DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Metodología de la Programación

Grado en Ingeniería Informática

Problemas

Curso 2013/2014

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 efecuará 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 basatará 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/iostream/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 incluído).

   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 declaracón:

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

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

a) El elemento mayor del vector.

b) El elemento mayor de la primera mitad.

c) 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).

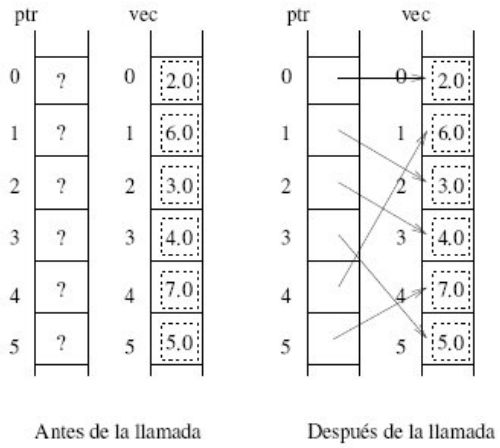

Figura 2: Resultado de ordenar el vector de punteros

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:

a) Ordenando punteros, mostrar los elementos del vector, ordenados.

b) 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 -> 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 }
```

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 mostará 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 almacernar 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 almacernar 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.

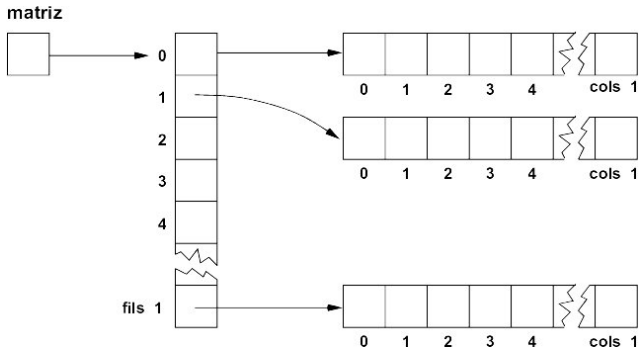

Figura 4: Tipo `Matriz2D1`: datos guardados en filas independientes

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.

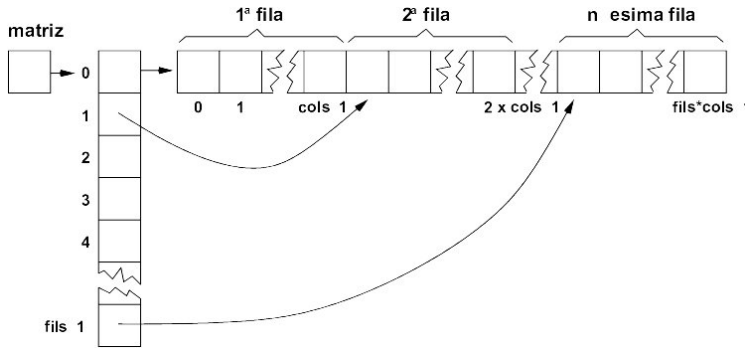

Figura 5: Tipo `Matriz2D2`: datos guardados en una sola fila

   a) Reescribir todos los módulos propuestos en el ejercicio 8 usando esta nueva representación.
   b) 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.
   c) 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,l) se representa como se indica en la figura 6.

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

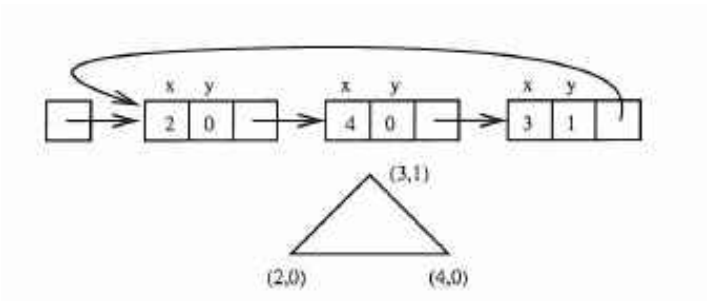

Figura 6: Representación de un polígono y un ejemplo (un triángulo)

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

a) 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.

b) Desarrollar un módulo que permita liberar la memoria reservada por una variable `Poligono`.

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

d) Suponiendo que existe una función llamada `PintaRecta (Punto2D pl, 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`.

e) 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).

f) 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).

g) 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).

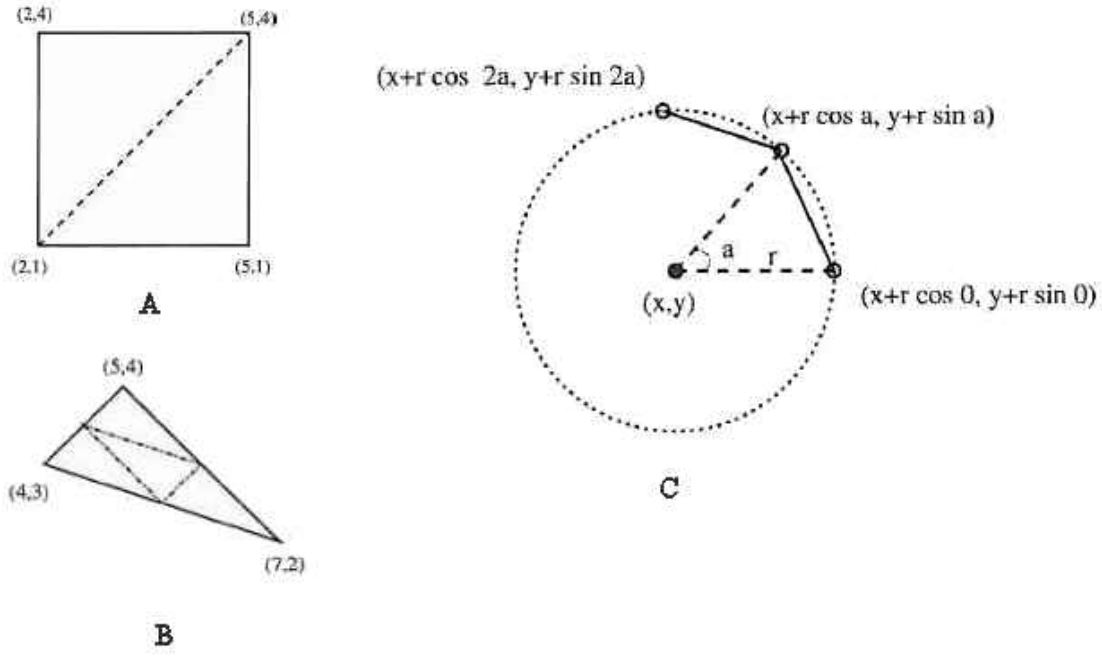

Figura 7: A y B) Cálculo de algunos triángulos. C) Construcción de un polígono inscrito en una circunferencia

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 modularmente la clase `MiVectorDinamico` para trabajar con vectores de datos de tipo `TipoBase`. Los vectores dinámicos tendrán un 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 con un número de casillas predeterminado.
   b) Constructor con un argumento, que crea un vector dinámico con un número de casillas indicado en el argumento.
   c) Constructor con dos argumentos, que crea un vector dinámico con un número de casillas indicado en el primer argumento. Inicia todas la casillas al valor indicado en el segundo argumento. Por defecto tomará el valor 0.
   d) Destructor.
   e) Métodos para consultar el número de casillas ocupadas/reservadas.
   f) Método para añadir un valor (siempre al final).
   g) Método para devolver el valor que ocupa una posición dada.
   h) Método para establecer/consultar el tipo de redimensionamiento.

   Escribir una función `main()` que permita probar la clase.

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.
   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.

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);
```

h) Método que inicializa todas las casillas de la matriz al valor indicado como argumento. Si no se especifica ninguno, inicia todas las casillas al valor *nulo*.

Escribir una función `main()` que permita probar la clase.

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

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

   Escribir una función `main()` que permita probar la clase.

4. Implementar la clase `Lista` para trabajar con listas (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.

   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 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: $< 5, 6, 8, 4, 3, 2, 9 >$

   h) 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.)

i) Método para añadir un valor en la lista. La adición siempre se hace al final de la lista. Modifica la lista. Responderá al siguiente prototipo:
```
void Aniadir (TipoBase val);
```
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.)

k) Método que inicializa todos los nodos al valor indicado como argumento. Si no se especifica ninguno, inicia todos los nodos al valor *nulo*.

Escribir una función `main()` que permita probar la clase.

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** (**l**ast **i**n **f**irst **o**ut).

   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.

   Escribir una función `main()` que permita probar la clase.

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** (**F**irst **i**n **f**irst **o**ut).

   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.

   Escribir una función `main()` que permita probar la clase.

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 de acceso `[]` y `()`
- la sobrecarga de operadores relacionales
- la sobrecarga de operadores aritméticos
- la sobrecarga de operadores combinados
- 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.

   b) Una sobrecarga alternativa del operador de asignación, que recibe como argumento un dato de tipo `TipoBase` e inicia **todo** el vector al valor especificado.

   c) Sobregargar el operador `[]` para que sirva de operador de acceso a los elementos del vector dinámico y pueda actuar tanto como *lvalue* como *rvalue*.

   d) 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.

   e) 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.

   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.

   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 binarios `+` y `-` para implementar la suma y resta de matrices. Si los dos operadores son de tipo `Matriz2D-1` sólo se hará la suma o la resta si las dos matrices tienen las mismas dimensiones. Si no fuera así, devolverá una matriz vacía. Se admite la posibilidad de que algún operador fuera de tipo `TipoBase`.

   Importante: ninguno de los operandos se modifica.

   f) Sobregargar los operadores combinados `+=`, `-=`, `*=` y `/=` de manera que el argumento explícito sea de tipo `TipoBase` y modifiquen la matriz convenientemente.

   g) 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.

   h) 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.

   b) Una sobrecarga alternativa del operador de asignación, que recibe como argumento un dato de tipo `TipoBase` e inicia **toda** la lista al valor especificado.

   c) 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.)

   d) Sobrecargar los operadores combinados `+=` y `-=` de manera que el argumento explícito sea de tipo `TipoBase` y añadan o eliminen de la lista un nodo con el valor dado por el argumento explícito.

   - El nuevo nodo se añade al final de la lista.
   - Si hay más de una instancia del valor a borrar, se borra la primera de ellas.
   - Si no hubiera ninguna instancia del valor a borrar no se hace nada.

   e) Sobrecargar los operadores binarios `+` y `-` de manera que se apliquen a dos listas, a una lista y un dato de tipo `TipoBase` o a un dato de tipo `TipoBase` y a una lista, creando una *nueva* lista.

   - El operador `+` cuando se aplica a dos listas crea una lista uniendo a la primera lista el contenido completo de la segunda lista.
   - El operador `-` cuando se aplica a dos listas crea una nueva lista, eliminando de la primera lista *la primera aparición* de los valores que aparecen en la segunda. Si no hubiera ninguna instancia del valor a borrar, no se hace nada.

- Estudie si tiene sentido implementar las tres versiones (lista/lista, lista/valor y valor/lista) para los dos operadores.
- Los operandos no se modifican.

f) Sobrecargar los operadores relacionales para comparar dos listas. Los criterios de comparación entre dos listas serán idénticos a los que determinan la relación entre dos cadenas de caracteres clásicas.

g) Sobreescribir los operadores << y >> para leer/escribir una lista.

- 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.

   b) Sobrecargar el operador combinado += para añadir un dato de tipo `TipoBase` a la pila.

   c) Sobrecargar el operador unario -- para eliminar un dato de tipo `TipoBase` de la pila.

   d) Sobrecargar el operador unario ^ para obtener el elemento de tipo `TipoBase` que ocupa la cabecera de la pila. La pila no se modifica.

   e) 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.

   b) Sobrecargar el operador combinado += para añadir un dato de tipo `TipoBase` a la cola.

   c) Sobrecargar el operador unario -- para eliminar un dato de tipo `TipoBase` de la cola.

   d) Sobrecargar el operador unario ^ para obtener el elemento de tipo `TipoBase` que ocupa la primera posición de la cola. La cola no se modifica.

   e) 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 pertence al conjunto.

i) 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).

j) 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 $\{a\} \cup A$

k) Sobreescribir 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 eleminar del conjunto A el elemento a.

l) Sobreescribir 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 $\{a\} \cap A$

m) Añadir un módulo de interface (`Transformaciones.cpp` y `Transformaciones.h`) entre las clases `Conjunto` y `VectorDinámico` que ofrezca dos funciones, con los siguientes prototipos:

```
VectorDinamico ConjuntoToVectorDinamico (Conjunto & un_conjunto);
```

Devuelve en un VectorDinamico los datos de un Conjunto

```
Conjunto VectorDinamicoToConjunto (VectorDinamico & un_vector_dinamico);
```

Devuelve en un Conjunto los datos de un VectorDinamico

n) 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**.

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas V: Gestión de E/S. Ficheros (I)

Los ejercicios propuestos en esta relación tiene como finalidad la práctica con flujos y operaciones sencillas de E/S. Se trabajará en todos los casos con la entrada/salida estándar usando los objetos `cin` y `cout`, por lo que podrá emplearse la redirección y/o encauzamiento. De hecho, recomendamos la ejecución de los programas usando redirección y/o encauzamiento, por comodidad.

Cuando en los enunciados de los problemas encuentre "lea una secuencia indefinida de . . . de la entrada estándar" el programa debe usar `cin` para leer una secuencia de datos delimitada por el *fín del fichero*:

- Si los datos se introducen desde el teclado, debe finalizarse introduciendo manualmente el *fín del fichero* (Ctrl+D en GNU/Linux ó Ctl+Z en MS-DOS/Windows).
- Si se emplea la redirección de entrada, tomando los datos de un fichero de texto, el *fín de fichero* se inserta automáticamente en el flujo de entrada.

1. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y los copie literalmente en la salida estándar.
2. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y los copie en la salida estándar, exceptuando las vocales.
3. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y muestre en la salida estándar el número total de caracteres leidos.
4. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y muestre en la salida estándar el número de líneas *no vacías* que hay en esa secuencia.

   Nota: Se entenderá que una línea es *vacía* si contiene **únicamente** el carácter `'\n'`
5. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y muestre en la salida estándar únicamente las líneas *no vacías* que hay en esa secuencia.
6. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y "comprima" todas las líneas de esa secuencia, eliminando los separadores que hubiera en cada línea. Sólo se mantendrá el carácter `'\n'`
7. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y copie en la salida estándar las líneas que **no** comiencen por el carácter `#`
8. Escribir un programa que lea una serie indefinida de números enteros de la entrada estándar y los copie, en el mismo orden, en la salida estándar.

- En la secuencia de entrada se pueden usar espacios, tabuladores o saltos de líneas (en cualquier número y combinación) para separar dos números enteros consecutivos.
- En la secuencia de salida se separan dos enteros consecutivos con un salto de línea.

9. Escribir un programa que lea una serie indefinida de números enteros de la entrada estándar y los copie, *en orden inverso*, en la salida estándar.

   - En la secuencia de entrada se pueden usar espacios, tabuladores o saltos de líneas (en cualquier número y combinación) para separar dos números enteros consecutivos.
   - En la secuencia de salida se separan dos enteros consecutivos con un salto de línea.
   - Usar un objeto `Pila` para invertir la secuencia.

10. Escribir un programa que lea una serie indefinida de números enteros de la entrada estándar y los copie, en el mismo orden, en la salida estándar.

    - En la secuencia de entrada, dos números consecutivos están separados por el carácter `*`.
    - En la secuencia de salida se separan dos enteros consecutivos con un salto de línea.

11. Escribir un programa que lea un fichero como los generados en los problemas 8, 9 y 10 y que muestre en la salida estándar la suma de todos esos números.

12. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y reciba como argumento desde la línea de órdenes un dato de tipo `char`. El programa mostrará en la salida estándar el número de caracteres leidos de la entrada estándar iguales al argumento suministrado.

    Por ejemplo: `cuenta_letra a < ElQuijote.txt` mostrará el número de caracteres `a` que hay en `ElQuijote.txt`

13. Escribir un programa que lea una secuencia indefinida de caracteres de la entrada estándar y reciba como argumento desde la línea de órdenes un dato de tipo `int`.

    El programa mostrará en la salida estándar el número de palabras leidas de la entrada estándar cuya longitud sea igual al argumento suministrado.

    Por ejemplo: `cuenta_palabras 10 < ElQuijote.txt` mostrará el número de palabras que hay en `ElQuijote.txt` que tienen 10 caracteres.

Grado en Ingeniería Informática. Metodología de la Programación. Relación de Problemas VI: Gestión de E/S. Ficheros (II)

1. Escribir un programa que reciba los nombres de dos ficheros de *texto* de la línea de órdenes. El programa creará un fichero (cuyo nombre se especifica en el segundo argumento) a partir de un fichero existente (cuyo nombre se especifica en el primer argumento) copiando su contenido y añadiendo al principio de cada línea, su número.

2. Escribir un programa similar a `diff` para comparar dos ficheros de texto. El programa imprimirá el número de la primera línea en la que difieren y el contenido de éstas.
   Por ejemplo, la ejecución de `diff Fich1 Fich2` producirá como resultado:

   ```
   (  20) Fich1: en formato binario. Estos ficheros son especialmente adecuados para
          Fich2: en formato binario. Estos ficheros, aunque no son legibles, son especialmente
   ```

   si las 19 primeras líneas de `Fich1` y `Fich2` son idénticas, y la primera diferencia se encuentra en la línea 20.

   Nota: Este programa puede ser útil para comprobar si después de encriptar y desencriptar un fichero (problema 4), obtenemos un fichero idéntico al original.

3. Escribir un programa que reciba como parámetros tres nombres de ficheros de *texto*. Los dos primeros ficheros contienen números reales y están *ordenados*. El programa tomará los datos de esos ficheros y los irá copiando ordenadamente en el tercer fichero, de forma que al finalizar esté también ordenado.

4. Escribir un programa que permita encriptar y desencriptar el contenido de un fichero de texto. Para *encriptar* sustituiremos cada letra (mayúsculas y minúsculas) por la letra que está *p* posiciones más adelante en el alfabeto (para las últimas letras ciclamos el alfabeto). Los caracteres que no sean letras se quedarán igual. Para *desencriptar* la sustitución será a la inversa. La llamada al programa se realizará con este esquema:

   ```
   codifica <ficheroE> <ficheroS> <p> <tipo>
   ```

   donde:

   - `<ficheroE>` y `<ficheroS>` son los nombres de los ficheros de entrada y salida, respectivamente
   - `<p>` es el número entero positivo que se aplica para codificar/descodificar cada uno de los caracteres.
   - `<tipo>` es una cadena de caracteres que puede valer: `enc` para encriptar y `desenc` para desencriptar.

5. Un fichero de *texto* contiene números enteros que se disponen en líneas y en cada línea están separados por espacios en blanco. Todas las líneas tienen el mismo número de elementos.

   Se trata de escribir un constructor para las clases `Matriz2D_1` y `Matriz2D_2` que reciba el nombre de un fichero con la estructura descrita y rellene las casillas de la matriz con los datos contenidos en el fichero. Se sobreentiende que los datos están guardados *por filas* en el fichero.

   Las restricciones que se imponen, y que se deben cumplir en la resolución son:

   a) El fichero sólo puede ser leído una única vez, y no pueden copiarse completo en memoria.

   b) Se desconoce *a priori* el número de líneas del fichero.

   c) Las líneas del fichero tiene una longitud *indeterminada*, aunque nunca mayor de 500.

   d) Pueden haber líneas vacías y líneas que conttengan sólo espacios y otros separadores.

   e) El número de datos de cada línea es *indeterminado*, aunque éste es *común* para todas las líneas.

f) No puede emplearse una matriz con un número de filas "tentativo": la matriz ocupará en cada momento el espacio estrictamente necesario y los datos se copiarán conforme se lean cada una de las filas.

El número de líneas de datos del fichero debe coincidir con el número de filas de la matriz, y el número de elementos de cada fila con el número de columnas.

Nota: Es posible que sea necesario añadir un nuevo método a las clases `Matriz2D_1` y `Matriz2D_2` que permita redimensionar la matriz, añadiendo una nueva fila.

6. Se dispone de ficheros de *texto* que contienen un número indeterminado de líneas, cada una de ellas con los datos correspondientes a una serie de grupos de valores reales.
Por ejemplo, una línea de entrada podría ser la siguiente:

```
3 2 3.1 0.4 5 1.0 1.0 1.0 1.0 1.0 2 5.2 4.7
```

donde puede observar que se distinguen tres grupos de datos (indicado por el primer número de la línea) y cada grupo empieza por un valor entero (2, 5 y 2) seguido por tantos valores reales como indique el valor entero que encabeza cada grupo:

3	**2**	*3.1*	*0.4*	**5**	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	**2**	*5.2*	*4.7*

Escribir un programa que escriba en la salida estándar una línea de resultado por cada línea de entrada, y en cada línea mostrará las sumas de los valores de cada grupo que la componen.

Por ejemplo, en el caso anterior, debería escribir:

```
3.5    5.0    9.9
```

El programa se ejecutará desde la línea de órdenes y permitirá:

a) Llamarlo sin ningún argumento. En este caso, los datos de entrada se leerán desde la entrada estándar.

b) Llamarlo con un argumento. El argumento corresponde al nombre del archivo con las líneas de entrada.

Las restricciones que se imponen, y que se deben cumplir en la resolución son:

a) El fichero sólo puede ser leído una única vez, y no pueden copiarse completo en memoria.

b) Se desconoce *a priori* el número de líneas del fichero.

c) Las líneas del fichero tiene una longitud *indeterminada*, aunque nunca mayor de 500.

d) Pueden haber líneas vacías y líneas que conttengan sólo espacios y otros separadores.

7. Escribir un programa que reciba el nombre de dos ficheros. El programa copiará, en el mismo orden, los números que contiene el fichero de entrada en el fichero de salida.

- El primer fichero (entrada) contiene una serie indefinida de números enteros. Es un fichero de *texto* y puede contener *espacios*, *tabuladores* o *saltos de línea* (en cualquier número y combinación) separando dos números enteros consecutivos.
- El segundo fichero (salida) es un fichero *binario*.
- El programa leerá los números y los copiará **de uno en uno**.

8. Escribir un programa con las mismas características que las descritas en el problema 7 pero que escriba en el fichero de salida **bloques de 512 bytes**.

9. Escribir un programa que lea un fichero *binario* como los generados en los problemas 7 y 8 y que muestre en la salida estándar la suma de todos esos números. Para la lectura se empleará un **buffer de 512 bytes**.

10. Escriba dos programas para transformar ficheros con datos correspondientes a una serie de grupos de valores reales (como están descritos en el problema 6), para transformar entre formato binario y texto:

    a) Un programa que transforme un fichero de texto a binario:

    ```
    text2bin <FichText> <FichBin>
    ```

    b) Un programa que transforme un fichero de binario a texto:

    ```
    bin2text <FichBin> <FichText>
    ```

    Debe optimizarse el uso de los recursos, y por tanto, se aplican las restricciones enumeradas en el problema 6.

11. Construir un programa que divida un fichero de *texto* en diferentes ficheros indicando como argumentos el nombre del fichero original y el **máximo número de líneas** que contendrá cada fichero resultante.

    Se creará un fichero de control que contendrá con los datos necesario para la reconstrucción del fichero original.

    Por ejemplo, si `Fichero` contiene 1600 líneas, la ejecución de `parte_lineas Fichero 500` genera como resultado los ficheros `Fichero_1`, `Fichero_2`, `Fichero_3` y `Fichero_4`. Los tres primeros contienen 500 líneas de `Fichero` y el último, las 100 restantes. Se creará un fichero *oculto* llamado `.Fichero.ctrl` que contendrá (formato texto, en dos líneas separadas): nombre del fichero original y número de ficheros resultantes de la partición.

12. Construir un programa que divida un fichero de *cualquier tipo* en diferentes ficheros, indicando como argumentos el nombre del fichero original y el **máximo número de bytes** que contendrá cada fichero resultante.

    Por ejemplo, si el tamaño de `Fichero` es 1800 bytes, la ejecución de `parte_bytes Fichero 500` genera como resultado los ficheros `Fichero_1`, `Fichero_2`, `Fichero_3` y `Fichero_4`. Los tres primeros contienen 500 bytes de `Fichero` y el último, los 300 restantes. Se creará un fichero *oculto* llamado `.Fichero.ctrl` que contendrá (formato texto, en dos líneas separadas): nombre del fichero original y número de ficheros resultantes de la partición.

13. Construir un programa que reconstruya un fichero a partir de una serie de ficheros que contienen sus "partes". Los ficheros que pueden emplearse como origen se han creado con los programas descritos en los problemas 11 y 12 y por ese motivo se emplará el fichero de control creado por esos programas.

    Por ejemplo, la ejecución de `reconstruye Fichero` genera como resultado `Fichero`. Usará `.Fichero.ctrl` para conocer los ficheros que debe usar y el orden en que se debe hacer la reconstrucción.

14. Escribir un programa similar a `grep` que busque una cadena en una serie de ficheros de texto. La cadena a buscar y los ficheros en los que buscar se proporcionan en la línea de órdenes.
    Por ejemplo:

    ```
    busca Olga fich1 fich2 fich3
    ```

    busca la cadena `Olga` en los ficheros `fich1`, `fich2` y `fich3`.

    Cada vez que encuentre la cadena buscada, debe indicar el fichero en el que es localizada, el número de línea y la línea completa que la contiene. Un ejemplo de salida de este programa es:

```
fich1 (línea 33): Mi amiga Olga ha aprobado MP aunque no se
fich3 (línea 2): ya se lo dije ayer a Olga, pero ni caso
fich3 (línea 242): finalmente, Olga se puso a estudiar
```

Las restricciones que se imponen, y que se deben cumplir en la resolución son:

a) El número de ficheros que se pueden proporcionar es *ilimitado*.

b) Cada uno de los ficheros sólo puede ser leído una única vez, y no pueden copiarse completos en memoria.

c) Se desconoce *a priori* el número de líneas de los ficheros.

d) Las líneas de los ficheros tienen una longitud *indeterminada*, aunque nunca mayor de 500.

15. Implementar un programa que similar a `head` que muestre las primeras líneas de un fichero de texto.

    Por ejemplo, la ejecución de `cabecera 15 reconstruye.cpp` mostrará las primeras 15 líneas del fichero de texto `reconstruye.cpp`

    Se aplican las mismas restricciones que las indicadas en el problema 14(excepto la primera, evidentemente).

16. Implementar un programa que similar a `tail` que muestre las últimas líneas de un fichero de texto.

    Por ejemplo, la ejecución de `final 15 reconstruye.cpp` mostrará las últimas 15 líneas del fichero de texto `reconstruye.cpp`

    Se aplican las mismas restricciones que las indicadas en el problema 14 (excepto la primera, evidentemente).

17. Una empresa de distribución mantiene la información acerca de sus vendedores, productos y ventas en ficheros informáticos. Los ficheros almacenan la información en formato *binario* y la estructura de los *registros* es:

    - Fichero `Vendedores`:
      `RegVendedor`: `CodVendedor` (`unsigned char`), `Nombre` (50*`char`) y `CodZona` (`unsigned char`).
    - Fichero `Articulos`:
      `RegArticulo`: `CodArticulo` (10*`char`), `Descripcion` (30*`char`) y `PVP` (`float`).
    - Fichero `Ventas`:
      `RegVenta`: `NumFactura` (`int`), `CodVendedor` (`unsigned char`), `CodArticulo` (10*`char`) y `Unidades` (`int`).
      Se supone que el fichero `Ventas` contiene las ventas realizadas en un mes.

    Se trata de realizar programas para:

    a) Mostrar el total (número de ventas y cantidad total de ventas) de las ventas realizadas por un vendedor, dado su código (`CodVendedor`).

    b) Pedir una cantidad y crear un fichero (*binario*) llamado `VendedoresVIP` cuyos registros tengan la siguiente estructura:
    `RegVendedorVIP`: `CodVendedor` (`unsigned char`), `CodZona` (`unsigned char`), `TotalVentas` (`float`).
    donde `TotalVentas` es la cantidad total de ventas realizadas por el vendedor.
    El fichero `VendedoresVIP` tendrá únicamente los registros de los vendedores cuyo total de ventas sea superior a la cantidad leída.

18. Algunos ficheros se identifican mediante una "cabecera" especial, como los archivos PGM que tienen en la posición cero los caracteres `P5`. Este tipo de ficheros, y las marcas que los identifican son los que se gestionarán en este problema.

    Un fichero de descripciones es un fichero *binario* que almacena las distintas *marcas* que identifican a tipos de ficheros, así como información acerca de dónde se ubican en los archivos.

    Los registros del fichero `Descripciones` tienen longitud variable y su formato es el siguiente:

    `RegDescripcion`: `PosInicioMarca` (`int`), `LongMarca` (`int`), `Marca` (`LongMarca`*`char`) y `Comentario` (100*`char`)

    Un ejemplo (en forma tabular) de los contenidos de este archivo podría ser:

0	2	P5	Imagen PGM
0	2	P6	Imagen PPM
10	8	DAT CIEN	Datos Científicos

    Donde podemos ver que si un archivo tiene la cadena "DAT CIEN" (8 caracteres) a partir de la posición 10 del archivo, se trata de un archivo de tipo "Datos Científicos".

    Las tareas a realizar son:

    a) Escribir una función (`Insertar`) para añadir descripciones.
    La función recibirá el nombre de un archivo de descripciones junto con una nueva entrada (un dato de tipo `RegDescripcion`) y añadirá esa entrada a dicho archivo (creándolo si es necesario).

    b) Implementar una función (`TipoArchivo`) que determine el tipo de un fichero.
    La función recibirá el nombre del fichero del que queremos averiguar si tipo junto con el nombre del archivo de descripciones. Como resultado devuelve una cadena con el comentario asociado, o la cadena *Tipo desconocido* si no se ha localizado su tipo.

    c) Escribir un programa que, usando la función del apartado anterior, reciba de la línea de órdenes el nombre de un archivo y escriba en la salida estándar la descripción (*comentario*) asociado a su tipo. Tenga en cuenta los posibles casos de error.

19. Hacer un programa que permita formar el nombre de un fichero de la forma: `salidaXXX.Z.dat` a partir de dos números que recibe de la línea de órdenes:

    - `XXX` es un número de 3 dígitos (se rellena con ceros a la izquierda si es necesario), y
    - `Z` es un número con cualquier cantidad de dígitos.

    A continuación se presentan varios ejemplos de ejecución (nuestro programa se llamará `componer`):

    `componer 45 6` generaría el nombre de fichero `salida.045.6.dat`

    `componer 5 67` generaría el nombre de fichero `salida.005.67.dat`

    Nota: usar la clase `stringstream`