TD3: Algoritmos y Estructuras de Datos

Prof. Agustín Garassino, Gervasio Pérez

Segundo Semestre de 2024

Clase Teórica 3
Especificación Formal de Problemas (2)
Contratos

Contratos en TD3

En esta materia, nuestro formato de especificación será

- la declaración de la función en C++ como **encabezado**
- un apartado **Pre** que especifica la **Pre**condición (**requiere**) de la función
- un apartado **Post** que especifica la **Post**condición (**asegura**) de la función

El mismo ejemplo de TD1

Calcular el volumen de un cilindro de radio `r` *y altura* `h`*.*

```
float volumen_cilindro(float r, float h)
```

Pre: $r > 0 \land h > 0$

Post: $res = \pi * r^2 * h$

- *la variable especial* ***res*** *representa al valor de retorno de la función*

Pre y **Post** estarán escritos en un *lenguaje formal*:
Lógica de Primer Orden

Contratos: encabezado

El encabezado será simplemente la declaración de la función en C++ que resuelve el problema especificado, restringiendo los tipos a los que determinamos la clase pasada (**bool**, **char**, **int**, **float**, vector<T>).

Se admite pasaje de parámetros por copia/referencia/referencia constante.

Se admite cualquier tipo de retorno de los permitidos, incluyendo **void**.

Ejemplo: *Calcular la sumatoria de un vector de enteros:*

- `int sumatoria(vector<int> v)`
- `int sumatoria(const vector<int>& v)`
- `void sumatoria(vector<int> v, int & resultado)`
- `void sumatoria(const vector<int>& v, int & resultado)`

Contratos: **Pre**condición

La precondición da condiciones que tienen que cumplir los parámetros de entrada. También puede interpretarse como una descripción del *estado inicial* del programa.

- Para variables pasadas por **referencia no constante** (que pueden ser modificadas por el programa) suele ser necesario recordar su valor inicial en la Pre. Esto se consigue mediante metavariables:
  Pre: $v = v_0$
  Por convención usamos el mismo nombre de la variable real con un subíndice 0.
  Cuidado. v_0 no es una variable del programa, sino un **valor** que solo existe en la especificación.

Contratos: **Post**condición

La postcondición da condiciones que tiene que cumplir el valor de retorno y los parámetros de entrada pasados por referencia. También puede interpretarse como una descripción (de una parte) del *estado final* alcanzado por la función.

- Si el tipo de retorno **no es** `void`, en la Post contaremos con la variable especial **res** que representa al valor retornado por la función.
- Si un parámetro p fue pasado por **copia o referencia constante**: podemos asumir en la Post que su valor no cambió y cumple las mismas propiedades especificadas en la Pre.
- Si un parámetro de entrada p fue pasado **por referencia no constante**, necesitamos especificar en la Post qué condiciones cumple, usualmente con respecto a su valor inicial p_0 (que debemos haber **recordado** en la Pre).

Contratos: Pasaje por copia, valor de retorno (Ejemplo)

Dada una fracción por su numerador N y su denominador D enteros, minimizarla, devolviendo un par de valores en un vector.

```
vector<int> minimizar_fraccion(int N, int D)
```

Pre: $D \neq 0$

Post: $|res| = 2 \wedge res[0] = N/MCD(N,D) \wedge res[1] = D/MCD(N,D)$

- *$MCD(A,B)$ es una* **función auxiliar** *que calcula el máximo común divisor entre A y B.*
- *En la Post,* ***res*** *es el valor de retorno, de tipo* `vector<int>`

Notar que esta especificación no dice con qué algoritmo se minimizará la fracción, sólo dice qué condiciones debe cumplir la salida *res* para ser una minimización correcta de la entrada $\frac{N}{D}$.

Ejercicio: escribir función MCD

Contratos: Pasaje por referencia (Ejemplo)

En C++ tenemos pasaje *por copia* y *por referencia*. Al escribir contratos podremos especificar el valor final de cada parámetro pasado por referencia *no constante*.

Dada una fracción por su numerador N y su denominador D enteros, minimizarla, ***actualizando N y D****.*

```
void minimzar_fraccion(int & N, int & D)
```

Pre: $N = N_0 \wedge D = D_0 \wedge D > 0$

Post: $(\exists M : \texttt{int})\ esMCD(M, N_0, D_0) \wedge N = N_0/M \wedge D = D_0/M$

- *N_0 y D_0 son* **metavariables:** *una "foto" del* **valor inicial** *de N y D.*
- *$esMCD(A, B, C)$ es un* **predicado auxiliar** *que será verdadero si A es el máximo común divisor entre B y C.*

Ejercicio: escribir predicado *esMCD*

Ejercicios

Especificar los siguientes problemas:

1. Dado un vector de caracteres, indicar si éste es **palíndromo**.
2. Dado un `vector<int>`, modificarlo sumandole 1 a cada uno de sus elementos.
3. Dado un número N, devolver un vector con su factorización en números primos.
4. Dado dos `vector<char>` s_1, s_2, devolver la cantidad de posiciones que tienen el mismo caracter en s_1 y en s_2;
5. Dado un vector de enteros s, devolver un vector que sea el resultado de **ordenar** s.

Buenas y malas especificaciones

Una buena especificación...

- **modela con los tipos adecuados** las entradas y salidas del problema a resolver;
- **es descriptiva**, con predicados y funciones auxiliares que simplifiquen su lectura;
- **No subespecifica:**
  - Las entradas están lo restringidas al problema lo mínimo necesario;
  - las salidas están suficientemente restringidas para ser soluciones útiles;
- **No sobreespecifica:**
  - Las entradas están suficientemente restringidas como para no admitir entradas que no tienen sentido al problema;
  - Las salidas no piden condiciones extra que dejen soluciones válidas afuera.

Más ejercicios

Especificar los siguientes problemas:

1. **Problema de correspondencia:** Dados dos vectores de string (`vector<vector<char>>`) a y b de la misma longitud, devolver una secuencia s de K índices de ambos vectores tales que la concatenación $a_{s[0]}a_{s[1]}\dots a_{s[K-1]}$ sea igual a $b_{s[0]}b_{s[1]}\dots b_{s[K-1]}$.
   - Inputs: `a = {"a","ab","bba"}`, `b = {"baa","aa","bb"}`
   - Un output posible: `res = {3,2,3,1}`
2. **Problema de la suma de subconjuntos:** Dado un vector de enteros v, decir si es posible repartirlo en dos vectores v_1 y v_2 tales que la sumatoria de elementos de v_1 sea igual a la sumatoria de los elementos de v_2. La partición puede no ser consecutiva, es decir, el vector `{1,10,4,-5}` se puede partir en `v1 = {1,4}` y `v2 = {10,-5}`, pues ambos suman `5`.

Especificación: Algunos errores comunes

- No confundir especificación (contrato) con predicado/función auxiliar. La especificación de un problema **usa predicados y funciones auxiliares** para definir la Precondición y la Postcondición del contrato.
- No se puede "usar" un contrato como si fuera un predicado o función auxiliar. Pero sí se pueden reutilizar predicados/funciones auxiliares que se definieron para especificar un problema en la especificación de otros problemas.
- Los predicados lógicos **no se ejecutan**. En lógica no tiene sentido el concepto imperativo de "actualizar el valor de una variable", por ejemplo dentro de una sumatoria. Si sienten que están tratando de escribir algo "como si lo programaran en C++", probablemente estén cometiendo un error lógico.

Resumen

Hoy vimos

- Contratos en TD3. Pre y Post.
- Pasaje de parámetros.
- El problema de sub y sobre especificar.

Ya se puede hacer completa la Guia 2.