Problemes capítol 7

Realitzar cada exercici donant el pseudocodi i/o el diagrama de flux i el codi C.

Considerar, en cada cas, els procediments/funcions que simplifiquen la solució del problema i la fan més intel·ligible.

7.1 Donar la distància entre i el punt mig que hi ha entre dos punts a=(ax,ay) i b=(bx,by). Considerar l'entitat *punt* com a una estructura.

7.2 En l'exercici 6.2 es va realitzar un exercici en el que es demanava trobar la relació de canvi que s'establia amb 3 plats i 7 pinyons. Realitzar ara el mateix exercici però creant l'estructura que contingui plat, pinyó i raó.

7.3 Realitzar un programa que converteixi coordenades polars a rectangulars i viceversa.

7.4 Tenim un vector d'estructura

```
Estructura vector
    Enter v[100]
    Enter n
FiEstructura
```

on n indica el número d'elements que v[] conté realment (en té reservats fins a 100).

Escriviu una funció que elimini els valors duplicats d'un vector d'aquest tipus on els elements estan ordenats.

Per a realitzar un exemple complert, es recomana seguir el següent procediment (això és una mostra del que hi pot haver en el programa principal):

```
vector ve;                  //Declaració del vector
ve = entrar(ve);            //Generar el vector de n nombres aleatoris
imprimir(ve, "aleatori");   //Escriptura del vector generat de nombres aleatoris
ve = ordenar(ve);           //Ordenació del vector: Aplicar procediment de la bombolla
imprimir(ve, "ordenat");    //Escriptura del vector ordenat
ve = unificar(ve);          //Eliminar els elements duplicats del vector
imprimir(ve, "unificat");   //Escriptura del vector sense elements repetits
```

7.5 Crear l'estructura complex adequada per a treballar amb nombres complexes.

Amb l'estructura crear l'algorisme que permeti sumar, restar, multiplicar i dividir nombres complexes.

7.6 En una escola es vol informatitzar el tractament de les qualificacions dels estudiants d'un curs que són avaluats quatre cops a l'any de les assignatures de **càlcul, àlgebra, informàtica** i **programació.**

De cada avaluació es guarda la qualificació numèrica (de 0 a 10) i la literal (NP,S,A,N,E)

Els registres on s'emmagatzemaran les dades, a més de les qualificacions, hauran de contenir el nom de l'estudiant y el grup a què pertanyen (al curs hi ha **200** estudiants repartits en quatre grups: **I**, **II**, **III** i **IV**).

Dissenyeu en **C** una estructura de dades vàlida per a tractar eficientment la gestió de notes.

7.6 Sigui una base de dades de n punts tridimensionals (x,y,z). Es demana:

i) Crea l'estructura de dades (de nom *punt*) necessària per a crear aquesta base de dades (en C).

ii) Treballant amb aquesta estructura, fes l'algorisme que troba si els n punts estan en línia recta.

Per a l'algorisme, crea les funcions entrar_array( ), i calcul_linia_recta( ) i treballa amb variables locals.

Exercici 7.1

Pseudocodi:

Entrades: coordenades (x, y) de dos punts

Sortides: distància i punt mig entre els dos punts

```
Estructura punt
        Real x, y
FiEstructura

Real dist               //Distància entre els dos punts
Punt a, b, pm, d        //Punts a i b, punt mig i separació entre coordenades

Inici
      Llegir (a, b)
      d.x = a.x - b.x
      d.y = a.y - b.y
      dist = arrel_quadrada(d.x^2 + d.y^2)
      pm.x = (a.x + b.x)/2
      pm.y = (a.y + b.y)/2
      Escriure  (d, pm)
FiPer
Fi
```

Codificació en C

```
/*
Donades dos punts a=(ax,ay) i b=(bx,by) trobar la seva
distància i el seu punt mig
*/
#include <stdio.h>
#include <math.h>

struct punt
{
        float x, y;
};

void main()
{
        punt a, b, pm, d;                                   //punts a, b, punt mig i distància
        float dist;
        printf("Dona coordenada a (a1, a0): ");             //Llegir punts a, b
        scanf("%f%f", &a.x, &a.y);
        printf("Dona coordenada b (b1, b0): ");
        scanf("%f%f", &b.x, &b.y);
        d.x = a.x-b.x;                                      //Càlcul distància
        d.y = a.y-b.y;
        dist = sqrt(d.x*d.x + d.y*d.y);
        pm.x = (a.x+b.x)/2;                                 //Càlcul punt mig
        pm.y = (a.y+b.y)/2;
        printf("\tDistancia = %3.2f\n", dist);              //Imprimir resultats
        printf("\tPunt mig = (%3.1f, %3.1f)\n", pm.x, pm.y);
}
```

Exercici 7.2

Entrades: Array de 3 plats i array de 7 pinyons

Sortides: Impressió de tots els canvis possibles i les distàncies recorregudes

L'esquema principal de l'exercici és:

```
Estructura relació
        Enter plat, pinyo
        Real relació, distancia
FiEstructura

Enter pl[3] = {24, 32, 42}, pi[7]={28, 24, 21, 19, 17, 15, 13} //Relació de plats, relació de pinyons
Const Real PI=3.141592, R=0.34              //Constants PI, radi de la roda (m)
Real canvi, l                               //Variable relació de canvi, distància recorreguda

Inici
      Càlculs
      ImpressióRelacionsCanvi
      ImpressióDistànciaRecorreguda PerPedalada
Fi
```

Com es veu, el programa s'ha dividit en tres mòduls dels que, cadascun, només ha de realitzar el conseqüent recorregut per tots els plats i tots els pinyons (doble bucle).

Codificació en C

```
/* Donar la relacio de canvi d'una bicicleta. La bicicleta te 3 plats (24, 32 i 42 dents) i 7 pinyons
(28, 24, 21, 19, 17, 15, 13 dents).
Crear l'estructura que permeti ordenar la combinació plat-pinyó d'acord amb la relació de canvi.
Finalment, tenint en compta que la roda te un radi=68cm, donar l'avanç per pedalada.
*/
#include <stdio.h>
#define PLAT 3
#define PINYO 7
#define RADI .34                                    //Radi de la roda
#define PI 3.141593

struct relacio
{
        int pla, pin;
        float rao, distancia;
};

void relacions (relacio[PLAT][PINYO], int[PLAT], int[PINYO]);
void imprimirCanvis(relacio[PLAT][PINYO]);
void imprimirDistancies(relacio[PLAT][PINYO]);

void main()
{
        int pl[PLAT]={24, 32, 42};
        int pi[PINYO]={28, 24, 21, 19, 17, 15, 13};
        relacio canvi[PLAT][PINYO];

        relacions(canvi, pl, pi);
        imprimirCanvis(canvi);
        imprimirDistancies(canvi);
}
```

```
void relacions (relacio canvi[PLAT][PINYO], int pl[PLAT], int pi[PINYO])   //Càlculs
{
	int i, j;
	for (i=0; i<PLAT; i++)
	{
		for(j=0; j<PINYO; j++)
		{
			canvi[i][j].pla = i;
			canvi[i][j].pin = j;
			canvi[i][j].rao = pl[i]*1.0/pi[j];
			canvi[i][j].distancia = canvi[i][j].rao * 2 * PI * RADI;
		}
	}
}

void imprimirCanvis(relacio canvi[PLAT][PINYO])                    //Impressió relació de canvi
{
	int i, j;
	printf("Relacio de canvi\n");
	for (i=0; i<PLAT; i++)
	{
		for (j=0; j<PINYO; j++)
		printf("(%i,%i)=%3.2f ",canvi[i][j].pla+1, canvi[i][j].pin+1, canvi[i][j].rao);
		printf("\n");
	}
	printf("\n\n");
}

void imprimirDistancies(relacio canvi[PLAT][PINYO]) //Impressió distància recorreguda/pedalada
{
	int i, j;
	printf("Metres per pedalada\n");
	for (i=0; i<PLAT; i++)
	{
		for (j=0; j<PINYO; j++)
		printf("(%i,%i)=%3.2f ",canvi[i][j].pla+1, canvi[i][j].pin+1, canvi[i][j].distancia);
		printf("\n");
	}
	printf("\n\n");
}
```

Exercici 7.3

Les equacions de conversió polars-rectangulars venen donades per:

x=r cos(a)	r=sqrt(x^2+y^2)
y=r sin(a)	a=arctg(y/x)

En el programa, es treballa amb les dues estructures:

```
Estructura pol          //Estructura polars
{
        float r, a;     //radi, angle
}
```

```
Estructura rect         //Estructura polars
{
        float x, y;     //radi, angle
}
```

L'algorisme, per tant, no té cap complicació. Tant sols s'ha de tractar l'angle en la conversió de rectangulars a polars ja que, per exemple, arctan(x/y) = arctan(-x/-y)!

Pseudocodi del programa principal:

Entrades: Número en polars o en rectangulars

Sortides: El número en rectangulars o polars

```
Caracter a
pol po
rect re
Inici
        Fer
                Llegir (car)
                EnCasDe(car)
                        'r':    Llegir(po.r, po.a)                      //A rectangulars
                                re.x = po.r · cos(PI*po.a/180)
                                re.y = po.r · sin(PI*po.a/180)
                                Escriure(re.x, re.y)
                                FiCas_r
                        'p':    Llegir(re.x, re.y)                      //A polars
                                po.r = arrel_quadrada(re.x^2 + re.y^2)
                                po.a = 180 · arctan(re.y/re.x) /PI
                                po = tractarAngle(po, re)
                                Escriure(po.r, po.a)
                FiEnCasDe
        Mentre (c<>'s')
Fi
```

Codificació:

```
/*
Conversio polars-rectangulars
*/
#include <stdio.h>
#include <math.h>
#define PI 3.14159

struct rect                             //Estructura coordenades rectangulars
{
        float x, y;
};
struct pol                              //Estructura coordenades rectangulars
{
        float r, a;
};
pol tractar_a(pol, rect);               //Declaració funció tractar angle.
```

```
//Programa principal
void main()
{
	char c;
	rect re;
	pol po;
	printf("CONVERSIO RECTANGULARS-POLARS (en graus)\n\n");
	do
	{
		printf("\t--> Conversio a (r)ectangulars o a (p)olars, o (s)ortir? ");
		fflush(stdin);
		scanf("%c", &c);
		switch(c)
		{
		case 'r': printf("\t--> Entra r i a: ");
			scanf("%f%f", &po.r, &po.a);
			re.x = po.r*cos(PI*po.a/180);
			re.y = po.r*sin(PI*po.a/180);
			printf("\n\t--> (x, y) = (%5.2f, %5.2f)\n", re.x, re.y);
			break;
		case 'p': printf("\t--> Entra x i y: ");
			scanf("%f%f", &re.x, &re.y);
			po.r = sqrt(re.x*re.x + re.y*re.y);
			po.a = 180*atan(re.y/re.x)/PI;
			po = tractarAngle (po, re);
			printf("\n\t--> (r, a) = (%5.2f, %5.2f)\n", po.r, po.a);
		}
	}while (c!='s');
}

//Funció tractar angle
pol tractarAngle (pol po, rect re)			//Tractament x, y negatius
{
	if (po.a >0)
	{
		if (re.x<0) po.a += 180.0;		//cas x i y negatius
	}
	else
	{
		if (re.x<0) po.a += 180.0;		//cas x negatiu
		else po.a += 360.0;			//cas y negatiu
	}
	return(po);
}
```

Exercici 7.4

Exemple complert de treball.

L'enunciat clarifica els passos que s'han de seguir en el programa principal.

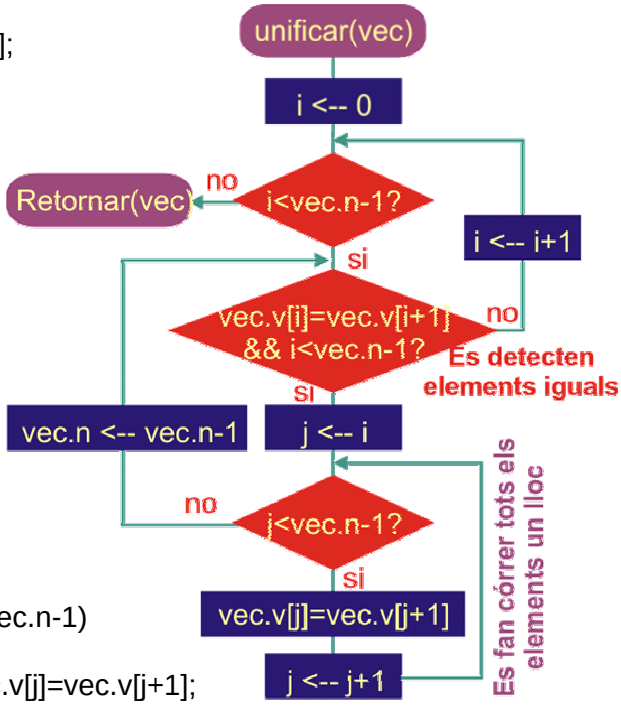
Només cal tenir present, sempre, el nombre d'elements que hi ha en l'array, comptabilitzat per la variable n de l'estructura.

Tots els procediments/funcions són senzills. Tant l'entrada dels nombres aleatoris com l'algorisme de la bombolla s'han explicat en el capítol 6. La impressió de resultats és recórrer un array. I la funció unificar (eliminar elements repetits del vector), s'ha fet d'acord amb el diagrama de flux que s'hi adjunta.

Codificació en C

```
/*
Estructura amb vector de nombres aleatoris no repetits
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 100                   //Número d'elements del vector
#define NMAX 25                 //Número aleatori (enter) major
#define NCAR 10                 //Número de caràcters que es passen

struct vector
{
        int v[100];
        int n;
};

vector entrar(vector);          //Declaració procediments/funcions
vector ordenar(vector);
vector unificar(vector);
void imprimir(vector, char[]);

//Funció principal
void main()
{
        vector ve;                      //Declaració del vector
        ve = entrar(ve);                //Entrar vector aleatori
        imprimir(ve, "aleatori");       //Escriptura vector aleatori
        ve = ordenar(ve);               //Ordenació del vector
        imprimir(ve, "ordenat");        //Escriptura vector ordenat
        ve = unificar(ve);              //Treure elements duplicats
        imprimir(ve, "unificat");       //Escriptura vector unificat
}

//Entrada del vector de nombres aleatoris
vector entrar(vector vec)
{
        printf("--> Quants numeros vols? ");
        scanf("%i", &vec.n);
        srand(time(NULL));                                      //Generador llavor
        for (int i=0;i<vec.n;i++)
                vec.v[i]=int(NMAX*rand()/RAND_MAX);             //Generació números aleatoris
        return (vec);
}
```

```
//Ordenació del vector: mètode de la bombolla
vector ordenar(vector vec)
{
	int i, j, temp, canvi=1;
	j=vec.n-1;
	while (canvi && j>0)
	{
		i=0;
		canvi = 0;
		while (i<j)
		{
			if (vec.v[i]>vec.v[i+1])
			{
				temp = vec.v[i];
				vec.v[i] = vec.v[i+1];
				vec.v[i+1] = temp;
				canvi = 1;
			}
			i++;
		}
		j--;
	}
	return (vec);
}
```




```
//Eliminar elements duplicats
vector unificar(vector vec)
{
	int i=0, j;
	while (i < vec.n-1)
	{
		while (vec.v[i]==vec.v[i+1] && i<vec.n-1)
		{
			for (j=i; j<vec.n-1; j++) vec.v[j]=vec.v[j+1];
			vec.n--;
		}
		i++;
	}
	return (vec);
}

//Imprimir vector
void imprimir(vector vec, char s[NCAR])
{
	printf("\nElements de vector (%s): ", s);
	for (int i=0;i<vec.n;i++) printf("%i ", vec.v[i]);
	printf("\n\n");
}
```

Exercici 7.5

Els nombres complexes formen una estructura ideal per a treballar amb estructures, tal com es veu amb aquest exemple que no necessita de gaires comentaris.

Aquí només es realitzen quatre funcions de les moltes que admet el problema.

La codificació, en aquest exemple, és immediata.

Codificació.

```
/*
Aritmètica amb nombres flotants
*/
#include <stdio.h>

typedef struct
{
	float x;
	float y;
}complex;

void entrar_nums(complex *, complex *);
complex sumar(complex, complex);
complex restar(complex, complex);
complex producte(complex, complex);
complex dividir(complex, complex);
void imprimir(complex, char);

void main()
{
	char cas;
	complex num1, num2, res;
	printf("ARITMETICA AMB NOMBRES COMPLEXES\n");
	printf("Que vols fer (+, -, *, /)? ");
	fflush(stdin);
	scanf("%c", &cas);
	while (cas != '\n')
	{
		entrar_nums(&num1, &num2);
		switch (cas)
		{
		case '+': res = sumar(num1, num2);
			imprimir(res, cas);
			break;
		case '-': res = restar(num1, num2);
			imprimir(res, cas);
			break;
		case '*': res = producte(num1, num2);
			imprimir(res, cas);
			break;
		case '/':if (num2.x == 0 && num2.y == 0)
					printf("		Aixo es infinit!\n");
			else
			{
				res = dividir(num1, num2);
				imprimir(res, cas);
			}
			break;
		}
```

```
                printf("Que vols fer (+, -, *, /)? ");
                fflush(stdin);
                scanf("%c", &cas);
        }
}

void entrar_nums(complex * a, complex * b)
{
        printf(" Entra el numero A (part_real, part_imaginaria): ");
        scanf("%f%f", &a->x, &a->y);
        printf(" Entra el numero B (part_real, part_imaginaria): ");
        scanf("%f%f", &b->x, &b->y);
}
complex sumar(complex a, complex b)
{
        complex c;
        c.x = a.x + b.x;
        c.y = a.y + b.y;
        return(c);
}
complex restar(complex a, complex b)
{
        complex c;
        c.x = a.x - b.x;
        c.y = a.y - b.y;
        return(c);
}
complex producte(complex a, complex b)
{
        complex c;
        c.x = a.x*b.x - a.y*b.y;
        c.y = a.x*b.y + a.y*b.x;
        return(c);
}
complex dividir(complex a, complex b)
{
        complex c;
        float cc;
        cc = b.x*b.x + b.y*b.y;
        c.x = (a.x*b.x + a.y*b.y)/cc;
        c.y = (a.y*b.x - a.x*b.y)/cc;
        return(c);
}
void imprimir(complex c, char car)
{
                printf(" --> A %c B = %3.2f ", car, c.x);
                if (c.y < 0) printf("%3.2f*i\n", c.y);
                else printf("+%3.2f*i\n", c.y);
}
```

Exercici 7.7

Exercici simple dividit en dos mòduls.
La comprovació de si els punts estan en línia recta està comprovant que la raó entre cada dos punts en els tres eixos és constant.

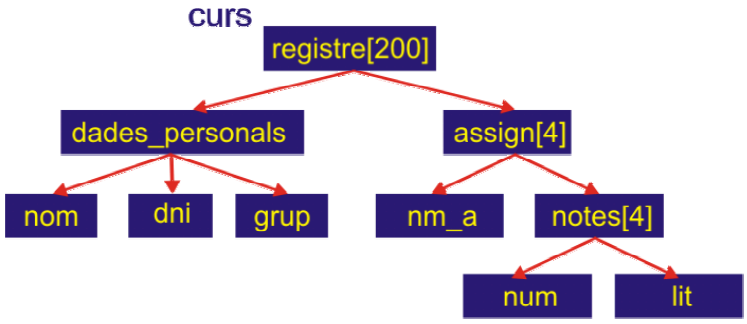
Codificació.

```
/*
Treballant amb punts tridimensionals
*/

#include <stdio.h>
#define N 100                                   //Número màxim de punts permesos
#define ER 0.01                                 //Error d'alinealitat permès
struct punt                                     //Estructura punt tridimensional
{
        float x, y, z;
};

int entrar_array (punt []);
void calcul_linia_recta (punt [], int);

void main()
{
        int n;
        punt ar[N];
        printf("TREBALLANT AMB PUNTS TRIDIMENSIONALS\n");
        n= entrar_array(ar);                    //Entrar els punts
        calcul_linia_recta(ar, n);              //Fer el càlcul
}

/*
Entrar els punts en l'array
*/
int entrar_array (punt p[N])
{
        int n;
        printf("Quants punts vols entrar? ");
        scanf("%i", &n);
        for (int i=0; i<n; i++)
        {
                printf("Entra coordenades del punt %i: ", i);
                scanf("%f%f%f", &p[i].x, &p[i].y, &p[i].z);
        }
        return(n);
}

/*
Calcular si els punts estan en línia recta
*/
void calcul_linia_recta (punt p[N], int n)
{
        int linia_recta=1, i;
        punt d, r;
        if (n<2) printf("...amb menys de dos punts no es fa res!\n");
        else
```

```
	{
		d.x=p[1].x-p[0].x;
		d.y=p[1].y-p[0].y;
		d.z=p[1].z-p[0].z;
		i=1;
		while (linia_recta && ++i<n)
		{
			r.x=(p[i].x-p[0].x)/d.x;			//Càlcul distància punts
			r.y=(p[i].y-p[0].y)/d.y;
			r.z=(p[i].z-p[0].z)/d.z;
			r.x = (r.x>r.y) ? (r.x-r.y):(r.y-r.x);		//Càlcul errors
			r.y = (r.y>r.z) ? (r.y-r.z):(r.z-r.y);
			if (r.x>ER || r.y>ER) linia_recta=0;
		}
		if (linia_recta) printf(" Punts en linia recta!!\n");
		else printf(" Punts dispars!!\n");
	}
}
```

Exercici 7.7

Un mètode fàcil de creació de l'estructura és representar-la en forma d'arbre:




```
struct dades_personals
{
        char nom[30];
        long dni;
        int grup;
};
struct notes
{
        float num;
        char lit[2];
};
struct assign
{
        char nom[15];
        float notes[4];
};
struct registre
{
        dades_personals dp;
        assign ass[4];
};

void main()
{
        registre curs[200];
        ...
```