Relación de ejercicios tema 2: punteros y gestión dinámica de memoria

Metodología de la programación, 2017-2018

Contenido:

1 Problemas básicos con punteros

1.1 Uso de punteros

1. Describa 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;
}
```

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;
}
```

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;
}
```

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;
}
```

2. Implementa una función que reciba un puntero a entero y que:

   (a) Eleve al cuadrado el dato apuntado.

   (b) Ponga a cero el puntero.

   Tal función podría usarse de la siguiente forma:

```
int a=6;
int *q;
q = &a;
elevarAlCuadrado(q);
cout << a << q; // Debería salir 36 0
```

3. Indica qué ocurre en las siguientes situaciones:

   (a)
```
int *p;
const int a=2;
p = &a;
```

   (b)
```
const int *p;
int a;
p = &a;
*p = 7;
a = 8;
```

   (c)
```
int entero=10;
const int enteroConst=20;
int *v1;
const int* v3;
v1=&entero;
v3=&enteroConst;
```

```
int* const v2=v1;
const int* const v4=v3;
v1 = v2;
v1 = v3;
v1 = v4;
```

```
int* const v2=v3;
```

```
int* const v2=v4;
```

```
int* const v2=v1;
v3 = v1;
v3 = v2;
v3 = v4;
```

```
const int* const v4=v1;
```

```
int* const v2=v1;
const int* const v4=v2;
```

```
int* const v2=v1;
const int* const v4=v3;
*v1=*v2;
*v1=*v3;
*v1=*v4;
```

```
int* const v2=v1;
const int* const v4=v3;
*v2=*v1;
*v2=*v3;
*v2=*v4;
```

```
int* const v2=v1;
const int* const v4=v3;
*v3=*v1;
*v3=*v2;
*v3=*v4;
```

```
int* const v2=v1;
const int* const v4=v3;
*v4=*v1;
*v4=*v2;
*v4=*v3;
```

1.2 Aritmética de punteros: punteros y arrays

4. Declare una variable `v` como un array de 1000 enteros. Escriba un trozo de código que recorra el array y modique 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.

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

6. Dado un array de 10 elementos, haz un bucle que busque el máximo y el mínimo (sin usar el operador `[]`). Al acabar el bucle tendremos un puntero apuntando a cada uno de ellos.

2 Punteros y funciones

2.1 Funciones que reciben arrays

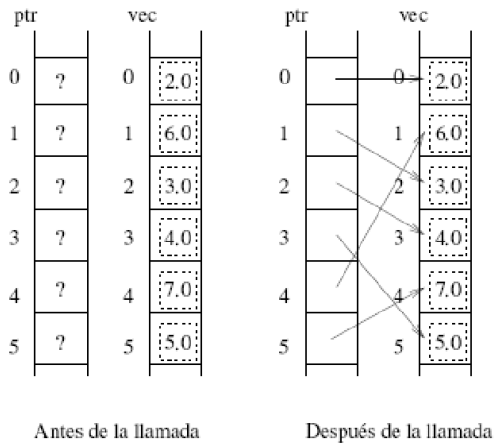
7. Implemente las siguientes funciones sobre cadenas de caracteres estilo C usando aritmética de punteros (sin usar el operador `[]`):

   (a) Función `comparar_cadenas` que compara dos cadenas. Devuelve un valor negativo si la primera es más *pequeña*, positivo si es más *grande* y cero si son *iguales*.

   (b) Función `insertar_cadena` que inserte una cadena dentro de otra, en una posición dada. Se supone que hay suficiente memoria en la cadenas de destino.

   Se supone que no es necesario pasar el tamaño de las cadenas (recordad que el carácter nulo delimita el final de la cadena).

8. Escriba una función `ordenacionPorBurbuja()` que reciba como entrada un array de números junto con su longitud y que nos devuelva un array de punteros a los elementos del array de entrada de forma que los elementos apuntados por dicho array de punteros estén ordenados (véase la siguiente figura).



Debe usarse el algoritmo de ordenación de la burbuja. Este algoritmo consiste en hacer una comparación entre cada dos elementos consecutivos desde un extremo del vector hasta el otro, de forma que el elemento de ese extremo queda situado en su posición final. Una posible implementación es la siguiente:

```
void ordenacionBurbuja (int vec[], int n) {
   for (int i=n-1; i>0; --i) {Seleccion
      for (int j=0; j<i; ++j) {
         if (vec[j] > vec[j+1]) {
            int aux = vec[j];
            vec[j] = vec[j+1];
            vec[j+1]= aux;
         }
      }
   }
}
```

Note que el array de punteros debe ser un parámetro de la función, y estar reservado previamente a la llamada con un tamaño, al menos, igual al del array. 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 la función, permita:

(a) Ordenando punteros, mostrar los elementos del array, ordenados.

(b) Ordenando punteros, mostrar los elementos de la segunda mitad del array, ordenados.

sin modificar el array de datos `vec`.

2.2 Punteros a funciones

9. El algoritmo de ordenación del ejercicio anterior nos ordena el array en orden creciente. La ordenación en orden decreciente se obtendría si cambiamos en la comparación del `if` , el operador `>` por el operador `<`. Para crear un algoritmo genérico podemos introducir un nuevo parámetro a la función `ordenacionPorBurbuja()` que indique el orden de los elementos. En nuestro caso, podemos pasarle la función que indica el orden que hay entre dos enteros, es decir, un parámetro adicional con un puntero a función que calcula el orden. Por ejemplo, podemos crear la siguiente función:

```
int ordenCreciente(int l, int r)
{
   return l-r;
}
```

La función devuelve un número negativo si el primero es menor que el segundo, un positivo si es al contrario, o un cero en caso de que sean iguales. Usando esta función, podemos indicar en el algoritmo de ordenación `ordenacionPorBurbuja()` que queremos un orden de menor a mayor, cambiando la comparación del `if` por:

```
if(ordenCreciente(vec[j], vec[j+1]) > 0)
```

Si queremos obtener un orden de mayor a menor, podemos usar otra función:

```
int ordenDecreciente(int l, int r)
{
  return r-l;
}
```

Ahora tendríamos que poner el `if` de `ordenacionPorBurbuja()` de la siguiente forma:

```
if(ordenDecreciente(vec[j], vec[j+1]) > 0)
```

Añada un nuevo parámetro puntero a función, a la función `ordenacionPorBurbuja()` del ejercicio anterior, según lo explicado más arriba. La idea es permitir la función `ordenacionPorBurbuja()` pueda usarse para ordenar un array de menor a mayor o de mayor a menor. Si llamamos a `ordenacionPorBurbuja()` con la función `ordenCreciente()`, el array se ordenaría de menor a mayor, y si usamos `ordenDecreciente()`, se ordenaría de mayor a menor.

3 Punteros, struct y class

10. Represente gráficamente la disposición en memoria de las variables del programa mostrado a continuación, e indique lo que escribe la última sentencia de salida.

```
#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 ;
}
```

11. Represente gráficamente la disposición en memoria de las variables del programa mostrado a continuación, e indique lo que escribe la última sentencia de salida. Tenga en cuenta que el operador `->` tiene más prioridad que el operador `*`.

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

struct SB; // declaración adelantada
struct SC; // declaración adelantada
struct SD; // declaración adelantada

struct SA {
    int dat ;
    SB *p1 ;
} ;

struct SB {
    int dat ;
    SA *p1 ;
    SC *p2 ;
} ;

struct SC {
    SA *p1 ;
    SB *p2 ;
    SD *p3 ;
} ;

struct SD {
    int *p1 ;
    SB *p2 ;
} ;

int main ( int argc , char * argv [ ] )
{
    SA a ;
    SB b ;
    SC c ;
    SD d ;
    int dat ;

    a.dat = b.dat = dat = 0 ;

    a.p1 = &b ;
    b.p1 = &a ;
    b.p2 = &c ;
    c.p1 = b.p1 ;
    c.p2 = &(*(a.p1)) ;
    c.p3 = &d;
    d.p1 = &dat;
    d.p2 = &(*(c.p1)->p1);
    *(d.p1) = 9;
    (*(b.p2)->p1).dat = 1;
    (*((b.p2)->p3->p2)->p1).dat = 7 ;
    *((*((*(c.p3->p2)).p2->p3)).p1) = (*(b.p2)).p1->dat + 5 ;
    cout << "a.dat=" << a.dat << " b.dat=" << b . dat << " dat=" << dat << endl ;
}
```

4 Memoria dinámica

12. Describa la salida del siguiente programa:

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


int main (){
   int *p1, *p2;

   p1 = new int;
   *p1 = 42;
   p2 = p1;
   cout << *p1 << " y " << *p2 << endl;

   *p2 = 53;
   cout << *p1 << " y " << *p2 << endl;

   p1 = new int;
   *p1 = 88;
   cout << *p1 << " y " << *p2 << endl;
}
```

13. Dadas las siguientes declaraciones

```
struct Electrica {
   char corriente[30];
   int voltios;
};

Electrica *p = new Electrica(), *q = new Electrica();
```

se pide averiguar qué hacen cada una de las siguientes sentencias. ¿Hay alguna inválida?

```
a.  strcpy(p->corriente,"ALTERNA");   e.  strcpy(p->corriente,"ALTA");
b.  p->voltios = q->voltios;          f.  p->corriente = q->voltios;
c.  *p = *q;                          g.  p = 54;
d.  p = q;                            h.  *q = p;
```

4.1 Vectores dinámicos

14. Queremos hacer un programa `leerCaracteres` que lea uno a uno los caracteres de un fichero de texto, los cargue en un array dinámico de caracteres y luego lo muestre en la salida estándar. El array dinámico será controlado con un puntero a `char` (`arraychar`) y un entero con el número de caracteres en el array. El array tendrá en todo momento el tamaño justo necesario para contener los caracteres leídos. La función `main()` se da a continuación.

```
#include <iostream>
#include <fstream> // ifstream
using namespace std;

int main(int argc, char* argv[])
{
   char* arraychar;
   int nCaracteres;

   inicializar(arraychar, nCaracteres);
   if (argc==1)
      leer(cin, arraychar, nCaracteres);
   else {
      ifstream flujo(argv[1]);
      if (!flujo) {
         cerr << "Error: Fichero " << argv[1] << " no válido." << endl;
         return 1;
      }
      leer(flujo, arraychar, nCaracteres);
   }
   mostrar(cout, arraychar, nCaracteres);
   liberar(arraychar, nCaracteres); // Libera la memoria dinámica reservada
}
```

Como puede verse, el programa puede ser ejecutado de dos formas:

(a) Dando un argumento de entrada a main con el nombre del fichero de texto:

```
prompt> ./leerCaracteres fichero.txt
Este es un fichero de prueba que contiene
solo dos líneas.
```

(b) Leyendo los datos de la entrada estándar:

```
prompt> cat datos | ./leerCaracteres
Este es un fichero de prueba que contiene
solo dos líneas.
```

Añade las siguientes funciones:

(a) `inicializar` para inicializar el array dinámico.

(b) `liberar` para liberar la memoria dinámica ocupada por el array dinámico.

(c) `mostrar` para mostrar en un flujo de salida los caracteres del array dinámico.

(d) `redimensionar` que amplía el tamaño de un array dinámico con un determinado valor entero `incremento` recibido como parámetro.

(e) `aniadir` que añade un carácter al final de un array dinámico, aumentando previamente su tamaño en uno.

(f) `leer` para leer los caracteres uno a uno de un flujo de entrada, y guardarlos en un array dinámico recibido como parámetro.

La función `leer` se da a continuación:

```
void leer(istream& flujo, char* &array, int& nchar){
   char caracter;

   while(flujo.get(caracter)){
      aniadir(array, nchar, caracter);
   }
}
```

15. Queremos hacer un programa (`mostrar`) que lea los números enteros que tenemos almacenados en un fichero de texto y que los cargue en un array dinámico `VectorSD`. El código de la función `main()` se da a continuación.

```
#include <iostream>
#include <fstream> // ifstream
using namespace std;

int main(int argc, char* argv[])
{
   VectorSD v;
   if (argc==1)
      v.leer(cin);
   else {
      ifstream flujo(argv[1]);
      if (!flujo) {
         cerr << "Error: Fichero " << argv[1] << " no válido." << endl;
         return 1;
      }
      v.leer(flujo);
   }
   v.mostrar(cout);
   v.liberar(); // Libera la memoria dinámica reservada
}
```

Como puede verse, el programa puede ser ejecutado de dos formas:

(a) Dando un argumento de entrada a main con el nombre del fichero de texto:

```
prompt> ./mostrar datos.txt
7 4 4 -3 13 16 16 -6 9 21
```

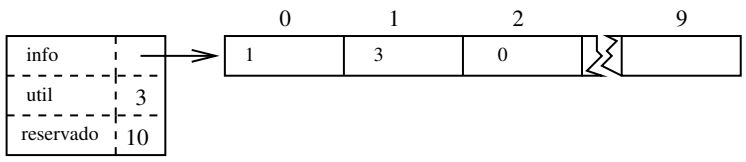
(b) Leyendo los datos de la entrada estándar:

```
prompt> cat datos | ./mostrar
7 4 4 -3 13 16 16 -6 9 21
```

Construye la clase `VectorSD` con los siguientes datos miembro:

```
class VectorSD {
   int *info;
   int util;
   int reservado;
};
```

donde `info` es un puntero que mantiene la dirección de una secuencia de enteros, `util` indica el número de componentes de la secuencia y `reservado` indica el número de posiciones reservadas de la memoria dinámica para almacenar la secuencia de datos. La siguiente figura muestra un ejemplo de este tipo de representación.



Añadir a la clase los siguientes constructores y métodos:

(a) Constructor sin parámetros que inicialice una variable de tipo `VectorSD` reservando 10 casillas de memoria dinámica y ponga el número de componentes usadas a 0.

(b) Constructor de la clase que inicialice una variable de tipo `VectorSD` reservando **n** casillas de memoria dinámica y ponga el número de componentes usadas a 0.

(c) Método que nos devuelva el dato (`double`) almacenado en una determinada posición (`int`).

```
int getDato(int posicion) const {
   ...
}
```

(d) Método que devuelva el número de datos guardados actualmente en un objeto `VectorSD`.

```
int nElementos() const {
   ...
}
```

(e) Método que añada un nuevo dato a un objeto `VectorSD`. Considerar el caso de que la inclusión del nuevo valor sobrepase el número de elementos reservados. En este caso, realojar el array reservando el doble de posiciones.

```
void aniadir(int dato){
   ...
}
```

(f) Método que copie el objeto `VectorSD` recibido como parámetro en el objeto actual. La copia debe reservar memoria para almacenar solo las componentes usadas del array dinámico del objeto recibido por parámetro. Recuerde, que en caso de que el objeto ya contenga memoria dinámica reservada, debe liberarla antes.

```
void copia(const VectorSD &vector){
   ...
}
```

(g) Método que libere la memoria reservada por un objeto `VectorSD`.

```
void liberar(){
   ...
}
```

aniadir(dato);

(h) Método para mostrar los elementos de un objeto `VectorSD` (separados por blancos) en un flujo de salida. Este método tendrá la siguiente forma:

```
void mostrar(ostream& flujo) const {
  for(int i=0; i<util;i++){
    flujo << info[i] << " ";
  }
  flujo<<endl;
}
```

(i) Método que recibe un flujo de entrada y carga uno a uno los datos enteros que contiene, en el objeto `VectorSD` hasta que llega al final de la entrada.

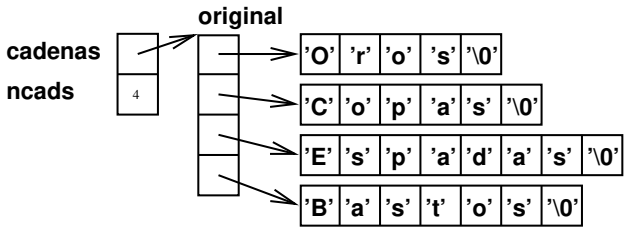
```
void leer(istream& flujo){
   int dato;

   while(flujo>>dato){
      aniadir(dato);
   }
}
```

4.2 Matrices dinámicas

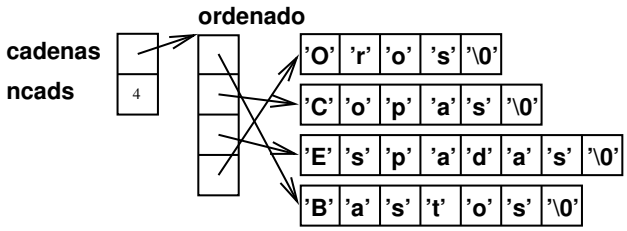
16. Queremos hacer un programa `ordenaLineas` que ordene por orden alfabético una serie de cadenas-C. Para ello definimos el siguiente tipo:

```
struct ListaCadenas{
  char** cadenas;
  int ncads;
}
```

Este tipo representa un array de cadenas-C que almacenaremos en memoria dinámica. Por ejemplo, la siguiente figura muestra un ejemplo con cuatro cadenas:



La siguiente figura muestra la estructura tras ordenar el array de cadenas.



Debemos hacer un programa que lea las cadenas de un fichero o de la entrada estándar, las vaya guardando en el array de cadenas y luego las ordene por orden alfabético. Guardaremos en memoria dinámica tanto el array de cadenas como cada una de las cadenas leídas. El array de cadenas tendrá en todo momento el tamaño justo necesario para guardar el número de cadenas que almacene. Cada cadena se almacena también en memoria dinámica con el tamaño justo necesario para guardar sus caracteres.

Para leer cada cadena, podemos hacer uso de la función `getline()` para así permitir espacios en blanco.

La función `main()` se da a continuación:

```
#include <iostream>
#include <fstream> // ifstream
using namespace std;

int main(int argc, char* argv[])
{
   ListaCadenas lista;

   crear(lista);
   if (argc==1)
      leer(cin, lista);
   else {
      ifstream f(argv[1]);
      if (!f) {
         cerr << "Error: Fmostrarichero " << argv[1] << " no válido." << endl;
         return 1;
      }
      leer(f, lista);
   }
   cout << "Original:" << endl;
   imprimir(cout, lista);
   ordenar(lista);
   cout << "\nResultado:" << endl;{\tt
   imprimir(cout, lista);
   liberar(lista); // Libera la memoria dinámica reservada
}
```

Como puede verse, el programa puede ser ejecutado de dos formas:

(a) Dando un argumento de entrada a main con el nombre del fichero de texto:

```
prompt> ./ordenaLineas fichero.txt
Original:
Úbeda Martínez, Alejandro
```

```
Sabiote Alvarado, Lucía
Cano Granados, Maite

Resultado:
Cano Granados, Maite
Sabiote Alvarado, Lucía
Úbeda Martínez, Alejandro
```

(b) Leyendo los datos de la entrada estándar:

```
prompt> cat fichero.txt | ./ordenaLineas
Original:
Úbeda Martínez, Alejandro
Sabiote Alvarado, Lucía
Cano Granados, Maite

Resultado:
Cano Granados, Maite
Sabiote Alvarado, Lucía
Úbeda Martínez, Alejandro
```

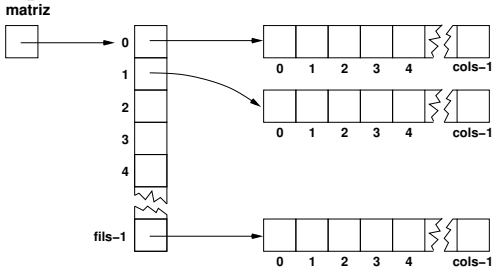
El programa contendrá las siguientes funciones:

- La función `main` para probar que todo funciona de forma correcta.
- La función `void crear(ListaCadenas& listaCadenas)` que inicializar un array de cadenas como vacío.
- La función `void imprimir(ostream& flujo, const ListaCadenas& listacadenas)` para mostrar en un flujo de salida las cadenas almacenadas en un array de cadenas (`ListaCadenas`).
- La función `void redimensionar(ListaCadenas& listaCadenas,int newSize)`.
- La función `void aniadir(ListaCadenas& listaCadenas, const char* cadena)`.
- La función `void leer(iostream& flujo, ListaCadenas& listaCadenas)` que todas las líneas que contenga un flujo de entrada y las guarde en el objeto recibido como parámetro.
- La función `void liberar(ListaCadenas& listaCadenas)` libera la memoria dinámica el objeto `listaCadenas`.
- La función `void ordenar(ListaCadenas& listaCadenas)` para ordenar las cadenas por orden alfabético.

La función `leer()` se da a continuación:

```
void leer(iostream& flujo, ListaCadenas& listaCadenas){
   const int NCARACTERES=1000; // Suponemos líneas con menos de 1000 char
   char linea[NCARACTERES];

   while(flujo.getline(linea,NCARACTERES)){
      aniadir(listaCadenas, linea);
   }
}
```

17. Supongamos que para definir matrices bidimensionales dinámicas usamos una estructura como la que aparece en la siguiente figura que denominaremos **Matriz2D_1**:



a) Define cómo sería la clase para representar una matriz según esa estructura.

b) Añade el constructor de la clase, para crear una matriz con `nf` filas y `nc` columnas.

c) Implementa métodos para obtener el número de filas de la matriz, el número de columnas y para modificar el dato que hay en una determinada fila y columna.

d) Implementa un método que libere la memoria dinámica ocupada por un objeto de la clase.

e) Construir un método que escriba en la salida estándar todas las componentes de una matriz bidimensional dinámica como la que se ha definido anteriormente.

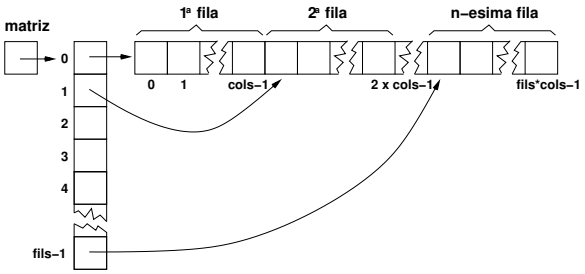
f) Construir un método que dada una matriz de este tipo cree una copia.

g) Implementar un método que extraiga una submatriz de una matriz bidimensional `Matriz2D_1`. Como argumento del método se introduce desde qué fila y columna y hasta qué fila y columna se debe realizar la copia de la matriz original.

h) Desarrollar un método que elimine una fila de una matriz bidimensional `Matriz2D_1`. Obviamente, la eliminación de una fila implica el desplazamiento hacia arriba del resto de las filas que se encuentran por debajo de la fila eliminada.

i) Realizar un método como el anterior, pero que en vez de eliminar una fila, elimine una columna.

18. Supongamos que ahora decidimos utilizar una forma diferente para representar las matrices bidimensionales dinámicas a la que se propone en el ejercicio anterior. En este caso, usaremos una estructura semejante a la que aparece en la siguiente figura que denominaremos **Matriz2D_2**:



a) Realizar los apartados a), b), c), d), e), f), g) y h) usando esta nueva representación de matrices bidimensionales dinámicas.

b) Construir un método que dada una matriz bidimensional dinámica `Matriz2D_1` realice una copia de la misma en una matriz bidimensional dinámica `Matriz2D_2`.

c) Desarrollar un método que realice el paso inverso al propuesto en el ejercicio anterior.

4.3 Listas de celdas enlazadas

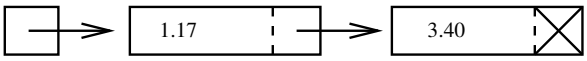
19. Dada la siguiente definición de tipo de dato abstracto:

```
struct Celda{
   double info;
   Celda *sig;
};

class Lista{
   Celda *l;
public:
   ....
};
```

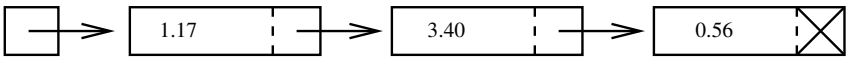
donde `info` es una variable `double` y `sig` es un puntero que apunta a una variable `Celda`. Añadir los siguientes métodos y constructores:

a) Constructor de la clase `Lista`, de forma que inicialice una variable de tipo `Lista`, como una lista vacía.

b) Método que permita añadir al final de una secuencia de celdas enlazadas un nuevo dato. Por ejemplo, dada la siguiente situación:



el resultado de añadir el valor `0.56` al objeto correspondiente sería:



NOTA: El puntero de la última celda contiene un 0.

c) Método que permita eliminar la última celda de un objeto de la clase.

d) Método que elimine y libere toda la información contenida en un objeto `Lista`.

e) Método que muestre en la salida estándar el contenido completo de un objeto de la clase.

f) Método que inserte un nuevo dato detrás de una celda concreta (cuya dirección de memoria se pasa como parámetro al método).

g) ¿Podría utilizarse el método anterior para insertar una celda al principio de la lista? Si no es así, haced los cambios necesarios para que sea posible.

h) Método que, dada la dirección de memoria de una celda, la elimine de la lista.

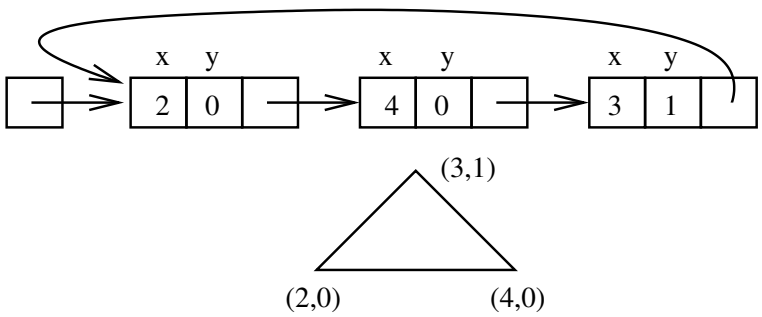
i) ¿Sería posible utilizar el método anterior para eliminar la primera de las celdas de la estructura de celdas enlazadas? Si no es así, haced los cambios necesarios para que sea posible.

j) Método que devuelva un puntero a la celda que se encuentra en una determinada posición (entera) en la lista.

20. Se desea desarrollar una clase 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 siguiente representación:

```
struct Punto2D{
   double x;
   double y;
};

struct Nodo{
   Punto2D punto;
   Nodo *sigPunto;
};

class Poligono{
  Nodo* Poligono;
}
```

Dada esta definición, un polígono se representa como una secuencia circular ordenada de nodos enlazados, por ejemplo, el triángulo de puntos (2,0),(4,0) y (3,1) se representa de la siguiente forma:



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

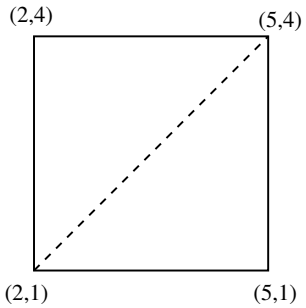
a) Construir un método en la clase `Poligono` que determine el número de lados que contiene la figura almacenada en una variable de tipo `Poligono`.

b) Suponiendo que existe un método llamado `pintaRecta (Punto2D p1, Punto2D p2)` que pinta una recta entre los 2 puntos que se le pasan como argumento, construir un método que permita pintar la figura que representa una determinada variable `Poligono`.

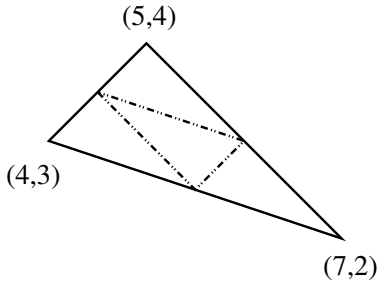
c) Implementar un método que permita crear en una variable de tipo `Poligono`, un triángulo a partir de los tres puntos que lo definen.

d) Desarrollar un método que permita liberar la memoria reservada por una variable `Poligono`.

e) Sabiendo que una variable `Poligono` almacena un cuadrado, implementar un método que devuelva los dos triángulos que resultan de unir mediante una recta la esquina inferior izquierda del cuadrado con su esquina superior derecha.



f) Construir un método que a partir de una variable `Poligono` que representa un triángulo devuelva el triángulo formado por los puntos medios de las rectas del triángulo original.



g) Desarrollar un constructor que permita construir un polígono regular de **n** lados inscrito en una circunferencia de radio **r** y centro **(x,y)**.