Implementación y uso de TADs

Isabel Pita

Facultad de Informática - UCM

23 de enero de 2023

Bibliografía Recomendada

- Estructuras de datos y métodos algorítmicos: 213 ejercicios resueltos. *Matí-Oliet, N.; Ortega Mallén, Y., Verdejo Lopez, A.* Garceta 2013, segunda edición.
  Capítulos 1 y 2.
- Data Structures and Algorithm Analysis in C++. *Weiss, M. A.* Pearson 2014, fourth edition.

Objetivos

1. ¿Que es un TAD?. TADs definidos en la librería STL.
2. TADs definidos por el usuario
   - Selección de la representación y de las operaciones del TAD.
3. Concepto de *interfaz*.
4. Implementación de TADs.
   - Problemas del uso de structs.
   - Concepto de clase y su implementación en C++.
   - Constructores y destructores.
   - Métodos observadores y modificadores.
   - Tratamiento de errores.
5. Uso de TADs.

Tipos Abstractos de Datos.

Tipos abstractos de datos = conjunto de valores + operaciones.

TADs predefinidos:

Valores	Representación	Operaciones
bool	un bit	*true*, *false*, $\neg$, $\wedge$, $\vee$
int	complemento a 2	$+, -, *, /, \%, ==, !=, < \ldots$
float	mantisa y exponente	$+, -, *, /, ==, !=, < \ldots$
char	1 byte	$+, -, <, >, ==, \ldots$
array	puntero	_[_],
ficheros	....	lectura, escritura...

- Sólo se necesita conocer las restricciones que la representación impone a los valores del tipo no la representación concreta.
- Los valores sólo se pueden utilizar con las operaciones que se proporcionan.

Tipos Abstractos de Datos.

TADs definidos en la librería STL:

Valores	Operaciones
complex	real, imag, norm, conj ...
string	[ ], push_back, insert, copy, find, substr,...
vector	[ ], push_back, pop_back, ...
set	insert, erase, find, lower_bound, upper_bound, empty,...
multiset	insert, erase, find, count, lower_bound,...
stack	empty, size, top, push, pop,...
queue	empty, size, front, back, push, pop,...
list	push_front, pop_front, push_back, pop_back,...
map	[ ], insert, erase, find, count, lower_boud,...
unordered_map	empty, size, [ ], find, insert, erase,...

TADs definidos por el usuario

- El uso de un TAD puede simplificar la solución de un problema.
  - Ejemplo: en el problema ¿Cuánto queda para las vacaciones?, el uso de un TAD *Fecha* nos permitiría operar con las fechas evitando oscurecer la solución del problema con el control de la creación de fechas correctas y la implementación de las operaciones que manejan las fechas.
- Crearemos un TAD cuando nuestro problema necesite un tipo cuyos datos tienen unas operaciones específicas y el problema se pueda resolver utilizando estas operaciones.
- Al crear un TAD debe pensarse en su aplicación en más de un problema, de forma que se pueda reusar el código implementado.
  - Ejemplo: en el TAD *Fecha* añadiremos todas las operaciones que puedan ser útiles para cualquier aplicación que utilice fechas, e incluiremos nuestro TAD en una librería para ser usado por muchas aplicaciones.

TAD Fecha

- Un TAD admite distintas implementaciones. Elegiremos la más eficiente para las operaciones del tipo.
- Por ejemplo, el TAD *Fecha*, admite al menos dos implementaciones:
  - Utilizar tres variables que representen el día, mes y año de la fecha en cuestión,
  - o utilizar el número de días transcurridos desde una fecha determinada.
- La primera implementación facilita la implementación de las operaciones que devuelven el día, mes y año de la fecha, pero dificulta el cálculo de la diferencia de días entre dos fechas.
- La segunda implementación por el contrario facilita el cálculo de la diferencia entre dos fechas, pero dificulta el cálculo del día y mes concretos a los que pertenece la fecha.

TAD Conjunto. Posibles representaciones

- Un TAD que represente conjuntos de elementos admite también diversas representaciones.
  - Utilizar un vector de elementos desordenados,
  - Utilizar un vector de elementos ordenados.
  - Utilizar vectores con elementos repetidos tanto en vectores ordenados como desordenados.
  - Utilizar una estructura arborescente
- Estudia la diferencia en el coste de las siguientes operaciones con cada una de las tres primeras representaciones.
  - añadir un elemento al conjunto,
  - decidir si un elemento pertenece al conjunto,
  - eliminar un elemento del conjunto,
  - la union o intersección de conjuntos

Concepto de interfaz

- La interfaz de un TAD son los datos y las operaciones que el TAD ofrece al usuario para que lo utilice.
- Si los datos están en la interfaz se permite al usuario que los modifique sin control del TAD. Cuidado, ¿Podemos dejar que un usuario modifique una fecha arbitrariamente?.
- Muchas aplicaciones no pueden permitir al usuario que utilice los datos del tipo libremente, también pueden necesitar utilizar funciones auxiliares para implementar las funciones de la interfaz que no son apropiadas para ser utilizadas por las aplicaciones de los usuarios.

Implementación de TADs en C++

Uso de structs vs clases de C++.

- struct define nuevos tipos de datos, pero no oculta los detalles de la implementación.
  - Ejemplo:

```
struct Fecha {
 int dia; int mes; int anyo;
 Fecha(int d,int m,int a):dia(d),mes(m),anyo(a){}
};
```

  - Crear una fecha: `Fecha f = Fecha(14, 7, 1789);`
  - Otra forma: `Fecha f(14, 7, 1789);`,
  - Es posible crear fechas inconsistentes; `f.mes = 13;`
- Programación OO / clases permite ocultar los detalles de la implementación de los tipos.

Implementación de TADs en C++

La implementación de las clases en C++ se explica en el problema

¿Cuanto queda para las vacaciones?

del cuaderno de problemas de la asignatura.

Tratamiento de errores.

Ejemplo: Lanzando y capturando excepciones en C++

```
#include <iostream>
int divide(int a, int b) {
    if (b==0) {
        throw "division por cero"; // 'const char *'
    }
    return a/b;
}

int main() {
    try {
        std::cout << divide(12,3) << '\n';
        std::cout << divide(1, 0) << '\n';
    } catch (const char * &s) {   // ref. al tipo
        std::cout << "ERROR: " << s << '\n';
    }
    return 0;
}
```

Tratamiento de errores.

- Es conveniente diferenciar los tipos de excepciones que se producen.
- Utilizaremos la librería estándar `<stdexcept>`
- Tipos de errores:
  - `domain_error`: Las funciones no están definidas para esos valores.
  - `invalid_argument`: Los argumentos no son correctos.
  - `length_error`: No hay memoria disponible
  - `out_of_range`: accesos fuera de rango en vectores

Tratamiento de errores.

```
int divide(int a, int b) {
    if (b==0)
       throw std::domain_error("division por cero.");
    return a/b;
}

int main() {
 try {
    cout << divide(1, 0) << "\n";  // lanza excepcion
} catch (domain_error &de) {
    cout <<  de.what() << '\n';
} catch (invalid_argument &ia) {
    cout << ia.what() << '\n';
} catch (...) {    // trata cualquier otra excepcion
    cout << "Excepcion distinta de las anteriores!\n";
}
 return 0;
}
```