Tarjeta de referencia ANSI C

Estructura de programa/funciones

tipo func(*tipo*$_1$,...)	declaración de funciones
tipo nombre	declaración de variables globales
`main() {`	función principal
declaraciones	declaración de variables locales
instrucciones	
`}`	
tipo func(*arg*$_1$,...) {	definición de función
declaraciones	declaración de variables locales
instrucciones	
`return` *valor*;	
`}`	
`/* */`	comentarios
`main(int argc, char *argv[])`	programa con argumentos

Preprocesador de C

incluir fichero de cabeceras	`#include <`*fichero*`>`
incluir fichero de usuario	`#include "`*fichero*`"`
sustitución de texto	`#define` *nombre texto*
macro con argumentos	`#define` *nombre*(*var*) *texto*
Ejemplo. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
anular definición	`#undef` *nombre*
entrecomillar al reemplazar	`#`
concatenar argumentos y reescanear	`##`
compilación condicional	`#if, #else, #elif, #endif`
¿*nombre* definido, no definido?	`#ifdef, #ifndef`
¿*nombre* definido?	`defined(`*nombre*`)`
carácter de continuación de línea	`\`

Tipos de datos. Declaraciones

carácter (1 byte)	`char`
entero	`int`
real (precisión simple)	`float`
real (precisión doble)	`double`
corto (entero de 16 bits)	`short`
largo (entero de 32 bits)	`long`
positivo y negativo	`signed`
sólo positivo	`unsigned`
puntero a `int`, `float`,...	`*int, *float,...`
enumeración	`enum`
valor constante (inalterable)	`const`
declaración de variable externa	`extern`
variable registro	`register`
variable estática	`static`
sin tipo	`void`
estructura	`struct`
crear un tipo de datos	`typedef` *tipo nombre*
talla de un objeto (devuelve un `size_t`)	`sizeof` *objeto*
talla de un tipo de datos (dev. un `size_t`)	`sizeof(`*tipo*`)`

Inicialización

Inicializar variable	*tipo nombre*`=`*valor*
Inicializar vector	*tipo nombre*`[]={`*valor*$_1$`,...}`
Inicializar cadena	`char` *nombre*`[]="`*cadena*`"`

Constantes

largo (sufijo)	`L` o `l`
real de precisión simple (sufijo)	`F` o `f`
notación científica	`E` o `e`
octal (prefijo cero)	`0`
hexadecimal (prefijo cero-equis)	`0x` o `0X`
carácter constante (char, octal, hex.)	`'a', '\`*ooo*`', '\x`*hh*`'`
nueva línea, ret. de carro, tab., borrado	`\n, \r, \t, \b`
caracteres especiales	`\\, \?, \', \"`
cadena constante (termina con `'\0'`)	`"abc...de"`

Punteros, vectores y estructuras

declarar un puntero a *tipo*	*tipo* `*`*nombre*
decl. una func. que dev. un punt. a *tipo*	*tipo* `*f()`
decl. un punt. a func. que devuelve *tipo*	*tipo* `(*pf)()`
puntero genérico	`void *`
valor de puntero a nulo	`NULL`
objeto apuntado por *puntero*	`*`*puntero*
dirección del objeto *nombre*	`&`*nombre*
vector	*nombre*`[`*dim*`]`
vector multidimensional	*nombre*`[`*dim*$_1$`][`*dim*$_2$`]...`

Estructuras

`struct` *etiqueta* `{`	plantilla de estructura
declaraciones	declaración de campos
`};`	
crear estructura	`struct` *etiqueta nombre*
campo de estructura	*nombre*`.`*campo*
campo de estructura a través de puntero	*puntero*`->`*campo*
Ejemplo. `(*p).x` y `p->x` son lo mismo	
estructura múltiple, valor único	`union`
campo de bits con *b* bits	*campo* `:` *b*

Operadores (según precedencia)

acceso a campo de estructura	*nombre*`.`*campo*
acceso por puntero	*puntero*`->`*campo*
acceso a elemento de vector	*nombre*`[`*índice*`]`
incremento, decremento	`++, --`
más, menos, no lógico, negación bit a bit	`+, -, !, ~`
acceso por puntero, direcc. de objeto	`*`*puntero*`, &`*nombre*
convertir tipo de expresión	(*tipo*) *expr*
tamaño de un objeto	`sizeof`
producto, división, módulo (resto)	`*, /, %`
suma, resta	`+, -`
desplazamiento a izda., dcha. (bit a bit)	`<<, >>`
comparaciones	`>, >=, <, <=`
comparaciones	`==, !=`
"Y" bit a bit	`&`
"O exclusiva" bit a bit	`^`
"O" bit a bit	`|`
"Y" lógico	`&&`
"O" lógico	`||`
expresión condicional	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
operadores de asignación	`=, +=, -=, *=, ...`
separador de evaluación de expresiones	`,`

Los operadores unarios, expresión condicional y operadores de asignación se agrupan de dcha. a izda.; todos los demás de izda. a dcha.

Control de flujo

finalizador de instrucción	`;`
delimitadores de bloque	`{ }`
salir de `switch`, `while`, `do`, `for`	`break`
siguiente iteración de `while`, `do`, `for`	`continue`
ir a	`goto` *etiqueta*
etiqueta	*etiqueta*`:`
valor de retorno de función	`return` *expr*

Construcciones de flujo

instrucción `if`	`if (`*expr*`)` *instrucción* `else if (`*expr*`)` *instrucción* `else` *instrucción*
instrucción `while`	`while (`*expr*`)` *instrucción*
instrucción `for`	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *instrucción*
instrucción `do`	`do` *instrucción* `while(`*expr*`);`

instrucción `switch`	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *instrucción*$_1$ `break;`
	`case` *const*$_2$`:` *instrucción*$_2$ `break;`
	`default:` *instrucción*
	`}`

Bibliotecas ANSI estándar

`<assert.h>` `<ctype.h>` `<errno.h>` `<float.h>` `<limits.h>`
`<locale.h>` `<math.h>` `<setjmp.h>` `<signal.h>` `<stdarg.h>`
`<stddef.h>` `<stdio.h>` `<stdlib.h>` `<string.h>` `<time.h>`

Consulta de tipos de carácter `<ctype.h>`

`c` es un carácter

¿alfanumérico?	`isalnum(c)`
¿alfabético?	`isalpha(c)`
¿carácter de control?	`iscntrl(c)`
¿dígito decimal?	`isdigit(c)`
¿carácter imprimible (excluído espacio)?	`isgraph(c)`
¿letra minúscula?	`islower(c)`
¿carácter imprimible (incl. espacio)?	`isprint(c)`
¿car. impr. excepto espacio, letra, dígito?	`ispunct(c)`
¿separador?	`isspace(c)`
¿letra mayúscula?	`isupper(c)`
¿dígito hexadecimal?	`isxdigit(c)`
convertir a minúscula	`tolower(c)`
convertir a mayúscula	`toupper(c)`

Operaciones con cadenas `<string.h>`

`s`,`t` son cadenas, `cs`,`ct` son cadenas constantes

longitud de `s`	`strlen(s)`
copiar `ct` en `s`	`strcpy(s,ct)`
...hasta `n` caracteres	`strncpy(s,ct,n)`
concatenar `ct` tras `s`	`strcat(s,ct)`
...hasta `n` caracteres	`strncat(s,ct,n)`
comparar `cs` con `ct`	`strcmp(cs,ct)`
...sólo los primeros `n` caracteres	`strncmp(cs,ct,n)`
puntero al primer `c` en `cs`	`strchr(cs,c)`
puntero al último `c` en `cs`	`strrchr(cs,c)`
copiar `n` caracteres de `ct` en `s`	`memcpy(s,ct,n)`
copiar `n` cars. de `ct` en `s` (sobreescribe)	`memmove(s,ct,n)`
comparar `n` caracteres de `cs` con `ct`	`memcmp(cs,ct,n)`
punt. al 1er `c` en los `n` 1os cars. de `cs`	`memchr(cs,c,n)`
poner `c` en los `n` primeros cars. de `cs`	`memset(s,c,n)`

Tarjeta de referencia ANSI C

Entrada/Salida <stdio.h>

E/S estándar

flujo de entrada estándar	stdin
flujo de salida estándar	stdout
flujo de error estándar	stderr
final de fichero	EOF
obtener un carácter	getchar()
imprimir un carácter	putchar(*car*)
imprimir con formato	printf("*formato*",arg_1,...)
imprimir en cadena s	sprintf(s,"*formato*",arg_1,...)
leer con formato	scanf("*formato*",&$nombre_1$,...)
leer de cadena s	sscanf(s,"*formato*",&$nombre_1$,...)
leer línea en cadena s	gets(s)
imprimir cadena s	puts(s)

E/S de ficheros

declarar puntero a fichero	FILE *fp
obtener puntero a fichero	fopen("*nombre*","*mode*")
modos: r (leer), w (escribir), a (añadir)	
obtener un carácter	getc(*fp*)
escribir un carácter	putc(*car*,*fp*)
escribir en fichero	fprintf(*fp*,"*formato*",arg_1,...)
leer de fichero	fscanf(*fp*,"*formato*",arg_1,...)
cerrar fichero	fclose(*fp*)
distinto de cero si error	ferror(*fp*)
distinto de cero si EOF	feof(*fp*)
leer línea en cadena s (< max cars.)	fgets(s,max,*fp*)
escribir cadena s	fputs(s,*fp*)

Códigos de E/S con formato: "%-+ 0*w.pmc*"

-	alineación a izquierda
+	imprimir con signo
space	imprimir espacio si no hay signo
0	rellenar por delante con ceros
w	anchura mínima del campo
p	precisión
m	carácter de conversión: h short, l long, L long double
c	carácter de conversión:
d,i	entero
u	sin signo
c	carácter
s	cadena de caracteres
f	doble
e,E	exponencial
o	octal
x,X	hexadecimal
p	puntero
n	número de caracteres escritos
g,G	como f o e,E según cuál sea el exponente

Lista variable de argumentos <stdarg.h>

declarar puntero a argumentos	va_list *nombre*;
inicializar puntero a args.	va_start(*nombre*,*ultarg*)
ultarg es el último parámetro con nombre de la función	
siguiente arg. sin nom., actualizar punt.	va_arg(*nombre*,*tipo*)
invocar antes de salir de la función	va_end(*nombre*)

Funciones útiles <stdlib.h>

valor absoluto del entero n	abs(n)
valor absoluto del largo n	labs(n)
cociente y resto de enteros n,d	div(n,d)
devuelve una estructura con div_t.quot y div_t.rem	
cociente y resto de largos n,d	ldiv(n,d)
devuelve una estructura con ldiv_t.quot y ldiv_t.rem	
entero pseudo-aleatorio en [0,RAND_MAX]	rand()
fijar la semilla aleatoria a n	srand(n)
finalizar ejecución del programa	exit(estado)
ejecutar cadena s en el sistema	system(s)

Conversiones

convertir cadena s a double	atof(s)
convertir cadena s a int	atoi(s)
convertir cadena s a long	atol(s)
convertir prefijo de s a double	strtod(s,finp)
convertir prefijo de s (base b) a long	strtol(s,finp,b)
igual, pero unsigned long	strtoul(s,finp,b)

Reserva de memoria

reserva memoria	malloc(talla), calloc(nobj,talla)
cambiar tamaño de la reserva	realloc(pts,talla)
liberar memoria	free(ptr)

Funciones de vectores

buscar clave en vect	bsearch(clave,vect,n,talla,cmp())
ordenar vect ascendentemente	qsort(vect,n,talla,cmp())

Funciones de hora y fecha <time.h>

tiempo de proc. usado por el programa	clock()
Ejemplo. clock()/CLOCKS_PER_SEC da el tiempo en segundos	
segundos desde 1/1/1.970 (hora de ref.)	time()
tpo_2-tpo_1 en segs. (double)	difftime(tpo_2,tpo_1)
tipos numéricos para representar horas	clock_t,time_t
estructura estándar usada para fecha y hora	tm

tm_sec	segundos en el minuto
tm_min	minutos en la hora
tm_hour	horas desde medianoche
tm_mday	día del mes
tm_mon	meses desde enero
tm_year	años desde 1.900
tm_wday	días desde el domingo
tm_yday	días desde el 1 de enero
tm_isdst	indicador del cambio de horario (verano/invierno)

convertir hora local a hora de ref.	mktime(tp)
convertir hora en tp a cadena	asctime(tp)
convertir hora de ref. en tp a cadena	ctime(tp)
convertir hora de ref. a GMT	gmtime(tp)
convertir hora de ref. a hora local	localtime(tp)
formatear fecha y hora	strftime(s,smax,"*formato*",tp)
tp es un puntero a una estructura de tipo tm	

Funciones matemáticas <math.h>

los argumentos y valores devueltos son double

funciones trigonométricas	sin(x), cos(x), tan(x)
funciones trig. inversas	asin(x), acos(x), atan(x)
arctg(y/x)	atan2(y,x)
funciones trig. hiperbólicas	sinh(x), cosh(x), tanh(x)
exponenciales y logaritmos	exp(x), log(x), log10(x)
exps. y logs. (base 2)	ldexp(x,n), frexp(x,*e)
división y resto	modf(x,*ip), fmod(x,y)
potencia y raíz	pow(x,y), sqrt(x)
redondeo	ceil(x), floor(x), fabs(x)

Límites del tipo entero <limits.h>

límites típicos para un sistema Unix de 32 bits

CHAR_BIT	bits en char	(8)
CHAR_MAX	máximo valor de char	(127 o 255)
CHAR_MIN	mínimo valor de char	(-128 o 0)
INT_MAX	máximo valor de int	(+32767)
INT_MIN	mínimo valor de int	(-32768)
LONG_MAX	máximo valor de long	(+2147483647)
LONG_MIN	mínimo valor de long	(-2147483648)
SCHAR_MAX	máximo valor de signed char	(+127)
SCHAR_MIN	mínimo valor de signed char	(-128)
SHRT_MAX	máximo valor de short	(+32767)
SHRT_MIN	mínimo valor de short	(-32768)
UCHAR_MAX	máximo valor de unsigned char	(255)
UINT_MAX	máximo valor de unsigned int	(65535)
ULONG_MAX	máximo valor de unsigned long	(4294967295)
USHRT_MAX	máximo valor de unsigned short	(65536)

Límites del tipo real <float.h>

FLT_RADIX	dígitos del exponente	(2)
FLT_ROUNDS	modo de redondeo	
FLT_DIG	precisión (dígitos decimales)	(6)
FLT_EPSILON	menor x tal que $1.0 + x \neq 1.0$	(10^{-5})
FLT_MANT_DIG	dígitos de la mantisa	
FLT_MAX	máximo número en coma flotante	(10^{37})
FLT_MAX_EXP	exponente máximo	
FLT_MIN	mínimo número en coma flotante	(10^{-37})
FLT_MIN_EXP	mínimo exponente	
DBL_DIG	precisión de double (díg. decimales)	(10)
DBL_EPSILON	menor x t.q. $1.0 + x \neq 1.0$ (double)	(10^{-9})
DBL_MANT_DIG	díg. de la mantisa (double)	
DBL_MAX	máx. núm. en coma flot.(double)	(10^{37})
DBL_MAX_EXP	máximo exponente (double)	
DBL_MIN	mín. núm. en coma flot.(double)	(10^{-37})
DBL_MIN_EXP	mínimo exponente (double)	

Puede enviar comentarios y correcciones a J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩

Traducido por F. Abad, C.D. Martínez, D. Picó, J.A. Sánchez