] [= значение 1] ... );
```

Например:

```
int drawCircle(double, int x = 0, int y = 0);
```

Другой пример:

```
int drawCircle(double r = 10.0, int = 0, int = 0);
```

Заголовок функции с аргументами по умолчанию описывается так:

```
тип имя(тип1 имя1 [= значение1] ... ) { ... }
```

Например:

```
int drawCircle(double r = 10.0, int x = 0, int y = 0)
{ ... }
```

Следующая программа вычисляет объем коробки, используя аргументы по умолчанию в случае, если часть аргументов опущена:

```
#include <stdio.h>

/* прототип функции с аргументами по умолчанию */
int box_volume(int length = 1, int width = 1, int
height = 1);

void main(void) {
  /* нет аргументов: используются все значения
     элементов по умолчанию */
```

```
  printf("Объем коробки: %d", box_volume());

  /* передана длина: ширина и высота – по умолчанию */
  printf("Объем коробки: %d", box_volume(10));

  /* передана длина и ширина: высота – по умолчанию */
  printf("Объем коробки: %d", box_volume(10, 5));

  /* передана длина, ширина и высота */
  printf("Объем коробки: %d", box_volume(10, 5, 2));;
}

/* значения по умолчанию в описании функции не указаны,
   т.к. они указаны при описании прототипа функции */
int box_volume(int length, int width, int height) {
      return length * width * height;
}
```

Варианты заданий

Написать функцию в соответствии с заданием и продемонстрировать её работу в программе.

1. Функция принимает натуральное число и возвращает произведение чисел, представленных цифрами этого числа.

2. Функция принимает два целых числа и возвращает остаток от деления первого числа на второе. Функция не должна использовать операцию встроенную операцию взятия по модулю `%`.

3. Функция принимает вещественное число и возвращает другое вещественное число – дробную часть исходного числа (с учетом его знака).

4. Функция принимает два целых числа и возвращает результат возведения одного числа в степень, равную второму числу. Функция не должна использовать функцию `pow` из библиотеки `math.h`.

5. Функция принимает целое число и возвращает число `2`, возведенное в степень, равную этому числу. Функция не должна использовать функцию `pow` из библиотеки `math.h`.

6. Функция принимает целое число и возвращает факториал этого числа.

7. Функция принимает целое число и возвращает его максимальный делитель (не являющийся самим этим числом).

8. Функция принимает два целых числа и возвращает наибольший общий делитель этих чисел.

9. Функция принимает два целых числа и возвращает наименьшее общее кратное этих чисел.

10. Функция принимает целое число и возвращает исходное число, цифры в котором поменяны местами, если это число двузначное, и `0` в остальных случаях.

11. Функция принимает целое число и возвращает исходное число, цифры в котором записаны в обратном порядке.

12. Функция принимает 8 вещественных чисел – координат выпуклого четырехугольника, и возвращает его площадь.

13. Функция принимает 3 целых числа – часы, минуты и секунды, и возвращает время в секундах.

14. Функция принимает шестизначное число и возвращает `1`, если данное число является счастливым (сумма первых трех его цифр равна сумме последних трех его цифр), и `0` в противном случае.

15. Функция принимает целое число и возвращает `1`, если число является простым числом Софи Жермен (такое простое число `p`, что число `2p + 1` также простое), и `0` в противном случае.

16. Функция принимает целое число и возвращает `1`, если число является избыточным (таким, что сумма всех его положительных делителей (отличных от самого числа) превышает само число).

17. Функция принимает целое число и возвращает `1`, если это число является автоморфным (десятичная запись квадрата которого оканчивается цифрами самого этого числа), и `0` в противном случае.

18. Функция принимает целое число и возвращает `1`, если это число является триморфным (десятичная запись куба которого оканчивается цифрами самого этого числа), и `0` в противном случае.

19. Функция принимает натуральное число и возвращает сумму чисел, представленных цифрами этого числа.

20. Функция принимает целое число и возвращает `1`, если это число является тау-числом (делящемся на число своих делителей), и `0` в противном случае.

Пример выполнения задания

Функция принимает натуральное число n и возвращает число трибоначчи с номером `n`.

```
#include <stdio.h>

    /* прототип функции tribonacci */
(1) int tribonacci(int);

  void main(void) {
    /* объявление переменной t для записи номера числа
       трибоначчи, которое необходимо получить */
    int t;
    /* запрос номера числа трибоначчи */
     printf("Введите номер желаемого числа трибоначчи:");
    /* запись введенного значения в t */
    scanf("%d", &t);
    /* вывод значения, которое возвращает функция
       tribonacci для фактического параметра t */
(2) printf("\nЧисло трибоначчи № %d: %d\n", t,
          tribonacci(t));
}

    /* Функция tribonacci                          */
    /* Назначение:                                 */
```

```
    /*  возвращает число трибоначчи с           */
    /*     заданным номером                     */
    /* Входные данные:                         */
    /*  n – номер необходимого числа трибоначчи */
    /* Выходные данные:                        */
    /*  отсутствуют                            */
    /* Возвращаемое значение:                  */
    /*  tn – число трибоначчи №n               */
(3) int tribonacci(int n) {
    /* установка значений первых чисел трибоначчи */
    int t0 = 0, t1 = 0, t2 = 1;
    /* если запрошено число №0 – вывести значение t0 */
(4) if (n == 0) return t0;
    /* если запрошено число №1 – вывести значение t1 */
(5) if (n == 1) return t1;
    /* если запрошено число №2 – вывести значение t2 */
(6) if (n == 2) return t2;

    /* числа с номером больше 2 необходимо вычислять */

    /* объявление переменной для хранения числа
       трибоначчи №n */
    int tn;
    /* цикл по числам трибоначчи от 3 до n */
(7) for (int i = 3; i <= n; i++) {
      /* текущее число трибоначчи есть сумма трёх
         предыдущих чисел трибоначчи */
      tn = t0 + t1 + t2;
      /* запись в t0, t1, t2 трёх последний вычисленных
         чисел трибоначчи для определения следующего
         числа */
      t0 = t1;
      t1 = t2;
      t2 = tn;
    }
    /* возвращение вычисленного числа трибоначчи №n */
(8) return tn;
  }
```

Строка 1: описания прототипа функции `tribonacci`. На основе прототипа компилятор определит, что функция с именем `tribonacci` будет принимать один аргумент типа **`int`** и возвращать значение типа **`int`**.

Строка 2: в качестве одного из параметров функции `printf` используется `tribonacci(t)`. При вызове `printf` вычислится значение функции `tribonacci` с фактическим параметром `t`, и результат работы этой функции станет фактическим параметром функции `printf` – значением, которое необходимо вывести на экран.

Строка 3: заголовок функции `tribonacci`. Если сравнивать заголовок функции с прототипом этой функции, можно увидеть, что в заголовке уточнены имена формальных параметров функции: указано, что единственный параметр типа **`int`** будет иметь имя `n`.

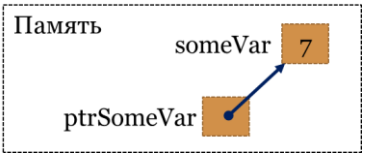
Строки 4, 5, 6, 7: в этих строках используется переменная `n`, которая ранее не была объявлена, и которой не присваивалось значение с помощью оператора присваивания. Но эта переменная является формальным параметром функции (описана в заголовке функции). Таким образом, она будет известна компилятору и будет иметь конкретное значение – значение, переданное функции `tribonacci` при вызове – значение переменной `t` из функции `main`.

Строка 8: при помощи оператора **`return`** функция возвращает вычисленное значение – значение переменной `tn`. Это значение будет подставлено в то место функции `main`, откуда была вызвана функция `tribonacci`, то есть это значение станет фактическим параметром функции `printf` из `main`.

11. Указатели

Указатель – переменная, значением которой является адрес в памяти. Переменная *непосредственно* ссылается на значение. Указатель *косвенно* ссылается на значение.

Ссылка на значение через указатель называется *косвенной адресацией.*



На рисунке переменная `someVar` непосредственно ссылается на значение `7` (содержит значение `7`), а переменная `ptrSomeVar` косвенно ссылается на значение `7` (содержит адрес переменной `someVar`).

Указатели объявляются следующим образом:

```
тип *идентификатор;
```

Например:

```
int *prtSomeVar, *a, someVar;
```

Здесь `ptrSomeVar` и `a` – указатели на целочисленное значение, `someVar` – обычная целочисленная переменная.

Перед использованием указатели должны быть инициализированы при помощи оператора присваивания либо значением `NULL`, либо адресом конкретной переменной.

```
int *prtVar = NULL;
```

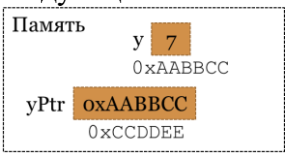
Указатель со значением `NULL` не указывает ни на что.

Получить адрес переменной можно при помощи оператора взятия адреса `&`.

```
/* объявляется переменная y типа int со значением 5 */
int y = 7;
/* объявляется переменная yPtr - указатель на целое
   число */
int *yPtr;
/* в yPtr записывается адрес переменной y */
```

```
yPtr = &y;
```

После выполнения данных операций в памяти компьютера появляется примерно следующее:



Предположим, что переменная `y` расположена по адресу `0xAABBCC`. При присвоении ей значения `7` в ячейку `0xAABBCC` записалось. Переменная `yPtr` может быть расположена, например, по адресу `0xCCDDEE`. При присвоении ей значения `&y` в ячейку `0xCCDDEE` записался адрес `y`, то есть `0xAABBCC`.

Оператор косвенной адресации (разыменования) `*` возвращает значение объекта, на который ссылается операнд (указатель), то есть позволяет выполнять чтение и запись значений по адресу, записанному в указателе.

Следующий фрагмент программы выводит на экран значение, на которое ссылается указатель `yPtr`.

```
printf("%d", *yPtr);
```

Изучите следующую программу, демонстрирующую работу с указателями:

```
#include <stdio.h>

void main(void) {
  /* объявление целочисленной переменной a и указателя
     на целое число aPtr */
  int a, *aPtr;
  /* присвоение переменной a значения 7 */
  a = 7;
  /* присвоение указателю aPtr адреса переменной a */
  aPtr = &a;
  /* вывод значения aPtr – адреса переменной a */
  printf("Значение указателя aPtr: %p", aPtr);
  /* вывод адреса переменной a */
  printf ("\nАдрес переменной a: %p", &a);
  /* вывод значения переменной a */
  printf ("\nЗначение переменной a: %d", a);
```

```
  /* вывод значения, на которое указывает aPtr */
  printf ("\nРазыменование aPtr: %d", *aPtr);
  /* изменения значения, на которое указывает aPtr */
  *aPtr = 5;
  /* вывод значения переменной a */
  printf ("\nЗначение переменной a: %d\n", a);
}
```

В результате работы программы на компьютере автора на экран было выведено следующее:

```
Значение указателя aPtr: 0086FB1C
Адрес переменной a: 0086FB1C
Значение переменной a: 7
Разыменование aPtr: 7
Значение переменной a: 5
```

Здесь следует обратить внимание на то, что при изменении значения по указателю `aPtr` изменилось значение переменной `a`, так как `aPtr` указывает на `a`.

С указателями можно выполнять следующие операции: `++`, `--`, `+`, `+=`, `-`, `-=`.

```
int v[10] = {0};
/* пусть vPtr содержит адрес элемента № 0 массива v */
int *vPtr = &v[0];
```

При прибавлении или вычитании из указателя целого числа значение его увеличивается или уменьшается на произведение числа на размер объекта, на который указатель ссылается.

Так, если, например, адресом `v[0]` является `3000`, то после прибавления к `vPtr` единицы в `vPtr` будет содержаться `3004` (данные типа **`int`** имеют размер 4 байта), то есть фактически указатель будет содержать адрес следующей ячейки памяти типа **`int`** – адрес элемента `v[1]`.

Имя массива есть указатель-константа на первый элемент массива (с номером 0). Пусть объявлен массив `b` и указатель `bPtr`:

```
int b[10] = {0}, *bPtr;
```

В этом случае оба следующих присвоения дадут один и тот же результат:

```
/* записывается адрес массива b – адрес его первого
   элемента */
bPtr = b;
/* записывается адрес первого элемента массива b – то
   есть адрес самого массива */
bPtr = &b[0];
```

Также одинаковый результат дадут следующие операторы (при условии, что `bPtr` содержит адрес `b`):

- адрес элемента массива `b` с номером `3`:

```
bPtr + 3
b + 3
```

- значение элемента массива `b` с номером `3`:

```
*(bPtr + 3)
*(b + 3)
```

Таким образом, запись `name[index]`, где `name` – имя массива, аналогична записи `*(name + index)`.

Указатели удобно использовать в случае, если необходимо внутри функции изменить значение передаваемого фактического параметра. Для этого функция должна получить адрес этого параметра – указатель на параметр – и работать со значением по переданному адресу (передача через указатель):

```
/* определение функции my_abs */
void my_abs(int *num) {
   if (*num < 0) {
      *num = -(*num);
   }
}

void main(void) {
  ...
  int a = -1;
  /* вызов функции myAbs */
  my_abs(&a);
  ...
}
```

Функция `myAbs` в качестве параметра принимает адрес целочисленной переменной и работает со значением по этому адресу, используя оператор разыменования `*`. Таким образом, если функция вызвана указанным способом, и ей передан адрес переменной `a`, то в результате работы функции изменится значение самой переменной `a`.

При работе с указателями могут быть допущены ошибки, которые могут создать угрозу безопасности и стабильной работе системы (обращение к «чужим» блокам памяти, выход за пределы массива) – программный код с указателями является небезопасным, непроверяемым. Но в языке С такой способ передачи параметров, чтобы значение изменялось и внутри функции, и вне функции, был единственным возможным.

В языке С++ появился новый способ передачи параметров «по ссылке». Описание функции, принимающей параметр по ссылке выглядит так:

```
/* определение функции my_abs */
void my_abs(int &num) {
   if (num < 0) {
      num = -(num);
   }
}
```

А вызов её – так:

```
void main(void) {
  ...
  int a = -1;
  /* вызов функции my_abs */
  my_abs(a);
  ...
}
```

Ссылку в С++ можно понимать или как альтернативное имя объекта, или как безопасный вариант указателей. Ссылки имеют три особенности, отличающие их от указателей:

- при объявлении ссылка обязательно инициализируется ссылкой на уже существующий объект данного типа. Ссылка (как и указатель) не может использоваться без инициализации;

- ссылка пожизненно указывает на один и тот же адрес;

- при обращении к ссылке операция разыменования производится автоматически.

При объявлении ссылок вместо звёздочки `*` следует писать амперсанд `&`.

В рассмотренном выше примере параметр `num` функции `myAbs` стал не указателем, а ссылкой. Поэтому теперь при вызове функции `myAbs` компилятор сам передаст адрес переменной a, при этом внутри функции `myAbs` обращение к переменной `num` ничем не будет отличаться от обращения к обычной переменной.

Так как имя массива является указателем на первый элемент массива, возможна передача массивов в качестве аргументов функции:

```
/* Функция summ                                */
/* Назначение:                                 */
/*  возвращает сумму элементов массива         */
/* Входные данные:                             */
/*  arr[] – адрес массива                      */
/*  size – количество элементов в массиве      */
/* Выходные данные:                            */
/*  отсутствуют                                */
/* Возвращаемое значение:                      */
/*  a – сумма элементов массива                */
int summ(int arr[], int size) {
   int a = 0;
   for (int i = 0; i < size; i++) {
      a += arr[i];
   }
   return a;
}

void main(void) {
  int cash[1000];
  ...
  int total = summ(cash, 1000);
}
```

Здесь описание формального параметра **`int`** `arr[]` аналогично описанию **`int`** `*arr`.

При описании формального параметра функции, являющегося многомерным массивом, первый индекс массива не указывается, а все остальные необходимы. Так как все элементы многомерного массива расположены в памяти последовательно, указание значений второго и следующих индексов дает возможность сообщить компилятору о способе отыскания нужных элементов в памяти. Для определения местонахождения элемента в конкретной строке компилятор должен знать, сколько в точности элементов находится в каждой строке, чтобы при обращении к конкретному элементу он мог пропустить соответствующее число блоков памяти.

Пример описания функции, принимающей двумерный массив:

```
void print_array(int a[][3], int size) {
  for (int i = 0; i < size; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
}
```

12. Символьные массивы

В языке C/C++ строка типа `"hello"` является массивом отдельных символов типа **`char`**.

Символьный массив может быть инициализирован строковым литералом:

```
char string1[] = "АБВГ";
```

Объявленный массив `string1` содержит 4 символа строки (`'А'`, `'Б'`, `'В'`, `'Г'`) и 1 нулевой символ – символ `'\0'` с кодом `0` (фактически массив содержит 5 элементов), так как строки в языке C являются нуль-терминированными.

Нуль-терминированная строка – строка, заканчивающаяся символом с нулевым кодом, то есть либо нулевым байтом в случае однобайтового представления символов, либо двумя нулевыми байтами в случае двухбайтового представления. Недостаток нуль-терминированных строк состоит в том, что для вычисления длины строки необходимо последовательно просмотреть все её элементы, начиная с первого, пока не будет найден нулевой байт. Такая операция может быть долгой для длинной строки.

Следующее объявление массива `string1` аналогично предыдущему:

```
char string1[] = {'А', 'Б', 'В', 'Г', '\0'};
```

При объявлении размер символьного массива следует задавать достаточно большим для хранения строки возможного размера и нуль-символа.

Так как строка – символьный массив, возможно прямое обращение к символу строки по номеру.

```
/* объявление и инициализация символьного массива s */
/* массив s будет содержать 15 символов: 14 символов
   самой строки и 1 нулевой символ */
char s[] = "мама мыла раму";
/* вывод на экран символа № 0; будет выведено 'м' без
   апострофов */
printf("%c", s[0]); // вывод М
/* вывод на экран кода символа № 0; будет выведено
   число 236, т.к. код символа 'м' в кодировке ASCII –
```

```
   236*/
printf("%d", s[0]);
/* замена содержимого элемента № 0 содержимым элемента
   № 10 */
s[0] = s[10];
/* запись в элемент массива № 10 символа 'м' */
s[10] = 'м';
/* вывод на экран содержимого строки s; на экран будете
   выведено "рама мыла маму" без кавычек */
printf("%s", s);
```

Несмотря на то, что и `printf("%c", s[0])` и `printf("%d", s[0])` выводят на экран `s[0]`, результат вывода будет различным. В первом случае, так как указан спецификатор `%c`, аргумент `s[0]` интерпретируется как символ, то есть на экран выводится изображение, соответствующее числу, хранящемуся в `s[0]` – коду символа. Во втором случает этот же `s[0]` интерпретируется как целое число, так как указан спецификатор `%d`, и на экран выводится само число, являющееся кодом символа `'м'`.

Ввод строки в массив возможен с клавиатуры:

```
char string2[1024];
scanf("%s", string2);
```

Обратите внимание, что перед `string2` отсутствует операция взятия адреса `&`. Это связано с тем, что значением переменной-массива (имеется в виду, например, в данном случае значением `string2`) является адрес этого массива. Адреса и указатели обсуждаются в разделе «Указатели».

Для работы с символьными массивами, как и для работы с любыми другими массивами, удобно использовать параметрический цикл. При этом цикл должен завершать работу в том случае, если текущий рассматриваемый символ является нулём (конец строки). Следующий фрагмент программы показывает, как можно использовать счетный цикл при работе с символьными массивами; цикл позволяет вывести строку посимвольно с пробелами после каждого символа.

```
for (int i = 0; string1[i] != 0; i++) {
  printf("%c ", string1[i]);
}
```

Варианты заданий

Написать функцию, реализующую операцию со строками, не используя специализированные функции для работы со строками библиотеки C.

1) **int** strcat_s(**char** *strDestination, **int** numberOfElements, **const char** *strSource) – добавляет strSource к strDestination, а затем к результирующей строке завершающий нуль-символ. Начальный символ strSource перезаписывает конечный нуль-символ strDestination. Возвращает 0, в случае успеха, в противном случае – число, отличное от 0.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/d45bbxx4.aspx

2) **char** *strchr(**const char** *string, **int** sym) – находит первое вхождение sym в str и возвращает указатель на это вхождение или возвращает NULL, если sym не найден. Нулевой конечный символ включен в поиск.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/b34ccac3.aspx

3) **int** strcmp(**const char** *string1, **const char** *string2) – сравнивает string1 и string2 лексикографически и возвращает значение, которое указывает их отношение: -1, если string1 меньше чем string2; 0, если string1 идентична string2; 1, если string1 больше чем string2.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/e0z9k731.aspx

4) **int** _stricmp(**const char** *string1, **const char** *string2) – лексикографически сравнивает версии string1 и string2 в нижнем регистре и возвращает значение, показывающее их взаимосвязь: -1, если string1 меньше чем string2; 0, если string1 идентична string2; 1, если string1 больше чем string2.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/k59z8dwe.aspx

5) **int** strcpy_s(**char** *strDestination, **int** numberOfElements, **const char** *strSource) – копирует содержимое в адресе strSource, включая конечный нуль-символ, в расположение, указанное strDestination. Строка назначения должна быть достаточно велика для хранения строки источника и его конечного нуль-символа. Возвращает 0 в случае успешного выполнения; в противном случае – отличное от 0 значение.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/td1esda9.aspx

6) **char** *ucwords(**char** *string) – преобразует в верхний регистр первый символ каждого слова в строке. Возвращает строку, где первая буква каждого слова записана в верхнем регистре. Под словом данная функция понимает последовательность одной и более букв, которая располагается в начале строки, либо которой предшествует не буква и не цифра.

7) **int** strrpos(**const char** *string, **const char** * str) – возвращает позицию последнего вхождения str в string. Если str не входит в string, возвращает -1.

8) **char** *strstr(**const char** *str, **const char** *strSearch) – возвращает указатель на первое вхождение strSearch в str. Поиск не распространяется на завершающий нулевой символ. Возвращает NULL, если strSearch не появляется в str. Если strSearch указывает на строку нулевой длины, то функция возвращает str.

Подробная информация о функции:
http://msdn.microsoft.com/ru-ru/library/z9da80kz.aspx

9) **char** *strtrim(**char** *string) – удаляет пробелы, символы табуляции и перевода строки в начале и в конце строки. Возвращает указатель на строку, в которой отсутствуют пробелы, символы табуляции и перевода строки в начале и в конце строки.

10) **int** strewth(**const** **char** *str, **int** numberOfElements, **const** **char** *endStr, **int** length) – определяет, заканчивается ли строка str размером numberOfElements строкой endStr размером length. Возвращает 1 в случае, если str заканчивается endStr, 0 – в противном случае.

11) **char** *_strspnp(**const** **char** *str, **const** **char** *charset) – возвращает указатель на первый символ в заданной строке str, не входящий в другую заданную строку charset. Если такого символа нет, возвращает NULL.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/k254awe8.aspx

12) **int** strcspn(**const** **char** *str, **const** **char** *strCharSet) – возвращает индекс первого вхождения символа в строку str, принадлежащего набору символов strCharSet. Если в str нет символов из strCharSet, то возвращается длина str.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/xe8sk0x7.aspx

13) **int** strncat_s(**char** *strDest, **int** numberOfElements, **const** **char** *strSource, **int** count) – пытается добавить первые D символов строки strSource в конец строки strDest, где D – меньшее из величины count и длины strSource. Если эти добавляемые D символов поместятся внутри strDest (чей размер задается как numberOfElements), и по-прежнему останется место для завершающего нуль-символа, тогда эти символы добавляются, начиная с исходного нуль-символа strDest, и добавляется новый завершающий нуль-символ; в противном случае strDest[0] становится равным нуль-символу, и возвращается отличное от 0 значение. В случае успеха возвращает 0.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/w6w3kbaf.aspx

14) **int** _strnset_s(**char** *str, **int** numberOfElements, **int** c, **int** count) – задаёт, по крайней мере, первые count символов str в c. Если count больше, чем размер str, размер str используется вместо count. В случае успеха возвращает 0. Ошибка возникает, если count больше numberOfElements, и оба этих параметра больше размера str, тогда возвращает значение, отличное от нуля.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/ms175783.aspx

15) **char** *strpbrk(**const** **char** *str, **const** **char** *strCharSet) – возвращает указатель на первое вхождение символа, принадлежащего набору символов strCharSet, в str. Поиск не распространяется на завершающий нулевой символ. Возвращает указатель на первое вхождение любого символа из strCharSet в str или указатель на NULL, если строковые аргументы не имеют общих символов.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/217yyhy9.aspx

16) **char** *strrchr(**const** **char** *str, **int** c) – находит последнее вхождение c (преобразованного к char) в str. Поиск включает конечный нуль-символ. Возвращает указатель на последнее вхождение c в str или NULL, если c не найден.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/ftw0heb9.aspx

17) **char** *_strrev(**char** *str) – обращает порядок символов в str. Завершающий нулевой символ остается на месте. Возвращает указатель на измененную строку.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/9hby7w40.aspx

18) **int** strspn(**const char** *str, **const char** *strCharSet) – возвращает индекс первого символа в str, который не входит в набор символов strCharSet. Поиск не распространяется на завершающий нулевой символ. Если str начинается с символа, не содержащегося в strCharSet, функция возвращает 0.

Подробная информация о функции:
https://msdn.microsoft.com/ru-ru/library/kthzzay9.aspx

19) **int** strins_s(**char** *str, **int** numberOfElements, **const char*** subStr, **int** length, **int** position) – вставляет подстроку subStr длиной length в строку str длиной numberOfElements, начиная с позиции position. В случае, если вставка прошла успешно, возвращает 0, иначе – 1.

20) **char** *strprc_s(**const char** *str, **int** numberOfElements, **const char** *strExt, **int** length) – определяет, является ли какая-либо конечная часть строки str длиной numberOfElements начальной частью строки strExt длиной length. Возвращает указатель на символ, начиная с которого окончание строки str является началом строки strExt, и NULL, если окончание строки str не является началом строки strExt.

Пример выполнения задания

int *strdup_s(**char** *strSource, **int** buffer) – добавляет строку strSource, размещенную в блоке памяти размером buffer к самой строке strSource, начиная с нуль-символа исходной строки strSource, если длина результирующей строки после добавления не превзойдет buffer; если размер результирующей строки превосходит buffer, исходная строка не изменяется. Возвращает 0 в случае успеха и значение, отличное от нуля, в противном случае.

```
#include <stdio.h>

/* прототип функции strdup */
int strdup_s(char*, int);
```

```
/* основная функция */
void main(void)
{
  /* переменная для считывания строки размером 50 */
  char a[50];
  /* приглашение ввести строку */
  printf("Введите строку, которую хотите
            дублировать: ");
  /* считывание строки, введенной пользователем, в a */
  scanf("%s", a, 50);
  /* вызов функции strdup_s и запись результата работы
     функции в success; здесь a – строка, которую
     необходимо дублировать, 50 – объем памяти для
     строки */
  int success = strdup_s(a, 50);
  /* вывод дублированной строки на экран */
  printf("Результат работы функции: %d\nДублированная
            строка: %s\n", success, a);
}

/* Функция strdup_s                          */
/* Дублирует входную строку                  */
/* Входные данные:                           */
/*  strSource - исходная строка              */
/*  buffer - размер блока памяти,            */
     в котором размещается strSource         */
/* Выходные данные:                          */
/*  strSource - результирующая строка        */
/* Возвращаемое значение:                    */
/*  0 - дублирование прошло успешно          */
/*  не 0 - дублирование не было выполнено    */
int strdup_s(char *strSource, int buffer) {
  /* счетчик для вычисления длины исходной строки */
  int i;
  /* вычисление длины исходной строки*/
  /* пока очередной символ - не нуль-символ,
     перемещаемся к следующему символу*/
  /* после выполнения цикла переменная i содержит номер
     позиции нуль-символа в исходной строке*/
  for (i = 0; strSource[i] != '\0'; i++);
  /* проверка, поместиться ли дублированная строка в
     доступную память */
  /* если не помещается (удвоенная длина строки
     плюс нуль-символ больше размера буфера) */
```

```
  if (i * 2 + 1 > buffer) {
    /* возвращается количество символов, которые не
       помещаются в память для строки */
    return i * 2 + 1 - buffer;
  }

  /* далее программа продолжится, если удвоенная строка
     помещается в буфер */
  /* объявление счетчика для дублирования символов */
  int j;
  /* Дублирование символов */
  for (j = 0; j < i; j++) {
    /* в i + j позицию новой строки записывается j-й
       символ из исходной строки */
    strSource[j + i] = strSource[j];
  }

  /* в позицию j + i (как раз позиция после последнего
     символа дублированной строки) вставляется нуль-
     символ – признак окончания строки */
  strSource[j + i] = '\0';
  /* возврат 0 - признака успешного завершения
     дублирования строки */
  return 0;
}
```

13. Структуры

Структуры – наборы логически связанных переменных, объединенных под одним именем (могут состоять из переменных различных типов данных).

Структура – производный тип данных – создается из элементов других типов.

Синтаксис описания структур следующий:

```
struct имя_структуры {
  тип1 имя_элемента1;
  тип2 имя_элемента2;
  ...
  типn имя_элементаn;
};
```

Такое определение структуры не резервирует место в памяти компьютера, а описывает новый тип данных.

Имя `имя_структуры` совместно с ключевым словом **`struct`** используются для объявления переменных типа структуры:

```
struct имя_структуры переменная1, переменная2, ...;
```

Пример описания структуры:

```
struct person {
    unsigned short age;
    char name[255];
    char sex;
};
```

Пример объявления переменных:

```
struct person student, people[52], *man;
```

Также переменные могут быть объявлены непосредственно при описании структуры:

```
struct {
    unsigned short age;
    char name[255];
    char sex;
} student, people[52], *man;
```

В этом примере видно, что указание имени структуры не является обязательным. Если определение структуры не содержит имени, переменные типа этой структуры могут быть объявлены только при определении структуры.

Спецификатор **typedef** позволяет создать псевдонимы (более короткие и значимые имена) как для определяемых пользователем типов данных, так и для уже существующих в языке.

Использование спецификатора **typedef**:

```
typedef определение_типа синоним
```

В отличие от объявления **struct**, объявления **typedef** не вводят новый тип – они вводят новые имена для уже существующих типов.

Создание синонима для типа **struct** person:

```
typedef struct person {
    unsigned short age;
    char name[255];
    char sex;
} Person;
```

После такого описания переменные типа **struct** person могут быть объявлены так:

```
Person student, people[52], *man;
```

К переменными типа структур могут быть применены следующие операции:

1) присваивание переменных структур переменным того же типа: `student = people[4];`;

2) взятие адреса структуры: `&student`;

3) применение операции **sizeof** для определения размера структуры: **sizeof**`(student)`;

4) обращение к элементам структуры.

Способ обращения к элементам структуры зависит от того, непосредственно или косвенно (через указатель, по адресу) переменная типа структуры ссылается на значение.

Пусть объявлено две переменных (обратите внимание на то, что переменная `student` инициализирована при объявлении путём указания значений полей в таком порядке, в котором эти поля указаны при описании структуры):

```
Person student = {18, "Пётр", 'м'};
Person *woman;
```

Обращение к элементам структуры, представленной переменной соответствующего типа, осуществляется при помощи операции-точки (`.`):

```
/* запись в поле age переменной student значения 5 */
student.age = 5;
/* запись в переменную d значения поля age переменной
   student */
d = student.age;
```

Обращение к элементам структуры через указатель на структуру осуществляется при помощи операции-стрелки (`->`):

```
woman->age = 7;
d = woman->age;
```

Если указатель на структуру будет разыменован, то обращение также осуществляется через операцию-точку:

```
(*woman).age = 7; d = (*woman).age;
```

Аналогично, если использовать операцию взятия адреса переменной структуры, то необходимо использовать операцию-стрелку:

```
(&student)->age = 100;
```

Структуры могут быть переданы функциям следующими способами:

1) посредством передачи отдельного элемента структуры:

```
foo(structInstance.someParam);
```

2) посредством передачи всей структуры:

```
bar(structInstance);
```

3) посредством передачи указателя на структуру (передача адреса структуры):

```
baz(&structInstance);
```

В представленной ниже программе описана структура. Элементы массива данных типа этой структуры передаются в функции для записи в них значений и для вывода этих значений на экран:

```
#include <stdio.h>

/* описание структуры _book для представления сведений
   о книге с одновременным описанием при помощи
   ключевого слова typedef синонима book типа структуры
   _book */
typedef struct _book {
  /* первое поле структуры – имя автора */
  char author[255];
  /* второе поле структуры – заголовок книги */
  char heading[255];
  /* третье поле структуры – заголовок книги */
  int quantity;
} book;

/* Функция add_book                                    */
/* Назначение:                                         */
/*  заполняет поля структуры значениями пользователя */
/* Входные данные:                                     */
/* item – указатель на переменную типа book           */
/* Выходные данные:                                    */
/*  переменная item с заполненными полями              */
/* Возвращаемое значение:                              */
/*  отсутствует                                        */
void add_book(book *item) {
  /* вывод на экран запроса имени автора книги */
  printf("Enter author's name: ");
  /* запись имени по адресу соответствующего поля */
  scanf("\n%s", &(item->author), 255);
  /* вывод на экран запроса заголовка книги */
  printf("Enter heading: ");
  /* запись заголовка по соответствующему адресу */
  scanf("\n%s", &(item->heading), 255);
  /* вывод на экран запроса количества экземпляров */
```

```
  printf("Enter quantity: ");
  /* запись количества экземпляров по соответствующему
     адресу */
  scanf("\n%d", &(item->quantity));
}

/* Функция print_catalog                                  */
/* Назначение:                                            */
/*  выводит на экран содержимое массива типа book         */
/* Входные данные:                                        */
/*  items – массив типа book                              */
/*  n – количество элементов в массиве                    */
/* Выходные данные:                                       */
/*  отсутствуют                                           */
/* Возвращаемое значение:                                 */
/*  отсутствует                                           */
void print_catalog(book *items, int n) {
  /* цикл по всем элементам массива books */
  for (int i = 0; i < n; i++) {
    /* вывод значений соответствующих полей */
    printf("Author: %s; Heading: %s; Quantity: %d\n",
      items[i].author, items[i].heading,
      items[i].quantity);
  }
}

/* Основная функция */
void main(void)
{
  /* объявление массива из 10 элементов типа book */
  book books[10];
  /* добавление в массиве первой книги; передается
     адрес структуры */
  add_book(&books[0]);
  /* добавление в массив второй книги; передается
     адрес структуры */
  add_book(&books[1]);
  /* вывод содержимого массива на экран */
  print_catalog(books, 2);
}
```

Варианты заданий

Написать программу для работы с указанной структурой, аналогичную программе из данного раздела.

1. Структура Книга (название, автор, издательство, год издания, количество страниц).

2. Структура Студент (фамилия, имя, отчество, возраст, группа).

3. Структура Учебная дисциплина (наименование, семестр, в котором изучается дисциплина, количество часов лекционных занятий, количество часов практических занятий, количество часов лабораторных занятий).

4. Структура Паспорт (серия, номер, код подразделения, дата выдачи, кем выдан).

5. Структура Смартфон (производитель, марка, модель, операционная система, версия операционной системы).

6. Структура Город (наименование, регион, страна, год основания, количество жителей).

7. Структура Планета (название, масса, период обращения вокруг солнца, температура поверхности, количество спутников).

8. Структура Автомобиль (марка, модель, количество лошадиных сил, объем двигателя, экологический класс).

9. Структура Насекомое (название, вид, количество крыльев, количество глаз, количество ног).

10. Структура Лекарство (название, фармакологическая группа, действующее вещество, производитель, дозировка).

11. Структура Авиарейс (номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест).

12. Структура Страна (название, площадь, язык, валюта, столица).

13. Структура Жильё (цена, площадь, количество комнат, адрес, риелтор).

14. Структура Кошка (порода, окрас, возраст, пол, фамилия владельца).

15. Структура Деталь (наименование, цвет, масса, материал, количество на складе).

16. Структура Озеро (название, глубина, площадь, наличие промысловой рыбы, тип по происхождению – тектонические, пойменные, горные, кратерные и пр.).

17. Структура Погода (температура, влажность, скорость ветра, осадки).

18. Структура Компьютер (производитель, наименование процессора, частота процессора, объем оперативной памяти, объем жесткого диска).

19. Структура Принтер (марка, модель, технология печати, возможность цветной печати, скорость печати).

20. Структура Носитель информации (производитель, тип, объём памяти, скорость чтения, скорость записи).

14. Поразрядные операции

Поразрядные (побитовые) операции используются для операций над битами целочисленных операндов (**`char`**, **`short`**, **`int`**, **`long`**) (обычно беззнаковых):

Операция (сокращенный оператор присваивания)	**Описание**	**Пример**
`&` (`&=`) поразрядное И	Бит результата устанавливается в 1, если соответствующие биты операндов равны 1.	7_{10} & 3_{10} = 3_{10} (111_2 & 011_2 = 11_2)
`|` (`|=`)- поразрядное включающее ИЛИ	Бит результата в 1, если хотя бы один из соответствующих битов операндов – 1.	8_{10} | 14_{10} = 14_{10} (1000_2 | 1110_2 = 1110_2)
`^` (`^=`) – поразрядное исключающее ИЛИ	Бит результата в 1, если только один из соответствующих битов операндов – 1.	10_{10} ^ 7_{10} = 13_{10} (1010_2 ^ 0111_2 = 1101_2)
`<<` (`<<=`) – сдвиг влево	Сдвигает биты первого операнда влево на число бит, задаваемых вторым операндом.	3_{10} << 1_{10} = 6_{10} (11_2 << 1_{10} = 110_2)
`>>` (`>>=`) – сдвиг вправо	Сдвигает биты первого операнда вправо на число бит, задаваемых вторым операндом.	7_{10} >> 2_{10} = 2_{10} (111_2 >> 2_{10} = 001_2)
`~` (`~=`) – дополнение	Все биты, равные 0, устанавливаются в 1, а равные 1 – в 0.	~7_{10} (1 байт) = 248_{10} (~00000111_2 = 11111000_2)

В качестве примера рассмотрим программу, выводящую двоичное представление числа:

```
void display_bits(unsigned short);

/* основная функция */
void main(void) {
  unsigned short x = 15;
  display_bits(x);
}

/* Функция display_bits                                    */
/* Назначение:                                             */
```

```
/*  выводит двоичное представление числа           */
/* Входные данные:                                 */
/*  value – число, двоичное представление которого */
/*          необходимо вывести                     */
/* Выходные данные:                                */
/*  отсутствуют                                    */
/* Возвращаемое значение:                          */
/*  отсутствует                                    */
void display_bits(unsigned short value) {
      unsigned short c;
(1)   unsigned short displayMask = 1 << 15;

      printf("%d = ", value);
      for (c = 1; c <= 16; c++) {
(2)     putchar(value & displayMask ? '1' : '0');
(3)     value <<= 1;
        /* зрительно разделим байты пробелами */
        if (c == 8) {
          putchar(' ');
        }
      }
      putchar('\n');
   }
```

Строка 1: двоичное представление 1_{10} типа **short** – это $00000000\ 00000001_2$; если сдвинуть 1_{10} на 15 бит влево, получится $10000000\ 00000000_2$.

Строка 2: переменная `value` занимает 2 байта в памяти, и `displayMask` (значение `10000000 00000000`) занимает 2 байта; так как у `displayMask` только в старшем бите установлена 1_2, то если выполнять "побитовое И", в зависимости от значения старшего бита `value` результатом будет либо 0_2, либо 1_2.

Строка 3: сдвинув `value` на 1 бит влево, получим в старшем бите следующий бит переменной `value`.

Поразрядные операции имеют следующие приоритеты (приоритет уменьшается построчно сверху вниз; операции, записанные в одной строке имеют равные приоритет; поразрядные операции имеют полужирное начертание):

Операции	**Ассоциативность**
`() [] . ->`	Слева направо
`+ - ++ -- ! (type) & * ~ sizeof`	Справа налево

* / %	Слева направо
+ -	Слева направо
<< >>	Слева направо
< <= > >=	Слева направо
== !=	Слева направо
&	Слева направо
^	Слева направо
|	Слева направо
&&	Слева направо
||	Слева направо
?:	Справа налево
= += -= *= /= %= **&= |= <<= >>=**	Справа налево
,	Слева направо

Операция поразрядного «ИЛИ» может использоваться, например, для установки определенных битов числа в единицы.

Пусть для программирования светофора, который может гореть одним из трех цветов, причём светофор может либо мигать, либо гореть непрерывно, используются 4 младших бита числа. Три младших бита устанавливают цвет, которым горит светофор, четвертый бит определяет, мигает ли светофор. Если описаны следующие константы:

```
#define RED 1    /* в двоичной системе 00000001 */
#define YELLOW 2 /* в двоичной системе 00000010 */
#define GREEN 4  /* в двоичной системе 00000100 */
#define BLINK 8  /* в двоичной системе 00001000 */
```

и есть функция **void** set(**short**), включающая светофор в соответствующем режиме, анализируя биты аргумента, то для того, чтобы включить красный, необходимо написать:

```
/* младший бит будет установлен в единицу */
set(RED);
```

а для того, чтобы включить мигающий красный:

```
/* в единицу будет установлен младший бит и бит,
   отвечающий за мигание */
```

```
set(RED | BLINK);
```

В свою очередь функция `set` может использовать операцию поразрядного «И» для проверки, какие биты установлены:

```
void set(short mode) {
  /* результатом проверки будет «истина» только тогда,
     когда бит переменной mode, соответствующий
     красному цвету, будет установлен в 1*/
  if (mode & RED) {
    /* зажечь красный */
    red();
  /* результатом проверки будет «истина» только тогда,
     когда бит переменной mode, соответствующий
     желтому цвету, будет установлен в 1*/
  } else if (mode & YELLOW) {
    /* зажечь жёлтый */
    yellow();
  } else {
    /* иначе зажечь зелёный */
    green();
  }
  /* если бит, отвечающий за мигание установлен в 1 */
  if (mode & BLINK) {
    /* мигать */
    blink();
  }
}
```

Задание: написать программу, использующую поразрядные операции.

Варианты заданий

1. Программа определяет количество `1` в двоичной записи числа.

2. Программа определяет количество `0` в двоичной записи числа.

3. Программа выводит результат сложения двух целых однобайтных чисел в обратном коде.

4. Программа выводит результат сложения двух целых однобайтных чисел в дополнительном коде.

5. Программа выводит биты, которые вышли бы за пределы восьмиразрядной сетки при сложении двух однобайтных положительных целых чисел.

6. Программа определяет, содержит ли двоичная запись одного числа двоичную запись другого числа (например, запись числа `701` – `1010111101` содержит запись числа `7` – `111`).

7. Программа определяет, является ли двоичная запись одного числа реверсивной записью другого числа.

8. Программа определяет, верно ли что при прибавлении `1` к данному числу увеличится количество значащих двоичных цифр в записи числа.

9. Программа определяет, каких двоичных цифр больше в двоичной записи числа – `0` или `1`.

10. Программа определяет, является ли двоичная запись числа симметричной (например, `10111000011101` является симметричной, `11101` – не является).

11. Программа определяет, представляет ли двоичная запись непрерывную последовательность `1` и непрерывную последовательность `0` (например, `11100000` представляет, `1101000` – не представляет).

12. Программа определяет, содержат ли двоичные записи двух чисел равное количество `0` и `1`.

13. Программа определяет, двоичная запись какого из двух чисел состоит из большего количества цифр.

14. Программа определяет, верно ли, что в двоичной записи числа на трёх заданных позициях стоят единицы (например, данное утверждении верно для числа `1011011011` при заданных позициях `0`, `1`, `4`).

15. Программа определяет, верно ли, что двоичная запись числа содержит больше `1`, чем `0`.

16. Программа реализует поразрядную операцию «И», не используя поразрядную операцию «И» языка С.

17. Программа реализует поразрядную операцию включающего «ИЛИ», не используя поразрядную операцию включающего «ИЛИ» языка С.

18. Программа реализует поразрядную операцию исключающего «ИЛИ», не используя поразрядную операцию исключающего «ИЛИ» языка С.

19. Программа реализует операцию «НЕ», не используя поразрядную операцию «НЕ» языка С.

20. Программа определяет значение числа после замены всех `0` на `1` в его двоичной записи; указать, сколько замен произведено.

Пример выполнения задания

Программа определяет, возможно ли из двоичных цифр одного числа составить другое число в двоичной системе.

```
#include <stdio.h>

/* основная функция */
void main(void) {
  /* объявление переменных: source – исходное число;
     tsource – переменная для модификации числа source;
     dest – число, которое предполагается составить из
     цифр числа source; tdest – переменная для
     модификации числа dest */
  unsigned short source, tsource, dest, tdest;
  printf("Введите исходное число: ");
  scanf("%d", &source);
  tsource = source;
  printf("Введите желаемое число: ");
  scanf("%d", &dest);

  /* в tdest записывается значение dest, так как далее
```

```
     необходимо изменять это значение, и при этом
     исходное значение не должно пропасть */
  tdest = dest;

  /* объявление переменных: s0 – информация о
     количестве нулей в записи чисел; s1 – информация о
     количестве единиц в записи чисел; current –
     текущая рассматриваемая двоичная цифра */
  short s0 = 0, s1 = 0, current = 0;

  /* объявление маски – того, с чем будет сравниваться
     цифра в младшем разряде чисел source и dest */
  unsigned short mask = 1;

  /* в цикле просматриваются все 16 бит чисел tsource и
     tdest */
  for (int i = 0; i < 16; i++) {
    /* получение младшего бита tsource */
(1) current = tsource & mask;
    /* если младший бит tsource равен 0 */
    if (current == 0) {
      /* увеличивается счетчик нулей */
      s0++;
    } else {
      /* иначе увеличивается счетчик единиц */
      s1++;
    }
    /* получение младшего бита tdest */
(2) current = tdest & mask;
    /* если младший бит tdest равен 0 */
    if (current == 0) {
      /* уменьшается счетчик нулей */
      s0--;
    } else {
      /* иначе уменьшается счетчик единиц */
      s1--;
    }
    /* сдвиг tsource на 1 бит вправо */
(3) tsource >>= 1;
    /* сдвиг tdest на 1 бит вправо */
(4) tdest >>= 1;
  }

  /* если счетчик «0» и «1» одновременно равны 0 */
  if (s0 == 0 && s1 == 0) {
```

```
    /* вывод, что число dest может быть составлено из
       цифр source*/
    printf("\nЧисло %d может быть составлено из
              двоичных цифр числа %d\n", dest, source);
  } else {
    /* иначе делается вывод, что число dest не может
       быть составлено из цифр source*/
    printf("\nЧисло %d не может быть составлено из
              двоичных цифр числа %d\n", dest, source);
  }
}
```

Идея алгоритма состоит в том, чтобы посчитать количество нулей и количество единиц в двоичном представлении двух чисел, и, если количество нулей в исходном числе совпадет с количеством нулей в желаемом числе и количество единиц в обоих числах также совпадает, то можно сделать вывод, что из цифр одного числа можно составить другое число.

Для этого в программе просматриваются побитово оба числа. Если в исходном числе встречается `0`, увеличивается счетчик нулей, если `1` – единиц. Если в желаемом числе встречается `0`, счетчик нулей уменьшается, если `1`, уменьшается счетчик единиц. Таким образом, если в двух числах одинаковое количество нулей и единиц, после просмотра этих чисел счётчики нулей и единиц будут содержать значение `0`.

Строки 1, 2: определяется младший бит числа. Для этого используется переменная `mask` со значением `1` (двоичное представление – `00000000 00000001`). Если делать побитовую операцию «И» для переменной `mask` и какого-либо числа, результатом будет младший разряд этого числа (например, `01111100 1001001`**`1`** `&` `00000000 0000000`**`1`** `=` **`1`**, или `00000100 0111010`**`0`** `&` `00000000 0000000`**`1`** `=` **`0`**).

Строки 3, 4: выполняется побитовый сдвиг вправо. В результате этой операции на месте младшего бита числа оказывается второй по старшинству бит числа (например, в результате сдвига числа на 1 вправо `01101101 0001101`**`1`** получится число `00110110 1000110`**`1`**). Таким способом получится просмотреть все биты числа.

15. Перегрузка функций

Перегрузка функций – возможность в C++ определить несколько функций с одним и тем же именем, если эти функции имеют различные наборы параметров (различие типа или числа параметров, либо порядка следования их типов).

Когда вызывается перегруженная функция, компилятор выбирает нужную путём анализа числа, типов и порядка аргументов.

Перегрузка функций используется для создания нескольких функций, выполняющих сходные задачи, но над данными разного вида.

В качестве примера рассмотрим программу, использующую функцию `pow3` для возведения в `3` степень целых, вещественных чисел или символьного массива (под возведением в третью степень символьного массива здесь будем понимать «утроение» строки: из строки «`aab`» должна получиться строка «`aabaabaab`»).

```
#include <stdio.h>

/* прототипы трех функций, возводящих данные в куб */
int pow3(int); // возводит в куб целое число
double pow3(double); // возводит в куб вещественное
char* pow3(const char*, int); // возводит в куб строку

/* основная функция */
void main(void) {
  /* вызывается функция int pow3(int) */
  printf("%d\n", pow3(3));
  /* вызывается функция double pow3(double) */
  printf("%lf\n", pow3(1.5));
  /* вызывается функция char* pow3(const char*, int) */
  printf("%s\n", pow3("abc", 3));
}

/* реализация функции int pow3(int) */
int pow3(int a) {
  return a * a * a;
}

/* реализация функции double pow3(double) */
double pow3(double a) {
```

```
  return a * a * a;
}

/* реализация функции char* pow3(const char*, int) */
char* pow3(const char* a, int size) {
  char result[1024];
  char *s = result;
  int i = 0, j = 0;
  while (i < size) {
    for (j = 0; a[j] != 0; j++) {
      *s = a[j];
      s++;
    }
    i++;
  }
  *s = 0;
  return result;
}
```

Варианты заданий

Написать функцию в соответствии с заданием и продемонстрировать её работу.

1. Функция вычисления площади фигуры. Перегрузить функцию для следующих фигур: квадрат, прямоугольник, трапеция. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен вещественным числом.

2. Функция вычисления площади фигуры. Перегрузить функцию для следующих фигур: квадрат, прямоугольник, трапеция. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

3. Функция вычисления объема тела. Перегрузить функцию для следующих тел: куб, пирамида, усеченная пирамида. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен вещественным числом.

4. Функция вычисления объема тела. Перегрузить функцию для следующих тел: куб, пирамида, усеченная пирамида. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

5. Функция вычисления площади поверхности тела. Перегрузить функцию для следующих тел: куб, цилиндр, усеченный конус. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен вещественным числом.

6. Функция вычисления площади поверхности тела. Перегрузить функцию для следующих тел: куб, цилиндр, усеченный конус. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

7. Функция вычисления периметра фигуры. Перегрузить функцию для следующих фигур: квадрат, прямоугольник, треугольник. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен целым числом.

8. Функция вычисления периметра фигуры. Перегрузить функцию для следующих фигур: квадрат, прямоугольник, треугольник. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

9. Функция для определения максимального числа. Перегрузить функцию для определения максимума из двух, трех, четырех чисел. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен целым числом.

10. Функция для определения максимального числа. Перегрузить функцию для определения максимума из двух, трех, четырех чисел. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

11. Функция для определения минимального числа. Перегрузить функцию для определения минимума из двух, трех, четырех чисел. Исходные данные должны быть представлены целыми числами. Результат должен быть представлен целым числом.

12. Функция для определения минимального числа. Перегрузить функцию для определения минимума из двух, трех, четырех чисел. Исходные данные должны быть представлены вещественными числами. Результат должен быть представлен вещественным числом.

13. Функция для вывода значений переменных на экран. Перегрузить функцию для вывода целого значения, вещественного значения, символьного массива.

14. Функция, вычисляющая общее сопротивление параллельно соединенных резисторов. Перегрузить функцию для случая соединения двух, трех, четырех резисторов.

15. Функция, вычисляющая общую индуктивность параллельно соединенных катушек индуктивности. Перегрузить функцию для случая соединения двух, трех, четырех катушек индуктивности.

16. Функция, возвращающая значение максимального элемента в целочисленном массиве. Перегрузить функцию для одномерного, двумерного и трехмерного массива.

17. Функция, возвращающая значение максимального элемента в массиве вещественных чисел. Перегрузить функцию для одномерного, двумерного и трехмерного массива.

18. Функция, возвращающая наибольший общий делитель. Перегрузить функцию для определения наибольшего общего делителя двух, трёх чисел.

19. Функция, возвращающая наименьшее общее кратное. Перегрузить функцию для определения наименьшего общего кратного двух, трёх чисел.

20. Функция, подсчитывающая количество неотрицательных элементов в целочисленных массивах. Перегрузить функцию для подсчета неотрицательных элементов в одном массиве, в двух массивах, в трех массивах.

16. Перегрузка операций

В C++ возможно перегрузить операции для пользовательских типов данных.

Пусть `@` – оператор языка C++, кроме `. , .* :: ?: sizeof`, тогда функция вида

```
тип operator@(список_аргументов) { ... }
```

будет выполнять необходимые действия над данными, имеющими тип, указанный в списке аргументов.

Перегрузку операций проиллюстрирует следующая программа (в программе перегружается сложение двух комплексных чисел и комплексного числа с вещественным числом).

```
#include <stdio.h>

/* определение типа complex – структуры, содержащей два
   целочисленных поля: r – действительная часть числа и
   i – мнимая часть числа */
typedef struct _complexn {
  int r, i;
} complexn;

/* Перегрузка оператора сложения для двух комплексных
чисел - операндов типа complexn */
complexn operator+(complexn a, complexn b) {
  /* объявление новой переменной для возвращения
     результата */
  complexn result;

  /* сложение действительных частей */
  result.r = a.r + b.r;
  /* сложение мниых частей */
  result.i = a.i + b.i;
  /* возврат результата сложения */
  return result;
}

/* Перегрузка оператора сложения для комплексного числа
   (операнда типа complex) и натурального числа */
complexn operator+(complexn a, int b) {
  /* объявление новой переменной для возвращения
     результата */
```

```
  complexn result;

  /* сложение действительных частей */
  result.r = a.r + b;
  /* мнимая часть числа не изменяется */
  result.i = a.i;

 /* возврат результата сложения */
  return result;
}

void main(void) {
  /* объявление трех комплексных чисел */
  complexn x = {1, 1}, y = {2, 2}, z;

  /* сложение двух комплексных чисел */
  z = x + y;
  printf("%d + i%d\n", z.r, z.i);

  /* сложение комплексного и натурального числа */
  z = x + 3;
  printf("%d + i%d\n", z.r, z.i);
}
```

Варианты заданий

Перегрузить указанную операцию для указанных типов данных в соответствии с заданием и продемонстрировать её работу.

1. Операция сложения для обыкновенных дробей.
2. Операция вычитания для обыкновенных дробей.
3. Операция умножения для обыкновенных дробей.
4. Операция деления для обыкновенных дробей.
5. Операция сложения для комплексных чисел.
6. Операция вычитания комплексных чисел.
7. Операция умножения для комплексных чисел.
8. Операция деления для комплексных чисел.

9. Операция отношения == для обыкновенных дробей.

10. Операция отношения != для обыкновенных дробей.

11. Операция отношения >= для обыкновенных дробей.

12. Операция отношения <= для обыкновенных дробей.

13. Операция сложения векторов в двумерном пространстве.

14. Операция сложения векторов в трехмерном пространстве.

15. Операция вычитания векторов в двумерном пространстве.

16. Операция вычитания векторов в трехмерном пространстве.

17. Операция скалярного произведения векторов в двумерном пространстве.

18. Операция скалярного произведения векторов в трехмерном пространстве.

19. Операция векторного произведения векторов.

20. Операция умножения вектора на число в трехмерном пространстве.

Для заметок

Учебное издание

Зубаиров Александр Фларитович

ПРОГРАММИРОВАНИЕ

Тираж экз. Заказ .

Редакционно-издательский отдел ОТИ НИЯУ МИФИ. 456783, Челябинская обл. ,г. Озерск, пр. Победы, д. 48, www.oti.ru