Soluciones examen parcial, 1 término, 2012

Tema 1

```
int listCompare(List *L1, List *L2, fnComp fn)
{
Nodelist iterator1,iterator2;

if (listIsEmpty(L1)|| listIsEmpty(L2)) return(0);
if (listGetSize(L1)!=listGetSize(L2))  return(0);

iterator1=listGetHeader(L1);
iterator2=listGetHeader(L2);

while(iterator1!=NULL){
        if (fn(iterator1,iterator2) !=1) return (0);
        iterator1=nodeListGetNext(L1);
        iterator2=nodeListGetNext(L2);
  }
return(1);
}
```

```
int equal(NodeList *N1,NodeList *N2){    //funcion para comparar
if (integerGet(nodeListGetCont(N1)==integerGet(nodeListGetCont(N2))
                return (1);
return(0);
}
```

Tema 2

salida=2,2,6,4,10,6,14,8,18,10

Tema 3

Posible solución

<Tweeter>:={<usuario>}
<usuario>:= <id> <mensajes_propios> <ultimos_mensajes> <usuario_sigo>
<usuario_sigo>:={ref_usuario}
<mensajes_propios>={<mensaje>}   //lista
<ultimos_mensajes>={<mensaje><iterador>} //cola FIFO estática
<iterador>=1|2|...|10
<id>={<string>}
<mensaje>:=<string> <tiempo>
<string>:={<caracter>}
<caracter>='a'| 'b'|....|'z'
<tiempo>:=<mes><dia><hora><min><seg>
<mes>=1|2|...|12
<dia>=1|2|...|31
<hora>=0|2|...|23

<min>=1|2|...|59
<seg>=1|2|...|59

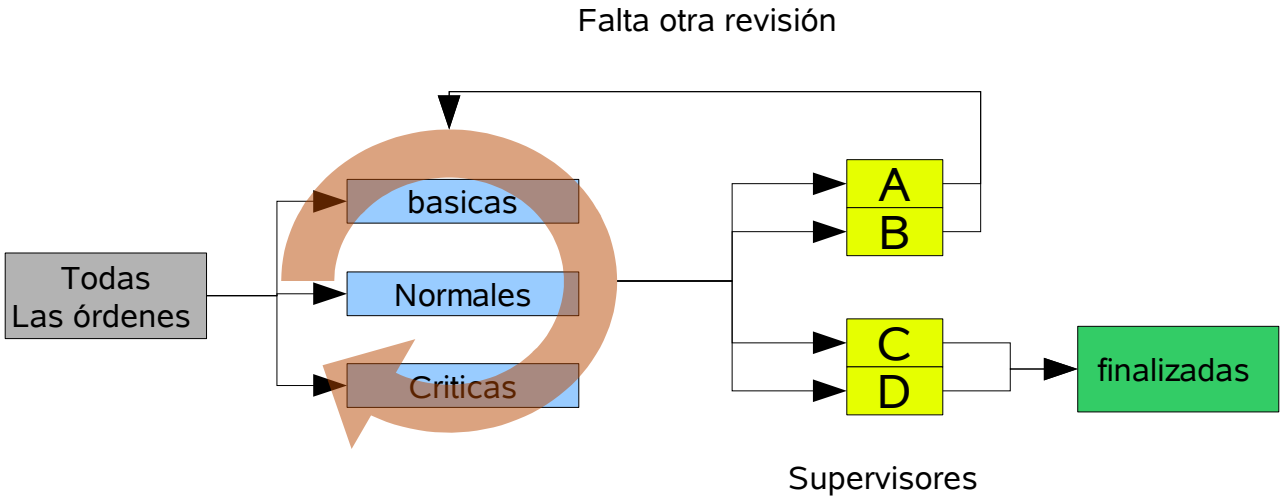
Los usuarios se pueden almacenar en una lista <tweeter>, cada usuario tiene un <id> una lista <mensajes_propios> con los mensajes que ha escrito, una cola de prioridad <ultimos_mensajes> con los mensajes de las personas que sigue y una lista <seguido_por> por donde están las referencias de los usuarios a los que sigue. Los mensajes ademas del string con el texto tiene el tiempo con al cual fue escrito. La información de tiempo sirve para buscar los mensajes más recientes.

Algoritmo (cualquier pseudo-codigo es valido>:

```
Por cada <usuario> en <Tweeter>
        for <id> en <usuario_sigo>
                if <usuario_sigo>.<mensajes_propios>
                        enqueue(<ultimos_mensaje><mensajes_propios>)
Por cada <usuario> en <Tweeter>
        for i=1 to 10
                printf(dequeue(<ultimos_mensaje>)
```

Tema 4

Posible esquema de solución




```
typedef struct orden{
        int id; //identificador de pedido
        int tipo,num;  //tipo=basica,norm,critica  num=0,1,2,3
        supervisor * último_supervisor;
}

typedef struct supervisor{
        char id; // 'A','B','C','D'
        int ordenes_procesadas //inicializado a 0
};
```

```
Queue *ordenes[3], *Q;   //0=basicas,1=normales,2=criticas
List * finalizadas;
supervisor supervisores[4];  //lleno con los datos de los supervisores

void Procesar(Queue * Q,supervisores super[]){
NodeList *orden;
int i=0, seguir_procesando=1;

while (!queueIsEmpty(Q)){   //clasifico ordenes en cada cola
        orden=queueDequeue(Q);
       queueEnqueue(ordenes[ordenTipoGet(orden)],orden);
}

while (seguir_procesando){
        orden=queueDequeue(ordenes[i]);
        switch ( ordenNumGet(orden)) {//numero de veces revisada
        case 0: revisarAoB(order,super);break;
        case 1: if ordenTipoGet(orden)<2 { revisarAoB(order,super);break;}
                else {revisarCoD(order,super);break;}
        case 2 :if  ordenTipoGet(orden)<2 { revisarCoD(order,super);break;}
                else {ListAdd(finalizadas,orden); break;}
        case 3 :  if  ordenTipoGet(orden)<2 {ListAdd(finalizadas,orden); break;}
        }
        i=(i+1)%3;
        if (queueIsEmpty(ordenes[0])==queueIsEmpty(ordenes[1])==queueIsEmpty(ordenes[2])==1)
                seguir_procesando=0;
}

for(i=0;i<4;i++)
        printf("%d", super[i]-> ordenes_procesadas);

}

void revisarAoB(NodeList * orden,supervisores super[])
{
if  (ordenSupervisorGet(order)==super[0])
        ordenSupervisorSet(order,super[1]);
else
        ordenSupervisorSet(order,super[0]);
}

void revisarCoD(NodeList * od,supervisores super[])
{
int s;
s=round((rand()/RAND_MAX))+3; //numero aleatorio 3 o 4
ordenSupervisorSet(od,super[s]);
}

void ordenSupervisorSet(NodeList *od, supervisor *s)
```

```
{
        orden * odt=ordenGet(od);
        odt->ultimo_supervisor=s; //asingno supervisor
        odt->num++:   //incremento el numero de revisiones de la orden
        s-> ordenes_procesadas++;   //incremento el numero de revisiones del supervisor
}

Supervisor* ordenSupervisorGet(NodeList *od)
{
        orden * odt=ordenGet(od);
        return(odt->ultimo_supervisor);
}

int ordenTipoGet(NodeList *od)
{
        orden * odt=ordenGet(od);
        return(odt->tipo);
}

int ordenNumGet(NodeList *od)
{
        orden * odt=ordenGet(od);
        return(odt->num);
}
orden* ordenGet(NodeList* od)
{
        return((orden*) nodeListGetCont(od));
}
```