Problema 5: Classes genèriques: La classe `Stack`

Esteve Brugulat Josep M. Ribó

30 d'octubre de 2009

1 Objectius

- Introduir les classes genèriques (templates)

Per fer aquest problema, cal que us llegiu:
- Els apunts (apartat 1.10)

2 Especificació de la classe `StackOfChar`

Una pila (en anglès, *stack*) és una estructura de dades tal que el darrer element en ser inserit és precisament el primer en sortir (LIFO: Last Input, First Output). Us podeu imaginar una pila de plats: per treure el de sota (que va ser el primer en ser inserit), cal treure primer tots els altres si no es vol provocar un cataclisme. les piles tenen gran importància en programació (per fer recorreguts en fondària o per seqüencialitzar la recursivitat, per exemple).

Considerem, doncs, la classe `StackOfChar` (*Pila de caràcters*) amb les operacions següents:

- `StackOfChar();`

  Crea una pila nova amb capacitat per a MAXCAR caràcters (MAXCAR és una constant amb un valor determinat).

- `StackOfChar(unsigned int n);`

  Crea una nova pila amb capacitat per a `n` caràcters.

- `void putFirst(char c, bool& err);`

  Afegeix `c` al capdamunt de la pila. Què significa `err`??

- `void removeFirst(bool& err);`

  Treu l'element del capdamunt de la pila.

- `char getFirst(bool& err) const;`

  Obté l'element del capdamunt de la pila.

- `bool isEmpty() const;`

  Indica si la pila és buida o no.

- `bool isFull() const;`

  Indica si la pila és plena o no.

- `void copy(const StackofChar& s);`
  Fa una còpia de la pila `s` a la pila sobre la qual es crida l'operació.
- `bool equals(const StackOfChar& s) const;`
  Retorna cert si la pila `s` conté els mateixos elements que la pila sobre la qual es crida l'operació.
- `void reset();`
  Buida la pila.
- `unsigned int getCapacity() const;`
  Retorna la capacitat de la pila.
- `void showContents() const;`
  Mostra els elements continguts a la pila.

1. Fes l'especificació completa de la classe `StackOfChar` **(APUNTS: 1.4, 1.5)**
   Fitxer: `StackOfChar.txt`:
   - `StackOfChar();`
   - `StackOfChar(unsigned int);`
     - **Crida:** `StackOfChar s(n);`
     - **Pre:**
     - **Post:** `s` és una pila nova, buida i amb capacitat per a `n` caràcters com a màxim[1].
   - `void putFirst(char, bool&);`
     - **Crida:** `s.putFirst(c,err);`
     - **Pre:**
     - **Post:** `s` és la mateixa pila que `s'` a la que s'ha afegit el caràcter `c` a la primera posició; `err=fals`
       Si `s'.isFull()` `err=cert` i `s=s'`
   - `void removeFirst(bool&);`
     - **Crida:** `s.removeFirst(err);`
     - **Pre:**
     - **Post:** `s` és la mateixa pila que `s'` sense l'element que ocupava la primera posició (el darrer en ser inserit); `err=fals`
       Si `s'.isEmpty()` `err=cert` i `s=s'`
   - `char getFirst(bool& err) const;`
     - **Crida:** `c=s.getFirst(err);`
     - **Pre:**
     - **Post:** `c` és el primer element de la pila `s` (el darrer en ser inserit); `s=s'` i `err=fals`.
       Si `s'.isEmpty()` `err=cert`; `s=s'` i `c` és indefinit.
   - `bool isEmpty() const;`
     - **Crida:** `b=s.isEmpty();`
     - **Pre:**
     - **Post:** Si `s` és la pila buida, aleshores `b=cert`. Altrament, `b=fals`. `s=s'`
   - `bool isFull() const;`

[1] A l'apartat 1.14 estudiarem com tractar l'error que es produeix si no hi ha suficient memòria per crear una pila de capacitat `n`

  - **Crida:** `b=s.isFull();`
  - **Pre:**
  - **Post:** Si `s` conté exactament `s.getCapacity()` caràcters, aleshores `b=cert`. Altrament, `b=fals`. `s=s'`

- `void copy(const StackofChar&);`
  - **Crida:** `s.copy(s2);`
  - **Pre:**
  - **Post:** La pila `s2` s'ha copiat a `s`. Ara, `s` conté els mateixos carácters, en el mateix ordre i la mateixa capacitat que `s2`.

- `bool equals(const StackOfChar& s) const;`
  - **Crida:** `b=s.equals(s2);`
  - **Pre:**
  - **Post:** Si `s` conté exactament els mateixos caràcters que `s2` i en el mateix ordre, aleshores `b=cert`. Altrament, `b=fals`.
    Notem que `s` i `s2` podríen tenir capacitats diferents i, així i tot, `s.equals(s2)` podria retornar cert.

- `unsigned int getCapacity() const;`
  - **Crida:** `n=s.getCapacity();`
  - **Pre:**
  - **Post:** `n` és el nombre màxim de caràcters que pot contenir `s`.
    `n` és la capacitat que s'assigna a `s` després de fer `s.create(n);` o com a conseqüència de `s.copy(s2);` on `s2` compleix que `s2.getCapacity()==n`.

- `void reset();`
  - **Crida:** `s.reset();`
  - **Pre:**
  - **Post:** `s` és la pila buida.

3 Representació de la classe `StackOfChar`

1. Representa la classe `StackOfChar` usant un array reservat en memòria dinàmica. Dóna el codi del fitxer `StackOfChar.h`. **(APUNTS: 1.5)**

   Fitxer `StackOfChar.h`:

```
class StackOfChar{
 private:
  char* v;
  unsigned int nelems;
  unsigned int capacity;
 public:

 StackOfChar();
 StackOfChar(unsigned int);
 void putFirst(char, bool&);
 void removeFirst(bool&);
 char getFirst(bool& err) const;
 bool isEmpty() const;
 bool isFull() const;
 void copy(const StackOfChar&);
```

```
 bool equals(const StackOfChar& ) const;
 unsigned int getCapacity() const;
 void reset();
};
```

2. Suposem ara que tenim una classe `StackOfInt` (pila d'enters). Enumera les seves operacions i fes la representació d'aquesta nova classe (en definitiva, dóna el codi del fitxer `StackofInt.h`).

   Fitxer `StackOfInt.h`**:**

```
class StackOfInt{
 private:
  int* v;                  //en lloc de char* v;
  unsigned int nelems;
  unsigned int capacity;
 public:

 StackOfInt();
 StackOfInt(unsigned int);
 void putFirst(int, bool&);  //en lloc de putFirst(char,bool&)
 void removeFirst(bool&);
 int getFirst(bool& err) const; //en lloc de char getFirst(...)
 bool isEmpty() const;
 bool isFull() const;
 void copy(const StackOfInt&);
 bool equals(const StackOfInt& ) const;
 unsigned int getCapacity() const;
 void reset();
};
```

   Notem que les operacions `equals` *i* `copy` *s'haurien pogut substituir per la sobrecàrrega dels operadors* `==` *i* `=`*, respectivament.* **Fes-ho**.

4 Implementació de les classes `StackOfChar` i `StackOfInt`

1. Implementa les operacions de la classe `StackOfChar` **(APUNTS: 1.5)**

```
const unsigned int MAXCAR=10;

StackOfChar::StackOfChar()
{
  v=new char[MAXCAR];
  capacity=MAXCAR;
  nelems=0;
}

StackOfChar::StackOfChar(unsigned int n)
{
  v=new char[n];
  capacity=n;
  nelems=0;
}

void StackOfChar::putFirst(char c, bool& err)
{
  if (nelems<capacity) {v[nelems]=c; err=false; nelems++;}
```

```
  else {err=true;}
}

void StackOfChar::removeFirst(bool& err)
{
  if (nelems>0) {nelems--; err=false;}
  else {err=true;}
}

char StackOfChar::getFirst(bool& err) const
{
  if (nelems>0) {return v[nelems-1];}
  else {err=true; return 0;}
}


bool StackOfChar::isEmpty() const
{
  return nelems==0;
}

bool StackOfChar::isFull() const
{
 return nelems==capacity;
}

void StackOfChar::copy(const StackOfChar& s2)
{
  int i;

  v=new char[s2.capacity];
  capacity=s2.capacity;
  nelems=s2.nelems;

  for (i=0;i<nelems;i++)
  {
    v[i]=s2.v[i];
  }
}

bool StackOfChar::equals(const StackOfChar& s2) const
{
  int i;

  if (s2.nelems!=nelems) {return false;}
  else {
    i=0;
    while (i<nelems && v[i]==s2.v[i]){i++;}
    return i==nelems;
  }
}

unsigned int StackOfChar::getCapacity() const
{
  return capacity;
}

void StackOfChar::reset()
```

```
{
  nelems=0;
}
```

2. Hi ha moltes diferències entre la implementació de `StackOfChar` i `StackOfInt`?

   De fet, la implementació de les operacions de totes dues classes és idèntic. L'única diferència és que en un cas es treballa sobre caràcters i en un altre sobre enters.

5 Classes genèriques

1. Especifica la classe genèrica `Stack<T>` capaç de contenir elements de qualsevol tipus `T` **(APUNTS: 1.10.1)**.

2. Ara representa i implementa la classe `Stack<T>`. A quin fitxer posaràs la representació de la classe `Stack` i la implementació de les seves operacions? **(APUNTS: 1.10.1)**.

   Quan dissenyem una classe genèrica (template) tant la representació de la classe com la implementació de les seves operacions es posa al fitxer .h.

   Per tant, posarem la representació de la classe i la implementació de les operacions al fitxer `Stack.h`.

```
const unsigned int MAXELEMS=100;

template <class T>
class Stack{

 private:

  T* v;
  unsigned int nelems;
  unsigned int capacity;

 public:

 Stack();
 Stack(unsigned int);
 void putFirst(T, bool&);
 void removeFirst(bool&);
 T getFirst(bool& err) const;
 bool isEmpty() const;
 bool isFull() const;
 void copy(const Stack<T>&);
 bool equals(const Stack<T>& ) const;
 unsigned int getCapacity() const;
 void reset();
};

template <class T>
Stack<T>::Stack()
{
  v=new T[MAXELEMS];
  capacity=MAXELEMS;
  nelems=0;
```

```
}


template <class T>
Stack<T>::Stack(unsigned int n)
{
  v=new T[n];
  capacity=n;
  nelems=0;
}

template <class T>
void Stack<T>::putFirst(T c, bool& err)
{
  if (nelems<capacity) {v[nelems]=c; err=false; nelems++;}
  else {err=true;}

}

template <class T>
void Stack<T>::removeFirst(bool& err)
{
  if (nelems>0) {nelems--; err=false;}
  else {err=true;}
}

template <class T>
T Stack<T>::getFirst(bool& err) const
{
  if (nelems>0) {return v[nelems-1];}
  else {err=true; return 0;}
}


template <class T>
bool Stack<T>::isEmpty() const
{
  return nelems==0;
}

template <class T>
bool Stack<T>::isFull() const
{
 return nelems==capacity;
}

template <class T>
void Stack<T>::copy(const Stack<T>& s2)
{
  int i;

  v=new T[s2.capacity];
  capacity=s2.capacity;
  nelems=s2.nelems;

  for (i=0;i<nelems;i++)
  {
    v[i]=s2.v[i];
```

```
  }
}

template <class T>
bool Stack<T>::equals(const Stack<T>& s2) const
{
  int i;

  if (s2.nelems!=nelems) {return false;}
  else {
    i=0;
    while (i<nelems && v[i]==s2.v[i]){i++;}
    return i==nelems;
  }
}

template <class T>
unsigned int Stack<T>::getCapacity() const
{
  return capacity;
}

void Stack<T>::reset()
{
  nelems=0;
}
```

6 Ús de la classe Stack

1. Escriu un programa que instancïi la classe `Stack<T>` amb `int` i amb `Person`

   Fitxer `Stack<T>` **:**

```
#include "Stack.h"
#include "Person.h"


int main()
{
  Stack<int> sint;     //S'instancia T amb  int
  Stack<Person> sper;  //S'instancia T amb Person
  int i;
  Person p;

  sper.putFirst(p,err);
  sint.putFirst(i);

}
```

2. Cal fer algun requeriment a l'especificació d'aquesta classe respecte alguna operació que han d'oferir els tipus que instancïin `T`? **(APUNTS: 1.10.2)**.

   Sí, cal fer el següent requeriment:

   Els tipus que instanïin `T` *han de tenir sobrecarregat els operadors* `==` *i* `=`.

Aquest requeriment ha d'estar inclòs a l'especificació de la classe `Stack<T>`

El motiu d'aquest requeriment és que les implementacions de les operacions `copy` *i* `equals` *suposen que el tipus que instancia* `T` *té aquests operadors sobrecarregats:*

```
void Stack<T>::copy(const Stack<T>& s2)
{
 ....
  for (i=0;i<nelems;i++)
  {
    v[i]=s2.v[i];  //AQUI ES NECESSITA QUE EL TIPUS QUE INSTANCII T
                   //TINGUI SOBRECARREGAT L'OPERADOR =
  }
}

bool Stack<T>::equals(const Stack<T>& s2) const
{
 ...
    i=0;
    while (i<nelems && v[i]==s2.v[i]){i++;}
                //AQUI ES NECESSITA QUE EL TIPUS QUE INSTANCII T
                //TINGUI SOBRECARREGAT L'OPERADOR ==
  ...
  }
}
```

7 Funcions genèriques

A l'assignatura de MTP vam presentar algorismes d'ordenació d'un array. Per exemple:

```
void bubblesort(int v[], unsigned int start, unsigned int end);
```

Aquesta acció ordena l'array d'enters `v[start..end]` usant el mètode de la bombolla.

1. Implementa l'acció genèrica bubblesort de manera que sigui capaç d'ordenar un array de *qualsevol tipus d'elements* (no només enters). Fes també un petit programa que cridi aquesta acció genèrica amb arrays de diferents tipus d'elements. **(APUNTS: 1.10.2)**.

```
template <class T>
void bubblesort(T v[], unsigned int start, unsigned int end)
{
  unsigned int i,j;

  i=start;
  while (i<end){
    j=end;
    while (j>i){
      if (v[j]<v[j-1]){
        aux=v[j]; v[j]=v[j-1]; v[j-1]=aux;
      }
      j--;
    }
    i++;
  }
}

int main()
```

```
{

  int v1[10];
  Person v2[10];

  initializeInt(v1,0,9);    //Crida a una accio que omple v1
  initializePerson(v2,0,9); //Crida a una accio que omple v2

  bubblesort(v1,0,9);       //Ordena un array de int

  bubblesort(v2,0,9);       //Ordena un array de Person
}
```

2. Cal fer algun requeriment especial a l'especificació d'aquesta acció genèrica? **(APUNTS: 1.10.2)**.

   Sí, cal fer-lo:

   La classe (o el tipus predefinit) amb el qual s'instancïi el paràmetre `T` *ha de tenir sobrecarregats els operadors* `=` *i* `<` *(per què?).*

   Noteu que a la implementació que vam fer al problema 2, la classe `Person` *no té sobrecarregat l'operador* `<`*. Per tant la crida* `bubblesort(v2,0,9)` *és incorrecta.*