

EN224 - Test et vérification

ALBERTY Maxime

9 février 2021

Contents

1	Introduction	2
2	Software	2
2.1	Etape 1	2
2.2	Etape 2	2
2.3	Etape 3	3
2.4	Etape 4	4
2.5	Etape 5	4
2.6	Etape 6	4
3	Hardware	5
3.1	Etape 1	5
3.2	Etape 2	5
3.3	Etape 3	5
4	Conclusion	5

1 Introduction

2 Software

2.1 Etape 1

L'implémentation de la fonction PGCD fût une simple traduction de l'algorithme présenté en langage C.

```
1 int PGCD(int A, int B)
2 {
3     while(A != B){
4         if (A > B){
5             A = A - B;
6         } else {
7             B = B - A;
8         }
9     }
10    return A;
11 }
12
```

Cependant il est important de resté concentré, car même si l'algorithme est trivial, une étourderie est vite arrivé et peut faire perdre du temps inutilement.

Le test de la fonction est effectué au moyen de couples de valeurs d'on aura préalablement calculé le PGCD.

```
1 printf("PGCD(1024,800) = %d\n",PGCD(1024,800));
2 printf("PGCD(800,1024) = %d\n",PGCD(800,1024));
3 printf("PGCD(32767,65535) = %d\n",PGCD(32767,65535));
4 printf("PGCD(65535,32767) = %d\n",PGCD(65535,32767));
5 printf("PGCD(512,2048) = %d\n",PGCD(512,2048));
6 printf("PGCD(2048,512) = %d\n",PGCD(2048,512));
7 printf("PGCD(458,6272) = %d\n",PGCD(458,6272));
8 printf("PGCD(6272,458) = %d\n",PGCD(6272,458));
9 printf("PGCD(783,125) = %d\n",PGCD(783,125));
10 printf("PGCD(125,783) = %d\n",PGCD(125,783));
```

Il faut ensuite comparé manuellement les résultats aux calculs.

2.2 Etape 2

Afin de pouvoir test plus de valeurs sans avoir à dupliquer les lignes de tests, nous ajoutons une génération aléatoires des valeurs de A et B comprised entre 0 et 65535.

```
1 #define MAXRAND 65535
2 #define MINRAND 0
3
4 int RandA(void){
5     int A = (rand() % (MAXRAND + 1 - MINRAND)) + MINRAND;
6     return A;
7 }
8
9 int RandB(void){
10    int B = (rand() % (MAXRAND + 1 - MINRAND)) + MINRAND;
11    return B;
12 }
```

En ajoutant une boncle FOR dans le main, on peut ainsi test beaucoup plus de valeurs.

```
1 for(int i = 0 ; i < 200000 ; i++){
2     A = RandA();
3     B = RandB();
4     printf("%d\t%d\t%d\t%d\n", i, A, B,PGCD(A, B));
5 }
```

Avec l'ajout de cette fonctionnalité, j'ai pu remarquer que la fonction PGCD ne prenait pas en compte les cas où $A = 0$ ou $B = 0$. Conformément à l'annexe du sujet, la fonction PGCD devient alors :

```

1 int PGCD(int A, int B)
2 {
3     while(A != B){
4         if (A==0) return B;
5         if (B==0) return A;
6         if (A > B){
7             A = A - B;
8         } else {
9             B = B - A;
10        }
11    }
12    return A;
13 }

```

La vérification des résultats de la fonction PGCD est possible mais beaucoup trop long car il est nécessaire de comparer manuellement les résultats avec les calculs préalablement effectués.

2.3 Etape 3

Afin de tester plus de couple d'entrée, il peut être intéressant de comparer ma fonction PGCD avec une autre déjà éprouver. La nouvelle approche est la suivante :

- Assignez à N_1 la valeur de N_2 et à N_2 la valeur du reste de la division de N_1 par N_2 ;
- Recommencez jusqu'à ce que le reste de la division soit nul;
- A ce moment, N_1 contient alors le PGCD de N_1 et N_2 .

Cette approche correspond à cette fonction C :

```

1 int PGCD2(int A, int B){
2     int reste;
3
4     while(B != 0){
5         reste = A % B;
6         A = B;
7         B = reste;
8     }
9     return A;
10 }

```

En modifiant la boucle de l'étape 2, il est possible de tester plus de valeur en comparant le résultat des fonctions PGCD1 et PGCD2.

```

1 for(i = 0 ; i < 65536 ; i++){
2     A = RandA();
3     B = RandB();
4     pgcd_1 = PGCD1(A,B);
5     pgcd_2 = PGCD2(A,B);
6     test = (pgcd_1==pgcd_2)?true:false;
7     printf("%d\t%d\t%d\t%d\t%d\n", i, A, B,PGCD1(A, B), test);
8 }

```

Cette méthode nous permet de comparer les résultats de deux fonction effectuant la même opération mais avec des approches différentes. Seulement si les deux fonctions ont des résultats identiquement faux pour des couples de valeurs, le test sera positif au lieu d'indiquer une erreur de calcul.

2.4 Etape 4

La vérification des résultats peut commencer en assurant une cohérence entre les valeurs d'entrée et de sortie de la fonction PGCD. Cela est réalisé par des assertions.

```
1 int PGCD(int A, int B)
2 {
3     //Pre-condition
4     assert(A>=0);
5     assert(B>=0);
6     assert(A<=65535);
7     assert(B<=65535);
8
9     while(A != B){
10         if (A > B){
11             A = A - B;
12         } else {
13             B = B - A;
14         }
15     }
16     return A;
17 }
```

Ces pré-conditions vont permettre d'assurer que les valeurs de A et B font partie de la plage des valeurs pour laquelle la fonction est conçue.

Ajouter limitation des pré-conditions

Il faut également noter que les assertions ne servent que pour le développement. Lors de la compilation de la version finale du programme, il faut désactiver les assertions.

```
1 gcc mon_prog.c -o mon_prog //Compilation avec les assertions
2 gcc mon_prog.c -o mon_prog -NDEBUG //Compilation sans les assertions
```

2.5 Etape 5

En plus de vérifier si les données fournis à la fonction sont conformes, on va vérifier que le résultat est cohérent.

- La valeur de sortie ne peut pas être plus grande qu'une des valeurs d'entrée,
- Si $A \neq 0$ et $B \neq 0$, la valeur à la sortie de la boucle WHILE ne peut pas être également à 0.

```
1 int PGCD(int A, int B)
2 {
3     int firstA = A;
4     //pre-condition
5     assert(A>=0);
6     assert(B>=0);
7     assert(A<=65535);
8     assert(B<=65535);
9     while(A != B){
10         if(A == 0) return B;
11         if(B == 0) return A;
12         if (A > B){
13             A = A - B;
14         } else {
15             B = B - A;
16         }
17     }
18     //Post-condition
19     assert(A > 0);
20     assert(A <= firstA);
21     return A;
22 }
```

Ajouter limitation des post-conditions

2.6 Etape 6

Les tests unitaires.

3 Hardware

3.1 Etape 1

3.2 Etape 2

3.3 Etape 3

4 Conclusion

Todo list

■ Ajouter limitation des pré-conditions	4
■ Ajouter limitation des post-conditions	4