Ejercitación: Uso de clases

La ejercitación consiste en completar las funciones especificadas. Todos los ejercicios pueden resolverce en 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. Especifiaciones 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 pararse en el directorio raíz del projecto. En este debería haber un archivo CMakeLists.txt.
2. Ejecutar el comando $> cmake . Esto generará el 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.

Ejercicio 1

QUITAR_REPETIDOS(secu(int) s) $\rightarrow$ *res* : secu(int)
Pre $\equiv$ {true}
Post $\equiv$ {

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

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

Ejercicio 2

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

Ejercicio 3

MISMOS_ELEMENTOS(secu(int) a, secu(int) b) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res $\iff$ ($\forall$ x : int) (pertenece(x, a) $\iff$ pertenece(x, b))}
Descripción: Verifica si las dos secuencias tienen los mismos elementos, sin importar el órden 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(secu(int) s) $\rightarrow$ *res* : `dict(int, int)`
Pre $\equiv$ {true}
Post $\equiv$ {

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

cantidad_apariciones : α x $\times$ secu(α) s $\longrightarrow$ Nat

cantidad_apariciones(x, s) $\equiv$ **if** long(s) == 0 **then** 0 **else**
if prim(s) == x **then**
1
else
0 **fi** +
cantidad_apariciones(x, fin(s))
fi

Ejercicio 6

FILTRAR_REPETIDOS(secu(int) s) $\rightarrow$ *res* : `secu(int)`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ x : int) (pertenece(x, res) $\iff$ pertenece(x, s) $\wedge$ 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(conj(int) a, conj(int) b) $\rightarrow$ *res* : `conj(`α`)`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ x : α) (x $\in$ res $\iff$ x $\in$ a $\wedge$ x $\in$ b)}
Descripción: Intersección de los conjuntos a y b.

Ejercicio 8

AGRUPAR_POR_UNIDADES(secu(int) s) $\rightarrow$ *res* : `dict(int, conj(int))`
Pre $\equiv$ {Sin repetidos: ($\forall$ i, j : Nat) (i, j < long(s) $\Rightarrow$ s[i] == s[j] $\iff$ i == j)}
Post $\equiv$ {

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

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

Ejercicio 9

TRADUCIR(secu(tupla(char, char)) tr, secu(char) str) $\rightarrow$ *res* : `secu(char)`
Pre $\equiv$ {Sin claves repetidas: ($\forall$ i, j : Nat) (i, j < long(tr) $\Rightarrow$ π_1(tr[i]) == π_1(tr[j]) $\iff$ i == j)}
Post $\equiv$ {

- Mismos elementos: long(res) == long(str)
- Traducción: ($\forall$ i : Nat) (i < long(str) $\Rightarrow$
  ($\exists$ j : Nat) (j < long(tr) $\wedge$ π_1(tr[j]) == str[i] $\wedge$ res[i] == π_2(tr[j]))$\vee$
  ($\forall$ j : Nat) (j < long(tr) $\wedge$ π_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 envíar 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(secu(Mail) s) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {($\forall$ m : Mail) (pertenece(m, s) $\Rightarrow$ asunto_valido(m))}
Post $\equiv$ {res == $\neg$ ($\forall$ i, j : Nat) (i, j < long(s) $\wedge$ i $\neq$ j $\Rightarrow$
libretas(s[i]) == libretas(s[j]) $\vee$ libretas(s[i]) $\cap$ libretas(s[j]) == $\emptyset$)}
Descripción: No hay grupos en los mails que repiten integrantes y no son el mismo grupo.

Ejercicio 11

Por motivos varios, es común que los grupos entregen 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 si 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(secu(Mail) s) $\rightarrow$ *res* : `dicc(conj(LU), Mail)`
Pre $\equiv$ {($\forall$ m : Mail) (pertenece(m, s) $\Rightarrow$ asunto_valido(m))}
Post $\equiv$ {

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

}
Descripción: Asociar para cada grupo que haya hecho alguna entrega con adjuntos, el último mail que envió con algun archivo.

Apendice

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 $\forall s, f, b$: 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 $\forall n, a, l_1, l_2$: nat,nat,lu,lu

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

año(nuevaLU(n,a)) $\equiv$ a

$\text{lu}_1 == lu_2 \equiv \text{numero}(lu_1) == \text{numero}(lu_2) \wedge \text{año}(lu_1) == \text{año}(lu_2)$

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 adjuntos();
  // 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);
};
```