DS2 MP2I

Avant-propos

Avertissement Les réponses sont à écrire exclusivement sur le document réponse fourni. Le nom de l'étudiant.e doit figurer sur toutes les feuilles. Les pages doivent être rendues dans l'ordre et non écornées. Tout passage peu lisible dans une copie sera ignoré au moment de la correction même si la réponse qu'il contient est correcte. Le non respect de ces consignes sera fortement pénalisé dans la note.

- Il est toujours possible de répondre à une question en utilisant la réponse d'une question précédente même si cette dernière a été laissée de côté. L'inverse n'est pas autorisé.
- Les complexités demandées sont toujours les complexité temporelles au pire. Toutes les opérations standards sur les entiers et les bouléens sont de complexité $O(1)$.
- Lorsqu'une étude de complexité est attendue, le résultat est exprimé sous la forme $O(f(n))$ où n est la *taille du problème* et f est une fonction la plus simple possible. Si la taille du problème est composé de plusieurs paramètres, par exemple n et m, on attend un résultat comme $O(f(n, m))$.
- On fait de la *science* : tout résultat (en particulier les complexités ou les conversion de formats -binaire, CA2, etc.-) doit être expliqué.
- Comme il est d'usage en CPGE, on réserve les notations droites pour les objets informatique et les notations en italique pour les expressions mathématiques. Par exemple, `x` désigne une variable en C et x désigne le même objet mais dans un contexte mathématique.
- Une mauvaise note n'évalue qu'une copie pas une personne. Un devoir noté 0 n'est pas le signe que l'étudiant concerné est nul, mais indique juste que son devoir est très mauvais ce jour là.

1 Entiers

1.1 Généralités

On abrège dans ce qui suit *complément à 2* en CA2.

Q.1 Quels nombres peut-on représenter en CA2 sur 1 bit ?

Q.2 Convertir -47 en CA2 sur 8 bits avec la méthode du cours.

Q.3 Convertir le nombre $\overline{11100111}$ exprimé en CA2 sur 8 bits en son expression en base 10.

Q.4 Convertir 11.3 comme un flottant binaire avec 5 bits d'exposant et 7 bits de mantisse. Pour les arrondis, on poursuivra le calcul de mantisse jusqu'au 10ème bit et on appliquera la règle de l'arrondi au plus proche pair.

Q.5 Convertir le flottant 010111001101 avec 5 bits d'exposant en base 10.

Q.6 Exprimer en binaire le plus grand flottant FINI à 5 bits d'exposant et 7 bits de mantisse. Puis l'exprimer en base 10.

Q.7 Exprimer en binaire puis en base 10 le flottant $-\infty$ du format à 5 bits d'exposant et 7 bits de mantisse.

Solution. Chaque exo sur 4. Si calculs non détaillés : 1/4. En cas de méthode correcte et d'erreur de calcul : 3/4.

— 0 et -1

— $-47 + 2^8 = 209$ et $\overline{11010001}$

— -25. C'est à dire $2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0 - 2^8$.

— Décalage $2^{5-1} - 1 = 15$. Exposant 3. Exposant décalé $3 + 15 = 18$.
La mantisse sur 10 chiffres : [0, 1, 1, 0, 1, 0, 0, 1, 1, 0]. Arrondi au plus proche pair s'impose.
([0], [1, 0, 0, 1, 0], [0, 1, 1, 0, 1, 0, 1])

— Exposant décalé : $1, 0, 1, 1, 1$ soit 23. Donc l'exposant est 8.
Le résultat est

$$(-1)^0 2^8 (1 + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^6}) = 2^8 + 2^5 + 2^4 + 2^2 = 308$$

— Le flottant est $0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1$.
Exposant décalé 30. $e = 30 - 15 = 15$.

$$2^{15}(\sum_{i=0}^{7} 2^{-i}) = 2^{15} \frac{1 - \frac{1}{2^8}}{1 - \frac{1}{2}} = 2^{15} \times (2 - 2^{-7}) = \underbrace{2^{16} - 2^8}_{\text{attendu}} = 65280$$

— Flottant 1 11111 0000000, en base 10 : -2^{16}

□

1.2 Opposé en CA2

Pour un nombre $a \in \{0, 1\}$, on note $\bar{a} \in \{0, 1\}$ le nombre défini par $a + \bar{a} = 1$. On se donne un nombre entier $n > 2$.

Q.8 Donner le codage U du nombre 1 en CA2 sur n bits.

Q.9 Exprimer le plus petit nombre en CA2 sur n bits.

Q.10 On considère $A = \overline{a_{n-1} a_{n-2} \dots a_1 a_0}$, un nombre en CA2 sur n bits différent du plus petit nombre. On note $\bar{A}$ son *complémentaire* $\bar{A} = \overline{\overline{a_{n-1}}\,\overline{a_{n-2}}\,\dots\,\overline{a_1}\,\overline{a_0}}$.

(a) Pourquoi est-on sûr de ne pas déclencher d'overflow en effectuant l'addition en CA2 $A + \bar{A}$?

(b) Exprimer le résultat de l'addition en CA2 $A + \bar{A}$.

(c) En déduire une méthode pour calculer le CA2 de l'opposé de A.

Solution. — $n - 1$ zéros suivis de 1 : $\overline{0 \dots 01}$.

— 1 suivi de $n - 1$ zéros : $\overline{0 \dots 01}$.

— 1. L'addition $A + \bar{A}$ ne soulève pas d'overflow parce que A et $\bar{A}$ sont de signes opposés.
2. $A + \bar{A} = \overline{11 \dots 1}$, ce qui représente -1.
3. Puisque $A + \bar{A} = \overline{11 \dots 1}$, alors l'opposé de A est codé par $\bar{A} + U$. Cela nous donne une méthode pour calculer $-A$.

□

1.3 Implémentation

Dans toute cette sous-section, les tableaux statiques sont interdits. Tous les pointeurs sont alloués sur le tas (et libérés dès qu'ils ont fini de servir). Par ailleurs, on utilise uniquement le clavier de Bastien. Ce dernier a un gros problème : son clavier est cassé. Il n'a plus accès à la touche pourcentage **%** ni à la touche division /.

1.3.1 Restes, divisions, digits

Q.11 Écrire une fonction `int reste(int n)` qui renvoie le reste de la division euclidienne de n par 2.

Q.12 Écrire une fonction `int divise(int n)` qui prend en paramètre un nombre n supposé positif et renvoie le résultat de la division euclidienne de n par 2.
Le comportement de la fonction si $n < 0$ n'est pas imposé.

Q.13 Écrire une fonction `int digit(int n, int p)` qui prend en paramètres deux nombres n et p supposés positifs. La fonction renvoie le coefficient du monôme 2^p (ce coefficient est appelé un *digit*) dans l'écriture binaire de n . Le comportement de la fonction si $n < 0$ ou $p < 0$ n'est pas imposé.
Par exemple, `digit(11,3)` renvoie 1 et `digit(11,2)` renvoie 0.

Solution. Voici

```
7  int reste(int n){
8    return n & 1;
9  }//L9
```

```
18 int digit(int n, int p){
19   return (n & 1<<p)>>p;
20 }//L20
```

```
29 int divise(int n){
30   return n >> 1;
31 }//L31
```

□

On donne la structure `binaire` et son alias `bin` :

```
1  typedef struct binaire bin;
2
3  struct binaire{
4    int n;// format du CA2
5    int * array;//tableau des digits
6  };
```

Elle modélise l'écriture d'un nombre en binaire sur n bits. On choisit la convention big-endian avec bit de poids fort en premier.

1.3.2 Entiers non signés

Le champ `n` d'un objet de type `bin` représente le nombre de bits de la représentation binaire. BONNE NOUVELLE : LE CLAVIER DE BASTIEN EST RÉPARÉ !

Q.14 Écrire la fonction `bin * zero(int n)` qui renvoie l'écriture de 0 en binaire sur n bits. On suppose $n \geq 1$.

Q.15 Écrire la procédure `void display_bin(bin *b)` qui affiche le codage binaire en big-endian du nombre représenté à l'adresse pointée par `b`.

```
bin *r = zero(6);
display_bin(r);

```

Affiche 000000.

Q.16 Écrire la procédure `void free_bin(bin *b)` qui libère le pointeur `b` et son tableau de digits.

Q.17 Écrire la fonction `bin * tobin(int n, int x)` qui prend en paramètres deux entiers positifs et renvoie l'expression binaire de x sur n bits en big-endian. On ne vérifie pas que x tient dans le format ni que x et n sont positifs.

Q.18 Écrire la fonction `bin * add (bin * b1, bin * b2)` qui prend en paramètres deux codages binaires `b1` et `b2` supposés écrits sur le même nombre de bits.
La fonction réalise l'addition $b_1 + b_2$ en binaire avec gestion de la retenue. Il est interdit d'exprimer b_1 et b_2 comme des `int` puis de revenir au binaire.
On soulève une erreur d'assertion si les formats de `b1` et `b2` sont différents ou si le résultat de l'addition binaire ne tient pas dans le format de `b1`.
Avec :

```
bin * b1 = tobin(5,11);
bin * b2 = tobin(5,7);
bin * res = add(b1,b2);
display_bin(b1);printf("+\n");display_bin(b2);
printf("--------------------\n");
display_bin(res);
free_bin(b1);free_bin(res);

```

on obtient l'affichage

```
01011
+
00111
--------------------
10010
```

Q.19 Écrire la fonction `int bin2int(bin *b)` qui transforme en `int` le binaire pointé par `b`.
Avec

```
int n = 11;
bin *r = tobin(6,n);
display_bin(r);
printf("%d\n",bin2int(r));

```

On obtient

```
001011
11
```

La complexité de la fonction doit être en $O(n)$ où n est la taille du tableau de digits du binaire pointé par `b`. Toute complexité supérieure sera pénalisée.

Solution. Chaque question sur 4. Si la complexité attendue n'est pas respectée mais que la fonction fait bien ce qu'on attend : 2/4

```
62 bin * zero(int n){
63     bin *res = malloc(sizeof(bin));
64     int *dg = calloc(n,sizeof(int));
65     res->n=n;
66     res->array=dg;
67     return res;
68 }//L68
```

```
50 void free_bin(bin *b){
51   free(b->array);
52   free (b);
53 }
```

```
55 void display_bin(bin *b){
56   for(int i=0; i<b->n;i++){
57     printf("%d",b->array[i]);
58   }
59   printf("\n");
60 }
```

```
77 bin * tobin(int n, int x){
78   bin *res = zero(n);
79   for(int i=0;i<n;i++)
80     res->array[i]=digit(x,n-1-i);
81   return res;
82 }//L82
```

```
95 bin * add (bin * b1, bin * b2){
96   assert(b1->n == b2->n);
97   bin *b = zero(b1->n);
98   int r = 0;//retenue
99   for(int i=b1->n-1; i>=0;i--){
100    int x = b1->array[i], y=b2->array[i];
101    b->array[i] = reste (x + y + r); //(x + y + r) % 2;
102    r = (x + y + r) >> 1 ;
103  }//for i
104  assert(r==0);//overflow interdits
105  return b;
106 }//L106
```

Si complexité non linéaire : 2/4

```
126 int bin2int(bin *b){//algo de Horner
127   int r = 0;
128   for(int i=0; i<b->n;i++){
129     r = b->array[i] + r * 2;
130   }
131   return r;
132 }//L132
```

□

1.3.3 Complément à 2

Le champ `n` d'un objet de type `bin` représente le nombre de bits sur lequel s'effectue le CA2.

Q.20 Écrire la fonction `bin * ca2(int n, int x)` qui calcule le complément à 2 de x sur n bits par la méthode du cours. Une erreur d'assertion est soulevée si x ne tient pas dans le format choisi.

Q.21 Écrire la fonction `void extension(bin *b, int n)` qui modifie `b` de sorte que le binaire pointé désigne le même nombre mais dans un CA2 de format strictement plus grand. RESTER EN BINAIRE

Avec :

```
int x = -3;
bin * b = ca2(8,x);
display_bin(b);
extension(b,12);
display_bin(b);

```

on obtient :

```
11111101
111111111101
```

Une erreur d'assertion est soulevée si le nouveau format est plus petit strictement que l'ancien.

Q.22 Écrire la fonction `bin* minformat(bin * b)` qui renvoie un pointeur sur un binaire représantant le même nombre que b mais dans le plus petit format de CA2 possible (ce format peut-être un CA2 sur 1 bit). Il n'y a pas d'effet de bord. RESTER EN BINAIRE

Q.23 Écrire la fonction `bool pluspetit(bin * b)` qui renvoie vrai si b désigne le plus petit nombre du format et faux sinon.

Par exemple, $\overline{10000000}$ est le plus petit nombre du CA2 sur 8 bits mais pas $\overline{00110011}$.

Q.24 Écrire la fonction `bin * opposite(bin *b)` qui renvoie un pointeur sur binaire représentant l'opposé du nombre pointé par `b`. Une erreur d'assertion est soulevée si le nombre n'a pas un opposé qui tient dans le format de `b`. RESTER EN BINAIRE

Solution. Voici

```
bin * ca2(int n, int x){
  assert(-1<<(n-1) <= x && x<1<<(n-1));
  if (x>=0)
    return tobin(n,x);
  return tobin(n,x + (1<<n));
}//L124
```

```
void extension(bin *b, int n){
  assert(n>=b->n);
  int* nouveau = calloc(n, sizeof(int));
  for(int i=n-b->n; i<n ; nouveau[i]=b->array[i-(n-b->n)],i++);
  for(int i=0; i< n-b->n; nouveau[i]=b->array[0],i++);
  free(b->array);
  b->n=n;
  b->array=nouveau;
}//L189
```

```
int _position(bin* b){
  if (b->n==1)
    return 0;
  int next = 1;
```

```
204   while (b->array[0] == b->array[next] && next < b->n)
205     next++;
206   next=next-1;
207   return next;//position du dernier bit égal au 1er
208 }
209
210 bin* minformat(bin * b){
211   int n = _position(b);
212   printf("n=%d\n",n);
213   bin* nouveau = zero(b->n - n);
214   for(int i=n; i<b->n; nouveau->array[i-n]=b->array[i],i++);
215   return nouveau;
216 }//L216
```

```
247 bool pluspetit(bin * b){
248   if (b->array[0]==0)
249     return false;
250   int i = 1;
251   while(i<b->n){
252     if (b->array[i]==1)
253       return false;
254     i++;
255   }
256   return true;
257 }//L257
```

Si oubli de vérifier que le nombre a bien un opposé qui tient dans le format : 2/4

```
284 bin * opposite(bin *b){
285   assert(!pluspetit(b));
286   bin * c = ca2(b->n,0);//complementaire
287   for(int i=0; i<b->n; c->array[i]=1-b->array[i],i++);
288   bin * un = ca2(b->n,1);
289   bin * res = add(un,c);
290   free_bin(c);free_bin(un);
291   return res;
292 }//L292
```

□

2 Ensembles

Dans cette section on se donne le type

```
1   typedef bin set;
```

Il représente des ensemble d'entiers et non plus des représentations de nombres en CA2. Le champ `n` représente le cardinal de l'ensemble représenté.

Voici comment on représente l'ensemble $\{1, 5, 2\}$:

```
1   set *s1 = zero(3);
2   s1->array[0]=1;s1->array[1]=5;s1->array[2]=2;
```

Soit `s` un `set` représentant un ensemble S. Si le tableau `s.array` possède des éléments au delà de la position `s.n`, ces éléments ne sont pas considérés comme des éléments de S. On peut donc facilement représenter l'ensemble vide : c'est un objet de type `set` dont le champ `n` vaut 0 (le contenu du champ `array` étant quelconque).

Q.25 Écrire la fonction `bool appartient(set * s, int x)` qui renvoie vrai si x est dans l'ensemble pointé par `s` et faux sinon.

Q.26 On veut représenter des ensembles et pas des multi-ensembles : il ne doit pas y avoir de doublon de valeurs dans le tableau. Écrire la fonction `bool doublon(set * s)` qui renvoie vrai si l'ensemble représenté par `*s` contient au moins un doublon. La fonction renvoie faux si aucun doublon n'est détecté.

Solution. Voici

```
321 bool appartient(set * s, int x){
322   for(int i=0; i<s->n;i++)
323     if (s->array[i]==x)
324       return true;
325   return false;
326 }//L326
```

```
368 bool doublon(set * s){
369   for(int i=0; i<s->n;i++)
370     for(int j=i+1; j<s->n;j++){
371       if (s->array[i]==s->array[j]){
372   return true;
373       }//if
374     }//for j
375   return false;
376 }//L376
```

□

Par hypothèse, on suppose que les objets manipulés de type `set` ne possèdent pas de doublon de valeur (pas de $\{1, 5, 2, 5\}$).

On se donne la fonction

```
1  set * mystere(set * s1, set *s2){
2    set *s = zero(s1->n);
3    int pos = 0;
4    for(int i=0; i<s1->n;i++)
5     for(int j=0; j<s2->n;j++){
6       if (s1->array[i]==s2->array[j]){
7        s->array[pos]=s1->array[i];
8        pos++;
9      }//if
10   }//for j
11   s->n=pos;
12   return s;
13 }
```

Q.27 Indiquer brièvement ce que réalise l'appel `mystere(s1,s2)`.

Q.28 On note n le cardinal de l'ensemble représenté par `s1` et m celui de l'ensemble représenté par `s2`. Exprimer en fonction de ces deux valeurs, la complexité de l'appel `mystere(s1,s2)`.

Solution.
— Cette fonction calcule l'intersection de deux ensembles ;
— son coût est en $O(n \times n)$ (double boucle imbriquée)

□

Remarque. L'implémentation des ensembles présentée ici est dite *naïve* car les opérations ensemblistes usuelles peuvent être coûteuses. Pour $n \in \mathbb{N}^*$ fixé, on peut utiliser efficacement la représentation binaire des entiers $[\![0, 2^{n-1}]\!]$ pour implémenter les sous-ensembles de $E_n = [\![0, n-1]\!]$.

Un 0 en position $0 \leq i \leq n-1$ d'une expression binaire indique que le nombre i n'est pas dans le sous-ensemble mais un 1 en position j révèle la présence de j.