TEMA 1
Introducció als TAD. Els tipus lineals

PROGRAMACIÓ I ESTRUCTURES DE DADES

Introducció. Els tipus lineals

- 1. Introducció als TAD
- 2. Vectors
- 3. Llistes
- 4. Piles
- 5. Cues

1. Introducció als TAD

- TAD: tipus abstracte de dades
- *Tipus de dades*:
  - **Classifica els objectes dels programes (variables, paràmetres, constants) i determina els valors que poden prendre**
  - **També determina les operacions que s'apliquen**
    - **Enter: operacions aritmètiques enteres (suma, resta…)**
    - **Booleano: operacions lògiques (and, or…)**
- *Abstracte*:
  - **La manipulació de les dades només depèn del comportament descrit en la seua especificació (*què fa*) i és independent de la seua implementació (*com es fa*)**
  - **Una especificació → Múltiples implementacions**

1. Introducció als TAD

- Especificació d'un TAD:
  - Consisteix a establir les propietats que el definixen
  - Perquè siga útil ha de ser:
    - Precisa: nomes produïsca l'imprescindible
    - General: siga adaptable a diferents contextos
    - Llegible: siga un comunicador entre especificador i implementador
    - No ambigua: evite problemes d'interpretació.
  - Definició informal (llenguatge natural) o formal (algebraica)

1. Introducció als TAD

- Implementació d'un TAD:
  - Consisteix a determinar una representació per als valors del tipus i a codificar-ne les operacions a partir d'aquesta representació
  - Perquè siga útil ha de ser:
    - Estructurada: facilita el seu desenvolupament
    - Eficient: optimitza l'ús de recursos → Avaluació de distintes solucions per mitjà de la complexitat (espacial i temporal)
    - Llegidor: facilita la seua modificació i manteniment

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (I)

- Especificació algebraica (equacional): estableix les propietats d'un TAD per mitjà d'equacions amb variables quantificades universalment, de manera que les propietats donades es compleixen per a qualsevol valor que prenguen les variables
  Passos:
  - Identificació dels objectes del TAD i les seues operacions (declaració del TAD, mòduls que usa, paràmetres)
  - Definició de la signatura (sintaxi) d'un TAD (nom del TAD i perfil de les operacions)
  - Definició de la semàntica (significat de les operacions)
- Operació: és una funció que pren com a paràmetres (entrada) zero o més valors de diversos tipus, i produeix com a resultat un sol valor d'un altre tipus. El cas de zero paràmetres representa una constant del tipus de resultat.

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (II)

MÒDUL …

USA …

PARAMETRE TIPUS …

OPERACIONS

…

…

FPARAMETRE

TIPUS (GÈNERE) …

OPERACIONS

…

…

FMÒDUL

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (III)

MODULO NATURAL1

TIPO natural

OPERACIONES

cero : → natural

suc : natural → natural

FMODULO

Per mitjà d'aplicació successiva de *cero* i *suc* s'obtenen els distints valors del tipus:

cero, suc(cero), suc(suc(cero)), …

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (IV)

MODULO NATURAL2

TIPO natural

OPERACIONES

cero : → natural

suc : natural → natural

suma : natural natural → natural

FMODULO

suma(cero, suc(cero)) i suc(cero) denoten valors distints?

suma(cero, suc(cero)) i suma(suc(cero), cero) denoten el mateix valor?

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (V)

- Solució:
  - Utilització d'equacions de la forma $t_1 = t_2$, on t_1 i t_2 són termes sintàcticament correctes del mateix tipus
  - Semànticament, expressa que el valor construït per mitjà del terme t_1 és el mateix que el valor construït per mitjà del terme t_2
  - Per a no haver d'escriure infinites equacions, s'admet que els termes que apareixen en una equació tinguen variables

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (VI)

MODULO NATURAL3

TIPO natural

OPERACIONES

cero : → natural

suc : natural → natural

suma : natural natural → natural

VAR

x, y: natural

ECUACIONES

suma(x, cero) = x

suma(x, suc(y)) = suc(suma(x, y))

FMODULO

1. Introducció als TAD

EXERCICIS

- Siga el conjunt dels nombres *naturals* amb les operacions *cero* i *suc*. Defineix la sintaxi i la semàntica de les operacions “==“ i “<=“, que permeten fer una ordenació dels elements del conjunt.

1. Introducció a ls TADs

EXERCICIS

- Completa en aquesta fulla les equacions que apareixen a continuació i que expresen el comportament de les operacions de: *resta* en el conjunt dels nombres Naturals en el quual només existeixen les operacions *cero:* $\rightarrow$ *natural* i l'operació *suc: natural* $\rightarrow$ *natural* (torna el succesor d'un nombre Natural). S'asumeix que el primer operand de la *resta* és sempre major o igual que el segon.

resta(natural,natural) $\rightarrow$ *natural*

resta(,) = ...

resta(,) = ...

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (VII)

- Com podem estar segurs que no són necessàries més equacions?
- Propietats importants: *consistència* i *completitut*
  - Si es posen equacions de més, es poden igualar termes que estan en classes d'equivalència diferents, mentres que si se'n posen de menys, es pot generar un nombre indeterminat de termes incongruents amb els representants de les classes existents.

1. Introducció als TAD
ESPECIFICACIÓ ALGEBRAICA (VIII)

- Classificació de les operacions:
  - Constructores: tornen un valor del tipus
    - Generadores: permeten generar, per aplicacions successives, tots els valors del TAD a especificar
    - Modificadores: la resta
  - Consultores: retornen un valor d'un tipus diferent.
- En general, les operacions modificadores i consultores s'especifiquen en termes de les generadores. A vegades, una operació modificadora pot especificar-se en termes d'altres modificadores o consultores. Direm que es tracta d'una operació *derivada*.

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (IX)

- Equació condicional: és equivalent a un conjunt finit d'equacions no condicionals

si (n1 <> n2) entonces

    saca(añade(s, n1), n2) = añade(saca(s, n2), n1)

sino

    saca(añade(s, n1), n2) = saca(s, n2)

fsi

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (X)

- Operacions auxiliars: s'introdueixen en una especificació per a facilitar la seua escriptura i llegibilitat. Són invisibles per als usuaris del TAD (també se‘n diu ocultes o privades).

1. Introducció als TAD

ESPECIFICACIÓ ALGEBRAICA (XI)

- Tractament d'errors: pot ocórrer que alguna operació siga una funció parcial (no es pot aplicar sobre certs valors del domini de les dades)

MÒDUL NATURAL4
TIPUS natural
OPERACIONS
cero : → natural
suc, pred : natural → natural
suma, mult : natural natural → natural
VAR x, y: natural;
ECUACIONS
suma(cero, x) = x
suma(x, cero) = x
suma(x, suc(y)) = suc(suma(x, y))
mult(cero, x) = cero
mult(suc(y), x) = suma(mult(y, x), x)
pred(suc(x)) = x
FMÒDUL

Quant val pred(cero)?

1. Introducció als TAD
ESPECIFICACIÓ ALGEBRAICA (XII)

- Tractament d'errors:
  - S'afig una constant a la signatura que modelitza un valor d'error: $error_{nat}$ → *natural*
  - S'afig una equació que completa l'especificació de pred: *pred(cero)* = $error_{nat}$
  - Se suposarà que els valors sobre els quals s'aplica una operació en una equació normal estan lliures d'error

1. Introducció als TAD

IMPLEMENTACIÓ (I)

- Donada una especificació d'un tipus, es poden construir diverses implementacions
- Cada implementació es defineix en un mòdul diferent, anomenat mòdul d'implementació
- La construcció d'aquests mòduls consta de dues fases:
  - Elecció d'una representació per als diferents tipus definits en l'especificació
  - Codificació de les operacions en termes de la representació triada

1. Introducció als TAD
IMPLEMENTACIÓ (II)

- Mecanismes d'abstracció en els llenguatges de programació:
  - Encapsulament de la representació del TAD
  - Ocultació de la informació, per a limitar les operacions possibles sobre el TAD
  - Genericitat, per a aconseguir implementacions genèriques vàlides per a distints tipus
  - Herència, per a reutilitzar implementacions
- Els llenguatges de programació tradicionals (Fortran, Basic, Pascal, C) resulten ineficients per a utilitzar els mecanismes d'abstracció
- És necessari utilitzar llenguatges moderns (ADA, C++, Java, C#)

1. Introducció als TADs

Preguntes de tipus test: Vertader vs. Fals

- EsVacia: PILA → BOOLEAN. Si P i Q són piles: Q = EsVacia (P), és un ús sintàcticament correcte de l'operació
- A l'especificació d'un TAD, una operació consultora torna un valor del tipus definit
- Siga el següent TAD:

MÒDUL NATURALEXAMEN

TIPOUSnatural

OPERACIONS

uno: → natural; següent: natural → natural

sumar: natural natural → natural

FMÒDUL

Si *N* és un *natural*: *N = sumar(uno,següent(uno))* és un ús sintàcticament incorrecte de l'operació *sumar*

2. Vectors

- Un vector és un conjunt ordenat de parells <índex, valor>. Per a cada índex definit dins d'un rang finit hi ha associat un valor. En termes matemàtics, és una correspondència entre els elements d'un conjunt d'índexs i els d'un conjunt de valors

2. Vectors

ESPECIFICACIÓ ALGEBRAICA

MÒDUL VECTOR USA BOOL, ENTERO
//en totes les equacions, c ≤ i, j ≤ f
PARAMETRE TIPUS item
OPERACIONS
c, f: → int //límites inf. y sup.
error( ) → item
FPARAMETRE

TIPUS vector

OPERACIONS
crear( ) → vector
asig(vector, int, item) → vector
recu(vector, int) → item
esvaciapos(vector, int) → bool

VAR v: vector; i, j: int; x, y: item;
ECUACIONS
si (i < > j) **entonces**
asig(asig(v, i, x), j, y) = asig(asig(v, j, y), i, x)
si no asig(asig(v, i, x), j, y) = asig(v, i, y) **fsi**

recu(crear( ), i) = error( )
recu(asig(v, i, x), j)
si (i == j) **entonces** x
si no recu(v, j) **fsi**

esvaciapos(crear( ), i) = CIERTO
esvaciapos(asig(v, i, x), j)
si (i == j) **entonces** FALSO
si no esvaciapos(v, j) **fsi**

FMÒDUL

2. Vectors

REPRESENTACIÓ DE VECTORS

```
//Vector d'item

const int kTam = 10;
class TVector {
      friend ostream& operator << ( ostream&, TVector&);

public:
TVector( );
TVector( const TVector &v );
~TVector( );
TVector& operator =( TVector &v );

TItem & Recu( int i );
void Asig( int i, TItem c );
bool Esvaciapos( int i );

private:
TItem  fv[ kTam ];     //grandària fixa
// TItem *fv;     grandària dinàmica
int fLong;
};
```

Canvi d'*asig* i *recu* per la sobrecarrega de l'operador claudàtor:
TItem & operator [] (int i);

```
TItem&
TVector::operator[] (int indice){
  if (indice>=1 && indice<=fLong)
     return (fv[indice-1]);
  else
     return (error); }
```

2. Vectors

EXERCICIS *eliminar*

- Siga un vector de nombres naturals. Utilitzant exclusivament les operacions *asignar* i *crear*, defineix la sintaxi i la semàntica de l'operació *eliminar,* que esborra les posicions pars del vector marcant-les amb "-1" (per a calcular el residu d'una divisió, es pot utilitzar l'operació MOD)

2. Vectors

EXERCICIS *operació M*

- Donada la sintaxi i la semàntica de l'operació M que actua sobre un vector:

M (vector) → vector

Var v: vector; i: int; x: item;

M(crear()) = crear()

si i == 1 entonces

M(asig(v, i, x)) = M(v)

si no M(asig(v, i, x)) = asig(M(v), i –1, x)

a) Aplicar l'operació M al vector següent:

asig(asig(asig(crear(), 3, a), 1, b), 2, c)

b) Explicar en un paràgraf què és el que fa l'operació M

2. Vectors

EXERCICIS palíndrom

- Utilitzant les operacions definides a classe per a la definició del tipus vector definir la sintaxi i la semàntica de l'operació *palindrom* que indica si un vector de naturals amb 100 elements és palíndrom. Per exemple, el vector 1,25,12,3,3,12,25,1 és palíndrom (s'ha simplificat l'exemple amb un vector de 8 elements). IMPORTANT: s'asumeix que el vector està creat amb tamany 100, està ple i el rang de les posicions és de 1 a 100 (en aquest ordre).

2. Vectors

Preguntes de tipus test: Vertader vs. Fals

- El tipus de dades vector es defineix com un conjunt en el qual els seus components ocupen posicions consecutives de memòria

- Un vector és un conjunt ordenat de pars <índex, valor>

3. Llistes

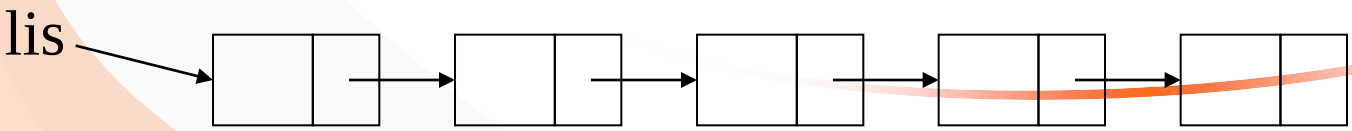
- Una llista és una seqüència de zero o més elements d'un mateix tipus de la forma $e_1, e_2, ..., e_n \qquad \forall n \geq 0$
- De forma més general: $e_p, e_{sig(p)}, ..., e_{sig(sig ... n) ... (p))}$

  Al valor *n* se l'anomena *longitud de la llista*. Si *n* = 0 tenim una *llista buida*. A e_1 se l'anomena *primer element*, i a e_n , *últim element*
- Propietats:
  - S'estableix un ordre seqüencial estricte sobre els seus elements per la *posició* que ocupen. D'aquesta manera e_i precedeix $e_{sig(i)}$ per a i = 1, 2, ... , n-1 i $e_{sig(i)}$ succeeix e_i per a i = 1, 2, ... , n-1. Finalment, l'element e_i ocupa la posició i
  - La llista ens permet conèixer qualsevol element que conté *accedint a la seua posició*, cosa que no podrem fer amb les piles i amb les cues. Utilitzarem el concepte generalitzat de posició, amb una ordenació definida sobre si mateixa; per tant no té per què correspondre's exactament amb els nombres enters, com clàssicament s'ha interpretat aquest concepte
- Una *llista ordenada* és un tipus especial de llista en què s'estableix una relació d'ordre definida entre els items de la llista

3. Llistes

REPRESENTACIÓ DE LLISTES

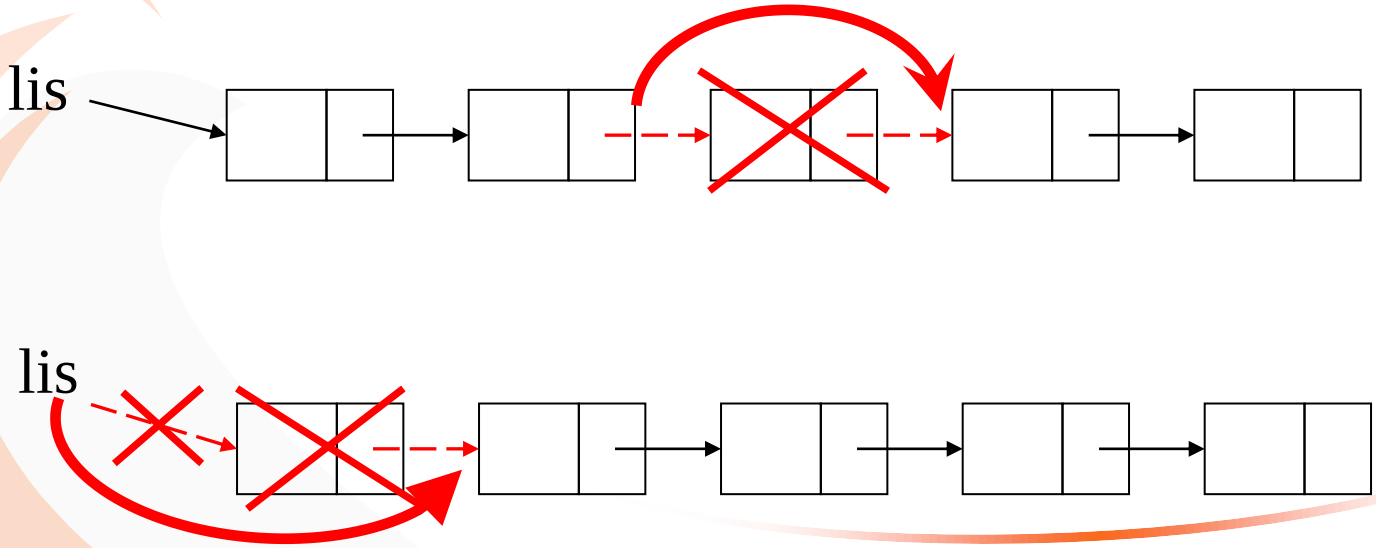
- El TAD *llista* s'utilitza per a emmagatzemar llistes d'un nombre **variable** d'objectes.
- Representació seqüencial (internament un array)
  - **A partir de tipus base ("arrays")**
  - **A partir de tipus definits per l'usuari ("tvector" –herència o layering –)**
- Representació enllaçada (internament punters a node)
  - **A partir de tipus base ("punters a node")**
  - Cada objecte s'emmagatzema en un node, que s'enllaça amb el següent.
  - La llista és un punter al primer node, o NULL si està buida.
  - Un node és un contenidor per a emmagatzemar informació, i té dues parts:
    - La informació de l'objecte que es desitja guardar.
    - Un o més punters per a enllaçar el node amb altres nodes i construir l'estructura de dades.



3. Llistes

REPRESENTACIÓ ENLLAÇADA DE LLISTES (I)

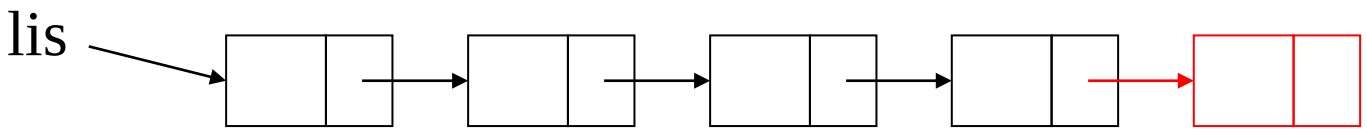
- Operacions Bàsiques sobre les llistes:
  - Esbrinar si la llista està buida
  - Cerca d'un element
  - Inserció i Esborrat d'un element: cal distingir si és al principi, en una posició intermèdia o al final de la llista
    - ESBORRAT D'UN ELEMENT INTERMEDI O FINAL



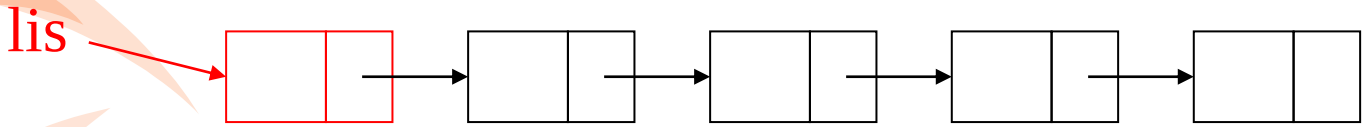
3. Llistes

REPRESENTACIÓ ENLLAÇADA DE LLISTES (II)

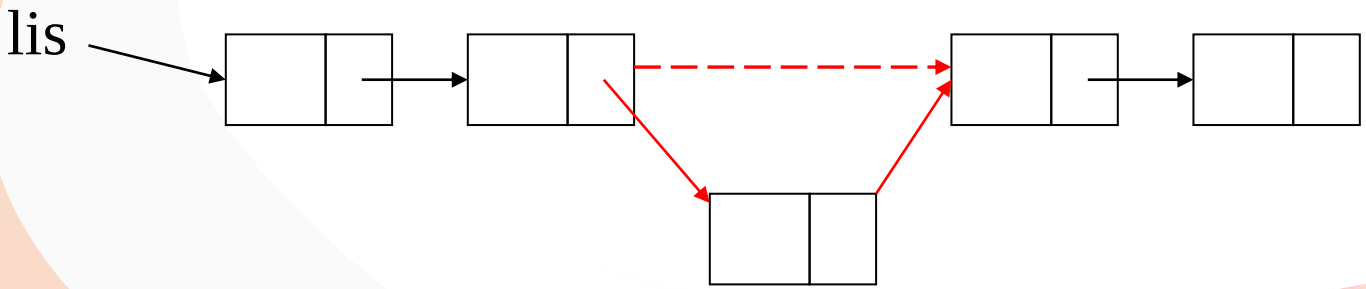
- INSERCIÓ D'UN ELEMENT AL FINAL



- INSERCIÓ D'UN ELEMENT AL PRINCIPI



- INSERCIÓ EN UNA POSICIÓ INTERMÈDIA



3. Llistes

LLISTES ORDENADES

- Una llista ordenada és una llista que en tot moment es manté ordenada.
- S'aconsegueix insertant sempre els nodes en la posició que els corresponga segons l'ordre definit (els esborrats no desordenen la llista).
- Avantatges enfront de la llista simple:
  - El temps mitjà de cerca es redueix (no és necessari recórrer tota la llista per a comprovar que un node no està).
  - No és necessari ordenar la llista.

3. Llistes

ESPECIFICACIÓ ALGEBRAICA (I)

MÒDUL LLISTA USA BOOL, NATURAL
PARAMETRE TIPUS item, posicio
OPERACIONS
eq(posicio, posicio) → bool
error_item( ) → item
error_posicio( ) → posicio
FPARAMETRE
TIPUS llista
OPERACIONS
crear( ) → llista
inscap(llista, item) → llista
esbuida(llista) → bool
concatenar(llista, llista) → llista
longitud(llista) → natural
primera, ultima(llista) → posicio
anterior, seguent(llista, posicion) → posicio
insertar(llista, posicio, item) → llista
esborrar(llista, posicio) → llista
obtindre(llista, posicio) → item

3. Llistes

ESPECIFICACIÓ ALGEBRAICA (II)

VAR L_1, L_2: lista; x: item; p: posicion;

ECUACIONS

esbuida(crear( )) = CERT

esbuida(inscabeza(L_1, x)) = FALS

concatenar(crear( ), L_1) = L_1

concatenar(L_1, crear( )) = L_1

concatenar(inscap(L_1, x), L_2) = inscap(concatenar(L_1, L_2), x)

longitud(crear( )) = 0

longitud(inscap(L_1, x)) = 1 + longitud(L_1)

primera(crear( )) = error_posicio( ); ultima(crear( )) = error_posicio( )

si esbuida(L_1) **entonces**

ultima(inscap(L_1, x) = primera(inscap(L_1, x))

si no ultima(inscap(L_1, x) = ultima(L_1)

anterior(L_1, primera(L_1)) = error_posicio( ); seguent(L_1, ultima(L_1)) = error_posicio( )

si p != ultima(L_1) **entonces** anterior(L_1, seguent(L_1, p)) = p

anterior(inscap(L_1, x), primera(L_1)) = primera(inscap(L_1, x))

3. Llistes

ESPECIFICACIÓ ALGEBRAICA (III)

si p != primera(L_1) **entonces** siguiente(L_1, anterior(L_1, p)) = p
siguiente(inscabeza(L_1, x), primera(inscabeza(L_1, x)) = primera(L_1)

insertar(crear( ), p, x) = crear( )
si p == primera(inscabeza(L_1, x)) **entonces**
    insertar(inscabeza(L_1, x), p, y) = inscabeza(inscabeza(L_1, y), x)
si no insertar(inscabeza(L_1, x), p, y) = inscabeza(insertar(L_1, p, y), x)

borrar(crear( ), p) = crear( )
si p == primera(inscabeza(L_1, x)) **entonces**
    borrar(inscabeza(L_1, x), p) = L_1
si no borrar(inscabeza(L_1, x), p) = inscabeza(borrar(L_1, p), x)

obtener(crear( ), p) = error_item( )
si p == primera(inscabeza(L_1, x)) **entonces**
    obtener(inscabeza(L_1, x), p) = x
si no obtener(inscabeza(L_1, x), p) = obtener(L_1, p)

FMODUL

3. Llistes

ENRIQUIMENT DE LES LLISTES

OPERACIONS

sublista(lista, posicion, natural) → lista

inversa (lista) → lista

VAR L: lista; x, y: item; n: natural; p: posicion;

ECUACIONS

sublista(L, p, 0) = crear( )

sublista(crear( ), p, n) = crear( )

si p == primera(inscabeza(L, x)) **entonces**

sublista(inscabeza(L, x), p, n) = inscabeza(sublista(L, primera(L), n - 1), x)

si no sublista(inscabeza(L, x), p, n) = sublista(L, p, n)

inversa(crear( )) = crear ( )

inversa(inscabeza(crear( ), x)) = inscabeza(crear( ), x)

inversa(inscabeza(L, x)) = insertar(inversa(L), ultima(inversa(L)), x)

3. Llistes

REPRESENTACIÓ DE LLISTES (I)

```
class TLista {
  friend ostream&
       operator<<(ostream&, TLista&);
  friend class TPosicion;
  public:
    TLista();
    ~TLista();
    void InsCabeza(int);
    TPosicion Primera();
    int& Obtener(TPosicion&);
    void Borrar(TPosicion&);
  private:
    TNodo *lis;
};
```

```
class TNodo {
  friend class TLista; friend class TPosicion;
  public:
    TNodo(); ~TNodo();
  private:
    int dato;   TNodo *sig; };

class TPosicion {
 friend class TLista;
  public:
    TPosicion(); ~TPosicion();
    bool EsVacia();
    TPosicion Siguiente();
    TPosicion& operator=(TPosicion&);
  private:
    TNodo* pos; };
```

3. Llistes

REPRESENTACIÓ DE LLISTES (II)

```
TLista::TLista() {  lis = NULL; }

TLista::~TLista() {
  TPosicion p, q;
  q = Primera();
  while(!q.EsVacia()) {
    p = q;
    q = q.Siguiente();
    delete p.pos;
  }
  lis = NULL;
}
```

```
void
TLista::InsCabeza(int i) {
  TNodo* aux = new TNodo;
  aux->dato = i;
  if(lis == NULL) {
    aux->sig = NULL;
    lis = aux;
  }
  else {
    aux->sig = lis;
    lis = aux;
  }
}
```

3. Llistes

REPRESENTACIÓ DE LLISTES (III)

```
TPosicion
TLista::Primera() {
  TPosicion p;  p.pos = lis;
  return p;}

int&
TLista::Obtener(TPosicion& p) {
  return p.pos->dato;}

ostream&
operator<<(ostream& os, TLista& l) {
  TPosicion p;
  p = l.Primera();
  while(!p.EsVacia()) {
    os << l.Obtener(p) << ' ';
    p = p.Siguiente();}
  return os; }
```

```
TNodo::TNodo() {
  dato = 0;  sig = NULL; }

TNodo::~TNodo() {
  dato = 0;  sig = NULL; }

TPosicion::TPosicion() {  pos = NULL; }

TPosicion::~TPosicion() {
  pos = NULL; }

bool
TPosicion::EsVacia() {
  return pos == NULL; }
```

3. Llistes

REPRESENTACIÓ DE LLISTES (IV)

```
TPosicion
TPosicion::Siguiente() {
  TPosicion p;
  p.pos = pos->sig;
  return p;
// ¿si pos es NULL?
}

TPosicion&
TPosicion::operator=(TPosicion& p) {
  pos = p.pos;
  return *this;
}
```

```
int
main(void)
{
  TLista l;
  l.InsCabeza(1); l.InsCabeza(3);
  l.InsCabeza(5); l.InsCabeza(7);
  cout << l << endl;
  TPosicion p;
  p = l.Primera();
  cout << "Primer element: "
       << l.Obtener(p) << endl;
  p = p.Siguiente();
  cout << "Segon element: "
       << l.Obtener(p) << endl;
}
```

Vectors i llistes

Aplicacions reals

- Classificació dels equips de la lliga de futbol.

- Gestió de la llist d'espera d'un hospital per a intervencions quirúrgiques.

Clasificación 1ª división

	EQUIPO	PTOS	PJ	PG	PE	PP	GF	GC
1	Barcelona	50	19	16	2	1	53	12
2	At. Madrid	50	19	16	2	1	47	11
3	Real Madrid	47	19	15	2	2	53	21
4	Ath.Bilbao	36	19	11	3	5	32	24
5	Villarreal	34	19	10	4	5	37	21
6	Real Sociedad	32	19	9	5	5	36	28
7	Sevilla FC	30	19	8	6	5	36	30
8	Valencia	23	19	7	2	10	26	31
9	Granada	23	19	7	2	10	19	25
10	Levante	23	19	6	5	8	18	27
11	Getafe	23	19	7	2	10	20	31
12	Espanyol	22	19	6	4	9	22	25
13	Osasuna	21	19	6	3	10	17	29
14	Málaga	20	19	5	5	9	19	24
15	Celta de Vigo	19	19	5	4	10	23	31
16	Almería	19	19	5	4	10	21	38
17	Elche	18	19	4	6	9	17	28
18	Valladolid	16	19	3	7	9	21	33
19	Rayo	16	19	5	1	13	19	45
20	Real Betis	11	19	2	5	12	16	38

3. Llistes

EXERCICIS *borraultimo*

- Completa les equacions que apareixen a continuació i que expressen el comportament de les operacions de *borraultimo* (borra l'últim element de la llista) en una llista d'accés per posició:

La sintaxi de l'operació és la següent:

borraultimo(lista) → lista

borraultimo(crear()) =.....

si esvacia(l1) **entonces** borraultimo(inscabeza(l1,x)) =.....

si no borraultimo(inscabeza(l1,x)) = ...

On: x ∈ element l1 ∈ llista

3. Llistes

EXERCICIS *quita_pares*

- Definir la sintaxi i la semàntica de l'operació *quita_pares* que actua sobre una llista i torna la llista original en què s'han eliminat els elements que ocupen les posicions parelles

3. Llistes

EXERCICIS *operació X*

- Explicar què fa l'operació X, la sintaxi i semàntica de la qual apareixen a continuació:

  X (lista) --> lista

  X (crear ( )) --> crear ( )

  X (inscabeza (l, i)) <==>

  si (longitud (l) == 0) **entonces** crear ( )

  si no inscabeza (X (l), i)

  On: l ∈ lista, i ∈ item

  - Simplificar la seguent expressió: (IC = inscabeza)

    X (IC (IC (IC (IC (crear ( ), a), b) ,c) ,d))

3. Llistes

Preguntes de tipus test. Vertader vs Fals

- L'operació BorrarItem té la següent sintaxi i semàntica:

  BorrarItem: LISTA, ITEM -> LISTA

  BorrarItem(Crear, i) = Crear

  BorrarItem(IC(L1,j), i) = si (i == j) entonces L1

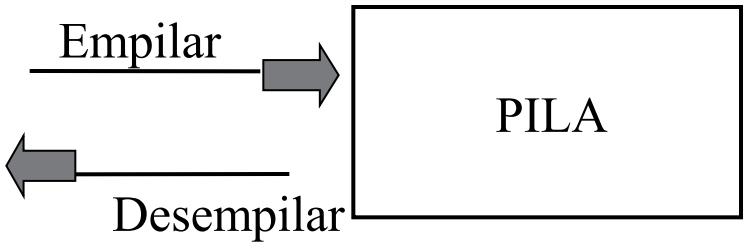
  sino IC (BorrarItem (L1, i), j)

  Aquesta operació borra totes les ocurrències de l'item que es troba en la llista

- La complexitat temporal d'obtenir un element en un vector ordenat mitjançant cerca binària o en una llista ordenada és la mateixa

4. Piles

- Una pila és una llista en la qual totes les insercions i esborrats es fan en un únic extrem, anomenat *cim*. Sabem per tant que l'últim element inserit en la pila serà el primer a ser-ne esborrat, i per això també se'ls anomena llistes "LIFO" (Last In, First Out). També podem conèixer quin és l'element que es troba en la *cima*



4. Piles

ESPECIFICACIÓ ALGEBRAICA

MÒDUL PILA USA BOOL

PARAMETRE

TIPUS item

OPERACIONS

error( ) → item

FPARAMETRE

TIPUS pila

OPERACIONS

crear( ) → pila

apilar(pila, item) → pila

desapilar(pila) → pila

cima(pila) → item

esvacia(pila) → bool

VAR p: pila, e: item;

ECUACIONS

desapilar(crear( )) = crear( )

desapilar(apilar(p, e)) = p

cima(crear( )) = error( )

cima(apilar(p, e)) = e

esvacia(crear( )) = VERITAT

esvacia(apilar(p, e)) = FALS

FMÒDUL

4. Piles

REPRESENTACIÓ SEQÜENCIAL DE PILES (I)

- Representació seqüencial (internament un *array*)
  - **A partir de tipus base (“arrays”)**
  - **A partir de tipus definits per l’usuari (“tvector” – herència o layering)**

- Tipus d’algoritmes
  - **Realitzant les insercions per la primera component. Ineficient**
  - **Utilitzant un cursor que indique la posició actual del primer element de la pila**

- Avantatges i desavantatges
  - **Desavantatge: grandària màxima de la pila**
  - **Avantatge: senzillesa d’implementació**

4. Piles

REPRESENTACIÓ SEQÜENCIAL DE PILES (II)

```
const kMax = 10;
class TPila {
    public:
        TPila( ); TPila( TPila & ); ~TPila( ); TPila& operator=( TPila &);
        TItem& Cima( );
        void Apilar( TItem& );
        . . .
    private:
        TItem fpila[ kMax ];   //grandària fixa
        // TItem *fpila;   grandària dinámica
        int ftope;
    };
TPila::TPila( ) {
    fpila = new TItem[ 10 ]; //només si el vector es dinàmic
    ftope = 0;   }
void
TPila::Desapilar( ) {
        ftope --;   }
```

4. Piles

REPRESENTACIÓ ENLLAÇADA DE PILES (I)

- Representació enllaçada (internament, *punters a node*)
  - **A partir de tipus base (“punters a node”)**
  - **A partir de tipus definits per l’usuari (“tlista” –herència o layering–)**

- Avantatges
  - **Avantatges: no hi ha definida una grandària per a la pila**

4. Piles

REPRESENTACIÓ ENLLAÇADA DE PILES (II)

```
class TPila {
    public:
        TPila( ); TPila( TPila & ); ~TPila( ); TPila& operator=( TPila &);
        TItem& Cima( );
        void Apilar( TItem& );
        . . .
    private:
        struct TNodo {
                TItem dato;
                TNodo *sig;   };
        TNodo *fp;
    };
TPila::TPila( ) {   fpila = NULL;   }
void
TPila::Desapilar( ) {
        TNodo *aux;
        aux = fp;
        fp = fp -> sig;
        delete aux;   }
```

4. Piles

REPRESENTACIÓ ENLLAÇADA DE PILES (III)

```
//HERENCIA PRIVADA
class TPila: private TLista {
  public:
    TPila( ); TPila( TPila & ); ~TPila( );
    void Apilar( TItem& );
    void Desapilar( );
            . . .
};

TPila::TPila( ): TLista( ) {   };

TPila::TPila( TPila &p ): TLista( p ) {   };

~TPila( ) {   }
void
TPila::Apilar( TItem &a ) { InsCabeza( a ); }

void
TPila::Desapilar( ) { Borrar( Primera( ) ); }
```

```
//LAYERING O COMPOSICIÓ
class TPila {
  public:
    TPila( ); TPila( TPila & ); ~TPila( );
    void Apilar( TItem& ); void Desapilar( );
            . . .
    private: TLista L;
};

TPila::TPila( ): L( ) {   };

TPila::TPila( TPila &p ): L( p.L ) {   };

~TPila( ) {   }
void
TPila::Apilar( TItem &a ) { L.InsCabeza( a ); }

void
TPila::Desapilar( ) { L.Borrar( L.Primera( ) ); }
```

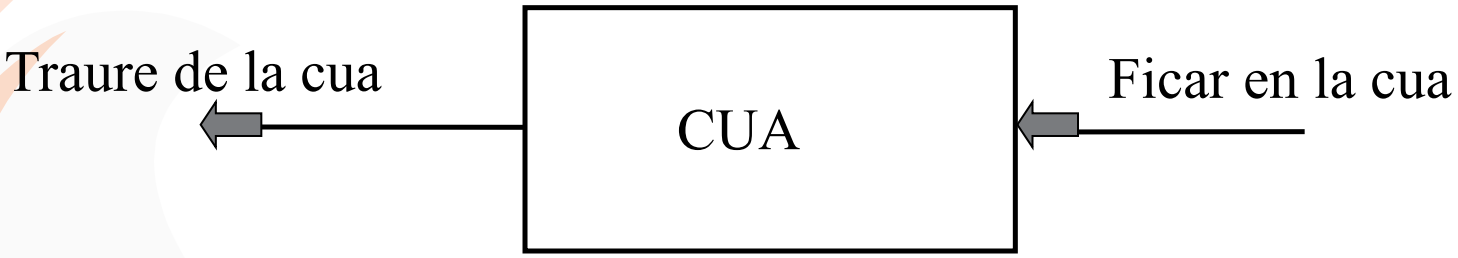
4. Piles

EXERCICIS

- Donar la sintaxi i la semántica de l'operació **base**, la qual actua sobre una pila i retorna la base de la pila (el primer element que s'ha empilat)

5. Cues

- Una cua és un altre tipus especial de llista en el qual els elements s'insereixen per un extrem (*fons*) i se suprimeixen per l'altre (cim). Les cues es coneixen també com a llistes "FIFO" (First Input First Out). Les operacions definides sobre una cua són semblants a les definides per a les piles, amb l'excepció del mode en que s'extrauen els elements



5. Cues

ESPECIFICACIÓ ALGEBRAICA

MÒDUL CUA USA BOOL

PARAMETRE

TIPUS item

OPERACIONS

error( ) → item

FPARAMETRE

TIPUS cua

OPERACIONS

crear( ) → cola

encolar(cola, item) → cola

desencolar(cola) → cola

cabeza(cola) → item

esvacia(cola) → bool

VAR c: cola, x: item;

ECUACIONS

desencolar(crear( )) = crear( )

si esvacia(c) **entonces**

desencolar(encolar(c, x)) = crear ( )

si no desencolar(encolar(c, x)) =

encolar(desencolar (c), x)

cabeza(crear( )) = error( )

si esvacia(c) **entonces**

cabeza(encolar(c, x)) = x

si no cabeza(encolar(c, x)) = cabeza(c)

esvacia(crear( )) = CIERTO

esvacia(encolar(c, x)) = FALSO

FMÒDUL

5. Cues

ENRIQUIMENT DE LES CUES

OPERACIONS

concatena(cua, cua) → cua

VAR c, q: cola; x: item;

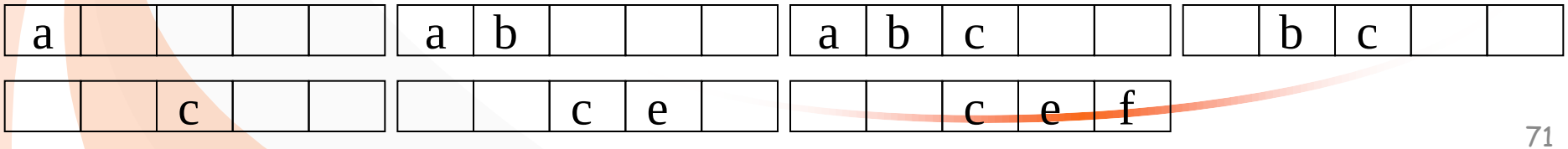
ECUACIONS

concatena(c, crear( )) = c
concatena(crear( ), c) = c
concatena(c, encolar(q, x)) = encolar(concatena(c, q), x)

5. Cues

REPRESENTACIÓ SEQÜENCIAL DE CUES (I)

- Representació seqüencial (internament un *vector*)
  - **A partir de tipus base ("vector")**
  - **A partir de tipus definits per l'usuari ("tvector" –herència o layering–)**
- Tipus d'algoritmes
  - **Utilitzant un vector (*fv*) per a emmagatzemar els elements i dos enters (cim i *fons*) per a indicar la posició d'ambdós extrems**
    - **Inicialitzar: cim = 0; fons = -1;**
    - **Condició de cua buida: fons < cim**
    - **Inserció: fons++; fv[fons] = x;**
    - **Esborrat: cim ++;**



5. Cues

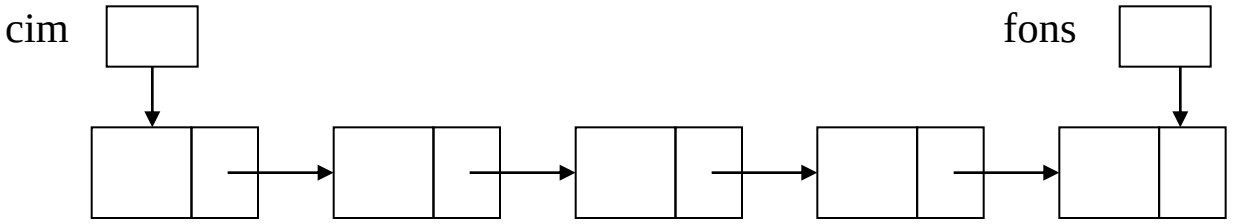
REPRESENTACIÓ SEQÜENCIAL DE CUES (II)

- Problema:
  - Hi ha buits, però no puc inserir
- Solucions:
  - Cada vegada que s'esborra un element, la resta es desplaça una posició a l'esquerra perquè cim sempre estiga en la primera posició. Quins problemes presenta aquesta solució? Augmentem la complexitat de l'operació *traure de cua*
  - Cues circulars. Vector com un cercle en què la primera posició segueix a l'última. Condició de cua buida cim == *fons*
- Avantatges i desavantatges
  - Desavantatge: grandària màxima de la cua
  - Avantatge: senzillesa d'implementació

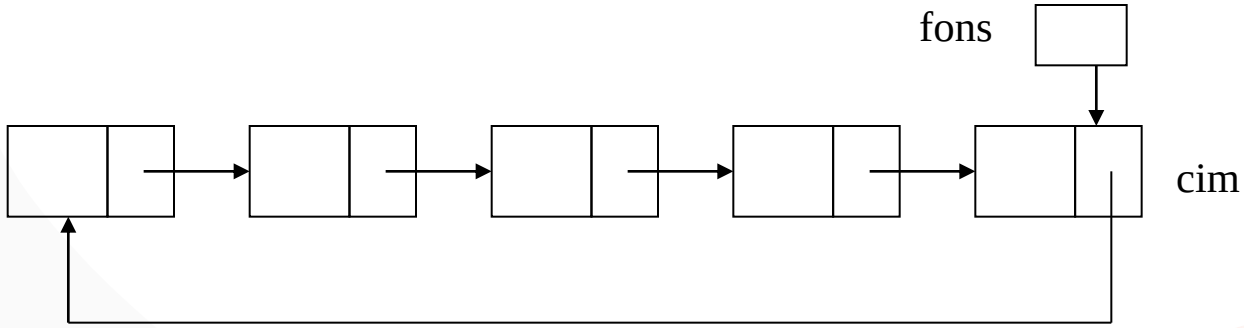
5. Cues

REPRESENTACIÓ ENLLAÇADA DE CUES (I)

- Representació enllaçada (internament, *punters a node*)
  - **A partir de tipus base ("punters a nodo")**



- **Cues circulars enllaçades, en les quals només és necessari un punter. El següent element apuntat per *fons* és el primer a traure de la cola**



5. Cues

REPRESENTACIÓ ENLLAÇADA DE CUES (II)

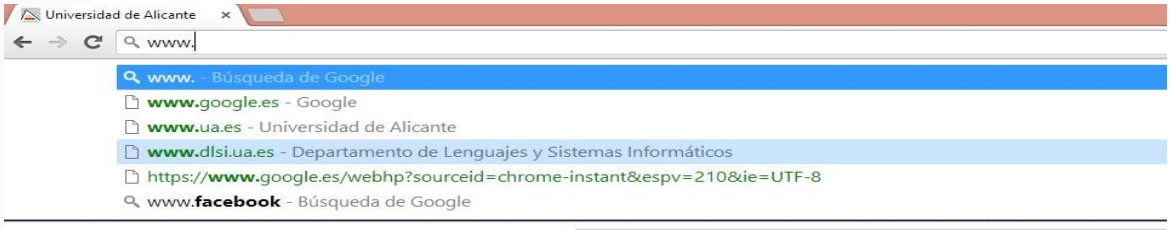
- **A partir de tipus definits per l'usuari ("tlista" –herència o layering–)**

- Avantatges
  - **Avantatge: no hi ha definida una grandària per a la cua**

Piles i cues

Aplicacions reals

- Gestió de l'ús de recursos compartits: cues d'impressió, cues de processos pendents d'execució, cues de missatges, etc.
- Els editors de text proporcionen normalment un botó desfer que cancel·la les operacions d'edició recents i restableix l'estat anterior del document (emmagatzematge en una pila).
- Els Navegadors en Internet emmagatzemen en una pila les adreces dels llocs més recentment visitats.



EXERCICIS

- Donada la classe *TVector* que emmagatzema un vector dinàmic de enters i un enter que conté la dimensió del vector, definir en C++:
  - La classe *TVector*
  - Constructor per defecte (dimensió 10 i components a -1)

EXERCICIS

- Donada la classe *TPila* definiu en C++ el mètode *Apilar*

Piles i cues

Preguntes de tipus test. Vertader vs Fals

- La semàntica de l'operació cim del tipus pila vista en classe és la següent:

  VAR p: pila, e: item;

  cima(crear( )) = error( )

  cima(apilar(p, e)) = cima(p)

- És possible obtenir una representació enllaçada d'una cua utilitzant un únic punter que apuntarà al fons de la cua