

## Partie 3 : Premier pas en C

- ▶ Du source à l'exécutable
- ▶ Variables
- ▶ Fonctions
- ▶ Bonnes habitudes de programmation

## Du source à l'exécutable

```
/*1er programme source*/  
  
#include <stdio.h> /*pour utiliser le printf */  
int main(void)  
{  
    printf("Hello, World!\n");  
    EXIT_SUCCESS;  
}
```

## Du source à l'exécutable

- ▶ Un programme C contient des définitions de fonctions.  
L'une d'elles s'appelle **main**.
- ▶ Les instructions sont exécutées séquentiellement à partir de la 1<sup>ère</sup> instruction de main.
- ▶ Le ; (point-virgule) est un terminateur d'instruction.
- ▶ Les commentaires sont placés entre /\* et \*/
- ▶ Le programmeur indique les groupes d'instructions avec des accolades {...} (l'indentation aide juste à la lisibilité).
- ▶ Un programme doit être traduit (compilé) avant d'être exécuté.

## Du source à l'exécutable

- ▶ Contrairement à python, C est un langage **compilé**.
- ▶ Cela signifie qu'un programme C ne peut pas être exécuté tel quel.
- ▶ Il est nécessaire de le traduire en langage machine.
- ▶ La phase de traduction

du langage C  
vers  
le langage machine

s'appelle la **compilation**.

- ▶ La traduction est faite par un logiciel : le compilateur



## Du source à l'exécutable



- ▶ Une compilation réussie crée un fichier : l'**exécutable** (l'application).
- ▶ Il dépend de la machine sur laquelle on veut exécuter le programme.
  - on doit compiler une fois par architecture d'utilisation.
- ▶ Si le compilateur rencontre une erreur,
  - Il continue pour rechercher d'autres erreurs.
  - Il ne **créé pas** l'exécutable.

AlgoProg 2015-16 24

## Du source à l'exécutable

- ▶ Avant d'exécuter le programme, on doit le traduire pour créer un fichier exécutable. Cette traduction s'appelle la **compilation**.
- ▶ On utilise le **compilateur** (= logiciel de traduction) **gcc**.

```
>>> gcc -Wall -std=c99 -o HelloWorld HelloWorld.c
```

compile le fichier `HelloWorld.c` dans lequel se trouve le source, et crée l'exécutable `HelloWorld`.

`-Wall` : gcc donne les principaux avertissements

`-std=c99` : vérifie la conformité à la norme C99

`-o` : permet de choisir le nom de l'exécutable

```
>>> ./HelloWorld exécute le fichier compilé
```

L'utilisation de l'exécutable ne nécessite plus le source.

AlgoProg 2015-16 25

## Options importantes de gcc

- `-o` permet de nommer le fichier exécutable (`a.out` par défaut).
- `-Wall` indique des avertissements sur le code.
- `-Werror` produit une erreur à la place d'un avertissement.
- `-std=c99` se conforme à la norme C99.

Ces options aident le programmeur à détecter les erreurs. On compilera systématiquement avec

```
-std=c99 -Wall -Werror
```

Autre option utile :

- `-g` permet d'utiliser ultérieurement le debugger

AlgoProg 2015-16 26

## Variables

- ▶ Contrairement à Python, on ne peut pas utiliser les variables avant de les avoir **déclarées**.
- ▶ Une déclaration permet entre autres de donner le **type** des variables.
- ▶ On utilisera au début des variables entières, de type **int**.

Exemples de **déclaration** :

```
int x ;      déclaration d'une variable de nom x et de type entier.
```

```
int x, y ;   déclaration de deux variables de noms x et y et de type entier.
```

AlgoProg 2015-16 27

## Variables : exemple

```
int pair(int n){
    int p=0;
    int i=1;
    if (n%2==0) {
        printf(`Pair`);
        return p; }
    else {
        printf(`Impair`);
        return i;}
}
```

AlgoProg 2015-16 28

## La fonction d'affichage : printf

```
#include<stdio.h>
/*pour la déclaration de printf*/

printf ("Hello world !\n" );

>>>Hello world !
>>>
```

Chaine de caractères

Retour à la ligne

AlgoProg 2015-16 29

## La fonction d'affichage : printf

```
printf("%d\n",x);

printf("La variable x vaut %d\n",x);

printf("%d * %d = %d\n", x, x, x*x);
```

AlgoProg 2015-16 30

## La fonction d'affichage : printf

```
Si x vaut 3
printf("%d\n",x);
>>> 3
>>>
printf("La variable x vaut %d\n",x);
>>> La variable x vaut 3
>>>
printf("%d * %d = %d\n", x, x, x*x);
>>> 3*3=9
>>>
```

AlgoProg 2015-16 31

## Exemple de fonction :

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

/* placer ici les definitions des fonctions */

int
main(void) {

/* placer ici les appels aux fonctions*/

return EXIT_SUCCESS;
}
```

AlgoProg 2015-16 32

## Exemple de fonction :

```
#include <stdio.h>
#include <stdlib.h>
int perimetreRectangle(int l,int h)
{
return (2*(l+h));
}
int main(void){
int a = 5;
int b = 7;
int p = perimetreRectangle(a,b);
printf("Le perimetre du rectangle de longueur %d et de
hauteur %d est %d\n", a, b, p);

return EXIT_SUCCESS;
}
```



Ne pas confondre **définition** et **appel** des fonctions

AlgoProg 2015-16 33

## Bonnes habitudes de programmation

- ▶ La compilation doit se faire **sans erreur ni avertissement (warning)**.
- ▶ Une mauvaise indentation n'est pas sanctionnée (≠ python). Les programmes doivent cependant être **correctement indentés**.
- ▶ Les noms des fonctions, variables, etc. doivent être
  - **lisibles**,
  - **pertinents**
  - **cohérents** (conventions **uniformes**).
- ▶ Les programmes doivent être **commentés**.

AlgoProg 2015-16 34