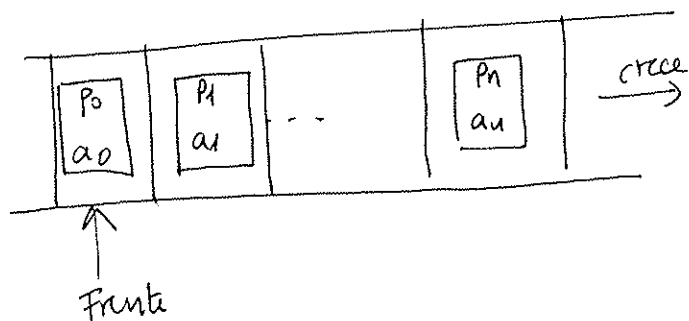


## ④ LECCION 10: COLAS CON PRIORIDAD

### COLA CON PRIORIDAD

Contienen una secuencia de valores especialmente diseñados para realizar los accesos y borrados por el Frente y las inserciones se realizan en cualquier pto de acuerdo a un criterio de prioridad.



### Operaciones

- + Frente: devuelve el elemento en el frente.
- + Prioridad\_Frente: devuelve la prioridad en el elemento del frente.
- + Quitar: Elimina el elemento del frente.
- + Vacio: indica si la cola está vacía.
- + Poner: Inserta un nuevo elemento con una prioridad.

```
#ifndef COLA_PRIO_H
#define COLA_PRIO_H

template <class Tprio, class T>
struct info {
    Tprio prio; // prioridad
    T elemento; // dato almacenado
};
```

```
template <class Tprio, class T>
struct celda {
```

```
    info <Tprio, T> dato;
    celda * sig;
```

```
};
template <class Tprio, class T>
class Cola_Prio {
```

```
private:
```

```
    celda <Tprio, T> * primera;
    void copiar(const Cola_Prio
    void Borrar();
```

```
public:
```

```
    Cola_Prio();
```

```
    Cola_Prio(const Cola_Prio <Tprio, T> & cp);
```

```
    ~Cola_Prio();
```

```
    Cola_Prio <Tprio, T> & operator=(const Cola_Prio
    <Tprio, T> & cp);
```

```
    T Frente() const;
```

```
    Tprio Prioridad_Frente() const;
```

```
    void Poner(const Tprio & TP, const T & e);
```

```
    void Quitar();
```

```
    int size() const;
```

```
    bool Vacio() const;
```

```
};
```

```
#endif
```

5

LECCION : COLAS. Pnondad

```
template <class Tpno, class T>
T Cola_Pno <Tpno, T>::Frente() const {
    assert (primera != 0);
    return primera->dato.elemento;
}
}
```

```
template <class Tpno, class T>
Tpno Cola_Pno <Tpno, T>::Pnondad_Frente() const {
    assert (primera != 0);
    return primera->dato->prior;
}
}
```

```
template <class Tpno, class T>
void Cola_Pno <Tpno, T>::Poner( const Tpno &Tp, const T &e) {
    Celda <Tpno, T> *aux = new Celda <Tpno, T>;
    aux->dato.pno = pno Tp;
    aux->dato.elemento = e;
    if (primera == 0) {
        primera = aux;
        primera->sig = 0;
    }
    else {
        if (primera->dato.prior < pno) {
            aux->sig = primera;
            primera = aux;
        }
        else { // buscamos donde ponerlo
            Celda <Tpno, T> *p = primera;
            while (p->sig->dato.pno < Tp && p->sig != 0)
                p = p->sig;
            aux->sig = p->sig;
            p->sig = aux;
        }
    }
}
}
```

```
template <class Tpno, class T>
void Cola_Pno <Tpno, T>::Quitar() {
    Celda <Tpno, T> *aux = primera;
    primera = primera->sig;
    delete aux;
}
}
```

## Colas con prioridad

Esquema de cola	Uso de una cola con prioridad
<p>Una posible clase <i>ColaPri</i> para almacenar datos de tipo <i>string</i> con una prioridad indicada por un valor entero puede tener la siguiente sintaxis.</p> <pre>#ifndef __COLAPRI_H__ #define __COLAPRI_H__  class ColaPri{ ... public:     ColaPri();     ColaPri(const ColaPri&amp; p);     ~ColaPri();     ColaPri&amp; operator=(const ColaPri&amp; p);      bool vacia() const;     void poner (int pri, string c)     void quitar();     string frente() const;     int prioridad_frente () const; };  #endif</pre>	<pre>#include &lt;iostream&gt; #include &lt;string&gt; #include &lt;colapri.h&gt; using namespace std;  int main() {     ColaPri q;     int nota;     string dni;      cout &lt;&lt; "Escriba una nota" &lt;&lt; endl;     cin &gt;&gt; nota;     while (0 &lt;= nota &amp;&amp; nota &lt;= 10) {         cout &lt;&lt; "Escriba un dni" &lt;&lt; endl;         cin &gt;&gt; dni;         q.poner(nota,dni);         cout &lt;&lt; "Escriba una nota" &lt;&lt; endl;         cin &gt;&gt; nota;     }     cout &lt;&lt; "Los elementos en el orden de las notas son:" &lt;&lt; endl;     while (!q.vacia()) {         cout &lt;&lt; "Nota:" &lt;&lt; q.prioridad_frente()             &lt;&lt; " DNI:" &lt;&lt; q.frente() &lt;&lt; endl;         q.quitar();     }     return 0; }</pre>