

資料

初めてのC言語入門(西東社)から抜粋

• C言語の主な標準関数

C言語で用意されている、主な標準関数をまとめておきます。

主なヘッダファイル	
<string.h>	文字列関数
<stdio.h>	標準入出力関数
<ctype.h>	文字クラステスト
<math.h>	数学関数
<stdlib.h>	数値変換と記憶割り当て
<assert.h>	プログラムに診断機能を付加する
<float.h>	浮動小数点演算に関する定数を定義
<stdarg.h>	可変引数リスト
<limits.h>	整数型のサイズを表す定数を定義
<setjmp.h>	非局所的ジャンプ
<signal.h>	外部機器からの割り込みや実行エラーを処理
<time.h>	日付と時刻を扱う
<dos.h>	MS-DOSに関連する定義を行う
<errno.h>	エラーコードを定義する

<string.h>文字列関数 (saは文字型変数、sbは文字型変数または定数、nはバイト数)		
種 類	機 能	使用 例
char *strcpy(sa, sb)	文字列をコピー	<pre>char *s; strcpy(s, "Hello"); printf("%s", s); //Helloと表示される</pre>
char *strcat(sa, sb)	文字列をつなぐ	<pre>char *a = "Good"; char *b = "Morning"; strcat(a, b); printf("%s", a); //GoodMorningと表示される</pre>
char *strncat(sa, sb, n)	sbからn文字をsaの終わりにつなぐ	<pre>char *a = "Good"; char *b = "byname"; strncat(a, b, 2); printf("%s", a); //Goodbyと表示される</pre>
int strcmp(ca, cb)	caとcbのASCIIコードを比べる ca > cb → > 0 ca = cb → = 0 ca < cb → < 0	<pre>char *a, *b; scanf("%s", a); scanf("%s", b); if(strcmp(a, b) == 0); //同じ文字が入力されるとif文の中身が実行される</pre>

<stdio.h>標準入出力関数			
種類	書式	機能	使用例
1.書式付き入出力関数(返値はint型)			
printf	printf("書式", 変数, ……)	指定された書式で、数字や文字を標準出力装置(画面)へ出力する	printf("a = %f, b = %f", a, b); ……画面に変数a、bの値を表示する
fprintf	fprintf(fp, "書式", 変数, ……)	指定された書式で、数字や文字をファイルへ出力する	fprintf(fp, "%s", s); ……fpで指し示すファイルへ配列sの内容を出力する
sprintf	sprintf(配列, "書式", 変数, ……)	別の文字列へ、書式を変えてコピーする	sprintf(s, "%d, %d", a, b); ……整数値a、bを文字型に変換して文字配列sに入れる
scanf	scanf("書式", ポインタ, ……)	標準入力装置(キーボード)から指定された書式で、文字や数字を入力する	scanf("%c%c", &a, &b); ……変数a、bにキーボードから1文字ずつ入力する
fscanf	fscanf(fp, "書式", ポインタ……)	ファイルから指定された書式で、文字や数字を入力する	fscanf(fp, "%d%d", &a, &b); ……fpで指し示すファイルから変数a、bへ整数値を入力する
2.文字入出力関数(返値はint型—但し、fgetc、getsは文字型へのポインタで「char *fgetc」「char *gets」と宣言する)			
fgetc	変数=fgetc(ファイルポインタ)	ファイルから1文字入力する	a=fgetc(fp); ……fpで指し示すファイルから1文字入力し、aに代入する
fgets	fgets(文字型配列、個数、ファイルポインタ)	ファイルから最大<n-1>文字を入力する。最後に「¥0」を追加	fgets(s,n,fp); ……fpから文字配列sにn-1文字入力する
fputc	fputc(文字型変数、ファイルポインタ)	ファイルへ1文字出力する	fputc(a,fp); ……fpで指し示すファイルへaの中身を1文字出力する
fputs	fputs(文字型配列、ファイルポインタ)	ファイルへ文字列を出力する	fputs(s,fp); ……fpへ、文字配列sの内容を出力する

種 類	書 式	機 能	使用 例
getc	変 数 = getc(ファイルポインタ)	指定した入力装置から1文字入力する	a=getc(fp); … … fpで指し示すファイルから1文字入力してaに代入する
getchar	変 数 = getchar()	標準入力装置(キーボード)から1文字入力する	a=getchar(); … … キーボードから入力した文字を変数aに代入する
gets	gets(文字型変数)	標準入力装置(キーボード)から文字列を入力する	gets(s); … … キーボードから1行ぶん文字を入力して配列sに入れる
putc	putc(文字型変数、ファイルポインタ)	指定した装置へ1文字出力する	putc(a,fp); … … fpで指し示すファイルへ変数aの内容を1文字出力する
putchar	putchar(文字型変数)	標準出力装置(画面)へ1文字出力する	putchar(a); … … 画面へ変数aの内容を1文字出力する
puts	puts(文字型配列)	標準出力装置(画面)へ文字列を出力する	puts(s); … … 画面へ文字配列sの内容を出力する

<ctype.h>文字クラステスト(cはint型、返値はすべてint型)		
種 類	機 能	使用 例
tolower(c)	小文字に変換する	int x, y; x = 'A'; y = tolower(x); //yには小文字のaが入る
toupper(c)	大文字に変換する	int x, y; x = 'a'; y = toupper(x); //xには大文字のAが入る
isdigit(c)	10進数の判定	int a; a = '9'; if(isdigit(a) == 0) puts("aは10進数ではない");
isprint(c)	印刷可能文字 (スペースを含む)	int a; a = '¥0'; if(isprint(a) == 0) a = '.';

<math.h>数学関数 (x、yはdouble型、返値はすべてdouble型)	
種 類	機 能
sin(x)	xの正弦関数 $\sin x$
cos(x)	xの余弦関数 $\cos x$
tan(x)	xの正接関数 $\tan x$
asin(x)	xの逆正弦関数 $\sin^{-1} x$
acos(x)	xの逆余弦関数 $\cos^{-1} x$
atan(x)	xの逆正接関数 $\tan^{-1} x$
sinh(x)	x双曲線正弦関数 $\sinh x$
cosh(x)	x双曲線余弦関数 $\cosh x$
tanh(x)	x双曲線正接関数 $\tanh x$
exp(x)	指数対数 e^x
log(x)	自然対数 $\ln x$
log10(x)	対数関数 $\log_{10} x$
pow(x,y)	べき乗 x^y $x>0$,または $x<0$ で y =整数のとき
sqrt(x)	平方根 $\sqrt{x}$
fabs(x)	絶対値 $ x $
fmod(x,y)	剰余 x を y で割った余り

<stdlib.h>数値変換と記憶割り当て (sは文字列定数へのポインタ)	
種 類	機 能
double atof(s)	文字列sをdouble型に変換
int atoi(s)	文字列sをint型に変換
long atol(s)	文字列sをlong型に変換
int rand()	乱数を返す
void *calloc(n,m)	<mバイト×n個>の領域を割り当てて初期化する
void *malloc(n)	nバイトの大きさの領域を割り当てる
void free(void *p)	ポインタpが指す領域を解放する
void abort()	プログラムを異常終了する
void exit(const)	プログラムを終了する
int abs(int n)	int型の変数nの絶対値
long labs(long n)	long型の変数nの絶対値

・エスケープ符号列

プログラムの中で「¥0」「¥n」「¥t」のように「¥」記号が付けられた制御文字が出てきました。この「¥」記号は日本だけのもので、同じ符号がアメリカでは「\」（バックスラッシュ）に対応しているものですが、制御文字は、実はそのそれぞれが1バイト文字コードで書き表すC言語特有のものです。「¥」の付いた制御文字はエスケープ符号列（またはエスケープシーケンス）と総称され、次の表にあるのがそのすべてです。

種類	意味	機能
¥a	警告（ベル）	ベルを鳴らす
¥b	バックスペース	1文字文左へ
¥f	改ページ	ページを変える
¥n	改行	次の行の先頭へ
¥r	復帰	同じ行の先頭へ
¥t	水平タブ	水平方向のタブ
¥v	垂直タブ	垂直方向のタブ
¥¥	「¥」を示す	-
¥?	「?」を示す	-
¥'	「'」を示す	-
¥"	「"」を示す	-
¥ooo	8進数	¥123=8進数123
¥xhh	16進数	¥xhh=16進数ff

「¥000」は任意の1バイトのビットパターンを表し、「¥n」に続いて「ooo」の部分に3桁以内の8進数（0～7）がかかれ、「¥0」はこのうちの一例です。「¥xhh」は同じく任意の1バイトのビットパターンを16進数で表す場合の書きかたで、「¥x」に続いて「hh」の部分に2桁の0～9、a～f（またはA～F）が入ります。

・演算子の優先順位 プログラミング講義C++(ソフトバンクパブリッシング)より抜粋

優先度	演算子	形式	名称	名称(英名)	結合規則
1	::	::x	スコープ解決演算子	scope resolution operator	—
	::	x::y	スコープ解決演算子		—
2	.	x . y	ドット演算子	. operator	左
	->	x -> y	アロー演算子	-> operator	左
	[]	x [y]	添え字演算子	array subscript operator	左
	()	x (y)	関数呼び出し演算子	function call operator	左
	++	x ++	後置インクリメント演算子	postfix increment operator	左
	--	x --	後置デクリメント演算子	postfix decrement operator	左
3	sizeof	sizeof x	sizeof 演算子	sizeof operator	右
	++	++ x	前置インクリメント演算子	prefix increment operator	右
	--	-- x	前置デクリメント演算子	prefix decrement operator	右
	~	~ x	補数演算子	complement operator	右
	!	! x	論理否定演算子	logical negation operator	右
	-	- x	単項－演算子	unary - operator	右
	+	+ x	単項＋演算子	unary + operator	右
	&	& x	アドレス演算子	address operator	右
	*	* x	間接演算子	indirection operator	右
	new	new x	new 演算子	new operator	右
	delete	delete x	delete 演算子	delete operator	右
	delete []	delete [] x	delete [] 演算子	delete [] operator	右
	()	x (y)	キャスト演算子	cast operator	右
4	()	(x) y	キャスト演算子	cast operator	右
5	.*	x .* y	間接ドット演算子	indirection .* operator	左
	->*	x -> * y	間接アロー演算子	indirection -> * operator	左
6	*	x * y	2 項 * 演算子	binary * operator	左
	/	x / y	2 項 / 演算子	binary / operator	左
	%	x & y	2 項 % 演算子	binary % operator	左
7	+	x + y	2 項 + 演算子	binary + operator	左
	-	x - y	2 項 - 演算子	binary / operator	左
8	<<	x << y	左シフト演算子	left-shift operator	左
	>>	x >> y	右シフト演算子	right-shift operator	左
9	<	x < y	< 演算子	< operator	左
	<=	x <= y	< = 演算子	< = operator	左
	>	x > y	> 演算子	> operator	左
	>=	x >= y	> = 演算子	> = operator	左
10	==	x == y	= = 演算子	== operator	左

	!=	x != y	!= 演算子	!= operator	左
11	&	x & y	ビット AND 演算子	bitwise AND operator	左
12	^	x ^ y	ビット排他 OR 演算子	bitwise exclusive OR operator	左
13		x y	ビット OR 演算子	bitwise OR operator	左
14	&&	x && y	論理 AND 演算子	logical AND operator	左
15		x y	論理 OR 演算子	logical OR operator	左
16	?:	x ? y : z	条件演算子	conditional operator	左
17	=	x = y	単純代入演算子	simple assignment operator	右
	*=	x *= y	* = 演算子	*= operator	右
	/=	x /= y	/ = 演算子	/= operator	右
	%=	x %= y	% = 演算子	%= operator	右
	+=	x += y	+ = 演算子	+= operator	右
	-=	x -= y	- = 演算子	-= operator	右
	<<=	x <<= y	<< = 演算子	<<= operator	右
	>>=	x >>= y	>> = 演算子	>>= operator	右
	&=	x &= y	& = 演算子	&= operator	右
	=	x = y	= 演算子	= operator	右
	^=	x ^= y	^ = 演算子	^= operator	右
18	,	x , y	カンマ演算子	comma operator	左

優先度と結合法則

演算子の一覧表は上側に行くほど優先度が高くなるように表記しています。例えば、乗除を行う*と/が、加減を行う+や-より優先度が高いのは、我々が実生活で使用する数学の規則と同じです。したがって、

$$a + b * c$$

は $a + (b * c)$ と解釈されるのであり、 $(a + b) * c$ ではありません。すなわち+のほうが先に書かれているにも関わらず、*の演算が優先されます。

結合法則については説明が必要ですね。たとえば2つのオペランドを要求する 2 項演算子を○と表した場合、 $a \circ b \circ c$ を

$$(a \circ b) \circ c \quad \text{左結合}$$

とみなすのが左結合演算子であり

$$a \circ (b \circ c) \quad \text{右結合}$$

とみなすのが右結合の演算子です。すなわち、同じ優先度の演算子が連続するときに、左右どちらの演算を先に行うかを示すのが結合法則です。たとえば、減算を行う 2 項 - 演算子は左結合ですから

$$5 - 3 - 1 \rightarrow (5 - 3) - 1 \quad // \text{左結合}$$

です。もし右結合だったら、 $5 - (3 - 1)$ と解釈され、演算結果も違うものとなってしまいます。代入を行う単純演算子 = は右結合ですから

$$a = b = a \rightarrow a = (b = 1) \quad // \text{右結合}$$

となります。