À faire en cours :
— correction de l'exercice concernant le calcul des C_n^p ;
— Écrire une fonction récursive (ex. fact) et en faire une version itérative simulant une pile.

```c
/*!\file pile.h
 * \brief Bibliothèque de gestion de (une) pile de taille fixe
 * \author Farès Belhadj amsi@ai.univ-paris8.fr
 * \date October 02, 2013
 */
#ifndef _PILE_H
#define _PILE_H
/*!\brief taille de la pile (statique) */
#define PILE_MAX 256

#ifdef __cplusplus
extern "C" {
#endif

  extern void push(int v);
  extern int pop(void);
  extern int vide(void);

#ifdef __cplusplus
}
#endif

#endif
```

```c
/*!\file pile.c
 * \brief Bibliothèque de gestion de (une) pile de taille fixe
 * \author Farès Belhadj amsi@ai.univ-paris8.fr
 * \date October 02, 2013
 */
#include "pile.h"
#include <stdlib.h>
#include <assert.h>

/*!\brief indice indiquant le haut de la pile.*/
static int haut = -1;
/*!\brief tableau static utilisé pour le stockage de la pile.*/
static int pile[PILE_MAX];

/*!\brief Empiler la valeur \a v dans la pile.
 * \param v la valeur à empiler
 */
void push(int v) {
  pile[++haut] = v;
}

/*!\brief dépiler et renvoyer la valeur de l'élément en haut de la pile.
 * \return la valeur en haut de la pile.
 */
int pop(void) {
  return pile[haut--];
}

/*!\brief Indique si la pile est vide.
 * \return vrai si la pile est vide, faux sinon.
 */
int vide(void) {
  return haut < 0;
}
```

FIGURE 1 – Bibliothèque de gestion de pile (`pile.h et pile.c`).

```c
#include <stdio.h>
#include "pile.h"
#define MAX 256
static void infixe2postfixe(char * s, char * d) {
  while(*s) {
    if(*s >= '0' && *s <= '9') {
      do {
        *d++ = *s++;
      } while( *s >= '0' && *s <= '9');
      *d++ = ' ';
      if(!*s) break;
    }
    if((*s == ')')  && !vide()) { *d++ = (char)pop(); *d++ = ' '; }
    else if(*s == '+') push((int) *s);
    else if(*s == '*') push((int) *s);
    s++;
  }
  while(!vide()) {*d++ = (char)pop(); *d++ = ' '; }
  *d = '\0';
}
int main(void) {
  char source[MAX], destination[MAX<<1];
  do {
    if(!fgets(source, MAX, stdin)) break;
    infixe2postfixe(source, destination);
    printf("l'expression infixee : %s\n", source);
    printf("s'ecrit : %s en postfixe\n", destination);
  } while(1);
  return 0;
}
```

```
SHELL = /bin/sh
#definition des commandes utilisees
CC = gcc
ECHO = echo
RM = rm -f
TAR = tar
MKDIR = mkdir
CHMOD = chmod
CP = cp
#declaration des options pour gcc
PG_FLAGS =
CFLAGS = -Wall -O3 $(PG_FLAGS)
CPPFLAGS = -I.
LDFLAGS = $(PG_FLAGS)
#definition des fichiers et dossiers
PROG = infix2postfix
PACKAGE=infix2postfix
VERSION = 0.1
distdir = $(PACKAGE)-$(VERSION)
HEADERS = pile.h
OBJ = infix2postfix.o pile.o
SOURCES = $(OBJ:.o=.c)
DISTFILES = $(SOURCES) Makefile $(HEADERS)

all: $(PROG)

$(PROG): $(OBJ)
        $(CC) $(OBJ) $(LDFLAGS) -o $(PROG)

%.o: %.c
        $(CC) $(CPPFLAGS) $(CFLAGS) -c $<

dist: distdir
        $(CHMOD) -R a+r $(distdir)
        $(TAR) czvf $(distdir).tgz $(distdir)
        $(RM) -r $(distdir)
distdir: $(DISTFILES)
        $(RM) -r $(distdir)
        $(MKDIR) $(distdir)
        $(CHMOD) 777 $(distdir)
        $(CP) -rf $(DISTFILES) $(distdir)
clean:
        @$(RM) $(PROG) $(OBJ) *~ $(distdir).tgz gmon.out
```

FIGURE 2 – Transformer une expression infixée en expression postfixée en utilisant la structure de données pile (`infix2postfix.c et Makefile`).