Objectius de la pràctica:
- Saber declarar i utilitzar el tipus de dades estructurat array.
- Saber emprar el tipus string i les funcions definides sobre ell.
- Saber declarar i usar el tipus de dades estructurat struct (registre).

Dades estructurades

Les dades de tipus estructurat o tipus compost són agrupacions d'altres tipus de dades. Segons com s'emmagatzemen les dades en la memòria, hom pot fer una primera classificació de les dades de tipus estructurat en:

- Estructures contigües, les quals s'emmagatzemen en memòria de forma consecutiva.
- Estructures enllaçades, les quals s'emmagatzemen de forma saltejada.

Atenent a si la seua grandària roman fixa al llarg de l'execució del programa, hom pot fer una segona classificació en:

- Estructures estàtiques, la grandària de les quals no canvia al llarg de l'execució del programa.
- Estructures dinàmiques, la grandària de les quals pot canviar durant l'execució del programa.

Normalment, les estructures enllaçades són també dinàmiques.

Vectors i Matrius

Vectors

Serveixen per a agrupar variables d'un mateix tipus sota un nom únic. Aquestes variables s'emmagatzemen en memoria de forma consecutiva, de manera que cadascuna d'elles constituisca un element del vector.

Les operacions permeses sobre els elements del vector (com ara l'assignació, la suma, etc.) són les mateixes que es permeten per a les dades del tipus base.

Per a identificar al vector, com amb la resta de variables, li assignarem un nom. Per a identificar a cadascun dels elements afegirem, darrere del nom, un o diversos índexs entre claudàtors. El nombre d'índexs defineix el nombre de dimensions del vector.

Com que són estructures estàtiques, la grandària i la dimensió del vector queden fixades en la seua declaració i no es poden modificar durant l'execució.

Sintaxi: `tipus nom [grandària];`

On:

`tipus` és el tipus base del vector, açò és, el tipus de dades de cadascun dels elements.
`nom` és l'identificador assignat al vector.
`grandària` és el nombre d'elements del vector. Aquesta grandària es fixa en la declaració per a que el compilador reserve prou espai en memòria per a tots ells. Aquesta grandària no es pot modificar durant l'execució del programa.

Quan declarem un vector estem declarant un conjunt de variables del tipus base del vector, les quals formen els elements del vector.

El rang dels índexs d'un vector comença sempre en zero i acaba en un valor menys de la grandària del vector. Així, doncs, si la grandària d'un vector és N, el primer element és aquell amb índex 0 i el darrer aquell que té l'índex N – 1.

Cadascun dels elements del vector és una variable independent sobre la qual es poden fer les mateixes operacions que sobre una variable senzilla del mateix tipus que el tipus base del vector. La peculiaritat de les variables que representen els elements del vector és que s'emmagatzemen en posicions de memòria adjacents i que totes s'anomenen igual, essent l'única diferencia l'índex que cal emprar per a accedir-les.

El nom del vector és també una variable que conté l'adreça de memòria on comença el vector (la referencia a la variable).

Per exemple, a l'hora de declarar 10 variables de tipus enter, fins ara declaràvem 10 variables independents:

```
int a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
```

Si les volguérem inicialitzar a 0, caldria escriure 10 assignacions. Fent servir una estructura de dades de tipus vector, les podríem declarar de la manera següent:

```
int a[10];
```

IMPORTANT: No es poden fer assignacions directes entre vectors. Açó és, la següent assignació no s'ha de fer MAI:

```
int a[10], b[10];
a = b;  // Açò està malament!!
```

Declaració de vectors amb iniciació

Com succeia amb la resta de tipus senzills, els vectors (i en general tots els arrays) es poden iniciar amb valors al mateix temps que es declaren. Per tal de fer-ho, cal envoltar amb claus el conjunt de valors separats per comes com mostra la sintaxi següent:

Sintaxi:	`tipus nom [grandària] = {valor1, valor2, ...};`
Exemple:	`float b[4] = {1.3, -3.78, 1.3e10, 0.0025};`

Quan s'inicia a la mateixa vegada que es declara un vector no cal especificar la grandària. En aquest cas, el compilador compta el nombre d'elements i reserva en memòria l'espai just per a tot ells.

Matrius

Anomenem matrius als arrays de dos dimensions, açò és, als arrays els elements dels quals vénen definits per dos índexs. Podem imaginar la seua organització com una taula o matriu, on cadascun dels elements té assignat dos índexs: el primer representa la fila I el segon la columna.

Sintaxi:	`tipus nomMatriu [files][columnes];`

On: tipus és el tipus base dels elements de la matriu.
nomMatriu és l'identificador de la matriu.

files	és el nombre de files de la matriu.
columnes	és el nombre de columnes de la matriu.

Tant el nombre de files com el de columnes queda fixat en la declaració i no es pot modificar en temps d'execució.

El rang de les files i el de les columens comença en zero i arriba, respectivament, a un valor menys de la grandària de les files i les columnes.

Tots els elements d'una matriu s'emmagatzemen en posicions de memòria consecutives, on cadascuna de les files apareix a continuació de l'altra. Dins d'una mateixa fila, tots els seus elements s'emmagatzemen també consecutivament.

Observeu el següent exemple de visualització dels elements d'una matriu d'enters de grandària TAM_X per TAM_Y:

```
int i, j;
int matriu[TAM_X][TAM_Y];

for (i = 0; i < TAM_X; i++)
    for (j = 0; j < TAM_Y; j++)
        cout << matriu[i][j];
```

Declaració de matrius amb iniciació

Com succeeix amb els vectors, les matrius poden inicialitzar-se en el moment de la declaració si col·loquem totes les files entre claus i separades per comes (on dins de cada fila hi ha, a la seua vegada, els seus elements entre claus i separats per comes):

```
float g[3][2] = { {-1.2, 2.3}, {-7, 8.23e13}, {89, -45.78}};
```

Arrays multidimensionals

Hom pot declarar arrays de més dimensions on cadascun dels elements té assignats diversos índexs. Com sempre, cada índex començarà amb el valor 0 i, per tant, acabarà en un índex equivalent a la dimensió menys 1.

Sintaxi: `tipus nom [tam1][tam2]...[tamN];`

Pas dels arrays com a paràmetres de les funcions

Per motius d'eficiència i de comoditat, quan es passa un vector o un array a una funció en C/C++ generalment sols es passa l'adreça de començament del vector i no una còpia del vector sencer. Açò és, el pas d'un vector com a paràmetre d'una funció és sempre un pas per referència.Com tot pas per referència, els vectors seran paràmetres d'entrada però al mateix temps seran paràmetres d'eixida.

En la crida a les funcions, passarem només el nom del vector com a argument. Encara que el pas de vector es faça per referència, no s'empra l'operador "&". Al contrari, aparentment, la declaració dels paràmetres es fa com si es tractara d'un pas per valor.

Un últim aspecte a tindre en compte és que una funció no pot retornar un tipus array, sinó que només el podrà rebre com a argument i, potser, modificar-lo. Per exemple:

```
void llegirVector (int vect[TAM])
{
    int i;
    for (i = 0; i < TAM; i++)
        cin >> vect[i];
}
```

Cadenes de caràcters (strings)

En C/C++, la manera de declarar variables que puguen contenir una seqüència de caràcter és mitjançant el tipus de dades string.

Sintaxi: `string nomVariable;`

Com succeïa amb els tipus de dades simples, les variables de tipus string es poden inicialitzar al mateix moment que es declaren. Per exemple:

```
string s;
string s2 = “Hola”;
```

Funcions per a manipular strings

Les cadenes de caràcters (string) permeten la manipulació de textos. A sovint, es considera que un string no és més que un array de caràters. Per aquest motiu, se sol relacionar el concepte de string amb el d'array. En C++ existeix el tipus (o classe, segons la nomenclatura dels llenguatges orientats a objectes) string. Per a treballar amb ell cal emprar la llibreria

```
#include <string>
```

A continuació mostrarem un conjunt d'operacions bàsiques definides sobre el tipus string:

- Creació de variables:
  ```
  string paraula, frase;
  ```
- Assignació: (Al contrari del que succeia amb els vectors, sí es poden fer assignacions entre strings)
  ```
  frase = paraula;
  frase = “hola”;
  ```
- Accés als caràcters (com en els arrays):
  ```
  paraula[0]
  ```
- Comparació lexicogràfica (==, !=, <, >):
  ```
  frase == paraula
  frase > paraula
  ```
- Lectura/escriptura:
  ```
  cin >> paraula;
  getline(cin, frase); //Llig fins a trobar el final de línia
  cout << frase << endl;
  ```

- Manipulació de textos: (suposem que existeix la variable unsigned int i;):

```
// nº de caràcters de paraula
i = paraula.length();

// Insereix paraula en la posició 3 de la frase
frase.insert(3, paraula);

// Uneix paraula amb “hola” i ho emmagatzema en frase
frase = paraula + “hola”;

// Afegeix paraula al final de la frase
frase += paraula;

// Esborra 7 caràcters de la frase des de la posició 3
frase.erase(3, 7);

// Substitueix 6 caràcters de frase (començant des de la
// posició 1), per paraula
frase.replace(1, 6, paraula);

// Cerca paraula com a subcadena dins de frase. Retorna la
// posició on la troba (o bé -1 si no hi és)
i = frase.find(paraula);
i = frase.find(paraula, posició);  // Posició des d'on buscar

// Retorna la subcadena formada per 3 caràcters des de la
// posició 5 de frase
paraula = frase.substr(5, 3);
```

Pas de strings com a paràmetres a les funcions

Com en el cas dels tipus de dades simples, es farà per valor o per referència, segons si volem deixar que la funció modifique la variable o no.

Registres (estructures)

Fins ara només hem treballat amb estructures de dades homogènies (arrays i strings), en les quals tots els elements eren del mateix tipus, però C/C++ també permet la definició d'estructures de dades heterogènies formades per un conjunt de dades de tipus diferents.

Els registres, també anomenats estructures, són un conjunt de dades de tipus diferents a què podem fer referència sota un sol nom i que s'emmagatzemen de forma consecutiva en la memòria. Cadascun dels elements que formen un registre rep el nom de camp i, com en el cas dels elements d'un array, es poden utilitzar de forma individual. Així, doncs, entendrem els registres o estructures com una agrupació de variables diferents que estan relacionades d'alguna forma lògica.

Abans d'usar una estructura cal definir-la

Una de les característiques de les estructures és que cal definir-les abans d'usar-les en la declaració de variables.

Sintaxi:

```
struct nomEstructura
{
    tipus1 nomVariable1;
    tipus2 nomVariable2;
    ...
    tipusN nomVariableN;
};
```

Observeu com la definició d'una estructura acaba amb punt i coma, ja que la pròpia definició és una sentència.

En la definició d'una estructura no es declara ni es reserva memoria per a cap variable, tan sols es crea un nou tipus de dades que està adaptat a les necessitats del programa. La declaració de les variables que siguen d'aquest nou tipus es farà després de la definició.

Habitualment, una estructura es crea (defineix) perquè puga ser utilitzada dins de diverses funcions, raó per la qual la definició de les estructures es fa, generalment, al principi del programa, fora de qualsevol funció i abans d'haver sigut utilitzada per cap d'aquestes. No obstant això, cap cosa impedeix definir una estructura dins d'una funció. En aquest darrer cas, com és obvi, l'àmbit de l'estructura es limitarà exclusivament a la funció.

A continuació es mostra un exemple de declaració d'estructura:

```
// Informació d'interès sobre un treballador d'una empresa
struct treballador
{
    string nom;
    long int sou;
    string numTelefon;
};
```

Sintaxi de la declaració de variables

Quan l'estructura ja ha estat definida, hom pot declarar variables del nou tipus de la manera següent:

Sintaxi: `struct nomEstructura nomVar1, ..., nomVarK;`

Utilització dels elements individuals (camps) d'una estructura

Quan una variable ha estat declarada de tipus estructura, hom pot accedir a cadascun dels camps especificant el nom de la variable i l'identificador del camp separats ambdós per un punt. Per exemple:

```
cout << "Nom " << pep.nom << endl;
joan.sou = 2000;
```

Com mostra el codi anterior, podem tractar cadascun dels camps d'una estructura com variables diferents. Açò és, podem assignar valors a un dels camps sense tocar la resta de camps (p. ex. `joan.sou = 2000;`) o podem accedir a un dels camps sense necessitat d'accedir a la resta dels valors de l'estructura (p. ex. `cout << "Nom " << pep.nom << endl;`).

Al contrari dels arrays, sí està permesa l'operació d'assignació entre estructures, amb la qual cosa es poden fer còpies de dues variables del mateix tipus d'estructura sense haver de fer-ho camp a camp.

Iniciació de variables de tipus estructurades

Com per a la resta de tipus, les variables de tipus estructura es poden inicialitzar al mateix moment que es declaren. La manera de fer-ho és similar a la dels vectors, açò és, els valors inicials de l'estructura s'especifiquen entre claus i separats per comes, tot seguint el mateix ordre en què es definiren els camps.

Estructures dins d'estructures

Una estructura pot contenir altres estructures definides prèviament. Llavors, quan es mencionen els elements de l'estructura s'indicarà, ordenadament, el nom de la variable de l'estructura principal, el nom del camp que és de l'estructura imbricada i el nom de l'element dins d'aquesta subestructura, tot separat per punts.

Exemple d'estructura amb camps de tipus estructura:

```
struct departament
{
    string nom;
    int nombreTreballadors;
    struct treballador cap;
};
```

Arrays dins d'estructures

Les cadenes no són els únics tipus d'arrays que poden formar part d'una estructura. En general, qualsevol array pot ser un camp d'una estructura.

Arrays d'estructures

Com en el cas de les variables simples, hom pot definir vector, matrius o arrays multidimensionals que tinguen elements de tipus estructura. En aquest cas, serà necessari que l'estructura estiga definida abans de declarar l'array.

Exemple:
```
struct treballador plantilla[30];
```

Operacions bàsiques

- Creació d'una variable estructurada:
  ```
  struct treballador josep, antoni;
  ```
- Assignació d'estructures:
  ```
  josep = antonio;
  ```
- Accés a les dades contingudes en l'estructura (operador “.”):
  ```
  josep.nom = “Josep”;
  ```
- La lectura i l'escriptura de la informació de l'estructura s'ha de fer camp a camp:
  ```
  cin >> josep.nom;
  ```

```
    cin >> josep.sou;
    cin >> josep.numTelefon;
    cout << antoni.nom << antoni.numTelefon << endl;
```

Les estructures dins de les funcions

Les estructures s'empren dins de les funcions de la mateixa manera que els tipus de dades senzills. Es poden definir funcions que retornen una estructura (mitjançant una sentència return) i també poden rebre estructures com a paràmetres per valor i/o per referència.

Exemple d'una funció que retorna una estructura:

```
struct treballador DemanarDades(void);
```