

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Н. В. Груздева, Ю. С. Корухова

Семинары по курсу «Алгоритмы и алгоритмические языки, язык Си»

(учебное пособие)

Москва

2026

Введение

В учебном пособии представлены материалы основной части семинарских занятий в поддержку лекционного курса «Алгоритмы и алгоритмические языки» для студентов 1 курса факультета ВМК МГУ. Курс состоит из трех разделов:

- элементы общей теории алгоритмов;
- язык программирования Си [1, 2], как пример алгоритмического языка;
- структуры данных и классические алгоритмы.

В дополнении к учебным пособиям [1, 2], рассматриваются основные темы семинарских занятий, связанные со второй и третьей частями курса. При рассмотрении тем кратко формулируются основные понятия и расставляются акценты. Приводятся решения типовых задач, которые авторы рекомендуют рассмотреть на семинарских занятиях, а некоторые из них могут быть предложены студентам в качестве домашних заданий.

Часть занятий проводится на компьютерах. Задачи, выполнение которых рекомендуется на компьютере выделены в списке задач подчеркнутым шрифтом.

В рамках курса предусмотрено проведение нескольких контрольных работ и проверка домашних заданий. Выполнение практических и домашних заданий предполагает составление алгоритма, его запись на языке программирования, приобретение навыков тестирования и отладки программ. В рамках семинаров рассматривается работа в системах семейства UNIX. Курс не ориентирован на изучение одной конкретной среды программирования: работа с программой, ее компиляция, запуск, отладка предполагаются непосредственно из командной строки.

Тема 1.

Первая программа на языке C.

Константы, переменные, типы данных, выражения (начало).

Первая программа

Код программы (содержимое файла `first.c`):

1	<code>#include <stdio.h></code>
2	<code>int main()</code>
3	<code>{ int x = 10, y = 3;</code>
4	<code>int t;</code>
5	<code>t = x / y;</code>
6	<code>printf("Result = %d и %lf, t = %d\n", 12/5, 12.0/5, t);</code>
7	<code>return 0;</code>
8	<code>}</code>

Прокомментируем, что же тут написано.

В первой строке директива подключения библиотеки, содержащей функции ввода и вывода. Со второй строки начинается описание функции с именем `main` – с этой функции начинается выполнение программы. `int` – тип результата функции, `main` – имя

функции, у неё не задаются аргументы – поэтому после имени пустые скобки. Тело функции – содержимое фигурных скобок (строки 3-8).

Строка 3 – описание двух переменных целого типа (`int`) с инициализацией начальными значениями – переменная с именем «`x`» и начальным значением 10, переменная с именем «`y`» и начальным значением 3. Одиночный знак равенства – операция присваивания.

Строка 4 – описание целочисленной переменной `t` без инициализации. Под эту переменную отводится память, но начального значения в этот участок памяти не записывается, там какой-то мусор – то, что осталось от прежних вычислений, в которых участвовал этот участок памяти.

Строка 5 – переменной `t` присваивается результат целочисленного деления `x` на `y`. Почему деление целочисленное? Потому, что его операнды – переменные `x` и `y` – имеют целочисленный тип. Если хотя бы один из операндов вещественный, то и деление тогда будет делением вещественных чисел.

Строка 6 – вывод на экран строки, содержащей три значения. Первый аргумент `printf` называется форматной строкой. Это записанная в двойных кавычках строка, внутри которой встречаются `%d` и `%lf`. В выводимой строке `%d` заменяется на целое число, `%lf` заменяется на вещественное число. Самый левый `%d` заменится на значение выражения, следующего за форматной строкой (второй аргумент, результат целочисленного деления `12/5`). `%lf` заменится на результат вещественного деления (третий аргумент, результат вещественного деления `12.0/5`). Правый `%d` заменится на значение целочисленной переменной `t`. Символ `'\n'` означают переход на новую строку.

Строка 7 – возврат в операционную систему значения ноль после завершения работы программы. Возвращаемый ноль означает, что вычисления прошли нормально, ошибок не возникло. В случае возникновения ошибок в процессе вычисления программы рекомендуется значение, отличное от нуля.

Исходный файл с программой на языке C должен иметь расширение «`.c`». Назовём файл с нашей первой программой `first.c` и сохраним в текущей директории. Чтобы программа могла исполниться, нужно её откомпилировать. Для этого в командной строке нужно набрать:

```
gcc first.c -ansi -o first
```

Обращение к компилятору языка C `gcc` для компиляции файла `first.c` согласно стандарту ANSI (флаг `-ansi`), имя выходного исполняемого файла `first` (флаг `-o` с именем `first` после него). Если опустить `-o first`, то имя выходного исполняемого файла будет `a.out`. После успешной компиляции программы можно запустить исполняемый файл. В Unix-системах для этого в командной строке нужно набрать имя исполняемого файла, возможно, с указанием пути к нему.

```
./first
```

Здесь `.` обозначает текущий каталог, в нем будет взят файл `first` (без расширения).

Типы данных

При описании программ необходимо уделить внимание двум основным аспектам – тому, как описываются действия (операторы), и тому, как представляются объекты (данные), с которыми работают эти операции. Мы начнем с изучения способов представления данных языка Си.

В программировании под типом данных понимается множество объектов, в чем-то близких друг к другу (например, чисел или символов), плюс набор операций, который можно к ним применять.

Типы используются при описании объектов языка. При описании переменных эта информация (тип переменной) нужна компилятору для того, чтобы отвести необходимый

для хранения значения этой переменной участок памяти (для каждого типа известно, сколько байт памяти нужно отвести для хранения значения этого типа). В описании функции описываются типы её аргументов и тип возвращаемого объекта. В языке С имя типа записывается перед именем объекта этого типа (перед именем переменной или именем функции). В базовый набор типов языка Си включены следующие простые типы: char (символы), int (целые), float (вещественные), double (вещественные двойной точности?).

К названиям некоторых типов могут быть добавлены квалификаторы - слова short или long (они влияют на размер типа – количество отводимых для хранения значений байт: short – короткие - меньше байт, или long – длинные – больше байт). Пара других слов-квалификаторов – это слова signed и unsigned. Они не влияют на размер типа, но влияют на то, как хранящиеся в памяти последовательности 0 и 1, кодирующие значение этого типа, будут восприниматься компьютером – как числа со знаком (signed) или числа без знака (unsigned).

Все возможные сочетания в названии типов и комментарии к ним приведены ниже.

char	символы, под хранение значений отводится 1 байт
signed char	знаковые байты, диапазон значений от -128 до 127
unsigned char	беззнаковые байты, диапазон значений от 0 до 255
signed <u>short</u> int	короткие знаковые целые
<u>unsigned short</u> int	короткие беззнаковые целые
signed <u>int</u>	знаковые целые
<u>unsigned</u> int	беззнаковые целые
signed <u>long</u> int	длинные знаковые целые
<u>unsigned long</u> int	длинные беззнаковые целые
signed <u>long long</u> int	очень длинные знаковые целые (C99)
<u>unsigned long long</u> int	очень длинные беззнаковые целые (C99)
float	вещественные
double	вещественные двойной точности
long double	вещественные расширенной точности (ещё больше байт для представления числа)

В названиях типов, состоящих из нескольких слов, часть слов названия может быть опущена. В названиях типов подчёркнутым выделена та часть, которая должна быть обязательно указана для типа, остальные слова названия типа могут быть опущены. Например, можно заметить, что по умолчанию все целые считаются знаковыми (signed может быть опущено в названии целочисленного типа).

Можно узнать размер типа с помощью функции sizeof(тип). Ее результатом является количество байт, отводимых под хранение значения указанного в аргументе типа. Размер типа зависит от реализации, но для любой реализации выполняются все следующие неравенства:

```
sizeof(char) < sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
sizeof(char) = 1
sizeof(short) < sizeof(long)
sizeof(int) < sizeof(long long)
```

Рассмотрим объекты основных типов данных.

К типу **char** относятся

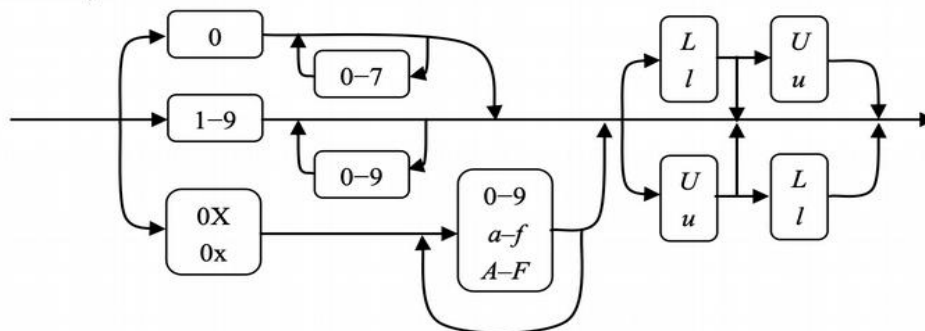
- символы, заключенные в одинарные кавычки (например, 'a', '4', '*'),
- специальные символы, такие как переход на следующую строку '\n', табуляция '\t'.

Каждому символу сопоставлен код — целое число, соответствие между символами и кодами зафиксировано в так называемой таблице кодировки, и именно этот код и хранится вместо символа в памяти машины. Он преобразуется в графическое

представление символа при выполнении операций вывода. Кроме задания самого символа в кавычках можно явно задать его код в десятичной, восьмеричной или шестнадцатеричной системе счисления, например, символ '*' может быть представлен как '\42', либо '\052', либо '\x2A').

Объектами **целого типа** являются целые числа. Введем определение целого числа, используя синтаксическую диаграмму

$\langle \text{целая константа} \rangle ::=$



Целочисленные константы могут быть записаны в восьмеричной системе счисления, тогда их запись начинается 0, или в шестнадцатеричной, начинающейся с 0x. Так константы 31, 037 и 0x1F представляют собой запись одного и того же числа. Число 0 в языке C является восьмеричным.

Объектами **вещественного типа** являются вещественные (действительные числа). В общем случае в записи вещественного числа указываются три вещи – целая и дробная части числа, (мантисса), а также порядок¹. Например, в записи вещественного числа 314.159E-2

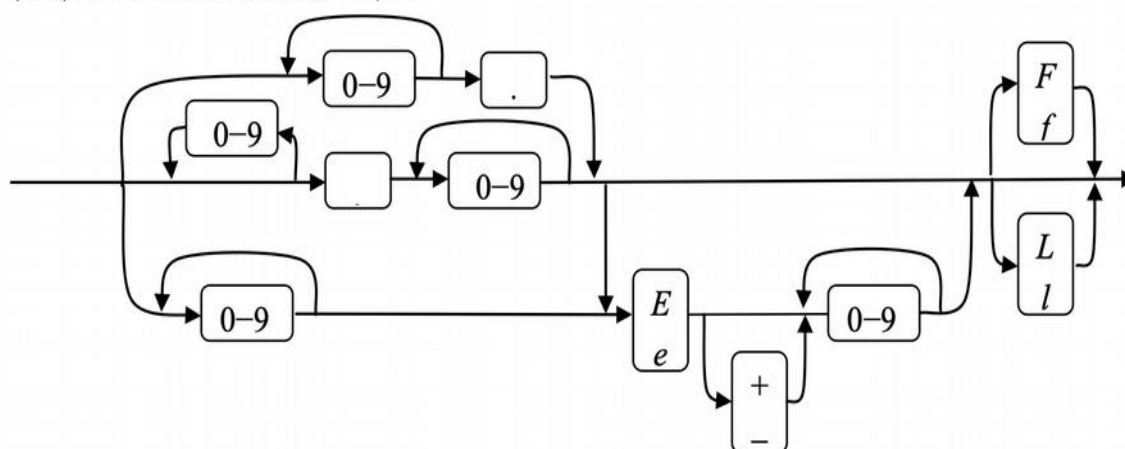
314 – целая часть,

159 – дробная часть,

E-2 –порядок.

При этом в записи вещественного числа необходимо присутствие либо точки, либо символа E (либо e). Опишем допустимые формы задания вещественных чисел в виде синтаксической диаграммы.

$\langle \text{вещественная константа} \rangle ::=$



В конце записи вещественного числа может быть указан суффикс F (либо f), указывающий

¹ О представлении вещественных чисел в памяти компьютера можно подробнее посмотреть в описании стандарта IEEE 754

тип константы - float, либо суффикс L (либо l), для указания типа long double. Если суффикс не указан, константа представляет собой число типа double.

Арифметические и логические операции

Обозначение арифметических операций такое же, как в большинстве языков:

Унарные плюс и минус + и -

Бинарные плюс, минус и умножение + - *

Деление обозначается как /. Если оба операнда целые, то это операция целочисленного деления (деления нацело). Если хотя бы один из операндов – вещественное число, то это операция вещественного деления. Типы операндов можно определить по типу операнда-переменной или по записи операнда-числа. Чтобы сказать, что число является вещественным, можно в его записи использовать точку. Например, 5 – это целое число, а 5.0 – это вещественное число, поэтому

11 / 5 будет равно 2 (частное для целочисленного деления)

11.0 / 5 будет равно 2.2 (частное для вещественного деления)

Операция нахождения остатка от целочисленного деления обозначается % (11 % 5 будет равно 1 – остаток целочисленного деления 11 на 5).

Значение выражения $1/2 + 1/3 + 1/4 + \dots + 1/1024$ будет равно нулю, поскольку в этом выражении используется операция целочисленного деления (результат деления единицы на большее число равен нулю для целочисленного деления, сумма нулей – нуль).

Для логических операций используются следующие обозначения:

Сравнения: > < >= <=

сравнение на равенство == ,

сравнение на неравенство !=

Логическое отрицание !

Логическая конъюнкция &&

Логическая дизъюнкция ||

В качестве лжи в языке C выступает целочисленное значение 0. Любое отличное от него целочисленное значение играет роль истины, но если истина подучилась в результате вычисления, то она представлена именно единицей.

Задача.

Выполняется ли в Си закон двойного отрицания: $!!x == x$?

Ответ:

Нет. Например, двойное отрицание значения 3 не равно этому значению (отрицание 3 – это 0, отрицание нуля – это 1, поэтому $!!3 \neq 3$)

Операции конъюнкции и дизъюнкции вычисляются слева направо, причем реализована так называемая короткая логика (ленивые вычисления): если по значению первого (левого) операнда уже можно вычислить значение операции, то второй и последующие операнды не вычисляются вовсе. Поэтому логический закон « $x \&\& y$ тождественно равно $y \&\& x$ » в языке Си не выполняется.

Рассмотрим пример:

$x != 0 \&\& y == b / x$

Если при вычислении этого выражения значение переменной x не равно нулю, то будет вычисляться второй операнд, в котором есть деление на x. Если же значение переменной x равно нулю, то второй операнд вычисляться не будет (значение первого операнда – ложь, поэтому значение конъюнкции будет 0 независимо от значения второго операнда, второй операнд не нужно вычислять, результат всей операции конъюнкции уже определён), поэтому ошибки деления на ноль не будет.

Операция присваивания

В языке С присваивание является операцией, в отличие от ряда языков программирования (Паскаль, Фортран и другие), в которые включен оператор присваивания. Синтаксис операции присваивания: `<переменная> = <выражение>`. Сначала выполняется вычисление правого операнда (выражения). В качестве левого операнда рассмотрим имя переменной. В нее будет положен результат вычисления выражения. Присвоенное значение и объявляется результатом всего выражения и может быть использовано в дальнейших вычислениях. Чтобы выражение, содержащее операции присваивания, сделать оператором, нужно после него поставить точку с запятой²

`x = 3;` //переменной x присваивается значение 3.
`y = x = 1;` //с учетом правой ассоциативности операции присваивания можно поставить опущенные скобки так: `«y = (x = 1);»`. Сначала вычисляется правое присваивание – переменной x присваивается единица, результат этой операции – тоже 1, она же присваивается переменной y.

`y = (x = 85 - 1) / 2;` //вычисление этого выражения происходит следующим образом: сначала нужно вычислить выражение в скобках. Операция присваивания имеет более низкий приоритет, чем операция минус, поэтому сначала вычисляется `85 - 1`, получается результат 84. Этот результат операции минус присваивается переменной x, результат операции присваивания – тоже 84, оно же является результатом вычисления всего подвыражения в скобках. Этот результат используется в операции деления (у неё приоритет тоже больше, чем у присваивания), 84 делится на 2 и получается 42. Значение 42 – правый операнд внешней операции присваивания, результатом её выполнения будет присваивание переменной y значения 42. В итоге после вычисления этого выражения переменная x получит значение 84, а переменная y – значение 42. Значение 42 будет и результатом вычисления всего выражения, но оно далее не используется.

Введенные группы операций рассматривались в порядке убывания их приоритетов. Полная информация о приоритетах операций языка Си приведена в таблице 1.

операции (в порядке убывания приоритета)	способ вычисления
<code>() [] → .</code> постфиксные <code>++</code> и <code>--</code>	слева направо (→)
<code>! ~</code> префиксные <code>++</code> и <code>--</code> унарные <code>+</code> <code>-</code> <code>*</code> <code>&</code> (тип) <code>sizeof</code>	справа налево (←)
<code>*</code> <code>/</code> <code>%</code>	слева направо (→)
<code>+</code> <code>-</code>	слева направо (→)
<code><<</code> <code>>></code>	слева направо (→)
<code>==</code> <code>!=</code>	слева направо (→)
<code>&</code>	слева направо (→)
<code>^</code>	слева направо (→)
<code> </code>	слева направо (→)
<code>&&</code>	слева направо (→)
<code> </code>	слева направо (→)
<code>?:</code>	справа налево (←)
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>&=</code> <code>^=</code> <code> =</code> <code><=<</code> <code>>=></code>	справа налево (←)
<code>,</code>	слева направо (→)

Таблица 1. Приоритет операций и порядок вычислений

2 Это частный случай оператора-выражения, подробнее об операторе-выражении – в материале темы 2,

Задачи

1.1 Верно ли записаны константы, представляющие целочисленные значения? Для верно записанных констант определить их значение, тип.

123	1E6	123456789LU	-5	0XFUL
'0'	058	'\x7'	0X-1AD	'\122'
00123	0xfffffL	01A	-'x'	"x"
'a'U	0731UL	'\n'	+0xaf	0X0

1.2 Верно ли записаны константы с плавающей точкой? Для верно записанных констант определить их значение, тип.

1.71	1E-6	0.314159E1F	.005	0051E-04
5.E+2	0e0	0x1	A1.5	05.5 0
0X1E6	0F	1234.56789L	1.0E-10D	3.1415U
1e-2f	-12.3E-6	+10e6	123456L	E-6

1.3 Есть ли разница (с точки зрения языка Си) между числами

- a) 100 и 100.0,
- b) 20 и 2E1

1.4 Указать неправильные записи идентификаторов:

- a) task1.3 б) test1_1 в) prog1v1 г) 4you д) x₁ е) LXIV

1.5 Убрать из выражения лишние скобки, т.е. такие, удаление которых не изменит порядок выполнения операций в выражении:

- a) ((a*b) / c) % (a+b);
- б) (-n) % m;
- в) (z > 0) && (z < 50);
- г) (x!=y) || ((a/x)/(b/y) > 0)

1.6 Проверить на компьютере

- a) как работает операция % с отрицательными операндами
- b) сколько байт отводится под типы char, short, int, long

1.7 Используя только операции сложения, вычитания и присваивания, поменять местами значения двухцелочисленных переменных x и y. Начальными значениями x и y являются положительные целые числа из отрезка [0,60].

1.8 Используя операцию присваивания и необходимые арифметические операции, записать в переменную t

- a) среднее арифметическое целых чисел x, y и z;
- б) сумму квадратов вещественных x и y;
- в) квадрат расстояния между точками с координатами (x1, y1) и (x2, y2);
- г) объем цилиндра радиуса r высоты h.

1.9. Присвоить целой переменной h третью от конца цифру в записи положительного целого числа k (например, если k==130985, то h==9).

1.10. Целой переменной s присвоить сумму цифр трехзначного целого числа k.

1.11. Идет k-я секунда суток. Определить, сколько полных часов (h) и полных минут (m)

прошло к этому моменту (например, $h=3$ и $m=40$, если $k=13257=3*3600+40*60+57$).

1.12 Записать на языке Си выражение, истинное при выполнении указанного условия и ложное (равное 0) в противном случае:

- а) целое k делится на 7;
- б) уравнение $ax^2+bx+c=0$ ($a \neq 0$) не имеет вещественных корней;
- в) точка (x, y) лежит вне круга радиуса r с центром в точке $(1, 0)$;
- г) точка x принадлежит отрезку $[-1, 1]$;
- д) $0 < x < 1$;
- е) x является наибольшим из x, y, z ;
- ж) $x \neq \max(x, y, z)$ (операцию ! не использовать);
- з) целое y делится на 4, но не делится на 100;
- и) целое n делится на 7, а целое k делится на 13;
- к) целое n делится на 2 или 5.

1.13 Написать эквивалентное выражение, не содержащее операции !

$!(a > b)$	$!(2Ta == b + 4)$	$!(a < b \ \&\& \ c < d)$
$!(a < 2 \ \ a > 5)$	$!(a < 1 \ \ b < 2 \ \&\& \ c < 3)$	

1.14 Используя логические операции и операции сравнения, записать выражение, истинное (равное 1) при выполнении указанного условия и ложное (равное 0) в противном случае:

- а) x принадлежит отрезку $[0, 1]$;
- б) x лежит вне отрезка $[0, 1]$;
- в) x принадлежит отрезку $[2, 5]$ или $[-1, 1]$ (операцию ! не использовать);
- г) x лежит вне отрезков $[2, 5]$ и $[-1, 1]$;
- д) каждое из чисел x, y, z положительно;
- е) хотя бы одно из чисел x, y и z положительно;
- ж) ни одно из чисел x, y и z не является положительным;
- з) только одно из чисел x, y и z положительно;
- и) год с порядковым номером y является високосным (год високосный, если его номер кратен 4, однако из кратных 100 високосными являются лишь кратные 400; например, 1700, 1800, 1900 и 2001 – невисокосные годы, а 1600 и 2000 – високосные).

1.15 Нарисовать на плоскости (x, y) область, в которой и только в которой истинно указанное логическое выражение:

- а) $y \geq x \ \&\& \ y + x \geq 0 \ \&\& \ y \leq 1$;
- б) $x * x + y * y < 1 \ || \ y > 0 \ \&\& \ fabs(x) \leq 1$;

$fabs$ – функция, возвращающая модуль своего аргумента из библиотеки `math`. Для подключения этой библиотеки необходимо в начале программы указать директиву компилятору

```
#include <math.h>
```

При компиляции такой программы в операционной системе Unix необходимо

указать флаг $-lm$, например

`gcc f.c -ansi -o f -lm`

в) $(fabs(x) \leq 1) > (fabs(y) \geq 1)$;

г) $(x * x + y * y \leq 4) == (y \leq x)$;

д) $(fabs(x) + fabs(y) \leq 1) == (x * x + y * y > 4)$;

1.16 Записать на Си выражение, зависящее от x и y , которое принимает значение `true`, только если точка с координатами (x, y) принадлежит заштрихованной области (см. рис. 1). (Пунктирная граница означает, что она не входит в область.)

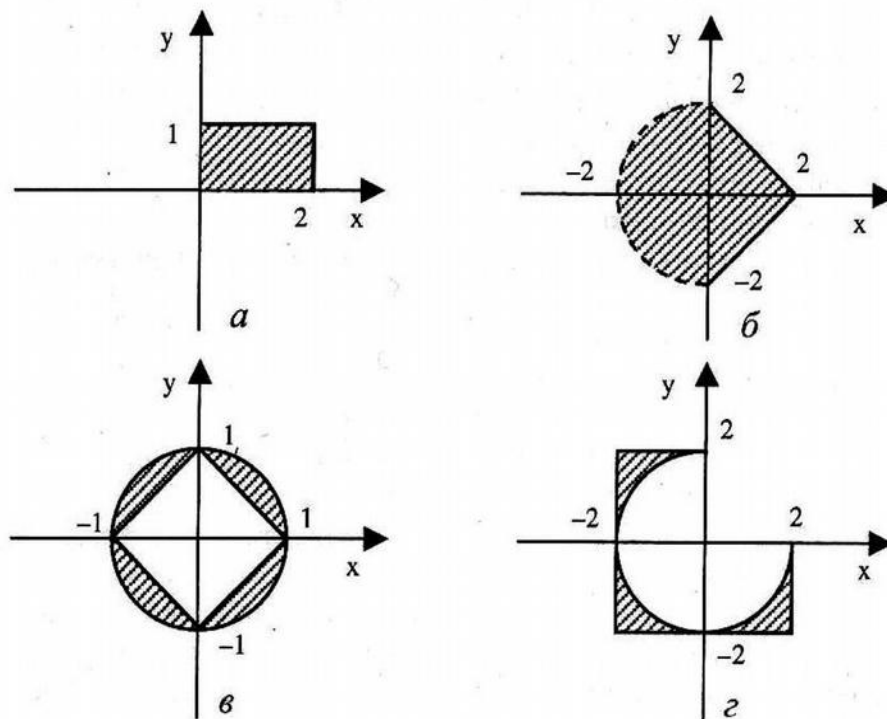


Рис. 1

1.17 Выписать оператор присваивания, в результате выполнения которого переменная t получает значение 1 (истина), если выполняется указанное условие, и значение 0 (ложь) иначе:

а) число x положительно;

б) целое число k нечетно;

в) p делится нацело на q (p и q – натуральные числа);

г) числа x, y, z равны между собой;

д) из чисел x, y, z только два равны между собой;

е) уравнение $ax^2+bx+c=0$, где a, b и c могут равняться 0, имеет ровно один корень;

ж) цифра 5 входит в десятичную запись трехзначного целого числа k

Тема 2

Оператор-выражение, операции с побочным эффектом, точки последовательных вычислений.

Составной оператор, условный оператор.

Одним из наиболее часто используемых операторов в программах на Си является **оператор-выражение**. Оператор-выражение получается из выражения, после которого поставлена точка с запятой.

Несмотря на то, что ; может превратить в оператор любое выражение (в том числе $1+2$; 44 ;) наибольший интерес представляют **выражения с побочным эффектом**, то есть такие, при вычислении которых не только есть результат вычислений, но и меняется состояние памяти, либо происходит запись информации в поток вывода, либо считывание данных. Ранее был рассмотрен пример операции с побочным эффектом: присваивание. Так у выражения $x = 0$ нам важен не столько его результат, сколько запись 0 в переменную x. В дополнение к операции присваивания = язык Си позволяет записывать целую группу операций присваивания вида $op=$, где op – арифметическая операция: $+=$, $-$, $=$, $*=$, $/=$, $\%=$.

Операции группы присваивания $X op= E$ сначала вычисляют выражение E, затем выполняют операцию op над значением X и переменной E, и присваивают полученный результат в X. Результатом операции считается присвоенное значение.

Приведем пример использования операций присваивания. Пусть в переменной x находится значение 8. После выполнения $x += 2$ значением переменной x станет 10. Если в одном выражении встречаются несколько операций группы присваивания, они выполняются справа налево: при вычислении выражения $a += b += c = 2$ сначала в c будет записано значение 2, затем исходное значение b увеличится на 2, затем к исходному значению a будет добавлено (b+2).

Рассмотрим и другие операции с побочным эффектом.

Операции ++ и --.

Префиксная операция ++ имеет один операнд-переменную (ячейку памяти), указывается перед ней, например, $++x$ – увеличивает значение переменной x на 1, результат её вычисления – новое увеличенное значение переменной. Если переменная x имела значение 5, то результат вычисления $++x$ будет 6, побочный эффект – значение переменной x тоже станет 6. Аналогичным образом вычисляется префиксная операция –, только происходит уменьшение значения на 1. Например, если значение переменной x было 5, то значением $--x$ будет 4, побочный эффект – значение переменной x станет 4.

Постфиксные ++ и -- имеют те же самые побочные эффекты, но результат их вычисления другой, чем у аналогичных префиксных операций. Например, если переменная x имела значение 5, то результатом вычисления $x++$ будет 5 ("старое" значение x), а побочный эффект – значение переменной x станет 6. Если переменная x имела значение 5, то результат выражения $x--$ равен значению 5, а побочный эффект вычисления этой операции – у переменной x значение станет равным 4.

Побочный эффект – изменение значения переменной – может быть произведён не сразу, а отложен на некоторое время. В программе есть так называемые точки последовательных вычислений, в которых гарантируется, что все присваивания, которые были записаны до нее в выражении, будут выполнены.

Точек последовательных вычислений (sequence points) всего пять:

1. **Конец выражения.** При завершении вычисления выражения все побочные эффекты, накопившиеся к соответствующему моменту, будут гарантированно произведены.

2. **Логические операции && и ||.**

После вычисления левого операнда и до вычисления правого будут произведены все присваивания, которые возникли при вычислении левого операнда.

3. **Точка в тернарной условной операции ? : .**

Тернарная операция ?: имеет следующий формат: E1 ? E2 : E3.

Сначала вычисляется условие – выражение E1. Если его значение отлично от нуля, то значением всего выражения будет значение выражения E2 (E3 при этом не вычисляется), в противном случае – значение выражения E3. Вопрос ? является точкой последовательных вычислений – после вычисления E1 (и до вычисления E2 или E3) гарантированно будут произведены все побочные эффекты, которые появились при вычислении выражения E1.

4. **Точка последовательных вычислений в операции ,**

Операция имеет следующий синтаксис E1, E2 (где E1 и E2 - выражения) . Она позволяет из нескольких выражений сделать одно. Вычисление сводится к последовательному вычислению сначала E1, затем E2. Результат E1 никуда не записывается, а результатом всего выражения объявляется результат E2 (правого операнда). После вычисления левого операнда и перед вычислением правого гарантируется, что все присваивания в левом выражении будут произведены.

Приведем пример: в выражении (x=2, y = x*10) сначала будет вычислено x = 2. При переходе через операцию , мы можем быть уверены, что изменение ячейки памяти x точно произошло, и в подвыражении y = x*10 уже использовать значение x=2. Результатом выражения будет 20.

5. **Точка входа в функцию.** К моменту входа в функцию гарантируется, что все побочные эффекты над параметрами будут выполнены.

Если между двумя точками последовательных вычислений одна и та же переменная меняется дважды (либо меняется и используется), то результат вычисления такого выражения не определен. Такие выражения будем считать ошибочными.

Рассмотрим еще несколько примеров выражений.

(a = 2) + a

Это «плохое» выражение – переменная a меняется операцией присваивания и используется между точками последовательных вычислений. Поскольку порядок вычисления аргументов сложения не фиксирован, точек последовательных вычислений внутри этого выражения нет, и совершенно не понятно, какое значение переменной a будет у правого аргумента сложения: старое, которое было до присваивания 2, или уже новое, равное числу 2. Поэтому результат вычисления такого выражения не определен.

(a = 1) + (b = 85)

«Хорошее» выражение – каждая переменная используется только один раз, хоть и в операциях с побочным эффектом. Результат вычисления всего выражения – 86, побочные эффекты после вычисления этого выражения – переменная a получит значение 1, переменная b получит значение 85.

(a = 2) && (b = a + 4) ;

Выражение «хорошее». Сначала вычисляется левый операнд конъюнкции, результат вычисления левого операнда – значение 2, оно же будет положено в переменную a. До вычисления правого операнда побочный эффект присваивания точно уже будет выполнен, так как мы проходим через операцию &&, а в ней есть точка последовательных вычислений. Поэтому переменной b присвоится значение 6. Результат вычисления всего выражения – 2 && 6 – будет равен 1 (два отличных от нуля значения – оба трактуются

как истина, результат конъюнкции – истина, в С логическая операции в этом случае возвращает 1).

```
x = 4; y = 7;  
(z = x > y) ? z += x-- : z -= ++y;
```

Значение последнего выражения зависит только от значений переменных x и y – несмотря на то, что переменная z входит в выражение три раза, причём является операндом операций с побочным эффектом. Выражение содержит точку последовательных вычислений - ?. Вычисление этого выражения происходит следующим образом: сначала вычисляется условие (то, что находится перед символом ?). Значение подвыражения $x > y$ ложно при заданных выше значениях переменных ($x=4, y=7$), поэтому операция $>$ вернёт значение 0, оно будет положено в переменную z и будет результатом этой операции присваивания. Поскольку условие ложно, далее будет вычисляться третья часть выражения, записанная после двоеточия. Но к моменту вычисления этого подвыражения побочный эффект операции присваивания переменной z значения 0 уже гарантированно будет выполнен, т.е. переменная z уже будет иметь значение 0. Вычисление выражения $z -= ++y$ начнётся с вычисления операции $++$, имеющей более высокий приоритет, чем операции присваивания. $++y$ вернёт 8, побочный эффект – переменная y получит значение 8 (старое значение увеличится на единицу), далее операция $-=$ уменьшит значение переменной z на 8, т.е. значение z станет -8. Второе подвыражение тернарной операции $z += x--$ при заданных значениях переменных x и y не будет вычисляться, поэтому содержащийся в нём побочный эффект уменьшения значения переменной y не будет выполнен.

```
printf("%d %d %d", ++a, ++a, ++a);
```

Это «плохое» выражение, поскольку порядок вычисления аргументов функции не фиксирован, точек последовательных вычислений в этом выражении нет, какое значение переменной a будет при вычислении каждого из аргументов – не определено. Если переписать его следующим образом:

```
x = ++a; y = ++a; z = ++a;  
printf("%d %d %d", x, y, z);
```

то проблема будет решена – до вычисления функции `printf` все побочные эффекты – изменения значения переменной a и присваивания переменным x, y и z этих значений будут выполнены.

Пустой, составной и условный операторы

Пустой оператор – это одиночная точка с запятой ; Такой оператор ничего не делает, но может быть использован там, где по синтаксису языка требуется указать какой-либо оператор, а делать ничего не нужно.

Если согласно синтаксису языка нужно написать один оператор, а необходимо выполнить последовательность действий, то несколько операторов можно объединить в один, используя составной оператор. **Составной оператор** – это несколько операторов, заключённые в фигурные скобки { }.

Условный оператор описывается следующим образом:

```
if (условие) Op1 else Op2
```

После `if` записывается условие – условное выражение, которое обязательно должно быть заключено в скобки. Если условие выполняется (значение выражения отлично от нуля – то есть истинно с точки зрения языка Си), то выполняется оператор `Op1`. Если значение условного выражения равно нулю (ложь с точки зрения языка Си), то выполняется `Op2`. Часть условного оператора `else Op2` может отсутствовать – в этом случае при ложном

условии сразу делается переход на следующее действие в программе.

Оператор Or1 или Or2 может быть составным, если нужно сделать несколько действий.

Пример программы с условным и составным операторами:

```
#include <stdio.h>
int main()
{   int x = 1, y;
    unsigned a;
    x = a = 5;
    if (x == 1)
    {
        x = 0;
        y = 1;
    }
    else
        printf("No");
    return 0;
}
```

Задачи

2.1 Напишите программу, решающую следующую задачу. В переменные x, y, z присвоены некоторые значения. Переупорядочить эти значения так, чтобы в x находилось наибольшее значение, в z — наименьшее из трех, а промежуточное — в y ($x > y > z$). Вывести в стандартный вывод новые значения x, y, z .

2.2 Написать программу для решения следующей задачи. В переменные a, b, c присвоить три числа. Найти корни квадратного уравнения с заданными коэффициентами a, b и c . Использовать функцию вычисления квадратного корня — `sqrt(аргумент)` из библиотеки `math`. Печать вещественного числа x — `printf("%lf\n", x)`

2.3. Верно ли решена задача: «значение целочисленной переменной c увеличить на 1; целочисленной переменной a присвоить значение, равное удвоенному значению переменной c ».

int a, c; c = 5;

а) $c++$;

$a = 2 \cdot c$;

б) $a = 2 \cdot c++$;

в) $c += 1$;

$a = c + c$;

г) $a = c++ + c$;

д) $++c$;

$a = c + c$;

е) $a = ++c + c$;

ж) $a = c += 1 + c$;

з) $a = (c+=1)+c$;

2.4 Верно ли решена задача: «значение целочисленной переменной c уменьшить на 1; целочисленной переменной a присвоить значение, равное частному от деления переменной c на 2».

int a, c; c = 5;

а) $--c$;

$a = c / 2$;

б) $a = --c / 2$;

в) $c -= 1$;

$a = c \% 2$;

г) $a = c -- / 2$;

д) $a = c -= 1 / 2$;

е) $a = (c = c - 1) / 2$;

ж) $a = (c -= 1) / 2$;

2.5 Верно ли записаны выражения? Если да, то вычислить их и написать, какими станут

значения переменных a, b, c. Если выражение записано неверно, объяснить, в чем ошибка.

а) `int a, b, c;`

`a = c = 1;`

`a = a++;`

`b = (c || 2) % (c && 2);`

б) `int a, b, c;`

`a = b = 7;`

`c = a++ <= --b || ++b >= 1 && 3.7`

в) `int a, b, c;`

`a = b = c = 3;`

`a += b += c;`

г) `int a, b, c;`

`a = b = 1;`

`c = ++a || ++b && ++a`

`a = 2.81 && 1 / 8`

2.6 Допустимо ли в Си? Если "да" - опишите семантику этих действий; если "нет" - объясните почему.

а) `int x;`

`x = 5;`

`++ x = 10;`

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

б) `int x, y;`

`x = 5;`

`y = x && ++ x;`

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

в) `int x = 2, y = 1, z = 0;`

`y = x && y || z;`

`x = x || !y && z;`

`z = x / ++x;`

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

г) `int x, y, z; x = y = z = 1;`

`x += y += z;`

`printf ("%d\n", x < y ? y++ : x++);`

`printf ("%d\n", z += x < y ? ++x : y--);`

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

`printf ("%d\n", z >= y && y >= x);`

д) `int x, y, z, i; x = y = z = 1;`

`i = ++x || ++y && ++z;`

`printf ("%d%d%d%d\n", x, y, z, i);`

`i = x++ <= --y || ++z >= i;`

`printf ("%d%d%d%d\n", x, y, z, i);`

```
е) int i, x, y;
   x = 5; y = 10; i = 15;
   x = ( y = 0, i = 1);
   printf("%d %d %d\n", i, x ,y);
   (x = y == 0) , i=1;
   printf("%d %d %d\n", i, x, y);

ж) int x = 2, y, z;
   x *= 3+2; x *= y = z = 4;
   printf ("%d %d %d\n", x, y, z);
   x = y == z;
   x == ( y = z );
   printf ("%d %d %d\n", x, y, z);
```

2.7 Для каких значений a и b результаты выражений будут различны? Указать хотя бы одну пару таких a и b , и вычислить выражения при этих значениях a и b . Какие значения будут в a и b после вычисления каждого из выражений?

Выражение 1: $a++ \ \&\& \ (b = a)$

Выражение 2: $++a \ \&\& \ (b = a)$

2.8 Определить, какое значение будет иметь переменная x после выполнения операторов:

а) $\text{if } (x > 0) \ x = 1; \text{ else if } (y > 0) \ x = 2; \text{ else } x = 3;$

б) $\text{if } (x > 0) \ x = 1; \text{ if } (y > 0) \ x = 2; \text{ else } x = 3;$

в) $\text{if } (x > 0) \ x = 1; \text{ else if } (y > 0) \ x = 2;$

г) $\text{if } (x > 0) \ x = 1; \text{ if } (y > 0) \ x = 2;$

д) $\text{if } (x > 0) \ \text{if } (y > 0) \ x = 1; \text{ else } x = 2; \text{ else } x = 3;$

е) $\text{if } (x > 0) \ \text{if } (y > 0) \ x = 1 \text{ else } x = 2;$

ж) $\text{if } (x > 0) \{ \text{if } (y > 0) \ x = 1; \} \text{ else } x = 2;$

если вначале переменным x и y присвоены следующие значения:

1) $x=4, y=4;$ 2) $x=-4, y=4;$ 3) $x=4, y=-4;$ 4) $x=-4, y=-4$

2.9 Известно, что из четырех чисел a_1, a_2, a_3 и a_4 одно отлично от трех других, равных между собой; присвоить номер этого числа переменной n . Записать указанное действие в виде одного условного оператора.

2.10 Написать программу для решения указанной задачи.

В переменные a, b, c присвоены три вещественных числа. Если нельзя построить треугольник с такими длинами сторон, то вывести 0, иначе вывести числа 3, 2 или 1 в зависимости от того, равносторонний это треугольник, равнобедренный или какой-либо иной.

2.11 Пусть в три вещественные переменные r_1, x_1, y_1 присвоены радиус и координаты центра первой окружности, а в переменные r_1, x_2, y_2 – второй окружности. Определить,

а) являются ли эти окружности концентрическими,

б) является ли первая окружность вложенной во вторую.

Литература

1. Л.С. Корухова, М.Р. Шура-Бура "Введение в алгоритмы" – М.: МГУ, 1997.
2. В.Н. Пильщиков, В.Г. Абрамов, А.А. Вылиток, И.В. Горячая "Машина Тьюринга и алгоритмы Маркова. Решение задач" – М.:ф-т ВМК МГУ, МАКС Пресс, 2016
3. Б. Керниган, Д. Ритчи "Язык программирования Си" 2-е изд. — М., С-Пб. : Диалектика (Вильямс), 2020.
4. Руденко Т.В. "Сборник задач и упражнений по языку Си (учебное пособие для студентов II курса)"– М.: Издательский отдел ВМК МГУ, 1999
5. В.Н. Пильщиков "Сборник задачи и упражнений по языку Паскаль" – М.: Научный мир, 2003