# Metodología de la Programación II

### Relación de Problemas

## 1. Problemas de Punteros.

**Problema 1.1** *Declare una variable* cadena *como un vector de 100 caracteres. Suponiendo que esta cadena almacena un número indeterminado de caracteres (menos de 100) seguido de un carácter nulo ('\0'). Escriba un trozo de código que localice la posición del primer carácter espacio (' ') usando aritmética de punteros (sin usar ningún entero).*

**Problema 1.2** *Declare una variable* numeros *como un vector de 1000 enteros. Escriba un trozo de código que recorra el vector y modifique todos los enteros negativos cambiándolos de signo. No se debe usar el operador '[]', es decir, se deberá usar aritmética de punteros. El bucle se controlará mediante un contador entero.*

**Problema 1.3** *Modifique el código del problema 1.2 para controlar el final del bucle con un puntero a la posición siguiente a la última.*

**Problema 1.4** *Consideremos una variable* cad *que almacena una cadena de caracteres. Escriba un trozo de código que imprima en* cout *la cadena saltándose la primera palabra, y evitando escribirla carácter a carácter. Considere que puede haber una o más palabras, y si hay más de una palabra, están separadas por espacios en blanco.*

**Problema 1.5** *Las cadenas de caracteres representan un ejemplo clásico en el uso de punteros. El tipo correspondiente para almacenarlas es un vector de caracteres* [1]*. Implemente las siguientes funciones:*

- *Función* copiar_cadena*. Copia una cadena de caracteres en otra.*
- *Función* encadenar_cadena*. Añade una cadena de caracteres al final de otra.*
- *Función* longitud_cadena*. Devuelve un entero con la longitud (número de caracteres sin contar el nulo) de la cadena.*
- *Función* comparar_cadena*. Compara dos cadenas. Devuelve un valor negativo si la primera es más "pequeña", positivo si es más "grande" y cero si son "iguales".*

*Teniendo en cuenta que se supone que hay suficiente memoria en las cadenas de destino y no es necesario pasar el tamaño de las cadenas (controlado por la terminación en carácter nulo).*

**Problema 1.6** *Se desea una función que reciba un vector de números enteros junto con su longitud y que devuelva un puntero al elemento mayor. Escriba dos versiones:*

1. *Devuelve el resultado a través de* return*.*
2. *Devuelve el resultado a través de un parámetro (función* void*).*

*Considere la siguiente declaracón:*

```
1 int vector[100];
2 int *max;
```

*Haga uso de la primera función para mostrar en la salida estándar:*

1. *El elemento mayor del vector.*
2. *El elemento mayor de la primera mitad.*
3. *El elemento mayor de la segunda mitad.*

**Problema 1.7** *Represente gráficamente la disposición en memoria de las variables del siguientre programa, e indique lo que escribe la última sentencia de salida.*

[1] Recordemos que un literal de cadena, es una secuencia de caracteres entre comillas. Por ejemplo, "Hola". El tipo de este ejemplo es const char [5] (incluye el '\0').

```
#include <iostream>
using namespace std;

struct Celda {
  int d;
  Celda *p1, *p2, *p3;
};

int main (int argc, char *argv[])
{
  Celda a, b, c, d;

  a.d = b.d = c.d = d.d = 0;

  a.p1 = &c;
  c.p3 = &d;
  a.p2 = a.p1->p3;
  d.p1 = &b;
  a.p3 = c.p3->p1;
  a.p3->p2 = a.p1;
  a.p1->p1 = &a;
  a.p1->p3->p1->p2->p2 = c.p3->p1;
  c.p1->p3->p1 = &b;
  (*((*(c.p3->p1)).p2->p3)).p3 = a.p1->p3;
  d.p2 = b.p2;
  (*(a.p3->p1)).p2->p2->p3 = (*(a.p3->p2)).p3->p1->p2;

  a.p1->p2->p2->p1->d = 5;
  d.p1->p3->p1->p2->p1->p1->d = 7;
  (*(d.p1->p3)).p3->d = 9;
  c.p1->p2->p3->d = a.p1->p2->d - 2;
  (*(c.p2->p1)).p2->d = 10;

  cout << "a="<<a.d<<"  b="<<b.d<<"  c="<<c.d<<"  d="<<d.d<<endl;
}
```

**Problema 1.8** *Consideremos un vector* v *de reales de tamaño* n. *Supongamos que se desea dividir el vector para que aparezcan en primer lugar todos los elementos menores o iguales al primero (*v[0]*), seguidos del resto (mayores). Para ello, ideamos un algoritmo que consiste en*

- *Colocamos un puntero al principio del vector y lo adelantamos mientras el elemento apuntado sea menor o igual.*
- *Colocamos un puntero al final del vector y lo atrasamos mientras el elemento apuntado sea mayor.*
- *Si los punteros no se han cruzado, es que se han encontrado dos elementos "mal colocados". Los intercambiamos y volvemos a empezar.*

*Este algoritmo acabará cuando los dos punteros se crucen, habiendo quedado todos los elementos ordenados según el criterio inicial.*

*Escriba un trozo de código que, en primer lugar, declare una variable constante (*n*) y un vector de reales con ese tamaño. En segundo lugar, escriba el código que corresponde al algoritmo antes descrito.*

**Problema 1.9** *Supongamos tres vectores* v1,v2,res *de valores reales. En* v1,v2 *se almacenan, respectivamente,* n,m *valores ordenados de menor a mayor. Escribir un trozo de código para mezclar, de manera ordenada, los valores en el vector* res *que tiene capacidad para almacenar al menos* n+m *valores. No se debe usar el operador '[]', es decir, se debe usar aritmética de punteros.*

**Problema 1.10** *Escriba una función que reciba como entrada un vector de números junto con su longitud y que nos devuelva un vector de punteros a los elementos del vector de entrada de forma que los elementos apuntados por dicho vector de punteros estén ordenados (véase figura 1). Note que el vector de punteros debe ser un parámetro de la función, y estar reservado previamente a la llamada con un tamaño, al menos, igual al del vector.*

*Una vez escrita la función, considere la siguiente declaración:*

```
int vec[1000];
int *ptr[1000];
```

*Y escriba un trozo de código que, haciendo uso de esa función, permita:*

1. *Ordenando punteros, mostrar los elementos del vector, ordenados.*
2. *Ordenando punteros, mostrar los elementos de la segunda mitad del vector, ordenados.*

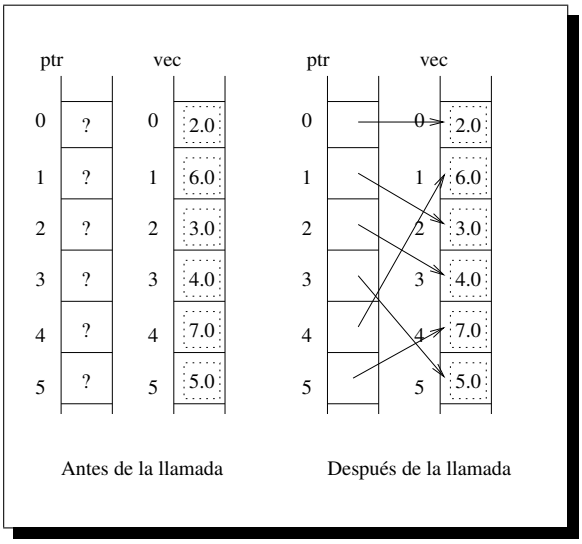


Figura 1: Resultado de ordenar el vector de punteros.

*sin modificar el vector de datos* vec.

**Problema 1.11** *Escriba una función a la que le damos una cadena de caracteres, una posición de inicio* P *y una longitud* L. *Como resultado, devuelve una subcadena que comienza en* P *y que tiene longitud* L. *Nota: Si la longitud es demasiado grande (se sale de la cadena original), se devolverá una cadena de menor tamaño. No se debe usar el operador '[]', es decir, se debe usar aritmética de punteros.*

**Problema 1.12** *Represente gráficamente la disposición en memoria de las variables del siguientre programa, e indique lo que escribe la última sentencia de salida. Tenga en cuenta que el operador* $->$ *tiene más prioridad que el operador* $*$.

```
1  #include <iostream>
2  using namespace std;
3
4  struct SB; // declaración adelantada
5  struct SC; // declaración adelantada
6  struct SD; // declaración adelantada
7
8  struct SA { int dat;  SB *p1;  };
9  struct SB { int dat;  SA *p1; SC *p2; };
10 struct SC { SA *p1;   SB *p2; SD *p3; };
11 struct SD { int *p1; SB *p2; };
12
13 int main (int argc, char *argv[])
14
15 {
16   SA a;
17   SB b;
18   SC c;
19   SD d;
20   int dat;
21
22   a.dat = b.dat = dat = 0;
23
24   a.p1 = &b;
25   b.p1 = &a;
26   b.p2 = &c;
27   c.p1 = b.p1;
28   c.p2 = &(*(a.p1));
29   c.p3 = &d;
30   d.p1 = &dat;
31   d.p2 = &(*(c.p1)->p1);
32   *(d.p1) = 9;
33   (*(b.p2)->p1).dat = 1;
34   (*((b.p2)->p3->p2)->p1).dat = 7;
35   *((*((*(c.p3->p2)).p2->p3)).p1) = (*(b.p2)).p1->dat + 5;
36   cout << "a.dat=" << a.dat << "  b.dat=" << b.dat << "  dat=" << dat << endl;
37 }
```

**Problema 1.13** *Considere la figura 2. Se presentan gráficamente un conjunto de estructuras de datos. Se puede observar que las matrices se representan indicando los índices y las estructuras indicando los nombres de los campos.*

*Escriba los trozos de código que corresponden a su creación. Nota: No se debe usar memoria dinámica (para cada caso se incluye el nombre de las variables necesarias).*

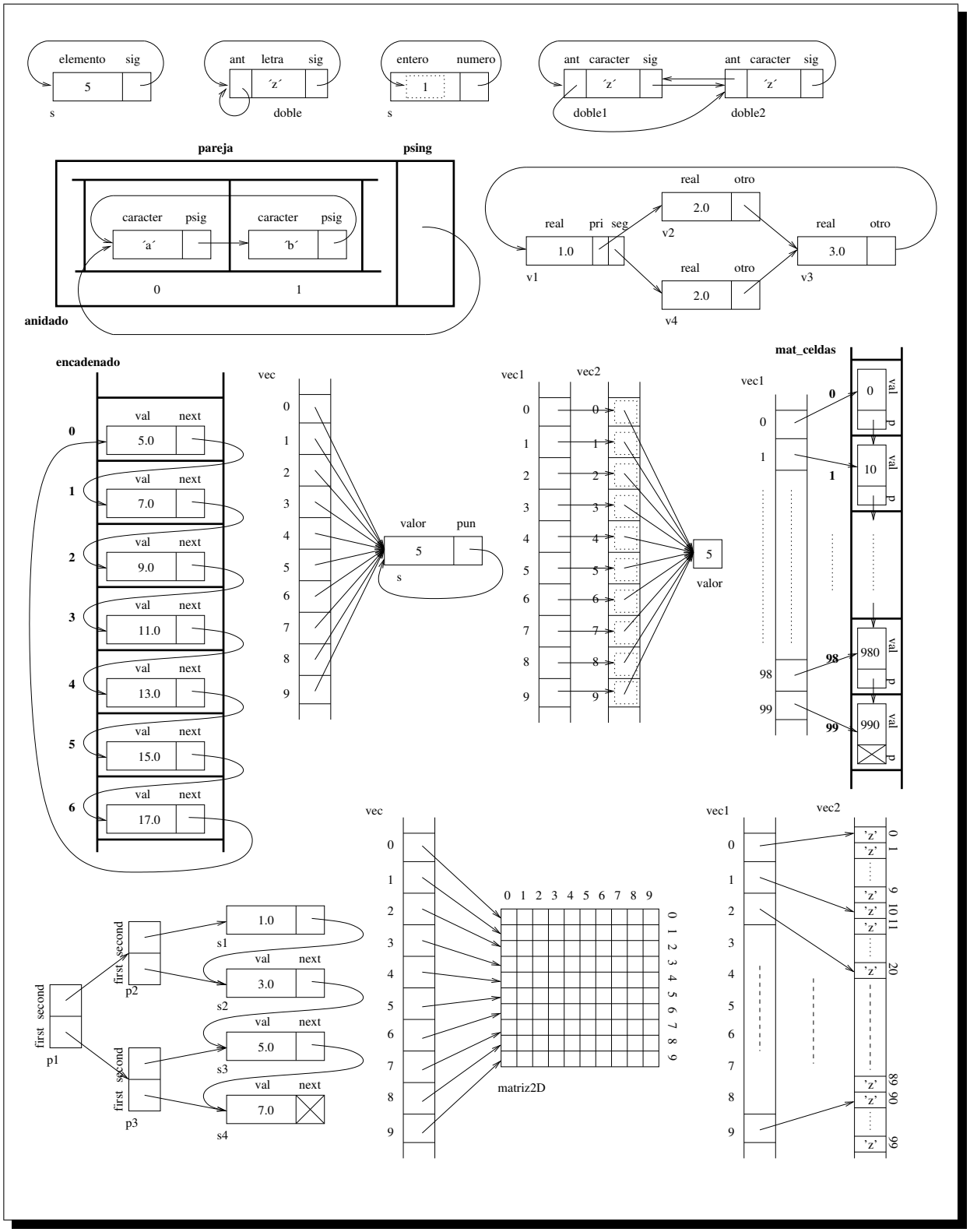


Figura 2: Ejemplos de estructuras.