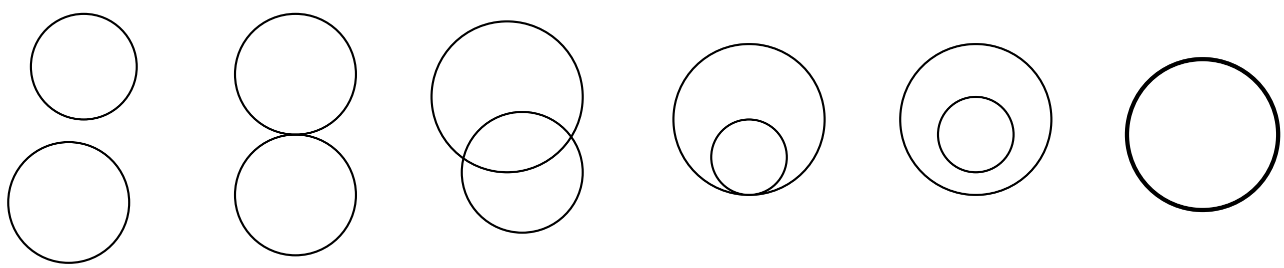
PROGRAMACIÓ CIENTÍFICA. CURS 2019-20. PRIMAVERA.
PRÀCTIQUES DE LABORATORI D'ORDINADORS

LLISTA 5. ESTRUCTURES

1 Es vol fer un programa per a saber la posició relativa de dues circumferències del pla $\mathbb{R}^2$.

Una circumferència està determinada pel seu centre (un punt del pla) i el seu radi (un valor real no negatiu). Per aquest motiu, es treballarà amb un tipus estructurat, de nom `circ`, format per dos camps: un, de nom `cent`, serà un vector de dues components `float`; l'altre, de nom `radi`, serà de tipus `float`.

Feu un programa que llegeixi un vector de dues components de tipus `circ` (o sigui, llegeix dues circumferències), que comprovi que els radis són no negatius, i que faci els càlculs adequats per tal d'escriure alguna de les sis coses següents: exteriors, tangents exteriors, secants, tangents interiors, interiors, coincidents:



2 [operacions aritmètiques amb complexos] Definiu un tipus estructurat, `comp`, amb dos components de tipus `double` anomenades `real` i `imag`. S'hi podrà guardar un nombre complex en coordenades cartesianes.

Feu funcions en C que efectuin les quatre operacions aritmètiques bàsiques. Els prototipus han de ser:

```
comp suma(comp, comp);
comp rest(comp, comp);
comp prod(comp, comp);
comp divi(comp, comp);
```

En el cas de la funció `divi`, se suposa que el segon argument no és l'element $0 \in \mathbb{C}$.

Feu una funció `main` que llegeixi dos nombres complexos, cadascun com un parell de nombres reals, i realitzi amb ells les 4 operacions elementals (o només 3 i un missatge adequat, si el segon nombre complex és 0).

3 [operacions aritmètiques amb fraccions] Es vol fer un programa que faci la suma, la resta, el producte i el quocient de dues fraccions donades.

Qualsevol fracció es guardarà en una variable d'un tipus estructurat especial, de nom `fraccio`, i de 3 camps, de noms `sig`, `num` i `den`, els quals guardaran, respectivament, el signe, el numerador i el denominador. El primer camp serà de tipus `char` (per a contenir `+` o `-`), i els altres dos, de tipus `unsigned int` (per a contenir enters no negatius).

La funció `main` llegirà les dues fraccions amb que es vol operar, anirà cridant les funcions adequades, i anirà escrivint les operacions senceres, usant el format de l'exemple següent. Si les dades són

```
+ 2 3 - 3 4
```

llavors cal escriure

```
( + 2 / 3 ) + ( - 3 / 4 ) = ( - 1 / 12 )
( + 2 / 3 ) - ( - 3 / 4 ) = ( + 17 / 12 )
( + 2 / 3 ) x ( - 3 / 4 ) = ( - 1 / 2 )
( + 2 / 3 ) : ( - 3 / 4 ) = ( - 8 / 9 )
```

Observi's que les fraccions resultat són sempre irreductibles. D'altra banda, si alguna fracció de les dades és incorrecta (el signe és diferent de $+$ o $-$, o el denominador és 0) llavors cal fer escriure un missatge adequat, i si la segona fracció té valor 0 llavors cal fer escriure que no es pot fer el quocient.

Cal programar també les següents funcions, de cadascuna de les quals es dona el prototipus i es diu què ha de fer (s'imposen prototipus variats per a practicar possibilitats diverses):

- `unsigned int mcd(unsigned int, unsigned int);`

  retorna el màxim comú divisor de dos enters no negatius (se sap que el segon no és 0).

- `void reduir(fraccio *);`

  transforma una fracció a forma irreductible.

- `void suma(fraccio, fraccio, fraccio *);`

  suma dues fraccions (i redueix el resultat).

- `fraccio resta(fraccio, fraccio);`

  retorna la resta (en forma reduïda) de dues fraccions.

- `fraccio producte(fraccio, fraccio);`

  retorna el producte (reduït) de dues fraccions.

- `unsigned int quocient(fraccio, fraccio, fraccio *);`

  fa el quocient de dues fraccions (si es pot) i redueix el resultat. Quan es pot, retorna un 0. Quan no es pot (la fracció del segon argument té el valor 0), retorna un 1.

- `void escriure(fraccio, char, fraccio, fraccio);`

  per a escriure una operació sencera, en el format de l'exemple.

4 [*conques d'atracció* del mètode de NR] El *mètode de Newton-Raphson* per a trobar solucions d'una equació $f(z) = 0$, on f és derivable, consisteix a generar successions

$$\begin{array}{l} z_0 \text{ valor inicial,} \\ z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)} \quad \forall i = 0, 1, 2, \dots \; ; \end{array}$$

i comprovar si són convergents.

Pot passar que no es pugui generar la successió perquè algun denominador $f'(z_i)$ sigui 0. També pot passar que es generi una successió i no sigui convergent. Però si es genera una successió i és convergent, llavors, prenent límit als dos costats de la igualtat que defineix la iteració, es veu trivialment que el límit a ha de verificar $f(a) = 0$; per tant, es troba alguna de les arrels de la funció $f(x)$, tal com es volia.

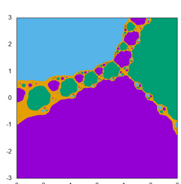
En el programa que es demana es considera una funció polinomial amb 3 arrels diferents, en la variable complexa z; per exemple $f(z) = (z-1)(z-2-i)(z-2i)$, amb arrels conegudes $a_1 = 1$, $a_2 = 2+i$ i $a_3 = 2i$. Es tracta de classificar els possibles valors inicials $z_0 \in \mathbb{C}$ en 5 subconjunts, segons el que passa quan s'aplica el mètode de Newton amb aquest valor inicial:

- $z_0 \in$ `53.res0` si no es pot generar la successió.
- $z_0 \in$ `53.res1` si la successió convergeix a a_1.
- $z_0 \in$ `53.res2` si la successió convergeix a a_2.
- $z_0 \in$ `53.res3` si la successió convergeix a a_3.
- $z_0 \in$ `53.res4` si la successió no convergeix.

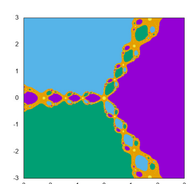
A efectes pràctics, es fixaran 4 paràmetres per a adaptar les definicions anteriors a un programa d'ordinador:

- Un pas (`h=1e-2`) de separació (en cada component: real i imaginària) del valors inicials z_0 al quadrat $[-3,+3] \times [-3,+3] \subset \mathbb{C}$.
- Una tolerància (`tol=1e-2`) per a decidir si un denominador es massa pròxim a 0.
- Una precisió (`prec=1e-10`) per a decidir si algun z_i ja és suficientment pròxim a alguna de les tres arrels.
- Un nombre màxim d'iterats (`iter=12`) que es calcularan per a cada z_0 inicial.

Feu un programa en C que generi aquests 5 conjunts (escrivint els punts en arxiu diferent), i pinteu-los usant `gnuplot`.



(a) $(x-1)(x-2-i)(x-2i)$



(b) $x^3 - 1$

És obligatori treballar amb un tipus estructurat complex definit per vosaltres, així com amb funcions adequades per a fer les operacions que faci falta amb variables i constants d'aquest tipus (exercici anterior).

Nota: Evidentment, es pot canviar l'exemple, o els valors dels paràmetres. Feu proves.

5 [Jacobi per a un sistema lineal amb matriu *esparsa*] Un mètode per a resoldre un sistema lineal $Ax = b$, de dimensió $n \times n$, amb A regular, és el *mètode iteratiu de Jacobi*:

Donada una aproximació inicial $x^{(0)}$ de la solució, es genera una successió d'aproximacions: per a cada $i \geq 0$, es calcula una nova aproximació $x^{(i+1)}$ a partir de l'aproximació actual $x^{(i)}$, mitjançant

$$\forall k = 0 \div n-1 \ : \quad x_k^{(i+1)} = \frac{1}{a_{kk}} \left(b_k - \sum_{\forall j \neq k} a_{kj} x_j^{(i)} \right) \ .$$

Observacions.

- Per a poder usar aquest mètode, cal que tots els elements de la diagonal de A siguin diferents de 0: $a_{kk} \neq 0 \ \ \forall k = 0 \div n-1$.
- El mètode és consistent, en el sentit que, si la successió generada és convergent, llavors el vector límit és solució del sistema lineal inicial.
- A la pràctica, només es pot generar una quantitat finita de vectors i, de fet, només cal usar 2 vectors: siguin x i y. En cada iteració: es calcula y a partir de x usant la fórmula anterior, es calcula després $error = \|y - x\|_2$ i, si aquest valor no és menor que una determinada precisió fixada d'entrada (per exemple `prec`$= 10^{-6}$), llavors es posa x en y i es fa una altra iteració. En canvi, si $error < prec$, es considera que hi ha hagut convergència.
- També es posa un límit a la quantitat d'iteracions que es faran (per exemple `iter`$= 50$). Si no s'assoleix la precisió demanada amb aquestes iteracions (o menys), llavors no es continua fent iteracions i se suposa que el mètode no ha convergit.

En aquest exercici es vol aplicar el mètode anterior al cas que la matriu A és *esparsa* o *escassa*: té molts pocs elements diferents de 0. Quan passa això, es pot decidir no guardar tots els n^2 elements a_{kj} ($\forall k, j = 0 \div n-1$), sinó només les ternes (k, j, a_{kj}) tal que $a_{kj} \neq 0$. Si n és gran llavors s'estalvia molta memòria. En contrapartida, aquesta manera de guardar la informació de A fa que el mètode de Jacobi sigui més complicat de programar.

Feu un programa que implementi el mètode iteratiu de Jacobi per a matrius escasses. Heu d'usar el tipus estructurat

```
typedef struct {
   int fil;
   int col;
   double val;
} ele;
```

i heu de declarar la matriu com `ele *A;`. S'haurà de reservar memòria només per als elements no nuls de A.

Llegiu les dades d'un fitxer, on hi haurà: la dimensió n, el nombre qu d'elements de A no nuls, a continuació les qu ternes corresponents, i finalment les n components del vector b.

Exemple: (*R.L. Burden, J.D. Faires: Numerical Analysis*, sec.12.1, ex.1)

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 25 \\ 50 \\ 150 \\ 0 \\ 0 \\ 50 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

que té per solució $x = (18.75, 37.50, 56.25, 12.50, 25.00, 37.50, 6.25, 12.50, 18.75)$

Anàlisi de l'ús de memòria. L'exemple anterior és un cas simplificat (dimensió $n = 9 = 3^2$) d'un tipus de sistemes lineals amb matriu esparsa que apareixen en molts camps de la ciència. En la realitat, interessa $n = N^2$ amb N més gran.

Suposem, doncs, que n és molt gran (per exemple $100^2 = 10000$), i suposem que cada fila de A només té, com a màxim, 5 elements no nuls (com passa a l'exemple). Quin és l'estalvi de memòria si no s'usa una matriu quadrada sencera?

6 [operacions habituals en una llista enllaçada] A la classe de teoria s'han vist alguns exemples d'operacions en una llista (simplement) enllaçada, els nodes de la qual són del tipus

```
struct elem {
   int clau;
   float info;
   struct elem *seg;
};
```

El camp `clau` ha de ser diferent per a cada node, i això permet identificar-lo.

Feu un programa interactiu que presenti un menú (numèric, per ex.) que permeti la creació d'una llista i realitzar les operacions, afegint-hi també lesoperacions que es demanen en aquest exercici. Heu de fer una funció diferent per a cada operació. Per a cadascuna, heu de preveure totes les possibilitats.

- Inserció d'un node nou al final de la llista.
- Inserció d'un node nou al davant d'un determinat node de la llista (localitzat pel seu camp `clau`).
- Eliminar el primer element de la llista.
- Eliminar l'últim element de la llista.
- Eliminar el node anterior a un determinat node de la llista (localitzat pel seu camp `clau`).