Relación STL

Alumno : **Alberto Llamas González***

2º Grado en Ingeniería Informática

Índice ejercicios

Ejercicios resueltos

Ejercicio 1

Implementar una función *int stablepartition (int x, vector & A);* que, dado un entero x y un vector de n enteros A, A= (A0,A1,...,An-1) reordene el contenido del vector alrededor de un índice k de forma que los elementos del subvector A[0..k] sean menores o iguales que x, y los elementos del subvector A[k+1..n-1] sean mayores que x, y devuelva ese índice k (k=-1 si todos los elementos de A son mayores que x).

```
int stablepartition(int x, vector<int> &A){
    int i = 0;
    int j = 0, k=0;
    vector<int> aux, mayor;
    while (i < A.size()){
        if (A[i] <= x){
           int num = A[i];
           aux.push_back(num);
        }
```

```
        i++;
    }

    if (aux.size() != 0){
        return aux.size()-1;
    }
    else
    {
        return -1;
    }

}
```

Ejercicio 3

Considere el problema de generar todas las subsecuencias ordenadas de la secuencia

X=(1,2,...,n).
Por ejemplo, si n = 4 las subsecuencias ordenadas de X = (1, 2, 3, 4) son: (1), (12), (123), (124), (13), (134), (14) (2), (23), (234) (24), (3), (34) y (4). Esta construcción se puede implementar mediante el uso de una pila S bajo las siguientes reglas:

- Inicializar la pila con el elemento 1.
- Si el tope t de la pila verifica t < n entonces apilamos t+ 1.
- Si t = n, entonces lo desapilamos y, a continuación, si la pila no quedara vacía, incrementamos el nuevo tope de la misma.
- El algoritmo termina cuando la pila queda vacía.

```
void imprimeAlReves(stack<int> &S){
    stack<int> aux;
    int i;
    while(!S.empty()){
        i = S.top();
        aux.push(i);
        S.pop();
    }

    cout << "Pila S: ";
    while(!aux.empty()){
        i = aux.top();
        cout << i << " ";
        S.push(i);
        aux.pop();
    }
    cout << endl;
}
void subsecuencias_ordenadas(int &n){
```

```
    stack<int> S;
    S.push(1);
    int i;
    while(!S.empty()){
        imprimeAlReves(S);
        i = S.top();
        if (n > i)
            S.push(i+1);
        else{
            S.pop();
            if (!S.empty()){
                i = S.top();
                S.pop();
                S.push(i+1);
            }
        }
    }

}
```

Ejercicio 6

Dada una pila P de enteros implementar una función

void transformarpila(stack & p);

que transforme una pila p en otra en la que los elementos aparezcan en el mismo orden original y habiendo sido eliminados los elementos que siendo consecutivos aparezcan repetidos.

Ejemplo:
P=<1,1,2,3,3,4,5,5,1,1,9,8,7,7,3> pasaría a quedar como P = <1,2,3,4,5,1,9,8,7,3>

```
void imprime(stack<int> &S){
    stack<int> aux;
    int i;
    cout << "Pila S: ";
    while(!S.empty()){
        i = S.top();
        cout << i << " ";
        S.pop();
    }
    cout << endl;
}

void transformarpila(stack<int> &p){
    stack<int> aux;
    p.swap(aux);
    int tope;
    while(!aux.empty()){
```

```
        tope = aux.top();
        if (aux.size() != 1){
            aux.pop();
            while(tope == aux.top() && aux.size() != 1){
                aux.pop();
            }
            p.push(tope);
        } else if (aux.size() == 1){
            if (tope == aux.top())
                aux.pop();
            else
            {
                p.push(tope);
                aux.pop();
            }

        }
    }
}
```

Ejercicio 7

Implementar una función

void rotación(queue & C);

que saque una cierta cantidad de enteros del frente de la cola C y los vuelve a insertar al final de cola, de forma que quede en el frente el primer número par que haya en la cola.
Por ejemplo, si C={1,3,5,2,4} ====> C={2,4,1,3,5}

```
void rotacion(queue<int> &C){
    int contador = 0; bool par = false;
    while (!par && contador < C.size()){
        if (C.front() % 2 != 0){
            C.push(C.front());
            C.pop();
            contador++;
        } else
        {
            par= true;
        }

    }

}
```

Ejercicio 10

Usando el TDA list, construir una función

void agrupar_elemento (list & entrada, int k)

que agrupe de forma consecutiva en la lista de entrada todas las apariciones del elemento k en la lista, a partir de la primera ocurrencia. Por ejemplo:
si entrada={1,3,4,1,4} y k = 1, entonces entrada={1,1,3,4,4}, o
si, entrada={3,1,4,1,4,1,1} y k = 1, entonces entrada={3,1,1,1,1,4,4}

```
void agrupar_elemento(list<int> & entrada, int k){

    list<int> aux, aux2;
    bool k_encontrado= false;
    aux = entrada;
    entrada.clear();
    while(!aux.empty()){
        if (aux.front() == k && !k_encontrado){
            entrada.push_back(aux.front());
            aux.pop_front();
            aux2 = aux;
            while(!aux2.empty()){
                if (aux2.front() != k){
                    aux2.pop_front();
                } else{
                    entrada.push_back(aux2.front());
                    k_encontrado = true;
                    aux2.pop_front();
                }
            }
        } else if (aux.front() != k){
            entrada.push_back(aux.front());
            aux.pop_front();
        } else {
            aux.pop_front();
        }
    }
}
```

Ejercicio 11

Dado un vector de listas VL y una lista L, implementar una función:

bool iscomb (vector<list>> VL, list L);

que devuelva true si L es una combinación de las listas de VL en alguna permutación dada. Cada una de las VL[j] debe aparecer una y sólo una vez en L. Por ejemplo: si VL=[(1,2,3),(4,5,6),(7,8)], y L=(7,8,4,5,6,1,2,3) entonces iscomb(VL,L)->true. Pero si L=(3,2,1,7,8,4,5,6)->false.
Ayuda: Escribir una funcion bool isprefix(Lpref,L); que retorna true si la lista Lpref es prefijo de L.Primero chequear que la longitud de L debe ser igual a la suma de las longitudes de los VL[j]. Si pasa este test se procede en forma recursiva. Para cada VL[j] se determina si es prefijo de L. Si ninguna de ellas es prefijo entonces retorna false. Si una (o varias de ellas) lo es entonces se aplica recursivamente iscomb() para determinar si L-VL[j] es una combinación de VL-VL[j]. Es

decir se quita VL[j] del comienzo de L y la posicion j de VL. La recursión se corta cuando L es vacía.

```
bool iscomb(vector<list<int>> VL, list<int> L)
{
    list<int> aux;
    int longitudVL = 0;
    static bool is_comb;
    //Miro si la suma de las longitudes de VL es igual a la de L
    for (int i = 0; i < VL.size(); i++){
        aux = VL[i];
        longitudVL+= aux.size();
    }

    if (longitudVL != L.size())
        return false;

    //Empiezo recursividad
    if (VL.size() == 1){ //Caso base
        if(VL[0] == L) is_comb = true;
        else is_comb = false;
    }

    for (int i = 0; i < VL.size(); i++){
        if (ispref(VL[i], L)){
            for (int j = 0; j < VL[i].size(); j++)
                L.pop_front();
            VL.erase(VL.begin()+i);
            iscomb(VL,L);
        }
    }
    return is_comb;
}

bool ispref(list<int> Lpref, list<int> L)
{
    while (!Lpref.empty())
    {
        if (Lpref.front() != L.front())
            return false;
        else
```

```
        {
            L.pop_front();
            Lpref.pop_front();
        }
    }
    return true;
}
```

Ejercicio 12

Implementar una función:
bool nullsum(list &L, list &L2);

que dada una lista L, devuelve true si hay un rango de iteradores [p,q)) tal que su suma de los elementos en el rango sea 0. En caso afirmativo debe retornar además el rango (uno de ellos, porque puede no ser único) a través de L2. En caso negativo L2 debe quedar vacía. El algoritmo debe ser como mucho cuadrático.

Por ejemplo: si L=(-10, 14, -2, 7, -19, -3, 2, 17, -8, 8) entonces debe retornar true y L2=(14, -2, 7, -19), ó L2=(-8, 8). Si L=(1,3,-5) entonces debe retornar false y L2=().

```
bool nullsum(list<int> &L, list<int> &L2)
{
    int suma = 0;
    bool encontrado = false;
    list<int> aux;
    int cont = 0;

    while ((L.size() != 0) && (!encontrado))
    {
        suma = L.front();
        if (L.front() == 0)
        {
            L2.push_back(0);
            return true;
        }
        aux = L;
        aux.pop_front();

        while ((aux.size() != 0) && (!encontrado))
        {
            suma = suma + aux.front();
            aux.pop_front();
            if (suma == 0)
                encontrado = true;
            cont++;
        }

        if (encontrado)
```

```
        {
            for (int i = 0; i < (cont + 1); i++)
            {
                L2.push_back(L.front());
                L.pop_front();
            }
        }
        else
        {
            cont = 0;
            L.pop_front();
        }
    }

    return encontrado;
}
```

Ejercicio 13

Implementar una función:

void expand(list &L,int m);
que transforme los elementos de una lista L de tal forma que todos los elementos de L resulten ser menores o iguales que m, pero de tal forma que su suma se mantenga inalterada. Para esto, si un elemento x es mayor que m entonces, lo divide en tantas partes como haga falta para satisfacer lacondición; por ejemplo si m=3 podemos dividir a 10 en 3,3,3,1. Es decir si L= (7,2,3,1,4,5), entonces después de hacer expand(L,2) debe quedar

L=(2,2,2,1,2,2,1,1,2,2,2,2,1).

```
void expand(list<int> &L, int m){
    list<int>::iterator it = L.begin();
    while (it != L.end())
    {
        int a_poner = *it;
        it = L.erase(it);
        while(a_poner > 0){
            int val;
            if (a_poner>m)
                val = m;
            else
                val = a_poner;

            it = L.insert(it,val);
            a_poner = a_poner-val;
            it++;
        }
    }
```

```
}
```

Ejercicio 14

Implementar una función

void subsecuencia (list &L);
que dada una lista de enteros, elimine todos aquellos que no sean mayores que todos los

anteriores. Ejemplo: L={1,3,4,2,4,7,7,1} ==> L={1,3,4,7}

```
void subsecuencia(list<int> &L){

    list<int> aux;
    aux = L;
    int maximo = aux.front();
    L.clear();
    L.push_back(maximo);
    aux.pop_front();
    while(!aux.empty()){
        if (maximo < aux.front()){
            maximo = aux.front();
            L.push_back(maximo);
            aux.pop_front();
        } else {
            aux.pop_front();
        }
    }
}
```

Ejercicio 15

Usando la clase set de la STL, construir una función que divida un conjunto de enteros c en dos subconjuntos cpar y cimpar que contienen respectivamente los elementos pares e impares de c y que devuelva true si el número de elementos de cpar es mayor que el de cimpar y false en caso contrario.

```
bool maspares(set<int> &s){
    set<int>::iterator it;
    set<int> cpar;
    set<int> cimpar;

    for (it=s.begin(); it != s.end(); ++it){
        if ((*it)%2 == 0)
            cpar.insert(*it);
        else
            cimpar.insert(*it);
    }
```

```
    return (cpar.size() > cimpar.size());

}
```

Ejercicio 16

Dados dos multiset con elementos enteros, implementar la función:
multiset multi_interseccion (const multiset & m1, const multiset & m2)

que calcula la intersección de dos multiset: elementos comunes en los dos multiset repetidos tantasveces como aparezcan en el multiset con menor número de apariciones del elemento.

```
multiset<int> multi_interseccion (const multiset<int> & m1, const multiset<int>
& m2){
    multiset<int> salida;
    multiset<int>::iterator it1 = m1.begin(),it2 = m2.begin();

    while(it1 != m1.end() && it2 != m2.end()){
        if (*it1 < *it2){
            ++it1;
        } else if(*it1 > *it2){
            ++it2;
        } else{
            salida.insert(*it1);
            ++it1; ++it2;
        }
    }
    return salida;
}
```

Ejercicio 17

Dado un vector de conjuntos vector< set > VS, implementar una función bool included(vector< set > &VS);

que devuelve true si los conjuntos del vector VS son subconjuntos propios en forma consecutiva, es decir, si Sj está incluido en Sj+1 para j = 0, ..., n-2. Debe devolver false en caso contrario.

```
set<int> differencia(set<int> &s1, set<int> &s2)
{
    set<int>::iterator it1 = s1.begin();
    set<int> differencia;
    while (it1 != s1.end())
    {
        if (s2.find(*it1) == s2.end()){
            differencia.insert(*it1);
            ++it1;
```

```
        } else
            ++it1;
    }
    return differencia;
}
//Un subconjunto propio(segun Google) es un subconjunto que esta incluido
dentro de otro que tiene al menos un elemento distinto
bool included(vector<set<int>> &VS)
{
    vector<set<int>>::iterator it = VS.begin();
    set<int> aux;

    for (; it != VS.end(); ++it)
    {
        vector<set<int>>::iterator it2 = VS.begin();
        ++it2;
        if (it2 == VS.end())
            break;
        aux = differencia(*it, *it2);
        if (!aux.empty())
            return false;
        aux = differencia(*it2, *it);
        if (aux.empty())
            return false;
        it = it2;
    }
    return true;
}
```

Ejercicio 18

Dadas dos map, M1 y M2, definidos como:
map<string,int> M1,M2;

siendo el primer campo el nombre de una persona y el segundo campo al número de seguidores. Obtener el mapa correspondiente a la unión en el que el número de seguidores será la suma de los seguidores en M1 y los seguidores en M2 para esa misma persona que aparece en M1.

```
map<string,int> Union(const map<string,int> &M1, const map<string,int> &M2){
    map<string,int> salida(M1);
    map<string,int>::const_iterator it;

    for(it = M2.begin(); it != M2.end();++it){
        if(M1.find(it->first) == M1.end()){
            pair<string,int> p(it->first,it->second);
            salida.insert(p);
        } else {
            map<string,int>::const_iterator its;
```

```
            its = salida.find(it->first);
            int suma = it->second + its->second;
            pair<string,int> p(it->first, suma);
            salida.insert(p);
        }
    }

    return salida;
}
```

Ejercicio 19

Dado un map<string,int> M queremos saber:

- El número de elementos en el map que hay entre dos claves en este caso de tipo string. Por ejemplo el número de elemento almacenados entre las entradas [patata,zanahoria].
- El número de elementos en el map que tiene como información asociada un valor en un rango,por ejemplo entre [20,100].

```
int NumElemIntervalo(const string& clave1, const string& clave2, const
map<string,int> &MP){
    map<string,int>::const_iterator it, itMin, itMax;
    int NumElem = 0;

    for(it = MP.begin(); it != MP.end(); ++it){
        if(clave1 == it->first)
            itMin = it;
        if(clave2 == it->first)
            itMax = it;
    }

    do{
        NumElem++;
        ++itMin;
    }while(itMin != itMax);

    return NumElem;
}

//b
int NumElemIntervalo(const int& clave1, const int& clave2, const
map<string,int> &MP){
    map<string,int>::const_iterator it;
    int NumElem = 0;

    for(it = MP.begin(); it != MP.end(); ++it){
        if(it->second >= clave1 && it->second <= clave2){
            NumElem++;
```

```
        }
    }

    return NumElem;
}
```

Ejercicio 20

1. Implementar una función:

   void apply_map(const list &L, map<int,int> &M, list &ML);

   que, dada una lista L y un map M devuelve en ML una lista con los resultados de aplicar M a los elementos de L. Si algún elemento de L no está en el dominio de M entonces el elemento correspondiente de ML no es incluido. No pueden usarse estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser O(n), donde n es el número de elementos en la lista (asumiendo que las operaciones usadas del map son O(1)).
   Por ejemplo, si L = (1, 2, 3, 4, 5, 6, 7, 1, 2, 3) y M = {(1, 2), (2, 3), (3, 4), (4, 5), (7, 8)} entonces después de hacer apply_map(L,M,ML), debe quedar ML = (2, 3, 4, 5, 8, 2, 3, 4).

```
void apply_map(const list<int> &L, const map<int,int> &M, list<int> &ML){
    list<int>::const_iterator it;

    for(it=L.begin(); it != L.end();++it){
        if(M.find(*it) != M.end())
            ML.insert(ML.end(), M.find(*it)->second);
    }
}
```

Ejercicio 21

Se desea construir un traductor de un idioma origen a un idioma destino. Una palabra en el idioma origen puede tener más de un traducción en el idioma destino.Dar una representación para el TDA Traductor usando el tipo .
Implementar la función insertar que añade una palabra del idioma origen junto con las traducciones en el idioma destino. Implementar la función consultar que obtiene las traducciones de una palabra en el idioma destino. Implementar la clase iteradora dentro de la clase Traductor para poder iterar sobre todas las palabras.

```
class Traductor{
    private:
        map<string, list<string>> datos;
    public:

    void insertar(const string &palabra, list<string> traduccion){
        pair<string, list<string>> p(palabra,traduccion);
        datos.insert(p);
    }
```

```
    list<string> consultar (const string &palabra){
        map<string,list<string>>::iterator i;
        list<string> consulta;
        consulta.clear();
        if((i=datos.find(palabra)) == datos.end()){
            break;
        } else
        {
            consulta.insert(i->second);
        }
        return consulta;
    }

    class iterator{
        private:
            map<string,list<string>>::iterator it;
        public:
            iterator(){}

            iterator& operator++(){
                ++it;
                return *this;
            }

            iterator& operator--(){
                --it;
                return *this;
            }

            pair<const string, list<string>> &operator*(){
                return *it;
            }

            bool operator== (const iterator& i){
                return i.it == it;
            }
            bool operator!= (const iterator& i){
                return i.it != it;
            }
            iterator& operator= (const iterator &i){
                it = i.it;
                return *this;
            }
            friend class Traductor;
    };
};
```

Ejercicio 22

Implementar un iterador que itere sobre las claves que sean números primos en una clase Diccionario definida como:

1 class Diccionario{
2 private:

3 map<int, list > datos;

4 .....
5 .....
6 };

Han de implementarse (aparte de las de la clase iteradora) las funciones begin() y end(). Se supone implementada una función bool primo (int x) que devuelve true si el entero x es primo.

```
class Diccionario{
private:
    map<int,list<string>> datos;
public:
    class iterator{
        private:
            map<int,list<string>>::iterator it;
            map<int,list<string>>::iterator final, inicio;
        public:
            pair<int,list<string>> &operator*(){
                return *it;
            }
            iterator& operator++(){
                while(true){
                    ++it;
                    if(primo(*it))
                        return *this;
                    else if(it == final)
                        return *this;
                }
            }

            iterator& operator--(){
                while(true){
                    --it;
                    if(primo(*it))
                        return *this;
                    else if(it == inicio)
                        return *this;
                }
            }

            bool operator==(const iterator& i){
```

```
                return i.it == it;
            }

            bool operator != (const iterator& i){
                return i.it != it;
            }

            friend class Diccionario;
    };

    iterator begin(){
        iterator i;
        i.it = datos.begin();
        i.inicio = datos.begin();
        i.final = datos.end();
        return i;
    }

    iterator end(){
        iterator i;
        i.it = datos.end();
        i.inicio = datos.begin();
        i.final = datos.end();
        return i;
    }
};
```