TADs y Secuencias

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

TAD
Tipo Abstracto de Dato

¿Qué es un TAD?

- TAD: Tipo Abstracto de Dato.
  - Nuevo tipo de dato bien especificado (propio).
  - Amplía el lenguaje de programación.
- Consta de:
  - Datos (representación):
    invisibles al usuario (protegidos o privados).
  - Funciones y procedimientos (operaciones):
    encapsulados dentro del TAD, acceso por interfaz.

¿Qué es un TAD?

- TAD: Tipo Abstracto de Dato.
  - Nuevo tipo de dato bien especificado (propio).
  - Amplía el lenguaje de programación.
- Definición:
  - Especificación (¿qué hace el TAD?)
    General, global. Formal, matemática.
  - Implementación (¿cómo lo hace el TAD?)
    Dependiente del lenguaje a usar.

¿Qué es un TAD?

Ventajas:

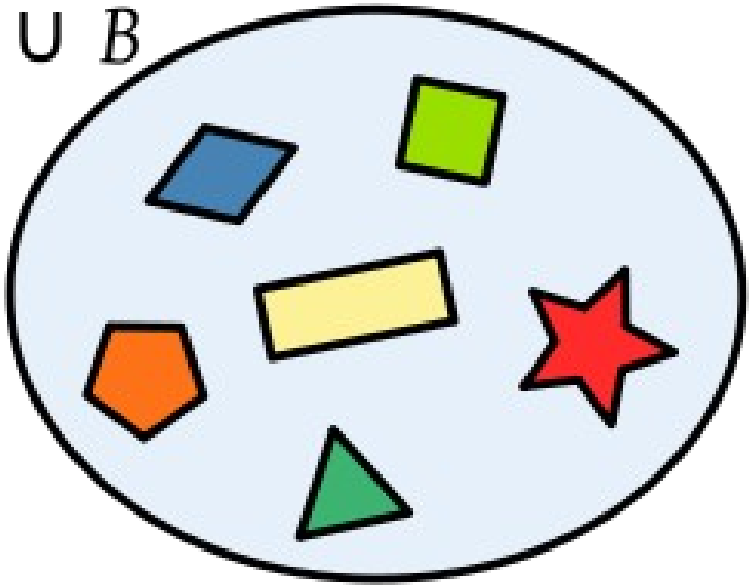
- Mejor conceptualización y modelización.
- Mejora la robustez.
- Mejora el rendimiento (optimización del tiempo de compilación).
- Separa la implementación de la especificación.
- Permite extensibilidad.

¿Qué es un TAD?

Ejemplos:

- TAD Conjunto
- TAD Carrera
- TAD Biblioteca
- TAD Vehículo

$A \cup B$



https://co.pinterest.com/pin/532832199639015903/

http://es.wikipedia.org/wiki/Unión_de_conjuntos

Diseño de TADs

- Programas = Algoritmos + Datos (ecuación de Wirth).
  https://web.archive.org/web/20130207170133/http://www.inf.ethz.ch/personal/wirth/books/AlgorithmE0/

- Diseño de TADs:
  - Comportamientos del objeto en el mundo real (interfaz/verbos).
  - ***Conjunto mínimo*** de datos que lo representan (estado/sustantivos).
  - TAD = Interfaz + Estado.

Diseño de TADs

- Plantilla base para diseño de TADs:

  TAD *nombre*

  Conjunto mínimo de datos:
  nombre valor (variable), tipo de dato, descripción
  ...

  Comportamiento (operaciones) del objeto:
  nombre operación (argumentos), descripción funcional
  ...

Diseño de TADs

- Plantilla base para diseño de TADs:

  TAD *Carrera*

  Conjunto mínimo de datos:
  - *nombre, cadena de caracteres, identificación de la carrera*
  - *numEst, número entero, cantidad de estudiantes inscritos*

  Comportamiento (operaciones) del objeto:
  ObtenerNombre(), retorna el nombre de la carrera
  ObtenerNumEst(), retorna la cantidad de estudiantes de la carrera
  FijarNombre(nuevoNom), modifica el nombre actual de la carrera
  AgregarEstud(), incrementa en uno el conteo de estudiantes
  EliminarEstud(), decrementa en uno el conteo de estudiantes

Diseño de TADs

- Diagrama de relación entre TADs:

¡Atención!
¡No es un diagrama de clases!

Implementación de TADs

- Diferentes posibilidades:
  - Estructura con datos y funciones propias o adicionales.
  - Clase con niveles de acceso.
- Utilizando librerías:
  - En el archivo de cabecera (.h) se realiza la declaración de los datos y los prototipos.
  - En el archivo de implementación (.cpp ó .cxx) se desarrolla la implementación de las operaciones.

Implementación de TADs

- Librería, 2 archivos:

  - **Encabezado (.h)**:

  Incluye la definición del TAD, en términos de datos y prototipos de operaciones.

  - **Implementación (.cpp ó .cxx):**

  Incluye la implementación específica de las operaciones del TAD.

  `#include "TAD.h"` (al principio del archivo)

Implementación de TADs

- Opción 1: datos en estructura, operaciones como funciones adicionales.

```
struct Carrera{
  std::string nombre;
  unsigned long numEst;
};


std::string ObtenerNombre(Carrera *c);
unsigned long ObtenerNumEst(Carrera *c);
void FijarNombre(Carrera *c, std::string nombre);
void AgregarEstud(Carrera *c);
void EliminarEstud(Carrera *c);
```

Implementación de TADs

- Opción 2: datos y operaciones en estructura.

```
struct Carrera{
  std::string nombre;
  unsigned long numEst;


  std::string ObtenerNombre();
  unsigned long ObtenerNumEst();
  void FijarNombre(std::string nombre);
  void AgregarEstud();
  void EliminarEstud();
};
```

Implementación de TADs

- Opción 3: datos y operaciones dentro de una clase de C++.

```
class Carrera {
  public:
    Carrera();
    std::string ObtenerNombre();
    unsigned long ObtenerNumEst();
    void FijarNombre(std::string name);
    void AgregarEstud();
    void EliminarEstud();
  protected:
    std::string nombre;
    unsigned long numEst;
};
```

Implementación de TADs

- Sintaxis básica en encabezado:

  - Regiones:

    - Públicas (`public`):
      Todas las operaciones del TAD.
      Dan acceso a los datos (atributos) del TAD.
    - Protegidas (`protected`) o Privadas (`private`):
      Todos los datos (atributos) del TAD.

Diseño e implementación de TADs

1. Diseñar el TAD:
   descripción + diagrama.
2. Escribir el archivo de encabezado (.h):
   un archivo por TAD.
3. Escribir el archivo de implementación (.cxx, .cpp):
   un archivo por TAD.
4. Escribir el archivo de programa (.cxx, .cpp):
   programa con procedimiento principal,
   incluye el encabezado de cada TAD a utilizar.

¡Atención!

- Recordatorio:

  El **diseño e implementación** descrito anteriormente será **exigido** en todas las actividades del curso (talleres, parciales, proyecto, quices) **de aquí en adelante, durante todo el semestre.**

Secuencias

Secuencias

- Una secuencia es una estructura que representa una lista de elementos del mismo tipo.
- Formalmente:

$$S = \{ s_i \in T : 1 \leq i \leq n , i \in \mathbb{N} \}$$

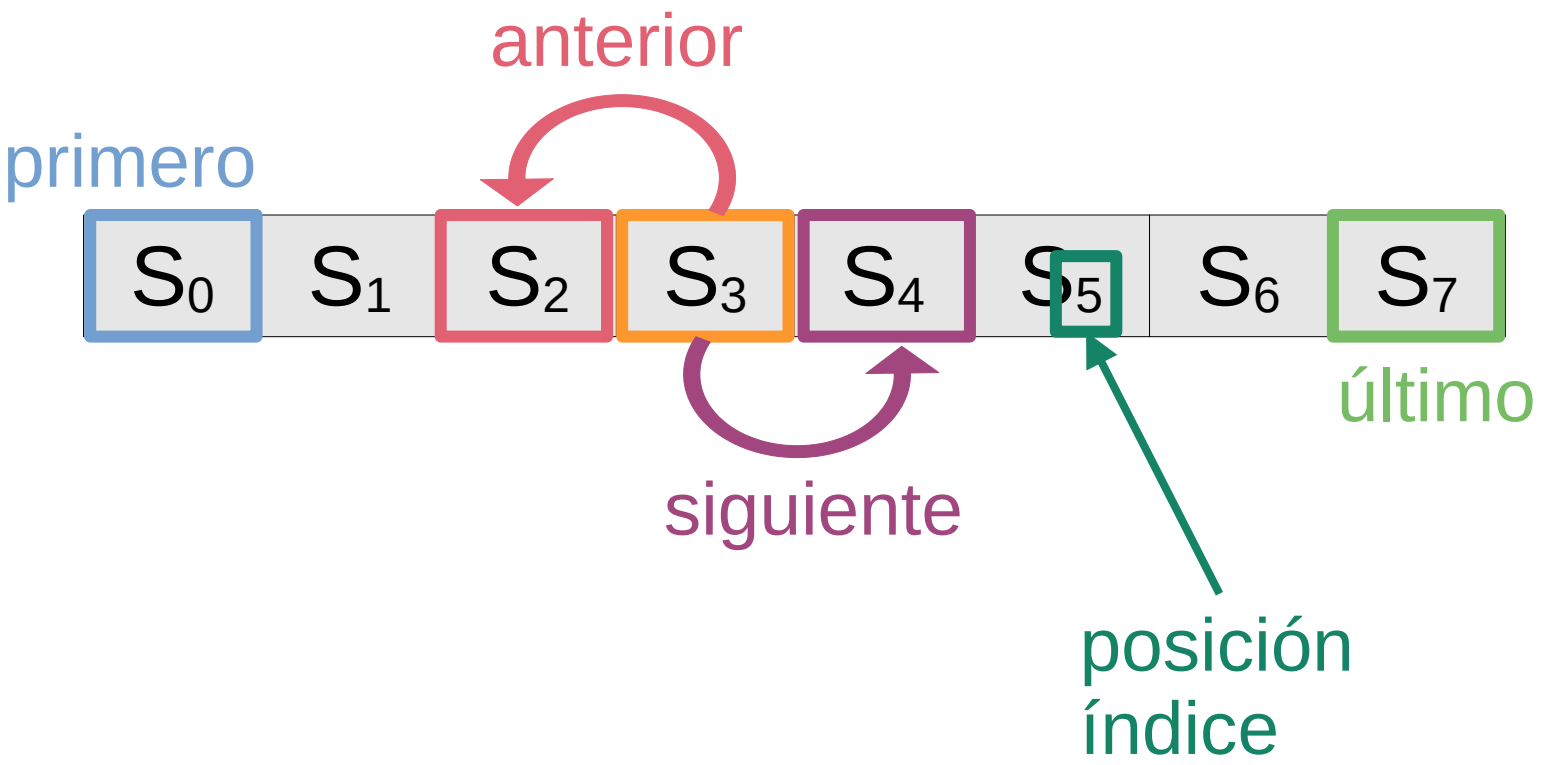
$$S = \{ s_1, s_2, s_3, \ldots, s_n \}$$

- Ejemplo: secuencia de los números de Fibonacci

1	1	2	3	5	8	13	...
S_0	S_1	S_2	S_3	S_4	S_5	S_6	...

Secuencias

- Conceptos relevantes:



Secuencias

- Conceptos relevantes:

Secuencia 1

S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7

Secuencia 2 ≠ Secuencia 1

S_1	S_3	S_6	S_0	S_2	S_5	S_7	S_4

Secuencias

¿Cómo implementarlas?

- Arreglo estático: tamaño fijo

```
int miArr[8] = {1,1,2,3,5,8,13,21};
```

1	1	2	3	5	8	13	21

```
string miStr = “palabras”;
```

p	a	l	a	b	r	a	s

Secuencias

¿Cómo implementarlas?

- Arreglo estático: tamaño fijo

  Problema: limitado a la capacidad predefinida


```
int miArr2[9];
for (int i = 0; i < 8; i++)
   miArr2[i] = miArr[i];
miArr2[8] = 34;
```

Secuencias

¿Cómo implementarlas?

- Arreglo por memoria dinámica: tamaño variable

```
int *miArr = new int[n];
for (int i = 0; i < n; i++)
   *(miArr+i) = i+1;
```

n = 8

1	2	3	4	5	6	7	8

n = 5

1	2	3	4	5

Secuencias

¿Cómo implementarlas?

- Arreglo por memoria dinámica: tamaño variable

  Problema: limitado a la capacidad con que se crea


```
int *miArr2 = new int[9];
for (int i = 0; i < 8; i++)
    *(miArr2+i) = *(miArr+i);
*(miArr2+8) = 9;
delete[] miArr;
```

Secuencias

¿Cómo implementarlas?

- Necesidad:

  - estructura totalmente dinámica, que ajuste su tamaño en tiempo de ejecución

  - estructura flexible, que acepte diferentes tipos de datos (no mezclados)

Continuará….

Programación genérica
Plantillas

Programación genérica

```
a = b + c;
```

En C++, ¿qué problemas tiene la línea anterior?

Programación genérica

```
int b = 5;
int c = 6;
int a = b + c;
```

```
int suma(    int a,    int b )
{ return( a + b ); }
```

Programación genérica

```
float b = 5.67;
float c = 6.923;
float a = b + c;
```

```
float suma( float a, float b )
{ return( a + b ); }
```

Programación genérica

- Uso de plantillas:
  Generalización → adaptabilidad, flexibilidad.

```
template< class identificador >
declaracion_funcion;

template< typename identificador >
declaracion_funcion;
```

Programación genérica

- Plantilla con un tipo de dato:

```
template< class T >
T suma( T a, T b )
{ return( a + b ); }

int a = suma<int>(5, 7);
double b = suma<double>(6.4, 1.7);
```

Programación genérica

- Plantilla con dos tipos diferentes de dato

```
template< class T, class U >
T suma( T a, U b )
{ return( a + b ); }

int i, j = 25;
long l = 4567;
i = suma<int,long> (j,l);
```

Programación genérica

- Plantilla para clases

```
template< class T >
class vec_par {
  T valores [2];
  public:
    vec_par (T uno, T dos)
    { valores[0] = uno;
      valores[1] = dos; }
    T minimo ();
};
```

Programación genérica

- Plantilla para clases

```
template< class T >
T vec_par<T>::minimo () {
  T resultado;
  if (valores[0]<valores[1])
    resultado = valores[0];
  else
    resultado = valores[1];
  return resultado;
}
```

Programación genérica

- Plantilla para clases

```
vec_par<int> obj_i (115,36);
int res;
res = obj_i.minimo();

vec_par<float> obj_f (32.56,76.98);
float res2;
res2 = obj_f.minimo();
```

Programación genérica

- Ejercicio:

  Utilizando plantillas, implementar un TAD Operación Binaria, definido por:

  - Operando1 (entero, real o caracter).
  - Operando2 (entero, real o caracter).
  - Operación (caracter, uno de: '+', '-', '*', '/').

  Y con la operación:

  - EvaluarOperación (OperaciónBinaria): retorna el resultado de aplicar la operación sobre los operandos.

Programación genérica

- Ejercicio:

```
template <class N>
struct OpBinaria {
   N op1;
   N op2;
   char operacion;

   N EvaluarOperacion();
};
```

https://repl.it/@Andrea_del_Pila/PC-plantillas-ejercicio

Programación genérica

- Ejercicio:

```
template <class N>
N OpBinaria<N>::EvaluarOperacion() {
  N resul;
  switch( operacion ) {
    case '+': resul = op1 + op2; break;
    case '-': resul = op1 – op2; break;
    case '*': resul = op1 * op2; break;
    case '/': resul = op1 / op2; break;
  }
  return resul;
}
```

https://repl.it/@Andrea_del_Pila/PC-plantillas-ejercicio

Programación genérica

Organización de las librerías con plantillas:

- Encabezado (.h)
- Implementación (.hxx, ¿por qué no .cxx?)
- ESTOS DOS ARCHIVOS ***NO SE COMPILAN***.
  - Se usan en un archivo compilable (.cxx, .cpp) donde se ***INSTANCIAN*** los elementos genéricos.

Programación genérica

Organización de las librerías con plantillas:

- Encabezado o cabecera (.h):
  - Incluye al final el archivo de implementación (.hxx).
- Archivo de código (implementación) (.hxx):
  - Incluye al principio la cabecera (.h).

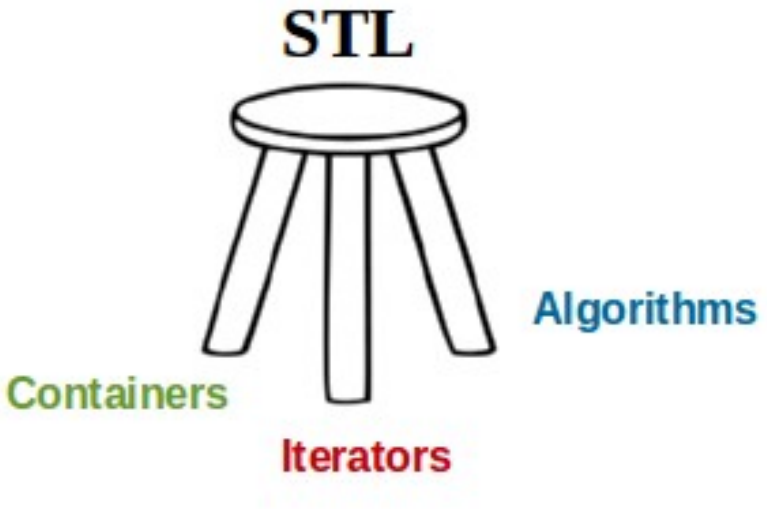
Para usar la librería con plantilla:

- Incluir la cabecera (.h) en un archivo con procedimiento principal (.cpp, .cxx).

STL
Standard Template Library

STL (Standard Template Library)

- ¡Librería con “muchas cosas” genéricas!
- Provee un conjunto de clases comunes, usables con cualquier tipo de dato y con operaciones elementales.
- Tres componentes:
  - Contenedores (*containers*).
  - Algoritmos (*algorithms*).
  - Iteradores (*iterators*).



www.bogotobogo.com/
cplusplus/stl_vector_list.php

http://www.sgi.com/tech/stl

STL (Standard Template Library)

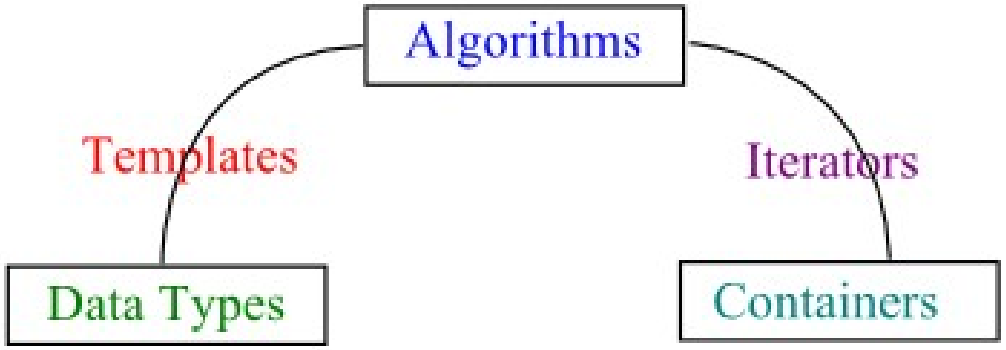
Componentes:

- Contenedores: clases predefinidas para almacenamiento de datos.
- Algoritmos: operaciones básicas como búsqueda y ordenamiento.
- Iteradores: permiten recorrer los datos en los contenedores (similar a apuntadores).

http://www.sgi.com/tech/stl

STL (Standard Template Library)

¿Cómo se conectan estos conceptos?



1. Templates
   make algorithms independent of the data types
2. Iterators
   make algorithms independent of the containters

www.bogotobogo.com/cplusplus/stl3_iterators.php

http://www.sgi.com/tech/stl

Tarea

- Revisar la STL e identificar:
  - ¿Cuáles contenedores provee?
  - ¿Qué diferencias hay entre ellos?

Referencias

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms, 3rd edition. MIT Press, 2009.
- L. Joyanes Aguilar, I. Zahonero. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, 2004.