Curso 2007/2008 Estructura de Datos y de la Información
I. Informática, I. T. Informática de Gestión y de Sistemas

PRÁCTICA 2

1 El problema

Partiendo del mismo problema planteado en la práctica 1, supongamos que puedan realizarse reservas de butacas no necesariamente consecutivas. Para gestionar las reservas y los pagos ahora será necesario almacenar el nombre y apellidos de los espectadores que han hecho la reserva y qué butacas han reservado. De nuevo, para simplificar el problema, en esta práctica supondremos que la platea se numera correlativamente, de esta forma tan sólo es necesario modelizar UNA SOLA FILA.

2 Descripción de la tarea

El alumno diseñará un único programa principal (`principal.pas`) que lea, a través de un fichero, las distintas operaciones de los usuarios—reserva, cancelación o pago—las procese y visualice por pantalla los resultados. El objetivo de la práctica es utilizar varios tipos abstractos de datos (TADs) implementados de forma dinámica para resolver el problema (una lista de butacas y una multilista para almacenar el nombre y la lista de reservas de cada espectador—ver figura).

El programa funcionará en modo batch (sin interactividad con el usuario) y de manera transparente ejecutando las siguientes tareas:

1. Lectura de un fichero de texto `platea.txt` con el número de butacas del teatro y creación de la lista de butacas correspondiente.

2. Lectura del fichero de texto `operaciones.txt` que contiene las operaciones a procesar e inserción de cada operación en una cola. En la operación consta el código—R (reserva), C (cancelación), P (pago) —apellidos y nombre de la persona y el número de butacas que reserva o cancela.

3. Procesar cada operación almacenada en la cola tal y como sigue:

   a. Imprimir un mensaje del tipo:

   ```
   ***********************************************
   Nueva operación XXXXXXX: Nombre – NumeroButacas
   ***********************************************
   ```

   donde XXXXXXX es el tipo de operación—reserva, cancelación o pago.

   b. Si la operación es `Reserva` buscar si ya existe una reserva a nombre de ese espectador e imprimir un mensaje del tipo:

   ```
   ***********************************************
   Error en Reserva: Ya existe reserva de XXXXXXXX
   ***********************************************
   ```

donde XXXXXXX es el nombre del espectador que quiere reservar.

Si no existe, para atender la solicitud, verificar si existe un número suficiente de butacas libres **correlativas** (como primera opción), o **no correlativas** (como segunda posibilidad). Si no hay butacas libres imprimir un mensaje del tipo:

```
Operación NO atendida. No hay butacas libres
```

Si hay almacenar el nombre del espectador y las butacas reservadas, modificar el estado de cada butaca a *reservada* e imprimir un mensaje del tipo:

```
Reserva de XXXXXXXXXXX:
        Butaca XX
        Butaca XX
```

donde XXXXXXX es el nombre del espectador, y XX es el número de la butaca.

c. Si la operación es `Pago` buscar si el espectador tiene reservas. Si es así eliminar la reserva, así como las butacas correspondientes. Imprimir un mensaje del tipo:

```
Pago de XXXXXXXXXXX
     Butaca XX
     Butaca XX
```

donde XXXXXXX es el nombre del espectador, y XX es el número de la butaca.

En caso de que la reserva no exista, imprimir un mensaje del tipo:

```
**********************************************
Error en Pago: No existe reserva de XXXXXXXXXX
**********************************************
```

donde XXXXXXX es el nombre del espectador.

d. Si la operación es `Cancelacion`, se permite cancelar toda o parte de las reservas, por lo que se eliminarán de la lista de reservas tantas butacas como se indiquen. Para ello buscar si el espectador tiene reservas, eliminar tantas de ellas como indique la operación, actualizar el estado de las butacas a *libre* e imprimir un mensaje del tipo:

```
Cancelacion de XXXXXXXXXX
     Butaca XX
     Butaca XX
```

donde XXXXXXX es el nombre del espectador, y XX es el número de la butaca.

En caso de que el espectador no tenga lista de reservas, imprimir un mensaje del tipo:

```
*****************************************************
Error en Cancelación: No existen reservas de XXXXXXXXXX
*****************************************************
```

donde XXXXXXX es el nombre del espectador.

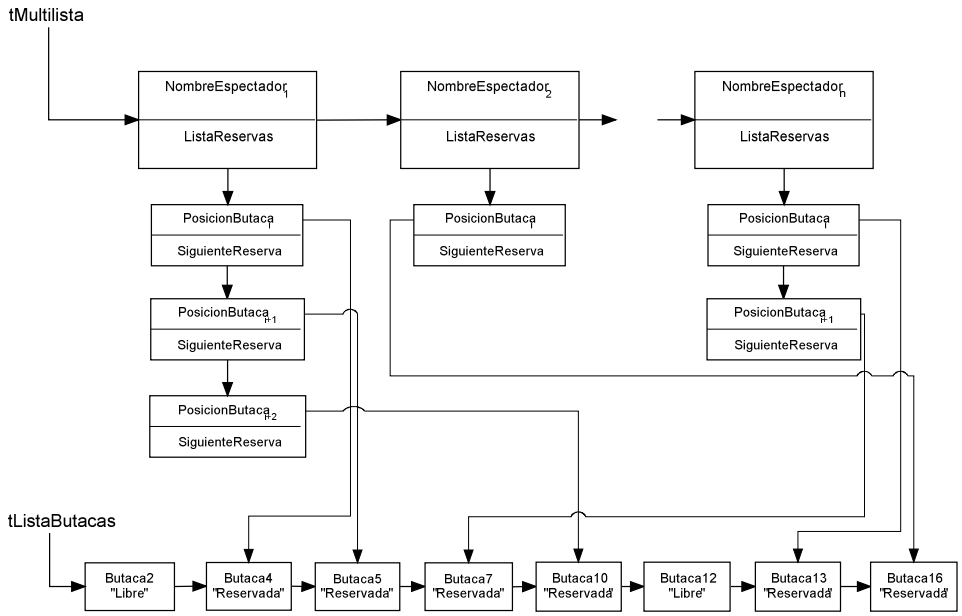
4. Visualizar en pantalla la platea (en nuestro caso una única fila). El formato será el siguiente:

```
Butaca XX        Estado
…
Butaca XX        Estado
```

donde XX es el número de la butaca.

3 Estructuras de datos

Para resolver el problema, se implementarán 4 tipos abstractos de datos: una cola para almacenar las distintas operaciones (TAD ColaOperaciones), una lista simplemente enlazada (TAD ListaButacas) para mantener actualizado el estado de las butacas, una multilista (TAD Multilista) para almacenar los datos de los espectadores y, para cada uno de ellos, una lista de reservas (TAD ListaReservas) que guarde las distintas reservas. Por tanto, el TAD Multilista manejará el TAD ListaReservas sólo a través de las operaciones permitidas. Lo mismo ocurre con el programa principal con respecto al TAD ColaOperaciones, TAD ListaButacas y TAD Multilista. Las estructuras se relacionan tal y como muestra la figura siguiente:



4 TAD ColaOperaciones

Para gestionar las operaciones se utilizará un TAD Cola en el que se almacenan las distintas peticiones de los espectadores según el orden en que las vayan efectuando.

4.1 Tipos de datos incluidos en el TAD

- **tColaOp** Representa a una cola de operaciones.
- **tOp** Tipo de la operación a ejecutar: P(ago), R(eserva), C(ancelación)
- **tNumB** Número de butacas afectadas por la operación.
- **tNomE** Nombre del espectador.

- **tInfoC** Dato de un elemento de la cola de operaciones, compuesto por los campos *Op*, *NomE* y *NumB*, que contienen el tipo de operación, el nombre del espectador y el número de butacas afectadas.
- **tPosC** Posición de un elemento de la cola de operaciones.
- **NULO** Constante utilizada para representar posiciones nulas.

4.2 Operaciones incluidas en el TAD

Una precondición común para todas las operaciones (salvo ColaVacia) es que la cola debe estar previamente inicializada.

- `ColaVacia (tColaOp)` → `tColaOp`
  Crea una cola de operaciones vacía.
- `EsColaVacia (tColaOp)` → `Boolean`
  Determina si la cola de operaciones está vacía.
- `Meter(tColaOp,tOp,tNomE,tNumB)`→ `tColaOp, Boolean`
  Inserta una nueva operación en la cola de operaciones. Devuelve un valor falso si no hay memoria suficiente para realizar la operación.
- `Frente (tColaOp)` → `tOp, tNomE, tNumB, Boolean`
  Devuelve el contenido del elemento más antiguo de la cola de operaciones y un valor verdadero/falso indicando si se ha podido obtener dicho elemento.
- `Sacar (tColaOp)` → `tColaOp, Boolean`
  Elimina el elemento que está en el frente de la cola de operaciones y devuelve un valor verdadero/falso indicando si ha podido eliminar dicho elemento.

5 TAD ListaButacas

Para mantener un mapa de la ocupación de las butacas de un local se utilizará un TAD Lista, en el que se registran las butacas, numeradas correlativamente de 1 a n y su estado—libre, o reservada—. Se implementará el TAD Lista de forma dinámica, con una lista simplemente enlazada con punteros (unit ListaButacas.pas).

5.1 Tipos de datos incluidos en el TAD

- **tListaB** Representa a una lista ordenada por número de butaca.
- **tNumB** Número de la butaca.
- **tEstadoB** Estado de la butaca—libre o reservada (tipo enumerado).
- **tInfoB** Dato de un elemento de la lista, compuesto por los campos *NumB* y *EstadoB* que contienen el número de la butaca, su estado.
- **tPosB** Posición de un elemento de la lista de butacas.
- **NULO** Constante utilizada para representar posiciones nulas.

5.2 Operaciones incluidas en el TAD

Una precondición común para todas las operaciones (salvo ListaButacasVacia) es que la lista debe estar previamente inicializada.

- `ListaButacasVacia (tListaB)` → `tListaB`
  Crea una lista vacía.
- `esListaButacasVacia (tListaB)` → `Boolean`
  Determina si la lista está vacía
- `PrimeroButaca (tListaB)` → `tPosB`
  Devuelve la posición del primer elemento de la lista (o NULO si la lista está vacía).
- `UltimoButaca (tListaB)` → `tPosB`
  Devuelve la posición del último elemento de la lista (o NULO si la lista está vacía).
- `SiguienteButaca (tListaB, tPosB)` → `tPosB`
  Devuelve la posición en la lista del siguiente elemento a la posición indicada (o NULO si la posición no tiene siguiente).
  PreCD: La posición tiene que ser válida.
- `AnteriorButaca (tListaB, tPosB)` → `tPosB`
  Devuelve la posición en la lista del anterior elemento a la posición indicada (o NULO si la posición no tiene anterior).
  PreCD: La posición tiene que ser válida.
- `AnadirButaca (tListaB, tNumB)` → `tListaB, Boolean`
  Añade una nueva butaca con número **tNumB** en la lista. Devuelve un valor falso si no hay memoria suficiente para realizar la operación.
  PreCD: No existe una butaca con ese número
  PosCD: Se asigna a la butaca el estado libre
- `BorrarButaca (tListaB, tPosB)` → `tListaB`
  Borra de la lista la butaca que está en la posición indicada.
  PreCD: La posición tiene que ser válida.
- `ObtenerDatoButaca (tListaB, tPosB)`→ `tNumB, tEstadoB`
  Devuelve el dato situado en la posición indicada de la lista.
  PreCD: La posición tiene que ser válida.
- `ActualizaDatoButaca(tListaB,tPosB,tEstadoB)`→ `tListaB`
  Actualiza el dato situado en la posición indicada con estado **tEstadoB**.
  PreCD: La posición tiene que ser válida.

6 TAD ListaReservas

Para mantener la lista de reservas asociadas a un espectador se utilizará un TAD Lista, en el cual se almacenan las posiciones de las butacas de la lista de butacas. El TAD Lista se implementará como una lista dinámica y simplemente enlazada (unit ListaReservas.pas).

6.1 Tipos de datos incluidos en el TAD

- **tListaR** Representa a una lista de simple enlace.
- **tInfoR** Dato de un elemento de la lista, en este caso, la posición de una butaca de la lista de butacas (tPosB).
- **tPosR** Posición de un elemento de la lista de reservas.
- **NULO** Constante utilizada para representar posiciones nulas.

6.2 Operaciones incluidas en el TAD

Una precondición común para todas las operaciones (salvo ListaReservasVacia) es que la lista debe estar previamente inicializada.

- `ListaReservasVacia (tListaR)` → `tListaR`
  Crea una lista vacía.
- `esListaReservasVacia (tListaR)` → `Boolean`
  Determina si la lista está vacía.
- `PrimeroReserva (tListaR)` → `tPosR`
  Devuelve la posición del primer elemento de la lista (o NULO si la lista está vacía).
- `UltimoReserva (tListaR)` → `tPosR`
  Devuelve la posición del último elemento de la lista (o NULO si la lista está vacía).
- `SiguienteReserva (tListaR, tPosR)` → `tPosR`
  Devuelve la posición del siguiente elemento a la posición indicada (o NULO si la posición no tiene siguiente).
  PreCD: La posición tiene que ser válida.
- `AnteriorReserva (tListaR, tPosR)` → `tPosR`
  Devuelve la posición del anterior elemento a la posición indicada (o NULO si la posición no tiene anterior).
  PreCD: La posición tiene que ser válida.
- `AnadirReserva(tListaR, tInfoR)`→ `tListaR, Boolean`
  Añade al final de la lista un nuevo dato de tipo **tInfoR**. Devuelve un valor falso si no hay memoria suficiente para realizar la operación.
- `BorrarReserva (tListaR, tPosR)` → `tListaR`
  Borra de la lista el elemento que está en la posición indicada.
  PreCD: La posición tiene que ser válida.
- `ObtenerDatoReserva (tListaR, tPosR)` → `tInfoR`
  Devuelve el dato situado en la posición indicada de la lista.
  PreCD: La posición tiene que ser válida.

7 TAD Multilista

Para mantener una lista de reservas, se utilizará un TAD Multilista, en el que se almacenan ordenadamente los espectadores, y para cada uno su lista de reservas. El TAD Multilista se implementará de manera dinámica como una unit (fichero Multilista.pas) y hará uso del TAD ListaReservas para organizar las reservas.

7.1 Tipos de datos incluidos en el TAD Multilista

- **tMultilista** Representa a una multilista **ordenada** de simple enlace.
- **tNomE** Nombre del espectador, criterio de ordenación de la lista.
- **tListaR** Lista de reservas.
- **tInfoM** Dato de un elemento de la multilista, compuesto por los campos *NombreEspectador* y *ListaReservas*, que contienen el nombre del espectador y la lista de sus reservas respectivamente.
- **tPosM** Posición de un elemento de la multilista.

- **NULO** Constante utilizada para representar posiciones nulas.

7.2 Operaciones incluidas en el TAD Multilista

Una precondición común para todas las operaciones (salvo MultilistaVacia) es que la multilista debe estar previamente inicializada.

- `MultilistaVacia (tMultilista)` → `tMultiLista`
  Crea una multilista vacía.
- `esMultilistaVacia (tMultilista)` → `Boolean`
  Determina si la multilista está vacía
- `PrimeroM (tMultilista)` → `tPosM`
  Devuelve la posición del primer elemento de la multilista (o NULO si la multilista está vacía).
- `UltimoM (tMultilista)` → `tPosM`
  Devuelve la posición del último elemento de la multilista (o NULO si la multilista está vacía).
- `SiguienteM (tMultilista, tPosM)` → `tPosM`
  Devuelve la posición del siguiente elemento a la posición indicada, en la multilista (o NULO si la posición no tiene siguiente).
  PreCD: La posición tiene que ser válida.
- `AnteriorM (tMultilista, tPosM)` → `tPosM`
  Devuelve la posición del anterior elemento a la posición indicada, en la multilista (o NULO si la posición no tiene anterior).
  PreCD: La posición tiene que ser válida.
- `InsertarOrdenadoM (tMultilista, tNomE, tListaR)` → `tMultilista, Boolean`
  Inserta en la multilista un nuevo espectador con nombre **tNomE** y su lista de reservas correspondiente. La multilista quedará ordenada con respecto a este campo. Devuelve un valor falso si no hay memoria suficiente para realizar la operación.
- `BorrarM (tMultilista, tPosM)` → `tMultilista`
  Borra de la multilista el espectador que está en la posición indicada.
  PreCD: La posición tiene que ser válida y la lista de reservas asociada debe estar vacía.
- `ObtenerDatoM (tMultilista, tPosM)` → `tNomE, tListaR`
  Devuelve el dato situado en la posición indicada de la multilista.
  PreCD: La posición tiene que ser válida.
- `ActualizaDatoM(tMultilista, tPosM, tNomE, tListaR)`→ `tMultilista`
  Actualiza el dato situado en la posición indicada con nombre **tNomE** y lista de reservas **tListaR**.
  PreCD: La posición tiene que ser válida.
- `BuscarM (tMultilista, tNomE)` → `tPosM`
  Devuelve la posición del elemento con nombre **tNomE** (o NULO si el elemento no existe).

8 Lectura de los ficheros

Para facilitar el desarrollo de la práctica se proporciona un fichero "`lectura.pas`". Este fichero contiene un ejemplo de lectura y manejo de los distintos ficheros mencionados. También se proporciona un ejemplo de prueba de los ficheros `platea.txt` y `operaciones.txt`.

9 Normas de realización

- Estructura que deberá de tener cada programa/unit desarrollado
  - Encabezamiento del programa/unit. Constará la siguiente información entre comentarios

```
TÍTULO: Prácticas de EDI
SUBTÍTULO: Practica 2
AUTOR 1: _____________________________ LOGIN 1: __________
AUTOR 2: _____________________________ LOGIN 2: __________
GRUPO: E.I / E.T.I.X. / E.T.I.S
FECHA: __/__/____
```

  - Cuerpo del programa/unit
    - El código irá comentado. Los comentarios han de ser concisos pero explicativos.
    - Después de la cabecera de cada procedimiento o función se incluirá lo siguiente: objetivo, entradas, salidas, precondiciones (condiciones que han de cumplir las entradas para el correcto funcionamiento de la subrutina) y poscondiciones (otras consecuencias de la ejecución de la subrutina que no quedan reflejadas en la descripción del objetivo o de las salidas).
- Criterios de valoración de la práctica:
  - *Eficacia*: que se cumplan las especificaciones.
  - *Control de Errores*: Control intensivo de todos los errores de ejecución que sea posible detectar.
  - *Claridad*: que el programa se pueda entender con facilidad, que contenga comentarios oportunos, indentación adecuada, nombres de variables significativos, etc.
  - *Modular*: que esté construido con módulos intercambiables y reutilizables.

10 Normas de entrega

- Las prácticas son OBLIGATORIAS y serán realizadas en grupos de DOS PERSONAS.
- La entrega de TODAS las prácticas en las FECHAS indicadas es requisito IMPRESCINDIBLE para aprobar la asignatura en la convocatoria ordinaria de JUNIO. Para las convocatorias de SEPTIEMBRE y DICIEMBRE las prácticas serán las mismas pero se fijarán otras fechas de entrega.
- Las prácticas entregadas tendrán que ser compilables en FreePascal en los ordenadores del laboratorio de docencia asignado.
- **Fecha límite de entrega: 4 de Junio de 2008.**
- Forma de entrega: Las prácticas quedarán depositadas en la red según el procedimiento del CECAFI. No se admitirán discos ni papel. El proceso para depositarlas será el siguiente:
  1. Conectarse a las máquinas **xurxo** o **limia**.
  2. Situarse en el directorio /PRACTICAS/XXXX/EDI/P2 donde XXXX es EI, ETIX o ETIS
  3. Situarse en el directorio que coincida con el login del usuario y copiar allí la práctica (sólo los ficheros fuentes `.pas`). IMPORTANTE: El nombre del fichero fuente que contenga el programa principal deberá ser `principal.pas`.
  4. Pasada la fecha de entrega no se permitirá el acceso a estos directorios.