Université de Nice-Sophia Antipolis
Polytech'Sophia

Université de Xidian
2023–2024

Examen de Langage C

Durée : 1h30
Aucun document autorisé – Mobiles interdits

Notez que les affirmations (antécédents, conséquents, rôles, et invariants) dans vos codes C entreront pour partie dans la note finale.

On souhaite représenter l'ensemble $\mathbb{Q}$ des nombres rationnels : $\mathbb{Q} = \{\frac{m}{n} | (m,n) \in \mathbb{Z} \times \mathbb{Z}^*\}$.

▶ 1. Avec **typedef**, déclarez le type structuré `rationnel` pour représenter les nombres rationnels de l'ensemble $\mathbb{Q}$.

```
typedef struct {
  int num;
  int dem;
} rationnel;
```

.....

▶ 2. Écrivez la fonction `newRationnel` qui prend en paramètre deux entiers `num` et `dem` et renvoie un pointeur sur un rationnel créé et initialisé avec ces deux entiers. L'en-tête de cette fonction est :

```
rationnel *newRationnel(const int num, const int dem) {
```

```
/*
 * Antécédent : dem!=0
 * Rôle : renvoie le rationnel num/dem
 *
 */
rationnel *newRationnel(const int num, const int dem) {
  assert(dem!=0);
  rationnel *r = malloc(sizeof(rationnel));
  r->num = num;
  r->dem = dem;
  return r;
}
```

.....

▶ 3. Écrivez la procédure `écrireRationnel` qui écrit sur la sortie standard au format *num/dem* un rationnel. Le paramètre de cette fonction est un pointeur sur le rationnel à écrire.

```
/*
 * Antécédent : r!=NULL
 * Rôle : écrit sur la sortie standard le rationnel *r au format
 *         num/dem
 */
void écrireRationnel(const rationnel *r) {
  assert(r!=NULL);
  printf("%d/%d", r->num, r->dem);
}

/*
 * Antécédent : r!=NULL
 * Rôle : écrit sur la sortie standard le rationnel *r au format
 *         num/dem suivi d'un newline
 */
void écrirelnRationnel(const rationnel *r) {
  écrireRationnel(r);
  printf("\n");
}
```

.....

▶ 4. Écrivez la fonction `simplifier` qui renvoie un pointeur sur un rationnel simplifié (si possible) d'un rationnel passé en paramètre. Par exemple, $\frac{21}{70}$ est simplifié en $\frac{3}{10}$, le rationnel $-\frac{42}{6}$ en $-\frac{7}{1}$; et $\frac{23}{10}$ reste inchangé. Cette fonction possède l'en-tête suivant :

```
rationnel *simplifier(const rationnel *r) {
```

```
/*
 * Antécédent : a>=0 et b>=0
 * Rôle : renvoie le pgcd des entiers  a et b
 *
 */
int pgcd(int a, int b) {
  while (a!=b) {
    // pgcd(a,b) = pgcd(a-b,b) et a>b
    // = pgcd(a,b-a) et a<b
    if (a>b)
      // pgcd(a,b) = pgcd(a-b,b) et a>b
      a-=b;
    else
      b-=a;
    // pgcd(a,b) = pgcd(a,b-a) et a<b
  }
  return a;
}

/*
 * Rôle : réduit le numérateur et le dénominateur
 *         du rationnel r par leur pgcd
 */
rationnel *réduire(const rationnel *r) {
  // calculer le pgcd du numérateur et du dénominateur
  int div = pgcd(abs(r->num), abs(r->dem));
  return newRationnel(r->num/div, r->dem/div);
}


/*
 * Antécédent : r!=NULL
```

```
 * Rôle : renvoie la simplification du rationnel r
 */
rationnel *simplifier(const rationnel *r) {
  assert(r!=NULL);
  if (r->num == 0) return newRationnel(0,1);
  if (r->num == r->dem) return newRationnel(1,1);
  if (r->num == -r->dem) return newRationnel(-1,1);
  // chercher les diviseurs et réduire
  return réduire(r);
}
```

.....

► 5. Écrivez la fonction `add` qui prend en paramètre deux pointeurs sur rationnel. La fonction renvoie un pointeur sur un rationnel qui est l'addition des deux rationnels passés en paramètres. Le résultat doit être simplifié.

```
/*
 * Antécédent : r1!=NULL && r2!=NULL
 * Rôle : renvoie *r1 + *r2
 */
rationnel *add(const rationnel *r1, const rationnel *r2) {
  assert(r1!=NULL && r2!=NULL);
  return simplifier(r1->dem == r2->dem ?
                    newRationnel(r1->num + r2->num, r1->dem)
                    : newRationnel(r1->num*r2->dem + r2->num*r1->dem, r1->dem*r2->dem));
}
```

.....

► 6. Écrivez la fonction `somme` qui prend en paramètre une chaîne de caractères qui est le nom d'un fichier de texte. Le fichier contient une suite de rationnels au format *n/d*. La fonction lit tous les rationnels contenus dans le fichier et renvoie leur somme. Pour écrire cette fonction, vous devez utiliser les fonctions des questions précédentes. Vous devrez aussi vérifier la bonne ouverture du fichier. Cette fonction possède l'en-tête suivant :

```
rationnel *somme(const char *fname) {
```

```
/*
 * Antécédent : le fichier de texte de nom fname contient une
 *              suite de rationnels au format n/d sans erreur
 * Rôle : renvoie la somme des rationnels contenus dans fname
 */
rationnel *somme(const char *fname) {
  FILE *fd;
  if ((fd = fopen(fname, "r")) == NULL) {
    perror(fname);
    exit(errno);
  }
  // le fichier fname est ouvert en lecture
  rationnel *r = newRationnel(0,1); // le rationnel courant
  rationnel  *som = newRationnel(0,1);
  // lire et sommer tous les rationnels
  while (fscanf(fd, "%d/%d", &(r->num), &(r->dem))>0)
```



```
    som = add(som, r);
  //
  // libérer r
  free(r);
  // fermer le fichier
  fclose(fd);
  return som;
}
```

.....

► 7. Écrivez le programme `sommeRat` qui prend le nom d'un fichier de texte en paramètre programme. Le fichier contient des rationnels au format *n/d*. Le programme doit écrire sur la sortie standard la somme des rationnels contenu dans le fichier. Vous vérifierez la validité du nombre de paramètres programme. Par exemple, si le fichier `fr` contient :

```
11/4 -3/12
4/8
```

l'exécution du programme donnera :

```
$ ./sommeRat fr
3/1
```

```
int main(int argc, char *argv[]) {
  //
  if (argc!=2) {
    fprintf(stderr, "Usage: sommeRat file\n");
    return EXIT_FAILURE;
  }
  // Ok : un seul parmètre programme => calculer et afficher
  // la somme des rationnels qu'il contient
  écrirelnRationnel(somme(argv[1]));

  return EXIT_SUCCESS;
}
```

.....