

```
1  /**
   * Projet C++      : Polynômes modulaires
3  * Matricule       : 410031
   * Nom             : Requena López
5  * Prénom          : Carlos
   * Année           : 2015-2016
7  */

9  #include <iostream>

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

15

17
   int main() {

19

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

25

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

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

31

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

35

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

37

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

39

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

43

       Poly<double> poly2(size_t(3));

45     poly2[0] = 3; poly2[1] = 2; poly2[2] = -1;
       std::cout << poly2 << "\n";      // -1x^2 + 2x^1 + 3

47

       poly1 = poly1 * poly2;

49     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.
57 // PolyMod<int, static_vector1, 4> polymod1;
59 return 0;
}
```

main.cpp

```

1  #ifndef IVECT_H
2  #define IVECT_H
3
4
5
6  #include <cstdlib>
7  #include <iostream>
8
9  // Interface for vectors. All static
10 template<typename VectType, typename TYPE>
11 class IVect
12 {
13 protected:
14     IVect() = default;
15     ~IVect() = default;
16 private:
17     const VectType& child() const {return *static_cast<const VectType*>(this)
18     ;}
19     VectType& child() {return *static_cast<VectType*>(this);}
20 public:
21     friend std::ostream& operator<<(std::ostream& os, const VectType& v) {
22         v.print(os); return os;}
23     friend std::istream& operator>>(std::istream& is, VectType& v) {
24         v.extract(is); return is;}
25     friend VectType operator+(const VectType& v1, const VectType& v2) {
26         return v1.add(v2);}
27     friend VectType operator-(const VectType& v1, const VectType& v2) {
28         return v1.sub(v2);}
29     const TYPE operator[] (std::ptrdiff_t i) const {
30         return child().get(i);}
31     TYPE& operator[] (std::ptrdiff_t i) {
32         return child().get(i);}
33     void operator+=(const VectType &other) {
34         child().addMe(other);}
35     void operator*=(const TYPE &elem) {
36         child().mulMe(elem);}
37     void operator-=(const VectType &other) {
38         child().subMe(other);}
39     void operator-() {
40         child().minus();}
41 };
42 #endif /* IVECT_H */

```

IVect.hpp

```
1 #ifndef STATICVECT_H
2 #define STATICVECT_H
3
4 #include "IVect.hpp"
5
6 // Forward declaration
7 template <typename TYPE>
8 class DynamicVector;
9
10 // size_t is used since that is the result type of sizeof expression
11 // and is preferred for counting in arrays etc.
12 template<typename TYPE, size_t SIZE>
13 class StaticVector : public IVect<StaticVector<TYPE, SIZE>, TYPE>
14 {
15 protected:
16     // Array declaration and INITIALIZATION
17     TYPE _array[SIZE] = {};
18 public:
19     StaticVector() = default;
20     explicit StaticVector (const DynamicVector<TYPE>&); // Conversion ctor
21     StaticVector& addMe (const StaticVector&);
22     StaticVector add (const StaticVector&) const;
23     StaticVector& subMe (const StaticVector&);
24     StaticVector sub (const StaticVector&) const;
25     StaticVector& mulMe (const TYPE&);
26     StaticVector& minus ();
27     const TYPE get (std::ptrdiff_t) const;
28     TYPE& get(std::ptrdiff_t);
29     void print (std::ostream&) const;
30     void extract(std::istream&);
31     ~StaticVector () = default;
32 };
33
34 template<typename TYPE, size_t SIZE>
35 StaticVector<TYPE, SIZE>::StaticVector(const DynamicVector<TYPE>& dv) {
36     for (size_t i = 0; i < SIZE; ++i) {
37         if (dv.getSize() > i) {
38             _array[i] = dv[i];
39         }
40     }
41 }
42
43
44 template<typename TYPE, size_t SIZE>
45 StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::addMe(const StaticVector
46 <TYPE, SIZE> &other)
47 {
48     for (size_t i = 0; i < SIZE; i++) _array[i] += other[i];
49     return *this;
50 }
```

```
50 }
51
52 template<typename TYPE, size_t SIZE>
53 StaticVector<TYPE, SIZE> StaticVector<TYPE, SIZE>::add(const StaticVector&
54     v2) const
55 {
56     StaticVector<TYPE, SIZE> result = *this;
57     result += v2;
58     return result;
59 }
60
61 template<typename TYPE, size_t SIZE>
62 StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::subMe(const StaticVector
63     <TYPE, SIZE> &other)
64 {
65     for (size_t i = 0; i < SIZE; i++) _array[i] -= other[i];
66     return *this;
67 }
68
69 template<typename TYPE, size_t SIZE>
70 StaticVector<TYPE, SIZE> StaticVector<TYPE, SIZE>::sub(const StaticVector&
71     v2) const
72 {
73     StaticVector<TYPE, SIZE> result = *this;
74     result.subMe(v2);
75     return result;
76 }
77
78 template<typename TYPE, size_t SIZE>
79 StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::mulMe(const TYPE& elem)
80 {
81     for (size_t i = 0; i < SIZE; i++) _array[i] *= elem;
82     return *this;
83 }
84
85 template<typename TYPE, size_t SIZE>
86 StaticVector<TYPE, SIZE>& StaticVector<TYPE, SIZE>::minus()
87 {
88     for (size_t i = 0; i < SIZE; i++) _array[i] = -(_array[i]);
89     return *this;
90 }
91
92 template <typename TYPE, std::size_t SIZE>
93 const TYPE StaticVector<TYPE, SIZE>::get(std::ptrdiff_t i) const
94 {
95     if (std::size_t(i) >= SIZE)
96         throw std::out_of_range("Vector index out of range");
97     return _array[i];
98 }
```

```
96 template <typename TYPE, std::size_t SIZE>
97 TYPE& StaticVector<TYPE, SIZE>::get (std::ptrdiff_t i)
98 {
99     if (std::size_t(i) >= SIZE)
100         throw std::out_of_range("Vector index out of range");
101     return _array[i];
102 }
103
104 template<typename TYPE, size_t SIZE>
105 void StaticVector<TYPE, SIZE>::print(std::ostream& os) const
106 {
107     os << "Vector :" << "\n[ ";
108     for (std::size_t i = 0; i < SIZE; ++i)
109         os << _array[i] << " ";
110     os << "]\n";
111 }
112
113 template<typename TYPE, size_t SIZE>
114 void StaticVector<TYPE, SIZE>::extract(std::istream& is)
115 {
116     // Simple solution... Component by component
117     for (size_t i = 0; i < SIZE; ++i) is >> _array[i];
118 }
119
120 #endif /* STATICVECT_H */
```

StaticVect.hpp

```

1  #ifndef DYNAMICVECT_H
2  #define DYNAMICVECT_H
3
4  #include "IVect.hpp"
5
6  // Forward declaration
7  template<typename TYPE, size_t SIZE>
8  class StaticVector;
9
10
11 template <typename TYPE>
12 class DynamicVector : public IVect<DynamicVector<TYPE>, TYPE> {
13     protected:
14         std::size_t _size=0;
15         TYPE* _val;
16     public:
17         explicit DynamicVector (size_t size = 0): _size(size), _val(new TYPE[size]
18             )() {} // 0-initialized with '()'
19         DynamicVector (size_t, const TYPE&); // TYPE& elem - initilized
20         DynamicVector (const DynamicVector&);
21         DynamicVector (const TYPE&);
22         template<size_t S> // Conversion from all sorts of static vectors.
23         explicit DynamicVector (const StaticVector<TYPE, S>&);
24         DynamicVector (DynamicVector&&);
25         std::size_t getSize () const {return _size;}
26         DynamicVector& addMe(const DynamicVector&);
27         DynamicVector add (const DynamicVector&) const;
28         DynamicVector& subMe(const DynamicVector&);
29         DynamicVector sub (const DynamicVector&) const;
30         DynamicVector& mulMe (const TYPE&);
31         DynamicVector& minus();
32         const TYPE& get(std::ptrdiff_t) const;
33         TYPE& get(std::ptrdiff_t);
34         virtual void print(std::ostream&) const;
35         void extract(std::istream&);
36         DynamicVector& operator=(const DynamicVector&);
37         DynamicVector& operator=(DynamicVector&&);
38         ~DynamicVector () {delete[] _val;}
39 };
40
41 // Ctors
42
43 template <typename TYPE>
44 DynamicVector<TYPE>::DynamicVector (std::size_t size, const TYPE& elem):
45     _size(size), _val(new TYPE[size]) {
46     for (std::size_t i = 0; i < size; ++i) _val[i] = elem;
47 }

```

```
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
54 template <typename TYPE>
55 DynamicVector<TYPE>::DynamicVector (DynamicVector&& v): _size(v._size), _val
    (v._val) {
    v._size = 0; v._val = nullptr;
57 }
58
59 template<typename TYPE>
60 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
66 template<typename TYPE>
67 DynamicVector<TYPE>::DynamicVector(const TYPE& elem) : _size(1), _val(new
    TYPE[1])
    {
69     _val[0] = elem;
    }
71
72 // Operations
73
74 template<typename TYPE>
75 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
88 template<typename TYPE>
89 DynamicVector<TYPE> DynamicVector<TYPE>::add(const DynamicVector<TYPE> &
    other) const
    {
91     DynamicVector<TYPE> result = *this;
    result += other;
```



```
93     return result;
94 }
95
96 template<typename TYPE>
97 DynamicVector<TYPE>& DynamicVector<TYPE>::mulMe(const TYPE& elem)
98 {
99     for (size_t i = 0; i < _size; i++) _val[i] *= elem;
100     return *this;
101 }
102
103 template<typename TYPE>
104 DynamicVector<TYPE>& DynamicVector<TYPE>::subMe(const DynamicVector<TYPE> &
105     other)
106 {
107     size_t end = other._size;
108     if (_size < end) {
109         (*this)[end - 1]; // adjusting size
110     }
111     for (size_t i = 0; i < end; ++i) _val[i] -= other[i];
112     return *this;
113 }
114
115 template<typename TYPE>
116 DynamicVector<TYPE> DynamicVector<TYPE>::sub(const DynamicVector<TYPE> &
117     other) const
118 {
119     DynamicVector<TYPE> result = *this;
120     result -= other;
121     return result;
122 }
123
124 template<typename TYPE>
125 DynamicVector<TYPE>& DynamicVector<TYPE>::minus()
126 {
127     for (size_t i = 0; i < _size; ++i) _val[i] = -(_val[i]);
128     return *this;
129 }
130
131 template <typename TYPE>
132 const TYPE& DynamicVector<TYPE>::get(std::ptrdiff_t i) const {
133     if (std::size_t(i) >= _size)
134         throw std::out_of_range("DynamicVector : Index out of range");
135     return _val[i];
136 }
137
138 template <typename TYPE>
139 TYPE& DynamicVector<TYPE>::get(std::ptrdiff_t i) {
140     if (std::size_t(i) >= _size) {
```

```
141     DynamicVector<TYPE> tempVect(i + 1, 0);
142     for (std::size_t i = 0; i < _size; ++i) tempVect[i] = _val[i];
143     *this = tempVect;
144     return _val[i];
145 }
146
147 return _val[i];
148 }
149
150 // Assignment
151
152 template <typename TYPE>
153 DynamicVector<TYPE>& DynamicVector<TYPE>::operator=(const DynamicVector& v)
154 {
155     if (this != &v) {
156         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];
158     }
159     return *this;
160 }
161
162 template <typename TYPE>
163 DynamicVector<TYPE>& DynamicVector<TYPE>::operator=(DynamicVector&& v) {
164     if (this != &v) {
165         delete[] _val; _size = v._size; _val = v._val;
166         v._size = 0; v._val = nullptr;
167     }
168     return *this;
169 }
170
171 // IO
172
173 template<typename TYPE>
174 void DynamicVector<TYPE>::print(std::ostream& os) const {
175     os << "Vector :" << "\n[ ";
176     for (std::size_t i = 0; i < _size; ++i)
177         os << _val[i] << ' ';
178     os << "]\n";
179 }
180
181 template<typename TYPE>
182 void DynamicVector<TYPE>::extract(std::istream& is)
183 {
184     // Simple solution... Component by component until Ctrl + D or non
185     // TYPE.
186     size_t cnt = 0;
187     TYPE coef;
188     while (std::cin >> coef) {
```

```
191     (*this)[cnt] = coef;  
192     ++cnt;  
193 }  
194  
195 #endif /* DYNAMICVECT_H */
```

DynamicVect.hpp

```
1 #ifndef IPOLY_H
2 #define IPOLY_H
3
4 #include <iostream>
5 #include <cstdlib>
6
7 #include "IVect.hpp"
8
9 template<typename PolyType, typename TYPE>
10 class IPoly
11 {
12     protected:
13         IPoly() = default;
14         virtual ~IPoly() = default;
15     private:
16         const PolyType& child() const {return *static_cast<const PolyType*>(this)
17             ;}
18         PolyType& child() {return *static_cast<PolyType*>(this);}
19     public:
20         TYPE operator() (const TYPE& elem) {
21             return child().horner(elem);}
22         friend PolyType operator*(const PolyType& p1, const PolyType& p2) {
23             return p1.mulPol(p2);}
24 };
25
26 #endif /* IPOLY_H */
```

IPoly.hpp

```

1  #ifndef POLY_H
2  #define POLY_H

4  #include <iostream>
   #include <cstdlib>

6

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

10 template<typename TYPE>
   class Poly :
12     public DynamicVector<TYPE>,
     public IPoly<Poly<TYPE>, TYPE>
14 {
   public:
16     using DynamicVector<TYPE>::DynamicVector;
     int deg() const {return static_cast<int>(this -> getSize() - 1);} //
       getter
18     virtual void print(std::ostream& os) const override;
     TYPE horner(const TYPE&) const; // operator(), evaluation
20     Poly mulPol(const Poly&) const; // polynomial multiplication
   };

22

   template<typename TYPE>
24 void Poly<TYPE>::print(std::ostream& os) const {
     os << "Polynomial :\n";
26     for (std::size_t i = this -> _size; i --> 0;) {
         if ((*this)[i]) {
28             if ((*this)[i] >= 0) os << " +";
             os << (*this)[i];
30             if (i) os << "x^" << i;
             os << " ";
32         }
     }
34     os << "\n";
   }

36

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

46

   template<typename TYPE>
48 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

```

1  #ifndef POLYMOD_H
2  #define POLYMOD_H
3
4
5  #include <iostream>
6  #include <cstdlib>
7
8  #include "Poly.hpp"
9  #include "Div.hpp"
10
11 template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
12 class PolyMod : public Poly<TYPE>
13 {
14 public:
15     using Poly<TYPE>::Poly;
16     int deg() const {return static_cast<int>(this -> getSize() - 1);} //
17     // getter
18     void print(std::ostream&) const override;
19     PolyMod mulPol(const PolyMod&) const override; // polynomial
20     // multiplication
21 };
22
23 template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
24 void PolyMod<TYPE, DEG, div>::print(std::ostream& os) const {
25     os << "Polynomial :\n";
26     for (std::size_t i = DEG + 1; i --> 0;) {
27         if ((*this)[i]) {
28             if ((*this)[i] >= 0) os << " +";
29             os << (*this)[i];
30             if (i) os << "x^" << i;
31             os << " ";
32         }
33     }
34     os << "\n";
35     os << "Mod ";
36     os << div;
37 }
38
39 template<typename TYPE, int DEG, const StaticVector<TYPE, DEG-1>& div>
40 PolyMod<TYPE, DEG, div> PolyMod<TYPE, DEG, div>::mulPol(const PolyMod& p2)
41 const
42 {
43     PolyMod<TYPE, DEG, div> p3;
44     int new_degree = deg() < 0 || p2.deg() < 0 ? -1 : deg() + p2.deg();
45     p3[new_degree]; // adjust size
46     for (int i = 0; i < DEG && i <= new_degree; ++i) {
47         int ci = 0;
48         for (int j = (p2.deg() < i ? i - p2.deg() : 0); j <= i && j <= deg(); ++

```

```
47     j)
        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


```
1 Vector :  
  [ 0 3 2 0 0 ]  
3  
5 Vector :  
  [ 3 3 3 3 3 ]  
7  
9 Vector :  
  [ 2 2 2 2 2 ]  
11  
13 Vector :  
  [ 0 -6.4 -6.4 11.2 -6.4 ]  
15  
17 Polynomial :  
  -2x^4 +1x^3 +3x^2 +5  
19  
21 Polynomial :  
  -1x^2 +2x^1 +3  
23  
25 Polynomial :  
  +2x^6 -5x^5 -7x^4 +9x^3 -1x^2 +20x^1 +30  
27  
29  
31  
33  
35  
37  
39  
41  
43  
45  
47  
49  
51  
53  
55  
57  
59  
61  
63  
65  
67  
69  
71  
73  
75  
77  
79  
81  
83  
85  
87  
89  
91  
93  
95  
97  
99  
101  
103  
105  
107  
109  
111  
113  
115  
117  
119  
121  
123  
125  
127  
129  
131  
133  
135  
137  
139  
141  
143  
145  
147  
149  
151  
153  
155  
157  
159  
161  
163  
165  
167  
169  
171  
173  
175  
177  
179  
181  
183  
185  
187  
189  
191  
193  
195  
197  
199  
201  
203  
205  
207  
209  
211  
213  
215  
217  
219  
221  
223  
225  
227  
229  
231  
233  
235  
237  
239  
241  
243  
245  
247  
249  
251  
253  
255  
257  
259  
261  
263  
265  
267  
269  
271  
273  
275  
277  
279  
281  
283  
285  
287  
289  
291  
293  
295  
297  
299  
301  
303  
305  
307  
309  
311  
313  
315  
317  
319  
321  
323  
325  
327  
329  
331  
333  
335  
337  
339  
341  
343  
345  
347  
349  
351  
353  
355  
357  
359  
361  
363  
365  
367  
369  
371  
373  
375  
377  
379  
381  
383  
385  
387  
389  
391  
393  
395  
397  
399  
401  
403  
405  
407  
409  
411  
413  
415  
417  
419  
421  
423  
425  
427  
429  
431  
433  
435  
437  
439  
441  
443  
445  
447  
449  
451  
453  
455  
457  
459  
461  
463  
465  
467  
469  
471  
473  
475  
477  
479  
481  
483  
485  
487  
489  
491  
493  
495  
497  
499  
501  
503  
505  
507  
509  
511  
513  
515  
517  
519  
521  
523  
525  
527  
529  
531  
533  
535  
537  
539  
541  
543  
545  
547  
549  
551  
553  
555  
557  
559  
561  
563  
565  
567  
569  
571  
573  
575  
577  
579  
581  
583  
585  
587  
589  
591  
593  
595  
597  
599  
601  
603  
605  
607  
609  
611  
613  
615  
617  
619  
621  
623  
625  
627  
629  
631  
633  
635  
637  
639  
641  
643  
645  
647  
649  
651  
653  
655  
657  
659  
661  
663  
665  
667  
669  
671  
673  
675  
677  
679  
681  
683  
685  
687  
689  
691  
693  
695  
697  
699  
701  
703  
705  
707  
709  
711  
713  
715  
717  
719  
721  
723  
725  
727  
729  
731  
733  
735  
737  
739  
741  
743  
745  
747  
749  
751  
753  
755  
757  
759  
761  
763  
765  
767  
769  
771  
773  
775  
777  
779  
781  
783  
785  
787  
789  
791  
793  
795  
797  
799  
801  
803  
805  
807  
809  
811  
813  
815  
817  
819  
821  
823  
825  
827  
829  
831  
833  
835  
837  
839  
841  
843  
845  
847  
849  
851  
853  
855  
857  
859  
861  
863  
865  
867  
869  
871  
873  
875  
877  
879  
881  
883  
885  
887  
889  
891  
893  
895  
897  
899  
901  
903  
905  
907  
909  
911  
913  
915  
917  
919  
921  
923  
925  
927  
929  
931  
933  
935  
937  
939  
941  
943  
945  
947  
949  
951  
953  
955  
957  
959  
961  
963  
965  
967  
969  
971  
973  
975  
977  
979  
981  
983  
985  
987  
989  
991  
993  
995  
997  
999  
1001  
1003  
1005  
1007  
1009  
1011  
1013  
1015  
1017  
1019  
1021  
1023  
1025  
1027  
1029  
1031  
1033  
1035  
1037  
1039  
1041  
1043  
1045  
1047  
1049  
1051  
1053  
1055  
1057  
1059  
1061  
1063  
1065  
1067  
1069  
1071  
1073  
1075  
1077  
1079  
1081  
1083  
1085  
1087  
1089  
1091  
1093  
1095  
1097  
1099  
1101  
1103  
1105  
1107  
1109  
1111  
1113  
1115  
1117  
1119  
1121  
1123  
1125  
1127  
1129  
1131  
1133  
1135  
1137  
1139  
1141  
1143  
1145  
1147  
1149  
1151  
1153  
1155  
1157  
1159  
1161  
1163  
1165  
1167  
1169  
1171  
1173  
1175  
1177  
1179  
1181  
1183  
1185  
1187  
1189  
1191  
1193  
1195  
1197  
1199  
1201  
1203  
1205  
1207  
1209  
1211  
1213  
1215  
1217  
1219  
1221  
1223  
1225  
1227  
1229  
1231  
1233  
1235  
1237  
1239  
1241  
1243  
1245  
1247  
1249  
1251  
1253  
1255  
1257  
1259  
1261  
1263  
1265  
1267  
1269  
1271  
1273  
1275  
1277  
1279  
1281  
1283  
1285  
1287  
1289  
1291  
1293  
1295  
1297  
1299  
1301  
1303  
1305  
1307  
1309  
1311  
1313  
1315  
1317  
1319  
1321  
1323  
1325  
1327  
1329  
1331  
1333  
1335  
1337  
1339  
1341  
1343  
1345  
1347  
1349  
1351  
1353  
1355  
1357  
1359  
1361  
1363  
1365  
1367  
1369  
1371  
1373  
1375  
1377  
1379  
1381  
1383  
1385  
1387  
1389  
1391  
1393  
1395  
1397  
1399  
1401  
1403  
1405  
1407  
1409  
1411  
1413  
1415  
1417  
1419  
1421  
1423  
1425  
1427  
1429  
1431  
1433  
1435  
1437  
1439  
1441  
1443  
1445  
1447  
1449  
1451  
1453  
1455  
1457  
1459  
1461  
1463  
1465  
1467  
1469  
1471  
1473  
1475  
1477  
1479  
1481  
1483  
1485  
1487  
1489  
1491  
1493  
1495  
1497  
1499  
1501  
1503  
1505  
1507  
1509  
1511  
1513  
1515  
1517  
1519  
1521  
1523  
1525  
1527  
1529  
1531  
1533  
1535  
1537  
1539  
1541  
1543  
1545  
1547  
1549  
1551  
1553  
1555  
1557  
1559  
1561  
1563  
1565  
1567  
1569  
1571  
1573  
1575  
1577  
1579  
1581  
1583  
1585  
1587  
1589  
1591  
1593  
1595  
1597  
1599  
1601  
1603  
1605  
1607  
1609  
1611  
1613  
1615  
1617  
1619  
1621  
1623  
1625  
1627  
1629  
1631  
1633  
1635  
1637  
1639  
1641  
1643  
1645  
1647  
1649  
1651  
1653  
1655  
1657  
1659  
1661  
1663  
1665  
1667  
1669  
1671  
1673  
1675  
1677  
1679  
1681  
1683  
1685  
1687  
1689  
1691  
1693  
1695  
1697  
1699  
1701  
1703  
1705  
1707  
1709  
1711  
1713  
1715  
1717  
1719  
1721  
1723  
1725  
1727  
1729  
1731  
1733  
1735  
1737  
1739  
1741  
1743  
1745  
1747  
1749  
1751  
1753  
1755  
1757  
1759  
1761  
1763  
1765  
1767  
1769  
1771  
1773  
1775  
1777  
1779  
1781  
1783  
1785  
1787  
1789  
1791  
1793  
1795  
1797  
1799  
1801  
1803  
1805  
1807  
1809  
1811  
1813  
1815  
1817  
1819  
1821  
1823  
1825  
1827  
1829  
1831  
1833  
1835  
1837  
1839  
1841  
1843  
1845  
1847  
1849  
1851  
1853  
1855  
1857  
1859  
1861  
1863  
1865  
1867  
1869  
1871  
1873  
1875  
1877  
1879  
1881  
1883  
1885  
1887  
1889  
1891  
1893  
1895  
1897  
1899  
1901  
1903  
1905  
1907  
1909  
1911  
1913  
1915  
1917  
1919  
1921  
1923  
1925  
1927  
1929  
1931  
1933  
1935  
1937  
1939  
1941  
1943  
1945  
1947  
1949  
1951  
1953  
1955  
1957  
1959  
1961  
1963  
1965  
1967  
1969  
1971  
1973  
1975  
1977  
1979  
1981  
1983  
1985  
1987  
1989  
1991  
1993  
1995  
1997  
1999  
2001  
2003  
2005  
2007  
2009  
2011  
2013  
2015  
2017  
2019  
2021  
2023  
2025  
2027  
2029  
2031  
2033  
2035  
2037  
2039  
2041  
2043  
2045  
2047  
2049  
2051  
2053  
2055  
2057  
2059  
2061  
2063  
2065  
2067  
2069  
2071  
2073  
2075  
2077  
2079  
2081  
2083  
2085  
2087  
2089  
2091  
2093  
2095  
2097  
2099  
2101  
2103  
2105  
2107  
2109  
2111  
2113  
2115  
2117  
2119  
2121  
2123  
2125  
2127  
2129  
2131  
2133  
2135  
2137  
2139  
2141  
2143  
2145  
2147  
2149  
2151  
2153  
2155  
2157  
2159  
2161  
2163  
2165  
2167  
2169  
2171  
2173  
2175  
2177  
2179  
2181  
2183  
2185  
2187  
2189  
2191  
2193  
2195  
2197  
2199  
2201  
2203  
2205  
2207  
2209  
2211  
2213  
2215  
2217  
2219  
2221  
2223  
2225  
2227  
2229  
2231  
2233  
2235  
2237  
2239  
2241  
2243  
2245  
2247  
2249  
2251  
2253  
2255  
2257  
2259  
2261  
2263  
2265  
2267  
2269  
2271  
2273  
2275  
2277  
2279  
2281  
2283  
2285  
2287  
2289  
2291  
2293  
2295  
2297  
2299  
2301  
2303  
2305  
2307  
2309  
2311  
2313  
2315  
2317  
2319  
2321  
2323  
2325  
2327  
2329  
2331  
2333  
2335  
2337  
2339  
2341  
2343  
2345  
2347  
2349  
2351  
2353  
2355  
2357  
2359  
2361  
2363  
2365  
2367  
2369  
2371  
2373  
2375  
2377  
2379  
2381  
2383  
2385  
2387  
2389  
2391  
2393  
2395  
2397  
2399  
2401  
2403  
2405  
2407  
2409  
2411  
2413  
2415  
2417  
2419  
2421  
2423  
2425  
2427  
2429  
2431  
2433  
2435  
2437  
2439  
2441  
2443  
2445  
2447  
2449  
2451  
2453  
2455  
2457  
2459  
2461  
2463  
2465  
2467  
2469  
2471  
2473  
2475  
2477  
2479  
2481  
2483  
2485  
2487  
2489  
2491  
2493  
2495  
2497  
2499  
2501  
2503  
2505  
2507  
2509  
2511  
2513  
2515  
2517  
2519  
2521  
2523  
2525  
2527  
2529  
2531  
2533  
2535  
2537  
2539  
2541  
2543  
2545  
2547  
2549  
2551  
2553  
2555  
2557  
2559  
2561  
2563  
2565  
2567  
2569  
2571  
2573  
2575  
2577  
2579  
2581  
2583  
2585  
2587  
2589  
2591  
2593  
2595  
2597  
2599  
2601  
2603  
2605  
2607  
2609  
2611  
2613  
2615  
2617  
2619  
2621  
2623  
2625  
2627  
2629  
2631  
2633  
2635  
2637  
2639  
2641  
2643  
2645  
2647  
2649  
2651  
2653  
2655  
2657  
2659  
2661  
2663  
2665  
2667  
2669  
2671  
2673  
2675  
2677  
2679  
2681  
2683  
2685  
2687  
2689  
2691  
2693  
2695  
2697  
2699  
2701  
2703  
2705  
2707  
2709  
2711  
2713  
2715  
2717  
2719  
2721  
2723  
2725  
2727  
2729  
2731  
2733  
2735  
2737  
2739  
2741  
2743  
2745  
2747  
2749  
2751  
2753  
2755  
2757  
2759  
2761  
2763  
2765  
2767  
2769  
2771  
2773  
2775  
2777  
2779  
2781  
2783  
2785  
2787  
2789  
2791  
2793  
2795  
2797  
2799  
2801  
2803  
2805  
2807  
2809  
2811  
2813  
2815  
2817  
2819  
2821  
2823  
2825  
2827  
2829  
2831  
2833  
2835  
2837  
2839  
2841  
2843  
2845  
2847  
2849  
2851  
2853  
2855  
2857  
2859  
2861  
2863  
2865  
2867  
2869  
2871  
2873  
2875  
2877  
2879  
2881  
2883  
2885  
2887  
2889  
2891  
2893  
2895  
2897  
2899  
2901  
2903  
2905  
2907  
2909  
2911  
2913  
2915  
2917  
2919  
2921  
2923  
2925  
2927  
2929  
2931  
2933  
2935  
2937  
2939  
2941  
2943  
2945  
2947  
2949  
2951  
2953  
2955  
2957  
2959  
2961  
2963  
2965  
2967  
2969  
2971  
2973  
2975  
2977  
2979  
2981  
2983  
2985  
2987  
2989  
2991  
2993  
2995  
2997  
2999  
3001  
3003  
3005  
3007  
3009  
3011  
3013  
3015  
3017  
3019  
3021  
3023  
3025  
3027  
3029  
3031  
3033  
3035  
3037  
3039  
3041  
3043  
3045  
3047  
3049  
3051  
3053  
3055  
3057  
3059  
3061  
3063  
3065  
3067  
3069  
3071  
3073  
3075  
3077  
3079  
3081  
3083  
3085  
3087  
3089  
3091  
3093  
3095  
3097  
3099  
3101  
3103  
3105  
3107  
3109  
3111  
3113  
3115  
3117  
3119  
3121  
3123  
3125  
3127  
3129  
3131  
3133  
3135  
3137  
3139  
3141  
3143  
3145  
3147  
3149  
3151  
3153  
3155  
3157  
3159  
3161  
3163  
3165  
3167  
3169  
3171  
3173  
3175  
3177  
3179  
3181  
3183  
3185  
3187  
3189  
3191  
3193  
3195  
3197  
3199  
3201  
3203  
3205  
3207  
3209  
3211  
3213  
3215  
3217  
3219  
3221  
3223  
3225  
3227  
3229  
3231  
3233  
3235  
3237  
3239  
3241  
3243  
3245  
3247  
3249  
3251  
3253  
3255  
3257  
3259  
3261  
3263  
3265  
3267  
3269  
3271  
3273  
3275  
3277  
3279  
3281  
3283  
3285  
3287  
3289  
3291  
3293  
3295  
3297  
3299  
3301  
3303  
3305  
3307  
3309  
3311  
3313  
3315  
3317  
3319  
3321  
3323  
3325  
3327  
3329  
3331  
3333  
3335  
3337  
3339  
3341  
3343  
3345  
3347  
3349  
3351  
3353  
3355  
3357  
3359  
3361  
3363  
3365  
3367  
3369  
3371  
3373  
3375  
3377  
3379  
3381  
3383  
3385  
3387  
3389  
3391  
3393  
3395  
3397  
3399  
3401  
3403  
3405  
3407  
3409  
3411  
3413  
3415  
3417  
3419  
3421  
3423  
3425  
3427  
3429  
3431  
3433  
3435  
3437  
3439  
3441  
3443  
3445  
3447  
3449  
3451  
3453  
3455  
3457  
3459  
3461  
3463  
3465  
3467  
3469  
3471  
3473  
3475  
3477  
3479  
3481  
3483  
3485  
3487  
3489  
3491  
3493  
3495  
3497  
3499  
3501  
3503  
3505  
3507  
3509  
3511  
3513  
3515  
3517  
3519  
3521  
3523  
3525  
3527  
3529  
3531  
3533  
3535  
3537  
3539  
3541  
3543  
3545  
3547  
3549  
3551  
3553  
3555  
3557  
3559  
3561  
3563  
3565  
3567  
3569  
3571  
3573  
3575  
3577  
3579  
3581  
3583  
3585  
3587  
3589  
3591  
3593  
3595  
3597  
3599  
3601  
3603  
3605  
3607  
3609  
3611  
3613  
3615  
3617  
3619  
3621  
3623  
3625  
3627  
3629  
3631  
3633  
3635  
3637  
3639  
3641  
3643  
3645  
3647  
3649  
3651  
3653  
3655  
3657  
3659  
3661  
3663  
3665  
3667  
3669  
3671  
3673  
3675  
3677  
3679  
3681  
3683  
3685  
3687  
3689  
3691  
3693  
3695  
3697  
3699  
3701  
3703  
3705  
3707  
3709  
3711  
3713  
3715  
3717  
3719  
3721  
3723  
3725  
3727  
3729  
3731  
3733  
3735  
3737  
3739  
3741  
3743  
3745  
3747  
3749  
3751  
3753  
3755  
3757  
3759  
3761  
3763  
3765  
3767  
3769  
3771  
3773  
3775  
3777  
3779  
3781  
3783  
3785  
3787  
3789  
3791  
3793  
3795  
3797  
3799  
3801  
3803  
3805  
3807  
3809  
3811  
3813  
3815  
3817  
3819  
3821  
3823  
3825  
3827  
3829  
3831  
3833  
3835  
3837  
3839  
3841  
3843  
3845  
3847  
3849  
3851  
3853  
3855  
3857  
3859  
3861  
3863  
3865  
3867  
3869  
3871  
3873  
3875  
3877  
3879  
3881  
3883  
3885  
3887  
3889  
3891  
3893  
3895  
3897  
3899  
3901  
3903  
3905  
3907  
3909  
3911  
3913  
3915  
3917  
3919  
3921  
3923  
3925  
3927  
3929  
3931  
3933  
3935  
3937  
3939  
3941  
3943  
3945  
3947  
3949  
3951  
3953  
3955  
3957  
3959  
3961  
3963  
3965  
3967  
3969  
3971  
3973  
3975  
3977  
3979  
3981  
3983  
3985  
3987  
3989  
3991  
3993  
3995  
3997  
3999  
4001  
4003  
4005  
4007  
4009  
4011  
4013  
4015  
4017  
4019  
4021  
4023  
4025  
4027  
4029  
4031  
4033  
4035  
4037  
4039  
4041  
4043  
4045  
4047  
4049  
4051  
4053  
4055  
4057  
4059  
4061  
4063  
4065  
4067  
4069  
4071  
4073  
4075  
4077  
4079  
4081  
4083  
4085  
4087  
4089  
4091  
4093  
4095  
4097  
4099  
4101  
4103  
4105  
4107  
4109  
4111  
4113  
4115  
4117  
4119  
4121  
4123  
4125  
4127  
4129  
4131  
4133  
4135  
4137  
4139  
4141  
4143  
4145  
4147  
4149  
4151  
4153  
4155  
4157  
4159  
4161  
4163  
4165  
4167  
4169  
4171  
4173  
4175  
4177  
4179  
4181  
4183  
4185  
4187  
4189  
4191  
4193  
4195  
4197  
4199  
4201  
4203  
4205  
4207  
4209  
4211  
4213  
4215  
4217  
4219  
4221
```