TADS

**Definicion (.h)**

#pragma once

#ifndef NOMBREDETAD\_H

#define NOMBREDETAD\_H

Using namespace std;

Template <class T>

class nomTAD;

Template <class T>

Class nomTAD abstract {

public:

// todo aca adentro

}

virtual void Vaciar() abstract;

// destructor

virtual ~nomTAD(){};

// si hay que crear uno de este tipo.

virtual nomTAD<T> \*crear() const abstract;

// si hay que modificar el que le pasamos por parámetros (ins, del)

virtual void nombreMetodo(const T &p) abstract;

// si hay que retornar un tipo T.

virtual const T& elem() const abstract;

// si hay que comparar con otro de este tipo.

virtual bool nombreComparacion(const nomTAD<T> &p) const abstract;

**Implementaciones (.cpp, public, private)**

PRIVATE - PLANTILLA GENERAL:

* Estructuras que voy a usar.
* Los métodos que reciben por parámetros.
* Los atributos de la clase.
* Los métodos auxiliares recursivos.

Struct nodoAB {

Dominio d;

Rango r;

nodoAB \*izq;

nodoAB \*der;

}

nodoAB \*ab;

Void borrarTodo(nodoAB \*a){

If (a != NULL) {

borrarTodo(a->izq);

borrarTodo(a->der);

delete(a);

}

}

PUBLIC:

* Constructor
* Destructor
* Métodos iterativos que pide la función

// constructor sin parámetros

template <class T>

NombreIMP<T>::NombreIMP() {

this->abb = new NombreIMP<T>();

}

// constructor por parámetros

template <class T>

NombreIMP<T>::NombreIMP(const NombreContrato<T> &var) {

this->lista = new NombreIMP<T>();

\*this = var;

}

// destructor

template <class T>

NombreIMP<T>::~NombreIMP() {

this->Vaciar();

}

**ESQUEMA DE SINTAXIS DE FUNCIONES:**

tipoDeDato NombreIMP<T>::nombreFuncion() *const* {

*// const va si no hay que modificar nada*

}

**EJEMPLOS:**

// retornar un elemento T. EJ: Elemento en pos, Elemento, Recuperar.

const T& NombreIMP<T>::nombreFuncion() const {

return elemento;

}

// retornar un entero. EJ: Ocurrencias, Cardinal.

unsigned int NombreIMP<T>::nombreFuncion(const T &e) const {

return entero;

}

// retornar un boolean. EJ: Existe, Pertenece, esVacio, esLleno.

bool NombreIMP<T>::nombreFuncion() const {

return boolean;

}

SET:

* {1,4} - {4,1} - {1,4,1,4} ES EL MISMO CONJUNTO
* Unión, Intersección y Diferencia
* Posibles: Borrar, Min, Max, Igual.

DICCIONARIO:

* SET con las operaciones Vacío, Ins, esVacio, Borrar y Pertenece.

IMPLEMENTACIONES:

* Arreglos de bits: A[x] pertenece → A[x] = 1 (ORDENADO)
  + Insertar, Borrar y Pertenece en orden 1.
  + Vacio y esVacio en orden n.
  + Union, intersec y diferencia en tiempo proporcional.
  + UNIVERSO FINITO. VERSION ACOTADA
* Listas encadenadas: (NO ORDENADO)
  + Insertar, Borrar y Pertenece en el peor caso O(n).
  + Union, intersec y fierencia en O(n\*m)
  + UNIVERSO INFINITO. VERSION NO ACOTADA.
* Arreglo con tope:
  + Ventaja: sencillez.
  + Desventajas: No puede crecer, borrar es difícil.
* ABB:
  + Tiempos proporcionales al ABB.
  + O(n) en peor caso de buscar.
  + NO SE USA. SE USAN AVL.

TAD COLA:

Utilizo NODOLISTA y lo pienso al contrario de lo normal.

Agrego al final y borro al inicio

Por ejemplo: cola: 5,4,3,2

Para encolar el elemento (1): 5,4,3,2,1\*

Para desencolar: ,4,3,2,1 //Se borró el nodo(5) y el “fin” pasa a ser this->fin->sig

TAD COLA PRIORIDAD:

Igual al tad cola (NODOLISTA), sin embargo con un atributo prioridad.

El elemento con mayor prioridad sale primero.

Si existen dos elementos con la misma prioridad, sigue la lógica FIFO.

TAD PILA:

Sigue la lógica al tad COLA (NODOLISTA).

TAD LISTA POS:

Utiliza un Arreglo como estructura.

TAD LISTA ORD:

Utiliza un NodoABB como estructura, este posee un atributo llamado “auxiliar” que permite contar cuantas veces se repite este elemento.

TAD TABLA:

Función parcial, de elementos de un tipo (dominio) a elementos del otro (rango).

Un valor es “mapeado” a una clave única.

Se necesita clave y valor para insertar nuevos mapeos.

T(d) = r.

Asociar estudiantes a materias aprobadas

Asociar salario semanal a cada empleado

OPERACIONES:

RECUPERAR T(d) - retornar el valor de la clave

ESTÁ DEFINIDO T(d) - existe la clave

ALTA EN EL DOMINIO DE T - alta de clave

ALMACENAR VALORES - dar valor a clave

CREAR VACIO - crear vacío

IMPLEMENTACIONES:

* Arreglo: Con el dominio como índice del arreglo.
  + Arreglo con dos elementos, int rango y bool isDef.
  + Crear, crearia con valores false en todos los isDef.
  + Insertar agregaria el valor del dominio y asignaria true en isDef.
  + Recuperar retornaria el valor del rango correspondiente al dominio parametro.
* Lista de pares.
* ABB: Si admite un orden total
* Lista Ordenada de Asociaciones ListaOrd<Asociacion<D,R>> \* lista

//////

MultiSets (no hay orden pero si se permite repet.)

Sets (no hay orden ni repetición de elementos)

MÉTODOS BÁSICOS POR CADA TAD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LISTA** | **PILA** | **COLA** | **SET** | **DICCIONARIO** | **TABLA** |
| void Vacia() | void vacia() | void Vacia() | void Vacio() | void Vacio() | void crearVacio() |
| void insComienzo(Etype) | void push(Etype) | void encolar() | void insertar(Etype) | void insertar(Etype) | void insertar(d, r) |
| bool esVacia() | bool esVacia() | bool esVacia() | bool esVacio() | bool esVacio() | bool esVacia() |
| Etype cabeza() | Etype top() | Etype front() | bool Pertenece(Etype) | bool Pertenece(Etype) | bool esDefinida(d) |
| void elimPrimero() | void pop() | void desencolar() | void union(Set) | void borrar(etype) | rango recuperar(d) |
| int cantElementos() |  |  | void insersec(Set) | void union(dicc) | bool recuperar(d, r) |
|  |  |  | void diferencia(Set) | void insersec(dicc) | eliminar(d) |
|  |  |  |  | void diferencia(dicc) |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

NO VIMO MATA