Problema 2: Especificació i implementació de la classe `Person`

Esteve Brugulat Josep M. Ribó

26 d'octubre de 2009

1 Objectius

- Introduir els apuntadors, els objectes dinàmics i les referències (1.6.2, 1.6.3, 1.6.4, apèndix A)
- Dissenyar una classe que té com a atributs objectes d'una altra classe
- Introduir els constructors de C++ amb els seus diferents tipus (1.7)
- Introduir els inicialitzadors per classes agregades (1.7, 1.11)
- Introduir les deixalles i els destructors (1.8)

2 Especificació de la classe `Person`

La classe `Person` està dissenyada per tal de modelitzar algunes informacions sobre persones. En particular, les informacions que ens interessen són: el nom, el nif, l'adreça i la data de naixement.

1. Quin fitxer usaràs per tal d'especificar la classe `Person`?

   `Person.txt`

2. Per tal d'especificar aquesta classe introduirem les operacions *constructores* proporcionades directament per C++. Per què és interessant disposar d'operacions constructores en lloc d'usar les operacions de l'estil `createPerson` com les que vam usar al problema 1? **(APUNTS: 1.7.1)**

   - **Si usem operacions de l'estil `createPerson`:**
     Immediatament després de ser creats, els objectes de la classe `Person` queden en estat indefinit. Aplicar operacions sobre ells genera errors.

```
int main()
{

 Person p;


 p.getName(nam);  //MALAMENT! No hem donat cap valor
                  //al nom de la persona p

 p.createPerson(...,"joe",...);

 p.getName(nam);  //Ara correcte

}
```

- **Si usem operacions constructores**
  L'anterior no passa. Podem usar un objecte de la classe `Person` immediatament després d'haver-lo creat.

```
int main()
{

 Person p(..."joe"...);


 p.getName(nam);  //Correcte


}
```

3. Especifica tres operacions constructores per la classe `Person` **(APUNTS: 1.7; DOCUMENT: Manaments de disseny de classes)**

   No les especifico completament, simplement les enumero. Us deixo l'especificació completa per vosaltres:

   - `Person();`
     Constructora per defecte.
   - `Person(char* pname, char* pnif, char* address , Date birth );`
     Inicialitza un objecte de classe `Person` *amb els paràmetres donats (pel nom, nif, adreça i data de naixement)*
   - `Person(const Person& p);`
     Constructor copiador. Inicialitza un objecte de la classe `Person` *amb una còpia del paràmetre* `p`.

4. De la mateixa manera, especifica una operació destructora **(APUNTS: 1.8; DOCUMENT: Manaments de disseny de classes)**

   `~Person()`

   Destrueix tota la memòria dinàmica associada a un objecte de la classe `Person`.

5. Ara identifica i especifica la resta de les operacions de la classe `Person`. Aquestes operacions hauran d'estar orientades a permetre gestionar la informació que emmagatzema un objecte de la classe `Person` (nom, nif, adreça, data naixement). **(DOCUMENT: Manaments de disseny de classes)**

   - `void setName(char* pname);`
   - `void setNif(char* pnif);`
   - `void setBirthdate(Date pbirth);`
   - `void setAddress(char* paddr);`
   - `char* getName();`
   - `char* getNif();`
   - `Date getBirthdate();`
   - `char* getAddress();`

   Les especificacions són molt similars a les de la classe `Date` *(Problema 1). Us les deixo per vosaltres.*

3 Representació de la classe Person

1. Quins atributs haurá de contenir aquesta classe?

   - `name`*: el nom de la persona*
   - `nif`*: el seu nif*
   - `address`*: la seva adreça*
   - `birthdate`*: la seva data de naixement*

   Per fer la representació de la classe `Person` *usarem el criteri següent:*

   - *Els atributs* `name, address` *els representarem en memòria dinàmica (ja que, a priori no sabem quants caràcters com a màxim integraran una adeça o un nom). Què guanyem si els representem en memòria dinàmica?* **(APUNTS: 1.6.2, 1.6.3)**
     Si els representem en memòria dinàmica (`char* name`*) en lloc de com un array reservat en temps de compilació (*`char name[40]`*) guanyem que quan compilem el programa no hem de fer cap previsió sobre el nombre de caràcters que s'han de reservar per emmagatzemar el nom d'una persona (ens podríem quedar curts o malbaratar espai). Aquesta reserva es farà en temps d'execució i es reservaran exactament el nombre de caràcters que ocupa el nom de la persona.*
   - *L'atribut* `nif` *el representarem com un simple array de caràcters. Per què no el representem en memòria dinàmica?*
     Perquè tots els nifs tenen 9 caràcters.
     El representarem així: `char nif[10];`
     (Reservem un caràcter addicional per la marca de final: `\0`*)*
   - *Com representarem l'atribut* `data-de-naixement`*?*
     Com un atribut de la classe `Date`*:*
     `Date birthdate;`
     Podem usar classes ja dissenyades per definir-ne de noves.

2. Finalment, fes la representació de la classe `Person`
   Fitxer `Person.h`:

```
#ifndef PERSON_H
#define PERSON_H

#include "Date.h"

class Person{

  char* name;
  char nif[10];
  char* address;
  Date birthdate;

 public:

  Person();
  Person(char* pname, char* pnif, char* paddress, Date pbirth);
  Person(const Person& p);
  ~Person();
  void setName(char* pname);
```

```
  void setNif(char* pnif);
  void setBirthdate(Date pbirth);
  char* getName();
  char* getNif();
  Date getBirthdate();
};

#endif
```

4 Implementació de les operacions de la classe Person

1. En quin fitxer hauràs de posar aquesta implementació? **(APUNTS: 1.5.4)**
   Al fitxer `Person.cc`
2. Implementa les operacions constructores **(APUNTS: 1.7)**.
   Per fer aquesta implementació has de recordar vàries coses:
   - **Abans d'assignar els valors als atributs dinàmics (name, nif, address) hauràs de reservar la memòria que ocuparan aquells valors.** Com ho faràs? **(APUNTS: 1.7.1)**
   - Les cadenes de caràcters no les pots assignar directament amb l'operador d'assignació (=).
   - Quan una classe té com a atribut un objecte d'una altra classe (`Person` té com a atribut un objecte de la classe `Date`) es diu que la primera (`Person`) és una *classe agregada*. El constructor d'una classe agregada pot usar *inicialitzadors*. Què és un inicialitzador, per a què serveix i com s'usa en la classe `Person`? **(APUNTS: 1.11, 1.11.2)**

   Sobre el darrer apartat és interessant recordar el següent (vegeu 1.11 i 1.11.2):

   Quan es crea un objecte `p` *de classe agregada (com* `Person`*), s'ha de crear també un objecte de classe* `Date` *que està contingut dins de* `p` *(es diu que* `Date` *és una classe component de* `Person`*). Per aquest motiu, el compilador de C++ genera automàticament una crida a una operació constructora de la classe* `Date` *que inicialitza adequadament l'atribut* `birthdate`*. Si no es diu el contrari, es crida al constructor per defecte de la classe* `Date` *(per la qual cosa,* `birthdate` *quedaria inicialitzat a 1-1-1900).*

   Si es vol cridar un altre constructor es pot fer mitjançant els anomenats inicialitzadors:

```
Person::Person(char* pname, char* pnif, char* paddress, Date pbirth)
:birthdate(pbirth.getDay(),pbirth.getMonth(),pbirth.getYear())
{
  strcpy(nif,pnif);

  name=new char[strlen(pname)+1];
  strcpy(name,pname);

  address=new char[strlen(paddress)+1];
  strcpy(address,paddress);
}
```

   Inicialitzar l'atribut de classe `Date` *d'aquesta manera és més elegant i més correcte que fer-ho dins del codi del constructor:*

```
birthdate=pbirth;
```

   o bé:

```
birthdate.copy(pbirth);
```

   Més avall es mostra la implementació de les operacions constructores.

3. I com faràs la implementació de l'operació destructora? **(APUNTS: 1.8.2, 1.8.3)**
   vegeu més avall

4. Implementa la resta de les operacions de la classe.
   Fitxer `Person.cc`**:**

```
#include "Person.h"
#include "Date.h"

Person::Person(char* pname, char* pnif, char* paddress, Date pbirth)
  :birthdate(pbirth.getDay(),pbirth.getMonth(),pbirth.getYear())
{
  strcpy(nif,pnif);

  name=new char[strlen(pname)+1];
  strcpy(name,pname);

  address=new char[strlen(paddress)+1];
  strcpy(address,paddress);

  //Tambe podriem usar el constructor copiador de la classe Date
  //com a inicialitador de la manera seguent:
  //   :birthdate(pbirth)
}

Person::Person(const Person& p)
  :birthdate(p.birth.getDay(),p.birth.getMonth(),
             p.birth.getYear())
{
  strcpy(nif,p.nif);

  name=new char[strlen(p.name)+1];
  strcpy(name,p.name);

  address=new char[strlen(p.address)+1];
  strcpy(address,p.address);
}

Person::Person()
{
  strcpy(nif,"noassign");

  name=new char[11];
  strcpy(name,"noassignat");

  address=new char[11];
  strcpy(address,"noassignat");

//En aquest cas no cal cap inicialitzador perque
//si no se'n posa cap, el compilador de C++
//crida al constructor per defecte de la
//classe component (Date)

}

Person::~Person()
```

```
{
  delete [] name;
  delete [] address;
}

void Person::setName(char* pname)
{
  delete [] name;

  name=new char[strlen(pname)+1];
  strcpy(name,pname);

}

void Person::setNif(char* pnif)
{
  strcpy(nif,pnif);
}

void Person::setBirthdate(Date pbirth)
{
  birthdate=pbirth;
}

char* Person::getName()
{
  char* pname=new char[strlen(name)+1];
  strcpy(pname,name);
  return pname;
}

char* Person::getNif()
{
  char* pnif=new char[strlen(nif)+1];
  strcpy(pnif,nif);
  return pnif;
}

void Person::getBirthdate(Date& pdate)
{
  pdate.copy(birthdate);
       //tambe correcte: pdate=birthdate;
}
```

5 Ús de la classe `Person`

1. En quin fitxer has de posar un programa que usi la classe `Person`? **(APUNTS: 1.5.4, 1.5.5)**

   `user.cc`

2. Quins fitxers has d'incloure per tal de poder usar la classe `Person`? **(APUNTS: 1.5.4)**

   `Person.h` i `Date.h`

   Si volem enviar coses a la sortida estàndar, també necessitarem incloure la biblioteca `iostream`

3. Implementa un petit programa que usi la classe `Person` **(APUNTS: 1.5.4)**

```
#include "Person.h"
#include "Date.h"
#include <cstring>
#include <iostream>

using namespace std;


int main()
{

  char nam[20];
  char nif[10];
  char address[20];
  bool err;
  char* nam2;

  Date birth(10,12,1990,err);


  strcpy(nam, "joe");
  strcpy(nif,"12345678A");
  strcpy(address,"C. Jaume II");

  Person p(nam,nif,address,birth);

  nam2=p.getName();

  cout<<"nom="<<nam2<<endl;

  return 0;
}
```

4. Què cal fer per tal d'executar aquest programa? **(APUNTS: 1.5.4)**

```
$ g++ -c Person.cc
$ g++ -c Date.cc
$ g++ -c user.cc
$ g++ user.o Date.o Person.o -o user
```