Langage C

Premier programme

```
#include <stdio.h>
int main() {
    printf("hello World\n");
    return 0;
}
```

Pour afficher des guillemets dans le texte j'utilise le caractère \

Ex : `printf("hello \"World\"");`

`\n` est un caractère d'échappement qui signifie le retour à la ligne. Il existe d'autres caractères d'échappement.

La fonction fflush(stdout) permet de déclencher l'affichage sur la sortie standard.

Format de la boucle for

```
for(type indice=début; indice=fin; step) {
    instructions à répéter;
}
```

Exemple :

```
for(int i=0; i<5; i++) {
    printf("boucle numéro : \n");
}
```

Les commentaires

```
// Ceci est un commentaire
/* Ceci est un commentaire
sur plusieurs lignes */
```

Affichage avec la fonction printf

```
int nb = 2;

char lettre = 'a';

printf("Le nombre vaut %d et la lettre vaut %c",nb,lettre);
```

Lecture de données en entrée avec la fonction scanf

```
int nb1;

double nb2;

scanf("%d %lf",&nb1, &nb2);
```

Remarque : Pour les chaine de caractères on n'utilise pas le &

Les symboles < et > permettent de rediriger respectivement l'entrée et la sortie standard depuis ou vers un fichier texte.

`mon_programme < mon_fichier` redirige le contenu de mon_fichier vers mon_programme.

Les différents types de variables

Type		*Taille en octet*	*printf*
Entier	`int`		`%d`
Nombre à virgule	`double`		`%lf`
Caractère	`char`		`%c`
Chaine de caractères	`char nom_var[n]`		`%s`
Pointeur	`<type> * nom_point`	Selon type	`%p`
Tableau	`<type> nom_tab[n]`		

Pour affecter un caractère il faut l'encadrer avec des apostrophes ou des guillemets.

Une chaine de caractères peut être déclarée comme un tableau de caractères.

Le dernier caractère d'une chaine de caractères est toujours `\0` donc n est égale à nombre de caractères dans la chaine + 1.

Les bibliothèques limits.h et float.h donnent des indications sur la structure des variables.

L'opérateur sizeof() permet de connaitre la taille d'une variable en mémoire exprimé en octet.

Pour afficher : printf("%zu", sizeof(char))

Remarque sur les formats numériques. Attention aux conversions dans les opérations avec plusieurs formats.

Pour convertir un format dans un autre on utilise :

```
nombre2 = (format) nombre1;
```

Exemple pour convertir une variable int en variable double :

```
int = nb1;

double = nb2;:

nb2 = (double) nb1;
```

Format de l'instruction conditionnelle if

```
if (condition) { // Test la valeur vrais ou faux de condition
      instructionsSiVrais;
} else {
      instructionsSiFaux;
}
```

Remarque : Faux = 0 ou 0.0. Toutes les autres valeurs sont vraies.

Les opérateurs de test sont : `<`, `>`, `<=`, `>=`, `!=`, `==`

Les opérateurs logiques : `&&` (et), `||` (ou), `!` *`var`* (négation de *`var`*)

Le résultat d'un test est toujours un int (0 si faux, 1 si vrais).

Les tableaux

Permet de stocker plusieurs variables de même type.

Exemple pour un tableau de 3 entiers :

```
int mon_tab[3];
```

Exemple pour une matrice 3 lignes 5 colonnes

```
int ma_mat[3][5];
```

L'indexation des cases commence à 0.

Format de la boucle while

```
while(condition) {
```

```
    instruction;
}
```

Si condition est différente de 0 (faux) instruction est exécutée sinon on quitte la boucle.

Les fonctions

Une fonction est un bloc d'instruction qui peut prendre des arguments en entrée et qui retourne une valeur en sortie.

```
type nom_fonction(type param1, type param2, …) {
    instructions;
    return valeur;
}
```

Les fonctions sont typées. Elles sont du type de la valeur qu'elles renvoient. Les fonctions qui ne retournent pas de valeur sont de type void.

La fonction main() est la première exécutée mais est placée en fin de programme.

De façon générale une fonction doit être déclarée avant d'être utilisée.

Pour contourner cette contrainte on peut déclarer des prototypes de fonction en début de programme.

```
type nom_fonction(type1, type2, …);
```

Par défaut les variables qui sont utilisées dans une fonction sont locales à la fonction.

Les pointeurs

Un pointeur est une variable qui référence une zone mémoire.

```
int a = <valeur>;
int * adresseDeA = &a;
```

adresseDeA contient l'adresse mémoire de a.

Les pointeurs sont typés ce qui permet de connaitre la longueur de l'emplacement mémoire occupé par la variable pointée.

On utilise %p pour afficher la valeur d'un pointeur.

```
printf("adresse de a: %p\n", adresseDeA);
```

Le déréférencement est l'opération qui consiste à récupérer la valeur contenue dans la zone mémoire pointée.

```
valeurDeA = * adresseDeA;
```

Remarque : Quand on déclare un tableau le nom du tableau contient l'adresse mémoire de la première case du tableau. C'est pourquoi il n'est pas nécessaire d'utiliser le symbole & pour récupérer la valeur mémoire dans un pointeur ou lorsque l'on fait un scanf.

En fait le nom d'un tableau est un pointeur.

Utilisation des pointeurs dans des fonctions

```
void Mafonction(int * a){
    * a = * a + 3;
}
int main(){
    int a = 5;
    MaFonction(&a);
    return 0;
}
```

On passe l'adresse mémoire (avec &) dans l'appel de la fonction et on utilise le déréférencement dans la fonction.

Tableau de pointeur

```
int tab1[3] = {13, 15, 17};
```

```
int tab2[2] = {23, 27};
```

```
int * pointab[2] = {tab1, tab2};
```

pointab est un tableau de pointeurs. Il existe 3 syntaxes différentes pour accéder aux valeurs des tableaux.

```
* pointab[0] = 13;
```

```
pointab[0][1] = 15;
```

```
* (pointab[1] + 1) = 27;
```

La fonction malloc

La fonction malloc(*taille*) appartient à la bibliothèque stdlib.h. Elle permet d'allouer une zone mémoire en dehors de la pile. Elle est souvent utilisée en combinaison avec la fonction sizeof pour spécifier la taille de la zone mémoire.

Ex : `int * a = malloc(sizeof(int))`.

Pour libérer une zone mémoire on utilise la fonction `free(a)`.