Estructures de dades lineals (amb introducció a tipus genèrics/parametritzats de dades)

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.5

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

On som?

- Tema 2: Estructures de dades lineals.
- 3a sessió.

Avui

- Tipus de dades genèrics o parametritzats (plantilles).
- Mes enllà dels vectors i matrius: piles.

Templates de C++

- Permet fer servir tipus de dades com a paràmetres.
  Exemple: instanciacions de la classe `vector<...>`

```
vector<bool> v1;
vector<double> v2;
```

- Classes contenidores genèriques permeten emmagatzemar qualsevol tipus d'element. Exemple: vector<...>
- Genèriques/parametrizades: tipus de dades apareix(en) com a paràmetre(s) en la definició d'una classe o mètode.
- Aquest tema: no implementarem estructures de dades genèriques però les farem servir.
  Veurem: `stack<...>`, `queue<...>` i `list<...>`.

Lectura i escriptura d'estructures de dades genèriques

- Format de lectura i d'escriptura depèn del tipus amb què la classe s'instancia.
- Per tant, cap mètode d'una classe genèrica pot tenir en el seu codi ni lectures, ni escriptures dels elements (encara no se sap a quina classe pertanyeran aquests elements).

Solució:

- Crear operacions fora de la classe genèrica per operacions de lectura i escriptura d'estructures que continguin elements d'una classe concreta.
- Per cada classe d'elements, caldrà implementar operacions externes (sense paràmetre implícit) de lectura i escriptura per l'estructura de dades contenidora.

Definició d'estructura de dades lineal

Els elements formen una seqüència. De forma abstracta:

$$e_1, e_2, \cdots, e_n$$

on $n \geq 0$.

- $n = 0$: estructura buida.
- $n = 1$: estructura té sol element que és al mateix temps primer i darrer , no té ni predecessor ni successor.
- $n = 2$: dos elements, el primer element, sense antecessor i que té com a successor el segon, i el segon element, sense successor i que té com a antecessor el primer.
- $n > 2$ tots els elements a partir del 2on fins al penúltim tenen antecessor i successor.

Tipus d'estructures de dades lineals

- piles: `stack<int> p;`
- cues: `queue<string> c;`
- llistes: `list<Estudiant> p;`

Diferència: com es pot accedir als seus elements.

- piles: LIFO
- cues: FIFO
- llistes: -

El concepte de pila

- Metàfora pila: pila de plats.
- Operacions bàsiques: empilar (push), desempilar (pop).
- *LIFO*: El darrer en arribar és el primer en sortir (*Last In, First Out*).

Exemple d'evolució d'una pila I

- Empilem successivament els valors 1, 2, 3, i 4
- Desempilem (1 cop)
- Empilem successivament els valors 5, 6, 7, 8 i 9.

Exemple d'evolució d'una pila II

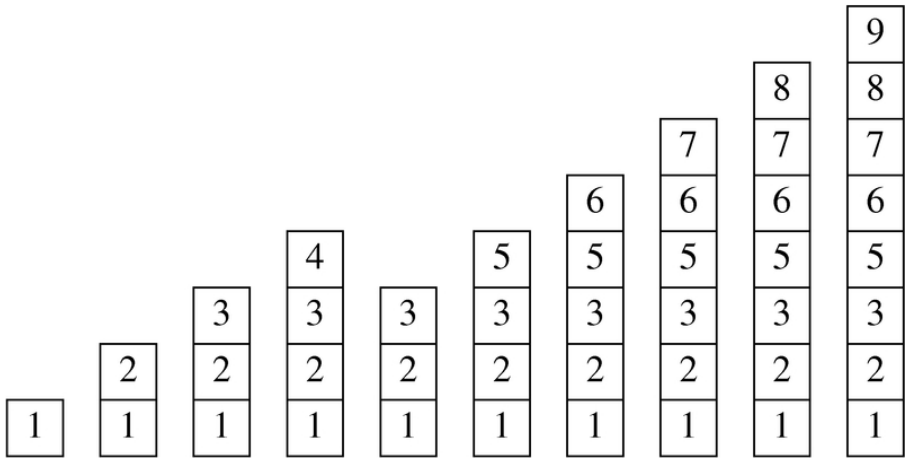

Figura: Exemple d'evolució d'una pila

Especificació de la classe stack

- Veure especificacio_implementacio_piles.pdf.
- Paraula reservada `template`
- El nom `T` fa referència al tipus dels elements que contindrà la pila, que pot ser qualsevol.
- Instanciació: `stack<tipus> nom_pila;`
  Exemples:

```
stack<int> p;
stack<Estudiant> q;
...
```

Filosofia dels exemples

- Exemples senzills d'ús de la classe `stack` (pila).
- Ometrem l'acció `main` i les inclusions.
- Tipus d'algorismes:
  - *algorismes per explorar*: desplaçament per l'estructura mitjançant consultores. No `push` en piles.
  - *algorismes per modificar*: modifiquen un nombre qualsevol d'elements de l'estructura. Sí `push` en piles.
  - *algorismes per crear o generar*: crear o generar l'estructura des del no res. Sí `push` en piles.

Alçària d'una pila

- Algorismes d'exploració on visitarem tots els elements de l'estructura.
- Calcularem propietats de l'estructura independents dels seus elements (no cal consultar els elements).
- Pila es passa per referència.
  - Evitar que es faci una còpia del paràmetre per cada crida recursiva (passat per valor).
  - Estalvi de temps i de memòria.
  - Cal fer còpia abans cridar a la funció (si volem mantenir la pila original).

Alçada d'una pila: versió recursiva

```
int alcada_pila_int(stack<int>& p)
/* Pre: p = P */
{
  int ret;
  if (p.empty()) ret=0;
  else {
    p.pop();
    ret = alcada_pila_int(p) + 1;
  }
  return ret;
}
/* Post: El resultat és el nombre d'elements de P */
```

- A la post caldria afegir que la p és una pila buida

Alçada d'una pila

Nota: el mètode pop no retorna cap valor (acció; retorn void).
Incorrecte:

```
ret = alcada_pila_int(p.pop())+1;
```

en comptes de

```
p.pop();
ret = alcada_pila_int(p)+1;
```

Suma dels elements d'una pila

Algorisme d'exploració on es visiten tots els elements de l'estructura (cal consultar el valor dels elements).

```
int suma_pila_int(stack<int> p)
/* Pre: p = P */
{
  int ret;
  if (p.empty()) ret = 0;
  else{
    int aux = p.top();
    p.pop();
    ret = suma_pila_int(p) + aux;
  }
  return ret;
}
/* Post: El resultat és la suma dels elements de P */
```

Cerca

- Algorismes de cerca: exemples d'algorismes d'exploració on no necessàriament visitarem tots els elements de l'estructura.
- PRO1:
  - *cerca* enfront *recorregut*.
  - Error greu: usar recorregut enlloc de cerca.

Cerca en una pila: versió recursiva

```
bool cerca_rec_pila_int(stack<int>& p, int x)
/* Pre: p = P */
{
  bool ret;
  if (p.empty()) ret = false;
  else if (p.top() == x) ret = true;
  else {
    p.pop();
    ret = cerca_ret_pila_int(p, x);
  }
  return ret;
}
/* Post: El resultat ens diu si x és un element de P o no */
```

Remarca: afegir a la post que p es buida; ara la pila es passa per referència (per estalviar còpies ennecessàries a cada crida recursiva)

Cerca en una pila: versió iterativa

```
bool cerca_iter_pila_int(stack<int> p, int x)
/* Pre: p = P */
{
  bool ret = false;
  while (not p.empty() and not ret){
    if (p.top() == x) ret = true;
    else p.pop();
  }
  return ret;
}
/* Post: El resultat ens diu si x és un element de P o no */
```

Remarca: ara la pila es passa per valor (per protegir el paràmetre real)

Exercici: cerca en piles instanciades amb la classe Estudiant

```
bool cerca_pila_Estudiant(stack<Estudiant>& p, int x)
/* Pre: p = P */
{
...
}
/* Post: El resultat ens diu si hi ha algun estudiant
   amb dni x a P */
```

Solució recursiva

```
bool cerca_rec_pila_Estudiant(stack<Estudiant>& p, int x)
/* Pre: p = P */
{
  bool ret;
  if (p.empty()) ret = false;
  else if (p.top().consultar_DNI() == x) ret = true;
  else {
    p.pop();
    ret = cerca_rec_pila_Estudiant(p, x);
  }
  return ret;
}
/* Post: El resultat ens diu si hi ha algun estudiant
   amb dni x a P */
```

Cal usar operacions de la classe dels elements per consultes i

Solució iterativa

```
bool cerca_iter_pila_Estudiant(stack<Estudiant> p, int x)
/* Pre: x > 0 */
{
  bool ret = false;
  while (not p.empty() and not ret){
    if (p.top().consultar_DNI() == x) ret = true;
    else p.pop();
  }
  return ret;
}
/* Post: El resultat ens diu si hi ha algun estudiant
   amb dni x a p */
```

Igualtat de piles

Igualtat de dues piles: cerca o recorregut?

Igualtat com seqüència (no de contingut)

Igualtat de piles (recursiva)

```
bool piles_iguals(stack<int>& p1, stack<int>& p2)
/* Pre: p1 = P1, p2 = P2 */
{
  bool ret;
  if (p1.empty() and p2.empty()) ret = true;
  else if (p1.empty() and not p2.empty()) ret = false;
  else if (not p1.empty() and p2.empty()) ret = false;
  else if (p1.top() != p2.top()) ret = false;
  else {
    p1.pop(); p2.pop();
    ret = piles_iguals(p1,p2);
  }
  return ret;
}
/* Post: El resultat ens indica si P1 i P2 són iguals */
```

Més adient donar una postcondició més informativa ("tots els elements de p1 son iguals a l'element de p2 que ocupa la seva mateixa posició, i viceversa").

Igualtat de piles: implementació alternativa amb calcul alçada eficient

```
bool piles_iguals(stack<int>& p1, stack<int>& p2)
/* Pre: p1 = P1, p2 = P2 */
{
  if (p1.size() == p2.size()) return piles_iguals_mateixa_alcada(p1, p2);
  else return false;
}
/* Post: El resultat ens indica si P1 i P2 són iguals */

bool piles_iguals_mateixa_alcada(stack<int>& p1, stack<int>& p2)
/* Pre: p1 = P1, p2 = P2; p1.size() == p2.size() */
  if (p1.empty()) return true;
  else if (p1.top() != p2.top()) return false;
  else {
    p1.pop(); p2.pop();
    return piles_iguals_mateixa_alcada(p1,p2);
  }
}
```

Sumar k als elements d'una pila: versió funció recursiva

Algorisme de creació o generació d'una pila.

```
stack<int> sumar_k_pila_func(stack<int>& p, int k)
/* Pre: p = P */
{
  stack<int> res;
  if (not p.empty()){
    int aux = p.top() + k;
    p.pop();
    res = sumar_k_pila_func(p, k);
    res.push(aux);
  }
  return res;
}
/* Post: Tots els elements del resultat son la suma de l'element
   de P que ocupa la seva posició, més el valor k */
```

Sumar k als elements d'una pila: versió funció - comentaris

- No cal cas base explícit donat que el resultat en aquest cas ha de ser una pila buida, que ja aconseguim al crear la pila `res`.
- Cal usar `push`.
- Exercici: pensar versió iterativa. Problemes?

Implementació de piles amb vectors I

Dues implementacions:

- `PilaInt`: Pila d'enters.
  - No parametritzada.
  - 2 fitxers: `PilaInt.hpp` i `PilaInt.cpp`
  - Compilació separada. Cal enllaçar amb el `PilaInt.o`.
- `Pila`: pila genèrica (versió simplificada de `stack` de STL).
  - Parametritzada.
  - 1 sol fitxer: `Pila.hpp`
  - No hi ha un `Pila.o`. Tot a `Pila.hpp`.

Implementació de piles II

- Implementació amb vectors.
- Precondició d'implementació a `push`: "el paràmetre implícit no està ple"(`cur < MAX_SIZE`)
- Veure especificacio_implementacio_piles.pdf