Objectius de la pràctica:
- Utilització de les estructures de control repetitives en la realització de programes.

Sentències de control repetitives (bucles)

Els bucles modifiquen l'execució seqüencial de les instruccions i permeten l'execució repetida d'un conjunt de sentències. Amb aquest tipus d'estructura podem executar una tasca repetides vegades i, mitjançant el canvi d'alguns valors, fer que cada nova volta no siga exactament igual a l'anterior. Quan un bucle finalitza la seua execució, el programa continua endavant amb les instruccions que resten.

En C/C++ existeixen tres estructures de control repetitives: el bucle `for`, el bucle `while` i el bucle `do..while`. Encara que aquestes estructures són en el fons equivalents, es recomana l'ús de l'estructura `for` quan el bucle s'ha d'executar un nombre conegut de vegades. Respectivament, els altres dos bucles seran més adequats quan el que es conega siga la condició que s'ha de complir perquè es continue repetint el bucle.

- SENTÈNCIA `while`

Aquest bucle se sol emprar quan no es coneix d'antuvi el nombre de vegades que es repetirà el bucle sinó que el que es coneix és la condició que s'ha de complir perquè es repetisca.

El bucle `while` es correspon amb l'estructura "mentre" que existeix en altres llenguatges de programació. Aquesta estructura repeteix el cos del bucle mentre siga verdadera una expressió condicional. Així, doncs, executa un conjunt de sentències zero o més vegades, segons el valor de l'expressió.

Sintaxi:

```
while (Expressió Condicional)
{
    Bloc de sentències;
}
```

Les claus no són necessàries quan el bloc de sentències està format per una única sentència. L'execució succeeix de la manera següent:

1. S'avalua l'expressió. Si el resultat és fals, el bucle s'acaba. Si és vertader, continua amb el punt 2.
2. S'executa el bloc de sentències.
3. Torna al punt 1.

Exemple:

```
while (xt < 39)
{
    df *= xt;
    xt += 2;
}
```

- SENTÈNCIA `do..while`

En C/C++ no existeix l'estructura "repetir..fins a", en canvi, l'estructura "fer..mentre" permet repetir el cos d'un blucle mentre siga verdadera una expressió condicional. Per tant, el bucle `do..while` executa un conjunt de sentències una o més vegades, depenent d'una expressió condicional. Com succeïa per a la sentència `while`, es recomana l'ús de la sentència do..while quan no es coneix el nombre de vegades que s'ha de repetir el bucle. La diferència amb el tipus de bucle anterior és que la condició es comprova després de l'execució del bloc de sentències.

Sintaxi:

```
do
{
    Bloc de sentències;
}
while (Expressió Condicional);
```

Atenció, la sentència `do..while` acaba sempre amb un punt i coma (;). L'execució succeeix de la manera següent:

1. S'executa el bloc de sentències.
2. S'avalua l'expressió condicional. Si el resultat és fals s'ix del bucle. Si és vertader, es torna al punt 1.

Exemple:

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

int main (void)
{
    char opcio;

    do
    {
        cout << "Polsa una tecla numèrica: ";
        cin >> opcio;
    }
    while ( (opcio < '0') || (opcio > '9') );

    return 0;
}
```

- SENTÈNCIA `for`

El bucle `for` s'empra quan es necessita executar un nombre conegut de vegades una sentència o bloc de sentències. En C, el bucle for coincideix més o menys amb el bucle "des de ..fins a" que utilitzen altres llenguatges com ara Matlab o R. Tanmateix, hi ha algunes diferències que el fan diferent i alhora més flexible.

El bucle `for` permet executar un bloc de sentències un nombre conegut de vegades, fent servir un comptador que pren valors des d'un valor inicial, que va augmentant o disminuint, mentre que (ací radica la diferència) siga verdadera una condició.

Sintaxi:

```
for (iniciació comptador; condició; progressió comptador)
{
    Bloc de sentències;
}
```

El bucle comença amb la paraula reservada for, seguida de parèntesis. El contingut de dins dels parèntesis es divideix en tres camps separats per punt i coma:

- Primerament, es fixa el valor inicial del comptador.
- En segon lloc, es defineix l'expressió condicional que farà que es repetisca el bucle.
- Darrerament, es determinen els canvis que s'han d'aplicar al comptador en cada volta del bucle.

A continuació dels parèntesis, tancat entre claus, s'especifica el bloc de sentències que s'executaran en cada volta del bucle. Com en ocasions anteriors, les claus només són necessàries si el bloc està format per més d'una sentència.

L'execució del bucle segueix aquest ordre:

1. S'inicia el comptador.
2. S'avalua l'expressió condicional. Si és falsa el bucle finalitza. Si és verdadera, continua en el pas 3.
3. S'executa el bloc de sentències.
4. S'actualitza el comptador.
5. Torna al pas 2.

Exemple:

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

int main (void)
{
    int i;

    for (i = 1; i <= 10; i++)
        cout << i;

    return 0;
}
```

El bucle `for` és molt versàtil

El bucle for és molt flexible pel que fa al contingut de cadascun dels camps que hi ha entre parèntesis. Així, pot contenir diverses expressions d'iniciació i diverses expressions de progressió. En aquest cas, aquestes expressions se separen amb comes.

Sintaxi:

```
for (v1 = a1, v2 = a2,...; condició; progressió variables)
{
    Bloc de sentències;
}
```

On:

`vi = ei`	Representen els valors inicials que adquireixen les variables de control.
condició	Si és certa, s'executa el bloc de sentències.
progressió	Determina com evolucionaran les variables de control

Exemple:

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

int main (void)
{
    short i, j;

    cout << "Els deu primers parells són...\n";

    for (i = 0, j = 2; i < 10; i++, j += 2)
        cout << " el " << i << " és " << j << endl;

    cout << "Els deu primers nombres són...\n";

    for (i = 0, j = 0; i < 10; i++, j ++)
        cout << " el " << i << " és " << j << endl;

    return 0;
}
```

Exemple:

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

int main (void)
{
    long k;
    float m;

    for (k = 1, m = 9.0; k * m <= 15.0; k++, m -= 0.5)
        cout << k * m << endl;

    return 0;
}
```

Nota important: Poden haver diverses expressions d'iniciació i progressió de les variables de control però només pot haver una sola expressió condicional.

D'una altra banda, el bucle `for` admet que algun dels camps entre parèntesis estiga buit. Açò és, que no existisca cap instrucció d'iniciació i/o cap expressió condicional i/o cap expressió de progressió. Fins i tot, és possible escriure un bucle `for` amb tots els seus camps buits.

Exemple:

```
for ( ; ; )
{
    Bloc de sentències;
}
```

Ara bé, un bucle `for` sense expressió condicional produirà un bucle infinit que s'executarà indefinidament, excepte si hi ha dins del bloc de sentències una instrucció de salt (`break;`) que interrompa el flux normal del programa i finalitze el bucle forçosament. Com les sentències de salt no són imprescindibles i provoquen codis de programa poc estructurats, es recomana restringir el seu ús. Llavors, al llarg de l'assignatura evitarem emprar bucles infinits.

Bucles imbricats

Com a part del bloc de sentències que formen el cos d'un bucle podem incloure altres bucles. Açò s'anomena imbricació de bucles i proporciona major flexibilitat per a resoldre certs problemes.

El cas més senzill d'imbricació de bucles és el de dos bucles imbricats, açò és, un bucle dins d'un altre. El bucle intern (b.i.) és una sentència més dins del bloc de sentències del bucle extern (b.e.). Per tant, el cicle complet del b.i. s'executa repetidament tantes vegades com s'execute el b.e. Açò vol dir que una sentència que estiga dins del cos del b.i. s'executarà "m*n" vegades, essent "n" la quantitat de vegades que s'executaria si el b.i. no estiguera imbricat i "m" el nombre de vegades que s'executa el bloc de sentències del b.e.

Pas per pas, l'execució d'un bucle imbricat seguiria aquest ordre:

1. S'avalua l'expressió condicional del bucle més extern. Si és falsa, el programa ix dels dos bucles i continua amb l'execució. Si és vertadera, va al pas 2.
2. S'executa el bloc de sentències del bucle més extern. Entre aquestes es trobarà el bucle més intern.
   2.1. S'avalua l'expressió del bucle més intern. Si és falsa, ix del bucle intern i continua amb l'execució del bucle extern (pas 3). Si és certa, va al pas 2.2.
   2.2. S'executa el bloc de sentències del bucle intern.
   2.3. Torna al pas 2.1.
3. Torna al pas 1.

Exemple:

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

int main (void)
{
    short i, j;

    cout << "La taula de multiplicar del 1 al 10 és\n";

    for (i = 1; i <= 10; i++)
    {
        for (j = 1; j <= 10; j++)
            cout << i << " per " << j << " = " << i * j<< endl;
    }

    return 0;
}
```

Es poden imbricar bucles a molts nivells i amb diferents tipus de bucles.

Qüestió d'estil

La imbricació de bucles provoca, en els programadors novells, autèntics desgavells mentals per causa de l'anarquia amb què, a sovint, s'escriu el codi. Per a fer més fàcil la comprensió i la lectura d'un programa amb bucles imbricats, es recomana a l'alumne que el bloc de sentències dels bucles interiors se situen en columnes més interiors que les del bucle exterior, de manera que quede ben clar on comença i on acaba cada bucle.

Resum de conceptes bàsics de les estructures iteratives

Sentència while	Sentència do..while	Sentència for

`while (condició)`
`{`
`  sentència1;`
`  sentència2;`
`}`
 	
`Exemple:`
 	
`cin >> n;`
`num = 1;`
`while (num <= n)`
`{`
`  cout << num << endl;`
`  num++;`
`}`
	`do`
	`{`
	`  sentència1;`
	`  sentència2;`
	`}`
	`while (condició);`
	
	`Exemple:`
	
	`cin >> n;`
	`num = 1;`
	`do`
	`{`
	`  cout << num << endl;`
	`  num++;`
	`}`
	`while (num <= n);`
		`for (ini; cond; prog)`
		`{`
		`  sentència1;`
		`  sentència2;`
		`}`
		
		`Exemple:`
		
		`cin >> num;`
		`for (n = 1; n <= num; n++)`
		`  cout << n << endl;`

Comentaris d'interès

1. Les sentències while() i for() no finalitzen amb ';', mentre que la sentència do..while(); sí ho fa.

2. En el cos d'un bucle for no hem de variar el valor de les variables que formen part de la condició de finalització del bucle, ja que es podrien produir efectes inesperats.

3. Quan hem d'emprar un bucle o un altre?
   - La sentència while se sol emprar quan no es coneix el nombre d'iteracions del bucle, sols que ha de ser major o igual a 0.
   - La sentència do..while se sol emprar quan no es coneix el nombre d'iteracions, sols que ha de ser major o igual a 1.
   - La sentència for se sol emprar quan es coneix exactament el nombre d'iteracions del bucle.

4. Tècniques de control dels bucles:
   - Bucles controlats per indicadors de bandera (*flags*): S'empra una variable booleana del valor de la qual depén l'acabament del bucle:

Sintaxi:
`bool continuar;`

```
continuar = true;
while (continuar)
{
    ...
    if (condició d'acabament)
        continuar = false;
    ...
}
```

- Bucles controlats per sentinella: En un procés d'introducció de dades, el sentinella és el valor que, quan es llegeix, indica que el procés ha de finalitzar:

Exemple:

```
suma = 0;
cout << "Introdueix els nombres a sumar, 0 per a acabar";
cin >> num;
while (num != 0)
{
    suma = suma + num;
    cout << "Introdueix els nombres a sumar, 0 per a acabar";
    cin >> num;
}
cout << suma;
```