



INFORMATICA I

Ordenamiento de un vector por el metodo burbuja

Ing. Juan Carlos Cuttitta

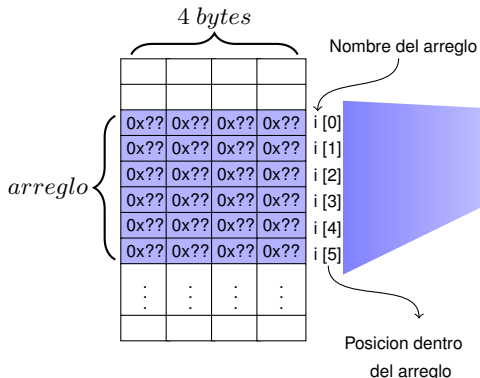
*Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Departamento de Ingeniería Electrónica*

14 de mayo de 2020

Declaración y disposición en memoria

Arquitectura X86-32 bits

Disposición de la variable *i*
en memoria



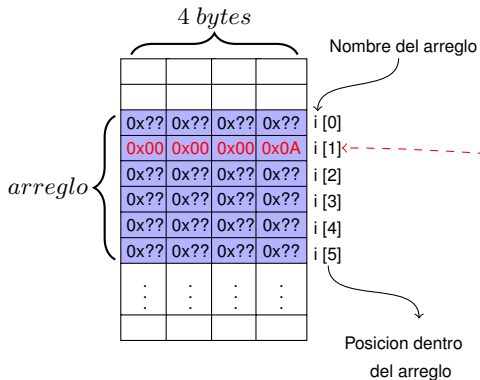
Código en programa fuente

```
1  int main ()
2  {
3      int i [6]; ← Declara arreglo
4      .....     de 6 enteros
5      i [1]=10;
6      .....
7  }
```

Acceso al contenido

Arquitectura X86-32 bits

Disposición de la variable *i*
en memoria



Código en programa fuente

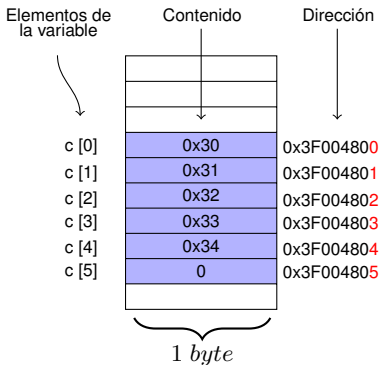
```
1  int main ()
2  {
3      int i [6];
4      .....
5      i [1]=10;
6      .....
7  }
```

Asigna el valor 10
al elemento 1
del arreglo

Dirección no es lo mismo que orden de elemento

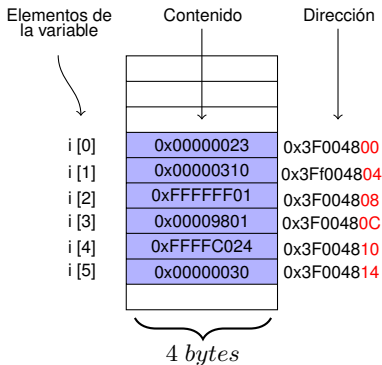
Declaración en código

`unsigned char c[6]`



Declaración en código

`int c[6]`



Dirección no es lo mismo que orden de elemento

Declaración en código `unsigned char c[6]`

Elementos de la variable

Contenido

Dirección

c [0]	0x30	0x3F004800
c [1]	0x31	0x3F004801
c [2]	0x32	0x3F004802
c [3]	0x33	0x3F004803
c [4]	0x34	0x3F004804
c [5]	0	0x3F004805

1 byte

*notar la diferencia
uno termina con '\0' y el otro no*

Declaración en código `int c[6]`

Elementos de la variable

Contenido

Dirección

i [0]	0x00000023	0x3F004800
i [1]	0x00000310	0x3F004804
i [2]	0xFFFFFFFF01	0x3F004808
i [3]	0x00009801	0x3F00480C
i [4]	0xFFFFC024	0x3F004810
i [5]	0x00000030	0x3F004814

4 bytes

Un caso especial de arreglo

Arreglo de caracteres

Se trata de un tipo muy usual de dato que llamamos cadena o string (del inglés).

Se inicializa de los siguientes modos:

```
1 /*Una forma de inicializar un arreglo de caracteres con una
   cadena*/
2
3 char cad []= "Hola mundo!";
4
5 /* Otra forma... */
6
7 char cad []={'H','o','l','a',' ','m','u','n','d','o','!','\0'};
```

Un caso especial de arreglo

Arreglo de caracteres

Se trata de un tipo muy usual de dato que llamamos cadena o string (del inglés).

Se inicializa de los siguientes modos:

```
1 /*Una forma de inicializar un arreglo de caracteres con una
   cadena*/
2
3 char cad []= "Hola mundo!";
4
5 /* Otra forma...*/
6
7 char cad []={ 'H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o', '!', '\0' };
```

En código....

```
1  /* Utiliza una lista de inicialización para
   2     inicializar el arreglo arr.*/
3  int arr[8] = {23,0,-669,-1,995,1277,90,-9000};
4  /* La cantidad de elementos en la línea anterior
   5     es redundante. Puede hacerse lo mismo de la
   6     siguiente forma*/
7  int arr[] = {23,0,-669,-1,995,1277,90,-9000};
8  /* Si lo vamos a inicializar con el mismo valor
   9     para todos los elementos, la forma adecuada
  10     es la siguiente*/
11
12 int arr[8], i;
13 for (i = 0 ; i < 8 ; i++)
14 {
15     arr[i] = 0;
16 }
```


Ordenamiento de un vector por el metodo *burbuja*

MEMORIA en
ARQUITECTURA x86 32-bits

0x7FFE6E03					0x7FFE6E00
0x7FFE6E07					
0x7FFE6E0B					
0x7FFE6E0F					
0x7FFE6E13					
0x7FFE6E17					
0x7FFE6E1B					
0x7FFE6E1F					
0x7FFE6E23					
0x7FFE6E27					
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF					0xFFFFFFFF

```
1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0;i < (CANT-resto);i++)
13         {
14             if (vInt [i] > vInt [i+1])
15             {
16                 aux=vInt [i];
17                 vInt [i]=vInt [i+1];
18                 vInt [i+1]=aux;
19                 flag=1;
20             }
21         }
22     }while(flag);
23     return (0);
24 }
```

MEMORIA en ARQUITECTURA x86 32-bits

contienen
cualquier valor
inicialmente

0xXX	0xXX	0xXX	0xXX
0xXX	0xXX	0xXX	0xXX
0xXX	0xXX	0xXX	0xXX
0xXX	0xXX	0xXX	0xXX
⋮	⋮	⋮	⋮

← aux

← resto

← i

← flag

```
1 #include <stdio.h>
2 #define CANT 6
3
4 int main (void)
5 {
6   int aux, resto, i, flag, vInt[CANT]={2,286,9,59,928,3};
7   resto=0;
8   do
9   {
10    resto++;
11    flag=0;
12    for(i=0; i < (CANT-resto); i++)
13    {
14      if (vInt[i] > vInt[i+1])
15      {
16        aux=vInt[i];
17        vInt[i]=vInt[i+1];
18        vInt[i+1]=aux;
19        flag=1;
20      }
21    }
22  } while(flag);
23  return (0);
24 }
```

0xFFFFFFFF

0xFFFFFFFFC

MEMORIA en ARQUITECTURA x86 32-bits

0xXX	0xXX	0xXX	0xXX	aux
0xXX	0xXX	0xXX	0xXX	resto
0xXX	0xXX	0xXX	0xXX	i
0xXX	0xXX	0xXX	0xXX	flag
0x00	0x00	0x00	0x02	vInt[0]
0x00	0x00	0x01	0x1E	vInt[1]
0x00	0x00	0x00	0x09	vInt[2]
0x00	0x00	0x00	0x3B	vInt[3]
0x00	0x00	0x03	0xA0	vInt[4]
0x00	0x00	0x00	0x03	vInt[5]
⋮	⋮	⋮	⋮	
0x00	0x00	0x00	0x00	0xFFFFFC

contienen
valores
iniciales

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

0xFFFFFFFF

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x00	← resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0xXX	0xXX	0xXX	0xXX	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0xXX	0xXX	0xXX	0xXX	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0;i < (CANT-resto);i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0xXX	0xXX	0xXX	0xXX	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the initialization of the array `vInt` and the execution of a bubble sort algorithm. A red dashed arrow points from the `i` variable in the code to the memory location `0x7FFE6E0B`. A red bracket highlights the calculation `6-1=5` in the `for` loop condition, indicating the number of elements to compare.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- Red dashed arrows point from the code to the memory table:
 - Line 10: `vInt[0]` points to the 5th column of the 5th row.
 - Line 12: `vInt[1]` points to the 4th column of the 6th row.
 - Line 14: `vInt[i]` points to the 4th column of the 7th row.
 - Line 15: `vInt[i+1]` points to the 5th column of the 7th row.
- A red bracket above line 12 indicates the calculation $6-1=5$.
- Red dashed arrows labeled '0' and '1' point to the `if` condition and the swap block, respectively.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the initialization of the array `vInt` and the execution of a bubble sort algorithm. A red dashed arrow points from the `for` loop condition `i < (CANT-resto)` to the `0x01` value in the `resto` memory location. A red bracket highlights the calculation `6-1=5` in the `for` loop condition, indicating the current range of indices being compared.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0xXX	0xXX	0xXX	0xXX	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-1=5$, representing the number of elements in the array after the first element is excluded.
- Red dashed arrows point from the code to the memory table:
 - Line 11 points to the `resto` variable (0x00).
 - Line 12 points to the `for` loop condition.
 - Line 14 points to the `if` statement.
 - Line 15 points to the `if` block.
 - Line 16 points to the `aux` variable (0x00).
 - Line 17 points to the `vInt[i]` element (0x00).
 - Line 18 points to the `vInt[i+1]` element (0x03).
 - Line 19 points to the `flag` variable (0x00).
- Red numbers 1 and 2 are placed near the swap logic in lines 16-18.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x01	0x1E	- vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Annotation: $6-1=5$ (underlined in red) is shown above the for loop condition.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-1=5$

$vInt[1] = vInt[2]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-1=5$
 vInt[2] = aux

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The variables aux, resto, i, and flag are located at 0x7FFE6E03, 0x7FFE6E07, 0x7FFE6E0B, and 0x7FFE6E0F respectively. The array vInt is located at 0x7FFE6E13. The code snippet shows the initialization of the array and the execution of the bubble sort algorithm. A red dashed arrow points from the flag variable to the while loop condition. A red bracket highlights the calculation (CANT-resto) = 6-1=5 in the for loop condition.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the code is on the right. A red dashed arrow points from the code line `for(i=0; i < (CANT-resto); i++)` to the memory address `0x7FFE6E0B`, which contains the value `0x02`. A red bracket above the code line indicates the calculation `6-1=5`, showing that the loop condition is `i < 5`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for (i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-1=5$.
- A red dashed arrow labeled '2' points from the expression $(CANT-resto)$ to the loop condition $i < (CANT-resto)$.
- A red dashed arrow labeled '3' points from the expression $vInt [i+1]$ to the comparison $vInt [i] > vInt [i+1]$.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x01	0x1E	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Annotations in the code:

- A red bracket above the `(CANT-resto)` expression in line 12 is labeled `6-1=5`.
- A blue number `2` is placed above the opening curly brace of the inner `if` statement in line 15.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-1=5$
 $vInt[2] = vInt[3]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-1=5$
 $vInt[3] = aux$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses and their corresponding values are shown in the table on the left. The code on the right shows the logic for sorting an array of integers. A red dashed arrow points from the `vInt[5]` label in the code to the `0x03` value in the memory cell at address `0x7FFE6E27`. A red bracket highlights the calculation `6-1=5` in the `for` loop condition, indicating the range of indices being compared.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the code. A red dashed arrow points from the `for` loop condition `(CANT-resto)` in the code to the `0x03` value in the memory cell at address `0x7FFE6E0B`. A red bracket under `(CANT-resto)` in the code is labeled `6-1=5`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-1=5$.
- A red arrow labeled '3' points from the condition $vInt[i] > vInt[i+1]$ to the swap operation on line 17.
- A red arrow labeled '4' points from the swap operation on line 17 to the next iteration of the loop.
- Red dashed arrows point from the memory addresses `vInt[3]` and `vInt[4]` in the memory table to the corresponding indices in the code.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The code defines an array `vInt` of size `CANT` (6) and iterates through it to swap adjacent elements. The memory addresses shown in the table correspond to the variables and array elements in the code. A red dashed arrow points from the `for` loop condition `i < (CANT-resto)` to the `0x7FFE6E0B` memory location, which contains the value `0x04`. A red bracket highlights the calculation `6-1=5` in the loop condition, indicating the current value of `resto`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	← vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	← vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-1=5$.
- Red dashed arrows point from the code to the memory table:
 - Arrow 4: from `vInt[i]` to the value 0x03 in vInt[4] (0x7FFE6E23).
 - Arrow 5: from `vInt[i+1]` to the value 0x00 in vInt[5] (0x7FFE6E27).

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	← aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x03	0xA0	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Annotations in the code:

- A red bracket above the `(CANT-resto)` expression in line 12 is labeled `6-1=5`.
- A blue number `4` is placed above the opening curly brace of the inner `if` statement in line 15.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x00	0x03	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-1=5$

$vInt[4] = vInt[5]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red dashed arrow points from the `aux` variable label to the value `0xA0` in the memory cell at address `0x7FFE6E27`.
- A red bracket above the `for` loop condition `(CANT-resto)` is labeled `6-1=5`.
- A red dashed arrow points from the `vInt[5]` label to the value `0xA0` in the memory cell at address `0x7FFE6E27`.
- A blue label `vInt[5] = aux` is placed to the right of the code, with a red dashed arrow pointing to the assignment `vInt[i+1]=aux;` in line 18.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux=0xA0, resto=0x01, i=0x04, flag=0x01, vInt[0]=0x02, vInt[1]=0x09, vInt[2]=0x3B, vInt[3]=0x1E, vInt[4]=0x03, vInt[5]=0xA0. A red dashed arrow points from the flag variable to the for loop condition. A red bracket highlights the calculation $6-1=5$ in the for loop condition, indicating the number of elements to compare.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x01	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the initialization of the array `vInt` and the execution of a loop that sorts the array. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the value `0x05` in the memory table, indicating that `resto` is 5. A red bracket highlights the calculation `6-1=5` in the code, showing that `CANT` is 6 and `resto` is 1.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The variable `resto` is highlighted in red in the table, and its value is updated in the code. A red dashed arrow points from the `resto` variable in the code to the `resto` variable in the table. A red bracket highlights the calculation `6-1=5` in the code, indicating the number of elements to be compared in the current pass.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x05	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux (0x00, 0x00, 0x03, 0xA0), resto (0x00, 0x00, 0x00, 0x02), i (0x00, 0x00, 0x00, 0x05), flag (0x00, 0x00, 0x00, 0x00), vInt[0] (0x00, 0x00, 0x00, 0x02), vInt[1] (0x00, 0x00, 0x00, 0x09), vInt[2] (0x00, 0x00, 0x00, 0x3B), vInt[3] (0x00, 0x00, 0x01, 0x1E), vInt[4] (0x00, 0x00, 0x00, 0x03), vInt[5] (0x00, 0x00, 0x03, 0xA0). The values are shown in a grid with colored cells. A red dashed arrow points from the 'flag' row to the 'for' loop in the code. A red bracket highlights the expression $6-1=5$ in the code, indicating the number of elements to be compared in the current pass.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, including the definition of the array `vInt` and the sorting algorithm. A red dashed arrow points from the `i` variable in the code to the memory location `0x7FFE6E0B`. A red bracket highlights the calculation `6-2=4` in the `for` loop, indicating the number of elements to be compared.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- Red dashed arrows point from `vInt[0]` and `vInt[1]` in the code to the corresponding memory addresses in the table.
- A red bracket above line 12 indicates the calculation $6-2=4$.
- Red dashed arrows point from the `>` operator in line 14 to the values `0` and `1` in the table, representing the comparison result.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, including the definition of the array `vInt` and the sorting algorithm. A red dashed arrow points from the `for` loop condition `i < (CANT-resto)` to the memory address `0x7FFE6E0B`, which contains the value `0x01`. A red bracket under the expression `(CANT-resto)` indicates the calculation `6-2=4`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	← vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-2=4$.
- Red dashed arrows point from line 12 to vInt[1] and from line 14 to vInt[2].
- Red dashed arrows point from line 15 to vInt[i] (labeled '1') and from line 17 to vInt[i+1] (labeled '2').

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, with a red dashed arrow indicating the relationship between the memory address 0x7FFE6E0B (labeled 'i') and the loop condition in the code: $i < (CANT - resto)$. A red bracket highlights the calculation $6 - 2 = 4$, which corresponds to the value of $(CANT - resto)$ at that point in the execution.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for (i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-2=4$.
- A red dashed arrow labeled '2' points from the `i` variable in the `for` loop to the `vInt[2]` memory address.
- A red dashed arrow labeled '3' points from the `i+1` expression in the `for` loop to the `vInt[3]` memory address.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, with a red dashed arrow indicating the relationship between the memory address 0x7FFE6E0B (where `i` is located) and the `for` loop condition `i < (CANT - resto)`. A red bracket highlights the calculation `6 - 2 = 4`, which corresponds to the value of `resto` at that point in the execution.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x03	0xA0	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-2=4$, representing the number of iterations for the inner loop when `resto=2`.
- A red dashed arrow points from the `if` condition on line 14 to the `vInt[3]` memory location in the table.
- A red dashed arrow points from the `vInt[i]` assignment on line 16 to the `vInt[4]` memory location in the table.
- A red dashed arrow points from the `vInt[i+1]` assignment on line 17 to the `vInt[5]` memory location in the table.
- A red dashed arrow points from the `vInt[i+1]` assignment on line 18 to the `vInt[4]` memory location in the table.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x01	0x1E	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Annotations in the code:

- Line 12: $6-2=4$ (indicated by a red bracket above the expression)
- Line 15: 3 (indicated by a blue number above the opening curly brace)

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x00	0x03	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-2=4$

$vInt[3] = vInt[4]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	← aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	← vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

6-2=4

vInt[4] = aux

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux (0x00, 0x00, 0x01, 0x1E), resto (0x00, 0x00, 0x00, 0x02), i (0x00, 0x00, 0x00, 0x03), flag (0x00, 0x00, 0x00, 0x01), vInt[0] (0x00, 0x00, 0x00, 0x02), vInt[1] (0x00, 0x00, 0x00, 0x09), vInt[2] (0x00, 0x00, 0x00, 0x3B), vInt[3] (0x00, 0x00, 0x00, 0x03), vInt[4] (0x00, 0x00, 0x01, 0x1E), vInt[5] (0x00, 0x00, 0x03, 0xA0). The values are shown in a grid with colored backgrounds. A red dashed arrow points from the flag value 0x01 to the code line 12. A red bracket above the code line 12 indicates the calculation 6-2=4.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x02	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the execution of the program, with a red dashed arrow indicating the relationship between the memory address 0x7FFE6E0B (labeled 'i') and the loop condition in the code: `for(i=0; i < (CANT-resto); i++)`. A red bracket highlights the calculation `6-2=4`, indicating the value of `(CANT-resto)` when `resto` is 2.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. A red dashed arrow points from the `resto` variable in the code to the memory cell at address `0x7FFE6E07`, which contains the value `0x03`. A red bracket above the `for` loop condition `(CANT-resto)` indicates the calculation `6-2=4`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x04	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux (0x00, 0x00, 0x01, 0x1E), resto (0x00, 0x00, 0x00, 0x03), i (0x00, 0x00, 0x00, 0x04), flag (0x00, 0x00, 0x00, 0x00), vInt[0] (0x00, 0x00, 0x00, 0x02), vInt[1] (0x00, 0x00, 0x00, 0x09), vInt[2] (0x00, 0x00, 0x00, 0x3B), vInt[3] (0x00, 0x00, 0x00, 0x03), vInt[4] (0x00, 0x00, 0x01, 0x1E), vInt[5] (0x00, 0x00, 0x03, 0xA0). The values are shown in a grid with colored backgrounds. A red dashed arrow points from the 'flag' row to the 'do' loop in the code. A red bracket above the 'for' loop indicates the calculation $6-2=4$.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The code defines an array `vInt` of size `CANT` (6) and iterates through it. A red dashed arrow points from the `for` loop condition `i < (CANT-resto)` to the memory address `0x7FFE6E0B` (labeled `i`), indicating the current value of `i`. A red bracket above the expression `(CANT-resto)` shows the calculation `6-3=3`, indicating the number of elements remaining in the array.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- Red dashed arrows point from `vInt[0]` and `vInt[1]` in the code to the corresponding memory addresses in the table.
- A red bracket above line 12 indicates the calculation $6-3=3$.
- A red bracket below line 15 indicates the calculation $0-1$.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are listed on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, including the definition of the array `vInt` and the sorting algorithm. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the value `3` in the table, which is calculated as `6-3=3`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	← vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-3=3$.
- Red dashed arrows point from the code to the memory table:
 - Line 11 points to the `flag` variable.
 - Line 12 points to the `for` loop condition.
 - Line 14 points to the `if` statement.
 - Line 15 points to the `if` block.
 - Line 16 points to the `aux` variable.
 - Line 17 points to the first swap operation.
 - Line 18 points to the second swap operation.
 - Line 19 points to the `flag` variable.
- Red dashed arrows also point from the memory table to the code:
 - From `vInt[1]` to line 12.
 - From `vInt[2]` to line 14.
 - From `vInt[3]` to line 16.
 - From `vInt[4]` to line 17.
 - From `vInt[5]` to line 18.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, including the definition of the array `vInt` and the sorting algorithm. A red dashed arrow points from the `i` variable in the code to the `0x02` value in the memory table at address `0x7FFE6E0B`. A red bracket highlights the calculation `6-3=3` in the `for` loop condition, indicating the number of elements remaining to be sorted.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x01	0x1E	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	<-vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	<-vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for (i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-3=3$.
- A red dashed arrow labeled '2' points from the expression $(CANT-resto)$ to the value 3 in the loop condition.
- A red dashed arrow labeled '3' points from the expression $vInt [i+1]$ to the value 3 in the array access.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	← aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x3B	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Annotations in the code:

- A red bracket above the `(CANT-resto)` expression in line 12 is labeled `6-3=3`.
- A blue number `2` is placed above the opening curly brace of the inner `if` statement in line 15.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x03	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-3=3$
 $vInt[2] = vInt[3]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-3=3$
 $vInt[3] = aux$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The variables aux, resto, i, and flag are located at 0x7FFE6E03, 0x7FFE6E07, 0x7FFE6E0B, and 0x7FFE6E0F respectively. The array vInt is located at 0x7FFE6E13. The code snippet shows the initialization of the array and the start of the bubble sort algorithm. A red dashed arrow points from the 'flag' variable to the 'if' condition in the code. A red bracket highlights the calculation $6-3=3$ in the for loop condition, indicating the number of elements to compare.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x03	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, including the definition of the array `vInt` and the sorting algorithm. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the value `3` in the table, which is calculated as `6-3=3`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux=0x00, resto=0x00, i=0x00, flag=0x01, vInt[0]=0x00, vInt[1]=0x00, vInt[2]=0x00, vInt[3]=0x00, vInt[4]=0x01, vInt[5]=0x03. A red dashed arrow points from the 'resto' variable to the 'for' loop condition in the code, where a calculation $6-3=3$ is shown, indicating the current number of elements to be compared.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x03	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, with a red arrow pointing from the `flag` variable in the code to the `flag` memory location in the table. A red bracket highlights the calculation `6-3=3` in the `for` loop, indicating the number of iterations.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses and their contents are shown in the table on the left. The code on the right shows the initialization of variables and a loop that iterates over the array `vInt`. A red dashed arrow points from the `i` variable in the code to the memory location `0x7FFE6E0B` in the table. A red bracket highlights the calculation `6-4=2` in the `for` loop condition, indicating the number of elements remaining in the array.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- Red dashed arrows point from code lines to memory addresses:
 - Line 10 points to vInt[0]
 - Line 12 points to vInt[1]
 - Line 14 points to vInt[2]
 - Line 15 points to vInt[3]
 - Line 16 points to vInt[4]
 - Line 17 points to vInt[5]
- A red bracket above line 12 indicates the calculation $6-4=2$.
- Red dashed arrows labeled '0' and '1' point to the comparison and swap logic in lines 14-15.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. A red dashed arrow points from the `for` loop condition in the code to the `0x7FFE6E0B` memory location, which contains the value `0x01`. A red bracket under the expression `(CANT-resto)` in the code is labeled `6-4=2`, indicating the current range of the inner loop.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x3B	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	← vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt [i] > vInt [i+1])
15             {
16                 aux=vInt [i];
17                 vInt [i]=vInt [i+1];
18                 vInt [i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram annotations:

- A red bracket above line 12 indicates the calculation $6-4=2$, representing the number of iterations for the inner loop when $i=0$ and $resto=4$.
- Red dashed arrows point from the code to the memory table:
 - Line 11 points to the `flag` variable.
 - Line 12 points to the `for` loop condition.
 - Line 14 points to the `if` statement.
 - Line 15 points to the swap logic.
 - Line 16 points to the assignment `aux=vInt[i]`.
 - Line 17 points to the assignment `vInt[i]=vInt[i+1]`.
 - Line 18 points to the assignment `vInt[i+1]=aux`.
 - Line 19 points to the assignment `flag=1`.
- Red numbers 1 and 2 are placed near the swap logic in lines 16 and 17, respectively.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	← aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x09	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Annotation: $6-4=2$ (under the expression $(CANT-resto)$ in line 12)

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x03	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

$6-4=2$

$vInt[1] = vInt[2]$

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	← aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	← vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

6-4=2

vInt[2] = aux

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux (0x09), resto (0x04), i (0x01), flag (0x01), vInt[0] (0x02), vInt[1] (0x03), vInt[2] (0x09), vInt[3] (0x3B), vInt[4] (0x1E), vInt[5] (0xA0). A red dashed arrow points from the flag variable to the loop condition in the code. A red bracket highlights the calculation $6-4=2$ in the for loop condition, indicating the number of elements to be compared in the current pass.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x04	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	< i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the logic of the program, with a red dashed arrow indicating the relationship between the code and the memory state. A red bracket highlights the calculation $6-4=2$ in the `for` loop, indicating the number of elements remaining to be processed.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x01	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0; i < (CANT-resto); i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses are shown on the left, and the corresponding values are shown in the table. The code on the right shows the initialization of the array `vInt` and the execution of a sorting algorithm. A red dashed arrow points from the `resto` variable in the code to the `0x05` value in the memory table. A red bracket highlights the calculation `6-4=2` in the `for` loop, indicating the current value of `resto`.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x02	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10         resto++;
11         flag=0;
12         for(i=0;i < (CANT-resto);i++)
13         {
14             if (vInt[i] > vInt[i+1])
15             {
16                 aux=vInt[i];
17                 vInt[i]=vInt[i+1];
18                 vInt[i+1]=aux;
19                 flag=1;
20             }
21         }
22     } while (flag);
23     return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the bubble sort algorithm. The memory addresses are shown on the left, and the corresponding values are shown in the table. The values are: aux (0x09), resto (0x05), i (0x02), flag (0x00), vInt[0] (0x02), vInt[1] (0x03), vInt[2] (0x09), vInt[3] (0x3B), vInt[4] (0x1E), vInt[5] (0xA0). The values are sorted in ascending order. A red dashed arrow points from the code line 11 (flag=0) to the flag cell in the memory table. A red bracket highlights the calculation 6-4=2 in the code line 12 (for(i=0;i < (CANT-resto);i++)), indicating the number of elements to be compared in the current pass.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses and their contents are shown in the table on the left. The code on the right shows the initialization of variables and a loop that iterates over the array `vInt`. A red dashed arrow points from the `i` variable in the code to the memory location `0x7FFE6E0B` (value `0x00`). A red bracket above the `for` loop condition `(CANT-resto)` indicates the calculation `6-5=1`, which corresponds to the value `0x01` in the `vInt[4]` memory location.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x00	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	< vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	< vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0; i < (CANT-resto); i++)
13        {
14            if (vInt[i] > vInt[i+1])
15            {
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    return (0);
24 }

```

Diagram annotations:

- Red dashed arrows point from `vInt[0]` and `vInt[1]` in the code to the corresponding memory addresses in the table.
- A red bracket above line 12 indicates `6-5=1`, representing the calculation of `CANT-resto`.
- A red bracket below line 15 indicates `0` and `1`, representing the indices `i` and `i+1` in the swap logic.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	< i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
0x7FFE6E2B					
0x7FFE6E2F					
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6 int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7 resto=0;
8 do
9 {
10 resto++;
11 flag=0;
12 for(i=0; i < (CANT-resto); i++)
13 {
14     if (vInt[i] > vInt[i+1])
15     {
16         aux=vInt[i];
17         vInt[i]=vInt[i+1];
18         vInt[i+1]=aux;
19         flag=1;
20     }
21 }
22 } while (flag);
23 return (0);
24 }

```

Diagram illustrating the memory layout and the execution of the provided C code. The memory addresses and their contents are shown in the table on the left. The code on the right shows the execution of a bubble sort algorithm. A red dashed arrow points from the code to the memory table, indicating the state of the array during execution. The code defines a constant `CANT` as 6 and initializes an array `vInt` with values {2, 286, 9, 59, 928, 3}. The code uses a `do-while` loop to repeatedly sort the array until it is sorted. The `for` loop iterates over the array, comparing adjacent elements and swapping them if they are out of order. The `flag` variable is used to indicate when the array is sorted. The `resto` variable is used to track the current position in the array. The `aux` variable is used to store the value of the element being swapped. A red bracket highlights the calculation `6-5=1` in the `for` loop, indicating the number of elements to be compared in the current iteration.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
	0x00	0x00	0x00	0x05	resto
	0x00	0x00	0x00	0x01	i
	0x00	0x00	0x00	0x00	flag
2	0x00	0x00	0x00	0x02	vInt[0]
3	0x00	0x00	0x00	0x03	vInt[1]
9	0x00	0x00	0x00	0x09	vInt[2]
59	0x00	0x00	0x00	0x3B	vInt[3]
286	0x00	0x00	0x01	0x1E	vInt[4]
928	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 int main (void)
5 {
6     int aux, resto , i , flag , vInt [CANT]={2,286,9,59,928,3};
7     resto=0;
8     do
9     {
10        resto++;
11        flag=0;
12        for(i=0;i < (CANT-resto);i++)
13        {
14            if (vInt [i] > vInt [i+1])
15            {
16                aux=vInt [i];
17                vInt [i]=vInt [i+1];
18                vInt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag); <-----
23    return (0);
24 }

```

Con while(0) sale del loop do-while y termina

Sugerimos ver la
continuación de este
documento una vez que
se tenga los
conocimientos de
PUNTEROS

Verificar valores con la función *imprimir*

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	vlnt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vlnt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vlnt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vlnt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vlnt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vlnt[5]
	⋮	⋮	⋮	⋮	
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```
1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vlnt [CANT]={2,286,9,59,928,3};
9     imprimir(&vlnt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if (vlnt [i] > vlnt [i+1]){
16                aux=vlnt [i];
17                vlnt [i]=vlnt [i+1];
18                vlnt [i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vlnt [0], CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for (j=0;j<n;j++){
30         printf ("%d \r\n", *(p+j));
31     }
32 }
```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003					
0xA0004007	0x00	0x00	0x00	0x06	← n
0xA000400B	0x7F	0xFE	0x6E	0x10	← p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf( "%d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0xXX	0xXX	0xXX	0xXX	← j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x00	← j - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for (j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 ← vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x00	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag,vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```


MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x01	<- j - - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	0x7FFE6E14 vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x01	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux,resto,i,flag,vInt[CANT]={2,286,9,59,928,3};
9     imprimir(&vInt[0],CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x02	<- j - - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	0x7FFE6E18 vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x02	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j < n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x03	<- j - - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	0x7FFE6E1C vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x03	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt[0],CANT);
24    return (0);
25 }
26 void imprimir( int*p,int n)
27 {
28     int j;
29     for(j=0;j<n;j++){
30         printf("%d \r\n",*(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x04	<- j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	0x7FFE6E20 vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x04	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT 6
3
4 void imprimir(int *p,int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0;i < (CANT-resto);i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    } while (!flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p,int n)
27 {
28     int j;
29     for (j=0;j<n;j++){
30         printf(" %d \r\n",*(p+j));
31     }
32 }

```


MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x05	<- j - - - - -
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	0x7FFE6E24 vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x05	j
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFFC

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j < n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03	0x00	0x00	0x00	0x09	aux
0x7FFE6E07	0x00	0x00	0x00	0x05	resto
0x7FFE6E0B	0x00	0x00	0x00	0x01	i
0x7FFE6E0F	0x00	0x00	0x00	0x00	flag
0x7FFE6E13	0x00	0x00	0x00	0x02	0x7FFE6E10 vInt[0]
0x7FFE6E17	0x00	0x00	0x00	0x03	vInt[1]
0x7FFE6E1B	0x00	0x00	0x00	0x09	vInt[2]
0x7FFE6E1F	0x00	0x00	0x00	0x3B	vInt[3]
0x7FFE6E23	0x00	0x00	0x01	0x1E	vInt[4]
0x7FFE6E27	0x00	0x00	0x03	0xA0	vInt[5]
	⋮	⋮	⋮	⋮	
0xA0004003	0x00	0x00	0x00	0x06	<- j - - - - - n
0xA0004007	0x00	0x00	0x00	0x06	n
0xA000400B	0x7F	0xFE	0x6E	0x10	p
0xFFFFFFFF	0x00	0x00	0x00	0x00	0xFFFFFFFF

```

1 #include <stdio.h>
2 #define CANT      6
3
4 void imprimir( int *p, int n);
5
6 int main (void)
7 {
8     int aux, resto, i, flag, vInt [CANT]={2,286,9,59,928,3};
9     imprimir(&vInt [0], CANT);
10    resto=0;
11    do{
12        resto++;
13        flag=0;
14        for(i=0; i < (CANT-resto); i++){
15            if(vInt[i] > vInt[i+1]){
16                aux=vInt[i];
17                vInt[i]=vInt[i+1];
18                vInt[i+1]=aux;
19                flag=1;
20            }
21        }
22    }while(flag);
23    imprimir(&vInt [0], CANT);
24    return (0);
25 }
26 void imprimir( int *p, int n)
27 {
28     int j;
29     for(j=0; j<n; j++){
30         printf(" %d \r\n", *(p+j));
31     }
32 }

```

```
carlos@carlos-R430-R480-R440: ~  
carlos@carlos-R430-R480-R440:~$ gcc -c burbuja.c -o burbuja.o -Wall  
carlos@carlos-R430-R480-R440:~$ gcc burbuja.o -o burbuja -Wall  
carlos@carlos-R430-R480-R440:~$ ./burbuja  
  
2      286      9      59      928      3  
  
2      3      9      59      286      928  
carlos@carlos-R430-R480-R440:~$
```

- compila con ***gcc -c burbuja.c -o burbuja.o -Wall***
- Linkea con ***gcc burbuja.o -o burbuja -Wall***
- Ejecuta con ***./burbuja***
- Primero imprime en la consola los valores desordenados
- Al finalizar el ordenamiento también los imprime en la consola