Apuntes Estructura de datos.

10 de marzo de 2017

1. Estructuras de datos

1.1. Tema 2 - Abstracción.

1.1.1. Uso de Template.

Nos permite seleccionar el tipo de dato que vamos a utilizar en tiempo de ejecución. Declarando:

```
template <class T, int n>

class array_n {
private:
    T items[n];
  };
```

Creando un objeto de esta clase de la forma:

```
array_n<int,1000> w
```

Creando un metodo de la forma:

```
template <class T>
T VectorDinamico::componente(int i) const
{
  return datos[i];
}
```

La compilación a la hora de usar templates es distinta a la que estamos acostumbrados. En lugar de hacer un *#include "clase.h"* en el archivo .cpp, se incluirá el archivo *#include "clase.cpp"* al final del archivo .h.

1.1.2. Clase vector dinámico

Vamos a estudiar una clase que tenga un vector de datos y el número de elementos.Un primer ejemplo con tipo de dato float sería:

```
class VectorDinamico{

  float* datos;
  int ns;
}
```

1.1.3. Iteradores

Pretendemos hacer recorridos mucho más rápido. No volvermos a recorrer vectores haciendo v[i]. Usaremos los punteros para iterar, de la forma:

```
double *v = &a.
double *p;
double * fin;
fin = v+n;
for(p = v; p!= fin; ++p)
  cout << *p << endl;
```

Nota: en un compilador moderno, simplemente activando la optimización de código se consigue el mismo aumento en el rendimiento.

Definiremos incluso un nuevo tipo de dato llamado iterator, haciéndolo de la forma:

```
typedef double* iterator;
iterator begin(double* v, int n){
  return v;
}
iterator end(double* v, int n){
  return v+n;
}

/**--------------------------*/
iterator p;
for(p=begin(v,n); p!=end(v,n);++p)
  cout << *p << endl;
```

1.1.4. Pilas

Son estructuras de datos lineales, secuencias de elementos dispuestos en una dimensión. Tienen estructura *LIFO* (last in, first out)

No usaremos el concepto de posición, sólo trabajaremos con el tope de la estructura. En realidad, sí podemos recorrerla pero debemos salvaguardar los elementos , pues para acceder al siguiente elemento tenemos que borrar el anterior.

- Para insertar un elemento, se inserta siempre al principio de la pila, añadiendo físicamente
- Para borrar un elemento, se elimina y el puntero que había al primero se lleva al segundo
- Como la pila no tiene recorridos, no tienen iteradores.
- Las funciones básicas son Tope, Poner, Quitar, Vaciar.

Hay que recordar que los elementos de la pila se guardan en orden contrario al que fueron insertados. Por tanto si los imprimimos, por ser de tipo *LIFO* salen del primero que hemos insertado al último. En la práctica, usaremos las pilas con Celdas Enlazadas.

```
#ifndef __PILA_H__
#define __PILA_H__

```

```
4  typedef char Tbase;
5
6  struct CeldaPila{
7      Tbase elemento;
8      CeldaPila *sig;
9  };
10
11 class Pila{
12
13     CeldaPila *primera;
14
15   public:
16
17     Pila();
18     Pila(const Pila& p);
19     ~Pila();
20     Pila& operator= (const Pila& pila);
21
22     void poner(Tbase c);
23     void quitar();
24     Tbase tope() const;
25     bool vacia() const;
26 };
```

1.1.5. Colas

Una cola es una estructura de datos lineal en la que lso elementos se suprimen e insertan por extremos opuestos. Son estructuras *FIFO*. En una cola, si tenemos dos elementos

Usaremos el operador %, que transforma una estructura lineal en una linear circular. Al igual que en las colas, no tengo acceso a los elementos si no destruimos la cola.