



DECSAI

Departamento de Ciencias de la Computación e I.A.

Universidad de Granada

Metodología de la Programación

Grado en Ingeniería Informática

Problemas

Curso 2012/2013

Francisco J. Cortijo Bon

Departamento de Ciencias de la Computación
e Inteligencia Artificial
ETS de Ingenierías Informática y de Telecomunicación
Universidad de Granada
cb@decsai.ugr.es

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas I: Punteros

1. Describir la salida de los siguientes programas:

a)

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

int main (){
    int a = 5, *p;

    a = *p * a;
    if (a == *p)
        cout << "a es igual a *p" << endl;;
    else
        cout << "a es diferente a *p" << endl;
    return 0;
}
```

b)

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

int main (){
    int a = 5, *p;

    *p = *p * a;
    if (a == *p)
        cout << "a es igual a *p" << endl;;
    else
        cout << "a es diferente a *p" << endl;
    return 0;
}
```

c)

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

int main (){
    int a = 5, *p = &a;

    *p = *p * a;
    if (a == *p)
        cout << "a es igual a *p" << endl;;
    else
        cout << "a es diferente a *p" << endl;
    return 0;
}
```

d)

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

int main (){
    int a = 5, *p = &a, **p2 = &p;

    **p2 = *p + (**p2 / a);
    *p = a+1;
    a = **p2 / 2;
    cout << "a es igual a: " << a << endl;
    return 0;
}
```

2. Represente gráficamente la disposición en memoria de las variables del programa mostrado en la figura 1, e indique lo que escribe la última sentencia de salida.
3. Declare una variable `v` 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 permite usar el operador `[]`, es decir, el recorrido se efectuará usando aritmética de punteros y el bucle se controlará mediante un contador entero.

Nota: Para inicializar aleatoriamente el vector con valores enteros entre -50 y 50, por ejemplo, puede emplearse el siguiente fragmento de código:

```

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

```

Figura 1: Código asociado al problema 2

```

#include <cstdlib>
#include <ctime>

...
const int MY_MAX_RAND = 50; // Queremos valores -50<=n<=50
time_t t;

...
srand ((int) time(&t)); // Inicializa el generador con el reloj del sistema
...
for int (i=0; i<1000; i++)
    v[i] = (rand() % ((2*MY_MAX_RAND)+1)) - MY_MAX_RAND;

```

Acerca de `srand()`, `rand()` y `time()`: <http://www.cplusplus.com>

4. Modifique el código del problema 3 para controlar el final del bucle con un puntero a la posición siguiente a la última.

5. Con estas declaraciones:

```
const int MAX = 100;
float v1 [MAX] = {2, 3, 8, 22, 44, 88, 99, 100, 101, 255, 665};
float v2 [MAX] = {1, 3, 4, 5, 6, 25, 87, 89, 99, 100, 500, 1000};
float res [2*MAX];

int tam_v1=11, tam_v2=12;      // 0 <= tam_v1, tam_v2 < MAX
int tam_res = tam_v1+tam_v2;  // 0 <= tam_res < 2*MAX
```

Escribir un trozo de código para mezclar, de manera *ordenada*, los valores de `v1` y `v2` en el vector `res`.

Nota: Observad que `v1` y `v2` almacenan valores *ordenados* de menor a mayor.

No se puede usar el operador `[]`, es decir, se debe resolver usando aritmética de punteros.

6. Consideremos un vector `v` de números reales de tamaño `n`. Supongamos que se desea dividir el vector en dos secciones: la primera contendrá a todos los elementos menores o iguales al primero y la otra, los mayor.

Para ello, proponemos un algoritmo que consiste en:

- Colocamos un puntero al principio del vector y lo adelantamos mientras el elemento apuntado sea menor o igual que el primero.
- Colocamos un puntero al final del vector y lo atrasamos mientras el elemento apuntado sea mayor que el primero.
- 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 declare una constante (`n`) con valor 20 y un vector de reales con ese tamaño, lo rellene con números aleatorios entre 0 y 100 y lo reorganice usando el algoritmo antes descrito.

7. Las cadenas de caracteres (tipo “C”, o cadenas “clásicas”) son una buena fuente para ejercitarse en el uso de punteros. Una cadena de este tipo almacena un número indeterminado de caracteres (para los ejercicios basará un valor siempre menor que 100) delimitados al final por el *carácter nulo* (`'\0'`).

Escriba un trozo de código que lea una cadena y localice la posición del primer *carácter espacio* (`' '`) en una cadena de caracteres “clásica”. El programa debe indicar su posición (0: primer carácter, 1: segundo carácter, etc.).

Notas:

- La cadena debe recorrerse usando aritmética de punteros y sin usar ningún entero.
- Usar la función `getline()` para la lectura de la cadena (Cuidado: usar el método público de `istream` sobre `cin`, o sea `cin.getline()`). Ver <http://www.cplusplus.com/reference/istream/istream/getline/>

8. Consideremos una cadena de caracteres “clásica”. Escriba un trozo de código que lea una cadena y la imprima pero saltándose la primera palabra, *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.

9. Considere una cadena de caracteres “clásica”. Escriba la función `longitud_cadena`, que devuelva un *entero* cuyo valor indica la longitud de la cadena: el número de caracteres desde el inicio hasta el carácter nulo (no incluido).

Nota: No se puede usar el operador `[]`, es decir, se debe resolver mediante aritmética de punteros.

10. Escriba una función a la que le damos una cadena de caracteres y calcule si ésta es un palíndromo.

Nota: No se puede usar el operador `[]`, es decir, se debe resolver mediante aritmética de punteros.

11. Considere dos cadenas de caracteres “clásicas”. Escriba la función `comparar_cadenas`, que devuelve un valor *entero* que se interpretará como sigue: si es *negativo*, la primera cadena es más “pequeña”; si es *positivo*, será más “grande”; y si es *cero*, las dos cadenas son “iguales”.

Nota: Emplead como criterio para determinar el orden el código ASCII de los caracteres que se están comparando.

12. Considere dos cadenas de caracteres “clásicas”. Escriba la función `copiar_cadena`, que copiará una cadena de caracteres en otra. El resultado de la copia será el primer argumento de la función. La cadena original (segundo argumento) **no** se modifica.

Nota: Se supone que hay suficiente memoria en la cadena de destino.

13. Considere dos cadenas de caracteres “clásicas”. Escriba la función `encadenar_cadena`, que añadirá una cadena de caracteres al final de otra. El resultado se dejará en el primer argumento de la función. La cadena que se añade (segundo argumento) **no** se modifica.

Nota: Se supone que hay suficiente memoria en la cadena de destino.

14. Escriba una función a la que le damos una cadena de caracteres, una posición de inicio `p` y una longitud `l` sobre esta cadena. Queremos obtener una *subcadena* de ésta, que comienza en `p` y que tiene longitud `l`.

Notas:

- Si la longitud es demasiado grande (se sale de la cadena original), se devolverá una cadena de menor tamaño (la que empieza en `p` y llega hasta el final de la cadena).
- No se puede usar el operador `[]`, es decir, se debe resolver mediante aritmética de punteros.

15. Escriba una función a la que le damos una cadena de caracteres. Queremos obtener una nueva cadena, resultado de invertir la primera.

Notas:

- La cadena original **no** se modifica.
- No se puede usar el operador `[]`, es decir, se debe resolver mediante aritmética de punteros.

16. 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:

- a) Devuelve el resultado como resultado de la función (`return`).
- b) Devuelve el resultado a través de un parámetro (función `void`).

Considere la siguiente declaración:

```
const int MAX = 100;
int vector [MAX];
int usados;          // 0 <= usados < MAX
```

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

- El elemento mayor del vector.
- El elemento mayor de la primera mitad.
- El elemento mayor de la segunda mitad.

17. 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 2).

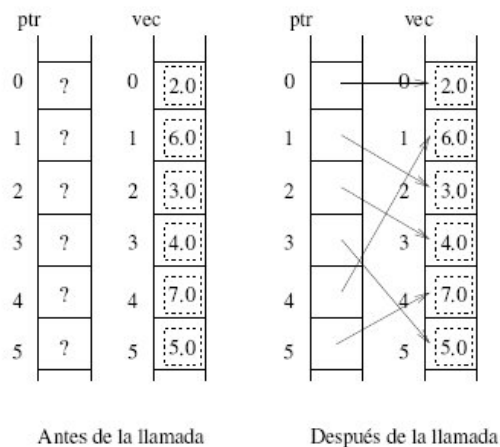


Figura 2: Resultado de ordenar el vector de punteros

Observe 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:

```
const int MAX = 20;
int vec [MAX];
int *ptr [MAX];
```

y escriba un trozo de código que, haciendo uso de la función, permita:

- Ordenando punteros, mostrar los elementos del vector, ordenados.
- Ordenando punteros, mostrar los elementos de la segunda mitad del vector, ordenados.

sin modificar el vector de datos `vec`.

Nota: Iniciar aleatoriamente el vector `vec`.

18. Represente gráficamente la disposición en memoria de las variables del programa mostrado en la figura 3, e indique lo que escribe la última sentencia de salida. Tenga en cuenta que el operador \rightarrow 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     SA a;
16     SB b;
17     SC c;
18     SD d;
19     int dat;
20
21     a.dat = b.dat = dat = 0;
22
23     a.p1 = &b;
24     b.p1 = &a;
25     b.p2 = &c;
26     c.p1 = b.p1;
27     c.p2 = &(*a.p1);
28     c.p3 = &d;
29     d.p1 = &dat;
30     d.p2 = &(*c.p1)->p1;
31     *(d.p1) = 9;
32     (*(b.p2)->p1).dat = 1;
33     ((*((b.p2)->p3->p2)->p1).dat = 7;
34     *(((*c.p3->p2).p2->p3).p1) = (*(b.p2)).p1->dat + 5;
35     cout << "a.dat=" << a.dat << " b.dat=" << b.dat << " dat=" << dat << endl;
36 }

```

Figura 3: Código asociado al problema 18

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas II: Memoria dinámica

1. Escriba un programa para que lea una secuencia con un número indefinido de valores `double` hasta que se introduzca un valor negativo. Estos valores (excepto el último, el negativo) los almacenará en una estructura de celdas enlazadas (una *lista*) y después mostrará los valores almacenados.

Escribir un programa para solucionar este problema, con dos funciones:

- a) Una para leer y almacenar los valores.
 - b) Otra para mostrarlos.
2. Ampliar el problema 1 de manera que una vez leídos los datos realice unos cálculos sobre los datos almacenados en la lista. Se pide que se escriban tres funciones para calcular:
 - a) el número de celdas enlazadas.
 - b) la media de los datos almacenados.
 - c) la varianza de los datos almacenados.
 3. Utilizando como base el problema 1, escribir un programa que lea una secuencia de valores y los almacene en una *lista*. Escribir una función que determine si la secuencia está ordenada.
 4. Considere una secuencia de datos almacenada en una *lista*. Implemente una función para **ordenar** la secuencia empleando el método de *ordenación por selección*.
 5. Considere una secuencia **ordenada** de datos almacenada en una *lista*.
 - a) Implemente una función para insertar un nuevo dato en su posición correcta.
 - b) Implemente una función para, dado un dato, eliminar la celda que lo contiene.
 6. Considere dos secuencias de datos **ordenadas** almacenadas en sendas *listas*. Implemente una función para *mezclar ordenadamente* las dos secuencias en una nueva, de forma que las dos listas originales se queden vacías tras realizar la mezcla y la lista resultante contenga todos los datos.

Observe se trata de una variante del algoritmo *mergesort*. Ahora se exige la modificación de las secuencias originales: en esta versión los datos se “mueven” hacia la lista resultante en lugar de copiarlos.

Nota: No es preciso (ni se permite) realizar ninguna operación de reserva ni liberación de memoria.
 7. Deseamos guardar un número indefinido de valores `double` para poder procesarlos posteriormente. Resuelva el problema almacenando los datos en un *vector dinámico* que vaya creciendo conforme necesite espacio para almacenar un nuevo valor.

Escribir tres funciones diferentes, de manera que el problema se pueda resolver con cualquiera de ellas. Las funciones se diferencian en la manera en que hacen crecer el vector dinámico, aumentando su capacidad cuando no haya espacio para almacenar un nuevo valor, ampliándolo:

 - a) en una casilla
 - b) en bloques de tamaño n
 - c) duplicando su tamaño

Nota: Reservar inicialmente 5 casillas.

Para la resolución de este ejercicio proponemos dos programas diferentes.

- 1) El primero tomará los valores a insertar en el vector dinámico directamente del teclado.
- 2) El segundo tomará los valores a insertar desde una lista.

Intentar que el almacenamiento en el vector dinámico (*procesamiento*) sea lo más independiente posible de la *entrada* de datos.

8. Supongamos que para definir matrices bidimensionales dinámicas usamos una estructura como la que aparece en la figura 4 (tipo `Matriz2D-1`). En los apuntes de clase se detalla cómo crear y liberar esta estructura.

Nota: Recuerde que los módulos que procesan estructuras de este tipo necesitan recibir como parámetros el número de filas y columnas de la matriz.

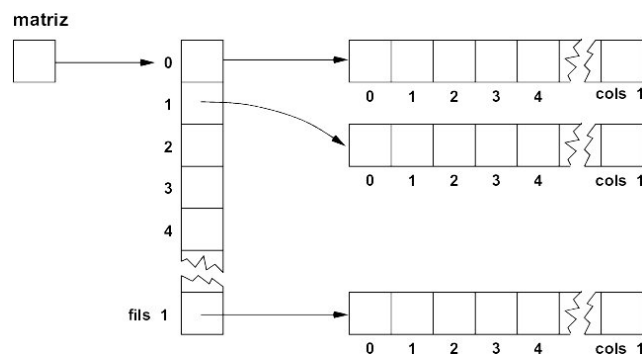


Figura 4: Tipo `Matriz2D1`: datos guardados en filas independientes

- a) Construir un módulo que lea del teclado `fils×cols` valores y los copie en la matriz.
- b) Construir un módulo que muestre los valores guardados en la matriz.
- c) Construir un módulo que reciba una matriz de ese tipo, cree y devuelva una copia.
- d) Construir un módulo que extraiga una submatriz de una matriz bidimensional `Matriz2D-1`. Como argumento de la función se introduce desde qué fila y columna y hasta qué fila y columna se debe realizar la copia de la matriz original. La submatriz devuelta es una *nueva* matriz.
- e) Construir un módulo que elimine una fila de una matriz bidimensional `Matriz2D-1`. Obviamente, no se permiten “huecos” (filas vacías). El módulo devuelve una *nueva* matriz.
- f) Construir un módulo como el anterior, pero que en vez de eliminar una fila, elimine una columna. El módulo devuelve una *nueva* matriz.
- g) Construir un módulo que devuelva la traspuesta de una matriz. La matriz devuelta es una *nueva* matriz.
- h) Construir un módulo que reciba una matriz y la modifique, de tal manera que “invierta” las filas: la primera será la última, la segunda la penúltima, y así sucesivamente. El módulo devuelve una *nueva* matriz.

9. Supongamos que ahora decidimos utilizar una forma diferente para representar las matrices bidimensionales dinámicas a la que se propone en el ejercicio 8. Usaremos una estructura semejante a la que aparece en la figura 5 (tipo `Matriz2D-2`). En los apuntes de clase se detalla cómo crear y liberar esta estructura.

Nota: Recuerde que los módulos que procesan estructuras de este tipo necesitan recibir como parámetros el número de filas y columnas de la matriz.

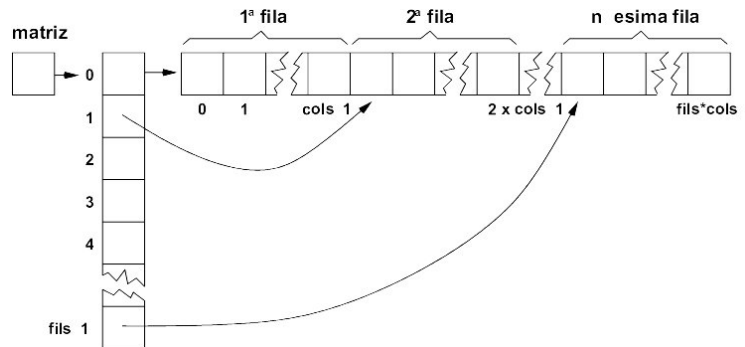


Figura 5: Tipo `Matriz2D2`: datos guardados en una sola fila

- Reescribir todos los módulos propuestos en el ejercicio 8 usando esta nueva representación.
 - Construir un módulo que dada una matriz bidimensional dinámica `Matriz2D-1` realice una copia de la misma en una matriz bidimensional dinámica `Matriz2D-2` y la devuelva.
 - Desarrollar un módulo que realice el paso inverso, convertir de `Matriz2D-2` a `Matriz2D-1` y devolverla.
10. Se desea desarrollar una estructura de datos que permita representar de forma general diversas figuras poligonales. Cada figura poligonal se puede representar como un conjunto de puntos en el plano unidos por segmentos de rectas entre cada dos puntos adyacentes. Por esta razón se propone la representación mostrada en la figura 6.

Así, un polígono se representa como una secuencia *circular* y *ordenada* de nodos enlazados. Por ejemplo, el triángulo de puntos (2,0), (4,0) y (3,1) se representa como se indica en la figura 6.

```
struct Punto2D {
    double x;
    double y;
};
struct Nodo {
    Punto2D punto;
    Nodo * sigpunto;
};
typedef Nodo * Poligono;
```

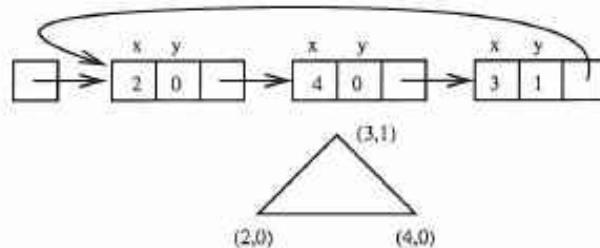


Figura 6: Representación de un polígono y un ejemplo (un triángulo)

Teniendo en cuenta esta representación, responder a las siguientes cuestiones:

- Implementar un módulo que permita iniciar y “rellenar” una variable de tipo `Poligono` proporcionándole un vector de datos de tipo `Punto2D` y el número de puntos que debe emplear para iniciar el polígono.
- Desarrollar un módulo que permita liberar la memoria reservada por una variable `Poligono`.
- Construir un módulo que determine el número de lados que contiene la figura almacenada en una variable de tipo `Poligono`.
- Suponiendo que existe una función llamada `PintaRecta (Punto2D p1, Punto2D p2)` que pinta una recta entre los dos puntos que se le pasan como argumentos, construir un módulo que permita pintar la figura que representa una determinada variable `Poligono`.
- Sabiendo que una variable `Poligono` almacena un cuadrado, implementar un módulo que devuelva los dos triángulos que resultan de unir mediante una recta la esquina inferior izquierda del cuadrado con su esquina superior derecha (figura 7.A).
- Construir un módulo que a partir de una variable `Poligono` que representa a un triángulo devuelva el triángulo formado por los puntos medios de las rectas del triángulo original (figura 7.B).
- Desarrollar un módulo que permita construir un polígono regular de n lados inscrito en una circunferencia de radio r y centro (x, y) (figura 7.C).

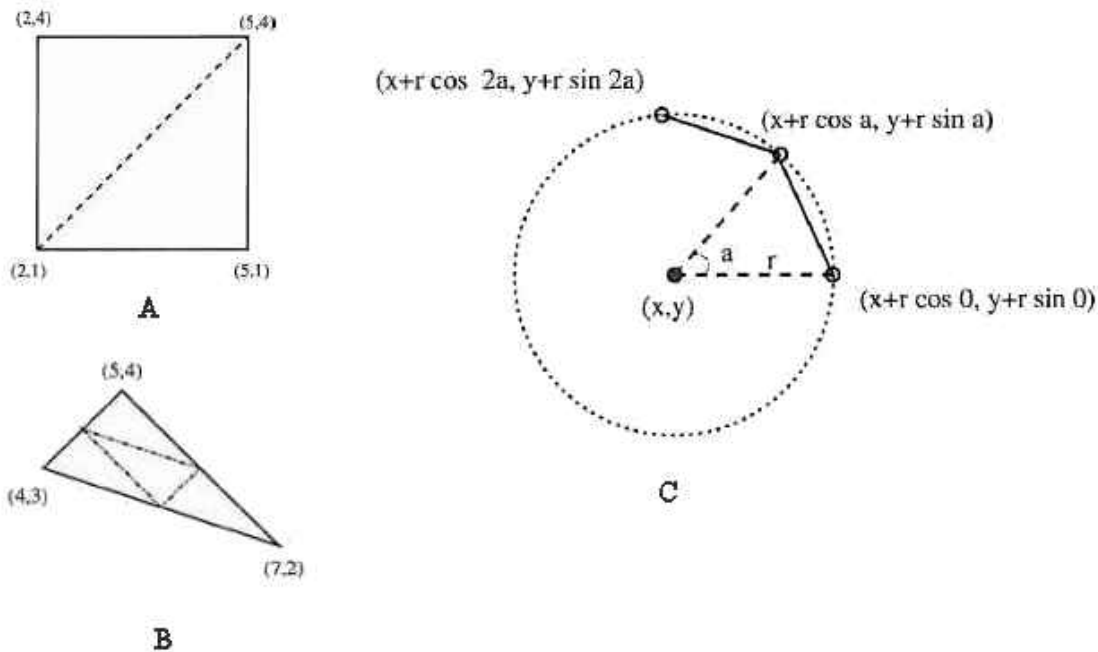


Figura 7: A y B) Cálculo de algunos triángulos. C) Construcción de un polígono inscrito en una circunferencia

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas III: Clases (I)

Los ejercicios propuestos tienen como finalidad que el alumno practique con los constructores y destructor de una clase, así como con métodos sencillos de acceso y manipulación de la clase.

Todos los ejercicios deben estar **completamente implementados** y **modularizados**. Significa que debe existir, para cada clase:

- Un fichero `.h` con las declaraciones.
- Un fichero `.cpp` con las definiciones.

Además, debe escribirse un fichero `.cpp` con la función `main()` que contenga ejemplos sobre el uso de la clase.

1. Implementar la clase `VectorDinámico` para trabajar con vectores de enteros de tamaño arbitrario, y no definido a priori.

Proponer una *representación* para la clase e implementar los siguientes *métodos*:

- a) Constructor sin argumentos, que crea un vector dinámico *vacío*.
- b) Constructor con un argumento, que crea un vector dinámico con un número de casillas indicado en el argumento. Inicia todas las casillas a cero.
- c) Constructor con dos argumentos, que crea un vector dinámico con un número de casillas indicado en el primer argumento. Inicia todas las casillas al valor indicado en el segundo argumento.
- d) Destructor.
- e) Método (valor devuelto: `bool`) que consulta si el vector dinámico está *vacío*.
- f) Método para consultar el número de casillas ocupadas.
- g) Métodos para leer/escribir un valor.
- h) Método para escribir el contenido del vector dinámico.
- i) Método para *redimensionar* un vector dinámico. Recibirá un solo argumento: el nuevo número de casillas, que suponemos siempre será mayor que el actual. El "nuevo" vector debe mantener los valores almacenados anteriormente, en las mismas posiciones.

2. Implementar la clase `Matriz2D-1` para elementos de tipo `TipoBase`. De acuerdo con `TipoBase` establecer el valor *nulo* (0 si `TipoBase` es `int`, 0.0 si `TipoBase` es `double` o `float`, "" si `TipoBase` es `string`,...).

Empleando la representación básica conocida, se trata de implementar los siguientes *métodos*:

- a) Constructor sin argumentos, que crea una matriz *vacía*.
- b) Constructor con un argumento, que crea una matriz *cuadrada* con el número de filas y columnas indicado en el argumento. Inicia todas las casillas al valor *nulo*.
- c) Constructor con dos argumentos, que crea una matriz con el número de filas indicado en el primer argumento y con el número de columnas indicado en el segundo. Inicia todas las casillas al valor *nulo*.

- d) Constructor con tres argumentos, que crea una matriz con el número de filas indicado en el primer argumento y con el número de columnas indicado en el segundo argumento. Además inicia todas las casillas de la matriz al valor especificado con el tercer argumento.
- e) Destructor.
- f) Método (valor devuelto: `bool`) que consulta si la matriz está *vacía*.
- g) Métodos para escribir/leer un valor. Responderán a los prototipos:


```
void PonValor (int fila, int col, TipoBase val);
TipoBase LeeValor (int fila, int col);
```

3. Implementar la clase `Matriz2D-2` para elementos de tipo `TipoBase`.

Se trata de implementar los mismos métodos que en el problema 2.

4. Implementar la clase `Lista` para trabajar con listas dinámicas (de tamaño arbitrario, y no definido a priori, cuyos nodos residen en el *heap*) de datos de tipo `TipoBase`. De acuerdo con `TipoBase` establecer el valor *nulo* (0 si `TipoBase` es `int`, 0.0 si `TipoBase` es `double` o `float`, "" si `TipoBase` es `string`,...).

Proponer una *representación* para la clase (basada en el almacenamiento de los nodos en memoria dinámica) e implementar los siguientes *métodos*:

- a) Constructor sin argumentos, que crea una lista *vacía*.
- b) Constructor con un argumento, que crea una lista con un número de nodos indicado en el argumento. Inicia todos los nodos de la lista al valor *nulo*.
- c) Constructor con dos argumentos, que crea una lista con un número de nodo indicado en el primer argumento. Inicia todos los nodos de la lista al valor indicado en el segundo argumento.
- d) Destructor.
- e) Método (valor devuelto: `bool`) que consulta si la lista está *vacía*.
- f) Método para consultar el número de nodos de la lista.
- g) Método para escribir el contenido de una lista.
- h) Método para insertar un valor en la lista. Modifica la lista. Responderá al siguiente prototipo:


```
void Insertar (TipoBase val, int pos);
```

 de manera que inserta un nuevo nodo en la lista con valor `val` en la posición `pos` (1 para el primer nodo, 2 para el segundo, etc.). La posición seguirá el siguiente convenio: `pos` indica el número de orden que ocupará el nuevo nodo que se va a insertar.
 Algunos ejemplos (si `TipoBase` es `int`):
 Antes: < 6,8,4,3,2,9 > Insertar (5, 2) Después: < 6,5,8,4,3,2,9 >
 Antes: < 6,8,4,3,2,9 > Insertar (5, 6) Después: < 6,8,4,3,2,5,9 >
 Antes: < 6,8,4,3,2,9 > Insertar (5, 7) Después: < 6,8,4,3,2,9,5 >
 Antes: < 6,8,4,3,2,9 > Insertar (5, 1) Después: < 5,6,8,4,3,2,9 >
- i) Método para borrar un nodo en la lista. Responderá al siguiente prototipo:


```
void Borrar (int pos);
```

 de manera que borra el nodo que ocupa la posición `pos` (1 para el primer nodo, 2 para el segundo, etc.)

j) Métodos para leer/escribir un valor.

```
TipoBase ObtenerValor (int pos);
```

```
void PonerValor (int pos, TipoBase val);
```

de tal manera que `pos` indica la posición del nodo (1 para el primer nodo, 2 para el segundo, etc.)

5. Implementar la clase `Pila`.

Una *pila* es una estructura de datos que permite la gestión de problemas en los que la gestión se realiza empleando un protocolo **LIFO** (last in first out).

Proponer una *representación* para la clase (basada en el almacenamiento de los nodos en memoria dinámica) e implementar los siguientes *métodos*:

- a) Constructor sin argumentos, que crea una pila *vacía*.
- b) Destructor.
- c) Método (valor devuelto: `bool`) que consulta si la pila está *vacía*.
- d) Método para añadir un valor. La pila se modifica.
- e) Método para sacar un valor. Obtiene (devuelve) el elemento extraído. La pila se modifica.
- f) Método para consultar qué elemento está en el **tope** de la pila. La pila no se modifica.

6. Implementar la clase `Cola`.

Una *cola* es una estructura de datos que permite la gestión de problemas en los que la gestión se realiza empleando un protocolo **FIFO** (First in first out).

Proponer una *representación* para la clase (basada en el almacenamiento de los nodos en memoria dinámica) e implementar los siguientes *métodos*:

- a) Constructor sin argumentos, que crea una cola *vacía*.
- b) Destructor.
- c) Método (valor devuelto: `bool`) que consulta si la cola está *vacía*.
- d) Método para añadir un valor. La cola se modifica.
- e) Método para sacar un valor. Obtiene (devuelve) el elemento extraído. La cola se modifica.
- f) Método para consultar qué elemento está en la **cabecera** de la cola. La cola no se modifica.

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas IV: Clases (II)

Los ejercicios propuestos tienen como finalidad que el alumno practique con:

- el constructor de copia y la sobrecarga del operador de asignación, empleando código reutilizable,
- la sobrecarga de los operadores `[]` y `()`
- la sobrecarga de operadores relacionales
- la sobrecarga de operadores aritméticos
- la sobrecarga de operadores sobre flujos

Muchos de estos ejercicios amplían las clases diseñadas e implementadas como solución a los ejercicios propuestos en la *Relación de Problemas III (Clases I)*. En cualquier caso, todos los ejercicios deben estar **completamente implementados y modularizados** continuando y complementando el trabajo ya realizado.

1. Ampliar la clase `VectorDinámico` de datos de tipo `TipoBase` con los siguientes métodos:

- a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribir el destructor en base a esta estrategia.
- b) Sobregargar el operador `[]` para que sirva de operador de acceso a los elementos del vector dinámico y pueda actuar tanto como *lvalue* como *rvalue*.
- c) Sobregargar los operadores relacionales binarios `==` y `!=` para comparar dos vectores dinámicos. Dos vectores serán iguales si tienen el mismo número de casillas ocupadas y los contenidos son iguales y en las mismas posiciones.
- d) Sobregargar los operadores relacionales binarios `>`, `<`, `>=` y `<=` para poder comparar dos vectores dinámicos. Usar un criterio similar al que se sigue en la comparación de dos cadenas de caracteres clásicas.
- e) Considerar una implementación **nueva** para *redimensionar* un vector dinámico: emplear los operadores binarios `+`, `-`, `+=` y `-=` de manera que, por ejemplo:

Si `v` es un `VectorDinámico`, la instrucción `v = v+1` crea un `VectorDinámico` con una casilla más que `v`, lo rellena a partir de éste y lo devuelve; a continuación se *asigna* a `v`. Si `v` es un `VectorDinámico`, la instrucción `VectorDinámico v2 = v+1` crea un vector dinámico con una casilla más que `v`, lo rellena a partir de `v` y lo devuelve; a continuación se *se crea* `v2` a partir del vector dinámico así construido.

Si `v` es un `VectorDinámico`, la instrucción `v -= 10` crea un `VectorDinámico` con 10 casillas menos que `v`, lo rellena a partir de éste (descartando las 10 últimas) y lo devuelve; a continuación se *asigna* a `v`. Se sobreentiende que antes de realizar esta operación estamos seguros que el número de casillas *ocupadas* seguirá siendo menor o igual que el número máximo de casillas *disponibles* que tendremos después de la reducción.

- f) Sobreescribir los operadores << y >> para leer/escribir un vector dinámico.

Notas:

- Para la implementación del operador >> leerá una secuencia indefinida de *valores*, hasta que se introduzca el valor *. Los valores se leerán en una *cadena de caracteres*, y sólo se convertirán al tipo `TipoBase` cuando se verifique que son válidos para su almacenamiento (no se ha introducido el terminador (*)).
- Los valores siempre se guardarán *al final*.

2. Ampliar la clase `Matriz2D-1` con los siguientes *métodos*:

- a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribir el destructor en base a esta estrategia.
- b) Una sobrecarga alternativa del operador de asignación, que recibe como argumento un dato de tipo `TipoBase` e inicia **toda** la matriz al valor especificado.
- c) Sobregargar el operador `()` para que sirva de operador de acceso a los elementos de la matriz dinámica y pueda actuar tanto como *lvalue* como *rvalue*.
- d) Sobregargar los operadores unarios `+` y `-`.
- e) Sobregargar los operadores `==` y `!=` para comparar dos matrices dinámicas: serán iguales si tienen el mismo número de filas y columnas, y los contenidos son iguales y en las mismas posiciones.
- f) Sobreescribir el operador << para mostrar el contenido de una matriz dinámica.

3. Ampliar la clase `Matriz2D-2`.

Empleando la representación básica conocida, se trata de implementar los mismos métodos que en el problema 2.

4. Ampliar la clase `Lista` de datos de tipo `TipoBase` con los siguientes *métodos*:

- a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribir el destructor en base a esta estrategia.
- b) Sobregargar el operador `[]` para que sirva de operador de acceso a los elementos de la lista y pueda actuar tanto como *lvalue* como *rvalue*. El índice hace referencia a la posición, de tal manera que 1 indica el primer nodo, 2 el segundo, etc.)
- c) Sobreescribir los operadores << y >> para leer/escribir una lista.

Notas:

- Para la implementación del operador >> leerá una secuencia indefinida de *valores*, hasta que se introduzca el valor *. Los valores se leerán en una *cadena de caracteres*, y sólo se convertirán al tipo `TipoBase` cuando se verifique que son válidos para su almacenamiento (no se ha introducido el terminador (*)).
- Los valores siempre se guardarán *al final*.

5. Ampliar la clase `Pila` de datos de tipo `TipoBase` con los siguientes *métodos*:

- a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribir el destructor en base a esta estrategia.
- b) Sobreescribir el operador <<.

6. Ampliar la clase `Cola` de datos de tipo `TipoBase` con los siguientes *métodos*:

- a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribir el destructor en base a esta estrategia.
- b) Sobreescribir el operador `<<`.

7. Implementa una clase `Conjunto` que permita manipular un conjunto de elementos de tipo `TipoBase`.

Para la representación interna de los datos usar una *lista* de celdas enlazadas. El orden de los elementos no es importante desde un punto de vista teórico, pero aconsejamos que se mantengan los elementos ordenados para facilitar la implementación de los métodos de la clase.

La clase `Conjunto` debe contener, al menos, las siguientes operaciones:

- a) Constructor sin argumentos: crea un conjunto vacío.
- b) Constructor con un argumento de tipo `TipoBase`: crea un conjunto con un único elemento (el proporcionado como argumento).
- c) Constructor de copia (empleando código reutilizable).
- d) Destructor (empleando código reutilizable).
- e) Método que consulta si el conjunto está *vacío*.
- f) Sobrecarga del operador de asignación (empleando código reutilizable).
- g) Método que nos diga cuantos elementos tiene el conjunto.
- h) Método que reciba un dato de tipo `TipoBase` y consulte si pertenece al conjunto.
- i) Sobreescribir los operadores `<<` y `>>` para leer/escribir un `Conjunto`.

Notas:

- Para la implementación del operador `>>` leerá una secuencia indefinida de *valores*, hasta que se introduzca el valor `*`. Los valores se leerán en una *cadena de caracteres*, y sólo se convertirán al tipo `TipoBase` cuando se verifique que son válidos para su almacenamiento (no se ha introducido el terminador `(*)`).
 - Evidentemente, **no se permiten elementos repetidos**.
- j) Método que devuelva un `VectorDinamico` con todos los elementos del conjunto.
 - k) Sobregargar los operadores relacionales binarios `==` y `!=` para comparar dos conjuntos. Dos conjuntos serán iguales si tienen el mismo número de elementos y los mismos valores (independientemente de su posición).
 - l) Sobreescribir el operador binario `+` para calcular la **unión** de dos conjuntos. Responderá a las siguientes situaciones:
 - Si `A` y `B` son datos de tipo `Conjunto`, `A+B` será otro dato de tipo `Conjunto` y contendrá $A \cup B$
 - Si `A` es un dato de tipo `Conjunto` y `a` es un dato de tipo `TipoBase`, `A+a` será un dato de tipo `Conjunto` y contendrá $A \cup \{a\}$
 - Si `A` es un dato de tipo `Conjunto` y `a` es un dato de tipo `TipoBase`, `a+A` será un dato de tipo `Conjunto` y contendrá $\{a\} \cup A$

- m) Sobrecribir el operador binario $-$ para calcular la **diferencia** de dos conjuntos. Responderá a las siguientes situaciones:
- Si A y B son datos de tipo `Conjunto`, $A-B$ será otro dato de tipo `Conjunto` y contendrá $A - B$, o sea, el resultado de quitar de A los elementos que están en B .
 - Si A es un dato de tipo `Conjunto` y a es un dato de tipo `TipoBase`, $A-a$ será un dato de tipo `Conjunto` y contendrá $A - \{a\}$, o sea, el resultado de eliminar del conjunto A el elemento a .
- n) Sobrecribir el operador binario $*$ para calcular la **intersección** de dos conjuntos. Responderá a las siguientes situaciones:
- Si A y B son datos de tipo `Conjunto`, $A*B$ será otro dato de tipo `Conjunto` y contendrá $A \cap B$
 - Si A es un dato de tipo `Conjunto` y a es un dato de tipo `TipoBase`, $A*a$ será un dato de tipo `Conjunto` y contendrá $A \cap \{a\}$
 - Si A es un dato de tipo `Conjunto` y a es un dato de tipo `TipoBase`, $a*A$ será un dato de tipo `Conjunto` y contendrá $\{a\} \cap A$