1 Premier programme

```c
/* Source code (hello.c) */
#include <stdio.h>

int main(){
  printf("Hello World!\n");
  return 0;
}
```

Compilation : dans un *terminal* :

```
clang-14 hello.c -o hello
```

On peut remplacer `clang` par `gcc` et ajouter les options `-Wall` ou `-Werror`.

Exécution : si la compilation réussit (sans erreur), on peut alors exécuter ce binaire, toujours dans un terminal, avec `./hello`.

2 Structure générale d'un programme

(ici, un seul fichier source, les seules inclusions sont des entête de bibliothèque standard)

```c
#include <stdio.h> // en-têtes de bibliothèques (sans ;)
#define TAILLE 100 // constante symbolique

int c;                          // définitions de variables globales (optionnel)

int mafonction(char u){         // définitions de fonctions (optionnel)
  ...
  return -42;
}

int main(){                     // fonction principale du programme (obligatoire)
  mafonction('u');
  ...
  return 0;                                  // ou EXIT_SUCCESS
}
```

3 Types de données

- **Types de base :** `int` (entier signé), `char` (caractère, sur 1 octet), `bool` (avec `stdbool.h`)
- **Type composé :** les tableaux statiques : `int tab[12]` par exemple.

4 Boucles for

Ces boucles sont à utiliser en priorité lorsque l'on connaît à l'avance le nombre d'itérations.

Exemple : somme des éléments de 1 à 12 :

```c
int i;
int s = 0;
for (i=1; i<=12; i++) // attention aux ;
  s = s +i ;
```

S'il y a plusieurs instructions dans le *corps*, alors les accolades sont utiles :

```c
for (int j=12; j>=0; j--) {// boucle décroissante
  // instruction1
  // instruction2
}
```

5 Boucles while

Lorsque la condition est plus complexe, et qu'on ne peut pas facilement borner le nombre d'itérations :

```c
int c = 0;
while(b > 1){
  b = b/2;
  c++;
} // calcule le log_2 de b.
```

La condition du while, qui est une *condition booléenne* (s'évalue à vrai/faux) peut être composée avec non (`!`), ou (`||`), et (`&&`). Attention aux parenthèses.

6 Fonctions

Une fonction sert à créer une portion de code que l'on peut appeler plusieurs fois.

Déclaration et implémentation d'une fonction *nommée* `mafonction`, qui a un unique *paramètre* de type caractère, et qui renvoie ("retourne") un entier (son *type de retour* est `int`) :

```c
int mafonction(char c){
  return -42;
}
```

Une fonction peut ne rien retourner, le type de retour est alors `void`. Une fonction sans paramètre (appelée *procédure*) n'a pas de type entre les parenthèses. Les modifications de paramètres (sauf tableaux) ne sont pas enregistrées à l'extérieur de la fonction.

Appel d'une fonction (à l'intérieur d'une autre fonction, le main, par exemple) :

```c
int resu; // variable du type de retour
resu = mafonction('u'); //le paramètre formel est remplacé par une valeur
```

Alternativement, on peut utiliser le résultat directement :

```c
printf("Le resultat est = %d", mafonction('Z')); //%d car le résultat est un entier (int)!
```

7 Tableaux

Déclaration d'un tableau :

```c
int t[12]; // comme variable locale d'une fonction
```

ou alors, avec une constante symbolique

```c
#define N 100 // au début du programme
...
int tab[N]; // quelque part dans une fonction
```

⚠ Le tableau n'est **pas** automagiquement initialisé. par contre :

```c
int tab[23]={1,12}; // le reste des cases est initialisé à 0.
```

Lecture, écriture

```c
t[2] = ...; // écriture dans la 3ième cellule
... = t[9]; // lecture de la 10ième cellule
```

Passage de paramètre tableau

```c
imprime_tableau(t); // et pas t[N] !
```

⚠ Toute modification du contenu tableau est enregistrée.

Tableaux de caractères Pas de "string" en C, mais des tableaux de `char` avec une sentinelle :

```c
// déclaration et initialisation
char s[21]="hello"; // ajoute `\0` à la fin
```

On peut utiliser les fonctions de la bibliothèque `string`, ou parcourir comme ceci :

```c
while(s[i] != '\0') {...
```

8 Entrées-sorties

Sortie : impression sur le terminal (printf a un nombre de paramètres variable) :

```c
printf("%d", x);// imprime la valeur entière contenue dans x
printf("Ou alors %c, %s\n", caractere, chaine);// autant de % que de variables à imprimer
```

Le premier argument est appelé *chaîne de formattage*.

Entrée : récupération d'une information entrée au clavier :

```c
scanf("%d",&x); // modifie la valeur de x (int) avec la valeur entrée au clavier
scanf("%c",&c);
scanf("%s", s); // une chaîne est un tableau, pas de &
```