ÍNDICE

II

CLASE 2

Estos son los apuntes no oficiales del curso CC3301-1 cursado en Otoño 2022 bajo las cátedras del docente Luis Mateu B. El material usado es directamente obtenido de cátedras, en adición a los materiales del curso. [1].

1 Esquema de la clase

- Enteros sin signo
- Enteros con signo
- Representación de enteros en base 2
- Representación de enteros negativos en complemento de 2
- Números reales y su representación
- Representación de caracteres en ASCII

2 Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
  - Expresiones y operadores
  - Reglas de inferencia para el tipo de una expresión
  - Mecanismos para definir nuevos tipos
- Tipos primitivos en C:
  - Enteros con signo: char, short int, int, long int, long long int
  - Abreviados: char, short, int, long, long long
  - Enteros sin signo: anteponer atributo unsigned, por ejemplo unsigned int
  - Reales: float, double
  - Punteros
- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)

3 Enteros sin signo

- Los enteros sin signo se representan internamente en binario (base 2)
- Usaremos la notación (x)b para indicar que la constante x esta representada en base b. Si no se indica la base, se supone base 10
- Ejemplo: $(13)_{10} = (1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^3 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13$
- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10
- ¡También se puede usar base 8 anteponiendo el prefijo 0! El 013 no es el mismo número que 13 porque está en octal: $013 = (13)_8 (011011)_2 = 8 + 2 + 1 = 11$
- Un cero a la izquierda sí vale en C. Es decir

  `int i= 010;` es equivalente a `int i= 8;`
- Un número en hexadecimal siempre comienza con el prefijo 0x y cada cifra representa 4 bits:

$(2001)_{10} = (11111010001)_2 = (7d1)_{16} = 0x7d1$

- Se puede usar base 16 anteponiendo el prefijo 0x: $0x13 = (13)_{16} = (00010011)2 = 16 + 2 + 1 = 19$
- No existe 0b0110 para binarios, ¡pero lo usaremos en las explicaciones!

4 Conversiones

- Para convertir un número binario $x_{n+1}...x_0$ a base 10 hay que calcular $\sum_{i=0}^{n-1} x_i 2^i$
- Ejemplo: $(1100101)_2 = 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 25 + 1^26 = 1 + 4 + 32 + 64 = 101$
- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0
3 / 2	1	1
1 / 2	0	1

Figura 2.1: Ejemplo: $25 = (?)_2$. Respuesta: $25 = (11001)_2$

5 Rango de representación de enteros sin signo

C ofrece enteros sin signo anteponiendo el atributo unsigned al tipo. La siguiente tabla describe los enteros sin signo. [2]

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	[0, 2^8[≡ [0, 255]	[0, 2^8[	[0, 2^8[
unsigned short	2	2	2
	[0, 2^{16}[≡ [0, 65535]	[0, 2^{16}[	[0, 2^{16}[
unsigned int	4	4	**2**
	[0, 2^{32}[≡ [0, 4.294.967.295]	[0, 2^{32}[	**[0, 2^{16}[**
unsigned long	8	**4**	**4**
	[0, 2^{64}[	**[0, 2^{32}[**	**[0, 2^{32}[**
unsigned long long	8	8	8
	[0, 2^{64}[	[0, 2^{64}[	[0, 2^{64}[

Figura 2.2: Rango de representación de enteros sin signo

- Que una máquina sea de n bits significa que los punteros son de n bits y por lo tanto puede direccionar hasta $2n$ bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits
- Las máquinas de 64 bits pueden correr los programas de las máquinas de 32 bits
- Muchos sistemas embebidos y dispositivos usan procesadores de 16 bits por razones de costo (mouse y teclado por ejemplo)
- Observe que si x e y son sin signo, x-y todavía se calcula como x+∼y+1.

6 Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

Figura 2.3: Rango de representación de enteros con signo

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128
- Los enteros positivos se representan en base 2 como si fuesen enteros sin signo
- Los enteros negativos se representan en complemento de 2
- Al operar con enteros, si se produce un desborde en la representación no se genera ningún tipo de error, ¡pero el resultado es incorrecto!
- El tipo char es unsigned en algunas plataformas (use signed char)
- En la plataforma Windows de 64 bits, el tipo long es de 32 bits (no es de 64 bits como en Unix).
- A partir del estandar C99 existe el tipo long long. Se especifica que debe ser de al menos 8 bytes.
- Observe que aún cuando los procesadores de PCs y smartphones son de 64 bits, la mayoría de los smartphones funcionan en modo 32 bits, a no ser que tengan al menos 4 GB de memoria RAM.
- Tampoco se fabrican PCs de 16 bits, pero se venden muchos procesadores para sistemas embebidos que son de 16 bits, con precios insignificantes al lado de sus hermanos de 32 o 64 bits. Por razones de costos nadie colocaría un procesador de 32 o 64 bits para controlar una lavadora.

Advertencia: Cuando se opera con números enteros y se produce un desborde, el runtime de C no genera ningún tipo de error. El resultado simplemente se trunca al tamaño que debe poseer el resultado. El tamaño está especificado por las reglas de inferencia de tipos que veremos en el curso. [1]

[1] El contenido en azul es extracción directa del apunte de Luis Mateu

7 Representación de números negativos (Apunte)

Los números negativos se representan en complemento de 2. Esto significa que si se representa un número positivo x en 8 bits, entonces -x se representa negando todos los sus bits (lo que se denomina complemento de 1) y sumando 1. Ejemplos: [2]

valor positivo	representación en binario	valor negativo	comp. de 1	comp. de 2
0	00000000	-0	11111111	000000000
1	00000001	-1	11111111	111111111
2	00000010	-2	11111111	111111110
3	00000011	-3	11111111	111111101
...	...	...	...	...
127	01111111	-127	10000000	100000001
-	-	-128	-	100000000

Cuadro 2.1: Tabla de Representación de números negativos (comp. = complemento)

- Observe que todos los números positivos tienen el primer bit en 0
- Todos los negativos tienen el primer bit en 1
- Se puede representar el -128, pero no el 128
- En C la expresión x entrega el complemento de 1 de x (la negación bit a bit)
- la expresión -x entrega el complemento de 2 de x: x + 1 (es decir le cambia el signo)

¿Por qué se escogió esta representación? El número de transistores que tenían los primeros computadores era muy limitado. La operación más compleja era lejos la suma y no quedaban transistores para implementar una resta, multiplicación o división. La gracia de la representación en complemento de 2 es que:

```
x-y == x+(-y) == x+(~y+1)
```

Es decir que podemos hacer la resta sumando el complemento de 2. No se necesita un nuevo circuito para hacer la resta. Por otra parte la multiplicación se puede realizar haciendo sumas, y la división haciendo sumas y restas. ¡La economía en transistores es mayúscula!

8 Representación de negativos en complemento de 2

¿Cómo se representa el -28 en binario en un char?

- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente llegar a 0): 11100 (16+8+4)
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100
- En resumen para representar un entero negativo, calcular complemento de 1 + 1 del valor positivo
- ¿Por qué se llama complemento de 2?
- Porque complemento de 1 + 1 = complemento de (1+1) = complemento de 2
- Es humor tecnológico
- En C: $-x \equiv \sim x + 1$

9 Tabla de valores y sus equivalencias

Número binario	Valor sin signo	Valor con signo
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
...		
01111111	127	127
10000000	128	-128
10000001	129	-127
...	...	...
11111100	252	-4
11111101	253	-3
11111110	254	-2
11111111	255	-1

Figura 2.4: Tabla de valores y sus equivalencias

- El -1 es 111....1111 en todos los tamaños (int, long, short, etc.)
- El primer bit de los negativos es siempre 1
- Por eso se llama el bit de signo

10 Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?
- Para convertir un número binario de n bits $x_{n-1}...x_0$ a base 10:
  - Si bit de signo $x_{n-1} \equiv 0 : calcular \sum_{i=0}^{n-1} x_i 2^i$
  - Si bit de signo $x_{n-1} \equiv 1 : calcular \sum_{i=0}^{n-1} x_i 2^i - 2^n$
- Para 11101101, $n \equiv 8 : \sum \equiv 128 + 64 + 32 + 8 + 4 + 1 = 237$
- Como el bit de signo es $1 : 237–256 = -19$
- ¿Por qué se eligió el complemento de 2?
- (La alternativa es signo y magnitud.)
- ¡Porque la suma de los enteros con signo es la misma que la suma de los enteros sin signo!
- Porque $x–y \equiv x+\sim y + 1$
- ¡El mismo sumador sirve para sumar o restar enteros con o sin signo!

11 Representación de números reales en punto fijo ¡No se usa!

- Se destina una cantidad fija de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  0000000000000110 . 0100000000000000 porque su valor es $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1.2^{-2}$
- El punto va en un lugar fijo
- En general si:

  $x = 15...x_0 x_{-1}...x_{-16}$ el valor sería $\sum_{i=-16}^{15} x_i 2^i$
- Habría que destinar otro bit para indicar el signo

12 Representación de números reales en punto flotante de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110{,}01)_2$
- Hay que normalizar el número: se reescribe de manera que esté en el formato $1.bbbb... \cdot 2^e$
- Ejemplo: $6.25 = (1.1001)_2 \cdot 2^2$
- En general x estará en el formato:

  $$\text{signo} \cdot 1\ .\ m_{-1}m_{-2}...m_{-23} \cdot 2^e$$

- En donde signo puede ser 1 o -1
- Los bits $m_{-1}m_{-2}...m_{-23}$ se llaman la mantisa
- El número x se representa como: s $e_7...e_0$ $m_{-1}m_{-2}...m_{-23}$
- Con s=0 si signo es 0 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo
- El valor sin signo de $e_7...e_0$ es $e + 127$ (129 para 6.25)
- Por lo tanto 6.25 = 0100 0000 1100 1000 00000000 00000000

13 Representación de números reales en punto flotante

- Los casos $e_7...e_0 = 0$ o 255 son especiales
- El 0 se representa como 000...0 (solo ceros)
- Hay una representación para el NaN: *not a number*
- El tipo float: entrega unos 7 dígitos de precisión
  - Ocupa 32 bits, la mantisa es de 23 bits y el exponente de 8
  - La máxima magnitud representable es $\sim 3.4 \cdot 10^{38}$
  - La mínima magnitud es $\sim 1.18 \cdot 10^{-38}$
- El tipo double: entrega unos 15 dígitos de precisión
  - Ocupa 64 bits, la mantisa es de 52 bits y el exponente de 11
  - La máxima magnitud representable es $\sim 1.79 \cdot 10^{308}$
  - La mínima magnitud es $\sim 2.23 \cdot 10^{-308}$
- Ud. encontrará más detalles en la Wikipedia
- Cuidado: 0.1 no es representable de manera exacta
- Cuidado: ¡Nunca escriba $x == y$! Use $|x - y| < \epsilon$
- Debido a las imprecisiones del cálculo x será aproximadamente y, no igual
- Conjetura: el error de $x + y + z + w$ es mayor al de $(x + y) + (z + w)$

14 Representación de caracteres

- Se usa la codificación ASCII
- Las constantes 'a' 'b' 'c' ... 'z' son 97 98 99 ... 122
- 'A' 'B' 'C' ... 'Z' son 65 66 67 ... 90
- '0' '1' '2' ... '9' son 48 49 50 ... 57
- '!' es 33 "es 34 ... etc.
- '\n' es 10
- Note que 'A'+1 es 'B' y que '0'+4 es '4'

En la página siguiente una tabla de ejemplo.

b7 b6 b5 / BITS / b4 b3 b2 b1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SYMBOLS NUMBERS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	16 DLE 10 20	32 SP 20 40	48 0 30 60	64 @ 40 100	80 P 50 120	96 ‘ 60 140	112 p 70 160
0 0 0 1	1 SOH 1 1	17 DC1 11 21	33 ! 21 41	49 1 31 61	65 A 41 101	81 Q 51 121	97 a 61 141	113 q 71 161
0 0 1 0	2 STX 2 2	18 DC2 12 22	34 " 22 42	50 2 32 62	66 B 42 102	82 R 52 122	98 b 62 142	114 r 72 162
0 0 1 1	3 ETX 3 3	19 DC3 13 23	35 # 23 43	51 3 33 63	67 C 43 103	83 S 53 123	99 c 63 143	115 s 73 163
0 1 0 0	4 EOT 4 4	20 DC4 14 24	36 $ 24 44	52 4 34 64	68 D 44 104	84 T 54 124	100 d 64 144	116 t 74 164
0 1 0 1	5 ENQ 5 5	21 NAK 15 25	37 % 25 45	53 5 35 65	69 E 45 105	85 U 55 125	101 e 65 145	117 u 75 165
0 1 1 0	6 ACK 6 6	22 SYN 16 26	38 & 26 46	54 6 36 66	70 F 46 106	86 V 56 126	102 f 66 146	118 v 76 166
0 1 1 1	7 BEL 7 7	23 ETB 17 27	39 ’ 27 47	55 7 37 67	71 G 47 107	87 W 57 127	103 g 67 147	119 w 77 167
1 0 0 0	8 BS 8 10	24 CAN 18 30	40 (28 50	56 8 38 70	72 H 48 110	88 X 58 130	104 h 68 150	120 x 78 170
1 0 0 1	9 HT 9 11	25 EM 19 31	41) 29 51	57 9 39 71	73 I 49 111	89 Y 59 131	105 i 69 151	121 y 79 171
1 0 1 0	10 LF A 12	26 SUB 1A 32	42 * 2A 52	58 : 3A 72	74 J 4A 112	90 Z 5A 132	106 j 6A 152	122 z 7A 172
1 0 1 1	11 VT B 13	27 ESC 1B 33	43 + 2B 53	59 ; 3B 73	75 K 4B 113	91 [5B 133	107 k 6B 153	123 { 7B 173
1 1 0 0	12 FF C 14	28 FS 1C 34	44 , 2C 54	60 < 3C 74	76 L 4C 114	92 \ 5C 134	108 l 6C 154	124 | 7C 174
1 1 0 1	13 CR D 15	29 GS 1D 35	45 − 2D 55	61 = 3D 75	77 M 4D 115	93] 5D 135	109 m 6D 155	125 } 7D 175
1 1 1 0	14 SO E 16	30 RS 1E 36	46 . 2E 56	62 > 3E 76	78 N 4E 116	94 ^ 5E 136	110 n 6E 156	126 ~ 7E 176
1 1 1 1	15 SI F 17	31 US 1F 37	47 / 2F 57	63 ? 3F 77	79 O 4F 117	95 _ 5F 137	111 o 6F 157	127 DEL 7F 177

LEGEND: dec CHAR hex oct

Victor Eijkhout
Dept. of Comp. Sci.
University of Tennessee
Knoxville TN 37996, USA

Figura 2.5: ASCII CONTROL CODE CHART

REFERENCIAS

[1] Mateu, Luis: *Programación de Software de Sistemas - Novedades.* `https://www.u-cursos.cl/ingenieria/2022/1/CC3301/1/novedades/`, 2022.

[2] Mateu, Luis: *Programación de Software de Sistemas - Apuntes.* `https://wiki.dcc.uchile.cl/cc3301/temario/`, 2022.