

```

1  #include <iostream>
2
3  #include "question1.hpp"
4  #include "question2.hpp"
5  #include "question3.hpp"
6
7  using question1::operator*;
8
9  std::ostream& operator<<(std::ostream& os, boost::numeric::ublas::matrix<int> mat) { // Fonction
    permettant l'impression d'une matrice
10     int rows = mat.size1(),
11         cols = mat.size2();
12
13     for(int row = 0 ; row < rows ; ++row) {
14         for(int col = 0 ; col < cols ; ++col) {
15             if(col == 0) {
16                 if(row == 0) {
17                     os << "[";
18                 } else if(row == rows - 1) {
19                     os << "L";
20                 } else {
21                     os << "|";
22                 }
23             }
24
25             os << mat(row, col) << "\t";
26
27             if(col == cols - 1) {
28                 if(row == 0) {
29                     os << "]" << std::endl;
30                 } else if(row == rows - 1) {
31                     os << "J" << std::endl;
32                 } else {
33                     os << "|" << std::endl;
34                 }
35             }
36         }
37     }
38
39     return os;
40 }
41
42 int main() {
43
44     boost::numeric::ublas::matrix<int> A(3,3);
45     boost::numeric::ublas::matrix<int> B(3,3);
46
47     int rows = A.size1();
48     int cols = A.size2();
49
50     for(int i = 0 ; i < rows ; ++i) {
51         for(int j = 0 ; j < cols ; ++j) {
52             A(i, j) = i + j;
53             B(i, j) = i - j;
54         }
55     }
56
57     std::cout << "A =" << std::endl << A << std::endl;
58     std::cout << "B =" << std::endl << B << std::endl;
59     std::cout << "A*B =" << std::endl << A*B << std::endl;
60     std::cout << static_cast<boost::numeric::ublas::matrix<int>>(boost::numeric::ublas::prod(A,
    B)) << std::endl;
61     std::cout << "A^3 =" << std::endl << question2::pow(A, 3) << std::endl;
62
63     for(int i = 0 ; i <= 30 ; ++i) {
64         std::cout << "F_" << i << " = " << question3::fibonacci(i) << std::endl;
65     }
66
67     return 0;
68 }

```

```
1  #include "question1.hpp"
2
3  using boost::numeric::ublas::matrix;
4
5  namespace question1 {
6
7      matrix<int> operator*(matrix<int> const& A, matrix<int> const& B) {
8          int rowsA = A.size1(), colsA = A.size2(),
9              rowsB = B.size1(), colsB = B.size2();
10
11          if(colsA != rowsB) // Vérification de la taille
12              throw new std::length_error("Matrices must have a the right sizes to be
multiplicated");
13
14          matrix<int> result(rowsA, colsB);
15
16          for(int row = 0 ; row < rowsA ; ++row) { // Traitement des lignes
17              for(int col = 0 ; col < colsB ; ++col) { // Traitement des colonnes
18                  int el = 0;
19                  for(int i = 0 ; i < colsA ; ++i) { // Un élément est la somme des produits des
lignes de A par les colonnes de B
20                      el += A(row, i)*B(i, col);
21                  }
22                  result(row, col) = el;
23              }
24          }
25
26          return result;
27      }
28
29  }
```

```

1  #include "question2.hpp"
2
3  using boost::numeric::ublas::matrix;
4  using boost::numeric::ublas::identity_matrix;
5  using boost::numeric::ublas::prod;
6
7  namespace question2 {
8  /**
9   * Cette fonction permet d'élever une matrice A à la puissance B
10  *
11  * Invariant :  $I(M,R,c)$ 
12  *  $\Rightarrow A^b = M * A^c * R$ 
13  *
14  * Initialisation :
15  *  $M = R = \text{Identité}(n)$  (n est la taille de A)
16  *  $c = b$ 
17  *
18  *
19  * Arrêt :
20  *  $c == 1$ 
21  *
22  * Progression :
23  *  $I(M,c) \wedge c \text{ pair} \Rightarrow I(M*M,c/2) \text{ (1)}$ 
24  *  $I(M,c) \wedge c \text{ impair} \Rightarrow I(M*M, (c-1)/2, R * M) \text{ (2)}$ 
25  */
26  matrix<int> operator*(matrix<int> const& A, matrix<int> const& B) {
27      return static_cast<matrix<int>>(prod(A, B));
28  }
29
30  matrix<int>& operator*=(matrix<int>& A, matrix<int> const& B) {
31      A = A*B;
32      return A;
33  }
34
35  matrix<int> pow(matrix<int> const& base, int exp) {
36      if(exp == 1) {
37          return base;
38      }
39
40      int rows = base.size1(),
41          cols = base.size2();
42
43      if(cols != rows)
44          throw new std::length_error("Matrix must have a the right sizes to be powered");
45
46      matrix<int> result = base;
47      matrix<int> rest = identity_matrix<int>(rows);
48
49      while(exp > 1) {
50          if(exp % 2 == 1) { // Si la puissance est impaire , on stocke le reste pour l'ajouter
51                          // à la fin, Cas (2)
52              rest *= result;
53              --exp;
54          }
55          result *= result;
56          exp /= 2;
57      }
58      return result * rest;
59  }
60
61 }
62
63 }

```

```
1  #include "question3.hpp"
2
3  #include "question2.hpp"
4
5  using boost::numeric::ublas::matrix;
6  using boost::numeric::ublas::prod;
7
8  namespace question3 {
9      /**
10       * Cette fonction permet de trouver le nombre de fibonacci d'indice n, en un temps  $\Theta(\log n)$ 
11       */
12     int fibonacci(int n) {
13
14         // Cas pathologiques
15         if(n == 0) {
16             return 0;
17         } else if(n == 1) {
18             return 1;
19         }
20
21
22         matrix<int> mult(2,2);
23         // i = 1
24         mult(0,0) = 1;
25         mult(0,1) = 1;
26         // i = 0
27         mult(1,0) = 1;
28         mult(1,1) = 0;
29
30         // i ++
31         mult = question2::pow(mult, n-1);
32
33         return mult(0,0);
34     }
35
36 }
```

```

1  #include <iostream>
2  #include <vector>
3
4  /**
5   * Cette fonction renvoie le premier plus long sous-tableau constant
6   *
7   * Invariant :  $I(d, f, d', k)$ 
8   *  $\Rightarrow \text{tab}[d : f]$  est le pplstc du tableau  $\text{tab}[0:k]$ 
9   *  $\Rightarrow \text{tab}[d':k]$  est le suffixe constant le plus long du tableau  $\text{tab}[0:k]$ 
10  *
11  * Initialisation :
12  *  $d = f = d' = 0$ ;
13  * Si  $n \neq 0$ 
14  *    $k = 1$ 
15  * Sinon
16  *    $k = 0$ 
17  *
18  * Arrêt :
19  *  $k == n$ 
20  * ( $n$  est la longueur de  $\text{tab}$ )
21  *
22  * Progression :
23  *  $I(d, f, d', k) \wedge \text{tab}[k+1] = \text{tab}[d'] \wedge d' - k + 1 > f - d \Rightarrow I(d', k+1, d', k+1)$  (1)
24  *  $I(d, f, d', k) \wedge \text{tab}[k+1] = \text{tab}[d'] \wedge d' - k + 1 \leq f - d \Rightarrow I(d, f, d', k+1)$  (2)
25  *  $I(d, f, d', k) \wedge \text{tab}[k+1] \neq \text{tab}[d'] \Rightarrow I(d, f, k+1, k+1)$  (3)
26  */
27
28 std::pair<int, int> pplstc(std::vector<int> tab) {
29
30     int size = tab.size(); //  $n$  est la longueur de  $\text{tab}$ 
31
32     // Initialisation //
33
34     std::pair<int, int> pplstcIndices(0,0);
35
36     if(size == 0)
37         return pplstcIndices;
38
39     pplstcIndices.second = 1;
40     std::pair<int, int> currentIndices = pplstcIndices;
41     int constant = tab[0];
42
43     // Progression //
44     for(int i = 1 ; i < size ; ++i) {
45         if(tab[i] == constant) {
46             currentIndices.second = i+1;
47             if(currentIndices.second - currentIndices.first > pplstcIndices.second -
pplstcIndices.first) {
48                 // Cas (1)
49                 pplstcIndices = currentIndices;
50             } // Sinon Cas (2)
51         } else {
52             // Cas (3)
53             constant = tab[i];
54             currentIndices.first = i;
55             currentIndices.second = i+1;
56         }
57     }
58
59     return pplstcIndices;
60 }
61
62 int main() {
63     std::vector<int> tab{1,1, 2,2,2,2,2, 3,3,3,3,3, 4,4,4,4};
64     std::pair<int, int> result = pplstc(tab);
65
66     std::cout << result.first << ", " << result.second << std::endl;
67
68     return 0;
69 }

```

```
1  #include <iostream>
2  #include <vector>
3  #include <random>
4
5  #include "question1.hpp"
6
7  std::ostream& operator<<(std::ostream& os, std::vector<int> const& tab) {
8      for(auto const& el : tab) {
9          os << el << " ";
10     }
11     return os;
12 }
13
14 int main (int argc, char const* argv[]) {
15
16     if(argc == 1) {
17         std::cout << "Usage: " << argv[0] << " n" << std::endl;
18         return 64;
19     }
20
21     std::default_random_engine random_engine((std::random_device())());
22     std::uniform_int_distribution<int> distribution(-100, 100);
23
24     distribution(random_engine);
25
26     std::vector<int> tab(std::stoi(argv[1]));
27
28     for(int& el : tab) {
29         el = distribution(random_engine);
30     }
31
32     std::pair<int, int> maxSubArrayIndices = question1::maxSubArray(tab);
33
34     // std::cout << tab << std::endl;
35     std::cout << maxSubArrayIndices.first << ", " << maxSubArrayIndices.second << std::endl;
36
37     return 0;
38 }
```

```
1  #include <stdexcept>
2
3  #include "question1.hpp"
4
5  namespace question1 {
6
7      std::pair<int, int> maxSubArray(std::vector<int> tab) {
8          int size = tab.size();
9          // T[d:f], de somme s, est le premier sous-tableau de T[0:n], de somme maximum.
10         int d = 0, f = 0, s = tab[0],
11             i = 1, dp = i, somme = 0;
12
13         for(i = 1; i < size; ++i) {
14             if(somme < 0) {
15                 somme = 0;
16                 dp=i;
17             }
18
19             somme += tab[i];
20
21             if(somme > s) {
22                 s = somme;
23                 d = dp;
24                 f = i+1;
25             }
26         }
27
28         std::pair<int, int> df(d,f);
29         return df;
30     }
31
32 }
```

```
1  #include <iostream>
2  #include <random>
3
4  #include "question1.hpp"
5  #include "question2.hpp"
6  #include "question3.hpp"
7
8  std::ostream& operator<<(std::ostream& os, std::vector<int> const& tab) {
9      for(auto const& el : tab) {
10         os << el << " ";
11     }
12     return os;
13 }
14
15 int main(int argc, char const* argv[]) {
16
17     if(argc == 1) {
18         std::cout << "Usage: " << argv[0] << " n" << std::endl;
19         return 63;
20     }
21
22     std::default_random_engine random_engine((std::random_device())());
23     std::uniform_int_distribution<int> distribution(-100, 100);
24
25     distribution(random_engine);
26
27     std::vector<int> tab(std::stoi(argv[1]));
28
29     for(int& el : tab) {
30         el = distribution(random_engine);
31     }
32
33     std::cout << tab << std::endl;
34     std::cout << "Minimum récursif :" << std::endl;
35     std::cout << question1::min(tab, 0, tab.size()) << std::endl;
36     std::cout << "Affichage récursif :" << std::endl;
37     question2::print(tab, 0, tab.size());
38     std::cout << "Affichage & l'envers récursif :" << std::endl;
39     question3::envers(tab, 0, tab.size());
40 }
```



```
1  #include "question1.hpp"
2
3  #include <limits>
4  #include <iostream>
5
6  namespace question1 {
7
8      int min(std::vector<int> const& tab, int begin, int end) {
9          if(begin == end)
10             return std::numeric_limits<int>::max();
11
12          if(begin + 1 == end)
13             return tab[begin];
14
15          int firstMin = question1::min(tab, begin, ((end-begin)/2)+begin),
16             secondMin = question1::min(tab, ((end-begin)/2)+begin, end);
17          return (firstMin < secondMin) ? firstMin : secondMin;
18      }
19
20 }
```

```
1  #include "question2.hpp"
2
3  namespace question2 {
4
5      void print(std::vector<int> const& tab, int begin, int end, bool first) {
6          if(begin + 1 == end || begin == end) {
7              std::cout << tab[begin] << " ";
8              return;
9          }
10
11          question2::print(tab, begin, ((end-begin)/2)+begin, false);
12          question2::print(tab, ((end-begin)/2)+begin, end, false);
13          if(first)
14              std::cout << std::endl;
15      }
16
17      void print(std::vector<int> const& tab, int begin, int end) {
18          print(tab, begin, end, true);
19      }
20 }
```

```
1  #include "question3.hpp"
2
3  namespace question3 {
4
5      void envers(std::vector<int> const& tab, int begin, int end, bool first) {
6          if(begin + 1 == end || begin == end) {
7              std::cout << tab[begin] << " ";
8              return;
9          }
10
11          question3::envers(tab, ((end-begin)/2)+begin, end, false);
12          question3::envers(tab, begin, ((end-begin)/2)+begin, false);
13          if(first)
14              std::cout << std::endl;
15      }
16
17      void envers(std::vector<int> const& tab, int begin, int end) {
18          envers(tab, begin, end, true);
19      }
20 }
```