Variables et affichage

		printf	scanf ( )
(unsigned)	char	%c	%c
(const)	int	%d	%d
	long	%ld	%ld
	float	%f	%f
	double	%f	%lf

Opérateurs

+ ; − ; ∗ ; / ; % ; ++ ; −− ; + = ; − = ; ∗ = ; / = ; % =

math.h

- fabs
- ceil
- floor
- pow
- sqrt
- sin, cos, tan
- asin, acos, atan
- exp, log, log10

Aléatoire : srand(time(NULL)) puis rand()%k

Opérateurs : == ; > ; < ; >= ; <= ; != ; && ; || ; !

Valeurs booléennes

- 0 = faux
- 1 ou autre valeur = vrai

Condition

```
01: if (cond)
02: {
03: }
04: else if (cond)
05: {
06: }
07: else
08: {
09: }
```

Switch

```
01: switch (variable)
02: {
03:   case ...:
04:     ...;
05:     break;
06:   case ...:
07:     ...;
08:     break;
09:   default:
10:     ...:
11:     break;
12: }
```

Ternaire

```
01: variable = (condition) ? val_si_true : val_si_false;
```

```
01: while (condition)
02: {
03: }
04:
05: do
06: {
07: }
08: while (condition)
09:
10: for (initialisation ; condition ; incrementation)
11: {
12: }
```

```
01: type/void mafonction(type param, ...)
02: {
03:   ....
04:   return ...;
05: }
```

Prototype Au début du fichier juste après les instructions préprocesseur
Ou dans un fichier header.h

Portée des variables

- Dans une fonction : locale
- Dans une fonction + static : locale mais la valeur est conservé entre chaque appel
- En dehors d'une fonction : globale (tout le programme)
- En dehors d'une fonction + static : tout le fichier

Portée d'une fonction

- En général : globale (tout le programme)
- + static : tout le fichier

Afficher l'adresse d'une variable : printf("%p",&mavariable)

Créer un pointeur (variable contenant l'adresse d'une autre variable)
type *monpointeur = NULL;
monpointeur = &mavariable;

Accéder à une variable via un pointeur
printf("%d",monpointeur) -> adresse de mavariable
printf("%d",*monpointeur) -> valeur de mavariable

Cas d'utilisation
Envoyer un pointeur à une fonction permet de l'autoriser à modifier ponctuellement la valeur d'une variable sans pour autant la rendre globale. On envoie à la fonction l'adresse d'une variable à modifier à cet instant t.

Différences tableau/pointeur : Un tableau est un pointeur constant (on ne peut pas changer l'adresse pointée par un tableau.

Type void* : En général, on donne un type au pointeur pour connaître la "taille" d'une case en mémoire (en fait pointeur = adresse + taille). Avec un type void*, le type n'est pas imposé mais l'indirection est impossible sans typecast.

exemple : memcopy

La pile d'éxécution : Elle contient les variables locales aux fonctions. Les fonctions s'empilent les unes sur les autres avec à la base la fonction main.

Le segment de données : Il contient les variables globales ou déclarées static.

Le tas : Il contient les variables allouées dynamiquement (malloc, calloc, realloc). La mémoire doit aussi être libérée dynamiquement avec free.

Exemple

Nom d'une fonction : Le nom d'une fonction correspond à l'adresse du code source binaire de la fonction + sa signature (type de retour + types des paramètres).

Définir un type pointeur de fonction : typedef TypeRetour (*NomTypePtrFonc) (TypeParam1, TypeParam2 ...)

Exemple :

Exemples :