



INFORMATICA I

Ordenamiento de un vector por el metodo burbuja

Ing. Juan Carlos Cuttitta

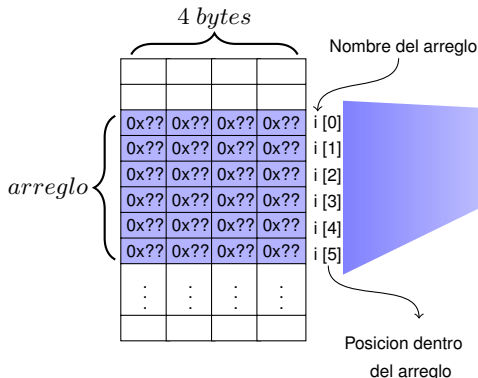
*Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Departamento de Ingeniería Electrónica*

4 de junio de 2020

Declaración y disposición en memoria

Arquitectura X86-32 bits

Disposición de la variable **i**
en memoria



Código en programa fuente

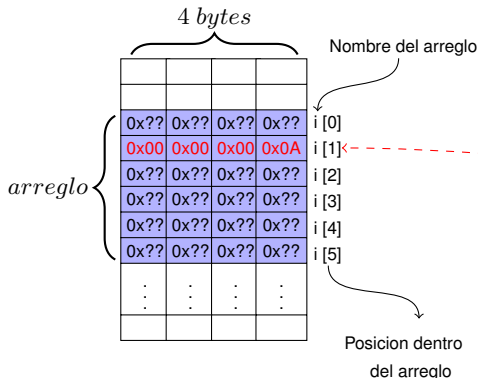
```
1  int main ()
2  {
3      int i[6];
4      .....
5      i[1]=10;
6      .....
7  }
```

Declara arreglo
de 6 enteros

Acceso al contenido

Arquitectura X86-32 bits

Disposición de la variable **i** en memoria



Código en programa
fuente

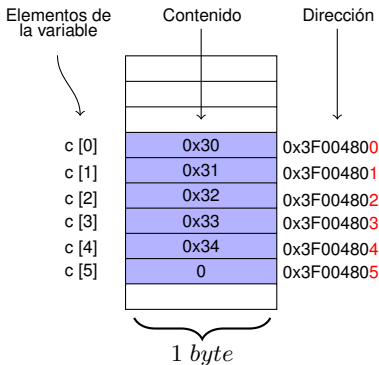
```
1  int main ()
2  {
3      int i[6];
4      .....
5  - - - i[1]=10;
6      .....
7  }
```

Asigna el valor 10 al elemento 1 del arreglo

Dirección no es lo mismo que orden de elemento

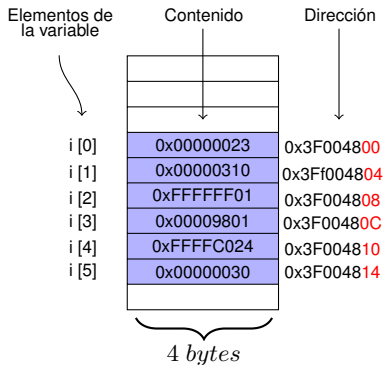
Declaración en código

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


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					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 }
```

contienen
cualquier valor
inicialmente

0xFFFFFFFF

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

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	

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

0xFFFFFC

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 selection sort algorithm. The memory layout shows the stack frame with variables: aux, resto, i, flag, and an array vInt of size 6. The initial values are: aux=0xXX, resto=0x01, i=0x00, flag=0x00, and vInt={0x00, 0x00, 0x00, 0x00, 0x03, 0xA0}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the 'i' variable in the memory layout to the 'i' variable in the code. A red bracket above the 'for' loop indicates the calculation $6-1=5$, showing the range of indices for the first pass of the sort.

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 memory addresses and values for variables `aux`, `resto`, `i`, `flag`, and `vInt` array. The `vInt` array is shown with values: `vInt[0]=0x02`, `vInt[1]=0x1E`, `vInt[2]=0x09`, `vInt[3]=0x3B`, `vInt[4]=0xA0`, `vInt[5]=0x03`. Red dashed lines connect the code to the memory state: `vInt[0]` points to `0x02`, `vInt[1]` points to `0x1E`, and the `if` condition `vInt[i] > vInt[i+1]` is shown with `i=0` and `vInt[0]=0x02` and `vInt[1]=0x1E`, where `0x02 > 0x1E` is false. A red bracket above the `for` loop indicates `6-1=5`.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition `(CANT-resto)` in the code, indicating that the value of `i` is used to calculate the upper bound of the inner loop. A red bracket above the loop condition shows the calculation `6-1=5`, representing the current range of elements 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:

- Red dashed line from `vInt[1]` to line 11.
- Red dashed line from `vInt[2]` to line 14.
- Red bracket above line 12: $6-1=5$.
- Red arrow from line 14 to line 15, labeled **1**.
- Red arrow from line 15 to line 16, labeled **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	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 }

```

Handwritten annotations in the code:

- A red bracket above line 12 indicates $6-1=5$.
- A blue '1' is written above line 15.

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	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[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	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 selection sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The variables aux, resto, i, and flag are located at addresses 0x7FFE6E03, 0x7FFE6E07, 0x7FFE6E0B, and 0x7FFE6E0F respectively. The array vInt is located at addresses 0x7FFE6E13 to 0x7FFE6E27. The diagram shows the state of memory after the first pass of the selection sort, where the element 286 (vInt[1]) has been swapped with the element 3 (vInt[5]). The flag is set to 1, indicating that a swap occurred. The calculation 6-1=5 is shown, indicating the number of elements to compare in the next pass.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-1=5`, showing that the loop iterates from `i=0` to `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:

- Red dashed line from line 12 to line 14: $6-1=5$
- Red bracket above line 12: $6-1=5$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 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	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 }

```

Handwritten annotations in the code:

- A red bracket above the `for` loop condition `(CANT-resto)` with the text `6-1=5`.
- A blue number `2` is written next to the `if` statement.

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 selection sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The array vInt is located at addresses 0x7FFE6E13 to 0x7FFE6E27. The flag variable is at 0x7FFE6E0F. The aux variable is at 0x7FFE6E03. The resto variable is at 0x7FFE6E07. The i variable is at 0x7FFE6E0B. The diagram shows the state of the array vInt after the first pass of the selection sort, where the element 286 has been swapped with the element 3. The calculation 6-1=5 is shown, 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	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 }

```

A red dashed arrow points from the expression $6-1=5$ in the code to the memory address `0x7FFE6E0B` in the memory table.

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:

- Red bracket above line 12: $6-1=5$
- Red arrow from line 15 to line 16: 3
- Red arrow from line 17 to line 18: 4
- Red dashed arrow from line 15 to line 17: 3
- Red dashed arrow from line 17 to line 18: 4

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-1=5`, showing that the loop iterates from `i=0` to `i=5`.

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 execution of the selection sort algorithm on the array `vInt`:

- Initial array: `vInt = {2, 286, 9, 59, 928, 3}`
- Iteration 1 (i=0): Comparing `vInt[0]` (2) with `vInt[1]` (286). Since `2 < 286`, no swap occurs.
- Iteration 2 (i=1): Comparing `vInt[1]` (286) with `vInt[2]` (9). Since `286 > 9`, a swap occurs. The array becomes `{2, 9, 286, 59, 928, 3}`.
- Iteration 3 (i=2): Comparing `vInt[2]` (286) with `vInt[3]` (59). Since `286 > 59`, a swap occurs. The array becomes `{2, 9, 59, 286, 928, 3}`.
- Iteration 4 (i=3): Comparing `vInt[3]` (286) with `vInt[4]` (928). Since `286 < 928`, no swap occurs.
- Iteration 5 (i=4): Comparing `vInt[4]` (928) with `vInt[5]` (3). Since `928 > 3`, a swap occurs. The array becomes `{2, 9, 59, 3, 286, 928}`.

The final array after the first pass is `{2, 9, 59, 3, 286, 928}`. The diagram shows the comparison between `vInt[4]` (928) and `vInt[5]` (3), resulting in a swap.

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:

- Line 12: $6-1=5$ (red bracket over $(CANT-resto)$)
- Line 15: 4 (blue number over the opening curly brace of the inner loop)

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	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[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 }

```

6-1=5

vInt[5] = aux

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 selection sort algorithm. The memory layout shows the initial state of variables: aux (0x000000A0), resto (0x00000001), i (0x00000004), flag (0x00000001), and an array vInt containing {2, 286, 9, 59, 928, 3}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the flag variable in the memory layout to the flag variable in the code. A red bracket above the for loop indicates the calculation 6-1=5, representing the number of elements to compare in the first pass.

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 }

```

A red dashed arrow points from the expression $6-1=5$ in the code to the memory address $0x7FFE6E0B$ in the memory table, indicating the current value of i .

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	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 selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory location in the table. A red bracket highlights the calculation `6-1=5` in the `for` loop, indicating the number of comparisons for the first 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 selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell in the table. A red bracket above the `for` loop indicates the calculation $6-1=5$, representing the number of comparisons for the first 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 selection sort algorithm. The memory layout shows the initial state of variables: aux (0x00), resto (0x00), i (0x00), flag (0x00), and an array vInt of size 6. The array vInt contains the values {2, 286, 9, 59, 928, 3}. The execution of the algorithm is shown in the code block, which includes a loop that sorts the array in ascending order. A red dashed arrow points from the 'i' variable in the code to the memory cell at address 0x7FFE6E0B, which contains the value 0x00. A red bracket highlights the calculation $6-2=4$ in the code, indicating the number of elements to compare in the current iteration.

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 illustrating the memory layout and the execution of the sorting algorithm. 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 for sorting the array. Red dashed lines connect the code to the memory values. Annotations include:

- Red dashed line from line 10 to 0x02 at vInt[0].
- Red dashed line from line 12 to 0x09 at vInt[1].
- Red dashed line from line 14 to 0x3B at vInt[2].
- Red dashed line from line 15 to 0x1E at vInt[3].
- Red dashed line from line 17 to 0x03 at vInt[4].
- Red dashed line from line 18 to 0xA0 at vInt[5].
- Red dashed line from line 19 to 0xA0 at vInt[5].
- Red dashed line from line 20 to 0x00 at vInt[5].
- Red dashed line from line 21 to 0x00 at vInt[5].
- Red dashed line from line 22 to 0x00 at vInt[5].
- Red dashed line from line 23 to 0x00 at vInt[5].
- Red dashed line from line 24 to 0x00 at vInt[5].
- Red dashed line from line 25 to 0x00 at vInt[5].
- Red dashed line from line 26 to 0x00 at vInt[5].
- Red dashed line from line 27 to 0x00 at vInt[5].
- Red dashed line from line 28 to 0x00 at vInt[5].
- Red dashed line from line 29 to 0x00 at vInt[5].
- Red dashed line from line 30 to 0x00 at vInt[5].
- Red dashed line from line 31 to 0x00 at vInt[5].
- Red dashed line from line 32 to 0x00 at vInt[5].
- Red dashed line from line 33 to 0x00 at vInt[5].
- Red dashed line from line 34 to 0x00 at vInt[5].
- Red dashed line from line 35 to 0x00 at vInt[5].
- Red dashed line from line 36 to 0x00 at vInt[5].
- Red dashed line from line 37 to 0x00 at vInt[5].
- Red dashed line from line 38 to 0x00 at vInt[5].
- Red dashed line from line 39 to 0x00 at vInt[5].
- Red dashed line from line 40 to 0x00 at vInt[5].
- Red dashed line from line 41 to 0x00 at vInt[5].
- Red dashed line from line 42 to 0x00 at vInt[5].
- Red dashed line from line 43 to 0x00 at vInt[5].
- Red dashed line from line 44 to 0x00 at vInt[5].
- Red dashed line from line 45 to 0x00 at vInt[5].
- Red dashed line from line 46 to 0x00 at vInt[5].
- Red dashed line from line 47 to 0x00 at vInt[5].
- Red dashed line from line 48 to 0x00 at vInt[5].
- Red dashed line from line 49 to 0x00 at vInt[5].
- Red dashed line from line 50 to 0x00 at vInt[5].
- Red dashed line from line 51 to 0x00 at vInt[5].
- Red dashed line from line 52 to 0x00 at vInt[5].
- Red dashed line from line 53 to 0x00 at vInt[5].
- Red dashed line from line 54 to 0x00 at vInt[5].
- Red dashed line from line 55 to 0x00 at vInt[5].
- Red dashed line from line 56 to 0x00 at vInt[5].
- Red dashed line from line 57 to 0x00 at vInt[5].
- Red dashed line from line 58 to 0x00 at vInt[5].
- Red dashed line from line 59 to 0x00 at vInt[5].
- Red dashed line from line 60 to 0x00 at vInt[5].
- Red dashed line from line 61 to 0x00 at vInt[5].
- Red dashed line from line 62 to 0x00 at vInt[5].
- Red dashed line from line 63 to 0x00 at vInt[5].
- Red dashed line from line 64 to 0x00 at vInt[5].
- Red dashed line from line 65 to 0x00 at vInt[5].
- Red dashed line from line 66 to 0x00 at vInt[5].
- Red dashed line from line 67 to 0x00 at vInt[5].
- Red dashed line from line 68 to 0x00 at vInt[5].
- Red dashed line from line 69 to 0x00 at vInt[5].
- Red dashed line from line 70 to 0x00 at vInt[5].
- Red dashed line from line 71 to 0x00 at vInt[5].
- Red dashed line from line 72 to 0x00 at vInt[5].
- Red dashed line from line 73 to 0x00 at vInt[5].
- Red dashed line from line 74 to 0x00 at vInt[5].
- Red dashed line from line 75 to 0x00 at vInt[5].
- Red dashed line from line 76 to 0x00 at vInt[5].
- Red dashed line from line 77 to 0x00 at vInt[5].
- Red dashed line from line 78 to 0x00 at vInt[5].
- Red dashed line from line 79 to 0x00 at vInt[5].
- Red dashed line from line 80 to 0x00 at vInt[5].
- Red dashed line from line 81 to 0x00 at vInt[5].
- Red dashed line from line 82 to 0x00 at vInt[5].
- Red dashed line from line 83 to 0x00 at vInt[5].
- Red dashed line from line 84 to 0x00 at vInt[5].
- Red dashed line from line 85 to 0x00 at vInt[5].
- Red dashed line from line 86 to 0x00 at vInt[5].
- Red dashed line from line 87 to 0x00 at vInt[5].
- Red dashed line from line 88 to 0x00 at vInt[5].
- Red dashed line from line 89 to 0x00 at vInt[5].
- Red dashed line from line 90 to 0x00 at vInt[5].
- Red dashed line from line 91 to 0x00 at vInt[5].
- Red dashed line from line 92 to 0x00 at vInt[5].
- Red dashed line from line 93 to 0x00 at vInt[5].
- Red dashed line from line 94 to 0x00 at vInt[5].
- Red dashed line from line 95 to 0x00 at vInt[5].
- Red dashed line from line 96 to 0x00 at vInt[5].
- Red dashed line from line 97 to 0x00 at vInt[5].
- Red dashed line from line 98 to 0x00 at vInt[5].
- Red dashed line from line 99 to 0x00 at vInt[5].
- Red dashed line from line 100 to 0x00 at vInt[5].

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 illustrating the memory layout and the execution of the selection sort algorithm. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the memory address `0x7FFE6E0B`

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:

- Red dashed arrows connect memory addresses to array indices:
 - 0x7FFE6E17 to vInt[1]
 - 0x7FFE6E1B to vInt[2]
- Red bracket above line 12: $6-2=4$
- Red bracket above line 14: 1 (under i) and 2 (under $i+1$)

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 selection sort algorithm. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the memory address `0x7FFE6E0B`, which contains the value `0x02`. A red bracket above the `for` loop 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	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:

- Red dashed line from line 12 to line 14: $6-2=4$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 3

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 selection sort algorithm. A red dashed arrow points from the variable `i` (line 6) to the memory address `0x7FFE6E0B` in the memory table. A red bracket above the `for` loop (lines 12-13) indicates the calculation `6-2=4`, representing the number of elements remaining to be sorted.

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	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 memory addresses and values for variables and arrays during the execution of the provided C code. The memory layout shows the state of variables and the array `vInt` at various points. Red dashed lines connect the code to the memory state, highlighting the swap operation in the bubble sort algorithm. Annotations include $6-2=4$ and 3 indicating the current iteration and index.

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:

- A red bracket above the `for` loop condition indicates the calculation $6-2=4$.
- A blue number `3` is placed above the `if` statement, likely referring to the value of `flag` or a specific iteration.

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	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 }

```

6-2=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. A red dashed arrow points from the `flag` variable in the memory table to the `flag` variable in the code. A red bracket above the `for` loop indicates the calculation $6-2=4$, representing the number of elements to compare in the current pass.

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 selection sort algorithm. A red dashed arrow points from the `for` loop condition `(CANT-resto)` to the memory address `0x7FFE6E0B`, which contains the value `0x04`. 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	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 memory layout and code execution:

- Memory addresses 0x7FFE6E03 to 0x7FFE6E2F are shown on the left, corresponding to variables aux, resto, i, flag, and array vInt.
- The code on the right shows a selection sort algorithm. A red dashed arrow points from the `resto` variable to the `for` loop condition `(CANT-resto)`.
- A red bracket highlights the calculation `6-2=4`, indicating the current range of the array being sorted.

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	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 selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell in the table. A red bracket above the `for` loop indicates the calculation $6-2=4$, representing the number of elements to compare in the current pass.

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 selection sort algorithm. The memory layout shows the state of variables and the array vInt. A red dashed arrow points from the 'i' variable (0x7FFE6E0B) to the 'for' loop condition in the code, indicating the current iteration index. A red bracket highlights the calculation $6-3=3$ in the loop condition, representing the number of elements to compare in the current pass.

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	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 memory access and comparison in the sorting algorithm:

- Red dashed arrows show the sequence of memory accesses: `vInt[0]`, `vInt[1]`, `vInt[2]`, `vInt[3]`, `vInt[4]`, and `vInt[5]`.
- A red bracket indicates the calculation $6-3=3$, representing the remaining elements to be compared.
- A red arrow points from the `if` condition to the memory locations `vInt[i]` and `vInt[i+1]`.
- A red arrow points from the `if` condition to the `flag` variable.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-3=3`, representing the number of elements to compare in the current pass.

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:

- Red dashed line from `vInt[1]` to `i=0` in the `for` loop.
- Red dashed line from `vInt[2]` to `i=1` in the `for` loop.
- Red bracket above `(CANT-resto)` with text `6-3=3`.
- Red bracket below `vInt[i]` with text `1`.
- Red bracket below `vInt[i+1]` with text `2`.

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 selection sort algorithm. A red dashed line connects the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-3=3`, representing the number of elements to compare in the current pass.

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:

- Red dashed line from line 12 to line 14: $6-3=3$
- Red dashed line from line 14 to line 16: 2
- Red dashed line from line 16 to line 18: 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	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 }

```

Handwritten annotations in the code:

- A red bracket above the `for` loop condition `(CANT-resto)` with the calculation $6-3=3$.
- A blue number `2` is written next to the opening curly brace of the inner `if` loop.

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	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 }

```

6-3=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	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 }

```

6-3=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	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 selection sort algorithm. The memory addresses range from 0x7FFE6E03 to 0xFFFFFFFF. The variables aux, resto, i, and flag are stored in the first four memory locations. The array vInt is stored in the next six locations. The diagram shows the state of memory after the first iteration of the selection sort, where the element 286 has been swapped with the element 3. The value 6-3=3 is highlighted in red, indicating the current position of the element being compared.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-3=3`, showing that the loop iterates over the last three elements of the array.

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 selection sort algorithm. A red dashed arrow points from the `resto` variable to the `for` loop condition. A red bracket highlights the calculation `6-3=3` in the loop condition, indicating the current range of elements being 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 selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the memory cell at address `0x7FFE6E0F`. A red bracket above the `for` loop indicates the calculation `6-3=3`, representing the number of elements to compare in the current pass.

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	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 selection sort algorithm. The memory layout shows the state of variables and the array vInt. A red dashed arrow points from the 'i' variable (0x00) to the 'for' loop condition in the code, indicating the current iteration index. A red bracket above the 'for' loop condition indicates the calculation $6 - 4 = 2$, representing the number of elements to compare in the current pass.

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	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 memory addresses and values for variables `aux`, `resto`, `i`, `flag`, and array `vInt`. The array `vInt` is shown with values 2, 286, 9, 59, 928, 3. The diagram highlights the calculation $6-4=2$ for the loop condition `(CANT-resto)` and the comparison `vInt[i] > vInt[i+1]` between elements 0 and 1.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-4=2`, showing that the loop iterates over the last two elements of the array.

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:

- Red dashed arrows connect memory addresses to array indices:
 - 0x7FFE6E17 to vInt[1]
 - 0x7FFE6E1B to vInt[2]
- Red bracket above line 12: $6-4=2$
- Red bracket above line 14: 1 (under $vInt[i]$) and 2 (under $vInt[i+1]$)

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	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 `for` loop indicates the calculation $6-4=2$.
- A blue number `1` is placed above the opening curly brace of the `if` statement.

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	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 }

```

6-4=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 memory access and comparison in the sorting algorithm. A red dashed arrow points from the `flag` variable to the `vInt[0]` memory location. A red bracket above the `for` loop indicates the calculation $6-4=2$, representing the number of elements to compare.

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the `for` loop indicates the calculation `6-4=2`, representing the number of elements remaining to be sorted.

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 selection sort algorithm. A red dashed arrow points from the `flag` variable in the code to the `flag` memory cell in the table. A red bracket above the `for` loop indicates the calculation $6-4=2$, representing the number of elements to compare 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	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	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 selection sort algorithm. The memory layout shows the initial state of variables: aux (0x09), resto (0x05), i (0x02), and flag (0x00). The array vInt contains the values {2, 286, 9, 59, 928, 3}. The algorithm starts with i=0 and resto=0. The first iteration of the inner loop (i=0) compares vInt[0] (2) with vInt[1] (286). Since 2 < 286, no swap occurs. The next iteration (i=1) compares vInt[1] (286) with vInt[2] (9). Since 286 > 9, a swap occurs, resulting in vInt[1] = 9 and vInt[2] = 286. This process continues until the array is sorted. The diagram highlights the calculation 6-4=2, indicating the number of elements remaining to be sorted after the first iteration.

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 illustrating the memory layout and the execution of the selection sort algorithm. The memory layout shows the initial state of variables: aux (0x09), resto (0x05), i (0x00), flag (0x00), and an array vInt containing {2, 286, 9, 59, 928, 3}. The code snippet shows the selection sort algorithm. A red dashed arrow points from the 'i' variable in the memory layout to the 'i' variable in the code. A red bracket highlights the calculation '6-5=1' in the code, indicating the number of elements to compare in the first pass of the selection sort.

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 memory addresses and values for variables aux, resto, i, flag, and vInt array. The vInt array is shown with values 0x00, 0x00, 0x00, 0x02, 0x03, 0x09, 0x3B, 0x1E, 0xA0, and 0x00. The code snippet shows a selection sort algorithm. Annotations include:
 - A red bracket above line 12 indicating $6-5=1$.
 - A red bracket below line 14 indicating 0 .
 - A red bracket below line 15 indicating 1 .
 - Red dashed arrows pointing from the code to the memory table:
 - From line 10 to vInt[0] (0x02).
 - From line 11 to vInt[1] (0x03).
 - From line 14 to vInt[2] (0x09).
 - From line 15 to vInt[3] (0x3B).
 - From line 16 to vInt[4] (0x1E).
 - From line 17 to vInt[5] (0xA0).

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 selection sort algorithm. A red dashed arrow points from the memory address 0x7FFE6E0B (where `i` is located) to the `for` loop condition in the code, specifically to the expression `(CANT-resto)`. A red bracket above the loop condition indicates the calculation `6-5=1`, showing that the loop will execute once.

MEMORIA en ARQUITECTURA x86 32-bits

0x7FFE6E03

2

3

9

59

286

928

0x00	0x00	0x00	0x09
0x00	0x00	0x00	0x05
0x00	0x00	0x00	0x01
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x02
0x00	0x00	0x00	0x03
0x00	0x00	0x00	0x09
0x00	0x00	0x00	0x3B
0x00	0x00	0x01	0x1E
0x00	0x00	0x03	0xA0
⋮	⋮	⋮	⋮
0x00	0x00	0x00	0x00

aux

resto

i

flag

vInt[0]

vInt[1]

vInt[2]

vInt[3]

vInt[4]

vInt[5]

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 }
```

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	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	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	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	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	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	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	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	0x06	<- 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 }

```

```
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