```cpp
#include <iostream>

#include "question1.hpp"
#include "question2.hpp"
#include "question3.hpp"

using question1::operator*;

std::ostream& operator<<(std::ostream& os, boost::numeric::ublas::matrix<int> mat) { // Fonction permettant l'impression d'une matrice
    int rows = mat.size1(),
        cols = mat.size2();

    for(int row = 0 ; row < rows ; ++row) {
        for(int col = 0 ; col < cols ; ++col) {
            if(col == 0) {
                if(row == 0) {
                    os << "⎡ ";
                } else if(row == rows - 1) {
                    os << "⎣ ";
                } else {
                    os << "⎢ ";
                }
            }

            os << mat(row, col) << "\t";

            if(col == cols - 1) {
                if(row == 0) {
                    os << "⎤" << std::endl;
                } else if(row == rows - 1) {
                    os << "⎦" << std::endl;
                } else {
                    os << "⎥" << std::endl;
                }
            }
        }
    }

    return os;
}

int main() {

    boost::numeric::ublas::matrix<int> A(3,3);
    boost::numeric::ublas::matrix<int> B(3,3);

    int rows = A.size1();
    int cols = A.size2();

    for(int i = 0 ; i < rows ; ++i) {
        for(int j = 0 ; j < cols ; ++j) {
            A(i, j) = i + j;
            B(i, j) = i - j;
        }
    }

    std::cout << "A =" << std::endl << A   << std::endl;
    std::cout << "B =" << std::endl << B   << std::endl;
    std::cout << "A*B =" << std::endl << A*B << std::endl;
    std::cout << static_cast<boost::numeric::ublas::matrix<int>>(boost::numeric::ublas::prod(A, B)) << std::endl;
    std::cout << "A³ =" << std::endl << question2::pow(A, 3) << std::endl;

    for(int i = 0 ; i <= 30 ; ++i) {
        std::cout << "F_" << i << " = " << question3::fibonacci(i) << std::endl;
    }

    return 0;
}
```

```cpp
#include "question1.hpp"

using boost::numeric::ublas::matrix;

namespace question1 {

    matrix<int> operator*(matrix<int> const& A, matrix<int> const& B) {
        int rowsA = A.size1(), colsA = A.size2(),
            rowsB = B.size1(), colsB = B.size2();

        if(colsA != rowsB) // Vérification de la taille
            throw new std::length_error("Matrices must have a the right sizes to be 
multiplicated");

        matrix<int> result(rowsA, colsB);

        for(int row = 0 ; row < rowsA ; ++row) { // Traitement des lignes
            for(int col = 0 ; col < colsB ; ++col) { // Traitement des colonnes
                int el = 0;
                for(int i = 0 ; i < colsA ; ++i) { // Un élément est la somme des produits des 
lignes de A par les colonnes de B
                    el += A(row, i)*B(i, col);
                }
                result(row, col) = el;
            }
        }

        return result;
    }

}
```

```cpp
#include "question2.hpp"

using boost::numeric::ublas::matrix;
using boost::numeric::ublas::identity_matrix;
using boost::numeric::ublas::prod;

namespace question2 {
/**
 *  Cette fonction permet d'élever une matrice A à la puissance B
 *
 *  Invariant : I(M,R,c)
 *  ⇒ A^b = M * A^c * R
 *
 *  Initialisation :
 *  M = R = Identité(n) (n est la taille de A)
 *  c = b
 *
 *
 *  Arrêt :
 *  c == 1
 *
 *  Progression :
 *  I(M,c) ∧ c pair   ⇒ I(M*M,c/2) (1)
 *  I(M,c) ∧ c impair ⇒ I(M*M, (c-1)/2 , R * M) (2)
 */
    matrix<int> operator*(matrix<int> const& A, matrix<int> const& B) {
        return static_cast<matrix<int>>(prod(A, B));
    }

    matrix<int>& operator*=(matrix<int>& A, matrix<int> const& B) {
        A = A*B;
        return A;
    }

    matrix<int> pow(matrix<int> const& base, int exp) {
        if(exp == 1) {
            return base;
        }

        int rows = base.size1(),
            cols = base.size2();

        if(cols != rows)
            throw new std::length_error("Matrix must have a the right sizes to be powered");

        matrix<int> result = base;
        matrix<int> rest = identity_matrix<int>(rows);

        while(exp > 1) {
            if(exp % 2 == 1) { // Si la puissance est impaire , on stocke le reste pour l'ajouter
à la fin, Cas (2)
                rest *= result;
                --exp;
            }

            result *= result;

            exp /= 2;
        }

        return result * rest;
    }

}
```

```cpp
#include "question3.hpp"

#include "question2.hpp"

using boost::numeric::ublas::matrix;
using boost::numeric::ublas::prod;

namespace question3 {
    /**
     *  Cette fonction permet de trouver le nombre de fibonacci d'indice n, en un temps O(log n)
     */
    int fibonacci(int n) {

        // Cas pathologiques
        if(n == 0) {
            return 0;
        } else if(n == 1) {
            return 1;
        }


        matrix<int> mult(2,2);
        // i = 1
        mult(0,0) = 1;
        mult(0,1) = 1;
        // i = 0
        mult(1,0) = 1;
        mult(1,1) = 0;

        // i ++
        mult = question2::pow(mult, n-1);

        return mult(0,0);
    }

}
```

```cpp
#include <iostream>
#include <vector>

/**
 *  Cette fonction renvoie le premier plus long sous-tableau constant
 *
 *  Invariant : I(d, f, d', k)
 *  ⇒ tab[d :f] est le pplstc du tableau tab [0:k]
 *  ⇒ tab[d':k] est le suffixe constant le plus long du tableau tab [0:k]
 *
 *  Initialisation :
 *  d = f = d' = 0;
 *  Si n ≠ 0
 *      k = 1
 *  Sinon
 *      k = 0
 *
 *  Arrêt :
 *  k == n
 *  (n est la longeur de tab)
 *
 *  Progression :
 *  I(d, f, d', k) ∧ tab[k + 1] = tab[d'] ∧ d' - k + 1 > f - d ⇒ I(d', k+1, d' , k+1) (1)
 *  I(d, f, d', k) ∧ tab[k + 1] = tab[d'] ∧ d' - k + 1 ≤ f - d ⇒ I(d , f  , d' , k+1) (2)
 *  I(d, f, d', k) ∧ tab[k + 1] ≠ tab[d']                      ⇒ I(d , f  , k+1, k+1) (3)
 */

std::pair<int, int> pplstc(std::vector<int> tab) {

    int size = tab.size(); // n est la longeur de tab

    // Initialisation //

    std::pair<int, int> pplstcIndices(0,0);

    if(size == 0)
        return pplstcIndices;

    pplstcIndices.second = 1;
    std::pair<int, int> currentIndices = pplstcIndices;
    int constant = tab[0];

    // Progression //
    for(int i = 1 ; i < size ; ++i) {
        if(tab[i] == constant) {
            currentIndices.second = i+1;
            if(currentIndices.second - currentIndices.first > pplstcIndices.second -
pplstcIndices.first) {
                // Cas (1)
                pplstcIndices = currentIndices;
            } // Sinon Cas (2)
        } else {
            // Cas (3)
            constant = tab[i];
            currentIndices.first = i;
            currentIndices.second = i+1;
        }
    }

    return pplstcIndices;
}

int main() {
    std::vector<int> tab{1,1, 2,2,2,2,2, 3,3,3,3,3, 4,4,4,4};
    std::pair<int, int> result = pplstc(tab);

    std::cout << result.first << ", " << result.second << std::endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>
#include <random>

#include "question1.hpp"

std::ostream& operator<<(std::ostream& os, std::vector<int> const& tab) {
    for(auto const& el : tab) {
        os << el << " ";
    }
    return os;
}

int main (int argc, char const* argv[]) {

    if(argc == 1) {
        std::cout << "Usage: " << argv[0] << " n" << std::endl;
        return 64;
    }

    std::default_random_engine random_engine((std::random_device()()));
    std::uniform_int_distribution<int> distribution(-100, 100);

    distribution(random_engine);

    std::vector<int> tab(std::stoi(argv[1]));

    for(int& el : tab) {
        el = distribution(random_engine);
    }

    std::pair<int, int> maxSubArrayIndices = question1::maxSubArray(tab);

    // std::cout << tab << std::endl;
    std::cout << maxSubArrayIndices.first << ", " << maxSubArrayIndices.second << std::endl;

    return 0;
}
```

```cpp
#include <stdexcept>

#include "question1.hpp"

namespace question1 {

    std::pair<int, int> maxSubArray(std::vector<int> tab) {
        int size = tab.size();
        // T[d:f], de somme s, est le premier sous-tableau de T[0:n], de somme maximum.
        int d = 0 , f = 0, s = tab[0],
            i = 1, dp = i, somme = 0;

        for(i = 1 ; i < size ; ++i) {
            if(somme < 0) {
                somme = 0;
                dp=i;
            }

            somme += tab[i];

            if(somme > s) {
                s = somme;
                d = dp;
                f = i+1;
            }
        }

        std::pair<int, int> df(d,f);
        return df;
    }

}
```

```cpp
#include <iostream>
#include <random>

#include "question1.hpp"
#include "question2.hpp"
#include "question3.hpp"

std::ostream& operator<<(std::ostream& os, std::vector<int> const& tab) {
    for(auto const& el : tab) {
        os << el << " ";
    }
    return os;
}

int main(int argc, char const* argv[]) {

    if(argc == 1) {
        std::cout << "Usage: " << argv[0] << " n" << std::endl;
        return 63;
    }

    std::default_random_engine random_engine((std::random_device()()));
    std::uniform_int_distribution<int> distribution(-100, 100);

    distribution(random_engine);

    std::vector<int> tab(std::stoi(argv[1]));

    for(int& el : tab) {
        el = distribution(random_engine);
    }

    std::cout << tab << std::endl;
    std::cout << "Minimum récursif :" << std::endl;
    std::cout << question1::min(tab, 0, tab.size()) << std::endl;
    std::cout << "Affichage récursif :" << std::endl;
    question2::print(tab, 0, tab.size());
    std::cout << "Affichage & l'envers récursif :" << std::endl;
    question3::envers(tab, 0, tab.size());
}
```

```cpp
#include "question1.hpp"

#include <limits>
#include <iostream>

namespace question1 {

    int min(std::vector<int> const& tab, int begin, int end) {
        if(begin == end)
            return std::numeric_limits<int>::max();

        if(begin + 1 == end)
            return tab[begin];

        int firstMin  = question1::min(tab, begin,  ((end-begin)/2)+begin),
            secondMin = question1::min(tab, ((end-begin)/2)+begin, end);
        return (firstMin < secondMin) ? firstMin : secondMin;
    }

}
```

```cpp
#include "question2.hpp"

namespace question2 {

    void print(std::vector<int> const& tab, int begin, int end, bool first) {
        if(begin + 1 == end || begin == end) {
            std::cout << tab[begin] << " ";
            return;
        }

        question2::print(tab, begin,  ((end-begin)/2)+begin, false);
        question2::print(tab, ((end-begin)/2)+begin, end, false);
        if(first)
            std::cout << std::endl;
    }

    void print(std::vector<int> const& tab, int begin, int end) {
        print(tab, begin, end, true);
    }
}
```

```cpp
#include "question3.hpp"

namespace question3 {

    void envers(std::vector<int> const& tab, int begin, int end, bool first) {
        if(begin + 1 == end || begin == end) {
            std::cout << tab[begin] << " ";
            return;
        }

        question3::envers(tab, ((end-begin)/2)+begin, end, false);
        question3::envers(tab, begin,  ((end-begin)/2)+begin, false);
        if(first)
            std::cout << std::endl;
    }

    void envers(std::vector<int> const& tab, int begin, int end) {
        envers(tab, begin, end, true);
    }
}
```