Estructuras de Datos y algoritmos 1

Solución (Tema: Listas) - Práctico 3

Tema: Punteros - Listas y árboles

Listas

2) Dada una lista lineal encadenada de enteros, implementar las siguientes operaciones:
- **a) Insertar un elemento al principio de la misma.**
- **b) Eliminar el primero de la misma.**
- **e) Retornar el mínimo (máximo) de la misma (una referencia al nodo donde se encuentra).**

3) Hacer una función que imprima una lista en sentido directo e inverso recorriendo explícitamente la lista solo una vez.

```
/*****************************************/
//             ListasSimple.h
/*****************************************/

#ifndef LISTASIMPLE_H
#define LISTASIMPLE_H

#include <iostream.h>
#include <malloc.h>

struct _nodoSimple;
typedef struct _nodoSimple nodoSimple;
typedef nodoSimple* nodoSimplePtr;

/*****************************************/
//2 - a) Insertan un entero al ppio de una lista
//Como procedimiento
void procInsertPpio(nodoSimplePtr *, int);
//Como funcion
nodoSimplePtr funcInsertPpio(nodoSimplePtr, int);

/*****************************************/
//2 - b) Eliminan el entero al ppio de una lista
//Como procedimiento
void procDeletePpio(nodoSimplePtr *);
//Como funcion
nodoSimplePtr funcDeletePpio(nodoSimplePtr);
```

```
/*******************************************/
//2 - e) Retorna la posición del mínimo en una lista
//Como funcion iterativa
nodoSimplePtr funcIterPosMin(nodoSimplePtr);
int invFuncIterPosMin(nodoSimplePtr l);
//Como funcion recursiva
nodoSimplePtr funcRecPosMin(nodoSimplePtr);
int invFuncRecPosMin(nodoSimplePtr l);
/*******************************************/
//3) Despliega la lista de ppio a fin y de fin a ppio
void listarDerRev(nodoSimplePtr );

#endif

/*******************************************/
//             ListaSimple.cpp
/*******************************************/
#include "ListaSimple.h"

struct _nodoSimple{
      int dato;
      nodoSimplePtr prox;
};

/*******************************************/
//2 - a) Insertan un entero al ppio de una lista
//Como procedimiento
void procInsertPpio(nodoSimplePtr *p, int d){
      nodoSimplePtr nuevo = new nodoSimple;

      nuevo->dato = d;
      nuevo->prox = *p;
      *p = nuevo;

}
//Como funcion
nodoSimplePtr funcInsertPpio(nodoSimplePtr l, int d){
      nodoSimplePtr nuevo = new nodoSimple;

      nuevo->dato = d;
      nuevo->prox = l;
      l = nuevo;
      return l;
}
```

```
/*******************************************/
//2 - b) Eliminan el entero al ppio de una lista
//Como procedimiento
void procDeletePpio(nodoSimplePtr *p){
      nodoSimplePtr borrar = *p;

      *p = (*p)->prox;
      delete borrar;
}
//Como funcion
nodoSimplePtr funcDeletePpio(nodoSimplePtr l){
      nodoSimplePtr borrar = l;

      l = l->prox;
      delete borrar;
      return l;
}

/*******************************************/
//2 - e) Retorna la posición del mínimo en una lista
//Como funcion iterativa
nodoSimplePtr funcIterPosMin(nodoSimplePtr l){
      nodoSimplePtr minimo = l;

            while(l){
                  if (l->dato < minimo->dato )
                        minimo = l;
                  l = l->prox ;
            }
      return minimo;
}

int invFuncIterPosMin(nodoSimplePtr l){
      nodoSimplePtr min = funcIterPosMin(l);
            if (min)
                  return min->dato;
            else
                  return -1111;//Error
}
```

```
//Como funcion recursiva
nodoSimplePtr funcRecPosMin(nodoSimplePtr l){
	nodoSimplePtr minimo;

	if (l == NULL || l->prox == NULL)
		return l;
	else{
		minimo = funcRecPosMin(l->prox );
		if(l->dato < minimo->dato )
			return l;
		else
			return minimo;
	}
}

int invFuncRecPosMin(nodoSimplePtr l){
	nodoSimplePtr min = funcRecPosMin(l);
		if (min)
			return min->dato;
		else
			return -1111;//Error
}

/*******************************************/
//3) Despliega la lista de ppio a fin y de fin a ppio
void listarDerRev(nodoSimplePtr l){
	if (l){
		cout<<l->dato<<" - ";
		listarDerRev(l->prox );
		cout<<l->dato<<" - ";
	}
	else
		cout<<"\n";
}
```

4) Asumiendo que trabajamos con listas ordenadas:

a) La inserción de un elemento.

```
//La solución de este ejercicio toma como base la misma declaración de listas
//que en los ejercicios anteriores.

//4 - a) Insercion en una lista ordenada
//Iterativo
//Como procedimiento
void procIterInsertOrd(nodoSimplePtr *p, int e){
      nodoSimplePtr nuevo;
      nodoSimplePtr ant;
      nodoSimplePtr q;

      if (*p == NULL || (*p)->dato >= e)
                  procInsertPpio(p, e);
      else{ q = *p;
                  while (q!= NULL && q->dato < e){
                        ant = q;
                        q = q->prox;
                  }
                  nuevo = (nodoSimplePtr) malloc (sizeof(nodoSimple));
                  nuevo->dato = e;
                  nuevo->prox = q;
                  ant->prox = nuevo;

      }
}
//Como funcion
nodoSimplePtr funcIterInsertOrd(nodoSimplePtr l, int e){
      nodoSimplePtr nuevo;
      nodoSimplePtr ant;
      nodoSimplePtr q;

      if (l == NULL || l->dato >= e)
                  l = funcInsertPpio(l, e);
      else{ q = l;
                  while (q!= NULL && q->dato < e){
                        ant = q;
                        q = q->prox;
                  }
                  nuevo = (nodoSimplePtr) malloc (sizeof(nodoSimple));
                  nuevo->dato = e;
                  nuevo->prox = q;
                  ant->prox = nuevo;

      }
      return l;
}
```

```
//Recursivo
//Como procedimiento
void procRecInsertOrd(nodoSimplePtr *p, int e){

      if (*p == NULL || (*p)->dato >= e)
            procInsertPpio(p, e);
      else{
            procRecInsertOrd(&(*p)->prox , e);
      }
}
//Como funcion
nodoSimplePtr funcRecInsertOrd(nodoSimplePtr l, int e){
      if (l == NULL || l->dato >= e)
            l = funcInsertPpio(l, e);
      else{
            l->prox  = funcRecInsertOrd(l->prox , e);
      }
      return l;
}
```

7) Implementar las siguientes operaciones si la lista es doblemente encadenada:

c) Insertar un elemento en la posición K.

f) Eliminar el último elemento de la lista.

(*) Para esta dos operaciones finales considerar la posibilidad de tener un puntero al final de la lista. Estudiar la eficiencia que se obtiene.

```
/*****************************************/
//             ListaDoblePpioFin.h
/*****************************************/
#ifndef LISTADOBLEPPIOFIN_H
#define LISTADOBLEPPIOFIN_H

#include<malloc.h>
#include<iostream.h>

struct _nodoCabeceraDoble;
typedef struct _nodoCabeceraDoble cabeceraDoble;
typedef cabeceraDoble *cabeceraDoblePtr;
/*****************************************/
//7 - c) Inserta un entero en la posicion k de la lista ó al final
//Iterativo
//Como procedimiento
void procIterInsertPosDoble(cabeceraDoblePtr, int, int);
/*****************************************/
//7 - f) Elimina el entero al final de una lista
//Como procedimiento
void procDeleteFinDoble(cabeceraDoblePtr);

#endif
```

```
/*****************************************/
//             ListaDoblePpioFin.cpp
/*****************************************/
#include "ListaDoblePpioFin.h"

struct _nodoPpioDobleFin;
typedef struct _nodoDoblePpioFin nodoDoblePpioFin;
typedef nodoDoblePpioFin* nodoDoblePpioFinPtr;

struct _nodoDoblePpioFin{
      int dato;
      nodoDoblePpioFinPtr ant, prox;
};

struct _nodoCabeceraDoble{
      nodoDoblePpioFinPtr ppio;
      nodoDoblePpioFinPtr fin;
};
/*****************************************/
//7 - c) Inserta un entero en la posicion k de la lista ó al final
//Iterativo
//Como procedimiento
void procIterInsertPosDoble(cabeceraDoblePtr c, int k, int e){
      nodoDoblePpioFinPtr actual, nuevo;

      if (k == 1 || (c->ppio == NULL && c->fin == NULL))
            procInsertPpioDoble(c, e);
      else{
            actual = c->ppio;
            while (actual->prox != NULL && k > 2){
                  actual = actual->prox;
                  k--;
            }
            if(actual->prox == NULL)
                  procInsertFinDoble(c, e);
            else{
                  nuevo = new nodoDoblePpioFin;
                  nuevo->dato = e;
                  nuevo->prox = actual->prox;
                  nuevo->ant = actual;
                  actual->prox = nuevo;
                  nuevo->prox ->ant = nuevo;
            }
      }
}
```

```
/*****************************************/
//7 - f) Elimina el entero al final de una lista
//Como procedimiento
void procDeleteFinDoble(cabeceraDoblePtr c){
      nodoDoblePpioFinPtr borrar;

      if (c->ppio != NULL && c->fin != NULL){
            borrar = c->fin  ;
            c->fin = c->fin->ant  ;
            if(c->fin == NULL)
                  c->ppio = NULL;
            else
                  c->fin->prox = NULL;
            free(borrar);
      }

}
```