

Tarjeta de referencia ANSI C

Estructura de programa/funciones

<i>tipo func(tipo₁,...)</i>	declaración de funciones
<i>tipo nombre</i>	declaración de variables globales
main() {	función principal
<i>declaraciones</i>	declaración de variables locales
<i>instrucciones</i>	
}	
<i>tipo func(arg₁,...)</i> {	definición de función
<i>declaraciones</i>	declaración de variables locales
<i>instrucciones</i>	
return <i>valor</i> ;	
}	
<i>/* */</i>	comentarios
main (int argc, char *argv[])	programa con argumentos

Preprocesador de C

incluir fichero de cabeceras	#include < <i>fichero</i> >
incluir fichero de usuario	#include " <i>fichero</i> "
sustitución de texto	#define <i>nombre</i> <i>texto</i>
macro con argumentos	#define <i>nombre</i> (<i>var</i>) <i>texto</i>
<i>Ejemplo.</i> #define max(A,B) ((A)>(B) ? (A) : (B))	
anular definición	#undef <i>nombre</i>
entrecomillar al reemplazar	#
concatenar argumentos y reescanear	##
compilación condicional	#if , #else , #elif , #endif
<i>¿nombre</i> definido, no definido?	#ifdef , #ifndef
<i>¿nombre</i> definido?	defined (<i>nombre</i>)
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 <i>tipo</i> <i>nombre</i>
talla de un objeto (devuelve un size_t)	sizeof <i>objeto</i>
talla de un tipo de datos (dev. un size_t)	sizeof (<i>tipo</i>)

Inicialización

Inicializar variable	<i>tipo</i> <i>nombre</i> = <i>valor</i>
Inicializar vector	<i>tipo</i> <i>nombre</i> []={ <i>valor</i> ₁ ,...}
Inicializar cadena	char <i>nombre</i> [" <i>cadena</i> "]

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', '\xhh'
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 <i>tipo</i>	<i>tipo</i> * <i>nombre</i>
decl. una func. que dev. un punt. a <i>tipo</i>	<i>tipo</i> * f ()
decl. un punt. a func. que devuelve <i>tipo</i>	<i>tipo</i> (* pf ())
puntero genérico	void *
valor de puntero a nulo	NULL
objeto apuntado por <i>puntero</i>	* <i>puntero</i>
dirección del objeto <i>nombre</i>	& <i>nombre</i>
vector	<i>nombre</i> [<i>dim</i>]
vector multidimensional	<i>nombre</i> [<i>dim</i> ₁][<i>dim</i> ₂]...

Estructuras

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

Operadores (según precedencia)

acceso a campo de estructura	<i>nombre</i> . <i>campo</i>
acceso por puntero	<i>puntero</i> -> <i>campo</i>
acceso a elemento de vector	<i>nombre</i> [<i>índice</i>]
incremento, decremento	++, --
más, menos, no lógico, negación bit a bit	+, -, !, ~
acceso por puntero, direcc. de objeto	* <i>puntero</i> , & <i>nombre</i>
convertir tipo de expresión	(<i>tipo</i>) <i>expr</i>
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	<i>expr</i> ₁ ? <i>expr</i> ₂ : <i>expr</i> 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 <i>etiqueta</i>
etiqueta	<i>etiqueta</i> :
valor de retorno de función	return <i>expr</i>

Construcciones de flujo

instrucción if	if (<i>expr</i>) <i>instrucción</i> else if (<i>expr</i>) <i>instrucción</i> else <i>instrucción</i>
instrucción while	while (<i>expr</i>) <i>instrucción</i>
instrucción for	for (<i>expr</i> ₁ ; <i>expr</i> ₂ ; <i>expr</i> ₃) <i>instrucción</i>
instrucción do	do <i>instrucción</i> while (<i>expr</i>);
instrucción switch	switch (<i>expr</i>) { case <i>const</i> ₁ : <i>instrucción</i> ₁ break ; case <i>const</i> ₂ : <i>instrucción</i> ₂ break ; default : <i>instrucción</i> }

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 (excluido 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 1 ^{er} c en los n 1 ^{os} 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	<code>stdin</code>
flujo de salida estándar	<code>stdout</code>
flujo de error estándar	<code>stderr</code>
final de fichero	<code>EOF</code>
obtener un carácter	<code>getchar()</code>
imprimir un carácter	<code>putchar(<i>car</i>)</code>
imprimir con formato	<code>printf("formato",<i>arg</i>₁,...)</code>
imprimir en cadena <i>s</i>	<code>sprintf(<i>s</i>,"formato",<i>arg</i>₁,...)</code>
leer con formato	<code>scanf("formato",&<i>nombre</i>₁,...)</code>
leer de cadena <i>s</i>	<code>sscanf(<i>s</i>,"formato",&<i>nombre</i>₁,...)</code>
leer línea en cadena <i>s</i>	<code>gets(<i>s</i>)</code>
imprimir cadena <i>s</i>	<code>puts(<i>s</i>)</code>
E/S de ficheros	
declarar puntero a fichero	<code>FILE *<i>fp</i></code>
obtener puntero a fichero	<code>fopen("nombre","mode")</code>
modos: <i>r</i> (leer), <i>w</i> (escribir), <i>a</i> (añadir)	
obtener un carácter	<code>getc(<i>fp</i>)</code>
escribir un carácter	<code>putc(<i>car</i>,<i>fp</i>)</code>
escribir en fichero	<code>fprintf(<i>fp</i>,"formato",<i>arg</i>₁,...)</code>
leer de fichero	<code>fscanf(<i>fp</i>,"formato",<i>arg</i>₁,...)</code>
cerrar fichero	<code>fclose(<i>fp</i>)</code>
distinto de cero si error	<code>ferror(<i>fp</i>)</code>
distinto de cero si EOF	<code>feof(<i>fp</i>)</code>
leer línea en cadena <i>s</i> (< max cars.)	<code>fgets(<i>s</i>,max,<i>fp</i>)</code>
escribir cadena <i>s</i>	<code>fputs(<i>s</i>,<i>fp</i>)</code>

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

-	alineación a izquierda
+	imprimir con signo
<i>space</i>	imprimir espacio si no hay signo
0	rellenar por delante con ceros
<i>w</i>	anchura mínima del campo
<i>p</i>	precisión
<i>m</i>	carácter de conversión:
	h short, l long, L long double
<i>c</i>	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	<code>va_list <i>nombre</i>;</code>
inicializar puntero a args.	<code>va_start(<i>nombre</i>,<i>ultarg</i>)</code>
<i>ultarg</i> es el último parámetro con nombre de la función	
siguiente arg. sin nom., actualizar punt.	<code>va_arg(<i>nombre</i>,<i>tipo</i>)</code>
invocar antes de salir de la función	<code>va_end(<i>nombre</i>)</code>

Funciones útiles <stdlib.h>

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

Conversiones

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

Reserva de memoria

reserva memoria	<code>malloc(<i>talla</i>), calloc(<i>nobj</i>,<i>talla</i>)</code>
cambiar tamaño de la reserva	<code>realloc(<i>pts</i>,<i>talla</i>)</code>
liberar memoria	<code>free(<i>ptr</i>)</code>

Funciones de vectores

buscar clave en vect	<code>bsearch(<i>clave</i>,<i>vect</i>,<i>n</i>,<i>talla</i>,cmp())</code>
ordenar vect ascendentemente	<code>qsort(<i>vect</i>,<i>n</i>,<i>talla</i>,cmp())</code>

Funciones de hora y fecha <time.h>

tiempo de proc. usado por el programa	<code>clock()</code>
<i>Ejemplo.</i> <code>clock()/CLOCKS_PER_SEC</code> da el tiempo en segundos	
segundos desde 1/1/1.970 (hora de ref.)	<code>time()</code>
<i>tpo</i> ₂ - <i>tpo</i> ₁ en segs. (double)	<code>difftime(<i>tpo</i>₂,<i>tpo</i>₁)</code>
tipos numéricos para representar horas	<code>clock_t,time_t</code>
estructura estándar usada para fecha y hora	<code>tm</code>

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

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

Funciones matemáticas <math.h>

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

Límites del tipo entero <limits.h>

límites típicos para un sistema Unix de 32 bits	
<code>CHAR_BIT</code>	bits en char (8)
<code>CHAR_MAX</code>	máximo valor de char (127 o 255)
<code>CHAR_MIN</code>	mínimo valor de char (-128 o 0)
<code>INT_MAX</code>	máximo valor de int (+32767)
<code>INT_MIN</code>	mínimo valor de int (-32768)
<code>LONG_MAX</code>	máximo valor de long (+2147483647)
<code>LONG_MIN</code>	mínimo valor de long (-2147483648)
<code>SCHAR_MAX</code>	máximo valor de signed char (+127)
<code>SCHAR_MIN</code>	mínimo valor de signed char (-128)
<code>SHRT_MAX</code>	máximo valor de short (+32767)
<code>SHRT_MIN</code>	mínimo valor de short (-32768)
<code>UCHAR_MAX</code>	máximo valor de unsigned char (255)
<code>UINT_MAX</code>	máximo valor de unsigned int (65535)
<code>ULONG_MAX</code>	máximo valor de unsigned long (4294967295)
<code>USHRT_MAX</code>	máximo valor de unsigned short (65536)

Límites del tipo real <float.h>

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

Octubre 2002 v1.3s. Copyright © 2002 Joseph H. Silverman

La copia y distribución de esta tarjeta están permitidas siempre que el copyright y este permiso se mantengan en todas las copias.

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