

# Programmation Langage C

Youssef ALJ

8 mars 2020

- 1 Concepts de base
  - Phases de création d'un programme C
- 2 Variables et types de bases - entrée/sortie
  - Variables et types de base
  - Lecture et écriture
    - la fonction printf
    - la fonction scanf
- 3 Structures de contrôle
  - if ... else
  - Opérateurs de test
  - Division euclidienne
  - switch ... case
- 4 Les boucles
  - Boucle for
  - Boucle do while
  - Boucle while
  - Break et Continue
  - Chaines de caractères en C

## Pré-requis :

- Initiation au langage C (cf cours du premier semestre).
- Mathématiques de base (Terminale S : un peu d'arithmétique).

## Organisation :

- 15 % participation (compte rendu de TP).
- 15 % contrôle continu.
- 70 % examen

- 1 Concepts de base
  - Phases de création d'un programme C
- 2 Variables et types de bases - entrée/sortie
  - Variables et types de base
  - Lecture et écriture
    - la fonction printf
    - la fonction scanf
- 3 Structures de contrôle
  - if ... else
  - Opérateurs de test
  - Division euclidienne
  - switch ... case
- 4 Les boucles
  - Boucle for
  - Boucle do while
  - Boucle while
  - Break et Continue
  - Chaines de caractères en C

- Le langage C a été développé aux laboratoires AT&T dans les années 1970 par Dennis Ritchie.

Le cycle de création d'un programme en langage C est le suivant :

- 1 Concevoir un algorithme.
- 2 Utiliser un éditeur pour écrire le code source.
- 3 Compilation à partir du code source.
- 4 Édition des liens.
- 5 Éventuellement corriger les erreurs de compilation.
- 6 Exécuter le programme et le tester.
- 7 Éventuellement corriger les bugs.
- 8 Éventuellement Recommencer depuis le début.

On n'a pas besoin de mémoriser ces étapes. Ceci viendra avec la pratique.

# Étape 1 : Créer un algorithme

Exemple : on veut écrire un algorithme qui affiche "bonjour tout le monde" à l'écran.

Début

Variables :

Écrire("Bonjour tout le monde")

Fin

## Étape 2 : Saisie et sauvegarde du programme

- Avec un éditeur (par exemple notepad++) on saisit le texte suivant :

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("bonjour tout le monde");
6      return 0;
7  }
```

- On sauvegarde (souvent avec les touches CTRL-s).
- Durant la sauvegarde, on fait attention à ce que le nom du fichier finisse par '.c' pour qu'il soit reconnu comme étant un programme C.
- Y a-t-il des mots qui vous sont familiers?

Le compilateur est un programme spécial qui :

- lit les instructions enregistrées dans le code source "bonjour.c"
- analyse chaque instruction.
- traduit ensuite l'information en langage machine compréhensible par le micro-processeur de l'ordinateur.



## Étape 3 : Compilation (suite)

- Il existe plusieurs compilateurs :
  - gcc : GNU Compiler Collection.  
GNU : acronyme récursif qui veut dire GNU's Not Unix.
  - Le compilateur : Microsoft Visual Studio.
  - ...
- On va utiliser le compilateur gcc. Voir TP.
- On compile en ligne de commandes windows (ou linux) via la commande suivante : `>gcc -c bonjour.c -o bonjour.o`
- Le résultat est la création d'un nouveau fichier appelé fichier objet qui a comme extension '.o' et qui a comme nom bonjour.o.

## Étape 4 : Édition des liens

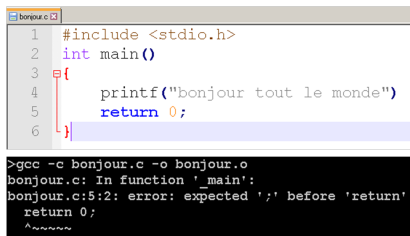
L'éditeur des liens (en anglais le linker) permet de :

- faire le liens entre les différents fichiers ".o"
- cette étape de link est aussi faite en utilisant gcc.  
`>gcc bonjour.o -o bonjour`
- on reviendra plus en détails sur le fonctionnement du linker dans les prochaines séances.

- Plusieurs compilateurs font la compilation et l'édition des liens en même temps.
- Cela se fait avec la commande suivante :  
`>gcc bonjour.c -o bonjour`

# Étape 5 : Erreurs de compilation

- Plusieurs erreurs de compilations peuvent apparaître.
- Elles sont dues par exemple à une mauvaise syntaxe (oubli du " ;", mot réservé mal écrit, etc.).
- Il faudra les corriger en éditant le fichier source, sauvegarder et recompiler puis ré-exécuter.

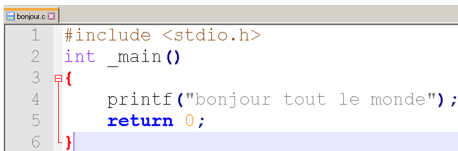


```
1 #include <stdio.h>
2 int main()
3 {
4     printf("bonjour tout le monde")
5     return 0;
6 }
```

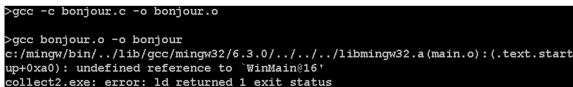
```
>gcc -c bonjour.c -o bonjour.o
bonjour.c: In function '_main':
bonjour.c:5:2: error: expected ';' before 'return'
    return 0;
    ^~~~~~
```

# Étape 5 : Erreurs de link

- Elles sont dues au fait que le linker n'arrive pas à faire le lien entre les fonctions définies par les programmes écrits.
- Une fonction principale doit toujours exister qui sert de point d'entrée aux autres fonctions.
- cette fonction principale est appelée la fonction main.

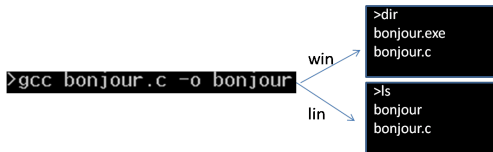


```
1 #include <stdio.h>
2 int _main()
3 {
4     printf("bonjour tout le monde");
5     return 0;
6 }
```



```
>gcc -c bonjour.c -o bonjour.o
>gcc bonjour.o -o bonjour
c:/mingw/bin/./lib/gcc/mingw32/6.3.0/./../libmingw32.a(main.o): (.text.startup+0xa0): undefined reference to `WinMain@16'
collect2.exe: error: ld returned 1 exit status
```

# Étape 6 : Exécuter



- Si la compilation s'est effectuée sans erreur on verra nouveau fichier qui apparait.
- Ce fichier est appelé un exécutable.
- Sous Windows, ce fichier exécutable porte l'extension '.exe' (voir figure ci-dessus).
- Pour lancer le fichier exécutable, dans notre exemple, on saisit en ligne de commande Windows : `>bonjour`.
- Dans un système d'exploitation de type Unix on saisit : `./bonjour`

## Étape 7 : Correction des bugs à l'exécution

- On utilise un programme appelé gdb.
- On reviendra sur la correction des bugs dans les prochaines séances.

# Synthèse des principales étapes :

1

Edition

```
bonjour.c
1 #include <stdio.h>
2 int main()
3 {
4     printf("bonjour tout le monde");
5     return 0;
6 }
```

2

Compilation  
+  
Edition des liens

```
>gcc bonjour.c -o bonjour
```

win

lin

```
>dir
  bonjour.exe
  bonjour.c
```

```
>ls
  bonjour
  bonjour.c
```

3

Exécution

win >bonjour → bonjour tout le monde

lin >./bonjour → bonjour tout le monde



# Commentaires en C

```
1  #include <stdio.h>
2  // ceci est un commentaire
3
4  int main()
5  {
6      // un autre commentaire
7
8      /*
9      Une autre facon
10     d ecrire un commentaire sur plusieurs lignes
11     */
12     printf("bonjour tout le monde");
13     return 0;
14 }
```

- Les commentaires sont un outil de base pour documenter son programme.
- Cette documentation n'est pas utile seulement pour les autres mais pour vous aussi.
- Les commentaires améliorent la lisibilité du code écrit.
- Sans commentaire la lecture est difficile.
- Il est donc recommandé d'ajouter des commentaires autant que possible.

Toujours par souci de lisibilité il est recommandé respecter les règles d'indentation :

- Décaler d'un cran (= trois espaces) vers la droite tout bloc inclus dans un précédent.
- Cela permet de repérer qui dépend de quoi et si tous les blocs ouverts sont

# Exemple d'un programme C mal écrit

```
1  #include <stdio.h>
2
3  void main()
4  {
5      int a = 0;
6      if (a == 0)
7
8
9      {
10     printf("a est egal a 0");
11     }
12     else
13     {
14     printf ("a est different de 0");
15     }
16 }
```

# Exemple d'un programme C mieux écrit

```
1  #include <stdio.h>
2  // fonction main indispensable pour chaque programme C
3  void main()
4  {
5      int a = 0;
6      // si a est nul on affiche : "a est nul"
7      if (a == 0)
8      {
9          printf("a est egal a 0");
10     }
11     // sinon on affiche : "a n est pas nul"
12     else
13     {
14         printf ("a est different de 0");
15     }
16 }
```

- 1 Concepts de base
  - Phases de création d'un programme C
- 2 Variables et types de bases - entrée/sortie
  - Variables et types de base
  - Lecture et écriture
    - la fonction printf
    - la fonction scanf
- 3 Structures de contrôle
  - if ... else
  - Opérateurs de test
  - Division euclidienne
  - switch ... case
- 4 Les boucles
  - Boucle for
  - Boucle do while
  - Boucle while
  - Break et Continue
  - Chaines de caractères en C

# Manipulation d'informations

- Quand on programme avec n'importe quel langage de programmation on veut utiliser des données.
- On veut stocker ces données dans ce qu'on appelle des variables.
- On veut programmer un jeu vidéo.
- On a un joueur de ce jeu vidéo qui veut manipuler un personnage.
- Ce personnage va avoir une position dans une carte.
- Cette position va changer en fonction du mouvement du personnage.
- On veut stocker cette information pour pouvoir interagir avec d'autres éléments du jeu.

Il y en a six : **void**, **int**, **char**, **float**, **double**, **long double**.

- **void** : c'est le type vide. Il est surtout utilisé pour définir les fonctions sans arguments ou sans valeur de retour.
  - **int** : c'est un type qui se décline sous plusieurs formes.
    - **short** : un entier court.
    - **long** : un entier long.
    - **signed** pour dire que l'entier qu'on manipule peut être positif ou négatif.
    - **unsigned** pour dire que l'entier qu'on manipule est uniquement positif
- N.B. Si on n'indique rien, le qualificatif **signed** est appliqué.



# Les types de bases

- **char** : ce type permet de stocker les caractères. Il peut aussi représenter les entiers sur 8 bits. On, peut aussi trouver un char signé **signed char** ou non signé **unsigned char**.
- **float** : ce type permet de représenter les réels.
- **double** : même chose que **float** mais avec une précision plus importante.
- **long double** : pareil que **double** mais avec une précision encore plus grande.

- La fonction `printf` est très utile car permet d'afficher des messages et les valeurs des variables.

- Exemple :

```
1 printf("Bonjour tout le monde!");  
2 printf("%d kilogramme vaut %d grammes", 1, 1000):
```

- L'argument de la fonction `printf` dit **format** est une chaîne de caractère qui détermine ce qui sera affiché par `printf` et sous quelle forme.
- Dans l'exemple 1 c'est "Bonjour tout le monde!".

# Exemple

```
1 printf("Bonjour tout le monde!");  
2 printf("%d kilogramme vaut %d grammes", 1, 1000):
```

- Dans l'exemple 2 c'est "%d kilogramme vaut
- Dans l'exemple 2 : la chaîne de caractère qu'on veut afficher est composée d'un texte normal et de séquence de contrôle permettant d'inclure des variables.
- le premier %d sera donc remplacé par la valeur 1.
- le deuxième %d sera remplacé par la valeur 1000.
- On aura comme résultat en sortie l'affichage suivant :  
"1 kilogramme vaut 1000 grammes".

Les séquences de contrôle commencent par le caractère % suivi d'un caractère parmi les suivants :

- d ou i pour afficher un entier signé.
- f pour afficher un réel (float ou double).
- c pour afficher comme un caractère.
- s pour afficher une chaîne de caractères.

- Cette fonction permet de lire des données formatées à partir de l'entrée standard (clavier).
- `scanf` utilise les mêmes formats que `printf` mais on fait précéder le nom de la variable du caractère `&`.

```
1 scanf("%d",&i);
```

- seul le format est passé en paramètre.
- Il ne faut ajouter de message ni aucun autre caractère.

# Exemple

```
1  /* Exemple pour tester "scanf" */
2  #include <stdio.h>
3  int main () {
4      int nb1 ;
5      float nb2 ;
6      printf("Saisissez une valeur entiere (positive ou
           negative) pour nb1 : ") ;
7      scanf("%d",&nb1) ;
8      printf("Saisissez une valeur reelle pour nb2 : ") ;
9      scanf("%f", &nb2) ;
10     printf("nb1 vaut %d ; nb2 vaut %f\n", nb1, nb2) ;
11     return 0;
12 }
```

- Éditer, sauvegarder puis compiler ce code.
- Faites en sorte que le programme affiche "hello monde" après avoir affiché les valeurs saisies par l'utilisateur.

- 1 Concepts de base
  - Phases de création d'un programme C
- 2 Variables et types de bases - entrée/sortie
  - Variables et types de base
  - Lecture et écriture
    - la fonction printf
    - la fonction scanf
- 3 Structures de contrôle
  - if ... else
  - Opérateurs de test
  - Division euclidienne
  - switch ... case
- 4 Les boucles
  - Boucle for
  - Boucle do while
  - Boucle while
  - Break et Continue
  - Chaines de caractères en C

- instruction1 n'est réalisée que si la condition est réalisée.

```
1  if (condition){  
2      instruction1;  
3  }
```



# Exemple

```
1 void main()  
2 {  
3     int var;  
4     printf("veuillez saisir un entier\n");  
5     scanf("%d", &var);  
6     if(var >= 0){  
7         printf("le nombre saisi est positif");  
8     }  
9 }
```

# if ... else

```
1  if (condition){  
2      instruction1;  
3  }  
4  else{  
5      instruction2;  
6  }
```

- On exécute instruction1 si la condition est réalisée.
- Sinon on exécute instruction2.

# Exemple

```
1 void main()
2 {
3     int var;
4     printf("veuillez saisir un entier\n");
5     scanf("%d", &var);
6     if(var >= 0){
7         printf("le nombre saisi est positif");
8     }
9     else{
10        printf("le nombre saisi est negatif");
11    }
12 }
```

# if ... else if ... else

```
1  if (condition1){
2      instruction1;
3  }
4  else if (condition2){
5      instruction2;
6  }
7
8  else if (condition3){
9      instruction3;
10 }
11
12 else{
13     instruction_par_defaut;
14 }
```

- Si condition1 est vérifiée on exécute instruction1.
- Si condition2 est vérifiée on exécute instruction2.
- Si condition3 est vérifiée on exécute instruction3.
- Si aucune des conditions n'est vérifiée alors on exécute instruction par défaut.

Opérateur	Signification
==	test d'égalité
!=	test de différence
>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
&&	ET logique
	OU logique
!	NOT logique
^	XOR logique

$A$	$B$	$A \&\& B$	$A    B$	$A \oplus B$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

# Que fait le programme suivant?

```
1 // cible.c
2 #define CIBLE_1 1000
3 #define CIBLE_2 100
4 #include <stdio.h>
5 #include <math.h>
6 int main() {
7     float x, y;
8     int danslemille, dehors, total_points=0;
9     printf("x ? "); scanf("%f", &x); printf("x = %.2f\n", x)
10    ;
11    printf("y ? "); scanf("%f", &y); printf("y = %.2f\n", y)
12    ;
13    float d = sqrt(x*x + y*y);
14    danslemille = (d < 1);
15    dehors = (d > 3);
16    if (danslemille) total_points = CIBLE_1;
17    else if (!dehors) total_points = CIBLE_2;
18    printf("total points = %d\n", total_points);
19    return (0);
20 }
```

## Théorème de la division euclidienne

Pour tout  $(a, b) \in \mathbb{N} \times \mathbb{N}^*$ , il existe un unique couple  $(q, r) \in \mathbb{N} \times \mathbb{N}$  tel que :

$$a = bq + r \quad \text{avec } 0 \leq r < b$$

## Exemple

Le reste de la division euclidienne de tout entier  $a$  par 2 est soit 0 soit 1.

## Conséquence immédiate

- un entier  $a$  est pair si le reste de division euclidienne de  $a$  par 2 est 0. Il est impair sinon.



- le quotient de la division euclidienne d'un entier  $a$  par un entier  $b$  est obtenu avec l'opérateur `/`.
- le reste est obtenu avec `%`.

# Exemple

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int a,b,q,r;
6      printf("veuillez saisir a\n");
7      scanf("%d",&a);
8      printf("veuillez saisir b\n");
9      scanf("%d",&b);
10     // calcul du quotient de a par b
11     q = a/b;
12     // calcul du reste de a par b
13     r = a%b;
14     printf("Le quotient est q=%d\n", q);
15     printf("Le reste est r=%d\n", r);
16     return 0;
17 }
```

- On veut vérifier si un nombre donné est multiple de 3.
- On veut vérifier si un nombre donné est pair ou pas.

# switch ... case

- Une solution afin d'éviter les imbrications des instructions if.
- Si variable prend valeur1 on exécute instruction10 et instruction11.

```
1      switch (variable){  
2      case valeur1:  
3          instruction10;  
4          instruction11;  
5          break;  
6      case valeur2:  
7          instruction20;  
8          instruction21;  
9          break;  
10     default :  
11         instruction_par_defaut;  
12     }
```

- 1 Concepts de base
  - Phases de création d'un programme C
- 2 Variables et types de bases - entrée/sortie
  - Variables et types de base
  - Lecture et écriture
    - la fonction printf
    - la fonction scanf
- 3 Structures de contrôle
  - if ... else
  - Opérateurs de test
  - Division euclidienne
  - switch ... case
- 4 Les boucles
  - Boucle for
  - Boucle do while
  - Boucle while
  - Break et Continue
  - Chaines de caractères en C

# Structure de la boucle pour

La boucle pour est constituée :

- d'initialisation exécuté **avant toutes les itérations**.
- de condition de boucle exécuté **avant chaque itération**.
- d'instruction de fin de boucle (souvent une incrémentation ou une décrémentation) exécuté **après chaque itération**.

```
1  for (initialisation; condition; incrementation)
2  {
3      instructions_a_repeter;
4  }
```

# Exemple 1

```
1  #include <stdio.h>
2  int main()
3  {
4      int i ;
5      for (i = 0 ; i < 10 ; i = i + 1)
6      {
7          printf ("iteration %d \n", i) ;
8      }
9      printf ("valeur de i apres la boucle : %d \n", i) ;
10     return 0 ;
11 }
```

## Exemple 2

On veut afficher les nombres pairs de 0 jusqu'à 10.



## Exemple 2

On veut afficher les nombres pairs de 0 jusqu'à 10.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      for (int i = 0; i <= 10; i++)
6      {
7          if (i%2 == 0)
8          {
9              printf("i=%d\n", i);
10             }
11         }
12     return 0;
13 }
```

# Initialisation

On peut initialiser la variable  $i$  à l'intérieur ou à l'extérieur de la boucle

```
1
2     for (int i = 0; i <= 10; i++)
3     {
4         // instructions qu on veut repeter
5     }
6     printf("i=%d\n", i);
```

ou

```
1
2     int i;
3     for (i = 0; i <= 10; i++)
4     {
5         // instructions qu on veut repeter
6     }
7     printf("i=%d\n", i);
```

Quelle est la différence entre les deux ?

# Incrémentation

On peut voir deux styles pour incrémenter dans la boucle.

```
1
2
3
4     for (int a =0; a<3; ++a)
5     {
6         printf("a=%d\n", a);
7     }
```

ou

```
1     for (int a =0; a<3; a++)
2     {
3         printf("a=%d\n", a);
4     }
```

Quelle différence y a-t-il?

# Incrémentation (suite)

Qu'affiche le programme suivant ?

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i=0,j=0;
6
7      int k,l;
8      k = ++i;
9      l = j++;
10     printf("i=%d\n",i);
11     printf("j=%d\n",j);
12     printf("k=%d\n",k);
13     printf("l=%d\n",l);
14     return 0;
15 }
```

# Boucle "faire tant que"

- Sert à répéter des instructions dans lesquelles la condition est vérifiée à la fin.
- On est donc sûr d'exécuter au moins une fois le bloc d'instructions à répéter.

```
1  do {bloc d instructions a repeter;  
2  } while (condition de rebouclage) ;
```

# Exemple de boucle "faire tant que"

```
1  #include <stdio.h>
2  int main () {
3      int i = 0 ;
4      do {
5          printf ("iteration %d \n", i) ;
6          i = i + 1 ;
7      } while ( i < 10 ) ;
8      printf ("valeur de i apres la boucle : %d \n", i) ;
9      return 0 ;
10 }
```

# Structure de la boucle "tant que"

- Dans la boucle while, le bloc peut ne jamais être exécuté.
- La condition est vérifiée avant le bloc.

```
1 while (condition de boucle) {  
2     bloc d instructions a repeter;  
3 }
```

# Boucle "tant que" exemple

```
1  #include <stdio.h>
2
3  int main () {
4      int i = 0 ;
5      while ( i < 10) {
6          printf ("iteration %d \n", i) ;
7          i = i + 1 ;
8      }
9      printf ("valeur de i apres la boucle : %d \n", i) ;
10     return 0 ;
11 }
```

Attention : il ne faut pas oublier d'incrémenter *i*.



# L'instruction break

- L'instruction break permet de sortir d'une boucle immédiatement si la condition est réalisée.
- Surtout utilisable avec if et switch.

```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```

```
do {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
} while (testExpression);
```

```
for (init; testExpression; update) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```


# Exemple break

Qu'affiche le programme suivant?


```
1  # include <stdio.h>
2  int main()
3  {
4      int i;
5      double nombre, somme = 0.0;
6      for(i=1; i <= 10; ++i)
7      {
8          printf("Entrer un nombre n%d: ", i);
9          scanf("%lf", &nombre);
10         // si nombre negatif saisi alors fin de la
            boucle
11         if(nombre < 0.0)
12         {
13             break;
14         }
15
16         somme += nombre; // somme = somme + nombre;
17     }
18     printf("Somme = %.2lf", somme);
19     return 0;
```

# L'instruction continue

L'instruction continue permet de sauter l'itération courante de la boucle et continue avec la prochaine itération.



```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```



```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

# Exemple continue

Qu'affiche le programme suivant?

```
1  # include <stdio.h>
2  int main()
3  {
4      int i;
5      double nombre, somme = 0.0;
6      for(i=1; i <= 10; ++i)
7      {
8          printf("Entrer un nombre n%d: ", i);
9          scanf("%lf", &nombre);
10
11         if(nombre < 0.0)
12         {
13             continue;
14         }
15
16         somme += nombre; // somme = somme + nombre;
17     }
18     printf("Somme = %.2lf", somme);
19     return 0;
20 }
```