PROG L3-MI
DS

2024-10-22

Consignes. *La durée de ce devoir surveillé est d'une heure trente. Seule une feuille A4 recto-verso manuscrite et non photocopiée est autorisée.* ***Dans tous les exercices, il est possible d'admettre le résultat d'une question pour répondre aux questions suivantes.*** *Toute réponse doit être justifiée.*

Pour l'ensemble des questions, on supposera que les entêtes nécessaires ont été incluses :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
```

Exercice 1 (environ 10 points)

Q.1 : Qu'affiche le programme suivant ? Expliquez le résultat.

```
void mA(uint32_t* v1, uint32_t* v2) {
        uint32_t x = *v1;
        *v1=*v2;
        *v2=x;
}
void mB(uint32_t* v1, uint32_t* v2) {
        uint32_t* x = v1;
        v1=v2;
        v2=x;
}
int main(void) {
        uint32_t x1=1, x2=1, y1=2, y2=2;
        mA(&x1,&y1);
        mB(&x2,&y2);
        printf("%d %d %d %d\n",x1,y1,x2,y2);
        return EXIT_SUCCESS;
}
```

Q.2 : Quel problème pose l'extrait de code suivant ? Comment pouvez-vous y remédier (sans modifier la signature de la fonction) ?

```
// crée un tableau de size éléments et retourne son adresse
uint8_t* truc(size_t size) {
        uint8_t t[size];
        uint8_t* res=t;
        for (size_t i=0 ; i<size ; i++) {
                t[i]=i;
        }
        return res;
}
```

Q.3 : L'exécution du programme suivant retourne-t-elle toujours le même résultat ? pourquoi ?

```
uint32_t m(size_t size, uint32_t tab[size]) {
        uint64_t res;
        for (size_t i=0 ; i<size ; i++) {
                res+=tab[i];
        }
        return res;
}
int main(void) {
        uint32_t tab[]={3,5,6,3};
        printf("%u\n",m(4,tab));
        return EXIT_SUCCESS;
}
```

Q.4 : Soit l'extrait de programme suivant :

```
#define SIZE 10000000
void init(size_t size, uint64_t tab[size]) {
        for (size_t i=0 ; i<size ; i++) {
                tab[i]=i+1;
        }
}
int main( int argc , char * argv [argc +1]) {
        // --ICI--
        init(SIZE,tab);
        // le reste du programme
        return EXIT_SUCCESS;
}
```

Pour déclarer et allouer le tableau `tab` à la place du commentaire `// --ICI--` , on a les choix suivants :

- `uint64_t tab[SIZE]`
- `static uint64_t tab[SIZE]`
- `uint64_t* tab=(uint64_t) malloc(SIZE*sizeof(uint64_t));`

1. Quelle déclaration est la plus susceptible d'engendrer une erreur à l'exécution ? pourquoi ?
2. Dans le cas ci-dessus, où l'on connaît la taille à la compilation, quelle déclaration est la plus efficace ? pourquoi ?
3. Dans le cas, où l'on ne connaît pas la taille du tableau avant l'exécution, quelles sont les déclarations possibles ?

Q.5 : Que signifie le mot clé `static` lorsqu'il est placé devant la signature d'une fonction ? A quoi cela peut-il servir ?

Exercice 2 (environ 10 points)

Nous voulons pouvoir représenter un ensemble d'entiers S par un entier $s = \sum_{i \in S} 2^i$. Par exemple, l'ensemble $\{4, 1, 7\}$ sera représenté par l'entier $2^1+2^4+2^7 = 146$. Si un ensemble contient un élément supérieur à 63 alors il n'est pas possible d'utiliser un seul `uint64_t`. Nous utiliserons ainsi un tableau d'entiers `uint64_t b[n]`, de telle sorte que $s = \sum_{i=0}^{n-1} b[i]2^{64i}$. Par exemple, l'ensemble $S' = \{74, 1, 4, 7\}$ sera représenté par tableau $[146, 1024]$ et donc $s' = 146 + 1024 \times 2^{64}$

Q.1 :

1. Combien d'entiers de type `uint64_t` faut-il pour pouvoir représenter n'importe quel ensemble d'entiers de type `uint8_t` ?
2. Étant donné n la taille de l'univers, dans quelle condition la représentation d'un ensemble S sous forme d'un entier s est plus économe en mémoire que sa représentation sous forme de tableau ?

Pour la suite, on suppose que le programme contient la macro `#define N XXX`, où `XXX` est la réponse à la question Q.1-1.

Q.2 : Définir la fonction de signature :

```
bool contains(const uint64_t bitset[N], const uint8_t value)
```

qui retourne `true` ssi l'entier `value` appartient à l'ensemble représenté par le tableau d'entiers `bitset`.

Q.3 : Définir la fonction de signature :

```
bool isIncluded(const uint64_t bs1[N], const uint64_t bs2[N])
```

qui retourne `true` ssi l'ensemble représenté par `bs1` est inclus dans celui représenté par `bs2`.

Q.4 : Définir la fonction de signature :

```
void convert(size_t size, const uint8_t tab[size], uint64_t bitset[N])
```

qui calcule et copie dans `bitset` la représentation du tableau d'entiers `tab` de taille `size`. La tableau n'est pas nécessairement trié et peut éventuellement comporter des doublons. Par exemple, l'extrait de code doit afficher `146 1024` :

```
int main(void) {
        uint8_t tab[] = {4,1,7,74,4};
        uint64_t  bitset[N];
        convert(5,tab,bitset);
        // on n'affiche que les 2 premiers uint64_t
        for (size_t i=0 ; i<2 ; i++) printf("%lu ",bitset[i]);
        return EXIT_SUCCESS;
}
```

Q.5 : En réutilisant la fonction précédente, proposez une fonction qui ne prend en entrée que le tableau d'entier `tab` ainsi que sa taille `size` et qui retourne la représentation associée (i.e le bitset).