```cpp
#include <iostream>
#include <string>
#include <stdio.h>
#include "Timer.h"
#include <vector>

using namespace std;

typedef vector<int> polinomio;

using namespace std;

/**
 * @brief operaciones de apoyo al código posterior...
 */

int max(int x, int y){
    return x>y?x:y;
}

int min(int x, int y){
    return x<y?x:y;
}

/**
 * @brief Muestra un polinomio por pantalla...
 */
void muestra(const polinomio& p){
    if(p.size()>0){
        for(int i=p.size()-1; i>0; i--)
            cout<<p[i]<<"x"<<i<<" ";
        cout<<p[0];
    }
    cout<< endl;
}

/**
 * @brief Función SumaPolinomios que suma dos polinomios...
 */
polinomio sp(const polinomio& p1, const polinomio& p2){

    int n=min(p1.size(), p2.size());
    int i;
    polinomio resul;
    /**<Acumulamos el resultado de la suma de cada uno en resul*/
    for(i=0;i<n;i++){
        resul.push_back(p1[i] + p2[i]);
    }
    /**Si los grados no son iguales, acumulamos en resul el valor de los coeficientes no comprobados del mayor...*/
    if(p1.size()>p2.size()){
        resul.insert(resul.end(), p1.begin()+n, p1.end());
    } else if(p1.size()<p2.size()){
        resul.insert(resul.end(), p2.begin()+n, p2.end());
    }

    return resul;
}

/**
 * @brief Funcion RestaPolinomios que resta dos polinomios...
 */
polinomio rp(const polinomio& p1,const polinomio& p2){

    unsigned int n=min(p1.size(), p2.size()); /**<Obtenemos el menor grado de los polinomios...*/
    unsigned int i;
```

```
66      polinomio resul;
67
68      for(i=0;i<n;i++){   /**<Y acumulamos el resultado de la resta de cada uno en resul*/
69          resul.push_back(p1[i] - p2[i]);
70      }
71
72      /**Si los grados no son iguales, acumulamos en resul el valor de los coeficientes no comprobados del
mayor...*/
73      if(p1.size()>p2.size()){
74          resul.insert(resul.end(), p1.begin()+n, p1.end());
75      } else if(p1.size()<p2.size()){
76          resul.insert(resul.end(), p2.begin()+n, p2.end());
77          for(i=n; i<resul.size(); i++)   /**<Si el sustraendo es el de mayor grado, cambiamos de signo los
coeficientes...*/
78              resul[i]=-resul[i];
79      }
80
81      return resul;
82  }
83
84  /**
85   * @brief Multiplicación Clásica de dos polinomios cualquiera.
86   */
87  polinomio multPoliClasico(const polinomio& p1, const polinomio& p2){
88      polinomio res;
89      int n=p1.size(), m=p2.size();
90      res.resize(n+m-1, 0);
91
92      for (int i=0;i<n;i++){
93          for (int j=0;j<m;j++){
94              res[i+j]+=(p1[i]*p2[j]);
95          }
96      }
97      return res;
98  }
99
100 /**
101  * @brief Multiplicación de dos polinomios de igual grado basado en Divide y Vencerás.
102  */
103 polinomio multPoli(const polinomio& p1, const polinomio& p2){
104     int n=max(p1.size(), p2.size());
105     polinomio aux, auxx;
106     if(n<=2){       /**<Si el polinomio es lo suficientemente pequeño, llamamos al algoritmo clásico...*/
107         return multPoliClasico(p1,p2);
108     }
109     else{           /**<Si no, aplicamos divide y vencerás.*/
110         /**Partimos cada polinomio en dos más pequeños*/
111         int s=(n/2);
112         polinomio x;
113         x.insert(x.end(),p1.begin(), p1.begin()+s);
114         polinomio w;
115         w.insert(w.end(),p1.begin()+s, p1.end());
116         polinomio z;
117         z.insert(z.end(),p2.begin(), p2.begin()+s);
118         polinomio y;
119         y.insert(y.end(),p2.begin()+s, p2.end());
120
121         aux.clear();aux=sp(w,x);
122         auxx.clear();auxx=sp(y,z);
123         /**Creamos polinomios auxiliares para reducir las llamadas al algoritmo clásico
124         (wz+xy)=(w+x)(y+z)-wy-xz*/
125         polinomio r(multPoli(aux, auxx)),
126                   p(multPoli(w,y)),
127                   q(multPoli(x,z));
128         polinomio o;
129
```

```
        aux.clear();aux=rp(r,p);
        o=rp(aux,q);

        /**Agrupamos de nuevo los polinomios en uno solo
        cuadrando las posiciones donde debe quedar cada coeficiente
        insertando ceros por la izda a los polinomios intermedios obtenidos
        p1*p2= 102s(wy) + 10s(wz+xy) + xz
                  (p)   +    (o)     + (q)   */

        for(int i=0; i<2*s; i++){
            p.insert(p.begin(),0); //insertamos 2s posiciones iniciales con 0
            if(i%2==0)
                o.insert(o.begin(),0); //insertamos s posiciones iniciales con 0
        }
        aux.clear();aux=sp(o,q);
        auxx.clear();auxx=sp(p,aux);
        return auxx;
    }
}


int main(){
    Timer t;
    polinomio x,y;
    x.push_back(-5);
    x.push_back(-7);
    x.push_back(3);
    x.push_back(2);
    x.push_back(-5);
    x.push_back(-7);
    //x.push_back(3);

    y.push_back(6);
    y.push_back(1);
    y.push_back(-8);
    y.push_back(3);
    y.push_back(6);
    y.push_back(1);
    y.push_back(-8);

    polinomio d, clas;

    t.start();
    d= multPoli(y,x);
    t.stop();
    cout << "El tiempo de ejecucion es de " ;
    cout  << t.getElapsedTimeInMilliSec() << " ms" << endl;
    cout<<"Algoritmo DyV:  \n";muestra(d);

    t.start();
    clas=multPoliClasico(y,x);
    t.stop();
    cout << "El tiempo de ejecucion es de " ;
    cout  << t.getElapsedTimeInMilliSec() << " ms" << endl;
    cout<<"Algoritmo Clas:  \n";muestra(clas);
```