

| | | |
|---|------------------|-----------------------|
| Algoritmos y Programación II (75.41) Cátedra Lic. Gustavo Carolo | | Parcial 15/05/2006 |
| Tema 1 | Ejercicio | Puntaje |
| Apellido y Nombre | | |
| Padrón | | |
| Cuat. Cursada: | | |
| Nota Final | | |
| Corrigió | | |

1) Se decidió implementar un TDA “SIMPRESIONES” que registra la información de las impresoras y documentos a imprimir un sistema de impresión de una compañía.

El TDA “SIMPRESIONES” tiene la siguiente estructura:

- Lista impresoras (LS): con la siguiente estructura:
 - ID_impresora (*clave de ordenamiento de la lista*)
 - Cantidad de documentos en cola
 - Cola documentos pendientes
 - Documento a imprimir
 - Prioridad
 - Fecha_hora
 - Páginas a imprimir
- Lista documentos impresos (LS) con la siguiente estructura:
 - Prioridad (*clave1 ordenamiento*)
 - Fecha_hora (*clave2 ordenamiento*) (*asumir un tipo_fecha que soporta las comparaciones < == > !=*)
 - Documento impreso

Se pide:

- a) Definir todos los elementos de la estructura dada.
- b) Desarrollar una función abstracta (indicando PRE y POST condiciones), llamada “**Imprimir**”, que reciba los siguientes parámetros: el TDA ADM_PEDIDOS y un total de páginas a imprimir, y que procese (imprima) para cada impresora de la lista de impresoras la cantidad de páginas pasadas, considerando que si un documento tiene más páginas de las que se pueden imprimir deberá guardarse con el saldo de páginas a imprimir actualizado, sino (se pudieron imprimir todas las páginas del documento) se eliminará el documento y se agregará en la lista de documentos impresos, luego se pasará a procesar (imprimir) el siguiente documento con las páginas que queden pendientes para la impresora, repitiendo el procedimiento. Considerar los errores y/o excepciones que crea necesarios y devolverlos como resultado de la primitiva.

Aclaraciones:

- Todas las listas son del tipo lista_simple y el orden por uno de sus campos es por cuenta de la aplicación y **NO** de las primitivas.
- Si una lista está ordenada por un campo **NO** debe recorrerse innecesariamente en una búsqueda por este campo.
- Una lista ordenada por dos claves se ordena en primer orden por la primera clave cuando hay elementos con igual clave1 se ordenan dentro esa clave por la segunda

2) Se pide:

- a) Definir el TDA_LS Lista simple (implementándola con punteros).
- b) Desarrollar una primitiva llamada “**Borrar siguiente corriente**”, que reciba como parámetro una lista simple (TDA_LS) defin en el punto a), y elimine el elemento siguiente del corriente. La primitiva devuelve el resultado de la operación. Definir las po condiciones. **Pre-condición:** Lista no vacía

Aclaraciones:

- No se pueden usar primitivas de tipo, como así tampoco otras estructuras auxiliares.
- No debe recorrerse la lista más de lo necesario.

Condiciones para aprobar el parcial:

- Deben estar hechos los dos ejercicios. Si alguno no está hecho o está Mal no se aprueba. Los ejercicios tienen que cumplir con lo pedido en enunciado.
- La resolución del ejercicio 1) tiene que demostrar que el alumno sabe utilizar las estructuras descriptas en clase en forma abstracta, y conoce las primitivas de las mismas, y sus pre y post condiciones.
- La resolución del ejercicio 2) tiene que demostrar que el alumno conoce la implementación de las primitivas de los TDA vistos en clase, funcionamiento, pre y post condiciones, como así también el manejo de punteros.

Entregar Teoría y Práctica en hojas separadas.

```
typedef struct {  char[20]  nombre;
                  int      prioridad;
                  tfecha   fecha;
                  int      pag_pend;
                } TElem_cp;
```

```
typedef struct {  int      cantidad;
                  int      id_imp;
                  TCola    c_pend;
                } TElem_lsi;
```

```
typedef struct {  int      prioridad;
                  tfecha   fecha;
                  char[20] nombre;
                } TElem_lsf;
```

```
typedef struct { TListaSimple ls_imp;
                TListaSimple ls_final;
                } TDA_S;
```

1) int Imprimir (TDA_S *S, int cant, int *fallidos)

```
{
  *(fallidos)=0;
  if ! (ls_vacia (S→ls_imp)) && (cant>0)
  {
    TElem_lsi ei;
    int error_proc=0;
    int error_mov=ls_moverCte (&S→ls_imp, PRIM);
    while !(error_mov)
    {
      ls_elemCte (S→ls_imp, &ei);
      Procesar (&S→ls_final, &ei, &error_proc, cant, fallidos);
      if (error_proc)
        return 1;
      ls_actualCte (&S→ls_imp, &ei);
      error_mov=ls_moverCte (&S→ls_imp, SIG);
    }
    return 0;
  }
}
```

void Procesar (TLista_f *lsf, TElem_lsi *ei, int *error_proc, int cant, int *fallidos)

```
{
  TMov pos;
  TElem_lsf ef;
  TElem_cp ecp;
  int error_ins=0;
  for (int cont=0, ei→cantidad, cont++)
  {
    c_sacar (&ei→c_pend, &ecp);
    if (cant < ecp→pag_pend)
    {
      ecp→pag_pend= ecp→pag_pend - cant;
      c_agregar (&ei→c_pend, &ecp);
    }
    else
    {
      Copiar_campos (ecp, &ep);
      Buscar_lugar (lsf, ep, &pos);
      *(fallidos)= *(fallidos) + error_ins=ls_insertar (lsf, &ef, pos);
      error_ins=0;
    }
  }
}
```

```
void Buscar_lugar (TLista_f *lsf, TElem_lsf ef, TMovim *pos)
```

```
{
    TElem_lsf aux;
    int error_mov;
    *(pos)=SIG;
    ls_elemCte (*lsf, &aux);
    if Mayor (aux, ef)
        *(pos)=PRIM;
    error_mov=ls_moverCte (lsf, *pos);
    while !(error_mov) && !Mayor (aux, ep)
    {
        ls_elemCte (*lsf, &aux);
        *(pos)=SIG;
        error_mov=ls_moverCte (lsf, *pos);
    }
}
```

```
void Copiar_campos (TElem_cp ecp, TElem_lsf *ef)
```

```
{
    ef→prioridad=ecp.prioridad;
    strcpy (ef→fecha, ecp.fecha);
    strcpy (ef→nombre, ecp.nombre);
}
```

```
int Mayor (TElem_lsf e1, TElem_lsf e2)
```

```
{
    return (e1.prioridad > e2.prioridad) || (e1.prioridad == e2.prioridad && strcmp (e1.fecha, e2.fecha) == -1)
}
```

2) PRE: Lista creada y no vacía.

POST: Devuelve 1 si se eliminó el siguiente del corriente, 0 en todo otro caso.

```
int Borrar_siguiente_corriente (TDA_LS *ls)
```

```
{
    if !(ls→Cte→Sig)
        return 0;
    TNode_Simple *paux=ls→Cte→Sig;
    ls→Cte→Sig=paux→Sig;
    free (paux→Sig);
    free (paux)
}
```