Ensembles - feedback

INFO0902 - Structures de données et algorithmes

Jean-Michel Begon

Université de Liège

8 mai 2015

Le positif
Bilan (en général)

(+) Exactitude des fonctions

(+) Bon choix de fonctions auxiliaires, bonne organisation du code

(+) Capacité, facteur de charge, etc. en `static const`

(+) Gestion élégante de la réallocation

(+) Respect des conventions de nommage

Le moins positif
Bilan (en général)

- (-) Gestion de la mémoire ! !
- (-) `includes` inutiles (`stdio` en particulier)
- (-) `includes` manquants (`StringArray` notamment)
- (-) Fonctions auxiliaires incomplètes
  - ∝ Pas de doc.
  - ∝ Pas de `static`
  - ∝ Pas de `const` à certains endroits

    (`http://www.thegeekstuff.com/2012/06/c-constant-pointers/`)
- (-) Vérification des retours incomplète
- (-) Redondance (`strcmp`)
- (-) *one-liner*

one-liner

```
static bool foo(int a)
{
    if(a == A)
        return true;
    else
        return false;
}

static bool foo(int a)
{
    return a == A;
}
```

Redondance, le cas de `strcmp`

1. Pas besoin de la réimplémenter
2. Mémoïsation (la fonction a un coût facilement évitable)

```
if(strcmp(element, node->elem) == 0)
    do_something();
else if(strcmp(element, node->elem) > 0)
    do_something_else();
else
    do_something_else_else();
```

Vérification des retours

(- - -) Pas de vérification

(- -) Vérification partielle

(-) Vérification complète mais mauvaise gestion en cas d'erreur

```
...
Set* set1 = createEmptySet();
if (!set1)
    return NULL;
Set* set2 = createEmptySet();
if (!set2)
    return NULL;
...
```

La mémoire

Problèmes

- Fuites
  - Mauvaise allocation
  - Mauvaise libération
  - Copie des mots (non respect de l'énoncé)
- Lectures invalides $\Rightarrow$ erreurs de segmentation

Mauvaise allocation
La mémoire

```
Node* x = (Node*) malloc(sizeof(Node));
x = set->root;
```

L'allocation dynamique n'est pas nécessaire :

```
Node* x = set->root;
```

Mauvaise libération : Valgrind

La mémoire

```
gcc main.c StringArray.c TreeSet.c Intersection.c  --std=c99
  --pedantic -Wall -Wextra -Wmissing-prototypes -g -o intersect

valgrind ./intersect French.txt English.txt
```

Compilation avec le flag -g pour avoir les informations de débogage. Résultat :

```
==8013== HEAP SUMMARY:
==8013==     in use at exit: 1,082,120 bytes in 67,633 blocks
==8013==   total heap usage: 370,672 allocs, 303,039 frees, 23,703,012
==8013==
==8013== LEAK SUMMARY:
==8013==    definitely lost: 721,416 bytes in 45,089 blocks
==8013==    indirectly lost: 360,704 bytes in 22,544 blocks
==8013==      possibly lost: 0 bytes in 0 blocks
==8013==    still reachable: 0 bytes in 0 blocks
==8013==         suppressed: 0 bytes in 0 blocks
==8013== Rerun with --leak-check=full to see details of leaked memory
```

```
==1840== Memcheck, a memory error detector
==1840== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al
==1840== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright i
==1840== Command: ./test 0 1
==1840==
==1840== Invalid read of size 4
==1840==    at 0x8049F87: freeSet (TreeSet.c:121)
==1840==    by 0x80493A4: main (tester.c:289)
==1840==  Address 0x4040444 is 12 bytes inside a block of size 20 freed
==1840==    at 0x4005ECD: free (vg_replace_malloc.c:366)
==1840==    by 0x8049FBE: freeSet (TreeSet.c:128)
==1840==    by 0x804A1FF: setIntersection (TreeSet.c:245)
==1840==    by 0x804929B: main (tester.c:272)
```

Tips and tricks

1. Initialiser un tableau alloué dynamiquement à `NULL`
   - `calloc(length, sizeof(char*))`
2. Enlever les "`#include<stdio>`"
   - `grep "stdio" *.c`
3. Indentation
   - `less HashSet.c`
4. Se conformer à la doc
   - `strcmp == -1` $\leftrightarrow$ dangereux !
5. Attention à la différence entre comparaison par valeur et comparaison par adresse