Лекция 7 Арифметические и логические выражения. Массивы

7.1. ***Арифметические типы данных***:

7.1.1. Целочисленные: **`int`**, **`short int`**, **`long int`**, **`char`**. Спецификатор .

7.1.2. Типы с «плавающей точкой»: **`float`**, **`double`**, **`long double`**, **`long long double`**.

7.2. ***Арифметические операции над целочисленными данными***.

7.2.1. *Одноместные*: изменение знака («одноместный минус»: **`–`**), одноместный плюс (**`+`**).

7.2.2. *Двухместные*: сложение (**`+`**), вычитание (**`–`**), умножение (**`*`**), деление нацело (**`/`**), остаток от деления нацело (**`%`**).

7.2.3. Пример программы. Дано целое число **`a`**. Найти сумму его цифр

```
#include <stdio.h>
int main() {
     int a, b, s;
     printf ("Введите число: ");
     scanf  ("%d ", &a);
     b = a;
     if (b < 0) b = -b;
     if (b = 0) s = 0;
     else {
          s = 0;
          while (b > 0) {
               s = s + b%10;
               b = b/10;
          }
     printf ("Сумма цифр числа %d равна %d\n ", a, s);
}
```

7.3. ***Арифметические операции над данными с плавающей точкой***.

7.3.1. *Одноместные*: изменение знака («одноместный минус»: **`–`**), одноместный плюс (**`+`**).

7.3.2. *Двухместные*: сложение (**`+`**), вычитание (**`–`**), умножение (**`*`**), деление (**`/`**).

7.4. Порядок выполнения арифметических операций в выражениях (*приоритет*). Самый низкий приоритет у двуместных **`+`** и **`–`**, более высокий приоритет у двуместных **`*`**, **`/`** и **`%`**, еще более высокий приоритет у одноместных **`+`** и **`–`**. В выражениях без скобок операции с более высоким приоритетом выполняются раньше. Скобки позволяют изменить порядок выполнения операций.

7.5. Вычисления с данными типа **`float`**. Нет ассоциативности и коммутативности.

7.5.1. ***Пример***. Вычисление суммы 5 чисел типа **`float`** (мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$$0.231876*10^{02} + 0.645391*10^{-03} + 0.231834*10^{-01} + 0.245383*10^{-02} + 0.945722*10^{-03} =$$

a) $0.231876*10^{02} + 0.645391*10^{-03} + 0.231834*10^{-01} + 0.245383*10^{-02} + 0.945722*10^{-03} =$ **$0.232147*10^{02}$**;

$23.1876 + 0.000645391 = 23.188245391 = 23.1882 = 0.231882*10^{02}$;

$23.1882 + 0.0231834 = 23.2113834 = 23.2114 = 0.232114*10^{02}$;

$23.2114 + 0.00245383 = 23.21385383 = 23.2138*10^{02}$;

$23.2138 + 0.000945722 = 23.214745722 = 23.2147 = 0.232147*10^{02}$;

b) $0.645391*10^{-03} + 0.9457*10^{-03} + 0.245383*10^{-02} + 0.231834*10^{-01} + 0.231876*10^{02} =$ **$0.232157*10^{02}$**;

$0.000645391 + 0.000945722 = 0.001591113 = 0.00159111 = 0.159111*10^{-02}$;

$0.00159111 + 0.00245383 = 0.00494493 = 0.494493*10^{-02}$;

$0.00494493 + 0.0231834 = 0.02812833 = 0.0281283 = 0.281283*10^{-01}$;

$0.0281283 + 23.1876 = 23.2157283 = 23.2157 = 0.232157*10^{02}$;

7.5.2. ***Пример***. Вычисление разности плавающих чисел (мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$0.238617*10^{02} - 0.238616*10^{02} + 0.645391*10^{04} - 0.645392*10^{04} + 0.845791*10^{00} - 0.835790*10^{00} =$

a) $0.238617*10^{02} - 0.238616*10^{02} + 0.645391*10^{04} - 0.645392*10^{04} + 0.845791*10^{00} - 0.835790*10^{00} =$ **$0.100000*10^{-05}$**

$0.238617*10^{02} - 0.238616*10^{02} = 23.8617 - 23.8616 = 0.0001 = 0.100000*10^{-03}$

$0.100000*10^{-03} + 0.645391*10^{04} = 0.0001 + 6453.91 = 6453.9101 = 0.645391*10^{04}$

$0.645391*10^{04} - 0.645392*10^{04} = - 0.000001*10^{04} = - 0.100000*10^{-01}$

$- 0.100000*10^{-01} + 0.845791*10^{00} = - 0.01 + 0.845791 = 0.835791 *10^{00}$

$0.835791 *10^{00} - 0.835790*10^{00} = 0.000001*10^{00} = 0.100000*10^{-05}$

b) $0.238617*10^{02} + 0.645391*10^{04} + 0.845791*10^{00} - (0.238616*10^{02} + 0.645392*10^{04} + 0.835790*10^{00}) =$ **$0.100000*10^{00}$**

$0.238617*10^{02} + 0.645391*10^{04} = 23.8617 - 6453.91 = 6478.6 = 0.647777*10^{04}$

$0.647777*10^{04} + 0.845791*10^{00} = 6477.77 + 0.845791 = 6478.615791 =$ *$0.647862*10^{04}$*

$0.238616*10^{02} + 0.645392*10^{04} = 23.8616 + 6453.92 = 6477.7816 = 6477.78*10^{04}$

$6477.78*10^{04} + 0.835790*10^{00} = 6477.78 + 0.835790 = 6478.61579 =$ *$0.647852*10^{04}$*

$0.647862*10^{04} - 0.647852*10^{04} = 0.000010*10^{04} = 0.100000*10^{-00}$

7.5.3. ***Выводы***. (1) *При вычислении суммы чисел с одинаковыми знаками* необходимо упорядочить слагаемые по возрастанию и складывать, начиная с наименьших слагаемых.

(2) *При вычислении суммы чисел с разными знаками* необходимо сначала сложить все положительные числа, потом – все отрицательные числа и в конце выполнить одно вычитание.

(3) Вычитание (сложение чисел с противоположными знаками) часто приводит к потере точности, которая у чисел с плавающей точкой определяется количеством значащих цифр в мантиссе[1] (при вычитании двух близких чисел мантисса «исчезает», что ведет к резкой потере точности). Итак, *чем меньше вычитаний, тем точнее результат*.

[1] Значащими цифрами числа с плавающей точкой называются все цифры его мантиссы за исключением нулей, стоящих в ее конце. Например, у числа **0.6700089**0000 * 10^3 все цифры, выделенные жирным шрифтом, значащие. При вычитании двух близких чисел почти все значащие цифры пропадают. Например, **0.6700089**0 * 10^3 - **0.6700088**0 * 10^3 = 0.00000010 * 10^3 = 0.10 * 10^{-4}. Таким образом, у результата всего одна значащая цифра, хотя у операндов было 7 значащих цифр. Подробнее этот вопрос освещается в курсе численного анализа.

(4) То же самое относится к умножению и делению.

7.6. ***Отношения и логические операции***.

7.6.1. *Отношения* – это операции: больше (**>**), больше или равно (**>=**), меньше (**<**), меньше или равно (**<=**). Имеют более низкий приоритет, чем сложение и вычитание (выражение **`i < lim - 1`** воспринимается как **`i < (lim - 1)`**).

7.6.2. *Операции сравнения*: равно (**==**), не равно (**!=**) имеют более низкий приоритет, чем отношения.

7.6.3. *Логические операции*: отрицание (**!**), конъюнкция (**&&**) и дизъюнкция (**||**) позволяют строить логические выражения. У одноместной операции **!** приоритет выше, чем у отношений и операций сравнения; у двухместных операций **&&** и **||** приоритет ниже, чем у отношений и операций сравнения; приоритет операции **&&** выше, чем приоритет операции **||**.

7.6.4. Результатами операций отношения и сравнения, операндами и результатами логических операций являются константы и переменные *булевского* типа (см. дополнение д7.1)

7.6.5. ***Пример***. Ввод строки символов. Фрагмент программы:

```
for (i = 0; i < lim - 1 && (c = getchar()) != '\n' && c != EOF; ++i)
     s[i] = c;
```

Замечания к примеру: (1) сначала проверка – поместится ли очередной символ в памяти, выделенной под строку (**s**); (2) если значение отношения **`i < lim - 1`** равно **false**, то значение всего логического выражения равно **false** (свойство операции **&&**) и дальнейшие проверки не нужны; (3) проверка **`(c = getchar()) != '\n'`** делается второй, так как нужен новый символ; скобки в **`(c = getchar())`** нужны потому, что операция присваивания (**=**) имеет более низкий приоритет, чем операция сравнения (**!=**); (4) **EOF** означает конец файла.

7.7. ***Символьный тип данных*** (**char**)

7.7.1. Программа подсчета числа строк во входном потоке

```
#include <stdio.h>
int main() {
     int c, nl;
     nl = 0;
     while (c = getchar())!= EOF)
          if (c == '\n')
          ++nl;
     printf ("%d\n ", nl);
}
```

7.7.2. В стандарте языка Си зафиксирован код для представления символьных данных. Символы представляются в коде *ASCII* (American Standard Code for Information Interchange). Код сопоставляет каждому символу число типа **char**, являющегося

кодом этого символа. Символы упорядочены в алфавитном порядке (отдельно для английского и русского алфавитов), к ним применимы операции отношения и сравнения. В Си-программе каждый символ, представляющий в программе самого себя, заключается в одинарные кавычки **'** и **'**, последовательность символов (строка) заключается в двойные кавычки **"** и **"**. Среди символов *ASCII* имеются английские и русские буквы (прописные и строчные), десятичные цифры, знаки препинания, знаки арифметических действий, специальные (управляющие) символы. Часть специальных символов представляются в программах на языке Си последовательностями из двух символов, начинающихся с символа ****: переход на начало новой строки (**\n**), знак табуляции (**\t**), возврат на один символ с затиранием (**\b**), двойная кавычка (**\"**), обратная косая черта (****) и др. В дополнении д7.2 такие специальные символы выделены розовым.

7.7.3. ***Важное замечание***: В коде *ASCII* буквы верхнего и нижнего регистра составляют непрерывные последовательности: между **a** и **z** (соответственно, между **A** и **Z**) нет ничего, кроме букв, расположенных в алфавитном порядке. Если бы код *ASCII* не обладал указанным свойством, нижеприведенные функции были бы невозможны.

7.7.4. Функция, преобразующая строку кодов цифр **s** в целое число (каждому символу цифры сопоставляется его код, и из полученных чисел формируется десятичное значение).

```
int atoi(char s[]) {
     int i, n;
     n = 0;
     for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
          n = 10 * n + (s[i] – '0');
     return n;
}
```

7.7.5. Классификацию, анализ и преобразование символов кода *ASCII* моно осуществить с помощью функций стандартной библиотеки **<ctype.h>** [1, с. 268].

7.8. ***Поразрядные операции***.

7.8.1. В языке Си определено 6 поразрядных операций, применимых к данным целочисленных типов. С их помощью можно ввести тип множество (для этого лучше использовать беззнаковые типы).

7.8.2. Список поразрядных операций: **&** (поразрядное И), **|** (поразрядное включающее ИЛИ), **^** (поразрядное исключающее ИЛИ)), **<<** (сдвиг влево), **>>** (сдвиг вправо), **~** (дополнение до 1, или *инверсия*). Первые 5 операций двуместные, инверсия – одноместная. Сдвиг влево заполняет освободившиеся биты нулями. Сдвиг вправо числа типа **unsigned** заполняет освободившиеся биты нулями. Сдвиг вправо числа со знаком зависит от реализации. В *GCC* освободившиеся биты заполняются значением знакового разряда («арифметический сдвиг»), однако есть реализации, в которых освободившиеся биты заполняются нулями («логический сдвиг»). Приоритет поразрядных операций **&**, **^** и **|** ниже, чем у операций сравнения, но выше, чем у двухместных логических операций (при этом приоритет операции **&** выше, чем у **^** , а приоритет операции **^** выше, чем у **|**). Приоритет операций сдвига ниже, чем у сложения и вычитания, но выше, чем у отношений. Приоритет всех одноместных операций (включая операцию **~**) выше, чем у умножения и деления. Сводную таблицу приоритетов см. [1], с. 66.

7.9. Приведение типов.

7.9.1. ***Явное приведение типов***. В языке Си определена одноместная операция *приведение типов*: (*имя типа*) *выражение*. Операция приведения типов имеет такой же приоритет, как и другие одноместные операции, т.е. выше, чем у умножения и деления.

7.9.2. ***Неявное приведение типов***. Если двуместная операция имеет операнды разных типов, то более «узкий» тип «расширяется» (преобразуется к более «широкому»), при этом область значений узкого типа остается прежней. Результат всегда принадлежит более «широкому» типу.

7.9.3. ***Правила неявного преобразования типов,*** или «*обычные арифметические преобразования*».

1) Если один из операндов имеет тип **`long double`**, то другой операнд преобразуется в **`long double`**.

2) В противном случае, если один из операндов имеет тип **`double`**, то другой операнд преобразуется в **`double`**.

3) В противном случае, если один из операндов имеет тип **`float`**, то другой операнд преобразуется в **`float`**.

4) В противном случае, к обоим операндам применяются следующие правила приведения целочисленных типов.

4.1) Если оба операнда имеют одинаковый тип, никаких преобразований не требуется.

4.2) В противном случае, если оба операнда имеют знаковый целочисленный тип, операнды более узкого типа преобразуются в более широкий тип: **`char`** и **`short`** преобразуются в **`int`**, если один из операндов имеет тип **`long`**, то и другой операнд преобразуется в **`long`**.

4.3) В противном случае, если тип операнда беззнакового целочисленного типа имеет область значений, большую либо равную области значений типа другого операнда, то тип операнда знакового целочисленного типа преобразуется к типу операнда беззнакового целочисленного типа.

4.4) В противном случае, если тип операнда знакового целочисленного типа обеспечивает представление всех значений типа операнда беззнакового целочисленного типа, то тип операнда беззнакового целочисленного типа преобразуется к типу операнда знакового целочисленного типа.

4.5) В противном случае типы обоих операндов преобразуются к типу операнда знакового целочисленного типа.

7.9.4. ***Замечание***. Числа типа **`float`** в выражениях автоматически (неявно) в **`double`** ***не преобразуются***.

7.9.5. Достаточная точность при вычислениях с плавающей точкой обеспечивается только использованием типа **`double`** (вместо **`float`**). В частности, все математические функции, объявленные в заголовочном файле **`<math.h>`**, используют тип **`double`** (см. дополнение д7.3). При использовании типа **`float`** слишком малая точность вычислений. Тип **`float`** в основном используется для экономии (памяти, времени счета).

7.9.6. При *присваивании* тип значения в правой части преобразуется (явно или неявно) к типу переменной, стоящей в левой части, и результат будет иметь именно этот тип.

7.10. ***Массивы***.

7.10.1. В примере 7.6.5 и в программе 7.7.4 использовались массивы. Массивы позволяют организовывать непрерывные последовательности нескольких однотипных элементов и обращаться к ним по номеру (индексу).

7.10.2. ***Пример***. Программа, подсчитывающая количество вхождений в строку (текст) каждой из десяти цифр, пробельных символов и остальных символов. Необходимо вычислить 12 сумм, для каждой суммы нужна переменная, в которой она будет накапливаться. Используя массив из 10 элементов, можно сократить число переменных до 3: **nwhite**, **nother** и **ndigit[10]**

```
#include <stdio.h>
int main() {
     int c, i, nwhite, nother, ndigit[10];
     nwhite = 0;
     nother = 0;
     for (i = 0; i < 10; ++i)
          ndigit[i] = 0;
     while (c = getchar())!= EOF)
          if (c >= '0' && c <= '9')
               ++ndigit[c – '0'];
          else if (c == ' ' || c == '\n' || c == '\t')
               ++nwhite;
          else
               ++nother;
     printf ("digits = ");
     for (i = 0; i < 10; ++i)
          printf ("%d\n ", ndigit[i]);
     printf (", white space = %d, other = %d\n", nwhite,
     nother);
}
```

7.10.3. Чтобы обратиться к элементу массива, достаточно написать имя массива и в квадратных скобках указать номер (*индекс*) этого элемента. Индексы массива меняются не с 1, а с 0. При объявлении массива указывается его длина. Массив **ndigit[10]** содержит элементы **ndigit[0], ..., ndigit[9]**

7.11. ***Строки*** – это одномерные массивы типа **char**, заканчивающиеся нулевым символом '**\0**' (конец строки). Объявляя массив символов, предназначенный для хранения строки, необходимо предусмотреть место для '**\0**', т.е. указать его размер на 1 символ больше, чем наибольшее предполагаемое количество символов в строке.

7.11.1. Записанная в тексте программы строка символов, заключенная в двойные кавычки, является *строковой константой* (например, **"строка"**). В конец строковой константы компилятор автоматически добавляет '**\0**'.

7.11.2. В язык Си включена стандартная библиотека функций работы со строками **<string.h>** (см. [1], с. 269). В частности, она содержит такие функции, как:

strcpy(s1, s2) (копирование **s2** в **s1**),

strcat(s1, s2) (конкатенация **s2** и **s1**),

strlen(s) (длина строки **s**),

strcmp(s1, s2) (сравнение строк **s2** и **s1** в лексикографическом порядке: возвращается 0, если строки совпадают, отрицательное значение, если **s1 < s2** и положительное значение, если **s1 > s2**),

strchr(s, ch) (возвращается указатель на первое вхождение символа **ch** в строку **s**),

strstr(s1, s2) (возвращается указатель на первое вхождение подстроки **s2** в строку **s1**)

7.11.3. ***Пример*** применения функций работы со строками:

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

int main() {
    char string1[80], string2[80], smp[3];

    gets (string1);
    gets (string2);
    printf("Строки имеют длину: первая %d вторая %d\n,
         strlen(string1), strlen(string2));

    if(!strcmp(string1, string2) printf("строки равны\n");

    strcat(string1, string2);
    printf("%s\n", string1);

    strcpy(string1, "Привет, ");
    printf("%s\n", string1);

    if(strchr(string1, 'и') printf("Буква и есть в %s\n",
         string1);
    smp[0] = 'и';
    smp[1] = 'в';
    if(strstr(string1, smp)) printf("Есть %s\n", smp);
    return 0;
}
```

Если эту программу выполнить, введя в **string1** строку **"Здравствуй, "** , а в **string2** – строку **"мир!"**, то на экран будет выведено:

```
Строки имеют длину 12 4
Здравствуй, мир!
Привет,
Буква и есть в Привет
Есть ив
```

Дополнение д7.1 Данные булевского типа.

Д7.1.1. В стандарте *ANSI C'99* определен тип **_bool**. **sizeof(_bool) = 1**, т.е. данные типа **_bool** занимают один байт (как и данные типа **char**). Определено две константы типа **_bool**: **true** (эквивалентно значению **1**) и **false** (эквивалентно значению **0**). В выражениях можно использовать как **true** и **false** , так и эквивалентные им **1** и **0**.

Д7.1.2. Если не нравится писать **_bool**, можно писать **bool**, но в этом случае к программе необходимо присоединить заголовочный файл **<stdbool.h>**: **#include <stdbool.h>**

Д7.1.3. В качестве булевских можно использовать и значения типа **int** и других целочисленных типов, помня, что компилятор переведет их (автоматически) в тип **_bool**, причем все ненулевые значения будут переводиться в **1**, а нулевые – в **0**.

Д7.1.4. Подробности см. в стандарте ANSI C'99 [2]

Дополнение д7.2 Код ASCII (таблица)

С сайта http://ru.wikipedia.org/wiki/ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	
0.	\0	SOH	STX	ETX	EOT	ENQ	ACK	\a	\b	\t	\n	\v	\f	\r	SO	SI	0.
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	1.
2.		!	"	#	$	%	&	'	(	)	*	+	,	–	.	/	2.
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	3.
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	4.
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	5.
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	6.
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	7.
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	

На оранжевом и желтом фоне – прописные и строчные латинские буквы, на голубом – десятичные цифры, на белом – различные вспомогательные символы (знаки пунктуации и т.п.). Символы на черном фоне в программах на языке Си используются только через свои коды (см. ниже).
Некоторые вспомогательные символы можно задавать в виде символьных констант. Например: **\?** (знак вопроса), **** (обратный слеш), **\"** (двойная кавычка), **\'** (одинарная кавычка). Код *ASCII* произвольного символа можно задать в виде **\ooo** (**ooo** – трехзначный восьмеричный код этого символа) или **\xnn** (**nn** – двухзначный шестнадцатеричный код этого символа). Двухзначный шестнадцатеричный код символа по таблице получается путем конкатенации номера строки вышеприведенной таблицы и номера столбца (в указанном порядке). Например, код символа @ равен 40_{16}. Трехзначный восьмеричный код символа получается аналогично, но столбцы нумеруются двухзначными восьмеричными числами: 00, 01, 02, 03, 04, 05, 06, 07, 10, 11, 12, 13, 14, 15. В этих обозначениях код символа @ равен 400_8.
Как видно, использовано меньше 128 из 256 возможных символов, так что есть много места для новых символов (например, для русских букв). К сожалению, расширение кода *ASCII* буквами

кириллических алфавитов, названное *КОИ-8* (код обмена информацией, 8 битов), до сих пор не стандартизовано. Существует несколько слабо отличающихся одна от другой версий *КОИ-8*.
В последнее время в системы программирования (компиляторы) языка Си внедряется *Unicode*. *Unicode* – это международный 16-разрядный код (символ в *Unicode* занимает не один, а два байта). В *Unicode* представлены не только латинские и кириллические символы, но и символы всех известных алфавитов и множество специальных символов. У компилятора есть опция, позволяющая использовать *Unicode* (а следовательно, русские идентификаторы, а также символьные и строковые константы). При использовании *Unicode* код произвольного символа можно задается в виде: **\uhhhh** (**hhhh** –четырехзначный шестнадцатеричный код этого символа).
К сожалению, *Unicode* содержит слишком много символов, и представить его в виде компактной таблицы не удается (таблица для *Unicode* содержала бы 256 столбцов и столько же строк). Для русских символов сейчас используется кодировка **ISO/IEC 8859-5:1999** (таблица взята с сайта http://en.wikipedia.org/wiki/ISO/IEC_8859-5):

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	
0.	\0	SOH	STX	ETX	EOT	ENQ	ACK	\a	\b	\t	\n	\v	\f	\r	SO	SI	0.
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	1.
2.		!	"	#	$	%	&	'	(	)	*	+	,	–	.	/	2.
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	3.
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	4.
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	5.
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	6.
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	7.
8.																	8.
9.																	9.
A.		Ё															A.
B.	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	B.
C.	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	C.
D.	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	D.
E.	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	E.
F.	№	ё															F.
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F	

Фиолетовая часть таблицы представляет собой копию предыдущей таблицы и содержит коды латинских букв, знаков пунктуации и управляющих символов. В нижней части приведены коды русских прописных (на оранжевом фоне) и строчных (на желтом фоне) букв и символа №, отсутствующего в латинском коде. Голубым фоном выделены позиции в которых помещены украинские, белорусские, сербские, болгарские буквы, не совпадающие с русскими.

Дополнение д7.3 Математические функции (заголовочный файл **<math.h>**)

Д7.3.1. В таблице, приводимой ниже, параметры **x** и **y** имеют тип **double**, **n** – тип **int**, все функции возвращают значения типа **double**. Таблица есть в [1, с. 270-271].

sin(x)	Синус **x**
cos(x)	Косинус **x**
tan(x)	Тангенс **x**
asin(x)	Арксинус **x**
acos(x)	Арккосинус **x**
atan(x)	Арктангенс **x**
atan2(y, x)	Арктангенс **y/x**
sinh(x)	Синус гиперболический **x**
cosh(x)	Косинус гиперболический **x**
tanh(x)	Тангенс гиперболический **x**
exp(x)	Экспонента ($\mathbf{e}^{\mathbf{x}}$)
log(x)	Логарифм по основанию 2
log10(x)	Логарифм по основанию 10
pow(x,y)	$\mathbf{x}^{\mathbf{y}}$
sqrt(x)	$\sqrt{x}$
cell(x)	«Потолок»: ближайшее целое число ≥ **x** ($\lceil x \rceil$).
floor(x)	«Пол»: ближайшее целое число ≤ **x** ($\lfloor x \rfloor$).
fabs(x)	**|x|**
ldexp(x, n)	$\mathbf{x} \cdot \mathbf{2}^{\mathbf{n}}$
frexp(x, int *exp)	Переводит **x** в «нормализованное» представление, т.е. разбивает **x** на два множителя: мантиссу (нормализованную дробь в интервале ½ ≤ **x** < 1) и порядок (целочисленный показатель степени двойки); мантисса является результатом функции, а порядок помещается в ***exp**. Если = 0, обе части результата равны 0.
modf(x, double *ip)	Разбивает **x** на целую (помещается в ***ip**) и дробную (результат функции) части.
fmod(x, y)	Остаток от деления **x** на **y** в виде вещественного числа. Знак результата совпадает со знаком **x**. Если **y** = 0, результат зависит от реализации языка.

Литература

[1] Б. Керниган и Д. Ритчи. Язык программирования Си. Издание второе. Издательский дом «Вильямс», Москва, Санкт-Петербург, Киев – 2010.

[2] Стандарт ANSI C 99 (в Интернете доступны черновики (draft), сам Стандарт можно приобрести в …). http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf (один из черновиков).