Algoritmos y Estructuras de Datos
Primer Parcial – Viernes 9 de mayo de 2025

Libreta	Apellido y Nombre	E1	E2	E3	E4	Nota Final	Corrigió
1		30	9	20	30	89	

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, más los dos apuntes del campus
- Cada ejercicio debe entregarse en **hojas separadas**
- Incluir en cada hoja el número de libreta, número de hoja, apellido y nombre
- El parcial se aprueba con 60 puntos y al menos 2 preguntas teóricas correctas

E1. TADs y especificación de problemas [40 pts]

1) La Panadería *El Progreso* recibe todos los días panes, facturas, masas, y otras delicias para venderla a sus clientes. Una vez que cierra sus puertas todo lo que no se vendió se envía a un comedero del barrio, por lo que todos los días se empieza con un stock completamente nuevo.
Los clientes son atendidos por estricto orden de llegada y realizan su pedido a algún empleado libre. Si no hay mercadería suficiente disponible para cumplir su pedido se retiran con las manos vacías. Caso contrario se le entrega y luego esperan para pagar, también en orden de llegada. Una vez que pagaron se retiran a disfrutar sus panificados.
Nos piden modelar en un TAD el funcionamiento de El Progreso según esta descripción, teniendo en cuenta que nos importa en todo momento saber cuántos clientes están esperando que los atiendan y cuántos están esperando para pagar.
   a) Indique las operaciones (procs) del TAD con todos sus parámetros y los renombres de tipo que considere necesarios.
   b) Describa el TAD en forma completa, indicando sus observadores, los requiere y asegura de las operaciones. Puede agregar los predicados y funciones auxiliares que necesite, con su correspondiente definición.
   c) Cuando presentamos nuestro tad nos dicen que ahora quieren saber qué empleado atiende más clientes cada día. ¿Debería modificar su TAD para reflejar esto? ¿Cómo? Responda en palabras, en forma breve y precisa.

2) Suponiendo que no cuenta con la operación *setKey*, complete la especificación de la operación **definir** en el siguiente TAD

TAD `Diccionario ⟨ K, V ⟩` {
  obs `data:` $dict(K, V)$
  proc `nuevoDiccionario` () : *Diccionario*(K, V) {
    asegura $\{res.data = \langle\rangle\}$
  }
  `// define la clave en el diccionario, la agrega si no existía`
  proc `definir` (inout d: *Diccionario*(K, V), in k: K, in v: V) {
    ...
  }
}

E2. Preguntas teóricas [10 pts]

Responder las siguientes preguntas **sin justificar** su respuesta.

1) ¿Qué programas S hacen válida la tripla de Hoare $\{True\}$ S $\{True\}$?
2) Si el cuerpo de un ciclo es un condicional y en el `then` la función variante propuesta se reduce estrictamente pero en el `else` puede quedar inalterada cuando se lo ejecuta, ¿Qué sucedería al aplicar el teorema del variante? ¿Cómo se puede arreglar?
   a) El ciclo termina y no hay nada que arreglar.
   b) El ciclo no termina, y hay que agregar un decremento de la función variante.
   c) El ciclo podría no terminar y se debería modificar el else para que se reduzca la función variante.
   d) El ciclo podría no terminar, hay que modificar la guarda del ciclo.
3) ¿Puede ocurrir que un invariante I satisfaga los puntos 1 y 3 del teorema del invariante, valga cada vez que termina de ejecutar S (el cuerpo del ciclo) pero no satisfaga el punto 2?
   a) Sí
   b) No
   c) Depende de la precondición del ciclo
   d) Depende de la postcondición del ciclo

HOJA 1/5

4) Si un invariante I cumple los puntos 1 y 2 del Teorema del Invariante pero no el 3, ¿qué tiene más sentido?

a) Debilitarlo
b) Ni debilitarlo ni fortalecerlo, cambiarlo por uno diferente.
c) Fortalecerlo
d) Debilitarlo en la precondición y fortalecerlo en la postcondición.

5) ¿Por qué la garantía del teorema del invariante no se expresa como tripla de Hoare y se habla de correctitud parcial?

a) Porque los ciclos no terminan.
b) Porque nos alcanza con la función variante.
c) Porque no sabemos cuántas veces se va a ejecutar.
d) Porque no tenemos una cota a la cantidad de veces que se va a ejecutar.

E3. Precondición más débil [20 pts]

Para los siguientes algoritmos **S** con sus post condiciones Q, indique cuál de las precondiciones propuestas es la *Precondición más débil* y justifique muy brevemente en palabras.

1) S $\equiv$

```
if (a mod 2 = 0)
    a := |a| + 1
else
    a := |a| * 2
endif
```

$Q \equiv \{a \bmod 2 = 0\}$

a) $P \equiv \{a \bmod 2 = 0\}$
b) $P \equiv \{a \bmod 2 \neq 0\}$
c) $P \equiv \{True\}$

2) S $\equiv$

```
j := i - 2
s[j] := 2 * i
```

$Q \equiv \{(\forall k : \mathbb{Z})(0 \leq k < |s| \rightarrow_L s[k] \bmod 2 = 0)\}$

a) $P \equiv \{(\forall k : \mathbb{Z})(0 \leq k < |s| \land k \neq i - 2 \rightarrow_L s[k] \bmod 2 = 0)\}$

$2 \leq i \leq |s|+2$

b) $P \equiv \{2 \leq i < |s| \land (\forall k : \mathbb{Z})(0 \leq k < |s| \land k \neq i - 2 \rightarrow_L s[k] \bmod 2 = 0)\}$
c) $P \equiv \{i \bmod 2 = 0 \land 2 \leq i < |s| \land (\forall k : \mathbb{Z})(0 \leq k < |s| \land k \neq i - 2 \rightarrow_L s[k] \bmod 2 = 0)\}$

3) S $\equiv$

```
if (x > y)
    y := x
else
    y := 3
endif
```

$Q \equiv \{y > 0\}$

a) $P \equiv \{x > y\}$
b) $P \equiv \{x > y \lor x \leq y\}$
c) $P \equiv \{(x > 0 \land x > y) \lor (x \leq y)\}$

4) S $\equiv$ NO HACER

```
if (i mod 2 = 0)
    s[i] := 1
else
    s[i] := 5
endif
```

$Q \equiv \{(\forall i : \mathbb{Z})(0 \leq i < |s| \land i \bmod 2 \rightarrow_L s[i] = 1)\}$

=0

a) $P \equiv \{0 \leq i < |s| \land i \bmod 2 = 0 \land (\forall j : \mathbb{Z})(0 \leq j < |s| \land j \bmod 2 = 0 \land j \neq i \rightarrow_L s[j] = 1)\}$
b) $P \equiv \{0 \leq i < |s| \land i \bmod 2 = 0 \land (\forall j : \mathbb{Z})(0 \leq j < |s| \land j \neq i \rightarrow_L s[j] = 1)\}$
c) $P \equiv \{i \bmod 2 = 0 \land (\forall j : \mathbb{Z})(0 \leq j < |s| \land j \bmod 2 = 0 \land j \neq i \rightarrow_L s[j] = 1)\}$

este item aclararon que no tenia solucion correcta

E4. Correctitud del ciclo [30 pts]

Dado el siguiente programa con su especificación

$P_c \equiv \{n > 0 \land res = 1 \land i = 1\}$

```
while(i < n)
    res := res * i
    i := i+1;
endwhile
```

$Q_c \equiv \{res = (n-1)!\}$

a) Escriba el Invariante del ciclo
b) Demuestre formalmente que el invariante propuesto cumple los axiomas del Teorema del Invariante
c) Decida si las siguiente funciones pueden o no usarse como funciones variatnes para demostrar la terminación del ciclo y justifique muy brevemente por qué.

I. $f_v = n$
II. $f_v = n - i$
III. $f_v = n - 1 - i$
IV. $f_v = i - n$
V. $f_v = n! - res$

E1- TADs

(A) y (B) Factura ES Int

Stock ES Int

Cliente ES Int

TAD Panadería {

obs facturas: dict<Factura, Stock>

obs filaPedir: seq<Cliente>

obs filaPagar: seq<Cliente>

proc clienteLlega(inout p: Panadería, in c: Cliente) {

requiere $\{c \notin p.filaPedir \land c \notin p.filaPagar\}$

requiere ~~[illegible]~~ $\{p = P_0\}$

asegura $\{p.facturas = P_0.facturas \land p.filaPagar = P_0.filaPagar\}$

asegura $\{p.filaPedir = P_0.filaPedir ++ \langle c \rangle\}$

}

proc tomarPedido(inout p: Panadería, in pedido: dict<Factura, int>) {

requiere $\{|p.filaPedir| > 0\}$

requiere $\{p = P_0\}$

requiere $\{(\forall f: \mathbb{Z})(f \in pedido \rightarrow_L pedido[f] > 0)\}$

asegura $\{(\exists f: \mathbb{Z})((f \in pedido \land f \in p.facturas) \land_L pedido[f] > facturas[f]$

$\rightarrow_L (p.facturas = P_0.facturas \land p.filaPagar = P_0.filaPagar \land$

$p.listaPedir = \text{tail}(P_0.listaPedir))\}$

asegura $\{(\forall f: \mathbb{Z})((f \in Pedido \land f \in p.facturas) \rightarrow_L$

$\times\ \underline{p.facturas} = \text{SetKey}(P_0.facturas, f, \cancel{[illegible]}\ P_0.facturas[f] - pedido[f]$

$\land\ p.listaPedir = \text{tail}(P_0.listaPedir) \land p.listaPagar = P_0.listaPagar ++ \text{head}(P_0.listaPedir))\}$

si f no es único, p. facturas toma valores contradictorios (para ≠ f p. facturas varía)

}

En tomar pedido, agarro al último cliente junto a su pedido. Si no hay stock lo saco y mantengo el estado igual en los otros obs.

Si hay stock, le vendo lo mando a pagar y actualizo.

proc clientepaga (inout p: Panaderia) {
requiere {|p. listaPagar| > 0}
~~requiere~~ requiere {p = P_0}
asegura {p.facturas = P_0.facturas ∧ p.listaPedir = P_0.listaPedir}
asegura {p.listaPagar = tail (P_0.listaPagar)}
}

proc nuevoDia (in facturas: dict<factura, stock>): Panaderia {
requiere {($\forall f: \mathbb{Z}$)(f ∈ facturas $\xrightarrow[L]{}$ facturas[f] > 0)}
asegura {res.facturas = facturas ∧
res.listaPedir = <> ∧
res.listaPagar = <>}
}

}

(c) Para saber qué empleado atiende mas clientes debería modificar mi TAD.
añadiría un obs empleados: conj<<id: int, ocupado: bool, clientesHoy: int>> struct
Así, al tomar un cliente debería buscar un empleado libre, cambiarlo a ocupado y sume uno a su cantidad de clientes hoy.
Luego, añadiría un proc mejorEmpleado (inout p: Panaderia): int que itere el conjunto para buscar quién atendió más.

- Último item en siguiente hoja.

Algoritmos y Estructuras de Datos - Primer Parcial -

E2) Preguntas Teóricas (sin justificar la respuesta)

① Cualquier programa S vuelve válida la tripla de Hoare puer tanto la precondición como la postcondición son True. En un estado de True, podría ejecutarse cualquier programa que seguiría valiendo la postcondición. → Todos (o) que terminan

② La respuesta correcta es la ⓒ, el ciclo podría no terminar y se debería modificar el "else" para que reduzca la fv. ✓

③ No ✓

④ Ni debilitarlo ni fortalecerlo, cambiarlo por uno diferente ✓

⑤ Porque no tenemos una cota a la cantidad de veces que se va a ejecutar ✓

② Completar definir, sin usar SetKey.

TAD Diccionario <K,V> {

obs data: dict<K,V>

proc NuevoDiccionario() : Diccionario<K,V> {

asegura {res.data = <>}

}

proc definir (inout d: Diccionario<K,V>, in k:K, in v:V) {

requiere {d = Do}

asegura { ¬(k∈d.data) → (|d.data| = |Do.data| + 1 ∧ k∈d.data ∧_L d.data[k] = v}

asegura { k∈d.data →_L d.data[k] = v ∧ |d.data| = |Do.data|}

asegura {(∀k2:K)(k2 ∈ d.data ∧ k2∉k) →_L d.data[k] = Do.data[k]}

}

#Me fijo, en ambos casos, que el resto de keys se mantenga igual.
#Me fijo que no se quiten o agreguen llaves de manera descontrolada, únicamente
#si quiero definir una nueva.

E3 - PRECONDICIÓN MAS DÉBIL

HOJA 4/5

① $S \equiv$ if (a Mod 2 = 0) $Q \equiv \{a \text{ Mod } 2 = 0\}$
  $a := |a| + 1$
else
  $a := |a| * 2$
endif

Respuesta: $Wp(S, Q) \equiv \{a \text{ Mod } 2 \neq 0\}$ porque, si a cumple la guarda, tenemos que $a \equiv 0(2)$ y se le toma el absoluto y aumenta en 1, lo que lleva a $a \equiv 1(2)$ y no cumple Q. Tampoco la Wp puede ser True, hay que limitar la entrada al then para todo a.

② $S \equiv j := i - 2$ (S1) $Q \equiv \{(\forall k:\mathbb{Z})(0 \leq k < |s| \rightarrow_L s[k] \text{ Mod } 2 = 0)\}$
  $s[j] := 2 * i$ (S2)

$Wp(S,Q) \equiv Wp(S1, \underbrace{\underbrace{Wp(S2, Q)}_{①})}_{②} \Rightarrow$ ① $Wp(s[j] := 2*i, Q) \equiv (\text{def}(1) \wedge \text{def}(s[j])) \wedge_L Q^{s}_{\text{SetAt}(s,j,2*i)}$

$\equiv \{(\forall k:\mathbb{Z})(0 \leq k < |s| \rightarrow_L \text{SetAt}(s, j, 2i)[k] \text{ Mod } 2 = 0)\}$

$\equiv \{(\forall k:\mathbb{Z})((0 \leq k < |s| \wedge k \neq j \rightarrow_L s[k] \text{ Mod } 2 = 0) \wedge$
  $(0 \leq k < |s| \wedge k = j \rightarrow_L 2*i \text{ Mod } 2 = 0))\}$

Se cumplirá siempre pues multiplica por dos.

$\equiv (\forall k:\mathbb{Z})(0 \leq k < |s| \wedge k \neq j \rightarrow_L s[k] \text{ Mod } 2 = 0) \equiv$ ①

luego $Wp(S,Q) \equiv Wp(j := i-2, \underbrace{①^{j}_{i-2}})$

Sé que la notación es confusa, me refiero a ①

$\equiv (\text{def}(j) \wedge \text{def}(i) \wedge \text{def}(s[i-2])) \wedge_L (\forall k:\mathbb{Z})(0 \leq k < |s| \wedge k \neq i-2 \rightarrow_L s[k] \text{ Mod } 2 = 0)$

⇒ Obs: Necesito que $i-2$ esté en rango, por eso $0 \leq i-2 < |s| \Rightarrow \boxed{2 \leq i < |s| + 2}$

Tanto la opción Ⓑ como Ⓒ respetan este rango, pero la Ⓒ tiene una condición de más que, por más que no rompe el programa, al ser más restrictivo con el i Mod 2 = 0 no puede ser nunca la Wp.

Por ejemplo, si $i = 1 \Rightarrow i \text{ Mod } 2 \neq 0 \wedge s[j] = 2*1 = 2 \wedge Q$.

Como b) es más débil que c), $\boxed{Wp(S,Q) \equiv b)}$

Respuesta

Scanned with CamScanner

③ $S \equiv$ if $(x>y)$ $\quad Q \equiv \{y>0\}$

$\quad y := x$

else

$\quad y := 3$

endif

OPCIONES

Ⓐ $P \equiv \{x>y\}$

Ⓑ $P \equiv \{x>y \vee x \leq y\}$

Ⓒ $P \equiv \{(x>0 \wedge x>y) \vee (x \leq y)\}$

→ Análisis de opciones

Ⓐ No garantiza nada, por ejemplo para $x=-10$, $y=-20$ tenemos $x>y$, entonces se le asigna a y el valor de x, que es menor a cero y no cumple la post.

Ⓑ Ya vimos que $x>y$ no significa nada, veamos la segunda opción $x \leq y$ logra que y sea positivo pues se le asigna el valor 3, pero esta P nos está diciendo que $x>y \wedge \neg(x \leq y) \Rightarrow y>0$, que para los valores $x=-1$ y $y=-2$ quedaría

$$(-1>-2) \wedge \neg(-1 \leq -2) \Rightarrow -2>0$$

$$\text{True} \wedge \text{True} \Rightarrow \text{False}$$ ⊗ DESCARTADA

③ Si $x>0 \wedge x>y$, para cualquier valor de y se va a cumplir la post pues le estamos asignando x que es positivo.

La otra opción es $(\neg(x>0 \wedge x>y)) \wedge (x \leq y) \equiv \neg B$ ✓

$\equiv x \leq y$. Esto nos llevaría al else, donde se le asigna 3 a y y se cumple la post.

$$\boxed{\text{Por el análisis, } wp(S,Q) \equiv (x>0 \wedge x>y) \vee (x \leq y) \quad \text{opción Ⓒ}}$$

④ No hacer, error en consigna

E4 - Correctitud de ciclo

$P_c \equiv \{N>0 \wedge rer=1 \wedge i=1\}$ $\quad Q \equiv \{rer=(N-1)!\}$

```
while(i<N) do
    res:=res*i
    i:=i+1
endwhile
```

TABLA DE VALORES PARA N=6

i	res	N
1	1	6
2	2	6
3	6	6
4	24	6
5	120	6
6	720	6

QUEDA i=N pero NO CUMPLE LA guarda, luego $1 \leq i \leq N$

(A) Escribir invariante de ciclo.

~~[illegible]~~

~~[illegible]~~

$\boxed{I \equiv 1 \leq i \leq N \wedge rer=(i-1)!}$

(B) Demostrar formalmente teorema del invariante cumplido por el encontrado

(1) $P_c \Rightarrow I$ VAMOS por partes $N>0 \wedge i=1 \Rightarrow 1 \leq i \leq N$ ✓

¿$res=1 \wedge i=1 \Rightarrow res=(i-1)!$?

$rer=(1-1)!=0!=1$ ✓ son iguales, se implican

Luego, $P_c \rightarrow I$

(2) $\{I \wedge B\} S \{I\}$ - VEAMOS SI $\{I \wedge B\} \rightarrow wp(S, I)$

$\Rightarrow wp(rer:=rer*i, wp(i:=i+1, I)) \equiv I^i_{i+1} \equiv$ (*)

~~[illegible]~~

~~[illegible]~~

(*) $I^i_{i+1} \equiv 1 \leq i+1 \leq N \wedge rer=(i+1-1)!$

$\equiv 0 \leq i \leq N-1 \wedge rer=i!$

AXIOMA DE ASIGNACIÓN, reemplazo

Luego $wp(rer=rer*i, 0 \leq i \leq N-1 \wedge rer=i!) \equiv 0 \leq i \leq N-1 \wedge \underbrace{rer*i=i!}$

luego $rer=(i-1)!$

$\Rightarrow$ ¿$\{I \wedge B\} \rightarrow 0 \leq i \leq N-1 \wedge rer=(i-1)!$?

$\underbrace{1 \leq i \leq N \wedge rer=(i-1)!}_{I} \wedge \underbrace{i<N}_{B} \rightarrow 0 \leq i \leq N-1 \wedge rer=(i-1)!$

OBS: LAS rer DE AMBOS LADOS SON IGUALES, SE IMPLICAN, LAS SACO.

$\{1 \leq i \leq N \wedge i < N \rightarrow 0 \leq i \leq N-1\}$

→ JUNTANDO LOS RANGOS, SE CUMPLE ②

③ $\{I \wedge \neg B\} \rightarrow Q$

$1 \leq i \leq N \wedge rer = (i-1)! \wedge i \geq N \rightarrow rer = (N-1)!$

REORDENO

AUX|

$1 \leq i \leq N \wedge i \geq N \rightarrow i = N$ LUEGO $i = N \wedge rer = (i-1)! \rightarrow rer = (N-1)!$ ✓

SE IMPLICAN PORQUE SON IGUALES.

⇒ QUEDA DEMOSTRADO QUE I CUMPLE EL TEOREMA DEL INVARIANTE Y EL PROGRAMA, SI TERMINA, CUMPLE LA POST.

ⓒ $fv = N$ → NO PORQUE NO ES DECRECIENTE

$\boxed{fv = N-i}$ → SI, PUEDE USARSE PUES ES DECRECIENTE Y VALE CERO AL FINAL DEL CICLO, NUNCA ANTES.

CUANDO SE DEJA DE CUMPLIR LA GUARDA, $fv = N-i \equiv 0$.

$fv = N-i-1$ → ES DECRECIENTE PERO SE PONE EN CERO CUANDO $i = N-1$, TODAVIA QUEDAN VALORES DE i POR LLEGAR $(i=N)$

$fv = i-N$ → ES CRECIENTE, TODO LO CONTRARIO A LO QUE QUEREMOS ENCONTRAR

$fv = N! - rer$ → ES DECRECIENTE, PERO rer LLEGA A MÁXIMO $(N-1)!$, NUNCA VALDRÁ CERO LA fv.