

## Arquitectura de Computadores (AC)

### 1.1.1 Cuaderno de prácticas.

### 1.1.2 Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Antonio Javier Rodríguez Romero  
Grupo de prácticas y profesor de prácticas: Álvaro Martínez. DGIM1

## 2 Parte I. Ejercicios basados en los ejemplos del seminario práctico

1. (a) Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? (b) Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:** Salta un error al compilar al no estar `n` declarada dentro de la estructura de `parallel`. Podemos solucionarlo añadiéndole la cláusula `shared(n)`

**CAPTURA CÓDIGO FUENTE:** `shared-clauseModificado.c`

```
int main()
{
    int i, n = 7;
    int a[n];

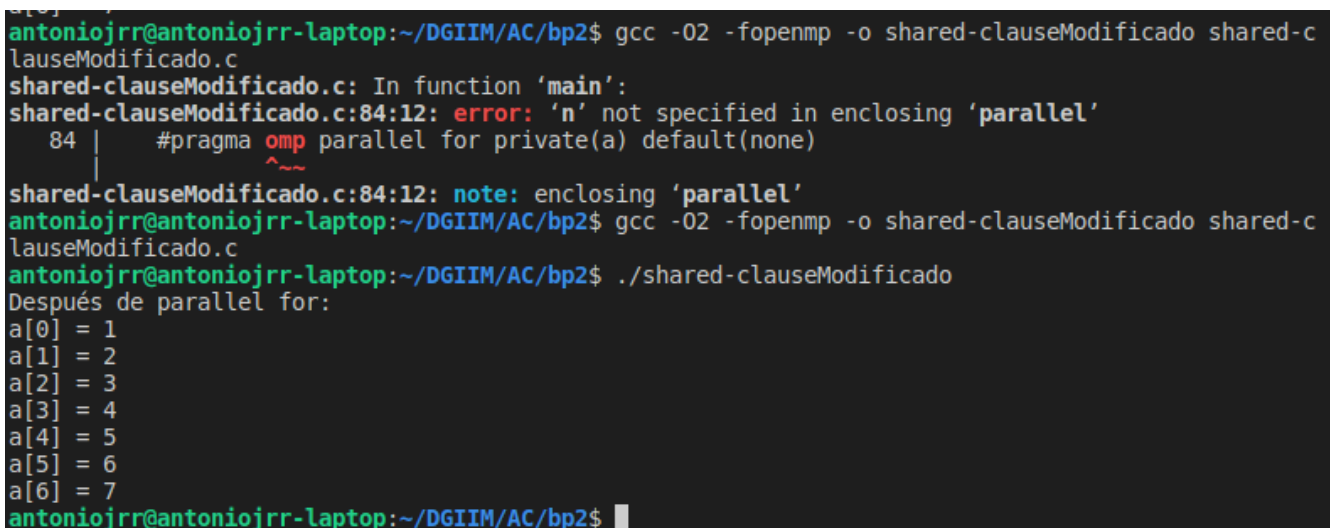
    for (i=0; i<n; i++)
        a[i] = i+1;

    // #pragma omp parallel for shared(a)
    #pragma omp parallel for private(a) default(none) shared(n)
    for (i=0; i<n; i++)
    {
        a[i] += i;
    }

    printf("Después de parallel for:\n");
    for (i=0; i<n; i++)
        printf("a[%d] = %d\n", i, a[i]);

    return(0);
}
```

**CAPTURAS DE PANTALLA:**



```
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:84:12: error: 'n' not specified in enclosing 'parallel'
  84 |     #pragma omp parallel for private(a) default(none)
      |           ^~
shared-clauseModificado.c:84:12: note: enclosing 'parallel'
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./shared-clauseModificado
Después de parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$
```

2. (a) Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar suma dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. (b) Modificar el código del apartado (a) para que se inicialice suma fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) **RESPUESTA:** Fuera del `parallel` siempre imprimirá 0 ya que las copias de suma que se hacen para cada hebra se descartarán una vez terminada la región de este.

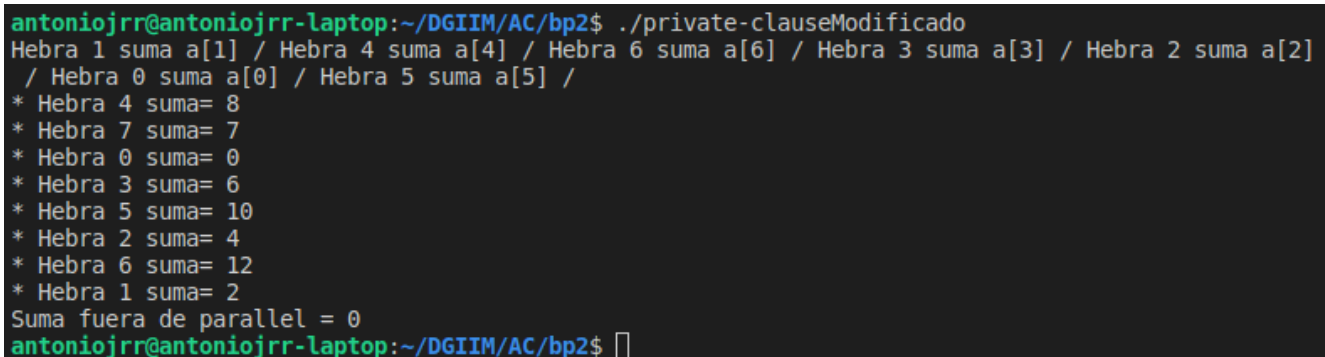
#### CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```
int main()
{
    int i, n = 7;
    int a[n], suma;

    for (i=0; i<n; i++)
        a[i] = i;

#pragma omp parallel private(suma)
{
    suma=omp_get_thread_num();
    #pragma omp for private(i)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf("Hebra %d suma a[%d] / ",
               omp_get_thread_num(), i);
    }
    printf("\n* Hebra %d suma= %d",
           omp_get_thread_num(), suma);
}
printf("\nSuma fuera de parallel = %d", suma);
printf("\n"); return(0);
}
```

#### CAPTURAS DE PANTALLA:



```
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./private-clauseModificado
Hebra 1 suma a[1] / Hebra 4 suma a[4] / Hebra 6 suma a[6] / Hebra 3 suma a[3] / Hebra 2 suma a[2]
/ Hebra 0 suma a[0] / Hebra 5 suma a[5] /
* Hebra 4 suma= 8
* Hebra 7 suma= 7
* Hebra 0 suma= 0
* Hebra 3 suma= 6
* Hebra 5 suma= 10
* Hebra 2 suma= 4
* Hebra 6 suma= 12
* Hebra 1 suma= 2
Suma fuera de parallel = 0
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$
```

(b) **RESPUESTA:** Como la clausula `private` solo reserva un espacio para la variable `suma` para cada `thread` pero no la inicializa, esta poseerá valores basura.

#### CAPTURA CÓDIGO FUENTE: `private-clauseModificado_b.c`

```
int main()
{
    int i, n = 7;
    int a[n], suma=0;

    for (i=0; i<n; i++)
        a[i] = i;

#pragma omp parallel private(suma)
{
    #pragma omp for private(i)
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf("Hebra %d suma a[%d] / ",
               omp_get_thread_num(), i);
    }
}
```

```

}
printf("\n* Hebra %d suma= %d",
      omp_get_thread_num(), suma);
}
printf("\nSuma fuera de parallel = %d", suma);
printf("\n"); return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./private-clauseModificado
Hebra 3 suma a[3] / Hebra 4 suma a[4] / Hebra 1 suma a[1] / Hebra 6 suma a[6] / Hebra 0 suma a[0]
/ Hebra 2 suma a[2] / Hebra 5 suma a[5] /
* Hebra 6 suma= 1805279942
* Hebra 7 suma= 1805279936
* Hebra 1 suma= 1805279937
* Hebra 5 suma= 1805279941
* Hebra 4 suma= 1805279940
* Hebra 3 suma= 1805279939
* Hebra 0 suma= 8
* Hebra 2 suma= 1805279938
Suma fuera de parallel = 0
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ 

```

3. (a) Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? (b) ¿A qué es debido?

**RESPUESTA:** Los resultados van variando con respecto a las ejecuciones debido a que múltiples *threads* tratan de acceder a una misma variable.

**CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c**

```

int main()
{
    int i, n = 7;
    int a[n], suma=0;

    for (i=0; i<n; i++)
        a[i] = i;

    #pragma omp parallel // private(suma)
    {
        #pragma omp for private(i)
        for (i=0; i<n; i++)
        {
            suma = suma + a[i];
            printf("Hebra %d suma a[%d] / ",
                  omp_get_thread_num(), i);

        }
        printf("\n* Hebra %d suma= %d",
              omp_get_thread_num(), suma);
    }
    printf("\nSuma fuera de parallel = %d", suma);
    printf("\n"); return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./private-clauseModificado
Hebra 1 suma a[1] / Hebra 0 suma a[0] / Hebra 2 suma a[2] / Hebra 4 suma a[4] / Hebra 6 suma a[6]
/ Hebra 5 suma a[5] / Hebra 3 suma a[3] /
* Hebra 4 suma= 8
* Hebra 3 suma= 8
* Hebra 5 suma= 8
* Hebra 6 suma= 8
* Hebra 0 suma= 8
* Hebra 7 suma= 8
* Hebra 2 suma= 8
* Hebra 1 suma= 8
Suma fuera de parallel = 8
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./private-clauseModificado
Hebra 0 suma a[0] / Hebra 1 suma a[1] / Hebra 4 suma a[4] / Hebra 2 suma a[2] / Hebra 5 suma a[5]
/ Hebra 3 suma a[3] / Hebra 6 suma a[6] /
* Hebra 4 suma= 5
* Hebra 5 suma= 5
* Hebra 7 suma= 5
* Hebra 3 suma= 5
* Hebra 0 suma= 5
* Hebra 1 suma= 5
* Hebra 2 suma= 5
* Hebra 6 suma= 5
Suma fuera de parallel = 5
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ 

```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

**(a) RESPUESTA:** Se imprime 9 porque es el valor que le asigna a `suma` por la última `thread` (`Thread 9`)

#### CAPTURAS DE PANTALLA:

```
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./firstlastprivate-clauseModificado
Hebra 2 suma a[4] suma=4
Hebra 5 suma a[7] suma=7
Hebra 0 suma a[0] suma=0
Hebra 0 suma a[1] suma=1
Hebra 1 suma a[2] suma=2
Hebra 1 suma a[3] suma=5
Hebra 3 suma a[5] suma=5
Hebra 6 suma a[8] suma=8
Hebra 7 suma a[9] suma=9
Hebra 4 suma a[6] suma=6

Fuera de la construcción parallel suma=9
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$
```

**(b) RESPUESTA:** Declarando el valor de cada uno de los valores del `array a`, podemos conseguir que la última hebra asigne un valor distinto a `suma`.

#### CAPTURAS DE PANTALLA:

```
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./firstlastprivate-clauseModificado
Hebra 2 suma a[4] suma=6
Hebra 0 suma a[0] suma=10
Hebra 0 suma a[1] suma=19
Hebra 1 suma a[2] suma=8
Hebra 1 suma a[3] suma=15
Hebra 3 suma a[5] suma=5
Hebra 4 suma a[6] suma=4
Hebra 5 suma a[7] suma=3
Hebra 7 suma a[9] suma=1
Hebra 6 suma a[8] suma=2

Fuera de la construcción parallel suma=1
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

**RESPUESTA:** Ocurre que solo se inicializará `a` en el valor introducido para la hebra que ejecute el `single`, para el resto se mantendrá el valor por defecto con el que comienza `a` (`=0`)

#### CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
int main()
{
    int n = 9;
    int i, b[n];

    for (i=0; i<n; i++)
        b[i] = -1;
#pragma omp parallel
{
    int a;
    #pragma omp single // copyprivate(a)
    {
        printf("Introduce valor de inicialización a: ");scanf("%d",&a);
        printf("Single ejecutada por la hebra %d\n",
            omp_get_thread_num());
    }
}
```

```

}

#pragma omp for
for (i=0; i<n; i++)
    b[i] = a;
}

printf("Depués de la región parallel:\n");
for (i=0; i<n; i++)
    printf(" b[%d] = %d\t",i,b[i]);
printf("\n");
return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./copy_private-clauseModificado
Introduce valor de inicialización a: 10
Single ejecutada por la hebra 3
Depués de la región parallel:
b[0] = 10    b[1] = 10    b[2] = 10    b[3] = 10    b[4] = 10    b[5] = 10
b[6] = 10    b[7] = 10    b[8] = 10
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ gcc -O2 -fopenmp -o copy_private-clauseModificado co
py_private-clauseModificado.c
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./copy_private-clauseModificado
Introduce valor de inicialización a: 10
Single ejecutada por la hebra 4
Depués de la región parallel:
b[0] = 0    b[1] = 0    b[2] = 0    b[3] = 0    b[4] = 0    b[5] = 10
b[6] = 0    b[7] = 0    b[8] = 0
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ █

```

6. En el ejemplo reduction-clause.c sustituya suma=0 por suma=10. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:** imprime el mismo de antes pero aumentándolo en 10.

**CAPTURA CÓDIGO FUENTE:** reduction-clauseModificado.c

```

int main(int argc, char **argv)
{
    int i, n=20;
    int a[n], suma=10;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

    for (i=0; i<n; i++) {
        a[i] = i;
    }

    #pragma omp parallel for default(none) private(i) shared(a,n) \
        reduction(+:suma)
    for (i=0; i<n; i++)
    {
        suma += a[i];
    }

    printf("Tras 'parallel' suma=%d\n",suma);
    return(0);
}

```

**CAPTURAS DE PANTALLA:**

```

antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./reduction-clauseModificado 16
Tras 'parallel' suma=120
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ gcc -O2 -fopenmp -o reduction-clauseModificado reduc
tion-clauseModificado.c
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./reduction-clauseModificado 16
Tras 'parallel' suma=130

```

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` de `#pragma omp parallel for` `reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo sin añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

**RESPUESTA:** En este caso el que se declare `suma` con el valor 10 provocará que el resultado aumente en 10 por cada hebra que ejecute el código.

**CAPTURA CÓDIGO FUENTE:** `reduction-clauseModificado7.c`

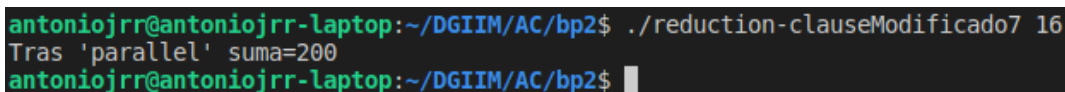
```
int main(int argc, char **argv)
{
    int i, n=20;
    int a[n], suma=10, sumafinal=0;
    if(argc < 2) {
        fprintf(stderr, "[ERROR]-Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}

    for (i=0; i<n; i++) {
        a[i] = i;
    }

    #pragma omp parallel default(none) private(i) shared(a,n,sumafinal) firstprivate(suma)
    {
        #pragma omp for
        for (i=0; i<n; i++)
        {
            suma += a[i];
        }
        #pragma omp atomic
        sumafinal+=suma;
    }

    printf("Tras 'parallel' suma=%d\n",sumafinal);
    return(0);
}
```

**CAPTURAS DE PANTALLA:**



```
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$ ./reduction-clauseModificado7 16
Tras 'parallel' suma=200
antoniojrr@antoniojrr-laptop:~/DGIIM/AC/bp2$
```

### 3 Parte II. Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

8. Implementar en paralelo el producto matriz por vector con OpenMP usando la directiva `for`. Partir del código secuencial disponible en SWAD. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- una segunda que paralelice el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

NOTAS: (1) el número de filas /columnas `N` de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, `v2`, para tamaños pequeños de los vectores (por ejemplo, `N = 8` y `N=11`); (4) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto

matriz vector, el número de hilos que usa y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

**CAPTURA CÓDIGO FUENTE :** pmv-OpenMP-a.c

**CAPTURA CÓDIGO FUENTE:** pmv-OpenMP-b.c

**RESPUESTA:**

**CAPTURAS DE PANTALLA:**

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

**CAPTURA CÓDIGO FUENTE:** pmv-OpenmMP-reduction.c

**RESPUESTA:**

**CAPTURAS DE PANTALLA:**

10. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar `-O2` al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

**CAPTURAS DE PANTALLA (que justifique el código elegido):**

**JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:**

**CAPTURA DE PANTALLA del script** pmv-OpenmMP-script.sh

**CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):**

**TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):**

**Tabla 1.** Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				atcgrid4				PC			
	Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000	
Nº de núcleos (p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)
<b>Código Secuencial</b>		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

**COMENTARIOS SOBRE LOS RESULTADOS:**