ESTRUCTURAS DE DATOS Y ALGORITMOS

CURSO 2009 – PRÁCTICO 7

1-
Definición e implementación de TAD's

a)
- *i.* Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos Lista de Naturales. Dar una especificación funcional mínima de constructores, predicados y selectores.
- *ii.* Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos Arbol Binario de Naturales. Dar una especificación funcional mínima de constructores, predicados y selectores.

b)
- *i.* Escribir en C/C++ un módulo de implementación para el Tipo Abstracto de Datos Lista de Naturales. Utilice la representación de lista encadenada simple de punteros e implemente las operaciones.
- *ii.* Escribir en C/C++ un módulo de implementación para el Tipo Abstracto de Datos Lista de Naturales. Utilice la representación de listas doblemente encadenadas de punteros e implemente las operaciones.
- *iii.* Escribir en C/C++ un módulo de implementación para el Tipo Abstracto de Datos Arbol Binario de Naturales. Utilice la representación clásica de punteros de Arboles Binarios e implemente las operaciones.

En los problemas que siguen, se usarán los tipos **lista y árbol binario** (en ambos casos, de naturales). Se entenderán como **tipos abstractos**, cada uno de ellos definido por un conjunto *mínimo* de *funciones* (constructoras, de predicados y selectoras)(Ej.1). Pruebe estos ejercicios en máquina. Escriba un módulo principal el cual importe los módulos de lista y árbol binario de naturales. Luego pruébelos con las diferentes implementaciones de los tipos abstractos.

2-
a) Escribir una función recursiva que determine si dos listas de naturales son o no iguales.
b) Escribir una función que determine si dos árboles binarios de naturales tienen o no la misma **frontera**. La frontera de un árbol binario es la lista de los nodos de éste que tienen a lo sumo un hijo, ordenados de izquierda a derecha según aparecen en la representación usual del árbol.

3-
a) Escribir un procedimiento recursivo llamado **Mitades**, que reciba una lista y reparta todos sus elementos en dos listas cuyos largos sumen el de la lista original y no difieran entre sí en más de uno.
   Ejemplo: si la lista de entrada es **[3, 11, 8, 2, 5]**, entonces la salida podrá consistir en las dos listas: **[3, 8, 5]** y **[11, 2]**.
b) Escribir una función recursiva llamada **IntercalOrd** que reciba dos listas ordenadas y las intercale de modo que el resultado sea una lista ordenada.

Ejemplo: si se reciben las listas **[3, 5, 8]** y **[2, 11]** entonces la salida deberá ser **[2, 3, 5, 8, 11]**.

c) Escribir una función recursiva que ordene cualquier lista dada, utilizando solamente **Mitades** e **IntercalOrd**.

4-

Considérense las siguientes operaciones, que especifican un Tipo Abstracto de Datos al que llamaremos de "Estructuras ordenadas" (abreviado *EstrOrd*):

- `NullEO : EstrOrd`
  `(* Es la estructura ordenada vacía *)`

- `InsertEO : Nat X EstrOrd -> EstrOrd`
  `(* Inserta un natural en una estructura ordenada, manteniéndola ordenada *)`

- `FlattenEO : EstrOrd -> Lista`
  `(* Forma una lista ordenada con todos los elementos de la estructura ordenada recibida como parámetro *)`

a) Escribir un módulo de especificación para el tipo EstrOrd y correspondientes módulos de implementación, que utilicen como representación para las estructuras ordenadas:
   i. Arboles binarios de búsqueda, implementados mediante punteros
   ii. Listas encadenadas simples ordenadas.
b) Escribir un procedimiento que ordene cualquier lista dada, usando sólo las primitivas del tipo abstracto EstrOrd además de las primitivas de listas.

5-

Considérense fórmulas que representan combinaciones de sumas y productos de naturales. Constituyen un tipo inductivo que llamaremos *Form* y que está definido por los siguientes constructores:

- **`Num:`** `Nat -> Form`
- **`Sum:`** `Form x Form -> Form`
- **`Prod:`** `Form x Form -> Form`

a) Escribir un módulo de especificación del TAD Form, conteniendo un conjunto *mínimo* de procedimientos constructores, de predicado, selectores y destructores.
b) Escribir un correspondiente módulo de implementación.
c) Escribir procedimientos *recursivos* que:

   i. Computen el valor de un fórmula dada.
   ii. Pase una fórmula dada a notación infija. El resultado será de tipo *String*. Se asumirá que éste es un tipo abstracto de cuyas operaciones se podrán usar:

   - Un procedimiento Concat que concatena dos strings dados.
   - Un procedimiento CardStr que convierte números naturales en strings.

La expresión infija de cada fórmula deberá contener el menor número posible de paréntesis, asumiendo la precedencia usual de operadores. Esto es: los productos se evalúan antes que las sumas y las secuencias de operaciones de igual prioridad se evalúan de izquierda a derecha.

6-

Una bolsa o multiconjunto (bag) es un tipo abstracto de datos que especifica una colección de elementos en la que, a diferencia del Conjunto, pueden existir ocurrencias repetidas de los mismos. Por ejemplo, la bolsa {1, 3, 3, 4} no es la misma bolsa que {1, 3, 4}.
Al igual que en el caso de conjuntos, sin embargo, el orden de los elementos es irrelevante, distinguiéndose así de la noción de lista.

Las operaciones del TAD Bolsa (de naturales) son las siguientes:

- **Vacia**, retorna una bolsa vacía.
- **Insertar**, dados dos naturales x y n, y una bolsa b, inserta n ocurrencias del elemento x en la bolsa b.
- **MinElem**, dada una bolsa b retorna el mínimo elemento de la misma (b no es vacía).
- **Borrar**, dados dos naturales x y n, y una bolsa b, borra n ocurrencias del elemento x en la bolsa b. Si hay menos de n ocurrencias de x en b, las borra todas.
- **Contar**, dado un natural x y una bola b retorna el número de ocurrencias de x en b.

a) Escribir un módulo de definición del TAD Bolsa (de cardinales) que se ha descripto.
b) Escribir en forma completa un correspondiente módulo de implementación, de forma que la representación de las bolsas sea una estructura de datos dinámica.

7-

Considere el problema de describir un simple editor de líneas. Suponga que una línea es una secuencia de caracteres *c1, c2, ..., cn* junto con un cursor p, donde 0 <= p <= n.

Las siguientes operaciones sobre líneas son requeridas:

- **NewLine**, crea una línea vacía.
- **Insert**, inserta un nuevo caracter en la posición indicada por el cursor.
- **MoveStart**, sitúa el cursor al comienzo de la línea.
- **MoveEnd**, sitúa el cursor al final de la línea.
- **MoveLeft**, mueve el cursor una posición a la izquierda (si es posible).
- **MoveRight**, mueve el cursor una posición a la derecha (si es posible).
- **Delete**, borra el caracter que se encuentra en la posición indicada por el cursor.

a) Escribir un módulo de definición del TAD Linea.
b) Sugiera una representación adecuada para este tipo abstracto y escriba el correspondiente módulo de implementación.