

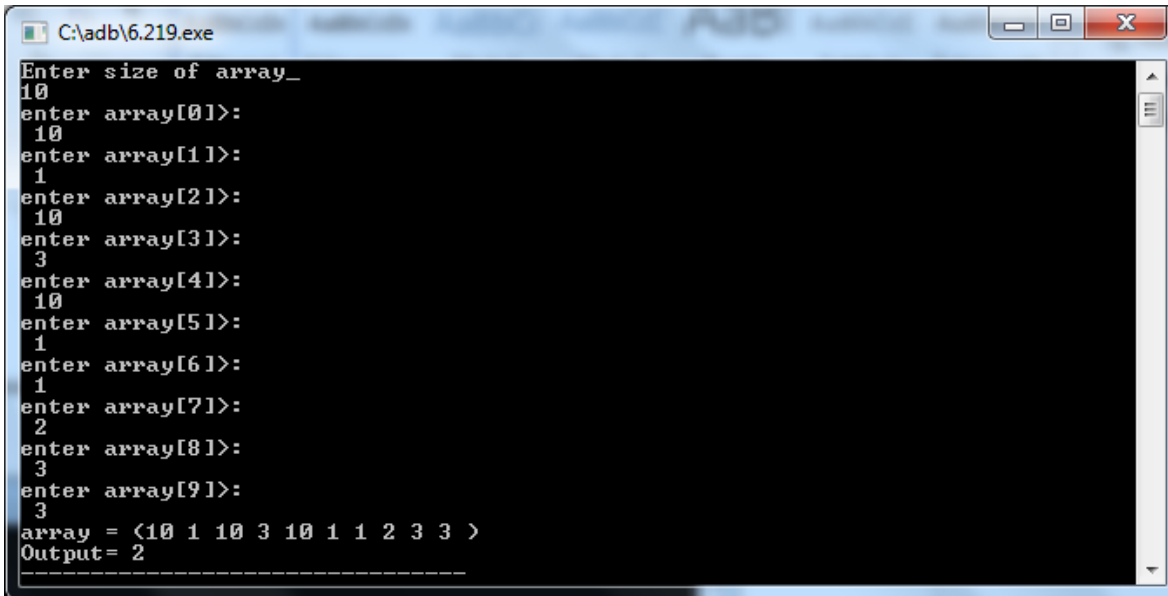
Practica 4. Operaciones binarias en C

Objetivos

Resultados

Problema 1 - Encuentra el único elemento de un arreglo.

Demostración:



```
C:\adb\6.219.exe
Enter size of array_
10
enter array[0]>:
10
enter array[1]>:
1
enter array[2]>:
10
enter array[3]>:
3
enter array[4]>:
10
enter array[5]>:
1
enter array[6]>:
1
enter array[7]>:
2
enter array[8]>:
3
enter array[9]>:
3
array = <10 1 10 3 10 1 1 2 3 3 >
Output= 2
```

Código – Solución:

```
//declarando variables
int k, z=0, i,yy,m;
int seen[] = {0};
int rest[] = {0};

int main(){
    printf("Enter size of array_\n");
    scanf("%d",&m);
    //se inicializa un arreglo de logitud dada y se piden sus valores uno por uno
    int array[m];
    for(w=0;w<m;w++){
        printf("enter array[%d]>:\n ",w);
        scanf("%d",&array[w]);
    }
    //se imprime el arreglo ingresado
    printf("array = (");
    for(w=0;w<sizeof(array)/4;w++){
        printf("%d ",array[w]);
    }
    printf(")\n");
    //se envia el arreglo mas su talla a la funcion secuencia
    yy = Secuencia(array,m);
    //se imprime el resultado de salida
    printf("Output= %d",yy);
}
```

```

int Secuencia(int x[],int x1){
    int len = x1; //len se define como la longitud del arreglo
    k = 0; //es la variable donde se almacenara la solucion

    //se recorren los valores del arreglo un ciclo for 1 sola vez
    //el arreglo seen[] contiene los numero recorridos
    //el arreglo rest[] contiene aquellos que ya se han
    //encontrado mas de 2 veces
    for(i =0;i<len;i++){
        //se evalua si el valor actual x1[i] es igual al valor de seen[] en su
        //posicion igual al valor actual, en referencia a que ha sido visto
        if(!(seen[x[i]]^x[i])){
            //si rest[] en su posicion igual al valor actual es igual al valor actual
            //implicando que se ha integrado a seen[] y se ha vuelto a encontrar
            if(!(rest[x[i]]^x[i])){
                k = k^x[i]; //se elimina el valor actual de k con un XOR
            }
            //se almacena el valor actual en rest[], en el indice igual al valor actual
            rest[x[i]] = x[i];
        }
        //Si el valor actual no sea visto, no es igual al valor de seen[] en su
        //indice igual a valor actual
        if(seen[x[i]]^x[i]){
            //seen en su indice igual al valor actual se iguala a x[i]
            seen[x[i]] = x[i];
            //se almacena en k con un XOR
            k = k^x[i];
        }
    }
    printf("\n");
    return (k); //se regresa k
}

```

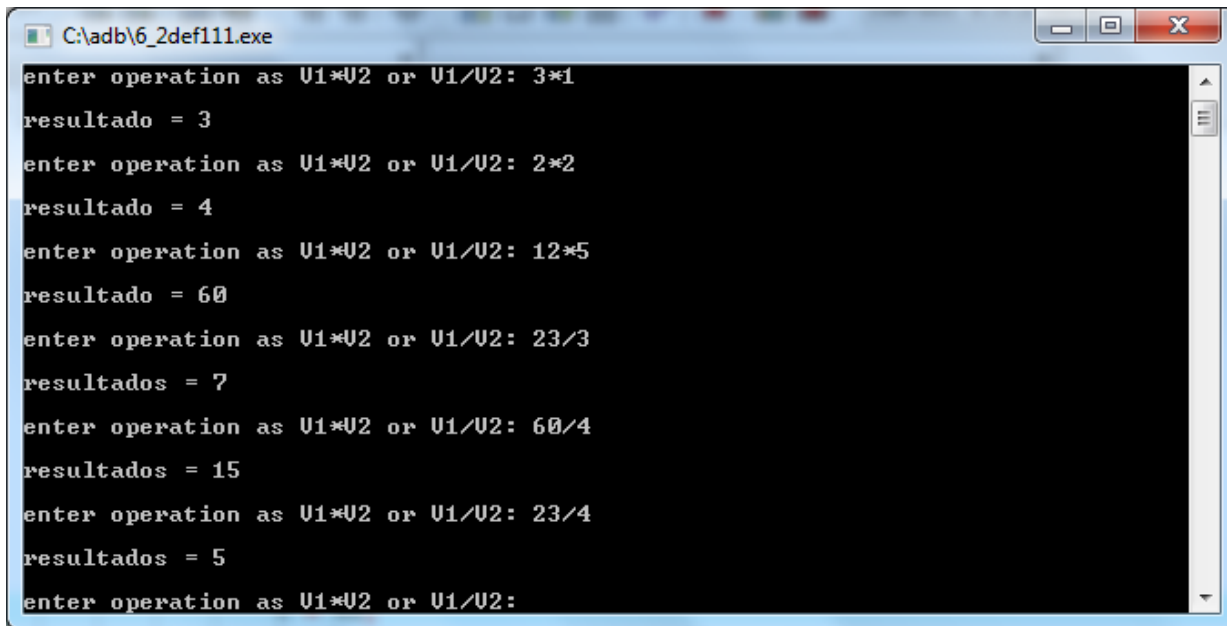
Explicación de la Solución

Para encontrar el valor en un solo recorrido del arreglo, se usaron 2 arreglos; uno (seen) para almacenar los valores ya vistos, para así recordar los números anteriores y el segundo (rest) para los valores que se han encontrado 2 veces. Entonces se lee el primer valor, se verifica que no se encuentre en seen[] y si no está, se almacena en ese arreglo; en la sig. Iteración de encontrarse el mismo valor se verifica que no esté en rest y luego se almacena indistintamente; si vuelve a leerse ese valor entonces al verificar que está en Seen[] y en Rest[] se elimina de la variable con el acumulado de todos los números encontrados, pues ya se ha encontrado más de 2 veces.

Para leer los arreglos Seen[] y Rest[] sin otro ciclo, al guardar un elemento se almacena en el índice con el valor del mismo elemento, para lo que se tienen arreglos sin longitud definida. Para la acumulación se utiliza un XOR de la variable con cada valor nuevo, para eliminarlos se hace lo mismo, donde el primer XOR solo sucede 1 vez cuando el valor no está en Seen[] aun.

Problema 2 - Multiplicaciones y divisiones

Demostración:



```
C:\adb\6_2def111.exe
enter operation as U1*U2 or U1/U2: 3*1
resultado = 3
enter operation as U1*U2 or U1/U2: 2*2
resultado = 4
enter operation as U1*U2 or U1/U2: 12*5
resultado = 60
enter operation as U1*U2 or U1/U2: 23/3
resultados = 7
enter operation as U1*U2 or U1/U2: 60/4
resultados = 15
enter operation as U1*U2 or U1/U2: 23/4
resultados = 5
enter operation as U1*U2 or U1/U2:
```

Código – Solución:

```
while(1){
    f = 0;
    printf("\n");
    printf("enter operation as V1*V2 or V1/V2: ");
    //se pide la operacion a traves de tres parametros
    //un operando y 2 operadores
    scanf("%d%c%d", &a , &h, &b);
    printf("\n");
    //dependiendo del caracter se escoge entra en una condicion
    if(h == '*'){
        //se manda llamar la funcion de multipliacion que recibe
        //los dos operandos enteros pedidos.
        printf("resultado = %d",multi(a,b));
        printf("\n");
    }

    x2 = 1;
    z = 1;
    if(h == '/'){
        while(multi(b,x2)<=a){
            //para encontrar el valor truncado, se verifica que
            //la multiplicacion del valor actual de x2 + 1, sea menor
            //que el dividendo, de esa forma se evita que se admitan valores
            //mayores cuando el resultado es decimal y queda entre medio
            if((multi(b,x2+1))>a){
                z = x2;
                break;
            }
            x2 = x2 + 1;
        }
    }
}
```

```

printf("resultados = %d",z);
printf("\n");
}
}

```

```

}

```

```

int multi(int a, int b){
    e = a; //se almacena el primer operando "a"
    c = b; //se almacena el segundo operando "b"
    d = 0; // se inicializa un acumulador
    c2 = 0;
    f = 0;
    //se determina si b es par o impar comparando
    //el doble de su division truncada con su valor --> (b/2)*2 < b
    if(((b>>1)<<1<b)){
        c = b - 1; //se resta 1 a c para que sea numero par
        f = 1; //se registra la entrada al condicional con f = 1
    }
    //se calcula el multiplo de 2 para c y se almacena en d
    for(w=1;w<c;w++){
        d = w<<1; //se multiplica w por 2
        //printf(" c2 = %d ",d);
        if(!(d^c)){ //cuando se encuentre w a la mitad de c se cumplira la condicion
            c2 = d>>1; //se divide d/2 para regresar rescatar la mitad de c
            //printf("c1 %d",c2);
            break; //se sale del ciclo
        }
    }
    d = c2; //se reinicializa el acumulador d
    //for(w=0;w<c2;w++){ //con el for se encuentra 2^c2
    // d = d<<1;
    //}
    p = 0; //se inicializa el acumulador p
    //se reinicia w y se inicializa y como 0
    w = 0;
    y = 0;

    break;
}
//si no se cumple la condicion, se incrementa w, hasta a*2 + y
//entonces en la siguiente corrida el primer ciclo comienza
//en y = w, para incrementarse hasta a*2 + su valor inicial.
for(w=y;w<(a<<1)+y;w++){
}
//se vuelve a hacer la misma condicional
p = p + 1;
if(p>=d){
    a = w; //a contiene ahora el resultado de la multiplicacion
    break;
}
}
}

```

```

if(f){//si b fue impar se le suma el primer operando al resultado
//a = a + e

    for(w=0;w<a;w++){//se incrementa w desde 0 hasta "a"
        }
    x = w<<2;//se multiplica es valor por 4 para que sea mayor a "a + e"
    for(y=w;y<x;y++){//se incrementa "y" con valor inicial de "w", hasta
        //x = 4*w
        if(y-w==e){//cuando y-w = e significa que y se ha incremntado "e" veces
            //donde "e" = al primer operando ingresado inicialmente.
            a=y;
        }

    }
    if(c == 0){//en caso de que el segundo operando fuese menor que 2,
        //se iguala el resultado al primer operando ingresado nada mas.
        a = e;

    }
}
if(!b){
    a = 0;
}
return(a);//se regresa el resultado
}

```

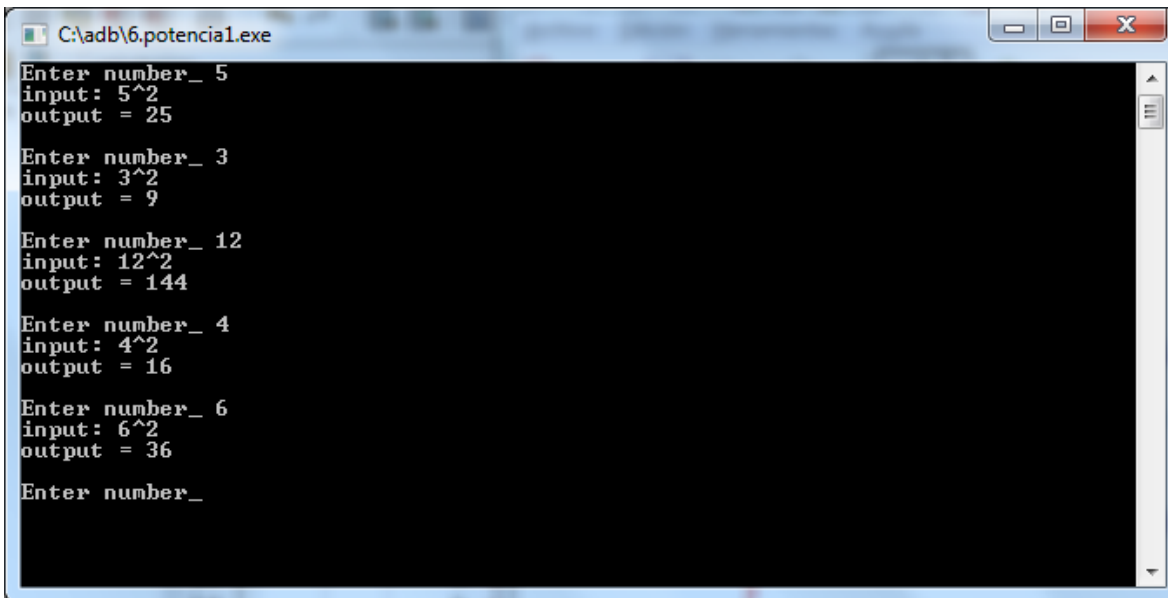
Explicación de la Solución

Para discernir entre división y multiplicación se utiliza los caracteres / y * como parte de la entrada. En el caso de la división se utilizó un ciclo while para verificar a través de la función de multiplicación (multit()) entre el divisor y un número que se va incrementando desde 1 hasta n, cuando el resultado es mayor al numerador o dividendo, donde en cada iteración se verifica si la siguiente será valida, de forma que al encontrar el número más cercano al resultado truncado, este se pueda almacenar en una variable y romper el ciclo, este continua siempre que la multiplicación $n * \text{divisor} < \text{dividendo}$.

Para la multiplicación el algoritmo consiste en encontrar la mitad del segundo término a través de un for que va multiplicando un numero por 2 con un corrimiento, hasta que se llega al doble; en caso de ser impar se le resta 1 y se añade una suma del primer operando al resultado final. Posteriormente se realiza una suma del primer operando por 2 el número de veces antes calculado, utilizando dos ciclos for dentro de un ciclo while, donde se utilizan dos acumuladores.

Problema 3 - Calcule el cuadrado de un número sin utilizar ningún operador numérico.

Demostración:



```
C:\adb\6.potencial.exe
Enter number_ 5
input: 5^2
output = 25

Enter number_ 3
input: 3^2
output = 9

Enter number_ 12
input: 12^2
output = 144

Enter number_ 4
input: 4^2
output = 16

Enter number_ 6
input: 6^2
output = 36

Enter number_
```

Código – Solución:

```
int main(){
    while(1){
        f = 0;
        printf("Enter number_ ");
        scanf("%d", &a); //se pide el numero a elevar
        e = a;
        b = a;
        c = b;
        d = 0;

        //se determina si b es par o impar comparando
        //el doble de su division truncada con su valor --> (b//2)*2 < b
        if(((b>>1)<<1<b)){
            c = b - 1; //se resta 1 a c para que sea numero par
            f = 1; //se registra la entrada al condicional con f = 1
        }
        //se calcula el multiplo de 2 para c y se almacena en d
        for(w=1;w<c;w++){
            d = w<<1; //se multiplica w por 2
            if(!(d^c)){ //cuando se encuentre w a la mitad de c se cumplira la condicion
                c2 = d>>1; //se divide d/2 para regresar rescatar la mitad de d
                break; //se sale del ciclo
            }
        }
        d = c2;
        p = 0;
        w = 0;
        y = 0;
        //se usa la siguiente secuencia para sumar a*2 el numero de veces
        //dado por el acumulador d.
        while(p<d){
            //se incrementa y, hasta a*2 + w, iniciando en w = 0
```

```

    for(y=w;y<(a<<1)+w;y++){
    }
    //si p es mayor o igual a "d" entonces se sale del ciclo while.
    p = p + 1;
    if(p>=d){
        a = y;//a contiene ahora el resultado de la multiplicacion
        break;
    }
    //si no se cumple la condicion, se incrementa w, hasta a*2 + y
    //entonces en la siguiente corrida el primer ciclo comienza
    //en y = w, para incrementarse hasta a*2 + su valor inicial.
    for(w=y;w<(a<<1)+y;w++){
    }
    p = p + 1;
    if(p>=d){
        a = w;//a contiene ahora el resultado de la multiplicacion
        break;
    }
}

//si al principio el "b" fue impar se realiza una suma de
// a + e, donde "e" tiene el valor inicial de "a", ingresado
//al principio del programa.
if(f){
for(w=0;w<a;w++){//se incrementa w desde 0 hasta "a"
}
x = w<<2;//se multiplica es valor por 4 para que sea mayor a "a + e"
for(y=w;y<x;y++){//se incrementa "y" con valor inicial de "w", hasta

//x = 4*w
    if(y-w==e){//cuando y-w = e significa que y se ha incremntado "e" veces
        //donde "e" = al primer operando ingresado inicialmente.
        a=y;
    }

}
if(c == 0){//en caso de que el segundo operando fuese menor que 2,
//se iguala el resultado al primer operando ingresado nada mas.
    a = e;


}
}
if(!b){
    a = 0;
}

//se imprime el resultado
printf("input: %d^2\n", e);
printf("output = %d\n\n",a);

}

```

Problema 4 - Generador de secuencias de Grey Code




```
C:\adb\6.grey_Chain21.exe

ingresa el numero de bits: 2
2-bit <n = 2>
00 01 11 10

ingresa el numero de bits: 3
3-bit <n = 3>
000 001 011 010 110 111 101 100

ingresa el numero de bits: 4
4-bit <n = 4>
0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

ingresa el numero de bits:
```



```

C:\adb\6.grey_Chain21.exe

5-bit (n = 5)
000000 000001 000011 000010 000110 000111 000101 000100 011000 011001 011011 011010 111100 111101 111110 111111
111111 111101 111100 101100 101101 101111 101110 001110 001111 001011 001010 011000 011001 011011 011010
1 01110 11110 11111 11101 11100

```

```

ingresa el numero de bits: 7

7-bit <n = 7>
00000000 00000001 00000011 00000100 00000110 00000111 00001001 00001000 00011000 00011001
00011111 00011110 00111110 00111111 00111101 00111100 01111100 01111101 01111111 01111110
11111110 11111111 11111011 11111000 10111100 10111101 10111111 10111110 00111110 00111111
00111101 00111100 01111100 01111101 01111111 01111110 11111110 11111111 11111011 11111000
10111100 10111101 10111111 10111110 00111110 00111111 00111101 00111100 01111100 01111101
01111111 01111110 11111110 11111111 11111101 11111100 10111100 10111101 10111111 10111110
00111110 00111111 00111101 00111100 01111100 01111101 01111111 01111110 11111110 11111111
11111101 11111100 10111100 10111101 10111111 10111110 00111110 00111111 00111101 00111100
01111100 01111101 01111111 01111110 11111110 11111111 11111101 11111100 10111100 10111101
10111111 10111110 00111110 00111111 00111101 00111100 01111100 01111101 01111111 01111110
11111110 11111111 11111101 11111100 10111100 10111101 10111111 10111110 00111110 00111111
00111101 00111100 01111100 01111101 01111111 01111110 11111110 11111111 11111101 11111100
10111100 10111101 10111111 10111110 00111110 00111111 00111101 00111100

```



```

int main(){
while(1){

    //se pide el numero de bits y se imprime
    // k-bit (n = k)
    printf("\n");
    printf("\n");
    printf("ingresa el numero de bits: ");
    scanf("%d",&k);
    printf("\n");
    k1 = k;
    printf("%d-bit ",k);
    printf("(n = %d)",k1);
    printf("\n");
    //se manda llamar la funcion para imprimir
    //las cadenas
    grey(k);|
}

}

```

```

int grey(int bit){
    limit = 3;
    t = 1;
    //se inicializa un arreglo de longitud = bit.
    int chain[bit];
    //se llena con zeros, y se calcula el numero
    //de cadenas para el codigo "t" = 2^bit
    for(i = 0;i<bit;i++){
        chain[i] = 0;
        t = t<<1;
    }
    t1 = 1;
    m = bit- 1; // "m" se define como la ultima posicion del arreglo

    while(t>0){
        //se imprime el arreglo
        for(i=0;i<bit;i++){
            printf("%d",chain[i]);
        }
        printf(" ");
        //se invierte el bit de la posicion "m", que inicia
        //al final del arreglo, en el bit menos significativo
        if(t1<4){
            chain[m] = (2-(chain[m]<<1))>>1;//toggle de chain[m]
            m = m - 1;//se recorre al siguiente bit
        }
    }
}

```

```

//cuando t1 = 4, significa que se ha completado la secuencia
//00 01 11 10 en los dos ultimos bits de Cadena
if(t1 == 4 ){
    //entonces se invierte el siguiente bit, el tercero.
    //cuya posicion esta dada por bit-limit, donde limit
    //comienza con el valor de 3
    chain[bit-limit] = (2-(chain[bit-limit]<<1))>>1;

    //si limit es mayor que la cantidad de bits de la cadena
    //se le resta 2 para que pueda volver a incrementarse
    //cuando t1 vuelva a ser 4.
    if(limit<=bit){
        limit = limit + 1;//cuando se entre de nuevo al condicional
        //se invertira el siguiente bit
    }
    if(limit>bit){
        limit = limit - 2;
    }

    //se reinicia "m", pues este cambio no cuenta
    //dentro de la secuencia 00 01 11 10
    m = bit - 1;
    //lo mismo para t1, que se iguala a 0 para cancelar
    //el ultimo incremento y que comience en 1 para el sig ciclo.
    t1 = 0;
}

//cuando la posicion m rebasa los dos ultimos lugares del arreglo
//el bit menos significativo y el siguiente, se reinicia "m"
//para completar la secuencia 00 01 11 10
if(m<(bit-2)){
    m = bit - 1;
}
//t contiene el valor del numero de cadenas a imprimir
//el ciclo se detiene cuando t = 0.
t = t - 1;

//se incrementa en 1 el acumulador "t1"
t1 = t1 + 1;
}

```

Explicación de la Solución

Patrón de la secuencia

0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Cada vez que se completa la secuencia 00 01 11 10 se hace un toggle del siguiente Bit fuera de la secuencia, donde para que ocurra de nuevo un toggle, del Bit que le sigue al antes hecho, la secuencia se repite pero invertida (si comenzó como se mencionó sería 10 11 01 00). De esta forma la secuencia se repite y hace el toggle hasta que se alcanza el Bit más significativo, para luego ir en descenso con cada ocurrencia, cumpliendo la secuencia 00 01 11 10.

Conclusión:

Al integrar operaciones de tipo bitwise queda una perspectiva más amplia de lo que realmente sucede detrás de las funciones y operaciones matemáticas que comúnmente se utilizarían para resolver los ejercicios de la práctica, es decir que estas se traducen a un nivel más bajo, deconstruido, dotando de mayor control sobre el código al poder modificar con un detalle más profundo las operaciones; lo que muchas veces puede ser utilizado para encontrar soluciones más eficientes o específicas. Por otro entender mejor el funcionamiento de las funciones y operandos comunes ayuda a encontrar su mejor uso e integración en un programa.