Trabajo práctico 2: Diseño v0.2

CalculadoraProgramable

Normativa

Límite de entrega: Domingo 3 de junio *hasta las 23:59 hs.* Enviar el zip al mail algo2.dc+tp2@gmail.com. Ver detalles de entrega abajo.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.

(http://campus.exactas.uba.ar)

1. Enunciado

Este TP consiste en diseñar módulos para implementar una calculadora programable similar al prototipo ya implementado en el TP anterior. Al igual que en el TP 1, un *programa* está dado por un conjunto de *rutinas*, cada una de las cuales consta de una lista de *instrucciones*, y hay ocho tipos de instrucciones (oPush, oAdd, oSub, oMul, oRead, oWrite, oJump, oJumpz). Las rutinas y variables se identifican por nombre (un string).

La principal diferencia con respecto al TP 1 es que la calculadora permite consultar el valor histórico de las variables. La calculadora comienza la ejecución en el instante 0. Cada vez que se ejecuta una instrucción, se incrementa en 1 el instante actual. La calculadora debe contar con una operación para observar cuál era el valor de una variable arbitraria x en un instante arbitrario $t \in \{0, 1, ..., \texttt{instante_actual}\}$.

La calculadora se inicializa con un parámetro entero W > 0, que llamamos **capacidad de ventana**. Se deben proveer las siguientes operaciones, con las complejidades temporales en **peor caso** indicadas:

- 1. Dado un programa p, una rutina r y una capacidad de ventana W, construir una calculadora inicializada para ejecutar la rutina r del programa $p O(\#p \cdot (|V| + |R|) + W \cdot \#V)$ donde:
 - #p es el tamaño del programa p, medido en cantidad total de instrucciones,
 - \bullet |V| es la longitud del nombre más largo de alguna de las variables que aparecen en el programa,
 - \blacksquare |R| es la longitud del nombre más largo de alguna de las rutinas que aparecen en el programa, incluyendo la rutina r,
 - ullet #V es la cantidad total de variables distintas que aparecen en el programa,
 - ullet W es la capacidad de ventana.
- 2. Determinar si la ejecución finalizó O(1).
- 3. Ejecutar la instrucción actual, actualizando correspondientemente el estado de la calculadora O(1). Nota: la restricción sobre la complejidad aplica a los ocho tipos de instrucciones por igual.
- 4. Darle valor a una variable arbitraria x O(|x|), donde |x| es la longitud del nombre de x. A diferencia de la operación owrite, este método **no** incrementa el instante actual.
- 5. Consultar el estado actual de la calculadora:
 - 5.1. Instante actual O(1).
 - 5.2. Nombre de la rutina actual O(1).
 - 5.3. Índice de la instrucción actual dentro de dicha rutina O(1).
 - 5.4. Valor de una variable arbitraria x en un instante arbitrario $t \in \{0, 1, ..., instante_actual\}$. La complejidad de esta operación depende de si se cumplen las dos condiciones siguientes:
 - 1) Acceso reciente. El instante t consultado está dentro del intervalo de tamaño W que llega hasta el instante actual (es decir, $t \ge \texttt{instante_actual} W$).
 - 2) Variable relevante. La variable x aparece en el código fuente del programa.

Si ambas condiciones se cumplen, la complejidad temporal debe ser $O(|x| + \log W)$ en peor caso, donde |x| es la longitud del nombre de x. Si no se cumple alguna de las dos condiciones anteriores, no hay ninguna restricción sobre la complejidad temporal de esta operación.

- 5.5. Valor de una variable arbitraria x en el instante actual O(|x|), donde |x| es la longitud del nombre de x.
- 5.6. Pila O(1).

Aclaraciones: No es necesario que la calculadora provea una operación para ejecutar un programa completo: eventualmente el usuario de la calculadora podría simular este comportamiento con un while. Para lograr la complejidad en peor caso de O(1) para todas las instrucciones, se sugiere convertir el programa que recibe la calculadora a una representación más conveniente, antes de la ejecución.

 $\{i < \log(s)\}$

1.1. Módulo Ventana(α)

Para diseñar la calculadora puede resultar útil utilizar un módulo VENTANA (α) con la interfaz que se detalla a continuación. Pueden utilizar el módulo VENTANA sin necesidad de diseñarlo.

El módulo Ventana (α) representa un arreglo en el que pueden registrarse elementos secuencialmente. Mantiene registro de los últimos W elementos registrados y "olvida" todos los anteriores. El parámetro W se llama la capacidad de la ventana. La especificación del TAD Ventana (α) puede consultarse en el apéndice.

iésimo : $secu(\alpha) s \times nat i \longrightarrow \alpha$

 $i\acute{e}simo(s, i) \equiv if i = 0$? then prim(s) else $i\acute{e}simo(fin(s), pred(i))$ fi

```
Interfaz del módulo Ventana(\alpha):
■ Parámetros formales: género \alpha.
• Géneros: ventana (\alpha).
• Se explica con: ventana(\alpha).
Operaciones:
         NuevaVentana(in W : nat) \rightarrow res : ventana(\alpha)
         Crea una nueva ventana con la capacidad W indicada.
         \mathbf{Pre} \equiv \{\hat{W} > 0\}
         \mathbf{Post} \equiv \{\widehat{res} =_{\mathrm{obs}} \mathrm{nuevaVentana}(\widehat{W})\}\
         Complejidad: O(W).
         REGISTRAR(in/out \ v : ventana(\alpha), in \ a : \alpha)
         Registra un elemento a al final de la ventana. Si la ventana excede la capacidad W, se mantiene
         registro de los últimos W elementos (incluyendo a) y se olvidan los anteriores.
         \mathbf{Pre} \equiv \{v_0 =_{\mathrm{obs}} \widehat{v}\}\
         \mathbf{Post} \equiv \{\hat{v} =_{obs} \operatorname{registrar}(v_0, \, \hat{a})\}\
         Complejidad: O(1).
         CAPACIDAD(in v: ventana(\alpha)) \rightarrow res: nat
         Devuelve la capacidad de la ventana.
         \mathbf{Pre} \equiv \{ \mathbf{true} \}
         \mathbf{Post} \equiv \{\widehat{\mathit{res}} =_{\mathrm{obs}} \mathrm{capacidad}(\widehat{v})\}\
         Complejidad: O(1).
         TAM(\mathbf{in}\ v : ventana(\alpha)) \rightarrow res : nat
         Devuelve el tamaño (cantidad de elementos) de la ventana. Es un entero entre 0 y W inclusive.
         \mathbf{Pre} \equiv \{ \mathbf{true} \}
         \mathbf{Post} \equiv \{\widehat{res} =_{obs} long(elementos(\widehat{v}))\}\
         Complejidad: O(1).
         \bullet[\bullet](in v: ventana(\alpha), in i: nat) \to res: \alpha
         Devuelve el i-ésimo elemento de la ventana. Si la ventana tiene tamaño n, el elemento 0-ésimo es el
         elemento más viejo del cual aún se tiene registro, y el elemento (n-1)-ésimo es el último elemento
         registrado.
         \mathbf{Pre} \equiv \{\hat{i} < \tan(\hat{v})\}\
         \mathbf{Post} \equiv \{\widehat{res} =_{obs} i\acute{e}simo(elementos(\hat{v}), \hat{i})\}
         Complejidad: O(1).
  La operación iésimo es la usual:
```

2. Documentación a entregar

Todos los módulos diseñados deben contar con las siguientes partes. Se debe diseñar el módulo principal (Calculadora) y todos los módulos auxiliares. La única excepción son los módulos disponibles en el Apunte de Módulos Básicos, que se pueden utilizar sin diseñarlos: lista enlazada, pila, cola, vector, diccionario lineal, conjunto lineal y conjunto acotado de naturales.

1. Interfaz.

- 1.1. Tipo abstracto ("se explica con ..."). Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación formal de la materia. Pueden utilizar la especificación que se incluye en el apéndice.
- 1.2. Signatura. Listado de todas las funciones públicas que provee el módulo. La signatura se puede escribir, dependiendo de sus preferencias:
 - Con la notación de módulos de la materia: apilar(in/out pila: PILA, in x: ELEMENTO)
 Con notación de C++: void Pila::apilar(const Elemento& x)
- 1.3. Contrato. Precondición y postcondición de todas las funciones públicas. Las precondiciones de la funciones de la interfaz deben estar expresadas **formalmente** en lógica de primer orden.¹
- 1.4. Complejidades. Complejidades de todas las funciones públicas, cuando corresponda.
- 1.5. Aspectos de aliasing. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay aliasing entre algunas de las estructuras.

2. Implementación.

- 2.1. Representación ("se representa con ..."). Módulo con el que se representan las instancias del módulo actual
- 2.2. Invariante de representación. Puede estar expresado en lenguaje natural o formal.
- 2.3. Función de abstracción. Puede estar expresada en lenguaje natural o formal.
- 2.4. Algoritmos. Pueden estar expresados en pseudocódigo, usando si es necesario la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.
- 3. **Servicios usados.** Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

Sobre el uso de lenguaje natural y formal.

Las precondiciones y poscondiciones de las funciones auxiliares, el invariante y la función de abstracción pueden estar expresados en lenguaje natural. No es necesario que sean formales. Asimismo, los algoritmos pueden estar expresados en pseudocódigo. Por otro lado, está permitido que utilicen fórmulas en lógica de primer orden en algunos lugares puntuales, si consideran que mejora la presentación o subsana alguna ambigüedad. El objetivo del diseño es convencer al lector, y a ustedes mismos, de que la interfaz pública se puede implementar usando la representación propuesta y respetando las complejidades pedidas. Se recomienda aplicar el sentido común para priorizar la claridad y legibilidad antes que el rigor lógico por sí mismo. Por ejemplo:

Más claro Menos claro

```
"Cada clave del diccionario D debe ser una lista sin elementos repetidos." (¿En qué estructura?). "SinRepetidos?(claves(D))" \checkmark

"Ordenar la lista A usando mergesort." \checkmark "Ordenar los elementos." (¿Qué elementos? ¿Cómo se "A.mergesort()" \checkmark "Miro las tuplas del conjunto, apilo la segunda componente en la primera y voy incrementando un contador." (Ambiguo y difícil de entender).
```

¹Si la implementación requiere usar funciones auxiliares, sus pre y postcondiciones pueden estar escritas en lenguaje natural, pero esto no forma parte de la interfaz.

Especificación formal $\mathbf{A}.$

Usaremos los siguientes tipos auxiliares. El tipo Int representa el tipo de las constantes numéricas que maneja la calculadora. Omitimos su especificación formal. Al implementarlo, esperamos que sea un entero de precisión acotada (un int en C++). El tipo STRING es un arreglo dimensionable de caracteres, extendido con una operación $\bullet = \bullet$: string \times string \to bool para compararlas por igualdad. Cada caracter es un número entre 0 y 255. El TAD OPERACIÓN representa un código de operación. Tiene 8 generadores y un único observador $\bullet = \bullet$: operación \times operación \rightarrow bool para compararlas por igualdad.

- **TAD** CHAR es ENUM(0, 1, ..., 255)
- TAD STRING es ARREGLO DIMENSIONABLE(CHAR)

≡ oJumpz

- TAD VARIABLE es STRING
- TAD RUTINA es STRING
- TAD OPERACIÓN es ENUM(oPush, oAdd, oSub, oMul, oRead, oWrite, oJump, oJumpz)

A.1. Instrucción

TAD INSTRUCCIÓN

```
instrucción
géneros
usa
               Int, Variable, Rutina, Operación
exporta
               observadores, generadores
observadores básicos
                         : instrucción
                                           — operación
                                                                                                 {\operatorname{op}(i) = \operatorname{oPush}}
  constante
Numérica : instrucción i
                                           \longrightarrow int
                                                                           {op(i) = oRead \lor op(i) = oWrite}
  {\bf nombre Variable}
                         : instrucción i \longrightarrow \text{variable}
  nombreRutina
                                                                           \{op(i) = oJump \lor op(i) = oJumpz \}
                         : instrucción i \longrightarrow rutina
generadores
  iPush
           : int
                           \rightarrow instrucción
  IADD
                            \rightarrow instrucción
  ISUB
                          → instrucción
  iMuL
                            → instrucción
                          \longrightarrow instrucción
  IREAD : variable
  IWRITE: variable
                         → instrucción
           : rutina
                          → instrucción
  I.JUMP
                           \rightarrow instrucción
  IJUMPZ: rutina
axiomas
(\forall i: \text{instrucción}, \forall n: \text{int}, \forall v: \text{variable}, \forall r: \text{rutina})
  op(IPUSH(n))
                                                                constanteNumérica(IPUSH(n)) \equiv n
                     ≡ oPush
  op(IADD)
                     \equiv oAdd
                                                                nombreVariable(IREAD(v))
  op(ISUB)
                     ≡ oSub
                                                                nombreVariable(IWRITE(v))
  op(IMUL)
                     ≡ oMul
  op(IREAD(v))
                     ≡ oRead
                                                                nombreRutina(IJUMP(r))
                                                                                                    \equiv r
  op(IWRITE(v))
                    ≡ oWrite
                                                                nombreRutina(IJUMPZ(r))
                                                                                                    \equiv r
  op(IJUMP(r))
                     ≡ oJump
  op(IJUMPZ(r))
```

A.2. PROGRAMA

TAD PROGRAMA

```
géneros
                  programa
                  CONJUNTO, RUTINA, INSTRUCCIÓN
usa
exporta
                  observadores, generadores
observadores básicos
   rutinas
                                                                     \longrightarrow conj(rutina)
                      : programa
                                                                                                                         \{r \in \operatorname{rutinas}(p)\}\
   longitud
                      : programa p \times rutina r
                                                                     \longrightarrow nat
   instrucción
                     : programa p \times \text{rutina } r \times \text{nat } n \longrightarrow \text{instrucción}
                                                                                            \{r \in \operatorname{rutinas}(p) \wedge_{\mathtt{L}} n < \operatorname{longitud}(p, r)\}
generadores
   nuevoPrograma
                                                                         \longrightarrow programa
   ag<br/>Instrucción : programa × rutina × instrucción \longrightarrow programa
axiomas
(\forall p: \text{programa}, \forall r, r': \text{rutina}, \forall i: \text{instrucción}, \forall n: \text{nat})
   rutinas(nuevoPrograma)
   rutinas(agInstrucción(p, r, i))
                                                          \equiv \operatorname{Ag}(r, \operatorname{rutinas}(p))
   longitud(agInstrucción(p, r, i), r') \equiv if r' \in \text{rutinas}(p) then longitud(p, r') else 0 fi +
                                                        if r = r' then 1 else 0 fi
   instrucción(agInstrucción(p, r, i), r', n) \equiv \mathbf{if} r = r' \wedge \text{longitud}(\text{agInstrucción}(p, r, i), r') = n+1 \mathbf{then}
                                                               else
                                                                    instrucción(p, r', n)
                                                               fi
```

A.3. CALCULADORA

TAD CALCULADORA

```
géneros
                calculadora
                Bool, Nat, Pila, Int, Variable, Rutina, Instrucción, Programa
usa
                observadores, generadores, ejecutando?, valorActualVariable
exporta
observadores básicos
  programa
                                : calculadora
                                                                              \rightarrow programa
  rutinaActual
                                : calculadora
                                                                               \rightarrow rutina
  índiceInstrucciónActual : calculadora
                                                                               \rightarrow nat
                                : calculadora
                                                                               \rightarrow pila(int)
                                : calculadora
                                                                                 nat
  instanteActual
  valorHistóricoVariable
                               : calculadora c \times \text{variable} \times \text{nat } t
                                                                                                   \{t \leq \text{instanteActual}(c)\}
                                                                            \longrightarrow int
generadores
  nueva
Calculadora : programa \times rutina
                                                               \longrightarrow calculadora
  ejecutarUnPaso
                        : calculadora c
                                                               → calculadora
                                                                                                           \{ejecutando?(c)\}
                         : calculadora \times variable \times int \longrightarrow calculadora
  asignarVariable
otras operaciones
  ejecutando?
                            : calculadora
                                                             \longrightarrow bool
  instrucciónActual
                            : calculadora c
                                                            → instrucción
                                                                                                           \{ejecutando?(c)\}
  opActual
                            : calculadora c
                                                            — operación
                                                                                                           \{e \text{ jecutando}?(c)\}
  haySalto?
                            : calculadora c
                                                             \longrightarrow bool
                                                                                                           \{ejecutando?(c)\}
  primeroPila
                            : calculadora
                                                            \longrightarrow int
                            : calculadora
                                                             \longrightarrow int
  segundoPila
  pilaSinUno
                            : calculadora
                                                            \longrightarrow pila(int)
  pilaSinDos
                            : calculadora
                                                            \longrightarrow pila(int)
  valorActualVariable : calculadora \times variable \longrightarrow int
  escribiendo
Variable? : calculadora \times variable \longrightarrow bool
axiomas
(\forall c: \text{ calculadora}, \forall t: \text{ nat}, \forall p: \text{ programa}, \forall r: \text{ rutina}, \forall v, v': \text{ variable}, \forall n: \text{ int})
  programa(nuevaCalculadora(p, r)) \equiv p
  programa(ejecutarUnPaso(c))
                                              \equiv \operatorname{programa}(c)
  \operatorname{programa}(\operatorname{asignarVariable}(c, v, n))
                                              \equiv \operatorname{programa}(c)
  rutinaActual(nuevaCalculadora(p, r))
  rutinaActual(ejecutarUnPaso(c))
                                                  \equiv if haySalto?(c) then
                                                          nombreRutina(instrucciónActual(c))
                                                      else
                                                          rutinaActual(c)
                                                      fi
  rutinaActual(asignarVariable(c,v,n))
                                                  \equiv \text{rutinaActual}(c)
  indiceInstrucciónActual(nuevaCalculadora(p, r))
  indiceInstrucciónActual(ejecutarUnPaso(c))
                                                               \equiv if haySalto?(c) then
                                                                   else
                                                                       1 + indiceInstrucciónActual(c)
  indiceInstrucciónActual(asignarVariable(c,v,n))
                                                               \equiv índiceInstrucciónActual(c)
  pila(nuevaCalculadora(p, r)) \equiv vacía
```

```
pila(ejecutarUnPaso(c))
      if
          opActual(c) = oPush
                                           _{
m then}
                                                    apilar(constanteNumérica(instrucciónActual(c)), pila(c))
           opActual(c) = oAdd
                                                    \operatorname{apilar}(\operatorname{segundoPila}(c) + \operatorname{primeroPila}(c), \operatorname{pilaSinDos}(c))
else if
                                           then
else if
           opActual(c) = oSub
                                           then
                                                    \operatorname{apilar}(\operatorname{segundoPila}(c) - \operatorname{primeroPila}(c), \operatorname{pilaSinDos}(c))
else if
           opActual(c) = oMul
                                                    \operatorname{apilar}(\operatorname{segundoPila}(c) * \operatorname{primeroPila}(c), \operatorname{pilaSinDos}(c))
                                           then
else if
          opActual(c) = oRead
                                           then
                                                    apilar(
                                                        valorActualVariable(c, nombreVariable(instrucciónActual(c))),
                                                        pila(c)
          opActual(c) = oWrite
                                                    pilaSinUno(c)
else if
                                           then
else if
          opActual(c) = oJump
                                                    pila(c)
                                           then
   else (opActual(c) = oJumpz)
                                                    pilaSinUno(c)
                                                                              fi fi fi fi fi fi
pila(asignarVariable(c,v,n)) \equiv pila(c)
instanteActual(nuevaCalculadora(p, r)) \equiv 0
instanteActual(ejecutarUnPaso(c))
                                                 \equiv 1 + \text{instanteActual}(c)
instanteActual(asignarVariable(c,v,n))
                                                 \equiv instanteActual(c)
valorHistóricoVariable(nuevaCalculadora(p, r), v)
valorHistóricoVariable(ejecutarUnPaso(c), v, t)
                                                               \equiv if instanteActual(c) = t then
                                                                       if escribiendoVariable?(v) then
                                                                           primeroPila(c)
                                                                       else
                                                                           valorHistóricoVariable(c, v, t-1)
                                                                       fi
                                                                   else
                                                                       valorHistóricoVariable(c, v, t)
valorHistóricoVariable(asignarVariable(c,v,n), v', t) \equiv if instanteActual(c) = t \wedge v = v' then
                                                                   else
                                                                       valorHistóricoVariable(c, v', t)
                                                                   fi
                       rutinaActual(c) \in rutinas(programa(c)) \land_{L}
ejecutando?(c) \equiv
                         indiceInstrucciónActual(c) < longitud(programa(c), rutinaActual(c))
instrucciónActual(c) \equiv instrucción(programa(c), rutinaActual(c), indiceInstrucciónActual(c))
opActual(c)
                    \equiv op(instrucciónActual(c)
                         op(instrucciónActual(c)) = oJump \lor
\text{haySalto}?(c)
                          (op(instrucciónActual(c)) = oJumpz \land_L primeroPila(c) = 0)
primeroPila(c)
                   \equiv if vacía?(pila(c)) then 0 else tope(pila(c)) fi
\operatorname{segundoPila}(c) \equiv \operatorname{if} \operatorname{vac\'a?}(\operatorname{pila}(c)) \vee_{\operatorname{L}} \operatorname{vac\'a?}(\operatorname{desapilar}(\operatorname{pila}(c))) \operatorname{then}
                        else
                            tope(desapilar(pila(c)))
pilaSinUno(c)
                    \equiv if vacía?(pila(c)) then vacía else desapilar(pila(c)) fi
pilaSinDos(c)
                    \equiv if vacía?(pila(c)) \vee_{\text{L}}vacía?(desapilar(pila(c))) then
                            vacía
                        else
                            desapilar(desapilar(pila(c)))
valorActualVariable(c, v) \equiv valorHistóricoVariable(c, v, instanteActual(c))
```

```
escribiendo
Variable?(c,v) \equiv \text{ejecutando}?(c) \land_{\text{L}} \text{opActual}(c) = \texttt{oWrite} \land_{\text{L}} \text{nombreVariable}(\text{instrucciónActual}(c)) = v
```

A.4. Ventana(α)

```
TAD Ventana(\alpha)
```

```
géneros
                   ventana
                   Nat
usa
exporta
                   observadores,\ generadores
observadores básicos
                        : ventana(\alpha)
   capacidad
                                                   \longrightarrow \ \mathrm{nat}
   elementos
                        : ventana(\alpha)
                                                   \longrightarrow \operatorname{secu}(\alpha)
generadores
                                                                                                                                       {c > 0}
   nueva
Ventana : na<br/>t\boldsymbol{c}
                                                   \longrightarrow ventana(\alpha)
   registrar
                       : ventana(\alpha) × \alpha \longrightarrow ventana(\alpha)
axiomas
(\forall v: \text{ventana}(\alpha), \forall a, b: \alpha, \forall i: \text{nat})
   capacidad(nuevaVentana(w)) \equiv w
   \operatorname{capacidad}(\operatorname{registrar}(v, a))
                                              \equiv \operatorname{capacidad}(v)
   elementos(nuevaVentana(w))
                                             ≡ <>
   elementos(registrar(v, a))
                                              \equiv if long(elementos(v)) < capacidad(v) then
                                                      elementos(v) \circ a
                                                  else
                                                       fin(elementos(v)) \circ a
                                                  fi
```