```c
/*
 * Ce programme est une calculette qui fournit des opérations sur des
 * nombres au format binaire. Ces nombres binaires représentent des
 * entiers longs non signés. Ils sont formés de 0 et de 1 placés dans
 * un tableau de NB_BITS. Le bit de poids faible est à l'indice 0.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>

#define OCTET_BITS 8 // nb de bits dans un octet

// le nombre de bits est indépendant de la réprésentation
// des unsigned long int.
#define NB_BITS (sizeof(unsigned long int)*OCTET_BITS)

// définition du type binaire
typedef short binaire[NB_BITS];

// la constant binaire qui représente 0
const binaire ZERO = {};

typedef enum {
              HELP='h', LIRE='l', EXIT='q', PRINT='p',
              ADD='+', MOINS='-', MULT='*', DIVI='/',
              CONV='c', SHIFTG='g', SHIFTD='d'
} CMDS;

/**
 *  rôle :renvoie le prochain caractère lu sur l'E/S
 */
int readNextChar(void) {
  int rep;
  while ((rep=getchar())=='\n');
  if (rep==EOF) exit(EXIT_FAILURE);
  return rep;
}

/**
 *  rôle : passe à la ligne suivante sur l'E/S
 */
void skipNextLine(char c) {
  while (c!='\n' && c!=EOF) c=getchar();
  if (c==EOF) exit(EXIT_FAILURE);
}

/*
 * Antécédent : le prochain caractère à lire sur l'E/S est un
 *              espace, un 0 ou un 1
 * Rôle : lit sur l'E/S une suite de 0 et 1 pour former
 *        le nombre binaire b
 */
void lireBinaire(binaire b) {
  // sauter les espaces de tête
  int c;
  while (isspace(c=getchar())) ;
  // c == 0, 1 ou autre
  binaire baux = {};
  int i=0;
  while (c=='0' || c=='1') {
    if (i!=NB_BITS)
      baux[i++] = c-'0';
    //
    c=getchar();
  }
  //
  skipNextLine(c);
  //
  // copier baux dans b
  int j=0;
  for (i-- ; i>=0; i--, j++)
    b[j] = baux[i];
  // compléter avec des 0
  for ( ; j<NB_BITS ; j++)
    b[j] = 0;
}

/*
 * Rôle : renvoie la conversion en entier long non signé
 *        du binaire b
 */
unsigned long int binaireToLongInt(const binaire b) {
  // sauter tous les 0 à partir du bit de poids fort
  int i=NB_BITS;
  do i--; while (i>=0 && b[i]==0);
  //
  if (i<0)
    // b ne contient que des 0
    return 0;
  // b[i] == 1
  unsigned int n = 0;
  for (; i>=0; i--)
    n = (n<<1) + b[i];
  // n valeur décimale de suite binaire contenue dans b
  return n;
}

/*
 * Rôle : écrit le binaire b sur la S/S (bit de poids faible à droite)
 */
void printBinaire(const binaire b) {
  int i=NB_BITS;
  do i--; while (i>=0 && b[i]==0);
  //
  if (i<0) printf("0");
  else
    for (; i>=0; i--)
      printf("%1d", b[i]);
}

/*
 * Rôle : écrit le binaire b sur la S/S (bit de poids faibl à droite)
 *        suivi d'un passage à la ligne
 */
void printLnBinaire(const binaire b) {
  printBinaire(b);
  printf("\n");
}

/*
 *  Rôle : convertit l'entier n en binaire b
 */
void longIntToBinaire(unsigned long int n, binaire b) {
  int i=0;
  // décomposer n en base 2
  do
    // prendre le dernier bit de n
    b[i++] = n&1;
  while ((n>>=1)!=0);
  // ajouter les 0 de fin
```

```
  while (i<NB_BITS) b[i++]=0;
}

/*
 * Antécédent : b2 nombre binaire initialisé
 * Conséquent : b1 = b2
 */
void affecter(binaire b1, const binaire b2) {
  for (int i=0; i<NB_BITS; i++)
    b1[i]=b2[i];
}

/*
 * Conséquent : b3 = b1 & b2
 */
void et(const binaire b1, const binaire b2, binaire b3) {
  for (int i=0; i<NB_BITS; i++)
    b3[i]=b1[i]&b2[i];
  //
}

/*
 * Conséquent : b3 = b1 | b2
 */
void ou(const binaire b1, const binaire b2, binaire b3) {
  for (int i=0; i<NB_BITS; i++)
    b3[i]=b1[i]|b2[i];
  //
}

/*
 * Conséquent : b3 = b1 ^ b2
 */
void xou(const binaire b1, const binaire b2, binaire b3) {
  for (int i=0; i<NB_BITS; i++)
    b3[i]=b1[i]^b2[i];
  //
}

/*
 * Conséquent : b3 = b1 + b2
 */
void plus(const binaire b1, const binaire b2, binaire b3) {
  longIntToBinaire(binaireToLongInt(b1)+binaireToLongInt(b2), b3);
}

/*
 * Antécédent : b1>= b2
 * Conséquent : b3 = b1 - b2
 */
void moins(const binaire b1, const binaire b2, binaire b3) {
  unsigned long int x = binaireToLongInt(b1);
  unsigned long int y = binaireToLongInt(b2);
  assert(x>=y);
  longIntToBinaire(x-y, b3);
}

/*
 * Conséquent : b3 = b1 * b2
 */
void mult(const binaire b1, const binaire b2, binaire b3) {
  longIntToBinaire(binaireToLongInt(b1)-binaireToLongInt(b2), b3);
}

/*
 * Antécédent : b2 !=0
```



```
 * Conséquent : b3 = b1 / b2
 */
void divi(const binaire b1, const binaire b2, binaire b3) {
  unsigned long int n = binaireToLongInt(b2);
  // le dénominateur doit être différent de 0
  assert(n!=0);
  longIntToBinaire(binaireToLongInt(b1)/n, b3);
}

/*
 *  Antécédent : 0 <= n <= NB_BITS
 *  Rôle : produit un décalage ouvert de n bits
 *         vers la gauche dans b (i.e multiplication par n^2)
 */
void decalageGauche(binaire b, const int n) {
  assert(n>=0 && n<=NB_BITS);
  // décaler de n positions vers la droite
  for (int i=NB_BITS-1-n; i>=0; i--)
    b[i+n] = b[i];
  // mettre 0 dans les n premiers éléments
  for (int i=0; i<n; i++)
    b[i] = 0;
}

/*
 *  Antécédent : 0 <= n <= NB_BITS
 *  Rôle : produit un décalage ouvert de n bits
 *         vers la droite de b (i.e division par n^2)
 */
void decalageDroit(binaire b, const int n) {
  assert(n>=0 && n<=NB_BITS);
  // décler de n bits
  for (int i=0; i<NB_BITS-n; i++)
    b[i] = b[i+n];
  // mettre 0 dans les n derniers éléments
   for (int i=0; i<n; i++)
     b[NB_BITS-i-1] = 0;
}

/*****************************************************************
 *
 *   les commandes
 *
 *****************************************************************/

/*
 * la commande help
 */
void cmdHelp(void) {
  printf("h : cette aide en ligne\n");
  printf("l : lire un nb binaire\n");
  printf("p : afficher le nombre binaire\n");
  printf("c : convertir le nombre binaire\n");
  printf("g n : décalage de n bits vers la gauche\n");
  printf("d n : décalage de n bits vers la droite\n");
  printf("+  : addition de 2 binaires\n");
  printf("-  : soustraction de 2 binaires\n");
  printf("*  : produit de 2 binaires\n");
  printf("/  : division de 2 binaires\n");
  printf("q : quit\n");
  printf("-----------------------\n");
}

/*
 * la commande de fin de programme
 */
```

```c
void cmdExit(void) {
  printf("Voulez vous quitter l'application [o/n] : ");
  if (readNextChar()=='o') exit(EXIT_SUCCESS);
}

/*
 * la commande de lecture d'un nombre binaire
 */
void cmdLire(binaire b) {
  lireBinaire(b);
}

/*
 * la commande d'écriture d'un nombre binaire
 */
void cmdPrint(binaire b) {
  printBinaire(b);
  printf(" = %lu\n", binaireToLongInt(b));
}

/*
 * la commande de conversion d'un nombre binaire
 */
void cmdConvertir(binaire b) {
  printf("%lu\n", binaireToLongInt(b));
}

/*
 * la commande de décalage d'un nombre binaire
 */
void cmdDecalage(binaire b, CMDS ou) {
  int dec;
  scanf("%d", &dec);
  ou == SHIFTG ? decalageGauche(b,dec) : decalageDroit(b,dec);
}

/*
 * la commande d'opération arithmétique (+, -, * et /)
 */
void cmdOperation(binaire b, char op) {
  binaire b1, b2;
  printf("b1 = ");  lireBinaire(b1);
  printf("b2 = ");  lireBinaire(b2);
  switch (op) {
  case '+' :   printf("b1 + b2 = "); plus(b1, b2, b); break;
  case '-' :   printf("b1 - b2 = "); moins(b1, b2, b); break;
  case '*' :   printf("b1 * b2 = "); mult(b1, b2, b); break;
  case '/' :   printf("b1 / b2 = "); divi(b1, b2, b); break;
  }
  printLnBinaire(b);
}

/*
 * rôle : écrit sur la sortie d'erreur standard le message msg
 */
void erreur(char msg []) {
  fprintf(stderr, "%s\n", msg);
}

int main(void) {

  binaire b;
  affecter(b, ZERO);

  while (1) {
    printf("> ");
```



```c
    switch (readNextChar()) {
    case HELP    : cmdHelp(); break;
    case LIRE    : cmdLire(b); break;
    case PRINT   : cmdPrint(b); break;
    case CONV    : cmdConvertir(b); break;
    case SHIFTG  : cmdDecalage(b,SHIFTG); break;
    case SHIFTD  : cmdDecalage(b,SHIFTD); break;
    case ADD     : cmdOperation(b,ADD); break;
    case MOINS   : cmdOperation(b,MOINS); break;
    case MULT    : cmdOperation(b,MULT); break;
    case DIVI    : cmdOperation(b,DIVI); break;
    case EXIT    : cmdExit(); break;
    default : erreur("commande inconnue. h (help)");
    }
  }

  return EXIT_SUCCESS;
}
```