C++

Decode

```
#include <iostream>     // C++ header file for input/output
#include <string>       // C++ header file for strings

std::string key = "FCTUNL";
int n = 4;

bool inKey(char s){
      return key.find(s) != std::string::npos;
}

int mod(int a, int b){
      if(a >= b)
            return a%b;
      else if(a < 0) return 26 + a;
      else return a;
}

std::string Encode(const std::string &message) {
      std::string encoded = "";
      int m = 0;
      int i;
      int size = message.size();
      for(i = 0; i < size; i++){
            char s = message[i];
            if(inKey(s)){
                  encoded += key[m];
                  m = mod(m + 1, key.size());
            }
            int s2 = (s - 'A') + n;
            int newS = mod(s2, 26);
            encoded += 'A' + newS;
            if(inKey(s))
                  encoded += key[m];
      }
      return encoded ;
}
std::string Decode(const std::string &message) {

      std::string decoded = "";
      int m = 0;
      int i;
      int size = message.size();
      for(i = 0; i < size; i++){
            char s = message[i];
            if(s == key[m] && i + 2 < size && message[i + 2] == key[m+1]){
                  m++;
                  int s2 = message[i+1] - 'A';
                  int newS = mod(s2 - n, 26);
                  decoded += 'A' + newS;
                  i = i+2;
            }
            else {
                  int s2 = message[i] - 'A';
                  int newS = mod(s2 - n, 26);
                  decoded += 'A' + newS;
            }
      }
      return decoded;
}
```

```
int main()
{
	std::string original = "ATACAMOSAOAMANHECER" ;
	std::string decode = "EFXCECGTEQSWESEQETRULIUGNIV" ;

	std::cout << "Escreva a chave:\n";
	std::cin >> key;

	std::cout << "Escreva o valor de n:\n";
	std::cin >> n;

	std::cout << "Escreva a palavra a cifrar:\n";
	std::cin >> original;

	std::string encoded = Encode(original) ;
	std::cout << encoded << std::endl ;

	std::cout << "Escreva a palavra a decifrar:\n";
	std::cin >> decode;

	std::string decoded = Decode(decode) ;
	std::cout << decoded << std::endl ;

	return 0;
}
```

Sucessões

```
#ifndef _SUCCESSION_
#define _SUCCESSION_

class Succession {
public:
	virtual int First() = 0; // Reinicializa a sucessão e retorna o seu primeiro elemento
a0.
	virtual int Curr() const = 0; // Retorna o elemento corrente sem fazer avançar a
sucessão.
	virtual int Next() = 0; // Faz avançar a sucessão e retorna o novo elemento corrente.
	virtual int At(int i) = 0; // Retorna o i-ésimo elemento da sucessão, isto é ai.
Entretanto o elemento corrente passa a ser o i-ésimo elemento da sucessão.
	virtual void Print(int n) = 0; // Escreve os n primeiros elementos da sucessão.
Entretanto o elemento corrente passa a ser o n-ésimo elemento da sucessão.
};

#endif
```

```
#ifndef SUCC_H_
#define SUCC_H_

#include <iostream>
#include "Succession.h"

class Succ: public Succession {
public:
	int First();
	int Curr() const;
	int At(int i);
	void Print(int n);
```

```
	Succ(int a0);
	~Succ();

protected:
	int current;
	int a0;
};

#endif
```

```
#include "Succ.h"

Succ::Succ(int a0): current(a0), a0(a0){}

Succ::~Succ(){};

int Succ::Curr() const {
	return current;
}

int Succ::First() {
	return current = a0;
}

int Succ::At(int i) {
	First();
	for(int a = 0; a < i; a++)
		Next();
	return Curr();
}

void Succ::Print(int n){
	for(int i = 0; i <= n; i++){
		current = At(i);
		std::cout << " " << At(i) << " ";
	}
	std::cout << "\n";
}
```

Sucessão composta alternante: Caracteriza-se por duas sucessões a e b. Os elementos da sucessão alternante são gerados pela seguinte ordem: a_1, b_1, a_2, b_2, ..., a_i, b_i, ...

```
#ifndef SUCCALTER_H_
#define SUCCALTER_H_

#include "Succ.h"

class SuccAlter: public Succ {
public:
	int First();
	int Next();

	SuccAlter(Succession *a, Succession *b);
	~SuccAlter();

protected:
	Succession *a, *b;
	char nextSucc;
};
#endif
```

```
#include "SuccAlter.h"

SuccAlter::SuccAlter(Succession *a, Succession *b):Succ(a->First()), a(a), b(b),
nextSucc('b'){}

SuccAlter::~SuccAlter(){};

int SuccAlter::First() {
      b->First();
      nextSucc = 'b';
      return current = a->First();
}

int SuccAlter::Next(){
      if(nextSucc == 'a'){
            nextSucc = 'b';
            return a->Next();
      } else {
            nextSucc = 'a';
            return b->Next();
      }
}
```

Sucessão aritmética: Caracteriza-se por um primeiro elemento a_0 e por um incremento fixo *inc*. Os elementos da sucessão são gerados usando a equação $a_i = a_{i-1}$+inc.

```
#ifndef SUCCARIT_H_
#define SUCCARIT_H_

#include "Succ.h"

class SuccArit:public Succ{
public:
      SuccArit(int a0, int inc);
      ~SuccArit();
      int Next();

protected:
      int inc;
};

#endif
```

```
#include "SuccArit.h"

SuccArit::SuccArit(int a0, int inc):Succ(a0), inc(inc){}

SuccArit::~SuccArit(){}

int SuccArit::Next(){
      return current = current+inc;
}
```

Sucessão aritmética otimizada: Sucessão aritmética em que a operação At é muito eficiente. [Por vezes define-se uma subclasse, só para se tornarem mais eficientes determinadas operações. É o caso.]

```
#ifndef SUCCARITOPT_H_
#define SUCCARITOPT_H_

#include "SuccArit.h"

class SuccAritOpt:public SuccArit{
public:
	SuccAritOpt(int a0, int inc);
	~SuccAritOpt();
	int At(int i);

protected:
	int inc;
};

#endif


#include "SuccAritOpt.h"

SuccAritOpt::SuccAritOpt(int a0, int inc):SuccArit(a0, inc){}

SuccAritOpt::~SuccAritOpt(){}

int SuccAritOpt::At(int i){
	return current = a0 + (i*inc);
}
```

Sucessão constante: Caracteriza-se pelo primeiro elemento a_0, apenas. Os elementos da sucessão constante são todos iguais ao primeiro, i.e. $a_i=a_0$.

```
#ifndef SUCCCONT_H_
#define SUCCCONT_H_

#include "Succ.h"

class SuccConst:public Succ{
public:
	SuccConst(int a0);
	~SuccConst();
	int Next();
};

#endif /* SUCCCONT_H_ */



#include "SuccConst.h"

SuccConst::SuccConst(int a0):Succ(a0){}

SuccConst::~SuccConst(){}

int SuccConst::Next(){
	return current;
}
```

Sucessão de fibonnaci: Caracteriza-se pelos dois primeiros elementos a_0 e a_1. Os elementos da sucessão são gerados usando a equação $a_i=a_{i-1}+a_{i-2}$

```
#ifndef SUCCFIB_H_
#define SUCCFIB_H_

#include "Succ.h"

class SuccFib:public Succ{
public:
	int First();
	int Next();
	SuccFib(int first, int second);
	~SuccFib();

protected:
	int a1, prev, index;
};

#endif /* SUCCFIB_H_ */
#include "SuccFib.h"

SuccFib::SuccFib(int first, int second):Succ(first), a1(second), prev(first), index(0){
	First();
}

SuccFib::~SuccFib(){}

int SuccFib::First(){
	index = 0;
	current = a1;
	return prev = a0;
}

int SuccFib::Next(){
	if(index == 0){
		index = 1;
		return prev;
	} else if (index == 1)
		return current;
	else {
		int aux = prev;
		prev = current;
		return current = current + aux;
	}
}
```

Sucessão filtro: Caracteriza-se por uma sucessão a e por um número inteiro *filtro*. Os elementos da sucessão filtro são, pela mesma ordem, os elementos da sucessão que são múltiplos de *filtro*.

```
#ifndef SUCCFILTER_H_
#define SUCCFILTER_H_

#include "Succ.h"

class SuccFilter:public Succ{
public:
	int First();
```

```
	int Next();
	SuccFilter(Succession *a, int number);
	~SuccFilter();

protected:
	Succession *a;
	int number;
	int next;
};


#endif


#include "SuccFilter.h"

SuccFilter::SuccFilter(Succession *a, int number): Succ(0), a(a), number(number), next(0){
	First();
}

SuccFilter::~SuccFilter(){}

int SuccFilter::First(){
	a->First();
	Next();
	return current = next;
}

int SuccFilter::Next(){
	current = next;
	do{
		next = a->Next();
	} while(next%3 != 0);
	return current;
}
```

Sucessão geométrica: Caracteriza-se por um primeiro elemento a_0 e por uma base fixa *base*. Os elementos da sucessão são gerados usando a equação $a_i=a_{i-1}$*base.

```
#ifndef SUCCGEOM_H_
#define SUCCGEOM_H_

#include "Succ.h"

class SuccGeom:public Succ{
public:
	SuccGeom(int a0, int base);
	~SuccGeom();
	int Next();

protected:
	int base;
};

#endif


#include "SuccGeom.h"

SuccGeom::SuccGeom(int a0, int base):Succ(a0), base(base){}
SuccGeom::~SuccGeom(){}
```

```
int SuccGeom::Next(){
      return current = current*base;
}
```

Sucessão composta soma: Caracteriza-se por duas sucessões a e b. A sucessão soma define-se pela equação $c_i=a_i+b_i$

```
#ifndef SUCCSUM_H_
#define SUCCSUM_H_

#include "Succ.h"

class SuccSum: public Succ {
public:
      int First();
      int Next();

      SuccSum(Succession *a, Succession *b);
      ~SuccSum();

protected:
      Succession *a, *b;
};

#endif
```

```
#include "SuccSum.h"

SuccSum::SuccSum(Succession *a, Succession *b):Succ( a->First() + b->First() ), a(a), b(b){}

SuccSum::~SuccSum(){};

int SuccSum::First() {
      return current = a->First() + b->First();
}

int SuccSum::Next(){
      return current = a->Next() + b->Next();
}
```

C

```
#ifndef _LinkedList_
#define _LinkedList_

#include <stdbool.h>

typedef double Data ;
typedef struct Node *List ;

List ListMakeRange(Data a, Data b) ;
int ListLength(List l) ;
bool ListGet(List l, int idx, Data *res) ;
List ListPutAtHead(List l, Data val) ;
List ListPutAtEnd(List l, Data val) ;
void ListPrint(List l) ;
```

```
List ListClone(List l) ;              // Cria uma cópia duma lista, em que todos os nós são
novos.
List ListAppend(List l1, List l2) ;   // No final de l1, acrescenta uma cópia da lista l2
(usando Clone, claro)
List ListRev(List l) ;                // Inverte uma lista, duplicando todos os nós da lista
original.
List ListRev2(List l) ;               // Inverte uma lista, aproveitando os nós da lista
original.
List ListUniq(List l) ;               // Elimina duma lista, todos os nós com valores
repetidos.
                                                        // Note que a lista não está ordenada.
Use a operação free.

#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "LinkedList.h"

typedef struct Node {
      Data data ;
      List next ;
} Node ;

static List NewNode(Data val, List next)
{
      List n = malloc(sizeof(Node)) ;
      if( n == NULL )
            return NULL ;
      n->data = val ;
      n->next = next ;
      return n ;
}

List ListMakeRange(Data a, Data b)
{
      if( a > b )
            return NULL ;
      double i ;
      List l = NewNode(a, NULL), p = l ;
      for( i = a + 1 ; i <= b ; i++ )
            p = p->next = NewNode(i, NULL) ;
      return l ;
}

/* Outra maneira, mais palavrosa, de escrever a funcao anterior:

List ListMakeRange(Data a, Data b)
{
    if( a > b )
        return NULL ;
    double i ;
    List l = NewNode(a, NULL) ;
    List p = l ;
    for( i = a + 1 ; i <= b ; i++ ) {
        List q = NewNode(i, NULL) ;
        p->next = q ;
        p = q ;
    }
    return l ;
}
```

```
 */

int ListLength(List l) {
	int count ;
	for( count = 0 ; l != NULL ; l = l->next, count++ ) ;
	return count ;
}

bool ListGet(List l, int idx, Data *res)
{
	int i ;
	for( i = 0 ; i < idx && l != NULL ; i++, l = l->next ) ;
	if( l == NULL )
		return false ;
	else {
		*res = l->data ;
		return true ;
	}
}

List ListPutAtHead(List l, Data val)
{
	return NewNode(val, l) ;
}

List ListPutAtEnd(List l, Data val)
{
	if( l == NULL )
		return NewNode(val, NULL) ;
	else {
		List p ;
		for( p = l ; p->next != NULL ; p = p->next ) ;
		p->next = NewNode(val, NULL) ;
		return l ;
	}
}

void ListPrint(List l)
{
	for( ; l != NULL ; l = l->next )
		printf("%lf\n", l->data) ;
}

List ListClone(List l)
{
	List p = NULL;
	if(l != NULL){
		p = malloc(sizeof(Node));
		p->data = l->data;
		p->next = NULL;
		l = l->next;
	}
	List result = p;
	for( ; l != NULL; l = l->next){
		p->next = malloc(sizeof(Node));
		p = p->next;
		p->data = l->data;
		p->next = NULL;
	}
	return result;
}

List ListAppend(List l1, List l2)
{
```

```
	List p = ListClone(l2);
	List result = l1;
	if(l1 == NULL){
		return p;
	}
	for( ; l1->next != NULL ; l1 = l1->next){}
	l1->next = p;
	return result;

}

List ListRev(List l)
{
	List p;
	for(p = NULL; l!=NULL; l = l->next){
		p = NewNode(l->data, p);
	}
	return p;
}

List ListRev2(List l)
{
	List p = l;
	if(p != NULL){
		p = l->next;
		l->next = NULL;
	}
	while(p != NULL) {
		List temp = p->next;
		p->next = l;
		l = p;
		p = temp;
	}
	return l;
}

List ListUniq(List l)
{
	List p;
	List result = l;
	for( ; l != NULL; l = l->next){
		for(p = l; p->next != NULL; p = p->next){
			List temp = p->next;
			if(temp->data == l->data){
				p->next = temp->next;
				free(temp);
			}
		}
	}
	return result;

}
```

OCAML

```
(* ---------- ex01 ---------- *)

1 + 5 ;;

let x = 1 ;;
let y = x*2 + 1 ;;

(* ---------- ex02 ---------- *)

print_string "Hello World!" ;;
print_int (10 + 3) ;;
print_char 'c' ;;
print_float 1.0 ;;

(* ---------- ex05 ---------- *)

let f x = x + 1 ;;
f 2 ;;

fun x -> x + 1 ;;
(fun x-> x + 1) 2 ;;

(* ---------- ex06 ---------- *)

let rec fact x = if x = 0 then 1 else x * fact (x-1) ;;
fact 5 ;;

(* ---------- ex07 ---------- *)

let f x = x + x ;;
f 3 ;;

let a x = x +. x ;;
a 3.0 ;;

let s x = x ^ x ;;
s "lol" ;;

let d (x,y) = x + y ;;
d (2,4) ;;

let g (x,y) = x +. y ;;
g (2.0,4.0) ;;

let h (x,y) = (y, x) ;;
h (3,9) ;;

let j (x,y) = x = y ;;
j (2,2) ;;
j (2,3) ;;

let k x y = x + y ;;
k 2 4 ;;

(* ---------- ex08 ---------- *)

let rec mdc (m,n) = if (n = 0) then m else mdc (n,(m mod n)) ;;
mdc (9,3) ;;
mdc (9,4) ;;
```

```
(* ---------- ex12 ---------- *)

let f1 x = x + 1 ;;
f1 2 ;;

let f2 x = f1 x ;;
f2 2 ;;

let f3 x = 1 + x 5 ;;
f3 (fun y -> y + 1) ;;

let f4 x = x ;;
f4 5 ;;

(* ---------- ex13 ---------- *)

let rec succAll l =
      match l with
            | [] -> []
            | el::xl -> (el+1) :: succAll xl
;;

succAll [] ;;
succAll [3; 6; 1; 0; -4] ;;

(* ---------- ex14 ---------- *)

let rec belongs n l =
      match l with
            | [] -> false
            | el::xl -> if (el = n) then true else belongs n xl
;;

belongs 4 [1;2;3;4;5;6] ;;
belongs 4 [1;2] ;;

let rec union l1 l2 =
      match l1 with
            | [] -> l2
            | el::xl1 -> if (belongs el l2) then union xl1 l2 else el::union xl1 l2
;;

union [7;3;9] [2;1;9] ;;

let rec inter l1 l2 =
      match l1 with
            | [] -> []
            | el::xl1 -> if (belongs el l2) then el::inter xl1 l2 else inter xl1 l2
;;

inter [7;3;9] [2;1;9] ;;

let rec diff l1 l2 =
      match l1 with
            | [] -> []
            | el::xl1 -> if (belongs el l2) then diff xl1 l2 else el::diff xl1 l2
;;

diff [7;3;9] [2;1;9] ;;

let rec ins v l =
      match l with
```

```
            | [] -> []
            | x::xl -> (v::x) :: ins v xl
;;

let rec power l =
      match l with
            | [] -> [[]]
            | el::xl -> power xl @ ins el (power xl)
;;

power [] ;;
power [2;3] ;;
power [1;2;3] ;;

(* ---------- ex15 ---------- *)

let rec nat n = if (n=0) then [] else (n-1) :: nat (n-1) ;;

nat 0 ;;
nat 1 ;;
nat 2 ;;
nat 5 ;;

(* ---------- ex16 ---------- *)

let equal n l =
      match l with
            | [] -> false
            | el::l -> if (el = n) then true else false
;;

equal 1 [1; 0] ;;
equal 0 [0; 1] ;;
equal 0 [1; 0] ;;


let rec count n l =
      match l with
            | [] -> 0
            | el::xl -> if (el = n) then 1 + count n xl else count n []
;;

count 10.1 [10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.1; 10.0] ;;
count 10.0 [10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.1; 10.0] ;;
count 10.0 [10.0; 10.0; 10.1; 10.0] ;;

let rec remove n l =
      match l with
            | [] -> []
            | (el,_)::xl -> if (n = el) then remove n xl else l
;;

remove 10.1 [(10.1, 7); (10., 2); (10.1, 1); (10., 1)] ;;

let rec pack l =
      match l with
            | [] -> []
            | el::xl -> (el, count el l)::remove el (pack xl)
;;

pack [] ;;
pack [10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.1; 10.0; 10.0; 10.1; 10.0] ;;

let rec unpackaux n el = if (n = 0) then [] else (el)::(unpackaux (n-1) el) ;;
```

```
unpackaux  7 10.1 ;;
unpackaux  2 10.0 ;;
unpackaux  1 10.1 ;;

let rec unpack l =
      match l with
            | [] -> []
            | (el,n)::xl -> (unpackaux n el)::unpack xl
;;

unpack [] ;;
unpack [(10.1, 7); (10.0, 2); (10.1, 1); (10.0,1)] ;;
```

```
(* ---------- ex19 ---------- *)

type 'a tree = Nil | Node of 'a * 'a tree * 'a tree ;;

let rec howMany x tree =
      match tree with
            | Nil -> 0
            | Node(e,l,r) -> (if (x = e) then 1 else 0) + (howMany x l) + (howMany x r)
;;

howMany 3 (Node(1, Node(2, Nil, Nil) , Node(3, Nil, Nil))) ;;
howMany 4 (Node(1, Node(2, Nil, Nil) , Node(3, Nil, Nil))) ;;

let rec eqPairs tree =
      match tree with
            | Nil -> 0
            | Node ((x,y),l,r) -> (if (x=y) then 1 else 0) + eqPairs l + eqPairs r
;;

eqPairs (Node((1,1), Node((2,1), Nil, Nil) , Node((3,0), Nil, Nil))) ;;
eqPairs (Node((1,1), Node((2,2), Nil, Nil) , Node((2,2), Nil, Nil))) ;;

let rec treeToList tree =
      match tree with
            | Nil -> []
            | Node (e,l,r) -> e :: treeToList l @ treeToList r
;;

treeToList (Node(1, Node(2, Nil, Nil) , Node(3, Nil, Nil))) ;;
treeToList (Node((1,1), Node((2,2), Nil, Nil) , Node((2,2), Nil, Nil))) ;;

(* ---------- ex20 ---------- *)

let rec height tree =
      match tree with
            | Nil -> 0
            | Node(_,l,r) -> 1 + max (height r) (height l)
;;

height (Node(1, Nil, Nil)) ;;
height (Node(1, Node(2, Nil, Nil) , Nil)) ;;
height (Node(1, Node(2, Nil, Nil) , Node(3, Nil, Node(4, Nil, Nil)))) ;;


let balanced tree =
      match tree with
            | Nil -> true
            | Node(_,l,r) -> if (height l = height r) then true else false
;;
```

```
balanced Nil ;;
balanced (Node(3,Node(5,Nil,Nil),Node(6,Nil,Nil))) ;;
balanced (Node(1,Nil,Node(2,Nil,Node(3,Nil,Nil)))) ;;

(* ---------- ex21 ---------- *)

let rec belongs n l =
      match l with
            | [] -> false
            | el::xl -> if (el = n) then true else belongs n xl
;;

let rec union l1 l2 =
      match l1 with
            | [] -> l2
            | el::xl1 -> if (belongs el l2) then union xl1 l2 else el::union xl1 l2
;;

let rec subtrees tree =
      match tree with
            | Nil -> [Nil]
            | Node(e,l,r) -> Node(e,l,r) :: union (subtrees l) (subtrees r)
;;

subtrees Nil ;;
subtrees (Node(5,Nil,Node(6,Nil,Nil))) ;;

(* ---------- ex22 ---------- *)

let rec spring x tree =
      match tree with
            | Nil -> Node (x, Nil, Nil)
            | Node (e,l,r) -> Node (e, spring x l, spring x r)
;;

spring 9 Nil ;;
spring 9 (Node(5,Nil,Node(6,Nil,Nil))) ;;

(* ---------- ex23 ---------- *)

let rec fall tree =
      match tree with
            | Nil -> Nil
            | Node (_,Nil,Nil) -> Nil
            | Node (e,l,r) -> Node (e, fall l, fall r)
;;

fall Nil ;;
fall (Node(5,Nil,Node(6,Nil,Nil))) ;;
fall (Node(1,Node(2,Node(4, Nil, Nil),Node(5, Nil, Nil)),Node(3,Node(6, Nil, Nil),Nil))) ;;

(* ---------- ex24 ---------- *)

type 'a ntree = NNil | NNode of 'a * 'a ntree list

let rec treeToListNaux l =
      match l with
            | [] -> []
            | NNil::xl -> treeToListNaux xl
            | NNode (el,v)::xl -> el :: treeToListNaux v @ treeToListNaux xl
;;

let treeToListN tree = treeToListNaux [tree] ;;
```

```
treeToListN (NNode(1, [NNode(2,[NNil]); NNode(3,[NNil]); NNode(4,[NNode(5,[NNil])])])) ;;
treeToListN (NNode(1, [NNode(2,[NNode(3,[NNil]); NNode(4,[NNil])])])) ;;

let rec springl v l =
match l with
  | [] -> []
  | NNil::xs -> NNode(v,[]) :: springl v xs
  | NNode(x,[])::xs -> NNode(v,[NNode(v,[])]) :: springl v xs
  | NNode(x,c)::xs -> NNode (x, springl v c) :: springl v xs
;;

let spring v t = List.hd (springl v [t]) ;;

spring 9 (NNode(1, [NNode(2,[NNil]); NNode(3,[NNil])])) ;;
spring 9 (NNode(1, [NNode(2,[NNil; NNode(3,[]); NNil; NNode(4,[])])])) ;;

(* ---------- ex25 ---------- *)
```

```
(* ---------- ex26 ---------- *)

type matrix = int list list ;;

let rec suml l =
      match l with
            | [] -> 0
            | el::xl -> el + suml xl
;;

let rec sum m =
      match m with
            | [] -> 0
            | []::xm -> sum xm
            | [v::vs] -> v + suml vs
            | (v::vs)::xm -> v + suml vs + sum xm
;;

sum [[1; 2; 3]; [4; 5; 6]] ;;
sum [[1; 2; 3]; []; [4; 6]] ;;
sum [[1; 2; 3]; []; [4; 5; 6]] ;;

let rec mxl m =
      match m with
            | [] -> 0

            | [el] -> el
            | el::xl -> max (el) (mxl xl)
;;

let rec mx m =
      match m with
            | [] -> 0
            | []::xm -> 0

            | [v::vs] -> max v (mxl vs)
            | (v::vs)::xm -> max (max v (mxl vs)) (mx xm)
;;

mx [[1; 2; 8]; [4; 6]] ;;
mx [[9; 2; 3]; [4; 5; 6]] ;;
mx [[1; 2; 3]; [4; 5; 6]] ;;

let rec makel x = if (x = 0) then [] else 0 :: makel (x-1) ;;
```

```
let rec make x y = if (y = 0) then [] else makel x :: make x (y-1) ;;

make 2 2 ;;
make 3 5 ;;
make 0 0 ;;
make 1 1 ;;

let rec showl l =
      match l with
            | [] -> ()
            | el::xl -> Printf.printf "%4d" el ; showl xl
;;

let rec show m =
      match m with
            | [] -> ()
            | []::xm -> () ; show xm
            | (v::vs)::xm -> Printf.printf "%4d" v ; showl vs ; print_string "\n" ; show xm
;;

show [[1; 2; 3]; [4; 5; 6]] ;;
show [[1; 2; 3]; [4; 5; 6]; [7; 8; 9]] ;;

let rec loadLine n ci =
      if n = 0 then []
      else let line = int_of_string (input_line ci) in line :: loadLine (n-1) ci
;;

let rec loadMatrix m n ci =
      if m = 0 then []
      else let line = loadLine n ci in line :: loadMatrix (m-1) n ci
;;

let rec load ni =
      let ci = open_in ni in
      let m = int_of_string (input_line ci) in
      let n = int_of_string (input_line ci)in
      let res = loadMatrix m n ci in
      close_in ci ; res
;;

   load "matrix.txt" ;;
```

```
(* ---------- exA ---------- *)

type 'a matrix = 'a list list ;;

let rec count b m =
      match m with
            | [] -> 0
            | []::xs -> 0 + count b xs
            | (v::vs)::xs -> (if (v = b) then 1 else 0) + count b [vs] + count b xs
;;

count 4 [[1; 2; 3]; [4; 5; 1]] ;;
count 1 [[1; 2; 3]; [4; 5; 1]] ;;

(* ---------- exB ---------- *)

let rec first m =
      match m with
            | [] -> []
            | []::xs -> first xs
```

```
            | [v::vs] -> [v]
            | (v::vs)::xs -> v :: first xs
;;

first [[1; 2; 3]] ;;
first [[1; 2; 3]; [4; 5; 1]] ;;
first [[1; 0; 0]; [2; 0; 0]; [3; 0; 0]; [4; 0; 0]] ;;

(* ---------- exC ---------- *)

let rec rest m =
      match m with
            | [] -> []
            | []::xs -> rest xs
            | [v::vs] -> [vs]
            | (v::vs)::xs -> vs :: rest xs
;;

rest [[1; 2; 3]] ;;
rest [[3]; [4; 6; 6]] ;;
rest [[1; 2; 3]; [4; 5; 1]] ;;
rest [[1; 0; 0]; [2; 0; 0]; [3; 0; 0]; [4; 0; 0]] ;;

(* ---------- exD ---------- *)

let rec returnEl n l =
      match l with
            | [] -> []
            | el::xl -> if (n=0) then [el] else returnEl (n-1) xl
;;

returnEl 0 [1; 2; 3; 4] ;;
returnEl 1 [1; 2; 3; 4] ;;
returnEl 2 [1; 2; 3; 4] ;;

let rec diag m =
      match m with
            | [] -> []
            | []::xm -> []

            | [el::xm] -> [el]
            | (el::xl)::xm -> el::(diag (rest xm))
;;

diag [[1; 2; 3]; [4; 5; 1]] ;;
diag [[1; 0; 0]; [0; 2; 0]; [0; 0; 3]; [4; 5; 6]] ;;

(* ---------- exE ---------- *)

let rec trans m =
      match rest m with
            | [] -> []
            | _  -> first m :: trans (rest m)
;;

trans [[1; 2; 3]; [4; 5; 1]] ;;
trans [[1; 0; 0]; [0; 2; 0]; [0; 0; 3]; [4; 5; 6]] ;;

(* ---------- exF ---------- *)
```

JAVASCRIPT

```
<HTML>

<HEAD>
<TITLE>A44</TITLE>

<SCRIPT TYPE="text/javascript">

function isPrime(num)
{
    for (i=2; i<num; i++)
        if( num % i == 0)
            return false;

    return true;
}

function Prime(form)
{
    if(isPrime(form.text1.value))
        form.text2.value = "Prime!"
    else
        form.text2.value = "Not Prime!"
}

</SCRIPT>
</HEAD>

<BODY>
<H1>Prime</H1>
<FORM NAME="form1">
     <INPUT TYPE="text" NAME="text1" VALUE="0" SIZE=10 style="{ text-align: right }">
   : <INPUT TYPE="text" NAME="text2" SIZE=10 style="{ text-align: right }">
 <p> <INPUT TYPE="button" NAME="button1" VALUE="Add" OnClick="Prime(form)">
</FORM>
</BODY>

</HTML>
```

```
<HTML>

<HEAD>
<TITLE>A45</TITLE>

<SCRIPT>

function RunForm(form, site)
{
    var keyword = form.keyword.value ;
    if( keyword == "" )
        alert("Error: Empty Search") ;
    else
        document.location = site + keyword ;
}

</SCRIPT>
</HEAD>

<BODY>
<H1>Search</H1>
```

```
<FORM NAME="form">
    <INPUT TYPE="text" NAME="keyword" SIZE=40>
    <P><INPUT TYPE="button" NAME="button1" VALUE="Google" OnClick="RunForm(form,
'http://www.google.com/search?q=')">
    <INPUT type="Reset">
</FORM>
</BODY>

</HTML>
```

```
<HTML>

<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=utf-8">
<TITLE>MyDocument</TITLE>

<SCRIPT TYPE="text/javascript">

function Compute(n) {
    var test = new Array();
    var res = new Array();
    var val = "";

    for(i = 2; i<= n ; i++)
        test[i]= true;

    for(i = 2; i<= n; i++){
        if(test[i]){
            res.push(i);
            for(j = 2*i; j<= n; j+=i)
                test[j] = false;
        }
    }

    for(i = 0; i< res.length ; i++)
        val+= (res[i] + " ");

    return val;

}

function RunForm1(form){
    form.text2.value = Compute(parseInt(form.text1.value));

}
</SCRIPT>
</HEAD>

<BODY>
<H1>Crivo</H1>
<FORM NAME="form1">
    Gerar primos até: <INPUT TYPE="text" NAME="text1" VALUE="" SIZE=10 style="{ text-align:
right }">
    <INPUT TYPE="button" NAME="button1" VALUE="Ok" OnClick="RunForm1(form)">
<p> <TEXTAREA NAME="text2" VALUE="" ROWS = "20" COLS = "100" READONLY style="{ text-align:
right; font-weight : bold }">
    </TEXTAREA>

</FORM>
</BODY>

</HTML>
```