```cpp
/**
 * Projet C++    : Polynômes modulaires
 * Matricule     : 410031
 * Nom           : Requena López
 * Prénom        : Carlos
 * Année         : 2015-2016
 */

#include <iostream>

#include "StaticVect.hpp"
#include "DynamicVect.hpp"
#include "PolyMod.hpp"




int main() {


  StaticVector<int, 5> static_vector1;
  static_vector1[2] = 2;
  static_vector1[1] = 3;
  std::cout << static_vector1 << "\n"; // [ 0 3 2 0 0 ]

  DynamicVector<double> dyn_vector1(5, 3); // [ 3 3 3 3 3 ]
  DynamicVector<double> dyn_vector2(5, 2); // [ 2 2 2 2 2 ]

  std::cout << dyn_vector1 << "\n";
  std::cout << dyn_vector2 << "\n";

  dyn_vector1 =  2 - dyn_vector2; // With implicit conversion
  dyn_vector1[3] = 3.5;
  dyn_vector1 *= 3.2;

  std::cout << dyn_vector1 << "\n"; // [ 0 -6.4 -6.4 11.2 -6.4 ]

  Poly<double> poly1(size_t(5)); // Avoid calling conversion ctor

  poly1[0] = 5; poly1[2] = 3; poly1[3] = 1; poly1[4] = -2;
  std::cout << poly1 << "\n";   // -2x^4 + 1x^3 + 3x^2 + 5
  poly1 += 5;                   // -2x^4 + 1x^3 + 3x^2 + 10

  Poly<double> poly2(size_t(3));
  poly2[0] = 3; poly2[1] = 2; poly2[2] = -1;
  std::cout << poly2 << "\n";   // -1x^2 + 2x^1 + 3

  poly1 = poly1 * poly2;
  std::cout << poly1 << "\n";   // +2x^6 -5x^5 -7x^4  +9x^3 -1x^2  +20x^1
```

```
    +30

51  -poly1;                          // -2x^6 +5x^5 +7x^4  -9x^3 +1x^2  -20x^1
    -30

53  std::cout << poly1(2) << "\n"; // 6

55  // constexpr as non type parameter for template.
    // PolyMod<int, static_vector1, 4> polymod1;
57

59  return 0;
}
```

main.cpp

```cpp
#ifndef IVECT_H
#define IVECT_H


#include <cstdlib>
#include <iostream>

// Interface for vectors. All static
template<typename VectType, typename TYPE>
class IVect
{
protected:
  IVect() = default;
  ~IVect() = default;
private:
  const VectType& child() const {return *static_cast<const VectType*>(this)
   ;}
  VectType& child() {return *static_cast<VectType*>(this);}
public:
  friend std::ostream& operator<<(std::ostream& os, const VectType& v) {
    v.print(os); return os;}
  friend std::istream& operator>>(std::istream& is, VectType& v) {
    v.extract(is); return is;}
  friend VectType operator+(const VectType& v1, const VectType& v2) {
    return v1.add(v2);}
  friend VectType operator-(const VectType& v1, const VectType& v2) {
    return v1.sub(v2);}
  const TYPE operator[] (std::ptrdiff_t i) const {
    return child().get(i);}
  TYPE& operator[] (std::ptrdiff_t i) {
    return child().get(i);}
  void operator+=(const VectType &other) {
    child().addMe(other);}
  void operator*=(const TYPE &elem) {
    child().mulMe(elem);}
  void operator-=(const VectType &other) {
    child().subMe(other);}
  void operator-() {
    child().minus();}

};

#endif /* IVECT_H */
```

IVect.hpp

```cpp
#ifndef STATICVECT_H
#define STATICVECT_H

#include "IVect.hpp"

// Forward declaration
template <typename TYPE>
class DynamicVector;

// size_t is used since that is the result type of sizeof expression
// and is preferred for counting in arrays etc.
template<typename TYPE, size_t SIZE>
class StaticVector : public IVect<StaticVector<TYPE, SIZE>, TYPE>
{
protected:
  // Array declaration and INITIALIZATION
  TYPE _array[SIZE] = {};
public:
  StaticVector() = default;
  explicit StaticVector (const DynamicVector<TYPE>&); // Conversion ctor
  StaticVector& addMe (const StaticVector&);
  StaticVector add (const StaticVector&) const;
  StaticVector& subMe (const StaticVector&);
  StaticVector sub (const StaticVector&) const;
  StaticVector& mulMe (const TYPE&);
  StaticVector& minus ();
  const TYPE get (std::ptrdiff_t) const;
  TYPE& get(std::ptrdiff_t);
  void print (std::ostream&) const;
  void extract(std::istream&);
  ~StaticVector () = default;
};

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE>::StaticVector(const DynamicVector<TYPE>& dv) {
  for (size_t i = 0; i < SIZE; ++i) {
    if (dv.getSize() > i) {
      _array[i] = dv[i];
    }
  }
}


template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::addMe(const StaticVector
   <TYPE, SIZE> &other)
{
  for (size_t i = 0; i < SIZE; i++) _array[i] += other[i];
  return *this;
```

```
}

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE> StaticVector<TYPE, SIZE>::add(const StaticVector&
    v2) const
{
  StaticVector<TYPE,SIZE> result = *this;
  result += v2;
  return result;
}

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::subMe(const StaticVector
    <TYPE, SIZE> &other)
{
  for (size_t i = 0; i < SIZE; i++) _array[i] -= other[i];
  return *this;
}

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE> StaticVector<TYPE, SIZE>::sub(const StaticVector&
    v2) const
{
  StaticVector<TYPE,SIZE> result = *this;
  result.subMe(v2);
  return result;
}

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::mulMe(const TYPE& elem)
{
  for (size_t i = 0; i < SIZE; i++) _array[i] *= elem;
  return *this;
}

template<typename TYPE, size_t SIZE>
StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::minus()
{
  for (size_t i = 0; i < SIZE; i++) _array[i] = -(_array[i]);
  return *this;
}


template <typename TYPE, std::size_t SIZE>
const TYPE StaticVector<TYPE, SIZE>::get(std::ptrdiff_t i) const
{
  if (std::size_t(i) >= SIZE)
    throw std::out_of_range("Vector index out of range");
  return _array[i];
}
```

```
96
98  template <typename TYPE, std::size_t SIZE>
    TYPE& StaticVector<TYPE, SIZE>::get (std::ptrdiff_t i)
    {
100   if (std::size_t(i) >= SIZE)
        throw std::out_of_range("Vector index out of range");
102   return _array[i];
    }
104
    template<typename TYPE, size_t SIZE>
106 void StaticVector<TYPE, SIZE>::print(std::ostream& os) const
    {
108   os << "Vector :" << "\n[ ";
      for (std::size_t i = 0; i < SIZE; ++i)
110     os << _array[i] << " " ;
      os << "]\n";
112 }

114 template<typename TYPE, size_t SIZE>
    void StaticVector<TYPE, SIZE>::extract(std::istream& is)
116 {
      // Simple solution... Component by component
118   for (size_t i = 0; i < SIZE; ++i) is >> _array[i];
    }
120
    #endif /* STATICVECT_H */
```

StaticVect.hpp

```cpp
#ifndef DYNAMICVECT_H
#define DYNAMICVECT_H

#include "IVect.hpp"

// Forward declaration
template<typename TYPE, size_t SIZE>
class StaticVector;


template <typename TYPE>
class DynamicVector : public IVect<DynamicVector<TYPE>, TYPE> {
 protected:
  std::size_t _size=0;
  TYPE* _val;
public:
  explicit DynamicVector (size_t size = 0): _size(size), _val(new TYPE[size
   ]()) {} // 0-initialized with '()'
  DynamicVector (size_t, const TYPE&); // TYPE& elem - initilized
  DynamicVector (const DynamicVector&);
  DynamicVector (const TYPE&);
  template<size_t S>  // Conversion from all sorts of static vectors.
  explicit DynamicVector (const StaticVector<TYPE, S>&);
  DynamicVector (DynamicVector&&);
  std::size_t getSize () const {return _size;}
  DynamicVector& addMe(const DynamicVector&);
  DynamicVector add (const DynamicVector&) const;
  DynamicVector& subMe(const DynamicVector&);
  DynamicVector sub (const DynamicVector&) const;
  DynamicVector& mulMe (const TYPE&);
  DynamicVector& minus();
  const TYPE& get(std::ptrdiff_t) const;
  TYPE& get(std::ptrdiff_t);
  virtual void print(std::ostream&) const;
  void extract(std::istream&);
  DynamicVector& operator=(const DynamicVector&);
  DynamicVector& operator=(DynamicVector&&);
  ~DynamicVector () {delete[] _val;}

};

// Ctors

template <typename TYPE>
DynamicVector<TYPE>::DynamicVector (std::size_t size, const TYPE& elem):
  _size(size), _val(new TYPE[size]) {
  for (std::size_t i = 0; i < size; ++i) _val[i] = elem;
}
```

```
49 template <typename TYPE>
   DynamicVector<TYPE>::DynamicVector (const DynamicVector& v): _size(v._size),
       _val(new TYPE[v._size]) {
51   for (std::size_t i = 0; i < v._size; ++i) _val[i] = v._val[i];
   }
53
   template <typename TYPE>
55 DynamicVector<TYPE>::DynamicVector (DynamicVector&& v): _size(v._size), _val
      (v._val) {
     v._size = 0; v._val = nullptr;
57 }

59 template<typename TYPE>
   template<size_t S>
61 DynamicVector<TYPE>::DynamicVector(const StaticVector<TYPE, S>& sv) : _size(
      S), _val(new TYPE[S])
   {
63   for (std::size_t i = 0; i < S; ++i) _val[i] = sv[i];
   }
65
   template<typename TYPE>
67 DynamicVector<TYPE>::DynamicVector(const TYPE& elem) : _size(1), _val(new
      TYPE[1])
   {
69   _val[0] = elem;
   }
71

73 // Operations

75 template<typename TYPE>
   DynamicVector<TYPE>& DynamicVector<TYPE>::addMe(const DynamicVector<TYPE> &
      other)
77 {
     size_t end = other._size;
79   if (_size < end) {
       (*this)[end - 1];           // adjusting size
81   }
     for (size_t i = 0; i < end; ++i) {
83     _val[i] += other[i];
     }
85   return *this;
   }
87
   template<typename TYPE>
89 DynamicVector<TYPE> DynamicVector<TYPE>::add(const DynamicVector<TYPE> &
      other) const
   {
91   DynamicVector<TYPE> result = *this;
     result += other;
```

```
93    return result;
}
95
template<typename TYPE>
97  DynamicVector<TYPE>& DynamicVector<TYPE>::mulMe(const TYPE& elem)
{
99    for (size_t i = 0; i < _size; i++) _val[i] *= elem;
    return *this;
101 }

103 template<typename TYPE>
    DynamicVector<TYPE>& DynamicVector<TYPE>::subMe(const DynamicVector<TYPE> &
       other)
105 {
    size_t end = other._size;
107   if (_size < end) {
      (*this)[end - 1];           // adjusting size
109   }
    for (size_t i = 0; i < end; ++i) _val[i] -= other[i];
111   return *this;
}
113
template<typename TYPE>
115 DynamicVector<TYPE> DynamicVector<TYPE>::sub(const DynamicVector<TYPE> &
       other) const
{
117   DynamicVector<TYPE> result = *this;
    result -= other;
119   return result;
}
121

123 template<typename TYPE>
    DynamicVector<TYPE>& DynamicVector<TYPE>::minus()
125 {
    for (size_t i = 0; i < _size; ++i) _val[i] = -(_val[i]);
127   return *this;
}
129

131 template <typename TYPE>
    const TYPE& DynamicVector<TYPE>::get(std::ptrdiff_t i) const {
133   if (std::size_t(i) >= _size)
      throw std::out_of_range("DynamicVector : Index out of range");
135   return _val[i];
}
137
template <typename TYPE>
139 TYPE& DynamicVector<TYPE>::get(std::ptrdiff_t i) {
    if (std::size_t(i) >= _size) {
```

```
141     DynamicVector<TYPE> tempVect(i + 1, 0);
        for (std::size_t i = 0; i < _size; ++i) tempVect[i] = _val[i];
143     *this = tempVect;
        return _val[i];
145   }

147   return _val[i];
}
149

151 // Assignment

153 template <typename TYPE>
DynamicVector<TYPE>& DynamicVector<TYPE>::operator=(const DynamicVector& v)
    {
155   if (this != &v) {
        delete[] _val; _size = v._size; _val = new TYPE[v._size];
157     for (std::size_t i = 0; i < v._size; ++i) _val[i] = v[i];
    }
159   return *this;
}
161
163 template <typename TYPE>
DynamicVector<TYPE>& DynamicVector<TYPE>::operator=(DynamicVector&& v) {
165   if (this != &v) {
        delete[] _val; _size = v._size; _val = v._val;
167     v._size = 0; v._val = nullptr;
    }
169   return *this;
}

171 // IO

173 template<typename TYPE>
void DynamicVector<TYPE>::print(std::ostream& os) const {
175   os << "Vector :" << "\n[ ";
    for (std::size_t i = 0; i < _size; ++i)
177     os << _val[i] << ' ';
    os << "]\n";
179

}
181
183 template<typename TYPE>
void DynamicVector<TYPE>::extract(std::istream& is)
{
185   // Simple solution... Component by component until Ctrl + D or non
    // TYPE.
187   size_t cnt = 0;
    TYPE coef;
189   while (std::cin >> coef) {
```

```
    (*this)[cnt] = coef;
191     ++cnt;
  }
193 }

195 #endif /* DYNAMICVECT_H */
```

DynamicVect.hpp

```hpp
#ifndef IPOLY_H
#define IPOLY_H

#include <iostream>
#include <cstdlib>

#include "IVect.hpp"

template<typename PolyType, typename TYPE>
class IPoly
{
protected:
  IPoly() = default;
  virtual ~IPoly() = default;
private:
  const PolyType& child() const {return *static_cast<const PolyType*>(this)
   ;}
  PolyType& child() {return *static_cast<PolyType*>(this);}
public:
  TYPE operator() (const TYPE& elem) {
    return child().horner(elem);}
  friend PolyType operator*(const PolyType& p1, const PolyType& p2) {
    return p1.mulPol(p2);}

};

#endif /* IPOLY_H */
```

IPoly.hpp

```cpp
#ifndef POLY_H
#define POLY_H

#include <iostream>
#include <cstdlib>

#include "IPoly.hpp"
#include "DynamicVect.hpp"

template<typename TYPE>
class Poly :
  public DynamicVector<TYPE>,
  public IPoly<Poly<TYPE>, TYPE>
{
public:
  using DynamicVector<TYPE>::DynamicVector;
  int deg() const {return static_cast<int>(this -> getSize() - 1);} //
   getter
  virtual void print(std::ostream& os) const override;
  TYPE horner(const TYPE&) const; // operator(), evaluation
  Poly mulPol(const Poly&) const; // polynomial multiplication
};

template<typename TYPE>
void Poly<TYPE>::print(std::ostream& os) const {
  os << "Polynomial :\n";
  for (std::size_t i = this -> _size; i --> 0;) {
    if ((*this)[i]) {
      if ((*this)[i] >= 0) os << " +";
      os << (*this)[i];
      if (i) os << "x^" << i;
      os << " ";
    }
  }
  os << "\n";
}

template<typename TYPE>
TYPE Poly<TYPE>::horner(const TYPE& elem) const
{
  TYPE res = (*this)[deg()];
  for (int i = deg() - 1; i >= 0; i--) {
    res = (res * elem) + (*this)[i];
  }
  return res;
}

template<typename TYPE>
Poly<TYPE> Poly<TYPE>::mulPol(const Poly& p2) const
```

```
{
50   Poly<TYPE> p3;
   if (deg() < 0 || p2.deg() < 0) {;} // NOP
52   else {
     p3[deg() + p2.deg()];                // adjust size
54     for (size_t j = 0; j <= static_cast<size_t>(p2.deg()); ++j)
       p3[j] = (*this)[0] * p2[j];
56     for (size_t i = 1; i <= static_cast<size_t>(deg()); ++i) {
       for (size_t j = 0; j < static_cast<size_t>(p2.deg()); ++j)
58       p3[i+j] += (*this)[i] * p2[j];
       p3[i + p2.deg()] = (*this)[i] * p2[p2.deg()];
60     }
   }
62   return p3;
}
64


66
#endif /* POLY_H */
```

Poly.hpp

```cpp
#ifndef POLYMOD_H
#define POLYMOD_H


#include <iostream>
#include <cstdlib>

#include "Poly.hpp"
#include "Div.hpp"

template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
class PolyMod : public Poly<TYPE>
{
public:
  using Poly<TYPE>::Poly;
  int deg() const {return static_cast<int>(this -> getSize() - 1);} // getter
  void print(std::ostream&) const override;
  PolyMod mulPol(const PolyMod&) const override; // polynomial multiplication
};

template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
void PolyMod<TYPE, DEG, div>::print(std::ostream& os) const {
  os << "Polynomial :\n";
  for (std::size_t i = DEG + 1; i --> 0;) {
    if ((*this)[i]) {
      if ((*this)[i] >= 0) os << " +";
      os << (*this)[i];
      if (i) os << "x^" << i;
      os << " ";
    }
  }

  os << "\n";
  os << "Mod ";
  os << div;
}

template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
PolyMod<TYPE, DEG, div> PolyMod<TYPE, DEG, div>::mulPol(const PolyMod& p2) const
{
  PolyMod<TYPE, DEG, div> p3;
  int new_degree = deg() < 0 || p2.deg() < 0 ? -1 : deg() + p2.deg();
  p3[new_degree];                // adjust size
  for (int i = 0; i < DEG && i <= new_degree; ++i) {
    int ci = 0;
    for (int j = (p2.deg() < i ? i - p2.deg() : 0); j <= i && j <= deg(); ++
```

```
   j)
47       ci += (*this)[j] * p2[i - j];
     p3[i] = ci;
49   }
   for (int i = DEG; i <= new_degree; ++i) {
51     int ci = 0, m = 0;
     for (int j = i - p2.deg(); j <= deg(); ++j) ci += (*this)[j] * p2[i - j
    ];
53     for (int j = i - DEG; j < DEG; ++j) p3[j] += ci*div[m++];
     for (int j = 0; j < i - DEG; ++j) p3[j] += ci*div[m++];
55   }
   if (new_degree < DEG)
57     for (int i = new_degree + 1; i < DEG; ++i) p3[i] = 0;
   else
59     for (new_degree = DEG - 1; new_degree >= 0 && p3[new_degree] == 0; --
    new_degree);
   p3[new_degree];                // adjust
61 }

63 #endif /* POLYMOD_H */
```

PolyMod.hpp

```
Vector :
[ 0 3 2 0 0 ]

Vector :
[ 3 3 3 3 3 ]

Vector :
[ 2 2 2 2 2 ]

Vector :
[ 0 -6.4 -6.4 11.2 -6.4 ]

Polynomial :
-2x^4  +1x^3  +3x^2  +5

Polynomial :
-1x^2  +2x^1  +3

Polynomial :
 +2x^6 -5x^5 -7x^4  +9x^3 -1x^2  +20x^1  +30

6
```

execution.txt