Estructures de dades lineals II

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.3

Avís: aquesta presentació no pretén ser un substitut dels apunts oficials de l'assignatura.

On som?

- Tema 2: Estructures de dades lineals.
- 4a sessió.

Avui

- Més estructures de dades lineals: cues i llistes.

El concepte de cua

- Metàfora de la cua: cua de gent.
- Operacions bàsiques: demanar tanda/torn (`push`), avançar (`pop`).
- *FIFO*: el primer en arribar és el primer en sortir (*First in, first out*)

Exemple d'evolució d'una cua I

Cua d'enters:

1. valors 1, 2 i 3 demanen tanda
2. avança
3. valors 4 i 5 demanen tanda
4. avança
5. valors 6 i 7 demanen tanda

Exemple d'evolució d'una cua II

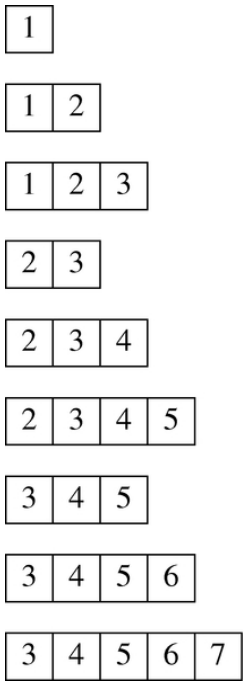


Figura: Exemple d'evolució d'una cua

Especificació de la classe Cua

- Veure especificacio_implementacio_cues.pdf.
- Instanciació: `queue<tipus> nom_cua;`.
  Exemple:

```
queue<int> c;
queue<Estudiant> q;
```

Exercici: llargada d'una cua (versió iterativa)

La cua es passa per valor (en aquest i altres exemples les piles/cues es passen per valor en versions iteratives i per ref en versions recursives per estalviar còpies.)

```
int longitud_cua_int(queue<int> c)
/* Pre: c = C */
{
...
}
/* Post: El resultat és el nombre d'elements de C */
```

Solució: llargada d'una cua (versió iterativa)

```
int longitud_cua_int(queue<int> c)
/* Pre: c = C */
{
  int ret = 0;
  while (not c.empty()) {
    ++ret;
    c.pop();
  }
  return ret;
}
/* Post: El resultat és el nombre d'elements de C */
```

Suma dels elements d'una cua: versió iterativa

```
int suma_cua_int(queue<int> c)
/* Pre: c = C */
{
  int ret = 0;
  while (not c.empty()) {
    ret += c.front();
    c.pop();
  }
  return ret;
}
/* Post: El resultat és la suma dels elements de C */
```

A la post no cal afegir que la c s'ha buidat perquè es passa per valor

Suma dels elements d'una cua: versió recursiva

```
int suma_cua_int(queue<int> &c)
/* Pre: c = C */
{
  int ret;
  if (c.empty()) ret = 0;
  else {
    ret = c.front();
    c.pop();
    ret += suma_cua_int(c);
  }
  return ret;
}
/* Post: El resultat és la suma dels elements de C */
```

Noteu pas per referència. Per què?

Cerca en una cua: versió recursiva

```
bool cerca_rec_cua_int(queue<int>& c, int x)
/* Pre: c = C */
{
  bool ret;
  if (c.empty()) ret = false;
  else if (c.front() == x) ret = true;
  else {
    c.pop();
    ret = cercar_rec_cua_int(c, x);
  }
  return ret;
}
/* Post: El resultat ens diu si x és un element de C o no */
```

Noteu pas per referència.

Cerca en una cua: versió iterativa

```
bool cerca_iter_cua_int(queue<int> c, int x)
/* Pre: c = C */
{
  bool ret = false;
  while (not c.empty() and not ret) {
    ret = (c.front() == x);
    c.pop();
  }
  return ret;
}
/* Post: El resultat ens diu si x és un element de C o no */
```

Exercici: escriure les dues versions de la cerca per cues genèriques instanciades amb la classe `Estudiant`.

Sumar k als elements d'una cua: versió acció

```
void suma_k_cua_acc(queue<int>& c, int k)
/* Pre: c = C */
{
  queue<int> c_aux;
  while (not c.empty()) {
    int aux = c.front();
    aux += k;
    c.pop();
    c_aux.push(aux);
  }
  c = c_aux;
}
/* Post: Cada element de c es la suma del valor de k i
   de l'element de C a la mateixa posició */
```

Exercici: versió recursiva. Problemes?

Implementació de cues

- Implementació amb vectors.
- Precondició d'implementació a `push`: "el paràmetre implícit no està ple" (head != (tail + 1) % elems.size())
- Idea clau: circularitat.
- Variants:
  - Vector de mida n i cua de màxim $n-1$ elements.
  - Vector de mida n i cua de màxim n elements (cal booleà per distingir cua buida de cua plena).
- Veure especificacio_implementacio_cues.pdf

Introducció a les llistes. Contenidors i iteradors

contenidor: estructura de dades on s'hi emmagatzemen objectes.
Exemple:

- `vector`, `stack`, `queue`,... de STL
- Avui: llista

iterador: classe que ens permet desplaçar-nos pel contenidor.
Assolir + llibertat de moviment que en piles i cues.

Iteradors: declaració (instanciació)

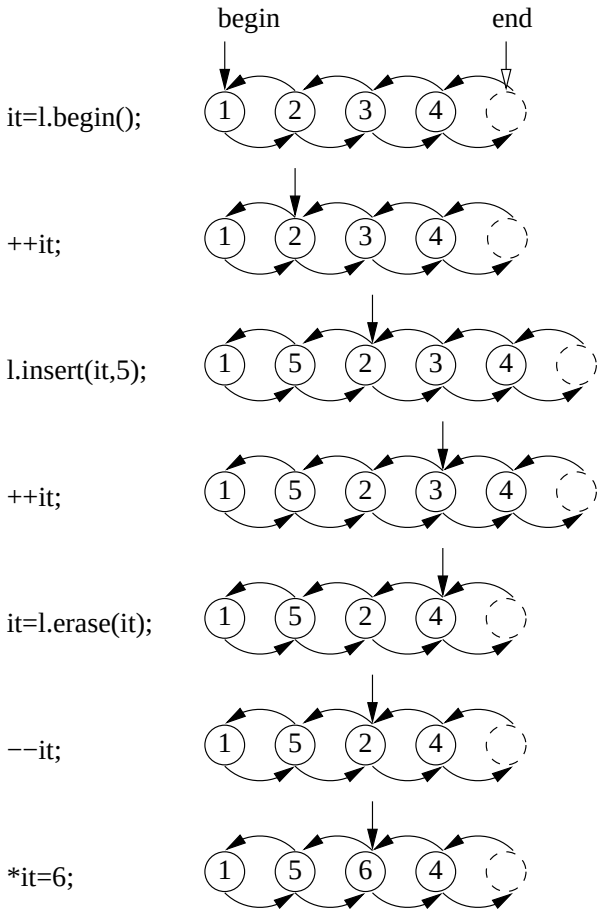
Mètode `begin`: retorna un iterador que referencia el primer element del contenidor si és que existeix. Exemple: llista d'`Estudiant`

```
list<Estudiant>::iterator it = l.begin();
```

- `it` referencia el 1er element de la llista
- `it` significat de "::". La definició de l'iterador pertany al contenidor.

Mètode `end`: retorna un iterador que referencia un element inexistent posterior al darrer element del contenidor (no desreferenciable).

Exemple d'evolució d'una llista



Iteradors: iteradors constants, desplaçament amb iteradors

- Iteradors constants: impedeixen que l'objecte referenciat per l'iterador es pugui modificar. Exemple:

  ```
  list<Estudiant>::const_iterator it;
  ```

- Desplaçament pel contenidor: `++it` (anar al següent element) i `--it` (anar a l'element anterior).
- Desreferenciar l'iterador (accés a l'objecte referenciat): notació `*`.
  - `*it`: l'objecte de tipus Estudiant referenciat per `it`
  - Consultar el dni: `(*it).consultar_DNI();`

Iteradors: operacions amb iteradors

- Assignació: `it1=it2;`
- Comparació: `it1==it2`, `it1!=it2`.
- Detecció de final de contenidor: comparar l'iterador amb el que ens movem amb l'iterador retornat pel mètode `end`. Exemple: `(it != l.end())` on `l` és una llista amb un iterador `it`.

Especificació de la classe genèrica `Llista`

Veure document especificacio_llistes.pdf

Avís: per simplicitat no inclou els mètodes `begin()` i `end()`.

Sumar tots els elements d'una llista d'enters I

- Algorisme d'exploració.
- La llista es passa per referència constant: estalviar còpia innecessària.
- Llistes enfront piles i cues: llistes permeten exploració sense modificació.
- Llista per referència constant $\rightarrow$ accés amb iteradors constants.

Sumar tots els elements d'una llista d'enters II

```
int suma_llista_int(const list<int>& l)
/* Pre: cert */
{
  int s = 0;
  for (list<int>::const_iterator it = l.begin(); it != l.end(); ++it) {
    s += *it;
  }
  return s;
}
/* Post: El resultat és la suma dels elements de l */
```

Si una llista `l` és buida, aleshores `l.begin()` = `l.end()`

Cerca senzilla en una llista d'enters

```
bool pert_llista_int(const list<int>& l, int x)
/* Pre: cert */
{
  bool b = false;
  list<int>::const_iterator it = l.begin();
  while (it != l.end() and not b) {
    if (*it == x) b = true;
    else ++it;
  }
  return b;
}
/* Post: El resultat indica si x hi és o no a l */
```

Excercici: cerca en una llista d'estudiants

```
bool pert_llista(const list<Estudiant>& l, int x)
/* Pre: cert */
{
  bool b = false;
  ...
  return b;
}
/* Post: El resultat ens indica si hi ha algun estudiant
   amb dni x a l o no */
```

Solució: cerca en una llista d'estudiants

```
bool pert_llista(const list<Estudiant>& l, int x)
/* Pre: cert */
{
  bool b = false;
  list<Estudiant>::const_iterator it = l.begin();
  b = false;
  while (it != l.end() and not b) {
    if ((*it).consultar_DNI() == x) b = true;
    else ++it;
  }
  return b;
}
/* Post: El resultat ens indica si hi ha algun estudiant
   amb dni x a l o no */
```

Modificar una llista sumant un valor k a tots els elements

```
void suma_llista_k(list<int>& l, int k)
/* Pre: cert */
{
  list<int>::iterator it = l.begin();
  while (it != l.end()){
    *it += k;
    ++it;
  }
}
/* Post: El valor de cada element del paràmetre implícit
   és el valor al paràmetre implícit original més k */
```

Solució alternativa 1: com a acció (no tan bona)

```
void suma_llista_k(list<int>& l, int k)
/* Pre: l = L */
{
  list<int>::iterator it = l.begin();
  while (it != l.end()) {
    int aux = (*it) + k;
    it = l.erase(it);
    l.insert(it, aux);
  }
}
/* Post: El valor de cada element d'l és
   el valor corresponent a L més k */
```

Solució alternativa 2: com a funció

```
list<int> suma_llista_k(const list<int>& l1, int k)
/* Pre: cert */
{
  list<int>::const_iterator it1 = l1.begin();
  list<int> l2;
  list<int>::iterator it2 = l2.begin();

  while (it1 != l1.end() {
    l2.insert(it2, *it1 + k);
    ++it1;
  }
  return l2;
}
/* Post: Cada element del resultat es la suma de k i
   l'element d'l1 a la seva mateixa posició */
```

Solució alternativa 2

Remarques:

- Per crear o modificar una llista no podem fer servir iteradors constants.
- L'iterador `it2` sempre val `l2.end()` (cada nou element a `l2` s'afegeix davant seu).