Polytech-Sophia
Elec3

10 décembre 2020
Durée : 1 mois

Correction projet C : Ensemble de nombres premiers

```c
/*
 * Elec3 - Projet n. 1  - auteur Vincent Granet (vg@unice.fr)
 *
 * 1er partie : définition d'une type ensemble pour représenter des ensembles
 *              d'entiers naturels
 *
 * 2ème partie : mise ne oeuvre de l'algorithme du cribe d'Ératosthène à l'aide
 *               du type Ensemble défini précédemment.
 */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

/*****************************************************
 *
 * Type boolean
 *
 *****************************************************/

#define true 1
#define false 1
typedef char boolean;


/*****************************************************
 *
 * Type Ensemble
 *
 *****************************************************/

#define CAPACITEMAX 1000
typedef boolean ensemble[CAPACITEMAX+2];
// les éléments de l'ensemble sont des valeurs booléennes (vrai/faux)
// pour dénoter la présence ou l'absence d'un élément

// Rôle : renvoie la capacité de l'ensemble e
int getCapacite(const ensemble e) {
   return e[1];
}

// Rôle : met à la valeur n la capacité de l'ensemble e
void setCapacite(int n, ensemble e) {
   e[1]=n;
}

// Rôle : renvoie le cardinal de l'ensemble e
int getCardinal(const ensemble e) {
   return e[0];
}
```



```c
// Rôle : met à la valeur n le cardinal de l'ensemble e
void setCardinal(int n, ensemble e) {
   e[0]=n;
}

// Conséquent : e = ensemble vide
void vide(ensemble e) {
  const int elemMax = getCapacite(e)+2;
  for (int i=2; i<elemMax; i++) e[i]=false;
  setCardinal(0,e);
}

// Antécédent : 0<n<=CAPACITEMAX
// Rôle : initialise l'ensemble e à vide de capacité max égale à n
void initEns(const int n, ensemble e) {
  assert(n>0 && n<=CAPACITEMAX);
  setCapacite(n,e);
  vide(e);
}

// Antécédent : 0<=n<CAPACITEMAX
// Rôle : fabrique l'ensemble e formé du seul élément n
// de capacité max CAPACITEMAX
void singleton(const int n, ensemble e) {
  assert(n>=0 && n<CAPACITEMAX);
  e[n+2]=true;
  setCardinal(1,e);
  setCapacite(CAPACITEMAX,e);
}

// Conséquent : e = ensemble plein
void plein(ensemble e) {
  const int elemMax = getCapacite(e)+2;
  for (int i=2; i<elemMax; i++) e[i]=true;
  setCardinal(getCapacite(e),e);
}

// Rôle : renvoie vrai si e est vide et faux sinon
boolean estVide(const ensemble e) {
  return getCardinal(e)==0;
}

// Rôle : affecte e2 à e1 : e1 = e2
void affecter(ensemble e1, const ensemble e2) {
  assert(getCapacite(e1) == getCapacite(e2));
  const int elemMax = getCapacite(e1)+2;
  for (int i=0; i<elemMax; i++) e1[i] = e2[i];
}

// Rôle : renvoie e1 == e2
boolean egal(const ensemble e1, const ensemble e2) {
  assert(getCapacite(e1) == getCapacite(e2));
  const int elemMax = getCapacite(e1)+2;
  int i=1;
  // note : en partant de 1, on vérifie d'abord que
  // les 2 ensemble ont la même cardinalité
  while (i++<elemMax)
    if (e1[i]!=e2[i]) return 0;
```

```
  return 1;
}

// Rôle : renvoie vrai si n appartient à e et faux sinon
boolean appartient(const int n, const ensemble e) {
  if (n<0 || n>=getCapacite(e)) return 0;
  return e[n+2];
}

// Rôle : ajoute n à l'ensemble e
void ajouter(const int n, ensemble e) {
  assert(n>=0 && n<getCapacite(e));
  if (!e[n+2]) {
    e[n+2] = 1;
    setCardinal(getCardinal(e)+1, e);
  }
}

// Rôle : enlève n à l'ensemble e
void enlever(const int n, ensemble e) {
  assert(n>=0 && n<getCapacite(e));
  if (e[n+2]) {
    e[n+2] = false;
    setCardinal(getCardinal(e)-1, e);
  }
}

// Rôle : écrit l'ensemble e sur la s/s
void printEns(const ensemble e) {
  const int elemMax = getCapacite(e)+2;
  printf("[ ");
  for (int i=2; i<elemMax; i++)
    if (e[i])
      printf("%d ", i-2);
   printf("]");
}

// Rôle : écrit l'ensemble e sur la s/s avec un passage à la ligne
void printlnEns(const ensemble e) {
  printEns(e); printf("\n");
}

// Rôle : e3 = e1 ∩ e2
void intersection(const ensemble e1, const ensemble e2, ensemble e3) {
  assert(getCapacite(e1) == getCapacite(e2));
  initEns(getCapacite(e1), e3);
  int cardinal=0;
  const int elemMax = getCapacite(e1)+2;
  for (int i=2; i<elemMax; i++)
     cardinal += e3[i] = e1[i] & e2[i];
  //
  setCardinal(cardinal, e3);
}

// Rôle : e e3 = e1 ∪ e2
void Union(const ensemble e1, const ensemble e2, ensemble e3) {
  assert(getCapacite(e1) == getCapacite(e2));
  initEns(getCapacite(e1), e3);
```



```
  int cardinal=0;
  const int elemMax = getCapacite(e1)+2;
  for (int i=2; i<elemMax; i++)
    cardinal += e3[i] = e1[i] | e2[i];
  //
  setCardinal(cardinal, e3);
}

// Rôle : e3 = e1 \e2
// e3 contient les éléments qui appartiennent à e1 mais pas à e2
// e1 \e2 = { x / x ∈ e1 et x ∉ e2 }
void difference(const ensemble e1, const ensemble e2, ensemble e3) {
  assert(getCapacite(e1) == getCapacite(e2));
  initEns(getCapacite(e1), e3);
  int cardinal=0;
  const int elemMax = getCapacite(e1)+2;
  for (int i=2; i<elemMax; i++)
    cardinal += e3[i] = (e1[i] & !e2[i]);
  //
  setCardinal(cardinal, e3);
}

// Rôle : e3 = e1 Δ e2
// e3 contient les éléments qui appartiennent à e1 ou à e2 mais pas les 2 à la fois
// e1 Δ e2 = {x|(x ∈ e1 et x ∉ e2) ou (x ∈ e2 et x ∉ e1)} = (e1 ∪ e2) (e1 ∩ e2)
void diffSym(const ensemble e1, const ensemble e2, ensemble e3) {
  assert(getCapacite(e1) == getCapacite(e2));
  initEns(getCapacite(e1), e3);
  int cardinal=0;
  const int elemMax = getCapacite(e1)+2;
  for (int i=2; i<elemMax; i++)
    cardinal += e3[i] = (e1[i] ^ e2[i]);
  //
  setCardinal(cardinal, e3);
}

// Rôle : e2 = complementaire de e1
void complementaire(const ensemble e1, ensemble e2) {
  initEns(getCapacite(e1), e2);
  plein(e2);
  int cardinal=0;
  const int elemMax = getCapacite(e1)+2;
  for (int i=2; i<elemMax; i++)
    cardinal += e2[i] = !e1[i];
  //
  setCardinal(cardinal, e2);
}
```

```
/*****************************************************
 *
 * Crible  d'Ératosthène
 *
 *****************************************************/

/*
 * Rôle : remplit l'ensemble r avec tous les
 *         nombres premiers compris entre 2 et n-1
 *         Algorithme utilisé : crible d'Ératosthène
 */
void eratosthene(const int n, ensemble r) {
  ensemble crible;
  initEns(n, crible);
  initEns(n, r);
  // initialise le crible à l'ensemble plein
  plein(crible); vide(r);
  // enlever 0 et 1 qui ne sont pas premiers
  enlever(0,crible); enlever(1, crible);
  int p=2;
  do {
    // Invariant : p est le plus petit élément du crible
    // il est PREMIER ⇒ on l'ajoute dans r
    ajouter(p, r);
    // retirer p et tous ses multiples
    // ne pas utiliser l'opérateur modulo !!!
    int m=p;
    do {
      enlever(m,crible);
      m+=p;
    } while (m<n);
    // on recherche le minimum du crible à partir de p
    while (p<n && !appartient(p, crible))
      p++;
  } while (!estVide(crible));
  // le crible est vide : tous les nombres premiers
  // ont été enregistrés dans l'ensemble résultat r
}
```



```
/*****************************************************
 *
 * main pour tester le type Ensemble et le crible  d'Ératosthène
 *
 *****************************************************/

int main(void) {
  //
  ensemble s1, s2;
  singleton(10,s1);
  printf("s1 = "); printlnEns(s1);
  singleton(4,s2);
  printf("s2 = "); printlnEns(s2);
  printf("------------------\n");
  // 1
  ensemble e1, e2;
  initEns(50, e1);  initEns(50, e2);
  // 2
  ajouter(2, e1); ajouter(19, e1); ajouter(31,e1);
  ajouter(10, e2); ajouter(19, e2); ajouter(34, e2);
  // 3
  printlnEns(e1);
  printlnEns(e2);
  // 4
  printf("%d\n", appartient(10, e1));
  printf("%d\n", appartient(19, e1));
  // 5
  ajouter(33, e1);
  ajouter(33, e2);
  // 6
  printf("e1 = ");printlnEns(e1);
  printf("e2 = ");printlnEns(e2);
  // 7
  ensemble e3;
  intersection(e1, e2, e3);
  printf("e3 = e1 ∩ e2 = "); printlnEns(e3);
  // 8
  ensemble e4;
  Union(e1, e2, e4);
  printf("e4 = e1 ∪ e2 = "); printlnEns(e4);
  // 8
  ensemble e5;
  difference(e1, e2, e5);
  printf("e5 = e1 - e2 = "); printlnEns(e5);
  // 9
  ensemble e6;
  complementaire(e1, e6);
  printf("e6 = ~e1 = "); printlnEns(e6);
  // 10
  printf("------------------\n");
  ensemble e7;
  printf("e1 = ");printlnEns(e1);
  printf("e2 = ");printlnEns(e2);
  diffSym(e1, e2, e7);
  printf("e7 = e1 Δ e2 = "); printlnEns(e7);
  // De morgan
  printf(" ---> De Morgan\n");
```

```
  ensemble mc, mr;
  complementaire(e3, mc); // mc complementaire e1 inter e2
  ensemble ce1, ce2;
  complementaire(e1, ce1);
  complementaire(e2, ce2);
  Union(ce1, ce2, mr);
  printf("mc = mr = %d\n", egal(mc, mr));
  // affectation
  printf(" ---> affectation\n");
  affecter(e1, e2);
  printf("e1 = e2 = %d\n", egal(e1, e2));
  printf(" ---> Ératosthène\n");
  int n;
  printf("n (<=%d) = ", CAPACITEMAX);
  scanf("%d", &n);
  //
  if (n<2 || n>CAPACITEMAX) {
    fprintf(stderr, "Erreur : 2 <= n <= %d\n", CAPACITEMAX);
    return EXIT_FAILURE;
  }
  // ok calculer les nombres premiers sur [2;n]
  ensemble p;
  eratosthene(n, p);
  printlnEns(p);

  return EXIT_SUCCESS;
}
```