Nombre:	
C.I.:	

Solución 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(){
  return NULL;
}

//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
   NodoListaString nuevo=new NodoListaString;
   nuevo->cadena=new char[strlen(elem) +1];
   strcpy(nuevo->cadena,elem);
   nuevo->sig=l;
   return nuevo;
}

//predicados
void esVaciaLista(ListaString l,bool& esvacia){
  espacia=(l==NULL);
}
//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.
   if(l==NULL)   return 0;
   else return (1 + cantidadDeElementos(l->sig));
```

```
}

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
      while(--pos!=0){
         l=l->sig;
      }
       return l->cadena;

}

//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 aux;
    if(pos==1){
        aux=l;
        l=l->sig;
        delete [] aux->cadena ;
        delete aux;
    } else{
      aux=l;
      while((pos-1)!=1){
         aux=aux->sig;
         pos--;
      }
       ListaString borrar;
      Borrar=aux->sig;
      aux->sig=aux->sig->sig;
      delete [] borrar->cadena ;
      delete borrar;

    }
}

ListaString resto(ListaString& l){
//Pre: la lista no es vacía
//Post: devuelve la lista sin su primer elemento
   return l->sig;

}

void destruirListaString(ListaString &l){
//Post libera toda la memoria utilizada por la lista l
          if(l!=NULL){
                destruirListaString(l->sig);
                 delete l;
                 l=NULL;
           }
```

```
}
```

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

TAD PALABRA

```
typedef struct palabra* Palabra;

#include <string.h>
#include <stdlib.h>



struct palabra{
   char* elem;
   ListaString defs;
};

//Constructoras
Palabra crearPalabra(char* pal, 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.
     Palabra p = new palabra;
     p->elem=new char[strlen(pal) + 1];
     strcpy( p->elem,pal);
     p->defs=defs;
     return p;
}

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
  if( ¡(estaPalabra(p->defs,nuevadef)) ){
      p->defs= agregarStringListaString(p->defs, nuevadef);

  }
}


Palabra copiarPalabra(Palabra p){
//Post: Crea una copia exacta de la palabra
//realiza una copia en profundidad de todas sus definiciones
     return CrearPalabra(p->elem, CopiarLista(p->defs));
}

//auxiliar solo visible en el .c
ListaString CopiarLista(ListaString l)
  //post: devuelve una copia sin compartir memoria de la lista l)
{
   int cant=  cantidadDeElementos(l);
```

```
   ListaString ret=crearListaString();
   for(int pos=1;pos<=cant;pos++){
    ret=agregarStringListaString( ret,obtenerCadenaPosNListaString(l,pos));
   }
  return ret;

}

int cantidadDeDefinicionesPalabra(Palabra p){
//Post: Devuelve la cantidad de definiciones asociadas a la palabra p.
 return   cantidadDeElementos(p->defs);
}

ListaString obtenerDefinicionesPalabra(Palabra p){
//Post: devuelve una lista con todas las definiciones asociadas a la palabra p
   return p->defs;
}

//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"
                ..... */
  cout<<p->elem<< endl;
  int cant=  cantidadDeElementos(p->defs);
   for(int pos=1;pos<=cant;pos++){
     cout<<"\t"<<obtenerCadenaPosNListaString(p->defs,pos)<<endl;
  }


}

//Destructoras
void destruirPalabra(Palabra& p){
//Post: Libera la memoria ocupada por la  palabra p

     destruirListaString(p->defs)
     delete []   p->elem;
     delete p;
      p=NULL;

}
```

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);***

Solución:
Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Si se está ordenando en forma creciente a un vector, se puede optar por partir de la parte baja del arreglo y que en cada iteración se lleve el elemento de mayor valor del subvector no ordenado(parte baja) a la parte alta del arreglo (subvector ordenado), o sino partir de la parte alta del arreglo y en cada iteración llevar el elemento de menor valor hasta la parte más baja del arreglo. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada

```
void burbuja(int a[], int tamanio){
   for(int i=0;i<tamanio-1;i++){
      for(int j=0;j<tamanio-i -1;j++){
       if(a[j]>a[j+1]){
                int temp=a[j];
                a[j]=a[j+1];
               a[j+1]=temp;
       }
      }
  }

}
```