TD1: Introduction au langage C

De python au langage C

Voici un exemple de programme python permettant de calculer la somme des n premiers entiers et le programme équivalent en C. Dans les deux cas, le programme affiche 55.

Version Python

```python
def somme_entiers(n: int):
    somme = 0
    for i in range (n+1):
        somme += i
    return somme

print(somme_entiers(10))
```

Version C

```c
#include stdio.h
int sommeEntiers(int n)
{
    int somme = 0;
    for(int i=0;i<n+1;i++){
        somme += i;
    }
    return somme;
}
int main(void)
{
    printf("%d\n",sommeEntiers(10));
    return 0;
}
```

1. Quel est le type de retour de la fonction `sommeEntiers`?
2. Quel est le type de retour de la variable `somme` définie dans la fonction `sommeEntiers`?
3. Supposons que nous modifions légèrement le code Python en remplacant `somme = 0` par `somme = 0.0`. Le résultat de l'exécution de ce programme serait `55.0`. Quel est le type de cette valeur? Si nous faisons la même chose avec notre programme C, le résultat reste pourtant toujours 55... Comment expliquez-vous cela?
4. Voici maintenant le message obtenu si on modifie le code C par `int somme = 0.1` et que l'on compile le programme avec la commande suivante (que nous décortiquerons en tp par la suite):

```
$ gcc -Wall -Wconversion -std=c11 td1.c -o exec
td1.c: In function ‘sommeEntiers’:
td1.c:7:17: warning: conversion from ‘double’ to ‘int’ changes value
from ‘1.0000000000000001e-1’ to ‘0’ [-Wfloat-conversion]
```

```
7 |     int somme = 0.1;
  |                 ^~~
```

Comment interprétez-vous cet avertissement? Quel sera le résultat du programme?

5. Voici un nouveau programme python.

```python
def mystere(n):
 res = 1
 for i in range(2,n+1):
     res *= i
 return res
```

Que fait cette fonction selon vous? Proposez une implémentation en C de cette fonction.

Trouver l'erreur

Les exemples suivants contiennent des erreurs de syntaxe. Trouvez-les et proposez une correction.

Exemple 1

```
int x = 12;
if (x > 9) {
    printf("Bravo! vous avez obtenu
    votre BUT");
    if (x > 11) {
        printf(" avec mention,
    bravo!");
    else {
        printf(" sans mention.");
    }
    printf("\n");
}
```

Exemple 2

```
int a = 1;
b = 2;
int tmp = a
a = b;
b = tmp;
printf("%d %d\n",a,b);
```

Exemple 3

```
int i ;
for (i =0; i < 10; i++){
    while j < 10 :
        print("bonjour!");
}
```

Evaluation d'expressions

Nous nous intéressons maintenant à l'évaluation d'expressions plus ou moins complexes. Pour chacune des expressions suivantes, donnez la valeur après évaluation. Vous donnerez le détails de l'évaluation.

- `1 - 2 + 3`
- `1 - (2 + 3)`
- `x < y && y < z && z < x`
- `1 != 4 - 3 + 3 - 2 / 2 + 1`
- `1 + - 2 * 3`
- `(!x && y) || (x && !y)`

Un peu d'ordre

On veut implémenter les algorithmes classiques de tri par sélection et par insertion afin de trier un tableau d'entiers.

Voici quelques rappels concernant l'utilisation des tableaux en C.

```c
int tab[10]; // on crée un tableau de 10 entiers
for (int i = 0; i<10; i++){ // on parcourt le tableau et on stocke la valeur 0 à chaque case
    tab[i] = 0; // l'opérateur [] permet d'accéder à la ieme case du tableau.
                // Les accès se font en temps constant.
}
```

1. Rappeler le principe du tri par insertion et du tri par sélection.
2. Quelle est l'ordre de grandeur, en fonction de la taille `n` du tableau, du temps d'exécution dans le pire cas de chacun de ces algorithmes? On supposera que toutes les opérations élémentaires (arithmétique, comparaison, affectation) ont un coût constant.
3. Donner pour chacun d'eux un exemple de tableau menant au pire cas.
4. Proposer une implémentation en C de ces deux algorithmes.