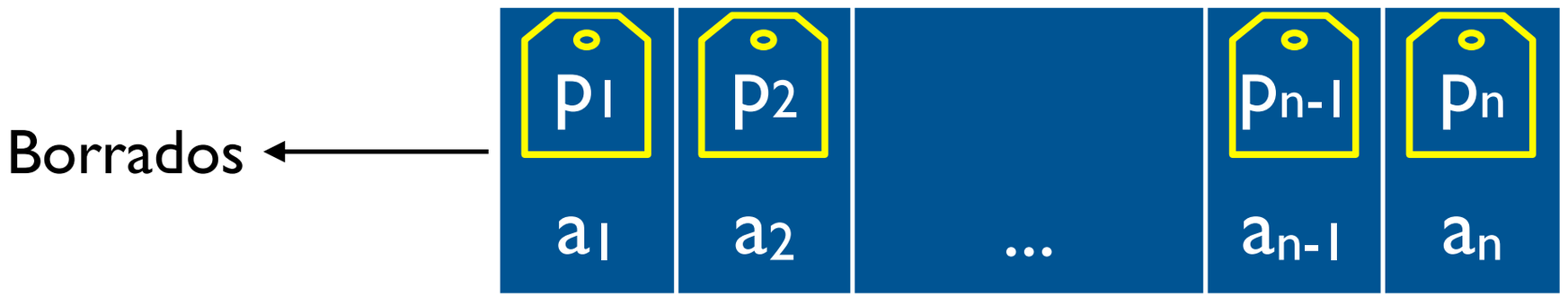
ESTRUCTURAS DE DATOS LINEALES

COLAS CON PRIORIDAD

Joaquín Fernández-Valdivia
Javier Abad
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

Colas con prioridad

- Una cola con prioridad es una estructura de datos lineal diseñada para realizar accesos y borrados en uno de sus extremos(frente). Las inserciones se realizan en cualquier posición, de acuerdo a un valor de prioridad



- **Operaciones básicas:**
  - Frente: devuelve el elemento del frente
  - Prioridad_Frente: devuelve la prioridad asociada al elemento del frente
  - Poner: añade un elemento con una prioridad asociada
  - Quitar: elimina el elemento del frente
  - Vacia: indica si la cola está vacía

Colas con prioridad

Esquema de la interfaz

```
#ifndef __COLA_PRI__
#define __COLA_PRI__

class ColaPri{

private:
  ...      //La implementación que se elija

public:
  ColaPri();
  ColaPri(const ColaPri& c);
  ~ColaPri();
  ColaPri& operator=(const ColaPri& c);

  bool vacia() const;
  Tbase frente() const;
  Tprio prioridad_frente() const;
  void poner(Tbase e, Tprio prio);
  void quitar();
};

#endif /* ColaPri_hpp */
```

Colas con prioridad

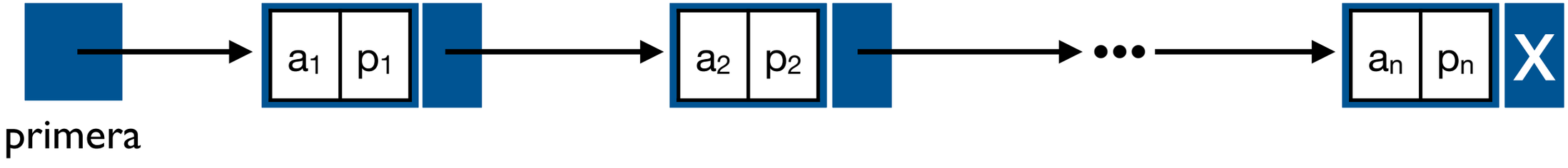
Uso de una cola

```
#include <iostream>
#include "ColaPri.hpp"
using namespace std;
int main(){
  ColaPri c;
  int nota;
  string dni;

  cout << "Escriba una nota: ";
  cin >> nota;
  while(nota >=0 && nota <=10){
    cout << "Escriba un dni: ";
    cin >> dni;
    c.poner(dni, nota);
    cout << "Escriba una nota: ";
    cin >> nota;
  }
  cout << "DNIs ordenados por nota:" << endl;
  while(!c.vacia()){
    cout << "DNI: " << c.frente() << " Nota: "
         << c.prioridad_frente() << endl;
    c.quitar();
  }
  return 0;
}
```

Colas con prioridad. Celdas enlazadas

Almacenamos la secuencia de parejas en celdas enlazadas



- Una cola vacía contiene un puntero nulo
- El frente de la cola está en la primera celda (muy eficiente)
- Si borramos el frente, eliminamos la primera celda
- En la inserción tenemos que buscar la posición según su prioridad

ColaPri.h

```
#ifndef __COLA_PRI__
#define __COLA_PRI__

#include <string>
using namespace std;
typedef int Tprio;
typedef string Tbase;

struct Pareja{
  Tprio prioridad;
  Tbase elemento;
};

struct CeldaColaPri{
  Pareja dato;
  CeldaColaPri* sig;
};
```

```
class ColaPri{
private:
  CeldaColaPri* primera;
public:
  ColaPri();
  ColaPri(const ColaPri& c);
  ~ColaPri();
  ColaPri& operator=(const ColaPri& c);

  bool vacia() const;
  Tbase frente() const;
  Tprio prioridad_frente() const;
  void poner(Tbase e, Tprio prio);
  void quitar();
};

#endif /* ColaPri_hpp */
```

ColaPri.cpp

```
#include <cassert>
#include <utility>
#include "ColaPri.hpp"

ColaPri::ColaPri(){
  primera = 0;
}

ColaPri::ColaPri(const ColaPri& c){
  if(c.primera==0) //Si está vacía
    primera = 0;
  else{              //Si no está vacía
    primera = new CeldaColaPri;       //Crea la primera celda
    primera->dato = c.primera->dato; //Copia el dato
    CeldaColaPri* src = c.primera;   //Inicializa punteros
    CeldaColaPri* dest = primera;
    while(src->sig!=0){                //Mientras queden celdas
      dest->sig = new CeldaColaPri;  //Crea nueva celda
      src = src->sig;                  //Avanza punteros
      dest = dest->sig;
      dest->dato = src->dato;          //Copia el dato
    }
    dest->sig = 0;                     //Ajusta el puntero de la última
  }
}
```

ColaPri.cpp

```
ColaPri::~ColaPri(){
  CeldaColaPri* aux;
  while(primera != 0){          //Mientras queden celdas
    aux = primera;              //Referencia a la primera celda
    primera = primera->sig;     //Avanza primera
    delete aux;                 //Borra la celda
  }
}

ColaPri& ColaPri::operator=(const ColaPri &c){
  ColaPri colatemp(c);     //Usamos el constructor de copia
  swap(this->primera, colatemp.primera);
  return *this;
  //El destructor libera el contenido de *this
}

bool ColaPri::vacia() const{
  return (primera==0);
}
```

ColaPri.cpp

```
Tbase ColaPri::frente()const{
  assert(primera!=0);
  return (primera->dato.elemento);
}

Tprio ColaPri::prioridad_frente() const{
  assert(primera!=0);
  return(primera->dato.prioridad);
}

void ColaPri::quitar(){
  assert(primera!=0);
  CeldaColaPri* aux = primera;
  primera = primera->sig;
  delete aux;
}
```

ColaPri.cpp

```
void ColaPri::poner(Tbase e, Tprio prio){
  CeldaColaPri* aux = new CeldaColaPri;  //Creamos una nueva celda
  aux->dato.elemento = e;                //Guardamos la información
  aux->dato.prioridad = prio;
  aux->sig = 0;
  if (primera==0)                        //Si la cola está vacía
    primera = aux;
  else if(primera->dato.prioridad<prio){ //Si no está vacía y tiene
                                         //prioridad máxima
    aux->sig = primera;                  //La insertamos la primera
    primera = aux;
  }
  else{                                  //Caso general
    CeldaColaPri* p = primera;
    while(p->sig!=0){                    //Avanza por las celdas
      if(p->sig->dato.prioridad<prio){
        aux->sig = p->sig;
        p->sig = aux;
        return;
      }
      else p = p->sig;
    }
    p->sig = aux;
  }
}
```

Colas con prioridad

Uso de una cola
STL

```
#include <iostream>
#include <queue>
using namespace std;
int main(){
  priority_queue<Pareja> c;
  Pareja p;
  cout << "Escriba una nota: ";
  cin >> p.nota;
  while(p.nota >=0 && p.nota <=10){
    cout << "Escriba un dni: ";
    cin >> p.dni;
    c.push(p);
    cout << "Escriba una nota: ";
    cin >> p.nota;
  }
  cout << "DNIs ordenados por nota:" << endl;
  while(!c.empty()){
    p = c.top();
    cout << "DNI: " << c.top().dni << " Nota: "
    << c.top().nota << endl;
    c.pop();
  }
  return 0;
}
```

```
struct Pareja{
  int nota;
  string dni;
  bool operator<(const struct Pareja & otra) const{
    return (this->nota < otra.nota);
  }
};
```

Colas con prioridad

Uso de una cola
STL

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
  priority_queue<pair<int, string>> c;
  pair<int, string> p;

  cout << "Escriba una nota: ";
  cin >> p.first;
  while(p.first >=0 && p.first <=10){
    cout << "Escriba un dni: ";
    cin >> p.second;
    c.push(p);
    cout << "Escriba una nota: ";
    cin >> p.first;
  }
  cout << "DNIs ordenados por nota:" << endl;
  while(!c.empty()){
    p = c.top();
    cout << "DNI: " << c.top().second << " Nota: "
    << c.top().first << endl;
    c.pop();
  }
  return 0;
}
```

Ejercicio propuesto

- Desarrollar una clase cola con prioridad genérica usando templates
- Podríamos pensar en desarrollar la clase patrón usando dos parámetros, uno para el tipo base y otro para la prioridad
- Sin embargo, el enfoque más cómodo y versátil, tanto para el desarrollador como para el usuario de la clase, es seguir el enfoque de la STL: dejar en manos del usuario de la clase la definición del tipo base, al que sólo se le exige que tenga definido el operador <
- De esta forma, podemos usar la cola para almacenar valores de cualquier tipo simple (enteros, caracteres…), parejas valor-prioridad (como en el ejemplo anterior) o cualquier tipo/clase más complejo que tenga implementado el operador <