

Ejercitación: uso de clases

La ejercitación consiste en completar las funciones especificadas. Todos los ejercicios pueden resolverse editando el archivo `src/funciones.cpp`. A continuación se presentan las especificaciones de las funciones. Los primeros ejercicios pueden resolverse usando los módulos `set`, `map` y `vector` de la biblioteca standard. Luego será necesario usar módulos de la cátedra. Especificaciones de referencia se encuentran al final del enunciado.

En el paquete del taller se encuentran tests para las funciones a implementar. Los tests pueden compilarse usando el target `tests.ejercitacion` en CLion.

Los tests también pueden compilarse y ejecutarse sin usar CLion. Para ello:

1. En una consola ubicarse en el directorio raíz del proyecto. En este debería haber un archivo `CMakeLists.txt`.
2. Ejecutar el comando `$> cmake .` (incluyendo el punto al final). Esto generará un archivo `Makefile`.
3. Ejecutar el comando `$> make TTT` donde `TTT` es uno de los targets mencionados anteriormente. Esto creará un ejecutable con el nombre del target en el directorio actual.
4. Ejecutar el comando `$> ./TTT` siendo `TTT` el nombre del target utilizado anteriormente. Esto correrá el ejecutable.

Una vez que se hacen modificaciones en el programa, no es necesario volver a ejecutar `cmake`, basta con hacer `make TTT && ./TTT`.

Ejercicio 1

QUITAR_REPETIDOS(**in** $s : \text{secu}(\text{int})$) $\rightarrow res : \text{secu}(\text{int})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{$

- Mismos elementos: $(\forall x : \text{int}) (\text{pertenece}(x, res) \iff \text{pertenece}(x, s))$
- Sin repetidos res: $(\forall i, j : \text{nat}) (i, j \leq \text{long}(res) \Rightarrow res[i] = res[j] \iff i = j)$

$\}$

Descripción: Devuelve una secuencia con los mismos elementos pero sin repetidos.

Ejercicio 2

Si al resolver el ejercicio 1 no usaste `set`, resolverlo nuevamente con este módulo.

Ejercicio 3

MISMOS_ELEMENTOS(**in** $a : \text{secu}(\text{int})$, **in** $b : \text{secu}(\text{int})$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res \iff (\forall x : \text{int}) (\text{pertenece}(x, a) \iff \text{pertenece}(x, b))\}$

Descripción: Verifica si las dos secuencias tienen los mismos elementos, sin importar el orden o la cantidad de apariciones.

Ejercicio 4

Si al resolver el ejercicio 3 no usaste **set**, resolverlo nuevamente con este módulo.

Ejercicio 5

CONTAR_APARICIONES(**in** $s : \text{secu}(\text{int})$) $\rightarrow res : \text{dict}(\text{int}, \text{int})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{$

- clave correctas: $(\forall x : \text{int}) (\text{pertenece}(x, s) \iff x \in \text{claves}(res))$
- apariciones correctas: $(\forall x : \text{int}) (x \in \text{claves}(res) \Rightarrow \text{obtener}(x, res) = \text{cantidad_apariciones}(x, s))$

$\}$

Descripción: Genera un diccionario donde cada elemento distinto que aparece en la secuencia de entrada se asocia con la cantidad de apariciones del mismo, siempre y cuando aparezca al menos una vez.

$\text{cantidad_apariciones} : \alpha \times \text{secu}(\alpha) \rightarrow \text{nat}$

$\text{cantidad_apariciones}(x, s) \equiv \text{if } \text{long}(s) = 0 \text{ then}$

0

else

if $\text{prim}(s) = x$ **then** 1 **else** 0 **fi** $+ \text{cantidad_apariciones}(x, \text{fin}(s))$

fi

Ejercicio 6

FILTRAR_REPETIDOS(**in** $s : \text{secu}(\text{int})$) $\rightarrow res : \text{secu}(\text{int})$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{(\forall x : \text{int}) (\text{pertenece}(x, res) \iff \text{pertenece}(x, s) \wedge \text{cantidad_apariciones}(x, s) = 1)\}$

Descripción: Elimina los elementos con más de una aparición de la lista de entrada.

Ejercicio 7: For-range

Implementar la intersección de conjuntos.

INTERSECCION(**in** $a : \text{conj}(\text{int})$, **in** $b : \text{conj}(\text{int})$) $\rightarrow res : \text{conj}(\alpha)$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{(\forall x : \alpha) (x \in res \iff x \in a \wedge x \in b)\}$

Descripción: Intersección de los conjuntos a y b .

Ejercicio 8

AGRUPAR_POR_UNIDADES(**in** $s : \text{secu}(\text{int})$) $\rightarrow res : \text{dict}(\text{int}, \text{conj}(\text{int}))$

Pre $\equiv \{\text{Sin repetidos: } (\forall i, j : \text{nat}) (i, j < \text{long}(s) \Rightarrow s[i] = s[j] \iff i = j)\}$

Post $\equiv \{$

- Están todos los elementos: $(\forall x : \text{int}) (\text{pertence}(x, s) \iff (\exists y : \text{int}) (\text{def?}(y, res) \wedge x \in \text{obtener}(y, res)))$
- Están en la definición correcta: $(\forall k : \text{int}) (k \in \text{claves}(res) \Rightarrow (\forall s : \text{int}) (\text{pertenece}(s, \text{obtener}(k, res)) \Rightarrow s \bmod 10 = k))$
- No hay grupos vacíos: $(\forall k : \text{int}) (k \in \text{claves}(res) \Rightarrow \neg \emptyset?(\text{obtener}(k, res)))$

$\}$

Descripción: Agrupa los elementos de la secuencia según el dígito menos significativo.

Ejercicio 9

TRADUCIR(**in** $tr : \text{secu}(\text{tupla}(\text{char}, \text{char}))$, **in** $str : \text{secu}(\text{char})$) $\rightarrow res : \text{secu}(\text{char})$

Pre $\equiv \{\text{Sin claves repetidas: } (\forall i, j : \text{nat}) (i, j < \text{long}(tr) \Rightarrow \pi_1(tr[i]) = \pi_1(tr[j]) \iff i = j)\}$

Post $\equiv \{$

- Mismos elementos: $\text{long}(res) = \text{long}(str)$
- Traducción: $(\forall i : \text{nat}) (i < \text{long}(str) \Rightarrow (\exists j : \text{nat}) (j < \text{long}(tr) \wedge \pi_1(tr[j]) = str[i] \wedge res[i] = \pi_2(tr[j])) \vee (\forall j : \text{nat}) (j < \text{long}(tr) \wedge \pi_1(tr[j]) \neq str[i] \wedge str[i] = res[i]))$

$\}$

Descripción: Traduce el string str caracter por caracter usando las asociaciones en tr . Si tr no tiene ninguna asociación, el caracter queda igual.

Ejercicio 10: Algobot

En Algoritmos y Estructuras de Datos 2 los talleres y TPs se entregan por mail. En el mail se pone como asunto la lista de libretas universitarias separadas por ;. Por ejemplo, un equipo de dos personas pondría como asunto: 103/92;05/04. Además, el mail tiene como adjunto los archivos de la entrega.

Quien se encarga de procesar estos mails es *el algobot*. Entre sus capacidades, el algobot puede conectarse al gmail de la materia y bajar los mails sin leer que hayan llegado. Una vez bajados estos mails es necesario procesarlos. En los siguientes ejercicios vamos a implementar algunos de estos procesos. Para ello, vamos a modelar parte del procesamiento de mails usando tres TADs: *Mail*, *Fecha* y *LU*. Para cada uno de los TADs vamos a tener una implementación en C++ que vamos a utilizar en los ejercicios siguientes. En el apéndice de este enunciado se encuentran los TADs y las interfaces de los módulos en C++ (ver al final).

Una de las primeras tareas que realiza el algobot con los mails con las entregas de TPs, es revisar que no haya ningún integrante que participe en más de un grupo. Considerando que cada mail tiene un conjunto de LUs en su asunto, tenemos que revisar que estas no se repitan. No obstante, un mismo grupo puede enviar varias entregas antes del horario de entrega, por lo que las libretas pueden repetirse pero siempre en conjuntos iguales. El problema a resolver es decidir si, dado una lista de mails, hay algún integrante que haya aparecido en más de un grupo.

INTEGRANTES_REPETIDOS(**in** $s : \text{secu}(\text{Mail})$) $\rightarrow res : \text{bool}$
Pre $\equiv \{(\forall m : \text{Mail}) (\text{pertenece}(m, s) \Rightarrow \text{asunto_valido}(m))\}$
Post $\equiv \{res = \neg (\forall i, j : \text{nat}) (i, j < \text{long}(s) \wedge i \neq j \Rightarrow \text{libretas}(s[i]) = \text{libretas}(s[j]) \vee \text{libretas}(s[i]) \cap \text{libretas}(s[j]) = \emptyset)\}$
Descripción: Devuelve true si entre los mails hay dos grupos distintos con integrantes en común.

Ejercicio 11

Por motivos varios, es común que los grupos entreguen los TPs varias veces antes del horario final. Por ejemplo, entre la primera y la última entrega encuentran errores, los corrigen y envían un nuevo mail. Otro error común es enviar el mail sin archivos adjuntos y luego mandar otro mail que sí tiene la entrega. Considerando esto, podría pasar que haya primero un mail del grupo con la entrega con errores, luego un mail sin adjuntos y luego el mail con la entrega corregida. Por este motivo, el algobot, al procesar todos los mails con entregas, debe quedarse con el último mail de cada grupo que tiene adjuntos. Esto nos da la siguiente especificación.

ENTREGAS_FINALES(**in** $s : \text{secu}(\text{Mail})$) $\rightarrow res : \text{dicc}(\text{conj}(\text{LU}), \text{Mail})$
Pre $\equiv \{(\forall m : \text{Mail}) (\text{pertenece}(m, s) \Rightarrow \text{asunto_valido}(m))\}$
Post $\equiv \{$

- Están los grupos con entregas: $(\forall g : \text{conj}(\text{LU})) (g \in \text{claves}(res) \iff (\exists i : \text{nat}) (i < \text{long}(s) \wedge \text{libretas}(s[i]) = g \wedge \text{adjunto}(s[i])))$
- El mail asociado es el último con adjunto: $(\forall g : \text{conj}(\text{LU})) (g \in \text{claves}(res) \Rightarrow (\forall i : \text{nat}) (i < \text{long}(s) \Rightarrow (\neg \text{adjuntos}(s[i]) \vee \text{libretas}(s[i]) \neq g) \vee (\text{fecha}(s[i]) \leq \text{fecha}(\text{obtener}(g, res))))))$

Descripción: Devuelve un diccionario que asocia, a cada grupo que haya hecho alguna entrega con adjuntos, el último mail que envió con algún adjunto.

Apéndice

TADs

TAD MAIL

géneros mail

igualdad observacional

$$(\forall m_1, m_2 : \text{mail}) \left(m_1 =_{\text{obs}} m_2 \iff \begin{pmatrix} \text{asunto}(m_1) & =_{\text{obs}} \\ \text{asunto}(m_2) & \wedge_{\text{L}} \\ \text{fecha}(m_1) & =_{\text{obs}} \\ \text{fecha}(m_2) & \wedge_{\text{L}} \\ \text{adjuntos}(m_1) & = \\ \text{adjuntos}(m_2) & \end{pmatrix} \right)$$

observadores básicos

asunto : mail \longrightarrow string

fecha : mail \longrightarrow fecha

adjuntos : mail \longrightarrow bool

generadores

nuevoMail : string asunto \times fecha fecha \times bool adjuntos \longrightarrow mail

otras operaciones

asunto_valido : mail \longrightarrow bool

libretas : mail m \longrightarrow conj(LU) {asunto_valido(m)}

axiomas $\forall s, f, b: \text{string, fecha, bool}$

asunto(nuevoMail(s, f, b)) \equiv s

fecha(nuevoMail(s, f, b)) \equiv f

adjuntos(nuevoMail(s, f, b)) \equiv b

Por simplicidad y falta de relevancia para el enunciado, no escribimos la axiomatización de las operaciones `libretas` y `asunto_valido`

Fin TAD

TAD LU

géneros lu

igualdad observacional

$$(\forall lu_1, lu_2 : \text{lu}) \left(lu_1 =_{\text{obs}} lu_2 \iff \begin{pmatrix} \text{numero}(lu_1) & =_{\text{obs}} \\ \text{numero}(lu_2) & \wedge_{\text{L}} \\ \text{año}(lu_1) =_{\text{obs}} \text{año}(lu_2) \end{pmatrix} \right)$$

observadores básicos

numero : lu \longrightarrow nat

año : lu \longrightarrow nat

generadores

nuevaLU : nat num \times nat a \longrightarrow lu

otras operaciones

$\bullet = \bullet$: lu \times lu \longrightarrow bool

axiomas $\forall n, a, l_1, l_2 : \text{nat}, \text{nat}, \text{lu}, \text{lu}$

numero(nuevaLU(n,a)) \equiv n

año(nuevaLU(n,a)) \equiv a

$lu_1 = lu_2 \equiv \text{numero}(lu_1) = \text{numero}(lu_2) \wedge \text{año}(lu_1) = \text{año}(lu_2)$

Fin TAD

TAD FECHA

géneros fecha

otras operaciones

$\bullet < \bullet$: fecha \longrightarrow bool

Por simplicidad y falta de relevancia para el enunciado, consideramos Fecha como un tad que representa un momento en el tiempo y puede compararse con el operador $<$.

Fin TAD

Clases C++

Mail

```
class Mail {
public:
    string asunto();
    Fecha fecha();
    bool adjunto();
    // PRE: El asunto tiene un formato válido para extraer libretas
    set<LU> libretas();
};
```

LU

```
class LU {  
public:  
    int numero();  
    int anio();  
};
```

Fecha

```
class Fecha {  
public:  
    // Operador para comparar tiempos.  
    bool operator<(Fecha otro);  
};
```