Tipos de Datos, Operadores y Operaciones con Bits

Tipos de Datos

- Estandar ANSI:
  - Formas “signed” y “unsigned” para enteros
  - Precisión sencilla y extendida para operaciones de punto flotante
  - Enumeraciones como parte del lenguaje
  - Existencia de constantes

Tamaño de datos

- Datos básicos:
  - char: 1 byte (8 bits)
  - int: 1 entero, su tamaño depende de la máquina en que se ejecuta (16 o 32 bits)
  - float: punto flotante de precisión normal
  - double: punto flotante de doble precisión
    - (Ejemplo: limits.h)

Tamaño de datos

- Calificadores:
  - short: calificador para entero (comúnmente 16 bits)
  - long: calificador para entero (comúnmente 32 bits)
  - unsigned: calificador para char o cualquier entero, son siempre positivos o cero, ej:
    - unsigned char: 0 hasta 255
  - signed: calificador para char o cualquier entero, rango negativo y positivo:
    - signed char: -128 hasta 127

(Ejemplo: ascii.c)

Operadores Aritméticos

- +, -, *, /, %
- Operador % no se puede aplicar a tipos float o double
- Ej: Un año es bisiesto si es divisible por 4, pero no por 100, excepto los divisibles por 400, que son bisiestos:

```
if ((year % 4 == 0 && year % 100 != 0) || year %400 == 0)
    prinf("%d es un año bisiesto\n", year);
else
    prinf("%d no es un año bisiesto\n", year);
```


Op. de Relación y Lógicos

- Operadores de relación:
  - >, >=, <, <=
  - menor precedencia que operadores aritméticos:
    - i < total – 1 se toma como i < (total - 1)
- Operadores Lógicos:
  - ||, &&
  - Se evaluan de izquierda a derecha
  - Termina la evaluación cuando se conoce el resultado (verdadero o falso)

Operadores Lógicos

- En C, verdadero = 1, falso = 0
- Operador de negación: !
  - if (!valido) es igual a: if (valido == 0)
- Ejemplo uso de precedencia, función getline:

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


Representación de Enteros

- En C, los enteros se representan internamente en binario:
  - $(2004)_{10} = (11111010100)_2$
  - $(2^{10}+2^9+2^8+2^7+2^6+0+2^4+0+2^2+0+0)$
  - $(1024+512+256+128+64+16+4)$

Representación de Enteros

- Base Octal:
  - $(2004)_{10} = (11111010100)_2$
  - Grupos de 3 bits:
    - $(11\ 111\ 010\ 100)_2$
    - $(3 \quad 7 \quad 2 \quad 4)_8$
    - En C es 03724, el “0” indica que es octal

Representación de Enteros

- Base Hexadecimal: dígitos del 0 al 9 y las letras A, B, C, D, E, F:
  - $(2004)_{10} = (11111010100)_2$
  - Grupos de 4 bits:
    - $(111\ 1101\ 0100)_2$
    - $(7 \quad D \quad 4)_{16}$
    - En C es 0x7D4, el “0x” indica que es hexadecimal

➔ (Ejemplo: bases.c)

Operaciones con Bits

- AND (&), OR (|):

&	0	1
0	0	0
1	0	1

Hacer 0 todos los bits de n, exepto los 7 de menor orden:
n = n & 0177

|	0	1
0	0	1
1	1	1

Encender bits:
n = n | SET_ON

Operaciones con Bits

- XOR (^), NOT (~):

^	0	1
0	0	1
1	1	0

~	
0	1
1	0

Fijar en 0 los últimos 6 bits de x:
x = x & ~077

(Ejemplo: enc.c)

Operaciones con Bits

- Shift left (<<):
  - Movimiento de bits hacia la izquierda:
    - $(1011\ 0110)_2$ << 3 = $(101\ 1011\ 0000)_2$
    - $(182)_{10}$ << 3 = $(1456)_{10}$
  - Multiplicación: $182 * 2^3 = 1456$

Operaciones con Bits

- Shift rigth (>>):
  - Movimiento de bits hacia la derecha:
    - $(1011\ 0110)_2$ >> 3 = $(0001\ 0110)_2$
    - $(182)_{10}$ >> 3 = $(22)_{10}$
  - División: $182 / 2^3 = 22.75$ (se considera sólo la parte entera)

Operaciones con Bits

- Función getbits(x, p, n), retorna 'n' bits de x, comenzando en 'p':

```
unsigned getbits(unsigned x, int p, int n){
    return (x >> (p + 1 – n)) & ~(~0<<n);
}
```


Big-Little Endian

- Big Endian: El byte más significativo se almacena en la dirección de memoria mas pequeña
- Little Endian: El byte menos significativo se almacena en la dirección de memoria mas pequeña:

```
char   c1 = 1;
char   c2 = 2;
short  s = 255;   // 0x00FF
long   l = 0x44332211;
```

Offset :	Memory dump
0x0000 :	01 02 FF 00
0x0004 :	11 22 33 44

Big-Little Endian

- Big Endian: Protocolo TCP, procesadores Sun Sparc, Motorola's 68K(Macintosh), Java Virtual Machine
- Little Endian: Procesadores Intel x86 y sus clones (AMD)
- Número: 42553 (2 bytes)

Big Endian: 10100110 00111001

Little Endian: 00111001 10100110

Big-Little Endian

- Importante saber en que formato se está trabajando: archivos binarios
- Sistema Little Endian debe convertir los paquetes que envía por TCP
- Estos formatos también pueden aplicarse a la ordenación de bits
- Pueden existir sistemas con big endian para bytes y little endian para bits

Big-Little Endian

- Para comunicaciones en TCP: network byte order
- En C:
  - htons(): "host to network short."
  - htonl(): "host to network long."
  - ntohs(): "network to host short."
  - ntohl(): "network to host long."

Big-Little Endian

```
#if defined(BIG_ENDIAN) && !defined(LITTLE_ENDIAN)

#define htons(A)  (A)
#define htonl(A)  (A)
#define ntohs(A)  (A)
#define ntohl(A)  (A)

#elif defined(LITTLE_ENDIAN) && !defined(BIG_ENDIAN)

#define htons(A)  ((((uint16)(A) & 0xff00) >> 8) | \
                   (((uint16)(A) & 0x00ff) << 8))
#define htonl(A)  ((((uint32)(A) & 0xff000000) >> 24) | \
                   (((uint32)(A) & 0x00ff0000) >> 8 )  | \
                   (((uint32)(A) & 0x0000ff00) << 8 )  | \
                   (((uint32)(A) & 0x000000ff) << 24))
#define ntohs     htons
#define ntohl     htohl

#else

#error "Either BIG_ENDIAN or LITTLE_ENDIAN must be #defined, but not both."

#endif
```