TP 2 : Introduction à C (Partie 2)

18 septembre 2023

1 Assertions

Exercice 1 :

Reprendre les programmes des exercices 9 et 11 du TP précédent pour y ajouter des assertions correspondant aux préconditions du programme. Tester sur des exemples.

Exercice 9 :

```
#include <stdio.h>

int main(){
    int x;
    int y;

    printf("Valeur de x : ");
    scanf("%d",&x);

    assert(x>0); // x est un entier naturel non-nul

    if (x%2==0){ // cas n pair
        y = x/2;
    }
    else{ // cas n impair
        y = 3*x+1;
    }

    printf("le successeur de %d dans la suite de Syracuse est %d",x,y)
}
```

Exercice 11 :

```
#include <stdio.h>

int main(){
    float l1;
    float l2;
    float l3;
```

```
    printf("Valeur de l1 : ");
    scanf("%f",&l1);
    printf("Valeur de l2 : ");
    scanf("%f",&l2);
    printf("Valeur de l3 : ");
    scanf("%f",&l3);

    assert (l1>=0 & l2>=0 & L3 >=0); // Longueurs positives
    assert (l1+l2 >= l3); // Premiere inegalite triangulaire
    assert (l1+l3 >= l2); // Deuxieme inegalite triangulaire
    assert (l3+l2 >= l1);

    if (l1*l1 + l2*l2 == l3*l3
    || l1*l1 + l3*l3 == l2*l2
    || l3*l3 + l2*l2 == l1*l1) // Trois cas possible pour un triangle rectangle
    {
        printf("Le triangle est rectangle");
    }
    else{
        printf("le triangle n'est pas rectangle");
    }
}
```

Exercice 2 :

```
#include <stdio.h>
#include <assert.h>

int syracuse(int n)
{
    assert (n>0);
    if (n%2==0){        // cas pair
        return n/2;
    }
    else{               // cas impair
        return 3*n+1;
    }
}

int main(){
    int n;      // termes de la suite
    int tdv = 1;    // temps de vol

    printf("Valeur de n : ");
    scanf("%d",&n);

    printf("%d \n",n);      // affichage du premier nombre
    while (n!=1){
```

```
        tdv++;              // augmentation temps de vol d'une unité
        n = syracuse(n);
        printf("%d \n",n);     // calcul et affichage du successeur
    }

    printf("temps de vol : %d",tdv);
}
```

2 Exercices

Exercice 3 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un flottant f positif, d'un entier naturel non-nul n et qui calcule la troncature à n décimales de $\sqrt{f}$ (sans utiliser la fonction `sqrt` !)

```
#include <stdio.h>
#include <assert.h>

float puissance(float k,int n){
    assert (n>0);
    float c = 1; // valeur a partir de laquelle on calcule k^n
    int i;
    for(i=1;i<=n;i++){  // n iterations de boucle
        c = c*k;// multiplication par k à chaque itération
    }
    return c;
}

int main(){

    float f;
    int n;
    printf("Valeur de f : ");
    scanf("%f",&f);

    assert(n>0);

    printf("Nombres de décimales : ");
    scanf("%d",&n);

    float c = 0;

    /*
    Pour éviter des imprécisions liées aux flottants :
    -On utilise un compteur qu'on augmente d'une unité à chaque itération
    -Ce compteur doit atteindre sqrt(f)*10^n : son carré doit atteindre
```

```
    f*10^(2n) grâce à une boucle
    -On multiplie par (1/10)^n à la fin
    */
    while(c*c<f*puissance(10,2*n)){
        c = c+1;
    }
    c=c-1; // on baisse d'une unité parce qu'on a dépassé à la fin de la boucle

    float p = puissance(0.1,n);

    c=c*p;
    printf("%f",c);
}
```

Exercice 4 :

Ecrire un programme qui demande à l'utilisateur de donner les solutions de 3 calculs successifs affichés dans le terminal (exemple : "3 x 7 = ?") et ne passe au calcul suivant (ou s'arrête à la fin) qu'une fois que la bonne valeur est rentrée.

```
#include<stdio.h>

int main(){

    int s1;
    int s2;
    int s3;

    printf("3 x 7 = ? \n");
    while(s1!=21){
        scanf("%d",&s1);
    }

    printf("300 - 180 = ? \n");
    while(s2!=120){
        scanf("%d",&s2);
    }

    printf("63 / 9 = ? \n");
    while(s3!=7){
        scanf("%d",&s3);
    }

}
```

Exercice 5 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel n, et affiche la valeur de $n!$.

```
#include <stdio.h>
```

```
#include <assert.h>

int main(){

    int n;

    printf("Valeur de n ? \n");
    scanf("%d",&n);

    assert(n>=0);

    int fact = 1;

    int c;

    for(c=1;c<=n;c++){
        fact = fact*c;
    }
    printf("factorielle(%d) = %d",n,fact);
}
```

Exercice 6 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel n, et affiche la somme des entiers de 1 à n.

```
#include <stdio.h>
#include <assert.h>

int main(){

    int n;

    printf("Valeur de n ? \n");
    scanf("%d",&n);

    assert(n>=0);

    int somme = 0;

    int c;

    for(c=1;c<=n;c++){
        somme = somme+c;
    }
    printf("La somme des entiers de 1 à %d vaut %d",n,somme);
}
```

Exercice 7 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel n, et affiche la somme des carrés des entiers de 1 à n.

```
#include <stdio.h>
#include <assert.h>

int main(){

    int n;

    printf("Valeur de n ? \n");
    scanf("%d",&n);

    assert(n>=0);

    int somme = 0;

    int c;

    for(c=1;c<=n;c++){
        somme = somme+c*c;
    }
    printf("La somme des carrés des entiers de 1 à %d vaut %d",n,somme);
}
```

Exercice 8 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel n, et affiche la somme des 2^k pour k compris entre 1 à n.

```
#include <stdio.h>
#include <assert.h>


/* On peut reprendre la fonction puissance, avec un type entier */

int puissance(int k,int n){
    assert (n>0);
    int c = 1; // valeur a partir de laquelle on calcule k^n
    int i;
    for(i=1;i<=n;i++){  // n iterations de boucle
        c = c*k;// multiplication par k à chaque itération
    }
    return c;
}

int main(){

    int n;
```

```
    printf("Valeur de n ? \n");
    scanf("%d",&n);

    assert(n>=0);

    int somme = 0;

    int c;

    for(c=1;c<=n;c++){
        somme = somme+puissance(2,c);
    }
    printf("La somme des puissances de 2 de 2^1 à 2^%d vaut %d",n,somme);
}
```

Exercice 9 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel non-nul n, d'un entier naturel non-nul k, et affiche les k premiers termes de la suite de Syracuse à partir de n.

```
#include <stdio.h>
#include <assert.h>

/* On reprend la fonction Syracuse définie précédemment */

int syracuse(int n)
{
    assert (n>0);
    if (n%2==0){        // cas pair
        return n/2;
    }
    else{               // cas impair
        return 3*n+1;
    }
}

int main(){
    int n;      // termes de la suite
    int tdv;    // temps de vol

    printf("Valeur de n : ");
    scanf("%d",&n);


    printf("Temps de vol : ");
    scanf("%d",&tdv);
```

```
    int i;

    for (i=1;i<=tdv;i++){
        printf("%d \n",n);
        n = syracuse(n);
    }
}
```

Exercice 10 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel non-nul n, d'un seuil s, et affiche les termes de la suite de Syracuse à partir de n jusqu'à ce que l'un d'entre eux soit inférieur à s, ainsi que le nombre de termes affichés.

```
#include <stdio.h>
#include <assert.h>

/* On reprend le programme de l'exercice 2, en demandant en plus
à l'utilisateur de donner la valeur du seuil et en modifiant la
condition "n différent de 1" en n supérieur ou égal au seuil" */

int syracuse(int n)
{
    assert (n>0);
    if (n%2==0){        // cas pair
        return n/2;
    }
    else{               // cas impair
        return 3*n+1;
    }
}

int main(){
    int n;      // termes de la suite
    int tdv = 1; // temps de vol
    int s;    // seuil

    printf("Valeur de n : ");
    scanf("%d",&n);

    printf("Seuil : ");
    scanf("%d",&s);

    assert(s>0);

    printf("%d \n",n);      // affichage du premier nombre
    while (n>=s){
        tdv++;              // augmentation temps de vol d'une unité
        n = syracuse(n);
```

```
        printf("%d",n);     // calcul et affichage du successeur
    }

    printf("temps de vol : %d",tdv);
}
```

Exercice 11 :

Ecrire un programme qui demande à l'utilisateur de rentrer la valeur d'un entier naturel non-nul n, d'un seuil s, et affiche les termes de la suite de Fibonacci jusqu'à ce que l'un d'entre eux soit inférieur à s.

3 Bonus : Utilisation de l'aléa

Pour utiliser des nombres générés aléatoirement, on peut utiliser la librairie `stdlib.h` ; la fonction `rand()` (sans arguments) renvoie un entier compris entre 0 et une borne supérieure appelée RAND_MAX.

Pour pouvoir bien utiliser l'aléa en ayant des générations de nombres qui soient différentes à chaque exécution, il faut mettre une "graine" (seed), c'est-à-dire un entier qui déterminera la génération de l'aléa dans la suite du programme : si un programme est exécuté deux fois avec une même graine, les nombres générés aléatoirement seront identique. La fonction permettant de rentrer le seed est `srand`.

Pour s'assurer d'avoir des graines différentes (et donc des exécutions avec des résultats différents), on choisit généralement de prendre comme graine la valeur `time(NULL)`, qui donne le nombre de secondes écoulées depuis le 1er janvier 1970. Pour cela, on utilise la librairie `time.h`.

Exercice 12 :

Lancer le programme suivant plusieurs fois :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    srand(time(NULL));
    int x;
    x = rand();
    printf("%d \n",x);
    x = rand();
    printf("%d \n",x);
    x = rand();
    printf("%d \n",x);
    x = rand();
    printf("%d \n",x);
    x = rand();
    printf("%d \n",x);
```

}

Remplacer `time(NULL)` par un entier constant et lancer le programme plusieurs fois. Que se passe-t-il ?

Souvent, lorsque l'on utilise l'aléa, on peut être amené à vouloir générer des valeurs dans un ensemble donné et selon une loi qui ne correspondent pas nécessairement à ce qui est généré par `rand`. Pour cela, on peut avoir recours à des écritures particulières. Par exemple :

— Pour avoir un entier naturel choisi uniformément entre 0 et $k-1$, on peut prendre le reste de la division euclidienne de `rand()` par k.

— Pour avoir un flottant choisi uniformément dans l'intervalle $[0;1]$, on peut utiliser l'expression suivante : `float x = ((float)rand()/(float)(RAND_MAX))`

Exercice 13 :

Ecrire un programme qui demande deux bornes entières a et b et génère et affiche un entier sélectionné uniformément entre a et b.

Exercice 14 :

Ecrire un programme qui demande deux bornes a et b et génère et affiche un flottant sélectionné uniformément dans l'intervalle $[a;b]$.

Exercice 15 (Algorithme de Monte-Carlo) :

Soit une surface S_1 d'aire A_1 finie inconnue à l'intérieur d'une surface S_2 d'aire A_2 finie connue. Pour avoir une estimation en temps borné de A_1/A_2, on peut générer n points uniformément dans la surface S_2 et compter la proportion de points générés à l'intérieur de S_1. Un tel algorithme pour une telle tâche, où le temps de calcul est déterministe mais le résultat est aléatoire, s'appelle un **algorithme de Monte-Carlo**.

Exemple : on considère que S_1 est le disque de centre (0,0) et de rayon 1, et S_2 est le carré dont les abscisses et ordonnées des sommets sont ± 1. Utiliser un algorithme de Monte-Carlo qui demande le nombre de points à générer dans S_2 pour estimer A_1/A_2. *Indice : pour générer un point dans S_2 uniformément, on peut générer son abscisse uniformément dans* $[0;1]$ *et faire de même pour son ordonnée.*