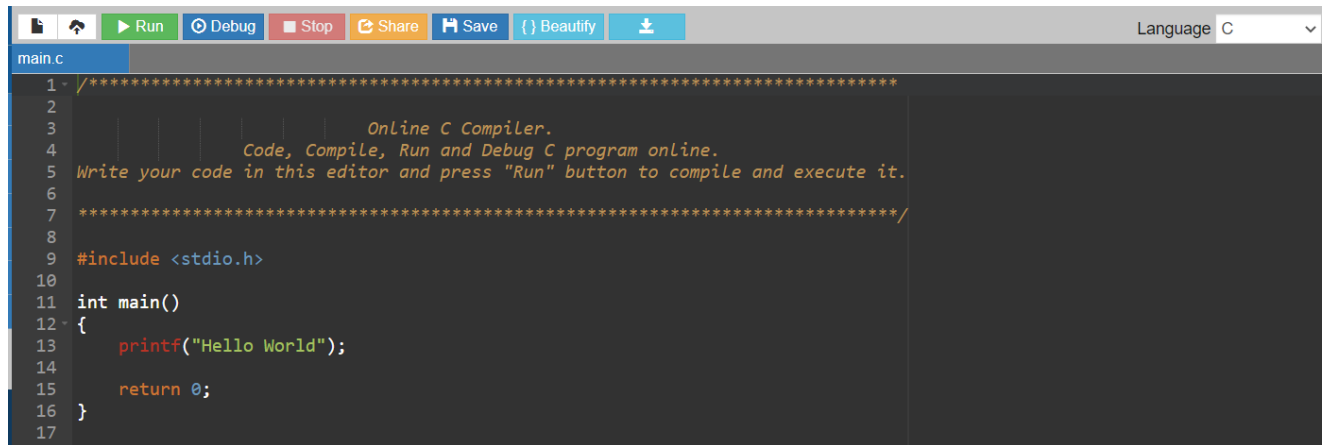


LC Bases

I/ Un premier programme

On utilisera dans un premier temps le compilateur en ligne proposé au lien suivant :

https://www.onlinegdb.com/online_c_compiler



```
1 - /*****
2
3      Online C Compiler.
4      Code, Compile, Run and Debug C program online.
5      Write your code in this editor and press "Run" button to compile and execute it.
6      *****/
7
8
9 #include <stdio.h>
10
11 int main()
12 {
13     printf("Hello World");
14
15     return 0;
16 }
17
```

Compiler et exécuter le programme en cliquant sur « Run ». Il doit s'afficher « Hello World » dans la console.

Quelques explications :

- **#include <stdio.h>** : permet d'inclure le module stdio.h. C'est proche du **import** en Python.
- **int main()** : c'est la fonction d'entrée d'un programme en C. La valeur de retour 0 indique que le programme s'est bien exécuté.
- **printf(« Hello World »)** : même rôle que la fonction print() en Python.

On remarquera les **accolades** permettant de délimiter les blocs et les **points virgules** à la fin de chaque instruction. Il n'y a pas d'indentation en C.

Les **commentaires** sont à écrire entre les caractères /* et */. On peut également utiliser // pour des commentaires sur une ligne.

On notera le caractère **\n** qui fait partie des *séquences d'échappement* qui permettent de représenter des caractères non imprimables ou difficiles à taper.

```
#include <stdio.h>

int main()
{
    printf("Hello\n");
    printf("World");

    return 0;
}
```

Résultat

Hello
World

Quelques *séquences d'échappement* en C :

- **\n** : saut de ligne,
- **\t** : tabulation,
- **\b** : retour en arrière (*backspace* en anglais),
- **\«** : guillemet,
- **** : antislash (*backslash* en anglais)

```
#include <stdio.h>

int main()
{
    printf("Hello\b");
    printf("World");

    return 0;
}
```

Résultat

HellWorld

On a bien l'effet du backspace.

1/ **Modifier** le code pour insérer une tabulation entre Hello et World.

II/ Variables, calculs et boucles

1/ Variables et calculs

Le programme suivant se sert de la formule $C = 5/9*(F-32)$ qui convertit des degrés Fahrenheit en degrés Celsius.

2/ Recopier et exécuter le programme suivant :

```
int main()
{
    // Définition des variables (type)
    int fahr, celsius;
    int mini, maxi;

    // Possibilité de déclarer une variable
    // et de lui affecter une valeur
    int intervalle = 20;

    mini = 0;
    maxi = 300;

    fahr = mini;
    while (fahr <= maxi) {
        celsius = (5.0/9.0)*(fahr - 32);
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + intervalle;
    }

    return 0;
}
```

Résultat

0	-17
20	-6
40	4
60	15
80	26
100	37
120	48
140	60
160	71
180	82
200	93
220	104
240	115
260	126
280	137
300	148

En C, il faut **déclarer** les variables avec leur type. On peut citer : **char** (un octet, caractère), **short** (nombre entier court), **int** (nombre entier), **long** (nombre entier long), **float** (nombre à virgule flottante), **double** (nombre à virgule flottante en double précision).

Contrairement à Python, une **variable conserve son type**. Pour modifier son type, le programmeur doit convertir la variable (conversion explicite). On remarquera que les degrés Celsius, bien que non entiers par le calcul, s'affichent en nombre entier car la variable *celsius* a un type *int*.

3/ Il peut toutefois y avoir des conversions implicites voire non voulues en C : il faut donc être vigilant !

Remplacer l'instruction « *celsius = (5.0/9.0)*(fahr - 32);* » par « *celsius = (5/9)*(fahr - 32);* ». Que se passe-t-il à l'exécution ? **Proposer** une explication.

Quelques explications pour l'instruction : `printf("%d\t%d\n", fahr, celsius);`

Le premier argument de la fonction *printf()* est une chaîne de caractères (reconnaissable grâce aux guillemets), dans laquelle chaque % indique l'endroit où l'un des arguments suivants (le deuxième, le troisième ...) doit se substituer et sous quelle forme il faut l'afficher.

Par exemple, *%d* provoque l'affichage de valeurs des deux entiers *fahr* et *celsius*, séparés par une tabulation (\t).

Voici quelques commandes supplémentaires associées au % : *%c* affiche un **caractère**, *%d* affiche un **entier**, *%6d* affiche un entier sur une largeur minimum de 6 caractères, *%f* affiche un **flottant**, *%6f* affiche un flottant sur une largeur minimale de 6 caractères, *%.2f* affiche un flottant avec deux chiffres après la virgule et *%6.2f* affiche un flottant sur au moins 6 caractères de large avec 2 chiffres après la virgule.

4/ **Modifier** le programme pour que les degrés Celsius s'affichent en nombre flottants avec 3 chiffres après la virgule.

A noter : Les **symboles** des **opérations arithmétiques** sont les mêmes en C qu'en Python. En revanche, les opérateurs logiques **and** et le **or** s'écrivent respectivement **&&** et **||** en C.

2/ Boucles *while* et *for*

La boucle *while* en C est exactement la même qu'en Python : on retrouve le système **d'accolades** au lieu de l'indentation et la condition à vérifier est entre parenthèses. Pas de difficulté majeure donc 😊.

Voici le programme précédent modifié. Comme toujours, il existe plusieurs méthodes pour résoudre un même problème donné.

L'utilisation de la boucle *for* est différente de celle en Python :

- le **compteur doit être déclaré**,
- c'est une **condition non vérifiée** qui stoppe la boucle (ici *fahr* <= 300) et non une valeur finale.

```
#include <stdio.h>

int main()
{
    int fahr; // Déclaration du compteur

    for(fahr = 0; fahr <= 300; fahr += 20) {
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-20));
    }
}
```

Attention : il faut être vigilant car une boucle *for* peut devenir infinie en C si la condition est toujours vérifiée. On veillera donc à utiliser des **compteurs entiers**, les seuls nombres à être exactement représentés en informatique.

Remarque : on notera l'absence de point-virgule après le *while* et le *for*. Ce sera toujours le cas s'il y a un bloc d'instructions après (délimités par des accolades).

5/ **Modifier** le programme pour que le programme affiche les conversions à l'envers, c'est-à-dire de 300 degrés à 0.

3/ Constantes symboliques

Ce n'est pas une bonne habitude de rentrer des nombres qui semblent sortis de nulle part (ici, le 0 ; 300 et 20 dans la boucle *for*). Hors contexte, on ne sait rapidement plus à quoi ils se rapportent.

Le mot clé **#define** permet d'**associer** une **valeur** à un texte, que l'on écrit en majuscules par convention.

```
#include <stdio.h>

#define MINI 0 // Borne inférieure de la table
#define MAXI 300 // Borne supérieure de la table
#define INTER 20 // Intervalle entre chaque valeur convertie

// Copie et affichage des caractères en entrée
int main()
{
    int fahr;

    for(fahr = MINI; fahr <= MAXI; fahr += INTER)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr - 32));
}
```

Ces constantes ne sont pas des variables : elle n'ont donc pas besoin d'être définies à l'avance. D'ailleurs, le mot clé **#define** peut être utilisé pour tout remplacement souhaité, pas seulement un nombre.

```
// Un grand classique en C
#define FOREVER for(;;)
```

Remarque sur les boucles : s'il n'y a qu'une instruction après une boucle (ou une instruction conditionnelle, on n'est pas obligé de mettre les accolades (boucle *for* dans l'exemple).

III/ Structures de contrôles

1/ L'instruction *if – else if – else*

Cette instruction conditionnelle est très proche syntaxiquement parlant de celle en Python. Cela sera donc simple à comprendre.

Voici sa construction et un exemple de programme :

```
if ( <expr1> )  
    <bloc1>  
else if ( <expr2> )  
    <bloc2>  
else if ( <expr3> )  
    <bloc3>  
else if ( <exprN> )  
    <blocN>  
else <blocN+1>
```

Là encore, les *accolades* ne sont pas obligatoires s'il n'y a qu'une instruction après la condition.

```
#include <stdio.h>  
  
int main()  
{  
    int A,B;  
    printf("Entrez deux nombres entiers :");  
  
    // Récupère les entiers entrés séparés par un espace  
    scanf("%i %i", &A, &B);  
  
    if (A > B)  
        printf("%i est plus grand que %i\n", A, B);  
    else if (A < B)  
        printf("%i est plus petit que %i\n", A, B);  
    else  
        printf("%i est égal à %i\n", A, B);  
  
    return 0;  
}
```

A l'exécution

```
Entrez deux nombres entiers :6 10  
6 est plus petit que 10
```

A noter : la fonction **scanf()** permet de récupérer des valeurs entrées au clavier et de les affecter à une ou plusieurs variables. **Il faut bien séparer les valeurs par un espace.**

6/ **Ecrire** et **tester** ce programme. Que se passe-t-il si l'on met autre chose qu'un nombre entier ?

7/ **Ecrire** un programme permettant d'attribuer le prix d'un billet de cinéma en fonction de l'âge de la personne.

Tarifs du cinéma :

- Moins de 12 ans : 5 euros
- Entre 12 et 18 ans : 6.5 euros
- Plus de 18 ans : 9 euros

2/ L'opérateur conditionnel

Il s'agit d'une solution élégante à l'instruction classique *if – else*. Elle ne fonctionne que dans des cas basiques, avec une seule instruction après un test.

Voici sa structure :

```
<expr1> ? <expr2> : <expr3>
```

Fonctionnement : si l'expression 1 est vraie (ne vaut pas zéro), alors la valeur fournie sera *expr2*, sinon, ce sera *expr3*.

Les deux programmes suivants sont équivalents :

```
#include <stdio.h>

int main()
{
    float A,B,C;
    printf("Entrez deux nombres :");

    // Récupère les entiers entrés séparés par un espace
    scanf("%f %f", &A, &B);

    if (A >= B)
        C = A;
    else
        C = B;

    printf("La plus grande valeur est %f", C);

    return 0;
}
```

```
#include <stdio.h>

int main()
{
    float A,B,C;
    printf("Entrez deux nombres :");

    // Récupère les entiers entrés séparés par un espace
    scanf("%f %f", &A, &B);

    C = (A >= B) ? A : B;

    printf("La plus grande valeur est %f", C);

    return 0;
}
```

8/

Que **fait** le programme à droite ?

Remarque : on ne testera pas le programme.

```
#include <stdio.h>

int main()
{
    int A;
    printf("Entrez un nombre entier :");

    // Récupère les entiers entrés séparés par un espace
    scanf("%i", &A);

    printf("Vous avez %i carte%c", A, (A <= 1) ? ' ' : 's');

    return 0;
}
```

9/ **Ecrire** un script avec un opérateur conditionnel permettant de déterminer si une personne est mineure ou majeure (en fonction de son âge).

3/ L'instruction **switch - case**

Cette instruction est une variante de l'instruction conditionnelle qui permet d'effectuer un bloc en fonction de la **valeur** d'une variable et non d'une condition.

```
switch(variable) {
    case valeur : instructions
        break ;
    case valeur : instructions
        break ;
    case valeur : instructions
        break ;
    default : instructions
}
```

A noter l'instruction **break** obligatoire qui permet de sortir du bloc sinon le programme effectuera les autres cas qui suivent!

```
#include <stdio.h>

int main() {
    char operation;
    float n1, n2;

    printf("Entrer une opération (+, -, *, /): ");
    scanf("%c", &operation);
    printf("Entrer deux nombres : ");
    scanf("%f %f", &n1, &n2);

    switch(operation)
    {
        case '+':
            printf("%f + %f = %f", n1, n2, n1+n2);
            break;
        case '-':
            printf("%f - %f = %f", n1, n2, n1-n2);
            break;
        case '*':
            printf("%f * %f = %f", n1, n2, n1*n2);
            break;
        case '/':
            printf("%f / %f = %f", n1, n2, n1/n2);
            break;
        // Mauvais opérateur
        default:
            printf("Erreur! l'opérateur est incorrect");
    }

    return 0;
}
```

10/ **Ecrire** un programme indiquant que parmi 4 directions au choix (N , S , O et E), seules N et E sont gagnants.

Ressources :

- *Le langage C, seconde édition, norme ANSI de Kernighan et Ritchie (Dunod)*
- https://www.itam.lu/Tutoriel_Ansi_C/prq-c_c.htm
- <https://www.programiz.com/c-programming/c-switch-case-statement>