

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo			Parcial 06/12/2010
Tema	Página 1	Ejercicio	Puntaje
Apellido y Nombre		1 (2 pts)	
Padrón		2 (3 pts)	
		3 (3 pts)	
Nota Final		4 (2 pts)	
Corrigió			

## Práctica

Entregar la resolución de la Teoría y la Práctica en hojas separadas

Se desea generar un TDA CalendarioCurso que representa la planificación de un curso de la Facultad. En el calendario se consignan fechas de clase, y a cada clase se le agregan temas que están cada uno a cargo de un ayudante determinado.

### **Se pide:**

- Compare los siguientes diseños de estructuras para el TDA CalendarioCurso, indicando ventajas y desventajas al aplicarlos en la implementación del contrato del TDA (que se incluye en el anexo). Sea breve.

#### *a. TCalendarioCurso1*

```
typedef struct {
    TListaSimple clases; /* ordenada por fecha de clase */
} TCalendarioCurso1;

typedef struct {
    long fecha;
    TListaSimple temas; /* queda en el orden en que se va insertando */
} TClase;

typedef struct {
    char* desctema;
    char* ayudante;
} TTema;
```

#### *b. TCalendarioCurso2*

```
typedef struct {
    TListaSimple clasestemas; /* ordenada por fecha de clase y nro. de tema */
} TCalendarioCurso2;

typedef struct {
    char fecha[9];
    int nrotema; /* asignado automáticamente , empieza en 1 */
    char* desctema;
    char* ayudante;
} TClaseTema;
```

- Elija una de las opciones presentadas para el diseño de la estructura del TDA, e implemente la primitiva **CC\_AgregarTemaClase** de TCalendarioCurso considerando que la estructura es la elegida. Defina y considere los errores que pudieran suceder.
- A continuación se propone una implementación de la primitiva **CC\_ObtenerTemasAyudante** utilizando uno de los diseños de estructura propuestos (a los fines de este ejercicio, es indistinto cuál se ha elegido para ejemplificar). También se muestra la implementación de una función de comparación para elementos de texto.  
La implementación de **CC\_ObtenerTemasAyudante** mostrada emplea un TDA ListaOrdenada para el resultado que devuelve. Este TDA ListaOrdenada mantiene los elementos en orden de acuerdo con una función de comparación que recibe en su primitiva LSO\_Crear.
  - Defina la estructura del TDA ListaOrdenada,
  - indique la declaración (firma o prototipo) de la primitiva **LSO\_InsertarOrdenado**
  - e implemente la primitiva de inserción **LSO\_InsertarOrdenado** de ListaOrdenada (sin apelar a ninguna otra primitiva, de ningún TDA, en su resolución),

tal que la invocación realizada desde **CC\_ObtenerTemasAyudante** y la función de comparación dada funcionen correctamente. No es necesario resolver a, b, y c en forma separada, pero sí responder a todo lo pedido.

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 25/10/2010
Padrón	Apellido y Nombre	Tema

*Función de comparación de textos propuesta:*

```
int compararStr(void* elem1, void* elem2) {
    return strcmp((char*)elem1, (char*)elem2);
}
```

*Implementación de ejemplo de **CC\_ObtenerTemasAyudante** (en este caso basada en la opción 2 de diseño de estructura)*

```
int CC_ObtenerTemasAyudante(TCalendarioCurso* calendariocurso, char* ayudante,
TListaOrdenada* temas) {

    TClaseTema claseTemaEnLista;

    if (ls_Vacia(calendariocurso->clasestemas)) return OK;

    ls_MoverCorriente(calendariocurso->clasestemas, LS_PRIMERO);

    do {
        ls_ElemCorriente(calendariocurso->clasestemas, &claseTemaEnLista);

        if (!strcmp(claseTemaEnLista.ayudante, ayudante)) {
            lso_InsertarOrdenado(temas, claseTemaEnLista.descitema);
        }

    } while (ls_MoverCorriente(&calendariocurso->clasestemas, LS_SIGUIENTE));

    return OK;
}
```

4. Dada la siguiente estructura de datos, de la que se muestra el diseño de su estructura y la implementación de algunas de sus primitivas,
- enuncie si corresponde a alguna de las estructuras que conoce, en caso afirmativo qué estructura es, y justifique en forma breve y clara
  - explique brevemente las características principales, funcionalidades, y limitaciones de la estructura de datos presentada

No es necesario resolver a y b en forma separada, pero sí responder a todo lo pedido.

```
typedef struct {
    int numElementos;
    int elementos[MAX];
    int indice;
} QueSera;

int agregar(QueSera* estructura, int elem){

    if ((estructura->indice == (estructura->numElementos)-1))
        return -1;

    estructura->elementos[++estructura->indice] = elem;
    estructura->numElementos++;

    return 0;
}

int quitar(QueSera* estructura, int* elem){

    if (estructura->indice == -1)
        return -1;

    *elem = estructura->elementos[estructura->indice--];
    estructura->numElementos--;

    return 0;
}
```

Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo		Parcial 25/10/2010
Padrón	Apellido y Nombre	Tema

**Nota:**

En todos los casos en que no se indique explícitamente lo contrario, los elementos con los que cuenta son las estructuras del lenguaje C, el TDA ListaSimple explicado en clase, el TDA Pila explicado en clase, y el TDA Cola explicado en clase. Toda funcionalidad por encima de estos elementos deberá ser desarrollada como parte del examen.

Para aprobar el examen, debe (las condiciones son aditivas, ninguna es opcional):

- demostrar claramente que ha aprendido el concepto de abstracción
- demostrar claramente que ha aprendido a utilizar y a implementar las estructuras de datos que se enseñaron
- demostrar claramente que ha aprendido los elementos de programación que se enseñaron
- resolver correctamente al menos el 60% del examen

**Anexo**

El contrato que deberá cumplir el TDA CalendarioCurso es el siguiente:

```
typedef struct {
    short int dia;
    short int mes;
    short int anio;
} TFecha; /* Tipo auxiliar TFecha */

/* Crea el CalendarioCurso, PRE: CalendarioCurso no creado, POST: CalendarioCurso creado */
int CC_Crear(TCalendarioCurso* calendariocurso);

/* Destruye el CalendarioCurso, PRE: CalendarioCurso creado, POST: CalendarioCurso
destruido */
int CC_Destruir(TCalendarioCurso* calendariocurso);

/* Registra que en una determinada fecha se da clase, PRE: CalendarioCurso creado, POST:
se ha agregado una clase para la fecha dada */
int CC_AgregarClase(TCalendarioCurso* calendariocurso, TFecha fecha);

/* Agrega el tema cuya descripción es desc tema, a cargo del ayudante cuyo nombre se
indica, en la fecha dada, PRE: CalendarioCurso creado, POST: se ha agregado el tema */
int CC_AgregarTemaClase(TCalendarioCurso* calendariocurso, TFecha fecha, char* desc tema,
char* ayudante);

/* Obtiene una lista ordenada (según el criterio de la lista ordenada pasada por
parámetro) de la descripción de los temas dados por el ayudante indicado, PRE:
CalendarioCurso creado, lista de temas creada y vacía POST: la lista temas contiene un
elemento por cada tema dado por el ayudante indicado; el elemento consiste simplemente en
un texto descriptivo */
int CC_ObtenerTemasAyudante(TCalendarioCurso* calendariocurso, char* ayudante,
TListaOrdenada* temas);

/* Elimina la clase correspondiente a la fecha dada (y elimina todos los temas
asociados), PRE: CalendarioCurso creado, POST: se ha eliminado la clase indicada */
int CC_EliminarClase(TCalendarioCurso* calendariocurso, TFecha fecha);

/* Obtiene las fechas para las que hay alguna clase registrada, PRE: CalendarioCurso
creado, listaClases creada y vacía POST: listaClases contiene las fechas pedidas */
int CC_ObtenerFechasClase(TCalendarioCurso* calendariocurso, TListaSimple* listaClases);

/* Obtiene los temas y ayudante a cargo de cada uno, para una fecha dada, PRE:
CalendarioCurso creado, listaTemas creada y vacía, POST: listaTemas contiene un elemento
que indica la descripción del tema y el ayudante, para cada tema de la fecha indicada*/
int CC_ObtenerTemasClase(TCalendarioCurso* calendariocurso, TFecha fecha, TListaSimple*
listaTemas);
```