UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

Especificació i ús de mòduls en C++ (II)

3.1 Ús de la classe `Cami`

Passem ara a treballar amb la classe `Cami` que hem vist a teoria. Podeu consultar la seva especificació en el fitxer `Cami.hpp` i també un exemple d'ús de la classe en el fitxer `main_cami.cpp`. Teniu el fitxer `Cami.hpp` a la ruta `INCLUSIONS` i el fitxer `Cami.o` a la ruta `OBJECTES`.

El programa `main_cami.cpp` usa dues classes, per això el procés complet per obtenir l'executable `main_cami.exe` serà:

```
pro1++ -c main_cami.cpp -I$INCLUSIONS

pro1++ -o main_cami.exe main_cami.o $OBJECTES/{Punt.o,Cami.o}
```

O si voleu fer els dos passos amb una sola comanda:

```
pro1++ -o main_cami.exe main_cami.cpp $OBJECTES/{Punt.o,Cami.o} -I$INCLUSIONS
```

3.1.1 Exercici: menú d'operacions sobre un camí

Ex.1 Escriviu una aplicació simple que llegeixi un camí i ofereixi la possibilitat de provar sobre ell les operacions següents:

- afegir un nou punt (opció -1)
- consultar les coordenades d'un punt a partir de la seva posició (opció -2)
- crear i escriure un camí nou resultant d'intercanviar les coordenades de tots els punts del camí original (opció -3)
- escriure el camí (opció -4)

El programa principal serà un procés iteratiu que, després de llegir el camí, haurà de llegir un valor entre -1 i -5 indicant l'operació a aplicar (l'opció -5 serà per sortir del programa), i les dades que necessiti per poder aplicar l'operació.

Les operacions -1, -2 i -4 ja són operacions de la classe, per això només us caldrà comprovar les seves precondicions per cridar-les. Si la precondició d'una operació no se satisfà, no s'ha de fer la seva crida i s'ha de mostrar un missatge indicant-ne el motiu.

L'opció −3 requereix programar una operació nova en el fitxer que conté el `main`. Aquesta operació estarà basada en les operacions de la classe `Cami` i la classe `Punt`.

El fitxer `menu_cami.ent` conté un exemple de dades i el fitxer `menu_cami.sort` conté un exemple de resultats.

Implementació de mòduls en C++

3.2 Implementació de la classe `Cami`

Ara estudiarem la implementació de la classe `Cami` que es troba en el fitxer `Cami.cpp`. També farem algunes modificacions sobre ella.

3.2.1 Elements públics i privats; elements visibles i ocults

A les classes de teoria hem explicat que l'especificació d'un mòdul ha de ser independent de la seva implementació. L'especificació d'un mòdul no ha de contenir elements que permetin a un usuari saber com està implementat ni tampoc ha de permetre a un usuari accedir directament als seus elements per consultar-los o modificar-los.

Quan s'implementa una classe en C++, existeixen diversos mecanismes per aconseguir aquesta independència. Podem veure un exemple a l'arxiu `Cami.hpp`, on hi ha dues seccions: la declaració d'**elements privats** i la declaració d'**elements públics**. A la primera s'hi troba, normalment, la declaració dels camps de dades (atributs) de la classe, i a la segona s'hi troben les capçaleres de les operacions (mètodes) públiques. Això permet que la part pública pugui ser usada fora de la classe mentre que no permet que la part privada pugui ser usada fora de la classe.

A les nostres implementacions, els camps privats seran "visibles" però tindrem en compte en tot moment que són privats.

3.2.2 Exemple: una primera implementació de la classe `Cami`

Obriu els arxius `Cami.hpp` i `Cami.cpp`. Podeu repassar l'arxiu `Cami.hpp` i hi veureu les dues seccions: els camps de dades (visibles, però privats) i les capçaleres de les operacions de la classe (públiques).

La representació escollida pel tipus `Cami` és una llista de camps de diferents tipus:

- un vector de punts (objectes de la classe `Punt`)
- una constant entera (`MAX_PUNTS`), per limitar la seva capacitat, declarada `static` perquè pugui ser compartida per tots els objectes de la classe
- un enter `npunts` que ens permet saber quina part del vector està ocupada

Noteu que no cal usar un `struct` o altres constructors del llenguatge C++ doncs el tipus `class` de C++ ja és un tipus estructurat (és un tuple on per defecte tots els seus atributs són privats).

El codi de les operacions apareix a l'arxiu `Cami.cpp`. Observeu que les capçaleres de les operacions de l'arxiu `Cami.cpp` estan precedides per la declaració `Cami::`[1] per indicar que aquestes operacions són les que s'han declarat a l'arxiu `Cami.hpp`. Qualsevol altra operació que trobéssim a `Cami.cpp`, seria considerada com aliena a la classe i no tindria dret a veure els seus camps i tampoc tindria paràmetre implícit.

3.2.3 Exercici: constructors i destructors

Si una classe no té cap constructor explícit, el C++ proporciona a la classe un constructor per defecte (en el cas de la classe `Cami`, aquest constructor seria Cami()), però si la classe ja en té un de definit, el constructor per defecte no existeix.

Els exercicis del 2 al 7 els podeu fer en fitxers nous, `Cami2.hpp` i `Cami2.cpp`, respectivament. La classe que especifiquen i implementen segueix sent la classe `Cami`.

Ex.2 Introduïu un constructor addicional que permeti declarar i copiar camins simultàniament. La seva capçalera serà:

```
Cami(const Cami &c)
/* Pre: cert */
/* Post: el resultat és un camí nou, còpia de c */
```

Recordeu que quan es programa dins d'una classe es pot accedir directament als camps dels objectes de la mateixa classe, no només als del paràmetre implícit de l'operació. Per exemple, si volem assignar el nombre de punts del camí `c` al paràmetre implícit d'aquesta operació, podem fer

```
npunts = c.longitud();
```

però també podem fer (de fet, és millor)

```
npunts = c.npunts;
```

Per aconseguir que al declarar un nou camí `c2`, aquest sigui una còpia de `c1`, cridarem a l'operació així:

```
// ... suposem que c1 ja ha estat creat

 Cami c2(c1);
```

[1] A l'operador `::` se l'anomena *operador de resolució d'àmbit.*

Ex.3 Modifiqueu el programa principal `menu_cami` fet anteriorment perquè faci totes les operacions sobre un camí creat a partir d'una còpia de l'original.

Pel que fa als destructors, el C++ es comporta de manera anàloga, és a dir, si una classe no defineix cap destructor, el C++ li'n proporciona un per defecte. Per ara, només ens caldrà això, però durant el curs veurem classes que sí necessitaran que programem el seu destructor.

Ex.4 Comproveu el funcionament del destructor `~Cami()` inserint en el seu codi una instrucció que escrigui un missatge per pantalla. Veureu que en executar el programa `menu_cami` (o qualsevol altre), el missatge s'escriu cada vegada que es destrueix un objecte de la classe `Cami`, és a dir, quan s'allibera l'espai d'un camí.

3.2.4 Exercici: canvis en les especificacions

Ex.5 Reprogrameu l'operació `allargar` perquè la seva precondició sigui només `cert` i sigui la pròpia operació la que faci la comprovació corresponent. La nova capçalera seria:

```
void allargar(const Punt &p, bool &b);
/* Pre: cert */
/* Post: b indica si el punt rebut es pot afegir al
   p.i.; si b és cert, el punt s'ha afegit al p.i. */
```

Ex.6 Feu un canvi similar a l'operació `consultar_punt`. Ara aquesta operació haurà de ser `void`: ja no retornarà cap `Punt` sinó que l'obtindrà com a paràmetre per referència, juntament amb el booleà que indicarà si la posició rebuda existeix o no al p.i.

```
void consultar_punt(int i, Punt &p, bool &b) const;
/* Pre: cert */
/* Post: p és el punt i-èssim del p.i. i b és true si
   1 <= i <= nombre de punts del camí, sinó b és false */
```

Ex.7 Reprogrameu `menu_cami` tenint en compte tots aquests canvis. Recompileu i munteu els fitxers necessaris per provar l'executable resultant de `menu_cami`. Per aquest exercici, redireccioneu el canal d'entrada per agafar les dades de `menu_cami.ent` i mireu els resultats pel canal estàndard de sortida. Si voleu fer `diff` entre la vostra sortida i `menu_cami.sort`, haureu d'eliminar o comentar el missatge que heu escrit en el destructor.

3.2.5 Exercici: ampliació de la classe `Cami`

Ex.8 Enriquiu la classe `Cami` amb les operacions `escurcar()` i `darrer_punt()`, definint-les **fora** de la classe. Feu un mòdul funcional amb aquestes dues operacions, presentades a

teoria, tenint en compte que no seran mètodes orientats a objecte.

Les especificacions d'aquestes operacions, que hem vist a teoria, són:

```
void escurcar(Cami &c);
/* Pre: el camí c no pot estar buit */
/* Post: c conté els mateixos punts que tenia menys el darrer */
```

```
Punt darrer_punt(const Cami &c);
/* Pre: el camé c no pot estar buit */
/* Post: el resultat és el darrer punt de c */
```

Ex.9 Amplieu el programa `menu_cami`, el podeu anomenar `menu_cami_ext`, afegint dues noves opcions per provar aquestes noves operacions. Així, el nou menú serà:

- afegir un nou punt (opció -1)
- consultar les coordenades d'un punt a partir de la seva posició (opció -2)
- crear i escriure un camí nou resultant d'intercanviar les coordenades de tots els punts del camí original (opció -3)
- escriure el camí (opció -4)
- escurçar el camí (opció -5)
- consultar el darrer punt del camí (opció -6)

L'opció -7 serà per sortir del programa.

El fitxer `menu_cami_ext.ent` conté un exemple de dades i el fitxer `menu_cami_ext.sort` conté un exemple de resultats.