

à refaire jusqu'au 8!

Premiers pas dans la programmation

1 Présentation du langage

2 Les variables

2.1 Déclaration et modification

Exercice 1. Déclarations de variables erronées.

Toutes les déclarations de variables suivantes sont erronées. Expliquez pourquoi.

- | | |
|--|-------------------------|
| 1) int variable ; | 6) char toto; int toto; |
| 2) float 0.02; <i>nom!</i> | 7) a; |
| 3) double 7var; | 8) double pi=3,14; |
| 4) int a, double b; <i>2 lignes</i> | 9) int a, int b; |
| 5) char car0; <i>pas de symbole</i> | 10) shart var 0; |

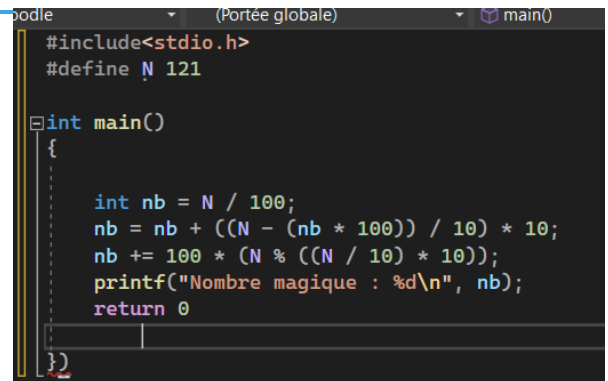
Exercice 2. De l'ordre!

Formatez le code suivant (retours à la ligne, indentation), compilez-le, puis exécutez-le. Quelle est son utilité? Pour vous aider, n'hésitez pas à modifier la valeur 121, recompilez puis lancez à nouveau le programme.

```
#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
;}
```



```
#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;
}
```

2.2 Affichage et saisie

Exercice 3. Les caractères d'espace.

Copiez le code suivant dans un main() et déduisez-en l'action des caractères \n et \t

```
printf("L1: Bonjour");
printf("L2: Bonjour\n");
printf("\tL3: Bonjour\n");
```

\n → retour à la ligne
\t → tab

Exercice 4. Le choix du formatage.

L'objectif de cet exercice est de comprendre l'action des caractères %c, %d, %f. Pour cela, on fournit le code suivant.

```
char c = 'A';
int a = 66;
float f = 12.3;
printf ("c=%c\n", c);
printf ("a=%d et f=%f\n", a, f);
```

Répondez aux questions ci-dessous, si besoin en modifiant et exécutant à nouveau le code présenté ci-dessus

- 1) Rédigez dans un `main()` le code ci-dessus.
- 2) Jetez un oeil à la table ASCII en réalisant une recherche Internet.
- 3) Déduisez-en l'utilité du premier `printf`.
- 4) À l'aide du second `printf`, décrivez la règle d'utilisation de cette fonction.

Exercice 5. Le choix du formatage, le retour.

Complétez le tableau suivant résumant les possibilités de formatage d'affichage.

à connaître

Formatage	Type de données
%d	int
%c	char
%f	float
%lf	double
%ld	long
%hd	short
%u	unsigned
%%	le caractère %

Exercice 6. Formats d'affichage.

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

```
#include <stdio.h>

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

73 73.0 73.0 73.0 I

Exercice 7. Saisie de valeurs.

L'objectif de cet exercice est de déduire le fonctionnement de la fonction `scanf`. On considère pour cela l'extrait de code suivant :

```
int a;
scanf ("%d", &a);
printf ("a = %d\n", a);
```

scanf("a=%d",&a);

→ l'utilisateur doit écrire "a=..."

Rédigez ce code dans un `main()` correctement formé puis focalisez votre attention sur la ligne comportant le `scanf()`.

- 1) Que dit le compilateur si l'on oublie le symbole `&` ?
- 2) Que se passe-t-il si l'on ajoute `\n` juste avant le second guillemet ?
- 3) Que se passe-t-il si l'on ajoute les caractères `a=` juste avant le symbole `%` ?
- 4) Modifiez le code pour que l'utilisateur soit obligé de saisir deux entiers séparés par un retour à la ligne.

Exercice 8. Rendu de monnaie.

Implémentez un programme qui calcule le rendu de monnaie. Pour cela, votre programme doit donner un montant à payer à l'utilisateur (prix de l'objet) et l'utilisateur doit entrer le montant qu'il donne. Le programme répond alors la monnaie à rendre tel que décrit dans l'exemple ci-dessous.

```
bodin@TeamIbijau:~$ ./monnaie
Montant a payer: 18,75 euros
Vous donnez: 20
Rendu: 1,25 euros
bodin@linux:~$
```

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

int main()
{
    float montant, a;

    printf("Combien devez-vous payer ?\n");
    scanf_s("%f", &a);
    printf("vous devez payer : %.2f\n", a);

    printf("montant donne : ");
    scanf_s("%f", &montant);

    montant -= a;
    printf("somme rendu = %.2f\n\n", montant);

    return 0;
}
```

2.3 *2 f* → 1.8 avec 2 chiffres après la virgule.

2.3 Détail de l'état de la mémoire

2.4 Taille des types standards

Exercice 9. Valeur maximale des entiers.

Sachant qu'un bit permet de stocker la valeur 0 ou 1, répondez aux questions suivantes.

- 1) Listez toutes les possibilités de valeurs binaires stockées sur deux bits.
- 2) De même avec trois bits.
- 3) Déduisez-en le nombre de valeurs qu'il est possible de stocker sur 8 bits soit 1 octet.
- 4) Complétez les cases vides du tableau ci-dessus.

1) 00 01 10 11

2) 000 001 010 011 100 101 111

3) $2^8 = 256$

Exercice 10. La mémoire.

Complétez l'état de la mémoire ci-contre à la fin de l'exécution du code ci-dessous.

```

int a=10, b=5;
char c,d,e,f='s';
a = b+2;
b += a;
c = d = f;
e = f-1;
a++;
b--;

```

Handwritten annotations on the code:

- ~~a = 10~~ b = 5
- ~~b = 5~~
- ~~a = 12~~
- ~~b = 22~~
- ~~c = 115 = d = 115 = e = 115~~
- ~~e = 114~~
- ~~a = 13~~
- ~~b = 21~~

Adresse	Nom	Valeur
0x0162		
0x0163		
0x0164		
0x0165		
0x0166	b	115
0x0167	e	114
0x0168	d	115
0x0169	c	115
0x016A		
0x016B	b	11
0x016C		
0x016D		
0x016E		
0x016F	a	8
0x0170		
0x0171		

Exercice 11. Échange de variables.

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

- 1) Rédigez un code permettant de récupérer deux entiers de la part de l'utilisateur. Stockez ces valeurs dans des variables a et b.
- 2) Écrivez un programme qui échange les valeurs des deux entiers a et b et affiche les variables après échange. Vous pouvez utiliser une variable temporaire tmp.
- 3) Écrivez maintenant un programme qui permute circulairement les valeurs de trois entiers a, b et c. Vous pouvez utiliser une variable temporaire tmp.
- 4) 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.

2.5 Forçage de type

Exercice 12. 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 avant de l'exécuter afin de vérifier.

```

#include <stdio.h>

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

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

```

$$a = 1 / (100 \times 150) = 6,67 \cdot 10^{-5}$$

$$y = 6,67 \cdot 10^{-5} \times 73 = 0,0049$$

Exercice 11

```

1 int main()
2 {
3     int a, b, c, i, n, tmp;
4
5     scanf("%d %d", &a, &b);
6     printf("a = %d, b = %d\n", a, b);
7
8     printf("Donnez la valeur de n :");
9     scanf("%d", &n);
10
11     // rotation circulaire
12     for(i = 0; i < n; i++)
13     {
14         tmp = a;
15         printf(" tmp = %d", tmp);
16
17         a = b;
18         printf(" a = %d", a);
19
20         b = tmp;
21         printf("b = %d\n", b);
22     }
23
24     // échange de a et de b sans tmp (addition)
25     a = a + b;
26     b = a - b;
27     a = a - b;
28     printf("a = %d, b = %d, c = %d\n", a, b, c);
29
30     // échange de a et de b sans tmp (XOR)
31     a = a ^ b;
32     b = a ^ b;
33     a = a ^ b;
34     printf("a = %d, b = %d, c = %d\n", a, b, c);
35
36     return 0;
37 }

```

Xor:

$$73 = 1001001$$

$$12 = 0001100$$

$$69 = 1000101$$

Exercice 12

```

1 #include<stdlib.h>
2
3 int main()
4 {
5
6     int a;
7     float x, y;
8
9     x = 100;
10    a = 1 / (x * 150);
11    y = a * 73.0;
12
13    printf(" y = %d\n", (int)y );
14
15    return 0;
16 }

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```

CH & TD EXERCICES F CORRECTION
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES/
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES$ ls
exercice11.c
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES$ touch exercice12.c
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES$ gcc exercice12.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES$ gcc exercice12.c
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
elin@LAPTOP-B705D891:/mnt/c/scolaire/ESIEA/Algo et prog en C/Chapitre 2/EXERCICES$

```

à refaire!

floatant / floatant
double / double
int

```
printf( "y = %d\n", (int)y );  
  
return 0;  
}
```

affiche la valeur de y

3 Pour aller plus loin

Exercice 13. Erreur de calcul flottant.

Recopiez le code suivant et exécutez-le. Le résultat de l'exécution devrait vous surprendre, du moins je l'espère. Comment l'expliquez-vous ? Détaillez pour cela le résultat de chacun des printf.

à voir

```
#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;  
}
```

Exercice 14. 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 $e^0 = 1$ et nous incrémentons l'exposant par pas de $h = 0.001$ en utilisant l'approximation précédente pour arriver jusqu'à $e^1 = e$. Le code suivant est une implémentation en C de cet algorithme.

```
#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`.

- 1) Exécutez ce code. La combinaison de touches « `Ctrl + c` » (ou « `Ctrl + z` ») vous sera probablement utile.
- 2) Expliquez ce qui pose problème puis corrigez le programme en conséquence.
- 3) Exécutez-le à nouveau.
- 4) Il est possible de montrer que notre approximation converge vers e quand h tend vers 0. Exécutez votre programme avec
`h = 1.0f, h = 0.1f, h = 0.01f, h = 0.001f` puis `h = 0.0001f`.
 Que constatez-vous ?
- 5) Exécutez maintenant le programme avec
`h = 0.00001f, h = 0.000001f` puis `h = 0.0000001f`.
 Comment expliquez-vous ces résultats ?