1. (2 pt) Sea V un conjunto de n actividades, donde cada actividad a viene definida por su nombre, su tiempo de inicio, $t(a)_i$, y su tiempo de fin, $t(a)_f$. Dadas dos actividades a_j y a_k se dice que se solapan en el tiempo cuando la intersección de sus intervalos temporales no es vacía.

   Se pretende dividir el conjunto V en grupos de actividades que no se solapen en el tiempo. Para ello se ha diseñado el siguiente algoritmo:

   1. Sea S el conjunto de actividades ordenadas por el tiempo de inicio
   2. Sea Q una cola con prioridad de grupos de actividades, ordenadas por el tiempo de finalización de la última actividad en el grupo.
   3. Para cada actividad a en S, en el orden:
      - Seleccionar el grupo g de actividades que termina antes (de Q)
      - Si $t(a)_f < t(g)_f$, entonces crear un nuevo grupo h con la actividad a, e insertar h en Q.
      - En otro caso, insertar a en g, y actualizar la prioridad de g en Q

   Implementa el algoritmo anterior en la función agrupar, que recibe un vector de actividades ordenadas alfabéticamente por nombre de actividad

```
list< list<actividades> > agrupar(const vector<actividades> & V);
```

   donde el tipo actividades está definido como

```
struct actividades{
  string _nombre;   int _inicio;   int _fin;
};
```

```
#include <vector>
#include <queue>
#include <list>
#include <iostream>

using namespace std;

//Vamos a suponer que cuando una actividad tiene inicio = 0 y fin = 5, comienza
//en 0 y acaba en 5 (no inclusive). Esto viene dado por el enunciado, cuando
//dice "t(a)f < t(g)f", que debe estar mal porque debería ser "t(a)i < t(g)f"

//Incluyo un main que se puede usar para probar el código.


struct actividades {
  string _nombre;
```

```
  int _inicio;
  int _fin;
};

class ClasificaTiempoDeInicio {
  public:

  //Debe devolver true si a va después de b, porque la cola con prioridad ordena
  //de mayor a menor.
  bool operator()(const actividades &a,const actividades &b) {
    return a._inicio>b._inicio;
  }

};

class ClasificaFinUltAct {
  public:

  //Debe devolver true si a va después de b, es decir, si la última actividad
  //de la lista a acaba después que la última actividad de la lista b.
  bool operator()(const list<actividades> &a,const list<actividades> &b) {
    return a.back()._fin>b.back()._fin;
  }

};

list< list<actividades> > agrupar(const vector<actividades> & V) {

  //Al final de este fichero tenéis una versión de este código sin comentarios
  //ni códigos de comprobación.


  //---
  //Inicializo lo que voy a devolver como resultado.
  list< list<actividades> > res;
  //---


  //---
  //"Sea S el conjunto de actividades ordenadas por el tiempo de inicio"

  //Lo primero que hacemos es ordenar las actividades por el tiempo de inicio.
  //Para ello, las introducimos en una cola con prioridad que use el comparador
  //ClasificaTiempoDeInicio que hemos descrito arriba.

  //Insertamos las actividades en S.

  //Este algoritmo de ordenación se llama Heapsort (porque una cola con
  //prioridad es un APO que es un heap (montón)) y es O(nlogn) en todo caso.
```

```
//Tened en cuenta una cosa, las actividades pueden desordenarse por nombre!
//Cuando lo ejecuto, a5 se me pone antes de a4 (sus horas de comienzo son
//iguales así que no hay problema). Lo digo por si veis raro que se meta
//antes que a4.

priority_queue<actividades,vector<actividades>,ClasificaTiempoDeInicio> S;
vector<actividades>::const_iterator itv;
for (itv = V.begin();itv != V.end();itv++)
  S.push((*itv));
//---


//Con este código podríamos comprobar si están bien ordenadas.
////////////////////
actividades x;
priority_queue<actividades,vector<actividades>,ClasificaTiempoDeInicio> Sa(S);
while (!Sa.empty()) {
  x = Sa.top();
  Sa.pop();
  cout<<"Act "<<x._nombre<<" (ini:"<<x._inicio<<") (fin:"<<x._fin<<")"<<endl;
}
cout <<endl<<endl;
cout << "Recordad, partimos de que a1 está ya en un grupo" << endl<<endl;
////////////////////

//---
//"Sea Q una cola con prioridad de grupos de actividades, ordenadas por el
// tiempo de finalización de la última actividad en el grupo".

//Pues nada, una cola con prioridad de listas de actividades, la ordeno usando
//ClasificaFinUltAct que ordena por la última actividad insertada en un
//vector.

priority_queue<list<actividades>,
               vector< list<actividades> >,
               ClasificaFinUltAct> Q;

//Nos va a facilitar mucho las condiciones posteriores si en Q insertamos una
//primera lista con una tarea (la primera). Si no, más adelante tendremos que
//comprobar que Q no está vacío y que la lista que sacamos no está vacía.

//De esto me he dado cuenta a posteriori, al tenerme que poner a programar las
//condiciones.

list<actividades> primero;
actividades primera;
primera = S.top();
S.pop();
primero.push_back(primera);
Q.push(primero);
```

```
//---


//---
//"Para cada actividad a en S, en el orden"

while (!S.empty()) {

  actividades a;
  a = S.top();
  S.pop();

  //"- Seleccionar el grupo g de actividades que termina antes"

  list<actividades> g;
  g = Q.top();

  //"- Si t(a)f < t(g)f" (donde t(g)f es tf de la última actividad de g)

  //OJO, AQUI HAY UN ERROR EN EL ENUNCIADO, DEBERIA SER:
  //"- Si t(a)i < t(g)f"
  //PORQUE SI NO EL RESULTADO NO TIENE SENTIDO, SE SOLAPAN TAREAS

  if (a._inicio < g.back()._fin) {

    // "entonces crear un nuevo grupo h con la actividad a, e insertar h en Q"

    //Código de comprobación
    ////////////////////
    cout<<"Act "<<a._nombre<<" (ini:"<<a._inicio<<") (fin:"<<a._fin<<")"
        <<" va a un grupo nuevo"<<endl;
    ////////////////////

    list<actividades> h;
    h.push_back(a);
    Q.push(h);

    //Ojo, como no hemos hecho pop de g no tenemos que volver a meterlo.

  }

  //"en otro caso"

  else {

    //"insertar a en g, y actualizar la prioridad de g en Q."

    //Código de comprobación
    ////////////////////
```

```
      cout<<"Act "<<a._nombre<<" (ini:"<<a._inicio<<") (fin:"<<a._fin<<")"
          <<" va al grupo con hora de fin "<<g.back()._fin<<endl;
      ////////////////////

      g.push_back(a);

      //Para actualizar la prioridad, saco g de Q y lo vuelvo a meter.
      Q.pop();
      Q.push(g);

    }

    //Este código de comprobación muestra todos los grupos con su hora final de
    //cada actividad.
    ////////////////////
    priority_queue<list<actividades>,
                   vector< list<actividades> >,
                   ClasificaFinUltAct> Qa(Q);
    cout << "---"<<endl;
    while (!Qa.empty()) {
      cout << "Grupo con hora de fin "<<Qa.top().back()._fin<<endl;
      Qa.pop();
    }
    cout << "---"<<endl;
    ////////////////////

  }
  //---


  //---
  //Paso de la cola de prioridad Q a una list< list<actividad> >, que es lo que
  //tenemos que devolver.

  list<actividades> la;

  while (!Q.empty()) {
    la = Q.top();
    Q.pop();
    res.push_back(la);
  }

  //Devuelvo el resultado.
  return res;
  //--

}


int main(int argc,char *argv[]) {
```

```
  vector<actividades> V;
  actividades a[20];
  int i = 0;

  //Defino 20 actividades (he puesto tabuladores para que sea más cómodo).
  a[i]._nombre = "a1";     a[i]._inicio = 0;   a[i]._fin = 2;  i++;
  a[i]._nombre = "a2";     a[i]._inicio = 0;   a[i]._fin = 3;  i++;
  a[i]._nombre = "a3";     a[i]._inicio = 0;   a[i]._fin = 1;  i++;
  a[i]._nombre = "a4";     a[i]._inicio = 1;   a[i]._fin = 5;  i++;
  a[i]._nombre = "a5";     a[i]._inicio = 1;   a[i]._fin = 7;  i++;
  a[i]._nombre = "a6";     a[i]._inicio = 2;   a[i]._fin = 4;  i++;
  a[i]._nombre = "a7";     a[i]._inicio = 2;   a[i]._fin = 6;  i++;
  a[i]._nombre = "a8";     a[i]._inicio = 3;   a[i]._fin = 5;  i++;
  a[i]._nombre = "a9";     a[i]._inicio = 3;   a[i]._fin = 4;  i++;
  a[i]._nombre = "a10";    a[i]._inicio = 4;   a[i]._fin = 6;  i++;
  a[i]._nombre = "a11";    a[i]._inicio = 5;   a[i]._fin = 8;  i++;
  a[i]._nombre = "a12";    a[i]._inicio = 7;   a[i]._fin = 11;  i++;
  a[i]._nombre = "a13";    a[i]._inicio = 9;   a[i]._fin = 10;  i++;
  a[i]._nombre = "a14";    a[i]._inicio = 9;   a[i]._fin = 12;  i++;
  a[i]._nombre = "a15";    a[i]._inicio = 10;  a[i]._fin = 14;  i++;
  a[i]._nombre = "a16";    a[i]._inicio = 11;  a[i]._fin = 13;  i++;
  a[i]._nombre = "a17";    a[i]._inicio = 14;  a[i]._fin = 16;  i++;
  a[i]._nombre = "a18";    a[i]._inicio = 15;  a[i]._fin = 18;  i++;
  a[i]._nombre = "a19";    a[i]._inicio = 16;  a[i]._fin = 20;  i++;
  a[i]._nombre = "a20";    a[i]._inicio = 18;  a[i]._fin = 20;

  //Las inserto en v en orden alfabético (o equivalente, para el caso da igual).
  for (i = 0;i < 20;i++)
    V.push_back(a[i]);

  list< list<actividades> >res;
  res = agrupar(V);

  //Muestro los diferentes grupos:
  list< list<actividades> >::const_iterator it1;
  list<actividades>::const_iterator it2;
  i = 1;
  for (it1 = res.begin();it1 != res.end();it1++) {
    cout << "--" << endl << "Grupo numero " << i << endl << endl;
    for (it2 = (*it1).begin();it2 != (*it1).end();it2++) {
      actividades a;
      a = *it2;
      cout<<"Act "<<a._nombre<<" (ini:"<<a._inicio<<") (fin:"<<a._fin<<")"<<endl;
    }
    i++;
    cout << "--" << endl << endl;
  }

}
```

```
//Versión del código limpia:

/*

struct actividades {
  string _nombre;
  int _inicio;
  int _fin;
};

class ClasificaTiempoDeInicio {
  public:

  bool operator()(const actividades &a,const actividades &b) {
    return a._inicio>b._inicio;
  }

};

class ClasificaFinUltAct {
  public:

  bool operator()(const list<actividades> &a,const list<actividades> &b) {
    return a.back()._fin>b.back()._fin;
  }

};

list< list<actividades> > agrupar(const vector<actividades> & V) {

  //Inicializo lo que voy a devolver como resultado.
  list< list<actividades> > res;

  //"Sea S el conjunto de actividades ordenadas por el tiempo de inicio"
  priority_queue<actividades,vector<actividades>,ClasificaTiempoDeInicio> S;
  vector<actividades>::const_iterator itv;
  for (itv = V.begin();itv != V.end();itv++)
    S.push((*itv));

  //"Sea Q una cola con prioridad de grupos de actividades, ordenadas por el
  // tiempo de finalización de la última actividad en el grupo".
  priority_queue<list<actividades>,
                 vector< list<actividades> >,
                 ClasificaFinUltAct> Q;

  //Inserto una lista con una tarea (la primera).
  list<actividades> primero;
  actividades primera;
```

```
primera = S.top();
S.pop();
primero.push_back(primera);
Q.push(primero);

//"Para cada actividad a en S, en el orden"

while (!S.empty()) {
  actividades a;
  a = S.top();
  S.pop();

  //"- Seleccionar el grupo g de actividades que termina antes"
  list<actividades> g;
  g = Q.top();

  //"- Si t(a)i < t(g)f" (donde t(g)f es tf de la última actividad de g)
  if (a._inicio < g.back()._fin) {

    // "entonces crear un nuevo grupo h con la actividad a, e insertar h en Q"
    list<actividades> h;
    h.push_back(a);
    Q.push(h);

  }

  //"en otro caso"
  else {

    //"insertar a en g, y actualizar la prioridad de g en Q."
    g.push_back(a);
    Q.pop();
    Q.push(g);

  }

}

//Paso de la cola de prioridad Q a una list< list<actividad> >, que es lo que
//tenemos que devolver.
list<actividades> la;
while (!Q.empty()) {
  la = Q.top();
  Q.pop();
  res.push_back(la);
}

//Devuelvo el resultado.
return res;
```

```
}
*/
```