Tipus recursius de dades

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2016-2017)
Versió 0.4

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

On som?

- Tema 7: Tipus recursius de dades (1a sessió)
- 10a sessió

Avui

- Com s'implementen estructures de dades dinàmicament: piles, cues,...
- Memòria dinàmica, apuntadors,...

Recursivitat

Operació recursiva: operació que es crida a ella mateixa per aplicar el mateix tractament a tots o una part dels elements dels seus paràmetres.
El nombre de crides recursives ha de ser finit:

- els paràmetres de l'operació han de "decréixer"a cada crida recursiva
- ha d'haver-hi com a mínim un cas directe (= opció sense crida recursiva)

Filosofia recursiva per estructures complexes de dades

- Elements distribuïts en una sèrie de nodes enllaçats
- Cada node conté els següents camps:
  - Element de l'estructura
  - Enllaç, que referirà a un altre node del mateix tipus (recursivitat).
- No es permet construir estructures d'infinits elements.
- Cas directe: valor nul al camp enllaç (marca de final d'estructura).

Virtuts estructures de dades recursives

- No cal saber a priori un nombre màxim d'elements per reservar la memòria necessària per emmagatzemar l'estructura.
  - Es pot anar demanant memòria per als nous nodes a mesura que es volen afegir elements a l'estructura.
  - Memòria dinàmica.
- Podrem inserir o esborrar elements a l'estructura sense necessitat de moure els altres elements (com passaria en un vector).
  - Només caldrà modificar alguns enllaços entre nodes.

Nodes i apuntadors

Nodes:

- Tipus: `struct` de C++
- Camp enllaç: tipus "apuntador a".

Tot objecte està emmagatzemat en la memòria de l'ordinador on s'executi el programa

- Apuntador "adreça de memòria" (per exemple, adreça d'inici).
- Apuntador de tipus T: "adreça de memòria d'un objecte de tipus T".
- Apuntador que no apunta a cap objecte: valor `nullptr` (requereix opció de compilació `-std=c++11`) (`NULL` en C++ obsolet).

Constructors de tipus en C/C++: `*`, `struct`, `class`, `union`, `array`, ...

Sobre el món dels apuntadors

Tipus dels apuntadors:

- PRO2: només apuntadors de tuples (`struct`).
- En general, apuntadors de qualsevol tipus (exemple: `this` és un apuntador al paràmetre implícit)

Llenguatges de programació:

- No tots disposen del constructor apuntador.
- Llenguatges de programació amb apuntadors: no hi ha una versió única d'apuntadors.

Conceptes bàsics sobre apuntadors I

- Construcció d'un tipus apuntador (en C++): `T1*` = tipus apuntador a tipus `T1`.
- Declaració d'un objecte o variable de tipus apuntador: com qualsevol altre tipus.

  ```
  T1* x;
  ```

  Important: `x` està **indefinida** (x és un apuntador no definit).

Conceptes bàsics sobre apuntadors II

Formes de definir d'un apuntador

- Fent-lo apuntar a un objecte del tipus `T1` ja existent (assignació d'un punter o d'una adreça)
- Reservant memòria perquè apunti a un objecte nou `new`
- Donant-li el valor `nullptr`

Accés a un objecte mitjançant un apuntador:

- *x
- *this

Accés a un camp d'un objecte mitjançant un apuntador:

- x->a, (*x).a
- this->nota, (*this).nota

Exemples I

```
struct T1 {
   int camp1;
   bool camp2;
};

T1 tup; // declara i reserva memòria per un objecte tup del tipus T1
tup.camp1 = 5;     // assigna valors als camps de l'objecte tup
tup.camp2 = true;  // sense usar cap punter

T1* x;  // x és un punter (no inicialitzat) a objectes del tipus T1,
        // qualsevol consulta dóna error fins que no fem l'inicialització

x = &tup;            // fa que x apunti a l'objecte tup (li assigna la seva adreça)
(*x).camp2 = false;  // i permet modificar els camps de tup usant el punter x

x = new T1; // reserva memòria per un nou objecte del tipus T1 i fa que x apunti a aquest,
            // els valors dels camps de *x poden ser qualssevol

(*x).camp1 = 0;  // donem valors als camps de *x; també es pot fer amb x->camp1 = 0;
(*x).camp2 = false;   // i  x->camp2= false;
```

Exemples II

```
T1* y;  // y és un punter (no inicialitzat) a objectes del tipus T1

y = x;   // y apunta al mateix objecte que x (aliasing)

(*y).camp1 = 20;  // modifiquem el camp1 d'y (i també el de x)

x = nullptr;  // x ja no apunta enlloc; qualsevol consulta als camps de *x dóna error

if (x == nullptr) (*y).camp2 = true;  // un punter inicialitzat es pot comparar
                                      // amb nullptr i amb d'altres punters

delete y;   // allibera memòria de l'objecte apuntat per y
            // (també es diu que s'esborra l'objecte)
            // només es poden esborrar objectes creats prèviament amb un new
```

Reflexions

Operacions de gestió de memòria dinàmica:

- `new`: reservar memòria dinàmica per un nou objecte del tipus especificat i retornar un punter a aquest.
- `delete`: alliberar la memòria a la qual apunta el punter ("esborrar" l'objecte apuntat).

Problemes a evitar pel programador:

- No deixar memòria sense alliberar (objectes sense esborrar).
- No intentar accedir memòria alliberada (objectes esborrats).
- Aliasing (més endavant)

Assignació de punters

- `a` `i` `b` punters al tipus `T1`,
- `a` no ha estat inicialitzat o té valor `nullptr`,

La instrucció `a = b;` comporta

- `a` passa a tenir el mateix valor que `b`.
- Si `b` apunta a un objecte de `T1`, `a` passa a apuntar al mateix objecte.
- Si `a` apuntava anteriorment a un altre objecte, aquest no sofreix cap canvi intern, però podria convertir-se en inaccessible per al programa, si no és apuntat per d'altres punters.

Aliasing

- Quan un mateix objecte és apuntat per més d'un punter
- Conseqüència: anar amb cura perquè amb un dels punters perquè afecta a la resta de punters.
- Per exemple: `delete` més d'un cop sobre la mateixa adreça de memòria.
- Resultats inesperats, incorrectes,...

Assignació, còpia i esborrament

Assignació:

- Assignació de variables de tipus simples: dues variables passen a tenir el mateix valor i són independents.
- Assignació entre apuntadors: passen a tenir el mateix valor però no són independents (exemple: delete).

Còpia:

- Assignació entre apuntadors a node no es una còpia de nodes.
- Per crear un node nou a partir d'un altre ja existent, s'ha de definir una **operació de còpia** per al corresponent tipus.

Esborrament:

- `delete n`, no alliberem els possibles nodes següents de `n`.
- Cal definir una operació d'esborrament.

Pas d'apuntadors

Pas d'apuntadors (o tipus que continguin apuntadors) com a paràmetre d'entrada d'una operació.

- Pas per valor:
  - No podrem comptar amb que es faci una còpia de tota l'estructura enllaçada a partir de l'apuntador.
  - Si dins de l'operació es modifica qualsevol element d'aquesta estructura, el canvi és permanent.
- Pas per referència constant: el mateix (no garanteix que no es pugui modificar l'objecte apuntat)

Definició d'una estructura de dades recursiva I

Dos nivells:

- Superior: classe amb atributs
  - Informació global de l'estructura (que no volem que es repeteixi per a cada element)
  - Punters a alguns elements distingits (el primer, l'últim, etc., segons el que calgui).
- Inferior: definició de tupla privada (*struct*) dins de la classe on es definirà el tipus dels nodes.

Definició d'una estructura de dades recursiva II

Cada node (típicament en estructures de dades seqüencials)

- Informació sobre l'element
- Un punter al següent element de l'estructura (més d'un, si l'estructura és arborescent)
- Un punter o l'anterior si volem poder recórrer l'estructura en sentit contrari
- ...

Definició d'una estructura de dades recursiva III: esquema bàsic

```
class estructura_rec {

  private:
     struct node {
         tipus_info info;
         node* següent;  // <--- aqui hi ha la recursivitat
     };

     tipus_que_sigui info_general;
     node*  element_distingit_1;
          ...

  public:
          ...
};
```

Implementació piles

- Cada element no nul té un següent i només un.
- Un atribut amb l'altura de la pila (cal actualitzar-lo cada vegada que entri o surti un element)
  - Eficiència operacions: `size` (alçada de la pila)...
- Piles genèriques de tipus `T`.

Definició de la classe genèrica

```
template <class T> class stack {
  private:
    struct node_pila {
        T info;
        node_pila* seguent;
    };
    int altura;
    node_pila* primer_node;
    ...  // especificació i implementació d'operacions privades
  public:
    ...  // especificació i implementació d'operacions públiques
};
```

Convenis

- La pila buida:
  - `altura` = 0
  - `primer_node` = nullptr.
- Cim d'una pila no buida: 1er node (accés amb `primer_node`).
- Mètodes públics: implementació senzilla tret de constructora i destructora (recorregut de tots els nodes de la pila).
- Mètodes privats: copiar i esborrar cadenes de `node_pila`.

Mètodes públics: construcció/destrucció (introducció)

```
stack() {
  altura = 0;
  primer_node = nullptr;
}
```

Mètodes públics: modificadors

```
void push(const T& x) {
  node_pila* aux;
  aux=  new node_pila; // reserva espai per al nou element
  aux->info = x;
  aux->seguent = primer_node;
  primer_node = aux;
  ++altura;
}
```

Mètodes públics: modificadors

```
void pop() {
/* Pre: el p.i. és una pila no buida <=> primer_node != nullptr */
  node_pila* aux;
  aux= primer_node; // conserva l'accés al primer node abans
                    //   d'avançar
  primer_node = primer_node->seguent; // avança
  delete aux; // allibera l'espai de l'antic cim
  --altura;
}
```

Mètodes públics: consultors

```
T top() const {
/* Pre: el p.i. és una pila no buida <=> primer_node != nullptr */
  return primer_node->info;
}

bool empty() const {
  return primer_node == nullptr;
}
```

Novetat:

```
int size() const {
  return altura;
}
```

Mètodes públics: construcció/destrucció, modificació

```
stack(const stack& original) {
  altura = original.altura;
  primer_node = copia_node_pila(original.primer_node);
}

~stack() {
  esborra_node_pila(primer_node);
}

void clear() {
  esborra_node_pila(primer_node);
  altura = 0;
  primer_node = nullptr;
}
```

Mètodes privats I

```
static node_pila* copia_node_pila(node_pila* m) {
/* Pre: cert */
/* Post: si m és nullptr, el resultat és nullptr; en cas contrari,
   el resultat apunta al primer node d'una cadena de nodes que són còpia de
   de la cadena que té el node apuntat per m com a primer */
  node_pila* n;
  if (m == nullptr) n = nullptr;
  else {
    n = new node_pila;
    n->info = m->info;
    n->seguent = copia_node_pila(m->seguent);
  }
  return n;
}
```

Mètodes privats II

```
static void esborra_node_pila(node_pila* m) {
/* Pre: cert */
/* Post: no fa res si m és nullptr, en cas contrari, allibera espai dels
         nodes de la cadena que té el node apuntat per m com a primer */
  if (m != nullptr) {
    esborra_node_pila(m->seguent);
    delete m;
  }
}
```

Mètodes públics: redefinició operador assignació

```
stack<int> p1, p2, p3;
...
p1 = p2 = p3;
```

L'assignació en C++ és un operador (una funció d'un paràmetre explícit i un paràmetre explícit), no una instrucció

Return (*this): necessari per a encadenaments d'assignacions

```
stack& operator=(const stack& original) {
  if (this != &original) {
    altura = original.altura;
    esborra_node_pila(primer_node);
    primer_node = copia_node_pila(original.primer_node);
  }
  return *this;
}
```

Avisos importants sobre l'operador &

```
stack& operator=(const stack& original) {
  if (this != &original) {
     ...
  }
  return *this;
}
```

- Només permetem retorns de funció tipus T& (p.e. stack&) als operadors d'assignació sobrecarregats.
- Fora de la capçalera d'una funció, només permetem & per veure si dos objecte son iguals.

Implementació de cues

- Cal poder accedir tant tant al primer element (per consultar-lo o esborrar-lo) com a l'últim (per afegir un de nou).
- Atribut per la llargada (o mida) de la pila.
  - Eficiència operacions: `mida`...

Definició de la classe

```
template <class T> class queue {
  private:
    struct node_cua {
        T info;
        node_cua* seguent;
    };
    int longitud;
    node_cua* primer_node;
    node_cua* ultim_node;
    ...  // especificació i implementació d'operacions privades
  public:
    ...  // especificació i implementació d'operacions públiques
};
```

Convenis

Cua buida es definix amb

- longitud = 0
- nodes primer i darrer nuls.

Mètodes privats: copiar i esborrar cadenes I

```
static node_cua* copia_node_cua(node_cua* m, node_cua* &u) {
/* Pre: cert */
/* Post: si m és nullptr, el resultat i u són nullptr; en cas contrari,
   el resultat apunta al primer node d'una cadena de nodes que són còpia de
   de la cadena que té el node apuntat per m com a primer, i u apunta a
   l'últim node */

  node_cua* n;
  if (m == nullptr) {n = nullptr; u = nullptr;}
  else {
    n = new node_cua;
    n->info = m->info;
    n->seguent = copia_node_cua(m->seguent, u);
    if (n->seguent == nullptr) u = n;
  }
  return n;
}
```

Mètodes privats: copiar i esborrar cadenes II

```
static void esborra_node_cua(node_cua* m) {  // op. privada
/* Pre: cert */
/* Post: no fa res si m és nullptr, en cas contrari, allibera espai dels nodes
         la cadena que té el node apuntat per m com a primer */
  if (m != nullptr) {
    esborra_node_cua(m->seguent);
    delete m;
  }
}
```

Mètodes privats: construcció/destrucció

```
queue() {
  longitud = 0;
  primer_node = nullptr;
  ultim_node = nullptr;
}

queue(const queue& original) {
  longitud= original.longitud;
  primer_node= copia_node_cua(original.primer_node,
                              ultim_node);
}

~queue() {
  esborra_node_cua(primer_node);
}
```

Mètodes públics: sobrecàrrega de l'operador d'assignació

```
queue& operator=(const queue& original) {
  if (this != &original) {
    longitud= original.longitud;
    esborra_node_cua(primer_node);
    primer_node = copia_node_cua(original.primer_node, ultim_node);
  }
  return *this;
}
```

Mètodes públics: modificadors I

```
void clear() {
  esborra_node_cua(primer_node);
  longitud = 0;
  primer_node = nullptr;
  ultim_node = nullptr;
}

void push(const T& x) {
  node_cua* aux;
  aux = new node_cua; // reserva espai per al nou element
  aux->info = x;
  aux->seguent = nullptr;
  if (primer_node == nullptr) primer_node = aux;
  else ultim_node->seguent = aux;
  ultim_node = aux;
  ++longitud;
}
```

Mètodes públics: modificadors I

```
void pop() {
/* pre: el p.i. és una cua no buida <=> primer_node != nullptr */
  node_cua* aux;
  aux = primer_node; // conserva l'accés al primer node abans
                     //   d'avançar
  if (primer_node->seguent == nullptr) {
    primer_node = nullptr;  ultim_node = nullptr;
  }
  else primer_node = primer_node->seguent; // avança
  delete aux; // allibera l'espai de l'antic cim
  --longitud;
}
```

Mètodes públics: consultors

```
T front() const {
/* pre: el p.i. és una cua no buida <=> primer_node != nullptr */
  return primer_node->info;
}

bool empty() const {
  return longitud == 0;
}
```

Novetat:

```
int size() const {
  return longitud;
}
```