TECNÓLOGO EN INFORMÁTICA

ESTRUCTURAS DE DATOS Y ALGORITMOS

Segundo Parcial

17 de Julio de 2008

(Son 4 carillas)
Tiempo Total: 2:30 hs.

Generalidades:
· La prueba es individual y sin material.
· Sólo se contestan dudas acerca de la letra de los ejercicios.
· Escriba las hojas de un sólo lado y con letra clara.
· Comience la solución de cada ejercicio en una nueva hoja.
· Numere cada hoja, indicando en la primera el total.
· Coloque su número de cédula y nombre en cada hoja.

Ejercicio 1(30 pts.)- Se consideran disponibles los módulos Símbolo y Valor, cuyos módulos de especificación aparecen a continuación:

```
Módulo de definición del TAD Símbolo:

typedef struct nodoSimbolo*  Simbolo;

bool igualSimbolo(Simbolo s1, Simbolo s2) //Igualdad de símbolos

bool menorSimbolo(Símbolo s1, Simbolo s2) // retorna TRUE sii s1 < s2
…

Módulo de definición del TAD Valor:

typedef char[MAX] Valor;
…
```

Se quiere definir un TAD HTabla que representa una tabla donde se asocian valores a símbolos. Este TAD cumple las siguientes propiedades:

- Dado un símbolo cualquiera, este puede tener algún valor asociado en la tabla o no.
- Un símbolo es ingresado en la tabla la primera vez que se le asigna un valor.
- Un valor asociado a un símbolo es el último que se le asignó. Sin embargo es posible recuperar los valores previos (la historia).
- Existe una operación de "desasignar" un símbolo, que deshace la última asignación de ese símbolo volviéndolo a su valor anterior, si éste existía.

En resumen el TAD HTabla es un **diccionario** que tiene la capacidad de **recordar** los valores anteriores asignados a un símbolo (con historia).

Las operaciones del TAD HTabla son las siguientes:

- ***Crear*** - Crea la tabla vacía sin ningún símbolo asociado.
- ***Asignar*** - Asigna un valor a un símbolo. El valor anterior, si existía, queda guardado en la historia del símbolo.
- ***Buscar*** - Retorna el valor asignado a un símbolo. El valor de un símbolo es el último valor asignado. Si al símbolo no se le ha asignado ningún valor
- (no tiene historia), retorna el valor nulo.
- ***DesAsignar*** - Deshace la última asignación hecha sobre un símbolo. El símbolo queda con su valor anterior; el valor actual se pierde y no queda en la historia. Si el símbolo no tenía valor anterior, entonces queda sin valor asociado. Si el símbolo no figuraba en la tabla, la operación no tiene ningún efecto; esto es: deja la tabla sin modificar.
- ***ReIniciar*** - Todas las variables de la tabla quedan con el valor asignado por primera vez. La historia se pierde.

Se Pide:

a) Escribir un módulo de definición para el TAD HTabla. Las operaciones que afectan la tabla deben implementarse como funciones. Dicho de otro modo: **no** deben ser procedurales.

b) Escribir un correspondiente módulo de implementación para HTabla. Los símbolos que no tienen ningún valor asignado no deben almacenarse en la estructura, en particular la operación "*desasignar*" puede implicar borrar el elemento. Puede utilizar cualquiera de los TADs vistos en el curso sin necesidad de implementarloscon excepción del TAD ABB.

Notas:

- No debe preocuparse por el problema de liberar memoria.
- Si usa los TADs vistos en clase explique la razón de su uso, esto es: las propiedades del TAD que le son útiles para resolver la parte b).

Ejercicio 2(10 pts.)-Preguntas Múltiple Opción. Solo existe una **única** respuesta correcta. Marque con un **círculo** la **letra** de la respuesta correcta. Cada respuesta correcta suma **2** puntos, las respuestas incorrectas restan **0.5** puntos.

1. Un árbol finitario es aquel que tiene:
    a. Una cantidad finita de nodos.
    b. Una cantidad *n* fija de hijos en cada nodo (o cero hijos)
    c. Una profundidad finita.
    d. Una cantidad *variable* de hijos por cada nodo (o cero hijos)

2. Cual de las siguientes es una operación de PREDICADO.
    a. insertar(Lista, Valor)
    b. esVacia(Lista) → Bool
    c. primero(Lista) → Valor
    d. crearVacia() → Lista

3. En una Queue (Cola), si aplico la operación "*Dequeue*" resulta en que:
    a. Quito el primer elemento que ingresó.
    b. Inserto un elemento en el último lugar.
    c. Quito el último elemento que ingresó.
    d. Ninguna de las anteriores.

4. Si para implementar el TAD Stack uso un "*Arreglo con Tope*", mi principal problema va a ser:
    a. Tengo que pre-inicializar todos los elementos del arreglo para que no quede basura cuando lo use.
    b. Tengo que saber previamente el largo máximo del arreglo.
    c. No lo puedo usar para modelar el guardado de variables usadas en un proceso de llamadas de interrupción de un S.O.
    d. Todas las anteriores.

5. Cual de los siguientes es un enunciado **inválido** para el TAD Set:
    a. Tiene las siguientes operaciones definidas: Unión, Intersección, Diferencia.
    b. Tiene las siguientes operaciones definidas: Unión, Intersección, Borrar.
    c. Tiene las siguientes operaciones definidas: Unión, Insertar, Diferencia.
    d. Tiene las siguientes operaciones definidas: Unión, Intersección, Pertenece.

Ejercicio 3 (20 pts.)
a) (5 ptos.) De una especificación mínima funcional del tipo abstracto Lista de caracteres(LChar). Comente brevemente y de forma clara las funciones especificadas.

b) (5 ptos.) De una especificación mínima del tipo abstracto de dato Árbol Binario de Búsqueda de caracteres (AbbChar). Comente brevemente y de forma clara las operaciones especificadas.

c) (7 ptos.) Implemente la función LChar postOrden(AbbChar t), que toma como parámetro e entrada un Árbol Binario de Búsqueda de Caracteres y retorna una lista con el recorrido en postOren del Árbol.

d)(3 ptos.) Dada una instancia de la siguiente lista de caracteres(LChar) **[z,x,f,g,h,d,w,y,b]** obtenida al aplicar la función postOrden al árbol t1. Se pide dibujar una posible instancia del árbol t1.

Ejemplos:

Lista	Instancia del Árbol
-[c,r,f]	f / \ c r
-[]	-
-[j,e,c]	c \ e \ j

Notas:

- Cuando se pide "Comente brevemente y de forma clara las operaciones especificadas", significa dar las pre y post condiciones de cada operación.
- En la parte c) NO DEBE ACCEDER A LA REPRESENTACION DE LOS TADS QUE UTILICE al implementar la función.
- En la parte c), utilice sin necesidad de implementar, las operaciones especificadas en las partes a) y b).Si utiliza alguna otra función que no se encuentre especificada en la parte a) y b) debe implementarla. Puede asumir como implementada la función:

  LChar Append(LChar l1, LChar l2)
  //Post: retorna la concatenación de la lista l2 al principio de la lista l1.
  //Si ambas listas son vacías retorna la lista vacía.

Soluciones

Ejercicio 1
a)
//HTabla.h

```
#include <Simbolo.h>
#include <Valor.h>

typedef struct nodoTabla * HTabla;

HTabla crear();//Crea la tabla vacia

HTabla asignar(Valor, Simbolo, HTabla);//Asigna un valor a un simbolo. El valor anterior, si existia, queda guardado en la historia del simbolo

Valor buscar(Simbolo, HTabla);//Retorna el ultimo valor asignado a un simbolo

HTabla desAsignar(Simbolo, HTabla);//Deshace la ultima asignacion hecha sobre un simbolo

HTabla reIniciar(HTabla);//Todas las variables de la tabla quedan con el valor asignado por primera vez
```

b) Se usará el TAD Pila para representar el Historial ya que se requiere que los elementos del mismo sean recuperados según el último que fuese ingresado.

//HTabla.c

```
#include <HTabla.h>

typedef struct nodoTabla{
       Simbolo s;
       Stack vs;//es un stack almacenando todos los valores
       HTabla izq, der;
}

HTabla crear(){
      return NULL;
}

HTabla asignar(Valor v, Simbolo s, HTabla t){
      if(t == NULL){
         t = new(HTabla);
         t->s = s;
         t->vs = null();//NULL del stack
         t->vs = push(v,t->vs);
      }else if (igualSimbolo(t->s,s)){
           t->vs = push(v,t->vs);
```

```
      }else if (menorSimbolo(s,t->s)){
          return asignar(v,s,t->izq);
      }else{
          return asignar(v,s,t->der);
      }

      return t;
}

Valor buscar(Simbolo s, HTabla t){
     if (t == NULL){
       return "";
     else if(igualSimbolo(t->s,s)){
         return top(t->vs);
     }else if (menorSimbolo(s,t->s)){
         return buscar(s, t->izq);
     } else
         return buscar(s, t->der);
}

//Procedimiento auxiliar para borrar un nodo de un ABB
//en la operacion DesAsignar
HTabla borrarMayor(HTabla t){
      HTabla pMayor;
      if (t->der == NULL){
        pMayor = t;
        t = t->izq;
        pMayor->izq = NULL;
      }
      else{
          return borrarMayor(t->hder);
      }
}

HTabla desAsignar(Simbolo s, HTabla t){
      HTabla pMayor;
      if (t != NULL){
        if (igualSimbolo(t->s,s)){
          t->vs = pop(t->sv);
          if (empty(t->vs)){//empty de Pilas
            if (t->izq != NULL){
              pMayor = borrarMayor(t->izq);
              pMayor->izq = t->izq;
              pMayor->der = t->der;
              t = pMayor;
            else{
                t = t->der;
             }
          }
        }
```

```
        else if (menorSimbolo(s,t->s)){
            desAsignar(s,t->izq);
        else
            desAsignar(s,t->der);
       }
}

HTabla reIniciar(HTabla t){
      Valor aux;
      if (t != NULL){
        while (!empty(t->vs)){
             aux = top(t->vs);
             t->vs = pop(t->vs);
         }
        t->vs = push(aux,v-vs);
        reIniciar(t->izq);
        reIniciar(t->der);
      }
      return t;
}
```

Ejercicio 2

1. D
2. B
3. A
4. B
5. B

Ejercicio 3

a)

```
typedef struct ABBNode* ABBChar;


ABBChar VacioABB();
ABBChar insertElementABB(char elem, ABBChar t);
//Inserta el elemento elem en el arbol t preservando las propiedades de arbol binario
de //busqueda
ABBChar getIzqABB(ABBChar t);
ABBChar getDerABB(ABBChar t);
char getRaizABB(ABBChar t);
bool IsEmptyABB(ABBChar t);
void destruir(ABBChar &t);
```

b)

```
typedef struct NodoLChar* LChar;
LChar Null();
bool IsEmpty (LChar l);
char Head (LChar l );
LChar Tail (LChar l );
LChar Cons (char x, LChar l);
//Post: Inserta el elemento x al principio de la lista.
//Operaciones agregadas para la parte c)

LChar InsAtEnd (char x, LChar l);
//Post: inserta el elemento x al final de la lista l
LChar Append (LChar l1, LChar l2);
//Concatena la lista l1 y l2 (l1 al principio)
```

c)

```
LChar postOrden(AbbChar t){
  If(IsEmptyABB(t))
     return (Null());
 else{
       LChar ret= Null();
       LChar lizq=postOrden(getIzqABB (t));
       LChar lder=postOrden(getDerABB (t));
       ret=Cons(getRaizABB(t),ret);
       lizq=Append(lizq,lder);
       ret=Append(lizq,ret);
       return ret;

 }

}
```

d)
[z,x,a,f,g,h,w,v,b]