UNIVERSITÉ CÔTE D'AZUR

Université Côte d'Azur — L2 Informatique
Programmation C

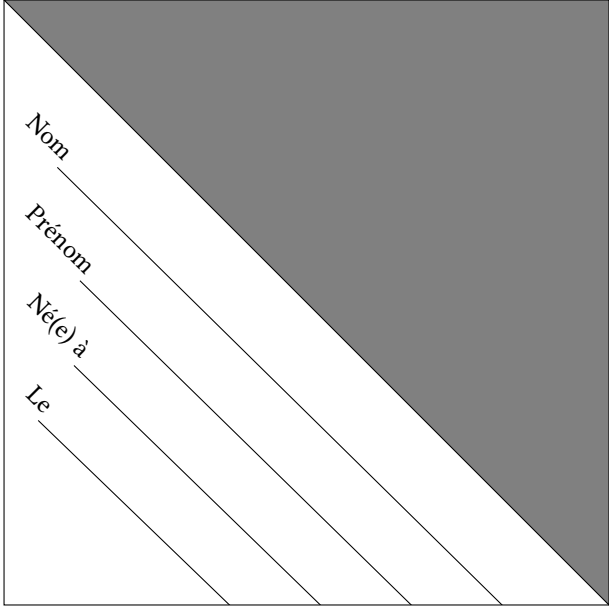
Seconde session : Programmation C

19 juin 2024

Durée : 1 heure
Tous documents autorisés. Il est interdit d'accéder à internet.

Note

Toutes les questions sont indépendantes.
Tous les codes devront être écrits en **Langage C ANSI**. La notation est donnée à titre indicatif.
Nombre de pages : 4



Ne surtout pas rabattre le triangle grisé. Il est là simplement pour faire sérieux. Dans tous les cas, votre copie ne sera pas anonyme car c'est le même enseignant qui corrige et qui rentre les notes.

Entiers et parités (8 points)

Dans cet exercice, on travaille sur des `char` codés sur un octet (8 bits). On ne va travailler qu'avec des nombres inférieurs à 127 codés sur les 7 bits de poids faible. L'objectif de l'exercice est de travailler avec des entiers codés avec un bit de parité. La parité d'un nombre vaut 0 s'il y a un nombre pair de 1 dans l'écriture en base 2 et 1 s'il y en a un nombre impair.

Exemple : le nombre $12 = 0001100_b$ a une parité de 0 (1 apparaît deux fois) quand $19 = 0001011_b$ a une parité de 1 (1 apparaît trois fois).

Le principe de l'exercice est de coder sur le bit b_0 la parité du nombre $b_1b_2b_3\ \ b_4b_5b_6b_7$. Au final l'entier obtenu tient sur un octet.

Par exemple, on codera 12 par 00001100 (qui vaut toujours 12) et 19 par 10001011 (le nombre obtenu vaut 144, mais si on enlève le premier bit on obtient bien 19). Ce bit d'information supplémentaire permet de s'assurer que les données en mémoire n'ont pas été corrompues (modification de la mémoire, erreurs de transmission, etc).

1. Que vaut la variable globale `MASQUE` définie ci-dessous ? Expliquer la notation <<. On pourra utiliser cette variable dans les deux questions suivantes.

```
1 int MASQUE = 1<<7;
```

```
MASQUE vaut 128 = 10000000 en binaire : 1 décalé 7 fois sur la gauche //
```

2. Écrire une fonction `int bit_de_parite(char c)` qui prend en argument un nombre codé sur 8 bits et renvoie le bit de poids fort (le premier : b_0). Pour avoir tous les points à la question il ne faut utiliser que des opérations binaires (pas de boucles).

```
int bit_de_parite(char c) {
  return (c & MASQUE) >> 7;
}
```

3. De même écrire une fonction `char entier(char c)` qui prend en argument un nombre codé sur 8 bits et qui renvoie le nombre obtenu en supprimant le bit de poids fort et en ne conservant que les 7 bits de poids faible. Pour avoir tous les points à la question il ne faut utiliser que des opérations binaires (pas de boucles).

```
char entier(char c) {
  return c & ~MASQUE;
}
```

4. Écrire une fonction `int calcul_parite(char c)` qui calcule la parité globale de `c`. Contrairement à la question 2 où il ne fallait récupérer que le premier bit, il faut cette fois calculer le *ou exclusif* de tous les bits de `c` selon la formule : parité($b_0b_1b_2b_3\ b_4b_5b_6b_7$) = $b_0 \otimes b1 \otimes b2 \otimes b3 \otimes b4 \otimes b5 \otimes b6 \otimes b7$.
Si vous ne vous rappelez pas du symbole en *C* pour le *ou exclusif*, vous pouvez utiliser le symbole mathématique $\otimes$ (mais vous perdrez 0,5 point).

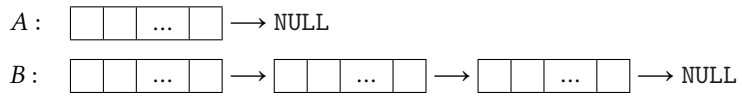
```
char calcul_parite(char c) {
  char p = 0;
  while (c != 0 ) {
    p ^= (c&1); /* On récupére le bit de poids faible (ou p = p^(c%2))*/
    c >>=1;     /* On divise par deux */
  }
  return p;
}
```

5. On se donne maintenant un tableau d'entiers (codés sous forme de `char` stockés avec parité) de taille `N`. Écrire une fonction `float moyenne(char notes[], int N)` qui calcule la moyenne des notes valides du tableau. Chaque note est codée avec son bit de parité. Une note est dite valide si la parité globale (avec le bit de parité) vaut 0. On fera attention de bien supprimer le bit de poid fort dans le calcul de la moyenne. Si le tableau est vide ou qu'aucune note n'est valide, on renverra 0.

```
float moyenne (char tableau[], int N) {
  float somme = 0;
  int nombre_notes = 0;
  int i;

  for (i=0; i<N; i++) {
    if (calcul_parite(tableau[i]) == 0) {
      nombre_notes += 1;
      somme += entier(tableau[i]);
    }
  }

  if (nombre_notes == 0) return 0;

  return somme / nombre_notes;

}
```

Problème (13 points)

On cherche à implémenter un tableau chaîné, un élégant mélange de tableaux et de listes chaînées. Les premiers éléments seront rangés dans un tableau de 10 cases, puis lorsqu'il n'y aura plus de place on pourra allouer un autre tableau auquel on accédera via un pointeur. Ci dessous vous avez l'exemple d'un tableau A de taille 10 et d'un autre B de taille 30.



1. Créez une structure `struct tchaine` contenant deux champs : un tableau de taille de 10 et un pointeur vers un autre `struct tchaine`. Ce pointeur pourra être `NULL` dans le cas du dernier élément de la chaîne.

```c
struct tchaine {
  int tab[10];
  struct tchaine* suivant;
};
```

2. Définir un type `tableau_chaine` comme étant un synonyme du type `struct tchaine`.

```c
typedef struct tchaine tableau_chaine;
```

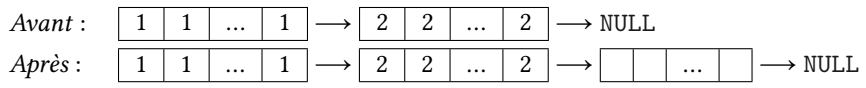
3. Écrire une fonction `tableau_chaine * nouveau(void)` qui alloue sur le tas un tableau chaîné de taille 10 comme illustré ci-dessous et renvoie le pointeur correspondant. Inutile d'initialiser les cases du tableau de la structure.


```c
tableau_chaine * nouveau() {
  tableau_chaine * tc = malloc(sizeof(tableau_chaine));
  tc->suivant = NULL;
  return tc;
}
```

4. Écrire une fonction `void liberer(tableau_chaine * tc)` qui libère toute la mémoire allouée par le tableau chaîné `tc`.

```c
void liberer(tableau_chaine * tc) {
  if (tc->suivant != NULL) liberer(tc->suivant);
  free(tc);
}
```

5. Écrire une fonction `void ajout_memoire(tableau_chaine * tc)` qui ajoute un nouveau tableau (non initialisé) à la fin de la chaîne de tableau `tc`.




```c
void ajout_memoire (tableau_chaine * tc) {
  if (tc->suivant == NULL) {
    tc->suivant = nouveau();
  } else {
    ajout_memoire(tc->suivant);
  }
}
```

6. Écrire une fonction `int nombre_de_cases(tableau_chaine * tc)` qui calcule le nombre total de case du tableau chaîné.

```c
int nombre_de_cases(tableau_chaine * tc) {
  int somme = 0;
  while (tc != NULL) {
    somme += 10;
    tc = tc->suivant;
  }
  return somme;
}
```

7. Écrire une fonction `int maximum(tableau_chaine * tc)` qui calcule le maximum des valeurs du tableau chaîné `tc`. On suppose que le tableau ne contient que des valeurs positives. Si le tableau `tc` est vide, on renverra `0`

```c
int maximum(tableau_chaine * tc) {
  int maxi = 0;
  int i;
  while (tc != NULL) {
    for (i=0; i<10; i++) {
      if (maxi < tc->tab[i]) maxi = tc->tab[i];
    }
    tc = tc->suivant;
  }
  return maxi;
}
```