Université de Xidian
Polytech'Sophia

Université de Xidian
2021–2022

Examen de Langage C *(V. Granet)*

Durée : 1h30
Aucun document autorisé
Mobiles interdits

Question 1 : Nombres parfaits

▶ 1. Écrivez la fonction `sommeDesDiviseurs` qui renvoie la somme des diviseurs stricts d'un nombre entier positif `n` (> 0) passé en paramètre. Par exemple, la somme des diviseurs stricts de 78 est égale à 90. Notez que 78 n'est pas diviseur strict de lui-même.

```
/*
 * Antécédent : n>0
 * Rôle : renvoie la somme des diviseurs stricts de n
 */
int sommeDesDiviseurs(const int n) {
  int s=0;
  // inutile de tester les diviseurs > à n/2 !
  for (int i=n/2 ; i>0 ; i--)
    if ((n%i)==0)
      // i est un diviseur
      s+=i;
  //
  return s;
}
```

.....

▶ 2. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs stricts. Par exemple, $28 = 1 + 2 + 4 + 7 + 14$. Écrivez la fonction booléenne `estParfait` qui teste si son paramètre (> 0) est un nombre parfait ou non.

```
/*
 * antécédent : n>0
 * rôle : renvoie 1 si n est un nombre parfait, 0, sinon
 */
int estParfait(const int n) {
  return sommeDesDiviseurs(n) == n;
}
```

.....

▶ 3. Écrivez la fonction `main` qui lit sur l'entrée standard un entier `n` (>0) et qui écrit sur la sortie standard tous les nombres parfaits sur l'intervalle $[1\,;n]$.

```
int main(void) {
  int n;
  scanf("%d", &n);
  assert(n>0);
  for (int i=1; i<=n; i++)
    if (estParfait(i))
      printf("%d\n", i);
  //
  return EXIT_SUCCESS;
}
```

.....

Question 2. Nombres binaires

On représente un entier long non signé (**unsigned long int**) sous forme binaire à l'aide d'un tableau d'entiers courts (**short**) contenant des 0 et des 1. Le bit de poids faible (2^0) est à l'indice 0, et le bit de poids fort ($2^{\texttt{NB\_BITS}-1}$) est à l'indice `NB_BITS-1`. Par exemple, si `NB_BITS` est égal à 8, le tableau qui contient `{0, 1, 1, 0, 1, 0, 0, 0 }` représente l'entier décimal non signé 22. On définira le type `binaire`, comme suit :

```
typedef short binaire[NB_BITS];
```

▶ 4. À l'aide d'un `define`, définissez la constante `NB_BITS` égale au nombre d'éléments du tableau nécessaires à la représentation d'un entier long non signé (**unsigned long int**).

```
#define NB_BITS (sizeof(unsigned long int)*8)
```

.....

▶ 5. Écrivez la fonction `binaireToLongInt` qui calcule et renvoie la valeur décimale d'un nombre binaire. Vous ne ferez **aucune** évaluation à la puissance. Cette fonction possède l'en-tête suivant :

```
unsigned int binaireToLongInt(const binaire b) {
```

```
/*
 * Antécédent : b nombre binaire
 * Rôle : renvoie la valeur entière décimale non signée du
 *        binaire b
 */
unsigned long int binaireToLongInt(const binaire b) {
  unsigned int n = 0;
  for (int i=NB_BITS-1; i>=0; i--)
    n = (n<<1) + b[i];
  // n valeur décimale de suite binaire contenue dans bits
  return n;
}
```

.....

▶ 6. Écrivez la procédure `et` qui renvoie la conjonction de deux nombres binaires représentés par des tableaux de même taille.

```
/*
 * Antécédent : b1 et b2 nombres binaires
 * 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];
  //
}
```

.....

Question 3. Matrices

▶ 7. Écrivez la procédure `printSubMat` qui écrit sur la sortie standard une sous-matrice issue d'une matrice de réels **double**. Cette procédure possède 7 paramètres :

- une matrice `mat`, un tableau de réels **double** à 2 dimensions ;
- 2 entiers `m` et `n` qui correspondent aux nombres de lignes et de colonnes ;
- 4 entiers `i1`, `i2`, `j1` et `j2`, avec `i1`$\geqslant$`0` et `i1`$<$`m` et `i2`$\geqslant$`0` et `i2`$<$`m` et `j1`$\geqslant$`0` et `j1`$<$`n` et `j2`$\geqslant$`0` et `j2`$<$`n` et `i1`$\leqslant$`i2` et `j1`$\leqslant$`j2` (vous vérifierez la validité des indices) ;

La procédure `printSubMat` écrit sur la sortie standard la sous-matrice de `mat` formée des lignes `i1` à `i2` et des colonnes `j1` à `j2`.

```
/*
 * rôle : écrit sur la sortie standard la sous-matrice issue de mat
 *        formée des lignes de i1 à i2 et des colonnes de j1 à j2
 */
void  printSubMat(const int m, const int n, const double mat[][n],
                  const int i1, const int i2, const int j1, const int j2)
  {
    assert(i1>=0 &&  i1<m && i2>=0 &&  i2<m);
    assert(j1>=0 &&  j1<n && j2>=0 &&  j2<n);
    assert(i1<=i2 && j1<=j2);
    for (int i=i1; i<=i2; i++)  {
      for (int j=j1; j<=j2; j++)
        printf("%5.1f", mat[i][j]);
      //
      printf("\n");
    }
  }
```

.....

▶ 8. Soit une matrice $m \times n$ qui contient des entiers (**int**). Écrivez la fonction booléenne `creuse` qui renvoie *vrai* si plus de la moitié des entiers contenus dans la matrice sont égaux à 0, et *faux* sinon. Cette fonction possède l'en-tête suivant :

```
int creuse(const int m, const int n, const double mat[][n])
```

```
/*
 * rôle : teste si plus de la moitié des éléments de la matrice
 *        d'entiers mat sont des zéros. renvoie 1 si oui, et 0 sinon
 */
int creuse(const int m, const int n, const double mat[][n]) {
  int moitie = (n+m)/2;
  int nbZero=0;
  for (int i=0; i<m; i++)
    for (int j=0; j<n; j++)
      if (mat[i][j]==0)
        // on a un zéro
        if (nbZero++==moitie)
          // on a plus de la moitié de 0
          return 1;
  // moins de la moitié
  return 0;
}
```

.....