Estructura de Datos (2016-2017)
Grupo C
Grado en Ingeniería Informática
Universidad de Granada

Relación 3: STL

Mario Rodríguez Ruiz

8 de enero de 2017

Índice

1. Definir una función que permita invertir un objeto de tipo list. Los elementos que contiene la lista son enteros

```
1  void Invertir(const list<int> & lsource, list<int>& ldestino)
2  {
3    list<int>::const_reverse_iterator iter(lsource.rbegin()),
       iter_end(lsource.rend()) ;
4
5    while (iter != iter_end){
6      ldestino.push_back(*iter) ;
7      ++iter ;
8    }
9  }
```

2. Definir una función que obtenga una cola de prioridad (priority_queue) con la información de todos los alumnos, de manera que la prioridad se define de mayor a menor valor de selectividad.

```
1  bool operator <(const alumno &a1, const alumno &a2){
2    if(a1.nota_selectividad < a2.nota_selectividad)
3      return true ;
4    return false ;
5  }
6
7  void ObtenerPrioridad (const list<alumno> & alumnos,
     priority_queue<alumno>  & pq){
8    list<alumno>::const_iterator it;
9
10   for(it = alumnos.begin(); it != alumnos.end(); ++it)
11     pq.push(*it) ;
12
13 }
```

3. Dada la clase list instanciada a enteros, crear una función que elimine los elementos pares de la lista. Para implementarla hacer uso de iteradores.

```
void EliminaPares(list<int> & lsource){
  vector<int> borrar ;
  list<int>::iterator it;

  for (it=lsource.begin(); it != lsource.end(); it++)
    if((*it)%2==0)
      borrar.push_back(*it) ;

  for(unsigned i = 0 ; i < borrar.size() ; i++)
    lsource.remove(borrar[i]) ;
}
```

4. Ejercicios 5 y 6

```
class Alumnos{
  private:
    pair< string, set<string> > alumno;
  public:
    Alumnos(){}

    Alumnos(const pair<string, set<string> > a){
      alumno.first = a.first;
      alumno.second = a.second;
    }

    pair<string, set<string> > GetAlumno(){
      return alumno;
    }

    // Devuelve la cadena origen de un alumno.
    string GetAlumnoOrigen(){
      return alumno.first;
    }

    // Devuelve las asignaturas de un alumno.
    set<string> GetAlumnoDestino(){
```

```
23        return alumno.second;
24      }
25
26      // Añade una asignatura a un alumno
27      void AniadeAsignatura(string asig){
28        alumno.second.insert(asig);
29      }
30  };
31
32  class Matriculas{
33    private:
34      map<string, set<string> > matriculas; /**< Objeto de tipo
            map */
35
36    public:
37      Matriculas(){
38        clear() ;
39      }
40
41      void clear(){
42        matriculas.clear() ;
43      }
44
45      void Insertar(const string& dni,const string &cod_asig){
46      pair<string, set<string> > d ;
47
48      d.first = dni ;
49      d.second.insert(cod_asig) ;
50      Alumnos al(d) ;
51      AniadeAlumno(al) ;
52      }
53
54      // Añade un alumno o asignaturas de éste
55      void AniadeAlumno(Alumnos &p){
56      Alumnos encontrado = GetAlumno(p.GetAlumnoOrigen());
57      // Si el alumno no existe se crea uno nuevo
58      // sino se inserta el nuevo código en el existente
59      if(encontrado.GetAlumnoOrigen().size() == 0){
60      pair<string, set<string> > e = p.GetAlumno();
61      matriculas.insert(e);
62      }
63      else{
64      set<string> enc_destino = p.GetAlumnoDestino();
65      set<string>::iterator it_d;
```

```
66      // Se recorre todas las nuevas asignaturas que
67      // se van a ir insertando en el alumno original que ya
            existía.
68      for ( it_d=enc_destino.begin(); it_d!=enc_destino.end();
            ++it_d){
69      matriculas[p.GetAlumnoOrigen()].insert(*it_d) ;
70      }
71      }
72      }
73
74      // Busca un alumno coincidente con la cadena pasada como
            parámetro
75      Alumnos GetAlumno(const string origen){
76      map<string, set<string> >::iterator it;
77      it = matriculas.find(origen);
78
79      if (it==matriculas.end())
80      return Alumnos();
81      else{
82      Alumnos p(*it);
83      return p;
84      }
85      }
86
87      void Borrar(const string &dni,const string &cod_asig){
88      Alumnos encontrado = GetAlumno(dni);
89      // Si el alumno no existe no se hace nada
90      if(encontrado.GetAlumnoOrigen().size() != 0){
91      matriculas[encontrado.GetAlumnoOrigen()].erase(cod_asig)
            ;
92      }
93      }
94
95      list<string> GetAsignatras(const string &dni){
96      list <string> resultado;
97      Alumnos encontrado = GetAlumno(dni);
98
99      set<string> enc_destino = encontrado.GetAlumnoDestino();
100     set<string>::iterator it;
101
102     // Se recorren las asignaturas del alumno y se añaden a
            la nueva lista.
103     for ( it=enc_destino.begin(); it!=enc_destino.end(); ++it
            ){
```

```
104     resultado.push_back(*it);
105     }
106
107     return resultado;
108     }
109
110     list<string> GetAlumnos(const string &cod_asig){
111     list<string> resultado ;
112     const_iterator it;    //Iterador para recorrer las
            matriculas
113     set<string>::iterator ip;       //Iterador para recorrer
            un set
114
115     // Se recorre todas las matriculas.
116     for (it = begin(); it!=end(); ++it){
117     // Cada alumno de la matricula original
118     Alumnos p(*it);
119     set<string> destino = p.GetAlumnoDestino();
120     // Se recorre el destino de cada Alumnos.
121     for (ip = destino.begin(); ip!=destino.end(); ++ip){
122     Alumnos buscar = GetAlumno(cod_asig);
123     string dest_origen = buscar.GetAlumnoOrigen();
124     // Si la alumno no existe se crea una nueva
125     // sino se inserta la nueva traducción en la existente
126     if(dest_origen.size() != 0){
127     set<string> enc_destino = buscar.GetAlumnoDestino();
128     set<string>::iterator it_d;
129     // Se recorre todas las nuevas asignaturas que
130     // se van a ir insertando en la lista.
131     for ( it_d=enc_destino.begin(); it_d!=enc_destino.end();
            ++it_d){
132     resultado.push_back(*it_d) ;
133     }
134     }
135     }
136     }
137     return resultado ;
138     }
139
140   class const_iterator{
141   private:
142   map<string, set<string> >::const_iterator it;
143   public:
144   const_iterator & operator++(){
```

```
145   ++it;
146   return *this;
147   }
148
149   const_iterator & operator--(){
150   --it;
151   return *this;
152   }
153
154   Alumnos operator *(){
155   Alumnos p(*it);
156   return p;
157   }
158
159   bool operator ==(const const_iterator &i){
160   return i.it==it;
161   }
162
163   bool operator !=(const const_iterator &i){
164   return i.it!=it;
165   }
166
167   friend class Matriculas;
168   };
169
170
171   const_iterator begin() const {
172   const_iterator i;
173   i.it=matriculas.begin();
174   return i;
175   }
176
177
178   const_iterator end() const {
179   const_iterator i;
180   i.it=matriculas.end();
181   return i;
182   }
183
184
185   const_iterator buscar(string f) {
186   const_iterator i;
187   i.it=matriculas.find(f);
188   return i;
```

```
189     }
190
191 };
```