esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TD n°2 INF1031

Variables

Arnaud Bannier
Nicolas Bodin
Aurélien Texier

0.1. Type et affichage

Pour chacune des valeurs suivantes, donnez le type de données le plus approprié pour la stocker.

Valeur	`3.14`	`-1234`	`37`	`'D'`	`1234567890`	`1.5e100`
Type						

Comme vous le savez, la fonction `printf` permet d'afficher des messages sur la sortie standard (c'est-à-dire votre console) qui peuvent contenir les valeurs de plusieurs variables. Pour indiquer que l'on souhaite afficher une variable, on utilise un *descripteur de type*. Le tableau suivant résume les principaux descripteurs qu'il faut connaître.

Descripteur	pour afficher ...
`%c`	un caractère
`%d`	une valeur entière signée
`%u`	une valeur entière non signée
`%f`	une valeur réelle
`%e`	une valeur réelle en notation scientifique
`%g`	une valeur réelle sous la notation la plus courte

Déterminez ce qu'affiche le code suivant (sans l'exécuter, bien entendu), sachant que le caractère `'A'` correspond à l'entier 65.

```c
#include <stdio.h>

int main()
{
  printf( "73 = %d = %f = %e = %g = %c\n",
    73, 73.0, 73.0, 73.0, 73 );
  return 0;
}
```

0.2. De l'ordre

Formatez le code suivant (retours à la ligne, indentation), compilez-le, puis exécutez-le. Quelle est son utilité ?

```c
#include <stdio.h>

#define N 121
int main(
){ int
nb = N/100; nb = nb + ((N - nb*100) /
10)* 10; nb+= 100 *(N % ((N/
10)*10)); printf("Nombre magique : %d\n"
,nb) ;return
0
;}
```

0.3. Le jeu des 7 casts

Explicitez tous les casts implicites et explicites du code suivant. Juste un indice, il y en a 7 en tout. Déduisez-en ce que ce code affiche.

```c
#include <stdio.h>

int main()
{
    int a;
    float x, y;

    x = 100;
    a = 1 / x * 150;
    y = a * 73.0;

    printf( "y = %d\n", (int)y );

    return 0;
}
```

0.4. Calcul sur les flottants

0.4.a. Opérations usuelles

Dans cette section, vous allez découvrir que le langage C est un peu comme vous, il fait des erreurs de calculs ... Recopiez le code suivant et exécutez-le.

```c
#include <stdio.h>

int main()
{
    float x, y, z;
    int i;

    printf( "Multiplication\n" );
    x = 0.0f;
    for ( i = 0; i < 1000; i++ )
    {
        x += 0.01f;
    }

    printf( "1000 * 0.01 = %f\n", x );
    x = 0.01f;
    printf( "1000 * 0.01 = %f\n\n", 1000.0f * x );

    printf( "Associativite et commutativite\n" );
    x =  100000000.0f;
    y = -100000000.0f;
    z =  1.0f;

    printf( "x = %f, y = %f, z = %f\n", x, y, z );
    printf( "x + y + z = %f\n", x + y + z );
    printf( "x + z + y = %f\n", x + z + y );
    printf( "(x + y) + z = %f\n", (x + y) + z );
    printf( "x + (y + z) = %f\n", x + (y + z) );

    return 0;
}
```

Le résultat de l'exécution devrait vous surprendre, du moins je l'espère. Comment l'expliquez-vous ?

0.4.b. Méthode d'Euler et exponentielle

Nous allons maintenant étudier un programme permettant de trouver une approximation du nombre e = exp(1) à l'aide de la méthode d'Euler. Le principe est relativement simple. Vous savez que la fonction exponentielle est égale à sa propre dérivée. Ainsi, la tangente à sa courbe représentative au point x a une pente de e^x. On peut donc approcher e^{x+h} par $e^x + e^x \times h$. Bien entendu, plus h est proche de 0 et plus e^{x+h} est proche de son approximation. Pour trouver une approximation de e, nous partons de 1 (puisque $e^0 = 1$)

et nous avançons par pas de $h = 0.001$ en utilisant l'approximation précédente pour arriver jusqu'à 1. Le code suivant est une implémentation en C de cet algorithme.

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

int main()
{
    float x, h = 0.001f, e;

    e = 1.0f;
    for ( x = 0.0f; x != 1.0f; x += h )
    {
        e += e * h;
    }
    printf( "e = %f\n", e );
    printf( "e = %f\n", expf(1.0f) );
    return 0;
}
```

Pour comparer notre approximation avec une valeur de référence, nous utilisons la fonction `expf` de la librairie `math.h` qui calcule la fonction exponentielle sur des `float`. La compilation avec `gcc` d'un programme utilisant cette librairie nécessite l'option `-lm`.

Exécutez ce code. La combinaison de touches « `Ctrl + c` » (ou « `Ctrl + z` ») vous sera probablement utile. Expliquez ce qui pose problème puis corrigez le programme en conséquence. Exécutez-le à nouveau.

Il est possible de montrer que notre approximation converge vers e quand h tend vers 0. Exécuter votre programme avec

`h = 1.0f`, `h = 0.1f`, `h = 0.01f`, `h = 0.001f` puis `h = 0.0001f`.

Que constatez-vous ? Exécutez maintenant le programme avec

`h = 0.00001f`, `h = 0.000001f` puis `h = 0.0000001f`.

Comment expliquez-vous ces résultats ?

0.5. Ca permute, ça permute !

Dans cette partie, nous allons apprendre à échanger les valeurs de plusieurs variables.

— Premièrement, écrivez un programme qui échange les valeurs de deux entiers `a` et `b`. Vous pouvez utiliser une variable temporaire `tmp`.

— Écrivez maintenant un programme qui permute circulairement les valeurs de trois entiers `a`, `b` et `c`. Vous pouvez utiliser une variable temporaire `tmp`.

— Essayez maintenant d'échanger les valeurs de `a` et `b` *sans* utiliser de variable temporaire. Plusieurs réponses sont possibles, pensez à votre cours de logique et au « ou exclusif » pour trouver la plus élégante.