2.2. Базовые операции. Функции ввода-вывода

Операции, выражения

Выражения используются для вычисления значений (определенного типа) и состоят из операндов, операций и скобок. Каждый операнд может быть, в свою очередь, выражением или одним из его частных случаев – константой или переменной. Операнды задают данные для вычислений.

Знак операции – это один или более символов, определяющих действие над операндами, т.е. операции задают действия, которые необходимо выполнить. Внутри знака операции пробелы не допускаются.

Операции делятся на унарные, бинарные и тернарные – по количеству участвующих в них операндов, и выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки.

Большинство операций выполняются слева направо, например, $a+b+c \rightarrow \rightarrow (a+b)+c$. Исключение составляют унарные операции, операции присваивания и условная операция (?:), которые выполняются справа налево.

В языке Си используются четыре унарные операции, имеющие самый высокий приоритет, их часто называют первичными:

– операция доступа к полям структур и объединений при помощи идентификаторов «**.**» – точка;

– операция доступа к полям структур и объединений при помощи указателей «**–>**» – стрелка;

– операция [ ] индексации, используемая при декларации массива и обращении к его элементам;

– операция ( ) обращения к функции.

Первичные операции будут рассмотрены в соответствующих разделах.

Полный список операций с указанием их приоритетов приводится в прил. 2.

Арифметические операции

Обозначения арифметических операций:
+ (сложение); – (вычитание); / (деление, для *int* операндов – с отбрасыванием остатка); * (умножение); % (остаток от деления целочисленных операндов со знаком первого операнда – деление «по модулю»).

Операндами традиционных арифметических операций (+ – * /) могут быть константы, переменные, обращения к возвращающим значения функциям, элементы массивов, любые арифметические выражения, указатели (с ограничениями).

Порядок выполнения действий в арифметических выражениях следующий: выражения в круглых скобках; операции *, /, %; операции **+, –.**

Унарные операции «знак числа» (+, –) обладают самым высоким приоритетом и определены для операндов числовых типов (имеющих числовой результат), при этом «+» носит только информационный характер, «–» меняет знак операнда на противоположный (неадресная операция).

Операции *, /, % обладают высшим приоритетом над операциями **+, –**, поэтому при записи сложных выражений нужно использовать общепринятые математические правила: $x + y \cdot z - \frac{a}{b+c}$ ↔ *x*+*y***z*–*a* /(*b*+*c*) , т.е. использовать круглые скобки.

Операция присваивания

Формат операции присваивания:

***Операнд*_1 = *Операнд*_2 ;**

*Операндом*_1 (левый операнд) может быть только переменная. Левый операнд операции присваивания получил название ***L–значение***, (*L–value*, *Left–value*) – *адресное выражение*. Так в Си называют любое выражение, адресующее некоторый участок оперативной памяти, в который можно записать некоторое значение. Переменная – это частный случай адресного выражения.

*Операндом*_2 (правый операнд) могут быть: константа, переменная или любое выражение, составленное в соответствии с синтаксисом языка Си. Правый операнд операции присваивания назвали ***R–значение***, (*R–value*, *Right–value*).

Присваивание значения в языке Си, в отличие от традиционной интерпретации, рассматривается как выражение, имеющее значение левого операнда после присваивания. Таким образом, присваивание может включать несколько операций присваивания, изменяя значения нескольких операндов, например:

```
int i, j, k;
float x, y, z;
   ...
i = j = k = 0;              ↔   k = 0, j = k, i = j;
x = i + (y = 3) – (z = 0);  ↔   z = 0, y = 3, x = i + y – z;
```

Примеры недопустимых выражений:

– присваивание константе: 2 = *x* + *y*;
– присваивание функции: *getch*() = *i*;
– присваивание результату операции: (*i* + 1) = 2 + *y*;

Сокращенная запись операции присваивания

В языке Си используются два вида сокращенной записи операции присваивания:

1) вместо записи: ***v* = *v* # *e***;

где # – любая арифметическая операция (операция над битовым представлением операндов), рекомендуется использовать запись ***v* #= *e***;

Например, *i* = *i* + 2; ↔ *i* += 2; (***знаки операций – без пробелов***);

2) вместо записи: *x* = *x* # 1;

где # – символы, обозначающие операцию инкремента (+1), либо декремента (–1), *x* – целочисленная переменная (или переменная-указатель), рекомендуется использовать запись:

##x; – префиксную, или ***x##;*** – постфиксную.

Если эти операции используются в чистом виде, то различий между постфиксной и префиксной формами нет. Если же они используются в выражении, то в префиксной форме (*##x*) сначала значение *x* изменится на 1, а затем полученный результат будет использован в выражении; в постфиксной форме (*x##*) – сначала значение переменной *x* используется в выражении, а затем изменится на 1. Операции над указателями будут рассмотрены в разд. 9.4.

Пример 1:

int i, j, k;	Смысл записи
float x, y;	
...	
x *= *y*;	*x* = x*y;
i += 2;	*i* = *i* + 2;
x /= *y*+15;	*x* = *x*/(*y* + 15);
--*k*;	*k* = *k* – 1;
k--;	*k* = *k* – 1;
j = *i*++;	*j* = *i*; *i* = *i* + 1;
j = ++*i*;	*i* = *i* + 1; *j* = i;

Пример 2:

int n, a, b, c, d;	Значения
n = 2; *a* = *b* = *c* = 0;	
a = ++*n*;	*n*=3, *a*=3
a += 2;	*a*=5
b = *n*++;	*b*=3, *n*=4
b –= 2;	*b*=1
c = --*n*;	*n*=3, *c*=3
c *= 2;	*c*=6
d = *n*--;	*d*=3, *n*=2
d %= 2;	*d*=1

Преобразование типов операндов арифметических операций

Если операнды арифметических операндов имеют один тип, то и результат операции будет иметь такой же тип.

Но, как правило, в операциях участвуют операнды различных типов. В этом случае они преобразуются к общему типу в порядке увеличения их «размера памяти», т.е. объема памяти, необходимого для хранения их значений. Поэтому неявные преобразования всегда идут от «меньших» объектов к «большим». Схема выполнения преобразований операндов арифметических операций выглядит следующим образом:

short, char → ***int*** → ***unsigned*** → ***long*** → ***double***

float → ***double***

Стрелки отмечают преобразования даже однотипных операндов перед выполнением операции. То есть действуют следующие правила:

– значения типов *char* и *short* всегда преобразуются в *int*;

– если один из операндов имеет тип *double*, то и другой преобразуется в *double*;

– если один из операндов *long*, то другой преобразуется в *long*.

Внимание. Результатом операции **1/3** будет значение **0**, чтобы избежать такого рода ошибок, необходимо явно изменить тип хотя бы одного операнда, т.е. записать, например: 1./3 .

Типы *char* и *int* могут свободно смешиваться в арифметических выражениях. Каждая переменная типа *char* автоматически преобразуется в *int*, что обеспечивает значительную гибкость при проведении преобразований, т.к. над типом *int* действия выполняются быстрее, чем над любым другим типом.

При выполнении операции присваивания значение правого операнда преобразуется к типу левого, который и является типом полученного результата. И здесь необходимо быть внимательным, т.к. при некорректном использовании операций присваивания могут возникнуть неконтролируемые ошибки. Так, при преобразовании *int* в *char* старший байт просто отбрасывается.

Пусть: *float x*; *int i*; тогда и $x = i$; и $i = x$; приводят к преобразованиям, причем *float* преобразуется в *int* отбрасыванием дробной части.

Тип *double* преобразуется в *float* округлением.

Длинное целое преобразуется в более короткое целое и *char* посредством отбрасывания бит в старших разрядах.

Итак, безопасным преобразованием типов является преобразование в порядке увеличения «размера памяти», обратное преобразование может привести к потере значащих разрядов.

Операция приведения типа

В любом выражении преобразование типов может быть осуществлено явно, для этого достаточно перед выражением поставить в круглых скобках атрибут соответствующего типа:

(*тип*) *выражение*;

ее результат – значение *выражения*, преобразованное к заданному *типу*.

Операция приведения типа вынуждает компилятор выполнить указанное преобразование, но ответственность за последствия возлагается на программиста. Использовать эту операцию рекомендуется везде, где это необходимо, например:

double x;
int n = 6, *k* = 4;
$x = (n + k)/3$; $\rightarrow$ $x = 3$, т.к. дробная часть будет отброшена;
$x = (double)(n + k)/3$; $\rightarrow$ $x = 3.333333$ – использование операции приведения типа позволило избежать округления результата деления целочисленных операндов.

Операции сравнения

В языке Си используются следующие операции сравнения, т.е. отношения между объектами:

==	– равно или эквивалентно;	!=	– не равно;
<	– меньше;	<=	– меньше либо равно;

> – больше; >= – больше либо равно.

Пары символов соответствующих операций разделять нельзя.

Общий вид операций отношений:

Операнд_1 Знак операции Операнд_2

Указанные операции выполняют сравнение значений первого операнда со вторым. Операндами могут быть любые арифметические выражения и указатели.

Значения арифметических выражений перед сравнением вычисляются и преобразуются к одному типу.

Арифметические операнды преобразуются по правилам, аналогичным для арифметических операций. Операнды-указатели преобразуются в целые числа необходимого типа. Результат сравнения указателей будет корректным в арифметическом смысле лишь для объектов одного массива.

В языке Си нет логического типа данных. Результат операции отношения имеет значение **1**, если отношение истинно, или в результате вычислений получено не нулевое значение, воспринимаемое компилятором Си как истина *(true)*, или **0** – в противном случае, т.е. – ложно *(false)*. Следовательно, операция отношения может использоваться в любых арифметических выражениях.

Операции сравнения на равенство и неравенство имеют меньший приоритет, чем остальные операции отношений.

Примеры использования операций отношений:

$y > 0$, $x == y$, $x != 2$.

Отношения между объектами сложных типов проверяются либо посредством последовательного сравнения их элементов (для массивов), либо используя стандартные библиотечные функции, которые будут рассмотрены позже.

Логические операции

Приведем логические операции в порядке убывания относительного приоритета. Их обозначения:

! – отрицание (логическое «НЕТ»);

&& – конъюнкция (логическое «И»);

|| – дизъюнкция (логическое «ИЛИ»).

Операндами (выражениями) логических операций могут быть любые скалярные типы. Ненулевое значение операнда трактуется как «истина», а нулевое – «ложь». Результатом логической операции, как и в случае операций отношения, может быть 1 или 0.

Общий вид операции ***отрицания***

! *выражение*

Примеры использования операции отрицания:

!0 $\rightarrow$ 1

!5 $\rightarrow$ 0

x = 10;

! (x > 0) $\rightarrow$ 0

Общий вид операций ***конъюнкции*** и ***дизъюнкции***:

*Выражение*_1 ***знак операции*** *Выражение*_2

Особенность операций конъюнкции и дизъюнкции – экономное последовательное вычисление выражений-операндов:

– если выражение_1 операции «конъюнкция» ложно, то результат операции – ноль и выражение_2 не вычисляется;

– если выражение_1 операции «дизъюнкция» истинно, то результат операции – единица и выражение_2 не вычисляется.

Например:

$y > 0$ && $x = 7$ → истина, если оба выражения истинны;

$e > 0$ || $x = 7$ → истина, если хотя бы одно выражение истинно.

Старшинство операции «И» выше, чем «ИЛИ» и обе они младше операций отношения и равенства.

Относительный приоритет логических операций позволяет пользоваться общепринятым математическим стилем записи сложных логических выражений, например:

$0 < x < 100$ ↔ $0 < x$ && $x < 100$;

$x > 0, y \le 1$ ↔ $x > 0$ && $y <= 1$.

Учет этих свойств очень существенен для написания правильно работающих программ.

Побитовые логические операции, операции над битами

В языке Си предусмотрен набор операций для работы с отдельными битами. Эти операции нельзя применять к переменным вещественного типа.

Обозначения операций над битами:

~ – дополнение (унарная операция); инвертирование (одноместная операция);

& – побитовое «И» – конъюнкция;

| – побитовое включающее «ИЛИ» – дизъюнкция;

^ – побитовое исключающее «ИЛИ» – сложение по модулю 2;

>> – сдвиг вправо;

<< – сдвиг влево.

Общий вид операции инвертирования (поразрядное отрицание):

~ выражение

инвертирует каждый разряд в двоичном представлении своего операнда.

Остальные операции над битами имеют вид:

*Выражение*_1 ***знак операции*** *Выражение*_2

Операндами операций над битами могут быть только *выражения*, приводимые к целому типу. Операции (~, &, |, ^) выполняются поразрядно над всеми битами операндов (знаковый разряд особо не выделяется):

~0*x*F0 ↔ *x*0F
0*x*FF & 0*x*0F ↔ *x*0F
0*x*F0 | 0*x*11 ↔ *x*F1
0*x*F4 ^ 0*x*F5 ↔ *x*01

Операция & часто используется для маскирования некоторого множества бит. Например, оператор *w* = *n* & 0177 передает в *w* семь младших бит *n*, полагая остальные равными нулю.

Операции сдвига выполняются также для всех разрядов с потерей выходящих за границы бит.

Операция (|) используется для включения бит *w* = *x* | *y*, устанавливает в единицу те биты в *x*, которые равны 1 в *y*.

Необходимо отличать побитовые операции & и | от логических операций && и || , если *x* = 1, *y* = 2, то *x* & *y* равно нулю, а *x* && *y* равно 1.

0*x*81 << 1 ↔ 0*x*02
0*x*81 >> 1 ↔ 0*x*40

Если *выражение*_1 имеет тип *unsigned*, то при сдвиге вправо освобождающиеся разряды гарантированно заполняются нулями (логический сдвиг). Выражения типа *signed* могут, но необязательно, сдвигаться вправо с копированием знакового разряда (арифметический сдвиг). При сдвиге влево освобождающиеся разряды всегда заполняются нулями. Если *выражение*_2 отрицательно либо больше длины *выражения*_1 в битах, то результат операции сдвига не определен.

Унарная операция (~) дает дополнение к целому, т.е. каждый бит со значением 1 получает значение 0 и наоборот.

Операции сдвига << и >> применяются к целочисленным операндам и осуществляют соответственно сдвиг вправо (влево) своего левого операнда на число позиций, задаваемых правым операндом, например, *x* << 2 сдвигает *x* влево на две позиции, заполняя освобождающиеся биты нулями (эквивалентно умножению на 4).

Операции сдвига вправо на *k* разрядов весьма эффективны для деления, а сдвиг влево – для умножения целых чисел на 2 в степени *k*:

x << 1 ↔ *x**2; *x* >> 1 ↔ *x*/2 ;
x << 3 ↔ *x**8 .

Подобное применение операций сдвига безопасно для беззнаковых и положительных значений *выражения*_1.

Операции сдвига не учитывают переполнение и потерю значимости.

В математическом смысле операнды логических операций над битами можно рассматривать как отображение некоторых множеств с размерностью не более разрядности операнда на значения {0,1}.

Пусть единица означает обладание элемента множества некоторым свойством, тогда очевидна теоретико-множественная интерпретация рассматриваемых операций:

~ – дополнение; **|** – объединение; **&** – пересечение.

Простейшее применение – проверка нечетности целого числа:

```
int i;
...
if ( i & 1) printf (" Значение i четно!");
```

Комбинирование операций над битами с арифметическими операциями часто позволяет упростить выражения.

Операция «,» (запятая)

Данная операция используется при организации строго гарантированной последовательности вычисления выражений (обычно используется там, где по синтаксису допустима только одна операция, а необходимо разместить две и более, например, в операторе *for*). Форма записи:

***выражение_*1, …, *выражение_N*;**

выражения 1, 2,…, *N* вычисляются последовательно друг за другом и результатом операции становится значение последнего выражения *N*, например:

***m* = (*i* = 1, *j* = *i* ++, *k* = 6, *n* = *i* + *j* + *k*);**

получим последовательность вычислений: ***i* =1, *j* = *i* =1, *i* = 2, *k* = 6, *n* = 2+1+6,** и в результате ***m* = *n* = 9.**

В заключение отметим следующую особенность языка Си – любые операции допускаются только со скалярными объектами, причем небольшого размера, порядка размера регистров процессора. Это объясняется ориентацией языка на задачи системного программирования. Любые действия с составными или сложными объектами – массивами, строками, структурами и т.п. реализуются с помощью стандартных библиотечных функций, работа с которыми будет рассмотрена позже.

Стандартная библиотека языка Си

В любой программе кроме операторов и операций используются средства библиотек, входящих в среду программирования. Часть библиотек стандартизована и поставляется с компилятором. Функции, входящие в библиотеку языка Си, намного облегчают создание программ. Расширение библиотечных файлов **.lib.*

В стандартную библиотеку входят также прототипы функций, макросы, глобальные константы. Это, как вы уже знаете, заголовочные файлы с расширением **.h*, которые хранятся в папке *include* и подключаются на этапе предпроцессорной обработки исходного текста программ.

Рассмотрим наиболее часто используемые функции из стандартной библиотеки языка Си.

Стандартные математические функции

Математические функции языка Си декларированы в файлах ***math.h*** и ***stdlib.h.***

В приведенных здесь функциях аргументы и возвращаемый результат имеют тип *double*. Аргументы тригонометрических функций должны быть заданы в радианах (2π радиан = 360^o).

Математическая функция	*ID* функции в языке Си
$\sqrt{x}$	sqrt(*x*)
$\|x\|$	fabs(*x*)
e^x	exp(*x*)
x^y	pow(*x*,*y*)
$\ln(x)$	log(*x*)
$\lg_{10}(x)$	log10(*x*)
sin(*x*)	sin(*x*)
cos(*x*)	cos(*x*)
tg(*x*)	tan(*x*)
arcsin(*x*)	asin(*x*)
arccos(*x*)	acos(*x*)
arctg(*x*)	atan(*x*)
arctg(*x* / *y*)	atan2(*x*)
$sh(x)=0.5\ (e^x-e^{-x})$	sinh(*x*)
$ch(x)=0.5\ (e^x+e^{-x})$	cosh(*x*)
tgh(*x*)	tanh(*x*)
остаток от деления *x* на *y*	fmod(*x*,*y*)
наименьшее целое >=*x*	ceil(*x*)
наибольшее целое <=*x*	floor(*x*)

Функции вывода данных на дисплей

В языке Си нет встроенных средств ввода/вывода данных. Ввод/вывод информации осуществляется с помощью библиотечных функций и объектов.

Декларации функций ввода/вывода, как уже упоминалось, приведены в заголовочном файле ***stdio.h.***

Для вывода информации на экран монитора (дисплей) в языке Си чаще всего используются функции: ***printf***() и ***puts***().

Формат функции форматного вывода на экран:

printf*(управляющая строка , список объектов вывода);*

В ***управляющей строке***, заключенной в кавычки, записывают: поясняющий текст, который выводится на экран без изменения (комментарии), список модификаторов форматов, указывающих компилятору способ вывода объектов (признак модификатора формата – символ **%**) и специальные символы, управляющие выводом (признак – символ \).

В ***списке объектов вывода*** указываются идентификаторы печатаемых объектов, разделенных запятыми: переменные, константы или выражения, вычисляемые перед выводом.

Количество и порядок следования форматов должен совпадать с количеством и порядком следования выводимых на экран объектов.

Функция *printf* выполняет вывод данных в соответствии с указанными форматами, поэтому формат может использоваться и для преобразования типов выводимых объектов.

Если признака модификации (%) нет, то вся информация выводится как комментарии.

Основные модификаторы формата:

%*d* (%*i*) – десятичное целое число;
%*c* – один символ;
%*s* – строка символов;
%*f* – число с плавающей точкой, десятичная запись;
%*e* – число с плавающей точкой, экспоненциальная запись;
%*g* – используется вместо ***f, e*** для исключения незначащих нулей;
%*o* – восьмеричное число без знака;
%*x* – шестнадцатеричное число без знака.

Для чисел ***long*** добавляется символ ***l***, например, %*ld* – длинное целое, %*lf* – число вещественное с удвоенной точностью – ***double***.

Если нужно напечатать сам символ %, то его нужно указать 2 раза:

printf ("Только %d%% предприятий не работало. \n",5);

Получим: ***Только 5% предприятий не работало***.

Управляют выводом специальные последовательности символов: ***n*** – новая строка; ***t*** – горизонтальная табуляция; ***b*** – шаг назад; ***r*** – возврат каретки; ***v*** – вертикальная табуляция; \\ – обратная косая; \' – апостроф; \" – кавычки; **\0** – нулевой символ (пусто).

Пример:

```
#define  PI   3.14159
                . . .
        int number = 5;
        float bat = 255;
        int cost = 11000;
                . . .
        printf(" %d студентов съели %f бутербродов. \n", number, but);
        printf(" Значение числа pi равно %f. \n", pi);
        printf(" Стоимость этой вещи %d %s. \n", cost, "Руб.");
        . . .
```

В модификаторах формата функции *printf* после символа % можно указывать число, задающее минимальную ширину поля вывода, например, %5*d* – для целых, %4.2*f* – для вещественных – две цифры после запятой для поля шириной 4

символа. Если указанных позиций для вывода целой части числа не хватает, то происходит автоматическое расширение.

Если после «%» указан знак «минус», то выводимое значение будет печататься с левой позиции поля вывода, заданной ширины, например: % – 10d.

Использование функции *printf* для преобразования данных:

1) printf("%d", 336.65); получим: 336;
2) printf("%o", 336); получим: 520, т.е. 5*8**2+2*8+0*1 = 336;
3) printf("%x", 336); получим: 150 (шестнадцатеричное).

Можно использовать функцию *printf* для нахождения кода ASCII некоторого символа:

```
printf (" %c – %d\n", 'a', 'a');
```

получим десятичный код ASCII символа a: a – 65 .

Функция ***puts***(*ID* строки); выводит на экран дисплея строку символов, автоматически добавляя к ней символ перехода на начало новой строки (\n).

Аналогом такой функции будет: printf(“%s \n”, *ID* строки);

Функция ***putchar***() выдает на экран дисплея один символ без добавления символа ‘\n’.

Функции ввода информации

Функция, предназначенная для форматированного ввода исходной информации с клавиатуры:

scanf (*управляющая строка , список адресов объектов ввода*);

в *управляющей строке* указываются только модификаторы форматов, количество и порядок следования которых должны совпадать с количеством и порядком следования вводимых объектов, а тип данных будет преобразовываться в соответствии с модификаторами.

Список объектов ввода представляет собой адреса переменных, разделенные запятыми, т.е. для ввода значения переменной перед ее идентификатором указывается символ *&*, обозначающий операцию «***взять адрес***».

Если нужно ввести значение строковой переменной, то использовать символ *&* не нужно, т.к. строка – это массив символов, а *ID* массива является адресом его первого элемента. Например:

```
int course;
double grant;
char name[20];
printf (" Укажите курс, стипендию, имя \n ");
scanf ("%d %lf %s", &course, &grant, name);
```

Вводить данные с клавиатуры можно как в одной строке через пробелы, так и в форме разных строк, нажимая после ввода текущего объекта клавишу *Enter*.

Функция *scanf*() использует практически тот же набор модификаторов форматов, что и *printf*(); отличия от функции вывода следующие: отсутствует формат %*g*, форматы %*e*,%*f* – эквивалентны. Для ввода коротких целых чисел введен модификатор формата %*h*.

Внимание. Функцией *scanf*() по формату %*s* строка вводится только до первого пробела.

Для ввода фраз, состоящих из слов, разделенных пробелами, используется функция

gets (*ID строковой переменной*);

Символы вводятся при помощи функции ***getch***(). Причем простой ее вызов организует паузу, при которой система программирования приостановит выполнение программы и будет ждать нажатия любой клавиши. Так поступают в том случае, когда нужно просмотреть какие-то результаты работы, при выводе их на экран монитора.

Если же использовать ее в правой части операции присваивания, например:

```
char c;
. . .
c = getch();
```

то символьная переменная *c* получит значение кода нажатой клавиши.

С началом работы любой программы автоматически открываются стандартные потоки для ввода (***stdin*)** и вывода данных (***stdout*)**, которые по умолчанию связаны с клавиатурой и экраном монитора соответственно.

Внимание. Ввод данных функциями *gets*(), *getch*() выполняется с использованием потока *stdin*. Если указанная функция не выполняет своих действий (проскакивает), перед использованием необходимо очистить поток (буфер) ввода с помощью функции

fflush(*stdin*);

В языке C++ существует наиболее простая с точки зрения использования возможность ввода-вывода – потоковый ввод-вывод, основы которого рассмотрены в разд. 16.1, 16.2.

Советы по программированию

При выполнении вариантов заданий придерживайтесь следующих ключевых моментов.

1. Выбирайте тип для переменных с учетом диапазона их возможных значений и требуемой точности представления данных.
2. Старайтесь давать переменным *ID* (имена), отражающие их назначение.
3. При вводе данных с клавиатуры выводите на экран пояснения: что нужно ввести, т.е. организуйте диалог. Для контроля сразу же после ввода выводите исходные данные на дисплей (хотя бы в процессе отладки).

4. До запуска программы подготовьте тестовые примеры, содержащие исходные данные и ожидаемые результаты. Отдельно нужно проверить реакцию программы на заведомо неверные исходные данные. Для таких ситуаций необходимо предусмотреть вывод сообщений, например, «Ошибка! Решения нет».

5. При составлении выражений учитывайте приоритет используемых операций.

6. В функциях ввода/вывода *printf* и *scanf* для каждой переменной указывайте спецификацию формата, соответствующую ее типу. Не забывайте, что в функции *scanf* передается адрес переменной, а не ее значение.

7. При использовании стандартных функций требуется с помощью директивы препроцессору *include* подключить к программе соответствующие заголовочные файлы. Установить, какой именно файл необходим, можно с помощью справочной системы *Visual* C++ 6.0 – «*MSDN*».

8. Данные при вводе разделяйте пробелами, символами перевода строки или табуляции, но не запятыми.

Не смешивайте в одной программе ввод/вывод с помощью стандартных функций (в стиле Си) с вводом/выводом в потоке (в стиле C++).

ЗАДАНИЕ 1. Составление линейных алгоритмов

Первый уровень сложности

Составить программу для расчета двух значений z_1 и z_2, результаты которых должны совпадать [32]. Ввод исходных данных можно задавать при декларации или вводить с клавиатуры. Игнорировать возможность деления на ноль. Значение $\pi = 3{,}1415926$.

1. $z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$, $z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$.

2. $z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$, $z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$.

3. $z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$, $z_2 = 2\sin\alpha$.

4. $z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\beta}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\beta}{4}\right)$, $z_2 = \frac{\sqrt{2}}{2}\sin\frac{\beta}{2}$.

5. $z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$, $z_2 = \cos^2\alpha + \cos^4\alpha$.

6. $z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$, $z_2 = 4\cos\frac{\alpha}{2}\cdot\cos\frac{5}{2}\alpha\cdot\cos 4\alpha$.

7. $z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$, $z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$.

8. $z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$, $z_2 = \sin(y+x)\cdot\sin(y-x)$.

9. $z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$, $z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cdot\cos(\alpha+\beta)$.

10. $z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$, $z_2 = ctg\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$.

11. $z_1 = \frac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$, $z_2 = \frac{1 - tg\alpha}{1 + tg\alpha}$.

12. $z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha}\cdot\frac{\cos 2\alpha}{1 + \cos 2\alpha}$, $z_2 = ctg\left(\frac{3}{2}\pi - \alpha\right)$.

13. $z_1 = \frac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$, $z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$.

14. $z_1 = \frac{(m-1)\sqrt{m} - (n-1)\sqrt{n}}{\sqrt{m^3 n} + nm + m^2 - m}$, $z_2 = \frac{\sqrt{m} - \sqrt{n}}{m}$.

15. $z_1 = \frac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2}$, $z_2 = \frac{1}{\sqrt{b+2}}$.

Второй уровень сложности

Составить программу для расчета заданных выражений. Вводить исходные данные с клавиатуры. Обязательно проверять исключительные ситуации. Значение π = 3,1415926.

1. $t = \frac{2\cos\left(x - \frac{\pi}{6}\right)}{0{,}5 + \sin^2 y}\left(1 + \frac{z^2}{3 - z^2/5}\right)$.

При x = 14.26, y = –1.22, $z = 3.5\times10^{-2}$, результат t = 0.564849.

2. $u = \frac{\sqrt[3]{8 + |x-y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|}\left(tg^2 z + 1\right)^x$.

При x = –4.5, $y = 0.75\times10^{-4}$, $z = 0.845\times10^{2}$, результат u = –55.6848.

3. $v = \dfrac{1+\sin^2(x+y)}{\left|x - \dfrac{2y}{1+x^2y^2}\right|} x^{|y|} + \cos^2\left(arctg\dfrac{1}{z}\right)$.

При $x = 3.74\times10^{-2}$, y = –0.825, $z = 0.16\times10^{2}$, результат v = 1.0553.

4. $w = \left|\cos x - \cos y\right|^{\left(1+2\sin^2 y\right)}\left(1 + z + \dfrac{z^2}{2} + \dfrac{z^3}{3} + \dfrac{z^4}{4}\right)$.

При $x = 0.4\times10^{4}$, y = –0.875, $z = -0.475\times10^{-3}$, результат w = 1.9873.

5. $\alpha = \ln\left(y^{-\sqrt{|x|}}\right)\left(x - \dfrac{y}{2}\right) + \sin^2 arctg(z)$.

При x = –15.246, $y = 4.642\times10^{-2}$, $z = 20.001\times10^{2}$, результат α = –182.036.

6. $\beta = \sqrt{10\left(\sqrt[3]{x} + x^{y+2}\right)} \cdot \left(\arcsin^2 z - |x - y|\right)$.

При $x = 16.55\times10^{-3}$, y = –2.75, z = 0.15, результат β = –38.902.

7. $\gamma = 5arctg(x) - \dfrac{1}{4}\arccos(x)\dfrac{x + 3|x-y| + x^2}{|x-y|z + x^2}$.

При x = 0.1722, y = 6.33, $z = 3.25\times10^{-4}$, результат γ = –172.025.

8. $\varphi = \dfrac{e^{|x-y|}|x-y|^{x+y}}{arctg(x) + arctg(z)} + \sqrt[3]{x^6 + \ln^2 y}$.

При $x = -2.235\times10^{-2}$, y = 2.23, z = 15.221, результат φ = 39.374.

9. $\psi = \left|x^{\frac{y}{x}} - \sqrt[3]{\dfrac{y}{x}}\right| + (y-x)\dfrac{\cos y - \dfrac{z}{(y-x)}}{1 + (y-x)^2}$.

При $x = 1.825\times10^{2}$, y = 18.225, $z = -3.298\times10^{-2}$, результат ψ = 1.2131.

10. $a = 2^{-x}\sqrt{x + \sqrt[4]{|y|}}\sqrt[3]{e^{x-1/\sin z}}$.

При $x = 3.981\times10^{-2}$, $y = -1.625\times10^{3}$, z = 0.512, результат a = 1.26185.

11. $b = y^{\sqrt[3]{|x|}} + \cos^3(y)\dfrac{|x-y|\cdot\left(1 + \dfrac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|} + \dfrac{x}{2}}$.

При x = 6.251, y = 0.827, z = 25.001, результат b = 0.7121.

$$12.\ c = 2^{\left(y^x\right)} + \left(3^x\right)^y - \frac{y \cdot \left(arctgz - \dfrac{\pi}{6}\right)}{|x| + \dfrac{1}{y^2 + 1}}.$$

При x = 3.251, y = 0.325, z = 0.466×10^{-4}, результат c = 4.025.

$$13.\ f = \frac{\sqrt[4]{y + \sqrt[3]{x - 1}}}{|x - y|\left(\sin^2 z + tgz\right)}.$$

При x = 17.421, y = 10.365×10^{-3}, z = 0.828×10^{5}, результат f = 0.33056.

$$14.\ g = \frac{y^{x+1}}{\sqrt[3]{|y - 2|} + 3} + \frac{x + \dfrac{y}{2}}{2|x + y|}(x + 1)^{-1/\sin z}.$$

При x = 12.3×10^{-1}, y = 15.4, z = 0.252×10^{3}, результат g = 82.8257.

$$15.\ h = \frac{x^{y+1} + e^{y-1}}{1 + x|y - tgz|}\left(1 + |y - x|\right) + \frac{|y - x|^2}{2} - \frac{|y - x|^3}{3}.$$

При x = 2.444, y = 0.869×10^{-2}, z = –0.13×10^{3}, результат h = –0.49871.