Nombre:	
C.I.:	

Examen 07 julio de 2009

Ejercicio 1 (45 puntos)

Se desea modelar el concepto de palabra para un diccionario. De una palabra nos interesa: la palabra en si misma (cadena de caracteres) y las posibles definiciones para esta palabra (una lista de definiciones).Cada definición será representada mediante una cadena de caracteres (char*) y el conjunto de definiciones asociada a una palabra, que se representara mediante el TAD ListaString, que es una lista cuyos elementos son cadenas de caracteres.

a) Implementar la siguiente especificación del TAD ListaString utilizando la representación :

```
struct NodoListaString{
  char* cadena;
  NodoListaString sig;
};.
```

TAD LISTASTRING

typedef struct NodoListaString* ListaString;
//El TAD ListaString consta de las siguientes operaciones

//Constructoras
ListaString crearListaString();
//Crea una lita de Strings vacía

ListaString agregarStringListaString(ListaString l, char* elem);
//post: agrega la cadena de caracteres elem a la lista l y retorna el resultado

//predicados
void esVaciaLista(ListaString l,bool& esvacia);
//devuelve en el parámetro esvacia verdadero si la lista l es vacía

//selectoras
int cantidadDeElementos(ListaString l);
//Devuelve si la lista l no es vacía la cantidad de definiciones que contiene
//de lo contrario retorna cero.

char* obtenerCadenaPosNListaString(ListaString l, int pos);
//Pre: pos>0 && pos<=cantidadDeElementos(l). Pre: l no es vacía
//Post: retorna la cadena de caracteres en la posición pos de la lista

//Destructoras
void borrarCadenaPosNLisraString(ListaString& l, int pos);
//Pre: pos>0 && pos<=cantidadDeElementos(l). Pre: l no es vacía
//Post: se quita de la lista l el elemento de la posición dada por pos.

ListaString resto(ListaString &l);
//Pre: la lista no es vacía

//Post: devuelve la lista sin su primer elemento

void destruirListaString(ListaString &l);
//Post libera toda la memoria utilizada por la lista l

b) Dar una representación para el TAD palabra e implementar la siguiente especificación de dicho TAD:

TAD PALABRA

typedef struct palabra* Palabra;

//Constructoras
Palabra crearPalabra(char* palabra, ListaString defs);
//Pre: palabra es una palabra válida. Pre: la lista defs es una lista con al menos un elemento
//Post: retorna una palabra con la lista de definiciones de la palabra igual a defs.

void agregarDefinicion(Palabra p, char* nuevadef);
//Pre: p no es vacío. Pre: nuevadef no es vacía
//Post: agrega a la palabra p una nueva definición

Palabra copiarPalabra(Palabra p);
//Post: Crea una copia exacta de la palabra
//realiza una copia en profundidad de todas sus definiciones

int cantidadDeDefinicionesPalabra(Palabra p);
//Post: Devuelve la cantidad de definiciones asociadas a la palabra p.

ListaString obtenerDefinicionesPalabra(Palabra p);
//Post: devuelve una lista con todas las definiciones asociadas a la palabra p

//auxiliares
void imprimirPalabra(Palabra p);
/*Pre: la palabra p no es vacía
Post: imprime en pantalla la Palabra p con el formato

Palabra:
"palabra"
Definiciones:
"definición 1"
"definición 2"
…… */
//Destructoras
void destruirPalabra(Palabra& p);
//Post: Libera la memoria ocupada por la palabra p

c) Dado el siguiente cabezal de procedimiento:

void imprimirPalabras(Palabra l[], int posInicial, int ultpos);
/* l es un arreglo de Palabras
Post: Imprime todas las palabras (la palabra y su lista de definiciones) del arreglo de palabras de la posición posInicial hasta la posición ultpos.

El orden de impresión esta dado por el orden en que están las palabras en el arreglo */

Implementar dicho procedimiento en forma recursiva.
Sugerencia: Utilizar la función imprimirPalabra del TAD Palabra.

Ejercicio 2(30 puntos)

Utilizando los TADS implementados en el ejercicio 1, de una representación e implemente utilizando árboles binarios de búsqueda el TAD diccionario de palabras.

TAD DICCIONARIO

typedef structNodo* Diccionario;

//constructoras
void crearDiccionario(Diccionario & d);
//Crea un diccionario vacío

void agregarPalabraDiccionario(Diccionario &d, Palabra p);
//post: agrega al diccionario d la palabra p, en caso de existir la palabra agrega las definiciones que
//no sean actualmente definiciones de la palabra que se encuentra en el diccionario

//predicados
bool pertenecePalabra(Diccionario d, char* palabra)
//post: retorna verdadero si la palabra se encuentra en el diccionario.

//selectoras
Palabra obtenerPalabraDiccionario(Diccionario d, char* palabra);
//pre: la palabra "palabra" se encuentra en el diccionario
//post: retorna la palabra "palabra"

//destructoras
void quitarPalabraDiccionario(Diccionario &d, Palabra p);
//pre: la palabra "palabra" se encuentra en el diccionario
//post: elimina la palabra p del diccionario.

void destruirDiccionario(Diccionario &d);
//post: libera el espacio en memoria ocupado por diccionario.

NOTA: Para el ejercicio 2 asuma que las palabras así como sus definiciones se encuentran todas en mayúscula.

Ejercicio 3(25 puntos)

Explique cómo funciona y luego codifique el método de la burbuja para ordenación de arreglos de enteros.

El cabezal del procedimiento será: ***void burbuja(int arreglo[], int tamanio);***