Laboratorio 7

Programación de Computadores (2025-2)

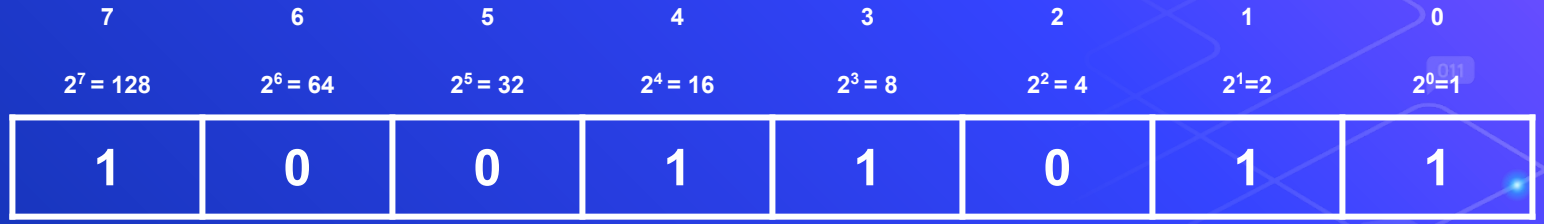
Temas del Laboratorio 7

- Operaciones a nivel de Bits
- Ejercicios

Datos como bits

Sabemos que los datos, en memoria, se representan como bits:

- Un int usa 32 bits, un char usa 8 bits, etc.



Esto puede ser un char en binario: representa el número 155 decimal

128 + 16 + 8 + 2 + 1 = 155

Ejemplo: int to bits

En el repositorio hay un archivo ***bit_representation.h*** que incluye una función ***int2bits*** que recibe un entero e imprime su representación binaria.

```
Representación binaria de 155
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1
```

A continuación realizaremos operaciones a nivel bit sobre este número y veremos cómo se va modificando

Operaciones a nivel de bit en C

Las operaciones a nivel de bit en C son operaciones que se realizan directamente en los bits individuales de los datos almacenados en variables.

Operadores a nivel de bit

OPERADOR	SÍMBOLO	EJEMPLO
AND bitwise	&	`1001 (9) & 1100 (12) = 1000 (8)`
OR bitwise	|	`1001 (9) | 1100 (12) = 1101 (13)`
XOR bitwise	^	`1001 (9) ^ 1100 (12) = 0101 (5)`
NOT bitwise	~	`~1001 (9) = 0110 (6)`
Left Shift	<<	`0000 0101 (5) << 2 = 0001 0100 (20)`
Right Shift	>>	`0001 0000 (16) >> 2 = 0000 0100 (4)`

Ejemplos visuales

En el repositorio hay un archivo **operadores.c** donde podemos ver cómo se modifican los bits a medida que se utilizan los **operadores bitwise (los anteriores)**

A continuación realizaremos operaciones a nivel bit sobre este número y veremos cómo se va modificando

Ejemplos: left shift

```
Ingrese número: 155
Representación binaria de 155
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1

Representación binaria de 310
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1  0

Representación binaria de 620
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1  0  0

Representación binaria de 1240
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1  0  0  0

Representación binaria de 2480
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  0  1  1  0  0  0  0
```

x<<0 (original)

x<<1

x<<2

x<<3

x<<4

Ejemplos: left shift

155<<0 = 155 = 155 * 2^0

155<<1 = 310 = 155 * 2^1

155<<2 = 620 = 155 * 2^2

155<<3 = 1240 = 155 * 2^3

155<<4 = 2480 = 155 * 2^4

Hacer “shifting” a la izquierda es multiplicar el número por la n-ésima potencia de 2

Ejemplos: left shift

Podemos ejecutar el resto del código de ***operadores.c*** para ver más de los demás operadores

```
49
50  printf("#####################################################\n");
51  printf("### Otros operadores a nivel de bits: &, |, ^ y ~ ###\n");
52  printf("#####################################################\n\n");
53  int y = 15;
54  int z = 203;
55
56  int2bits(y);
57  int2bits(z);
58
59  printf("Operador NOT - complemento (~)\n");
60  int2bits(~y);
61
62  printf("Operador AND (&)\n");
63  int2bits(y & z);
64
65  printf("Operador OR (|)\n");
66  int2bits(y | z);
67
68  printf("Operador XOR (^)\n");
69  int2bits(y ^ z);
```

Ejemplo 1: ejemplo simple que no imprime la representación binaria

```c
#include <stdio.h>

int main() {
    int num1 = 9;  // Representación binaria: 1001
    int num2 = 5;  // Representación binaria: 0101

    int res_and = num1 & num2;  // Realiza AND bitwise
    int res_or = num1 | num2;   // Realiza OR bitwise
    int res_xor = num1 ^ num2;  // Realiza XOR bitwise
    int res_not = ~num1;        // Realiza NOT bitwise

    printf("Resultado de AND bitwise: %d\n", res_and);
    printf("Resultado de OR bitwise: %d\n", res_or);
    printf("Resultado de XOR bitwise: %d\n", res_xor);
    printf("Resultado de NOT bitwise: %d\n", res_not);

    return 0;
}
```

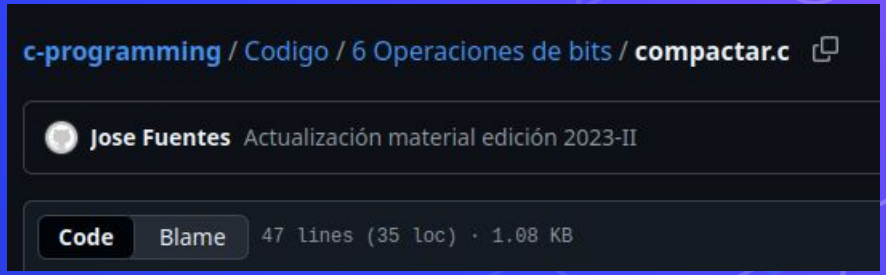
Ejemplo 2: toma *x*, y lo desplaza *n* veces en ambas direcciones

```c
#include <stdio.h>
#include <stdlib.h>

// Uso: ./programa <numero> <desplazamiento>
int main(int argc, char *argv[]) {

    if (argc != 3) {
        printf("Uso: %s <numero> <desplazamiento>\n", argv[0]);
        return 1;
    }

    int num = atoi(argv[1]);

    int desplazamiento = atoi(argv[2]);

    int des_izq = num << desplazamiento;
    int des_der = num >> desplazamiento;

    printf("Resultado de desplazamiento a la izquierda: %d\n", des_izq);
    printf("Resultado de desplazamiento a la derecha: %d\n", des_der);

    return 0;
}
```

Ejemplo 3

Para este tercer ejemplo, usaremos el código del siguiente link:

compactar.c



Ejemplo 3

compactar.c muestra cómo varios números pequeños pueden “empaquetarse” juntos en una sola variable usando operaciones a nivel de bit, ahorrando espacio y mostrando cómo manipular los bits correctamente.

Ejemplo 3

Se genera un arreglo de 8 números aleatorios (entre 0 y 15).

Busca el más grande para saber cuántos bits son necesarios para representar cualquier conjunto.

Con eso, toma cada número y lo coloca (usando desplazamiento y OR a nivel bit, para que queden juntos sin solaparse.)

```
13    int x;
14    int n = 8; // Número de elementos
15    int arr[n];
16    int max;
17
18    // Generación de elementos pequeños al azar. Los elementos pertenecen al rango
19    // [0,15]
20    for(int i=0; i < n; i++)
21      arr[i] = rand() % 16;
22
23    // Encontrar el elemento mayor
24    max = arr[0];
25    for(int i=1; i < n; i++)
26      if(max < arr[i])
27        max = arr[i];
28
29    // Cantidad mínima de bits para representar todos los elementos
30    int n_bits = (int)ceil(log2(max+1));
31    printf("Se necesitan %d bits para representar todos los elementos\n", n_bits);
32
33    // Usaremos los 32 bits de esta variable para almacenar 8 enteros entre 0 y 15
34    unsigned int compacto;
35
36    for(int i=0, d=0; i < n; i++, d+=n_bits) {
37      compacto = compacto | (arr[i] << d);
38    }
39
40    for(int i=0; i < n; i++)
41      int2bits(arr[i]);
42
43    printf("\n*** Compacto ***\n");
44    int2bits(compacto);
45
46    return 0;
47  }
```

Ejemplo 3: Desplazamiento y OR

```
32
33    // Usaremos los 32 bits de esta variable para almacenar 8 enteros entre 0 y 15
34    unsigned int compacto;
35
36    for(int i=0, d=0; i < n; i++, d+=n_bits) {
37      compacto = compacto | (arr[i] << d);
38    }
39
```

Si tenemos

a = 9

1	0	0	1

b = 6

0	1	1	0

ponemos a, luego b

1	0	0	1

desplazamos a la izquierda (<<4)

1	0	0	1	0	0	0	0

hacemos OR

1	0	0	1	0	1	1	0

a b

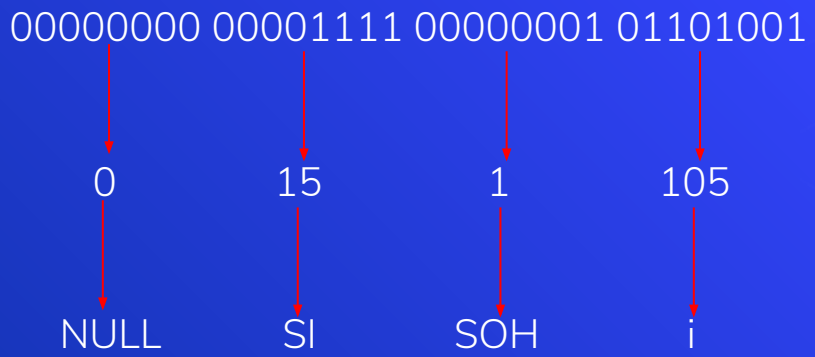
compacto

Ejercicio 1

Escribe un programa que pida un número entero (de 4 Bytes) y luego imprima tanto la representación entera como el código ASCII correspondientes a cada uno de los bytes que componen el número ingresado.

(puede que los caracteres de control no impriman nada)

Ejemplo: 983401 en binario es:



Ejercicio 2

Crear un programa que revierta la compactación del ejemplo 3

```
*** Compacto ***
Representación binaria de -890160793
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 1  1  0  0  1  0  1  0  1  1  1  1  0  0  0  1  0  0  1  1  1  0  0  1  0  1  1  0  0  1  1  1
```

```
resultado: 7 6 9 3 1 15 10 12
```