```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <ctype.h>

#define LGMAX 1000
#define BASE 10

typedef short BigNat[LGMAX]; // les chiffres du  BigNat sont compris entre 0 et 9
                             // le chiffre de poids faible est à l'indice 0
const BigNat BIG_ZERO = {0};
const BigNat BIG_UN   = {1};

/*****************************************************************************
 *                          Fonctions d'initialisation
 *****************************************************************************/

/*
 *   Rôle : n1 = n2;
 */
void initBn(BigNat n1, const BigNat n2) {
  for (int i=0; i<LGMAX; i++)
    n1[i]=n2[i];
}

/*
 *   Rôle : n1 = e;
 */
void initE(BigNat n, int e) {
  int i=0;
  do {
    n[i++] = e%BASE;
    e/=BASE;
  } while (e!=0);
  //
  while (i<LGMAX) n[i++]=0;
}

/*
 * Antécédent : s chaîne de caractères : suite de chiffres en base 10
 * Rôle : initialise le BigNat n à partir de la chaîne s
 */
void initS(BigNat n, const char s[]) {
  int j, i=strlen(s)-1;
  assert(i<LGMAX);
  for (j=0; i>=0; i--, j++) {
    assert(isdigit(s[i]));
    n[j] = s[i]-'0';
  }
  // compléter avec des 0
  while (j<LGMAX) n[j++]=0;
}

/*****************************************************************************
 *                          Fonctions de comparaison
 *****************************************************************************/

/*
 * Rôle : renvoie l'indice du chiffre de poids fort dans n
 */
int getIndexChiffreMax(const BigNat n) {
  int i=LGMAX-1;
  while (n[i]==0 && i>=0) i--;
  return i;
```



```c
}

/*
 * Rôle : renvoie la comparaison de 2 BigNat  :
 *         0 =>  n1==n2
 *         1 =>  n1>n2
 *        -1 =>  n1<n2
 */
int cmp(const BigNat n1, const BigNat n2) {
  int id1 = getIndexChiffreMax(n1);
  int id2 = getIndexChiffreMax(n2);
  if (id1<id2) return -1;
  if (id1>id2) return 1;
  // else même taille
  int i=id1;
  while (i>=0 && n1[i]==n2[i]) i--;
  if (i<0) return 0;
  return n1[i]-n2[i];
}

int egal(const BigNat n1, const BigNat n2) {
  return cmp(n1, n2)==0;
}

int inf(const BigNat n1, const BigNat n2) {
  return cmp(n1, n2)<0;
}

int infEgal(const BigNat n1, const BigNat n2) {
  return inf(n1,n2) || egal(n1, n2);
}

int sup(const BigNat n1, const BigNat n2) {
  return cmp(n1, n2)>0;
}

int supEgal(const BigNat n1, const BigNat n2) {
  return sup(n1,n2) || egal(n1, n2);
}


/*****************************************************************************
 *                              Fonctions d'E/S
 *****************************************************************************/

/*
 * Rôle : lit un BigNat sur le flot fd
 */
void flire(BigNat n, FILE *fd) {
  BigNat aux;
  int i=0, c;
  while (isspace(c=fgetc(fd)) || c=='0');
  // c 1er chiffre du BigNat à lire
  assert(isdigit(c));
  do {
    aux[i++]=c-'0';
  } while (isdigit(c=fgetc(fd)));
  // recopie aux dans n en sens inverse
  int k=0;
  for (int j=i-1; j>=0; j--, k++)
    n[k] = aux[j];
  // compléter avec des 0
  while (i<LGMAX) n[i++]=0;
}

/*
```

```
 * Rôle : lit un BigNat sur l'E/S
 */
void lire(BigNat n) {
  flire(n, stdin);
}

void fecrire(const BigNat n, FILE *fd) {
  int i=LGMAX-1;
  while (n[i]==0 && i>=0) i--;
  if (i<0) fprintf(fd, "0");
  else
    while (i>=0)
      fprintf(fd, "%ld", n[i--]);
}

void fecrireLn(const BigNat n, FILE *fd) {
  fecrire(n, fd);
  fprintf(fd, "\n");
}

void ecrire(const BigNat n) {
  fecrire(n, stdout);
}

void ecrireLn(const BigNat n) {
  fecrireLn(n, stdout);}


void erreur(char msg[]) {
  fprintf(stderr, "Erreur : %s\n", msg);
  exit(EXIT_FAILURE);
}

/*****************************************************************************
 *                          Fonctions d'arithmétique
 *****************************************************************************/

/*
 * Rôle : n3 = n2 + n1
 */
void plus(const BigNat n1, const BigNat n2, BigNat n3) {
  int retenue=0;
  for (int i=0; i<LGMAX; i++) {
    n3[i] = n1[i] + n2[i] + retenue;
    if (n3[i]>=BASE) {
      n3[i]-=BASE;
      retenue=1;
    }
    else
      retenue=0;
  }
  if (retenue==1) erreur("Overflow");
}

/*
 * Rôle : n=n+1
 */
void incr1(BigNat n) {
  plus(n, BIG_UN, n);
}

/*
 * Antécédent : n2>=n1
 * Rôle : n3 = n2 - n1
 */
void moins(const BigNat n1, const BigNat n2, BigNat n3) {
```



```
  assert(supEgal(n1, n2));
  int retenue=0;
  for (int i=0; i<LGMAX; i++) {
    n3[i] = n1[i]-n2[i]-retenue;
    if (n3[i]<0) {
      // => retenue
      n3[i]+=BASE;
      retenue=1;
    }
    else
      retenue=0;
  }
}

/*
 * Rôle : n=n-1
 */
void decr1(BigNat n) {
  moins(n, BIG_UN, n);
}

/*
 * antécédent : a>=0, b>0
 * rôle : calcule le quotientet le reste de la division
 *        euclidienne de a par b
 */
void divEuclide(const BigNat a, const BigNat b, BigNat q, BigNat r) {
  BigNat x, y;
  initBn(x, a);
  initBn(y, b);
  // méthode : par soustractions successives
  initBn(q, BIG_ZERO);
  initBn(r, x);

  // x = q*y + r
  while (supEgal(r,y)) {
    // x = q*y + r et r>=y
    // x = (q+1)*b + r-y  et r>=y
    moins(r, y, r);
    // a = (q+1)*b+r
    incr1(q);
    // x = q*y + r
  }
  // x = q*y+r et r<y
}


/*
 * rôle : calcule n3 = n1 % n2
 */
void divi(const BigNat n1, const BigNat n2, BigNat n3) {
  BigNat r; // unused
  divEuclide(n1, n2, n3, r);
}

/*
 * rôle : calcule n3 = n1 % n2
 */
void modulo(const BigNat n1, const BigNat n2, BigNat n3) {
  BigNat q; // unused
  divEuclide(n1, n2, q, n3);
}

/*
 * rôle : calcule n2 = n1 * e
 */
```

```
void multE(const int e, const BigNat n1, BigNat n2) {
  BigNat aux;
  initBn(aux, BIG_ZERO);
  for (int i=0; i<e; i++)
    plus(n1, aux, aux);
  initBn(n2, aux);
}

/*
 * rôle : calcule n3 = n1 * n2
 */
void mult(const BigNat n1, const BigNat n2, BigNat n3) {
  BigNat x, y;
  initBn(x, n1);
  initBn(y, n2);
  initBn(n3, BIG_ZERO);
  int p=1;
  for (int i=0; i<LGMAX; i++) {
    BigNat aux;
    initBn(aux, BIG_ZERO);
    // faire le produit si le chiffre est différent de 0
    if (x[i]!=0) {
      multE(x[i]*p, y, aux);
      plus(n3, aux, n3);
    }
    p=p*BASE;
  }
}

/*
 * Antécédent : n>=0
 * role : calcule fact = n!
 */
void factorielle(const int n, BigNat fact) {
  initE(fact, 1);
  for (int i=1; i<=n; i++)
    multE(i, fact, fact);
}

int main(void) {
  BigNat fact;

  factorielle(300, fact);
  ecrireLn(fact);


  return EXIT_SUCCESS;
}
```